Geomajas Utility plugins guide

Geomajas Developers and Geosparc

Geomajas Utility plugins guide

by Geomajas Developers and Geosparc

1.15.1

Table of Contents

List of Examples

Chapter 1. Introduction

This plug-in provides extra utility widgets for the Geomajas GWT faces (both SmartGWT and PureGWT). These widgets are not necessarily GIS related. Examples are a Ribbon (office like toolbar), a Wizard, a file upload widget, etc.

In order to know which widgets, are already present within this plug-in, visit the roadmap on the Geomajas issue tracker.

Caution

At this moment, all widgets are based upon the SmartGWT widget library, and are thus best used within the Geomajas SmartGWT face.

Chapter 2. Configuration

1. Dependencies

By default plug-ins are not added to a Geomajas project, so in order to use this plug-in make sure it is included in your project. If you are using Maven, you can add the following dependency to your pom to use the SmartGWT widgets:

Example 2.1. Plug-in dependency

```
<dependency>
    <groupId>org.geomajas.widget</groupId>
    <artifactId>geomajas-widget-utility-gwt</artifactId>
    <version>1.15.1</version>
</dependency>
```

2. GWT Modules

Once you have the plug-in on your classpath, you still need to load the required GWT modules. This plug-in contains GWT widgets based on the SmartGWT library. Therefore, we have tried to follow the SmartGWT way of working, whereby multiple GWT modules have been defined: one for the widgets and several for the supported SmartGWT themes.

By default SmartGWT supports the following themes: "Enterprise", "Enterprise Blue", "Graphite" and "Simplicity". The idea is to have an extension for each of these themes within this plug-in, so that you can apply the same style on these custom widgets as you have for other SmartGWT widgets.

> **Note**
>
> Currently only the SmartGWT "Enterprise" theme has been supported. Others will folow in future versions.

The following is an example that loads the GWT module for the widgets plus the Enterprise theme extension:

Example 2.2. Loading the GWT modules

```
&lt;!-- Plugin module: --&gt;
&lt;inherits name="org.geomajas.widget.utility.gwt.GwtUtilityWidgets" /&gt;

&lt;!-- Plugin Enterprise theme: --&gt;
&lt;inherits name="org.geomajas.widget.utility.theme.enterprise.EnterpriseResou

&lt;!-- Inherited SmartGWT themes: --&gt;
&lt;inherits name="com.smartclient.theme.enterprise.Enterprise"/&gt;
&lt;inherits name="com.smartclient.theme.enterprise.EnterpriseResources"/&gt;
```

3. Widgets

This part will go over all widgets one by one. Some of the widgets support backend configuration which will be discussed in detail below.

3.1. Ribbon

The ribbon is a office like broad toolbar with clear text and icons on the main buttons. It has support for not just buttons, but any other widget as well. The main idea for the ribbon is that it should easily

work together with a MapWidget, by providing a way of using navigation controllers etc. In order to accomplish this goal, it is possible to configure the ribbon within the backend Spring configuration, using the tools that are already available for the normal Toolbar widget (from the SmartGWT face).

This part will focus on how to configure a Ribbon widget from the backend Spring configuration. Visit the "Howto" chapter in order to find out how to create custom ribbon columns or how to define custom ribbon styles (theming). Let us start with an example Ribbon configuration for a simple Ribbon (without tabs):

