Customizing the CSegmentButton Control

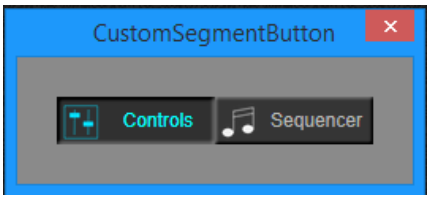
Will Pirkle

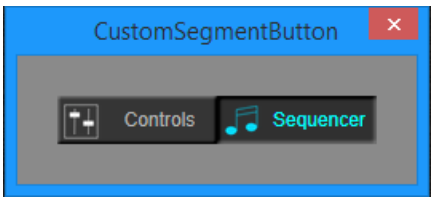
VSTGUI4.3 added a sorely needed control called *CSegmentButton*, making it very easy to create radio button controls and vertical or horizontal switch controls without needing any complicated graphics files. However, the *CSegmentButton* is actually highly customizable and it is easy to add graphics for the buttons and icons as well as customize the text font and color. This is a great use of the Custom View paradigm used in RackAFX. To understand this document, you should be familiar with the Advanced API GUI tutorials, at least through Module 4. For this project, I will create the same tab-controller *CSegmentButton* used in my MultiFX project which is used in all of the recent v6.8 videos.

On this GUI there is a *CSegmentButton* with two segments that switches the two views via a *UIViewSwitchContainer*. In the GUI Designer, the control looks like this (lower right corner):

However, on the final GUI, it looks like this (much nicer):

In the accompanying project, I will only be demonstrating the customized *CSegmentButton*, which looks like this:





You can see the customization:

- button graphics with on/off states
- cyan text color for highlighted (enabled or pressed) button and light grey color for disabled or un-pressed button
- icons with both normal (grey) and highlighted (enabled) colors

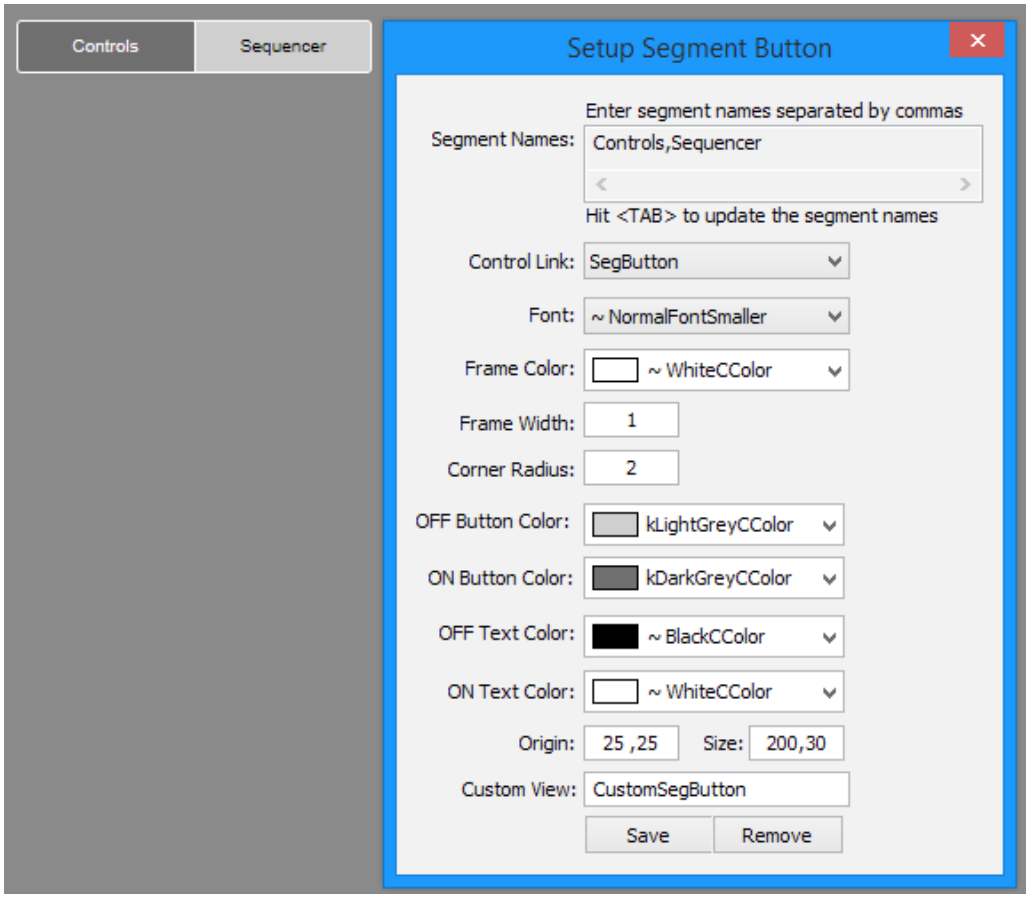
There are six graphics files needed for this setup:

1. button off state
2. button on state
3. controls icon disabled state
4. controls icon enabled state
5. sequencer icon disabled state
6. sequencer icon enabled state

I generated the button graphics with the KnobMan software and the icons using Omnigraffle for MacOS. once you have your graphics ready, follow these steps to perform the customization.

Create the Custom View

In the GUI Designer, drag a horizontal *CSegmentButton* control into the GUI. Give it a Custom View Name (CustomSegButton is what I used). It is linked to the trivial radio button control on the main UI, but it dos not need to be in order to work as a Tabbed View Switcher (see the video and documentation for Tutorial Module 1). You need to take care of the size of the control here as you will need to match your GUI graphics sizes. Since the button graphics are 100 x 30 each and will be side-by-side, the control needs to be 200 x 30 in size (200,30):



Prepare your project for Cusom Views

Follow the steps in the Tutorial Module 2 to enable your project for custom views; this includes:

1. installing the VSTGUI4 library
2. uncommenting the #include lines from your .h file

```
#include "GUIViewAttributes.h"
#include "../vstgui4/vstgui/vstgui.h"
```

3. adding the GUIViewAttributes.h and GUIViewAttributes.cpp files to your Visual Studio project

Add your graphics files to Visual Studio

Follow the steps in the video tutorial here:

http://www.willpirkle.com/support/video-tutorials/#PIGE

to add the graphics files to your Visual Studio project. After editing the .rc file you should have:

```
SEGBUTTON.PNG               PNG             "resources\\segbutton.png"
SEGBUTTONOFF.PNG            PNG             "resources\\segbuttonoff.png"
DISABLEDCONTROLS.PNG        PNG             "resources\\disabledcontrols.png"
DISABLEDSEQ.PNG             PNG             "resources\\disabledseq.png"
ENABLEDCONTROLS.PNG         PNG             "resources\\enabledcontrols.png"
ENABLEDSEQ.PNG              PNG             "resources\\enabledseq.png"
```

Note that you do NOT need to modify anything in the GUI Designer; these graphics won't be needed for the GUI Designer since we are using them only in our custom view.

Modify your Project for Custom Views

Declare the variables for your custom view; a *CSegmentButton** and a *CVSTGUIHelper* object:

```
// Add your code here: ----------------------------------------------------------- //
CSegmentButton* m_pTabCtrlSegButton = nullptr;
CVSTGUIHelper m_GUIHelper;

// END OF USER CODE -------------------------------------------------------------- //
```

Notice I'm using the C++11 capability of initializing the pointer right in the .h file.

Next, modify the *showGUI( )* function as detailed in the Advanced GUI API tutorials. First, we need to decode the custom view string and create the *CSegmentButton* control by replying to the GUI_CUSTOMVIEW message in *showGUI( )*:

```
case GUI_CUSTOMVIEW:
{
	// --- create custom view
	if (strcmp(info->customViewName, "CustomSegButton") == 0)
	{
		// --- get the needed attributes with the helper
		const CRect rect = m_GUIHelper.getRectWithVSTGUIRECT(
                                                        info->customViewRect);

		// --- create the control
		m_pTabCtrlSegButton = new CSegmentButton(rect,
                                        (IControlListener*)info->listener,
                                        info->customViewTag);

		// --- return control cloaked as a void*
		return (void*)m_pTabCtrlSegButton;
	}

	return NULL;
}
```

Apply the customization for the reply to the GUI_DID_OPEN message in *showGUI( )* which involves the following steps:

1. create the graphics
2. set the font, text color and highlighted text color for the *CSegmentButton* control (notice the use of setting the *CColor* on-the-fly as well as using a built-in color for cyan)
3. remove all the existing segments

This first chunk of code looks like this:

```
case GUI_DID_OPEN:
{
	if (m_pTabCtrlSegButton)
	{
	   // --- graphics
	   CBitmap* pSegmentOFF = m_GUIHelper.loadBitmap("segbuttonoff.png");
	   CBitmap* pSegmentON = m_GUIHelper.loadBitmap("segbutton.png");
	   CBitmap* pDisabledControls = m_GUIHelper.loadBitmap("disabledcontrols.png");
	   CBitmap* pEnabledControls = m_GUIHelper.loadBitmap("enabledcontrols.png");
	   CBitmap* pDisabledSeq = m_GUIHelper.loadBitmap("disabledseq.png");
	   CBitmap* pEnabledSeq = m_GUIHelper.loadBitmap("enabledseq.png");

	   // --- styles
	   m_pTabCtrlSegButton->setFont(kNormalFont);

	   // --- create a light grey color, stock grey is too dark
	   m_pTabCtrlSegButton->setTextColor(CColor(175, 175, 175, 255));

	   // --- light blue text-highlight
	   m_pTabCtrlSegButton->setTextColorHighlighted(kCyanCColor);

	   // --- begin customization
	   m_pTabCtrlSegButton->removeAllSegments();
```

Next you build each segment, one at a time and provide the following:

1. text string (name)
2. button graphic for un-pressed state (background)
3. button graphic for pressed state (backgroundHighlighted)
4. icon graphic for un-pressed state (icon)
5. icon graphic for pressed state (iconHighlighted)
6. location of icon relative to the text (*CDrawMethods::kIconLeft*)
7. finally, add the button to the *CSegmentButton*

You can choose the following icon locations (from *cdrawmethods.h* in VSTGUI4):

```
enum IconPosition {
	kIconLeft,		///< icon left, text centered in the area next to the icon
	kIconCenterAbove,	///< icon centered above the text, text centered
	kIconCenterBelow,	///< icon centered below the text, text centered
	kIconRight		///< icon right, text centered in the area next to the icon
};
```

Do this procedure for each button and remember to use *forget( )* on the graphics to prevent memory leaks. Notice that because of this we need to reload the button graphics as they are shared and the same for all buttons. That code looks like this:

```
// --- Controls segment button
CSegmentButton::Segment seg1;
seg1.name = "Controls";
seg1.background = pSegmentOFF;
seg1.backgroundHighlighted = pSegmentON;
seg1.icon = pDisabledControls;
seg1.iconHighlighted = pEnabledControls;
seg1.iconPosition = CDrawMethods::kIconLeft;

// --- add the segment
m_pTabCtrlSegButton->addSegment(seg1);

// --- forget graphics
pSegmentOFF->forget();
pSegmentON->forget();
pDisabledControls->forget();
pEnabledControls->forget();

// --- reload graphics
pSegmentOFF = m_GUIHelper.loadBitmap("segbuttonoff.png");
pSegmentON = m_GUIHelper.loadBitmap("segbutton.png");

// --- Sequencer segment button
CSegmentButton::Segment seg2;
seg2.name = "Sequencer";
seg2.background = pSegmentOFF;
seg2.backgroundHighlighted = pSegmentON;
seg2.icon = pDisabledSeq;
seg2.iconHighlighted = pEnabledSeq;
seg2.iconPosition = CDrawMethods::kIconLeft;

// --- add the segment
m_pTabCtrlSegButton->addSegment(seg2);
```

```
            // --- forget graphics
            pSegmentOFF->forget();
            pSegmentON->forget();
            pDisabledSeq->forget();
            pEnabledSeq->forget();
      }

      return NULL;
}
```

Finally, remember to null out the segment button pointer during the reply to the GUI_WILL_CLOSE message in *showGUI( ):*

```
case GUI_WILL_CLOSE:
{
      if (m_pTabCtrlSegButton) m_pTabCtrlSegButton = nullptr;

      return NULL;
}
```

Build and Test

Thats it! Build and test the plugin - experiment with your own graphics, fonts and colors.