Example 2.3. Example Ribbon configuration

```
<!-- application configuration -->
<bean name="app" class="org.geomajas.configuration.client.ClientApplicationInfo
   <property name="maps">
      <list>
         <ref bean="mainMap" />
         <ref bean="overviewMap" />
      </list>
   </property>
   <property name="widgetInfo">
      <map>
         <entry key="ribbon-bar-1">
            <ref bean="ribbon-bar-1" />
         </entry>
      </map>
   </property>
</bean>

<!-- ribbon configuration -->
<bean name="ribbon-bar-1" class="org.geomajas.widget.utility.configuration.Ribb
   <property name="groups">
      <list>
         <bean class="org.geomajas.widget.utility.configuration.RibbonGroupInfo
            <property name="title" value="Navigation" />
            <property name="columns">
               <list>
                  <bean class="org.geomajas.widget.utility.configuration.Ribbon
                     <property name="type" value="ToolbarActionButton" />
                     <property name="tools">
                        <list>
                           <ref bean="ZoomIn" />
                        </list>
                     </property>
                  </bean>
                  <bean class="org.geomajas.widget.utility.configuration.Ribbon
                     <property name="type" value="ToolbarActionButton" />
                     <property name="tools">
                        <list>
                           <ref bean="ZoomOut" />
                        </list>
                     </property>
                  </bean>
                  <bean class="org.geomajas.widget.utility.configuration.Ribbon
                     <property name="type" value="ToolbarActionList" />
                     <property name="tools">
                        <list>
                           <ref bean="ZoomPrevious" />
                           <ref bean="ZoomNext" />
```

```
                        </list>
                     </property>
                  </bean>
               </list>
            </property>
         </bean>
      </list>
   </property>
</bean>
```

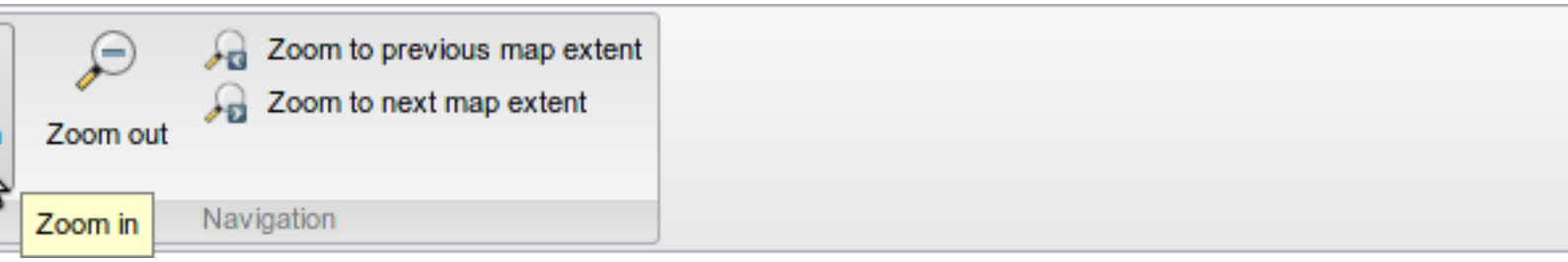
The example above defines a single Ribbon bar with a single group of columns. The three columns within the group are the most basic elements within a Ribbon. Such columns can be grouped together within a RibbonGroup under a single title. In the case above, the group's title is "Navigation". Since the columns within this group contain zooming actions (ZoomIn, ZoomOut, ZoomPrevious, ZoomNext) the title "Navigation" is quite appropriate. As for the names of the tools - those are the same as the tools used for the normal Toolbar widget from the SmartGWT face.

The zooming actions used within the columns are supported through 2 types of Ribbon columns: a single big button (type = "ActionButton") and a vertical list of action buttons (type = "ActionList"). By default these are the only types known to the Ribbon widget, but custom types can be defined as well. Visit the "Howto" chapter to see how to define such custom types.

Note also that the Ribbon above has been defined with a name "ribbon-bar-1" (top line). This name identifies the Ribbon configuration so that the client can request it during initialization. In order to create a Ribbon widget using the above configuration, use the following Java code on the client:

```
RibbonBarLayout ribbon = new RibbonBarLayout(mapWidget, "app", "ribbon-bar-1");
```

The Ribbon bar widget with the above configuration will look like this:



Ok, so that was a simple Ribbon widget with a single bar. Real world applications may often contain a great deal of actions within a menu, and so for complex applications a tabbed version of the Ribbon might be in order. The configuration for a tabbed Ribbon, is as follows:

Example 2.4. Example Ribbon configuration

```
<bean name="tabbed-ribbon-1" class="org.geomajas.widget.utility.configuration.R
   <property name="tabs">
      <list>
         <bean class="org.geomajas.widget.utility.configuration.RibbonBarInfo">
            <property name="title" value="TAB Nr 1" />
            ....
         </bean>
         <bean class="org.geomajas.widget.utility.configuration.RibbonBarInfo">
            <property name="title" value="TAB Nr 2" />
            ....
         </bean>
         <bean class="org.geomajas.widget.utility.configuration.RibbonBarInfo">
            <property name="title" value="TAB Nr 3" />
            ....
         </bean>
```

```
      </list>
   </property>
</bean>
```

As you can see, a tabbed Ribbon consists of a list of Ribbon bars. Note though that this time, the Ribbon bars each have a title. This is the label that will be used on the actual tabs.

Again, the top level has a name "tabbed-ribbon-1" so that the client widgets can use this configuration part. The following Java code will create a tabbed Ribbon using the above configuration:

```
RibbonTabLayout ribbon = new RibbonTabLayout(mapWidget, "app", "tabbed-ribbon-1
```

Note that a different type of widget is used (RibbonTabLayout instead of RibbonBarLayout)!

Chapter 3. How-to

1. Configuring a custom RibbonColumn widget

The basic element within a Ribbon, is the RibbonColumn interface. All buttons and other widgets within the Ribbon groups must implement this interface. By default only 2 types have been implemented: a single big button (type = "ActionButton") and a vertical list of action buttons (type = "ActionList").

This section describes how to expand this list with your own custom types. In order to accomplish this, the following steps need to be taken:

1. Create a new RibbonColumn implementation. This class will be the widget we want to be used within the Ribbon.
2. Register this custom RibbonColumn implementation, using a specific key.
3. Use this key in the backend Spring configuration.

> **Note**
>
> Note that this section assumes that you want your custom type to be used through configuration. If you assemble your Ribbon without configuration (but by using Java code only), then there is no need to register your type.

In the sections below, we will create a custom RibbonColumn that displays the lon-lat position of the mouse pointer on the map.

1.1. Create a custom RibbonColumn implementation

In this example we will implement a custom widget that displays lon-lat coordinates of the mouse pointer on the map. The code could look like this:

Example 3.1. Custom RibbonColumn implementation

```
public class MyCustomRibbonColumn extends VLayout implements RibbonColumn {

 private Label xLabel;
 private Label yLabel;
 private Listener listener;
 private MapWidget mapWidget;
 private String xText = "X";
 private String yText = "Y";

 public MyCustomRibbonColumn(MapWidget mapWidget) {
  super(8);
  this.mapWidget = mapWidget;

  xLabel = new Label(xText + ":");
  xLabel.setSize("80px", "16px");
  addMember(xLabel);

  yLabel = new Label(yText + ":");
  yLabel.setSize("80px", "16px");
  addMember(yLabel);
```

```
 listener = new MyMapListener();
 mapWidget.addListener(listener);
}

public Widget asWidget() {
 return this;
}

public void setShowTitles(boolean showTitles) {
}

public boolean isShowTitles() {
 return false;
}

public void setTitleAlignment(TitleAlignment titleAlignment) {
}

public TitleAlignment getTitleAlignment() {
 return TitleAlignment.BOTTOM;
}

public void setButtonBaseStyle(String buttonBaseStyle) {
}

/**
 * Can accept "X and "Y" text values to be printed out.
 */
public void configure(String key, String value) {
 if ("x".equalsIgnoreCase(key)) {
  xText = value;
 } else if ("y".equalsIgnoreCase(key)) {
  yText = value;
 }
}

// ------------------------------------------------------------------------
// SmartGWT methods overrides:
// ------------------------------------------------------------------------

@Override
protected void onDestroy() {
 mapWidget.removeListener(listener);
 super.onDestroy();
}

// ------------------------------------------------------------------------
// Private classes:
// ------------------------------------------------------------------------

/**
 * Private map listener that gets the world position of the mouse pointer and
 *
 * @author Pieter De Graef
 */
private class MyMapListener implements Listener {

 public void onMouseDown(ListenerEvent event) {
```

```
  }

  public void onMouseUp(ListenerEvent event) {
  }

  public void onMouseMove(ListenerEvent event) {
   Coordinate worldPosition = event.getWorldPosition();
   double x = ((double) Math.round(worldPosition.getX() * 1000)) / 1000;
   double y = ((double) Math.round(worldPosition.getY() * 1000)) / 1000;
   xLabel.setContents(xText + ": " + x);
   yLabel.setContents(yText + ": " + y);
  }

  public void onMouseOut(ListenerEvent event) {
  }

  public void onMouseOver(ListenerEvent event) {
  }

  public void onMouseWheel(ListenerEvent event) {
  }
 }
}
```

This piece of code implements the RibbonColumn interface, and extends the SmartGWT VLayout class. It is therefore a widget that can be used as part of a Ribbon. During construction, it registers a custom Listener (MyMapListener) on the map, which extracts the mouse pointer location and updates the label text to display that position.

The most important method within this class is the "configure" method. This method determines which parameters this particular RibbonColumn supports. In this case, a "X" and "Y" parameter can be set to be displayed on the labels. Later in the backend Spring configuration, we will use the "X" and "Y" parameters to configure this widget.

1.2. Register your custom RibbonColumn

In order to be able to use your custom RibbonColumn implementation in the backend Spring configuration, you have to register the class with a unique key. This key can then be used in the configuration to create instances of of your custom type. The following piece of Java code, registers our custom widget on the GWT client:

Example 3.2. Registering your custom class

```
RibbonColumnRegistry.put("MyCustomColumn", new RibbonColumnCreator() {

 public RibbonColumn create(List<ClientToolInfo> tools, MapWidget mapWidget) {
  return new MyCustomRibbonColumn(mapWidget);
 }
});
```

The above piece of code actually registers a new RibbonColumnCreator in the RibbonColumnRegistry. This creator will create new instances of the MyCustomRibbonColumn class when required. Note that we don't use the tools here - so no use configuring them for this type.

What's important here is the key that is used. In this case the key "MyCustomColumn" is used. This same key can now be used in the backend Spring configuration.

1.3. Using the custom type in the configuration

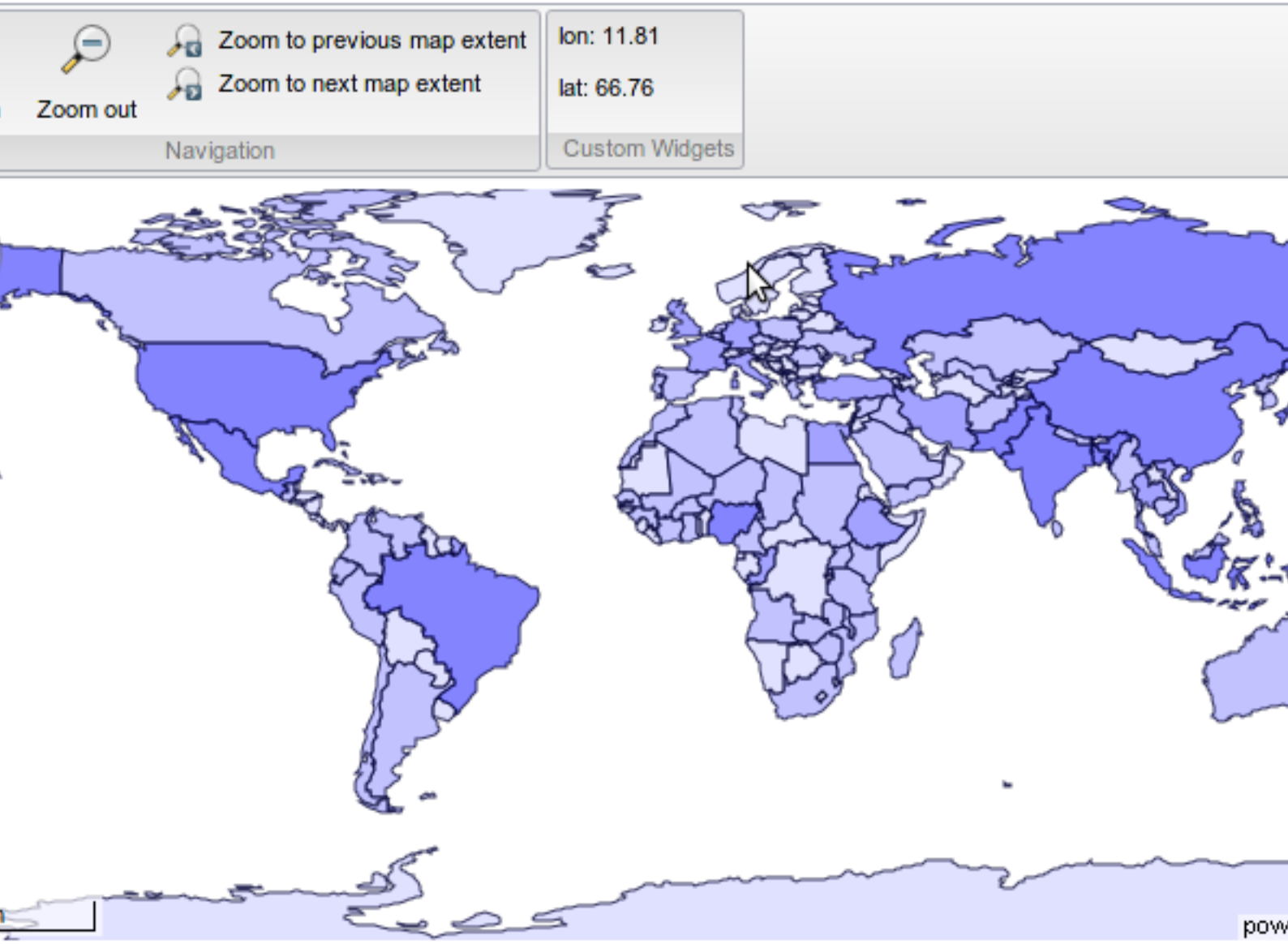
Now that we have a custom RibbonColumn implementation and it has been registered with the key "MyCustomColumn", it is time to actually use it in the backend Spring configuration. An example configuration for the Ribbon column could look like this:

Example 3.3. Custom type in configuration

```
...
<bean class="org.geomajas.widget.utility.configuration.RibbonColumnInfo">
 <property name="type" value="MyCustomColumn" />
 <property name="tools">
  <list>
  </list>
 </property>
 <property name="parameters">
  <list>
   <bean class="org.geomajas.configuration.Parameter">
    <property name="name" value="x" />
    <property name="value" value="lon" />
   </bean>
   <bean class="org.geomajas.configuration.Parameter">
    <property name="name" value="y" />
    <property name="value" value="lat" />
   </bean>
  </list>
 </property>
</bean>
...
```

The above configuration sample shows how to use the custom type "MyCustomColumn" within the XML configuration. Note that the list of tools is empty, as our registered RibbonColumnCreator does not use them anyway. Instead, we now see a list of parameters. Remember that when we defined our custom RibbonColumn implementation, there was support for 2 configuration parameters in the "configure" method, namely "X" and "Y". Now in our configuration, we can use those 2 parameters to attach a value to them ("lon" and "lat").

The custom widget used those 2 parameters to be displayed on the labels. See below for a screenshot of our custom type:



2. Defining a custom Ribbon style

In the configuration chapter it was mentioned that this plug-in defines extensions for the default SmartGWT themes specifically tailored for the widgets it provides. This means that new CSS classes have been defined for the widgets. This section will show how to override the default styles or how to create a new custom style for the Ribbon widget.

2.1. Overriding the default theme

By default, the themes used are the following:

- *ribbon*: The outer Ribbon bar. Typically defines border and background for the entire Ribbon bar.
- *ribbonGroup*: Style for a Ribbon group. Typically defines border and background colors for each group.
- *ribbonGroupBody*: Style for the upper part of a Ribbon group (where the buttons are). Typically defines padding, spacing, ...
- *ribbonGroupTitle*: Style for the group titles. Typically defines the type of font used for the titles.

- *ribbonButton*: Base style for the buttons within the Ribbon bar. Since the buttons follow the SmartGWT way of working, there are extra styles for hovering, selection, disabled.... (ribbonButton, ribbonButtonOver, ribbonButtonFocused, ribbonButtonFocusedOver, ribbonButtonDown, ribbonButtonFocusedDown, ribbonButtonSelected, ribbonButtonSelectedFocused, ribbonButtonSelectedDown, ribbonButtonSelectedFocusedDown, ribbonButtonSelectedOver, ribbonButtonSelectedFocusedOver, ribbonButtonDisabled, ribbonButtonSelectedDisabled)

By overriding these CSS classes can you override the default style for the ribbon.

2.2. Creating a custom style

Alternatively you can create an extra ribbon style to be used in conjunction with the default style. In this case none of the default styles will be overridden, and for each Ribbon widget that you use, you can choose which style to apply. All you have to do is come up with a base name for the new ribbon style, and replace the "ribbon" part in all CSS classes with your new name.

For example, lets create a Ribbon style with the name "myRibbon". The CSS classes within the Ribbon would then be: myRibbon, myRibbonGroup, myRibbonGroupBody, myRibbonGroupTitle, myRibbonButton.

In the client Java code, you then have to apply this new style on the Ribbon as follows:

Example 3.4. Applying a custom style

```
RibbonBarLayout ribbon = new RibbonBarLayout(mapWidget, "ribbon-bar-1");
ribbon.setStyleName("myRibbon");
```

Below is a CSS extract that defines the "myRibbon" style for the Ribbon widget:

Example 3.5. Defining the myRibbon style

```
.myRibbon {
 padding: 2px;
    background: #E0E0F8;
    background-image: -moz-linear-gradient(top, #D8D8F0, #E8E8FF);
    background-image: -webkit-gradient(linear, left top, left bottom, from(#D8D8
    border: 1px solid #A7ABD4;
}

.myRibbonGroup {
 border-right: 1px solid #A7ABB4;
 height: 100%;
}

.myRibbonGroupBody {
 padding: 3px 5px;
}

.myRibbonGroupTitle {
 font-family: Arial,Verdana,sans-serif;
 font-size: 11px;
 font-weight: bold;
 color: #555599;
 padding: 1px 5px;
 white-space: nowrap;
}

.myRibbonButton,
.myRibbonButtonOver,
.myRibbonButtonFocused,
```

```
.myRibbonButtonFocusedOver,
.myRibbonButtonDown,
.myRibbonButtonFocusedDown,
.myRibbonButtonSelected,
.myRibbonButtonSelectedFocused,
.myRibbonButtonSelectedDown,
.myRibbonButtonSelectedFocusedDown,
.myRibbonButtonSelectedOver,
.myRibbonButtonSelectedFocusedOver,
.myRibbonButtonDisabled,
.myRibbonButtonSelectedDisabled {
 padding: 2px 1px;
 font-family: Arial,Verdana,Bitstream Vera Sans,sans-serif;
 font-size: 11px;
 color: #000000;
    -moz-border-radius: 3px;
    -webkit-border-radius: 3px;
    border: 1px solid transparent;
}

.myRibbonButtonDisabled,
.myRibbonButtonSelectedDisabled {
 font-weight: normal;
 color: #AAAAAA;
}

.myRibbonButtonFocused,
.myRibbonButtonFocusedOver,
.myRibbonButtonSelectedFocused,
.myRibbonButtonSelectedFocusedOver,
.myRibbonButtonSelectedFocusedDown {
 border-left: 1px solid #B2D3FC;
 border-right: 1px solid #AACBF6;
 border-top: 1px solid #B2D3FC;
 border-bottom: 1px solid #AACBF6;
}

.myRibbonButtonOver,
.myRibbonButtonFocusedOver,
.myRibbonButtonSelectedFocused,
.myRibbonButtonSelectedFocusedOver,
.myRibbonButtonSelectedOver {
 border-left: 1px solid #999999;
 border-right: 1px solid #666666;
 border-top: 1px solid #999999;
 border-bottom: 1px solid #666666;
}

.myRibbonButtonDown,
.myRibbonButtonFocusedDown,
.myRibbonButtonSelectedDown,
.myRibbonButtonSelectedFocusedDown {
    background: #FFDDBE;
 border-left: 1px solid #b8cfef;
 border-right: 1px solid #b8cfef;
 border-top: 1px solid #87B6EC;
 border-bottom: 1px solid #75AEEC;
}
```

```
.myRibbonButtonSelected,
.myRibbonButtonSelectedFocused,
.myRibbonButtonSelectedFocusedOver,
.myRibbonButtonSelectedDisabled,
.myRibbonButtonSelectedOver {
    background: #FFD7B2;
}
```