Using XY Pads and VU Meter Objects in RackAFX
Will Pirkle

This brief documents the sample project ***XYPadsMeters.*** This simple plugin shows how to use the XY pad and the solid/segmented or analog VU Meters. The final GUI is shown here:

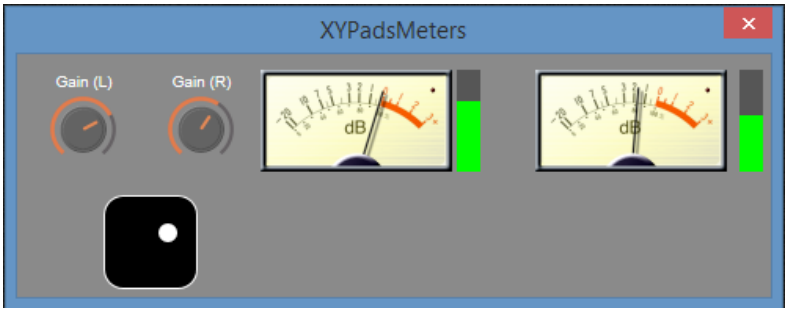

Figure 1: the plugin GUI controls and meters

The XY Pad has the Left Gain knob mapped to the X-dimension and the Right Gain knob mapped to the Y-dimension. This document explains how to use each of these controls in the RackAFX GUI Designer. You should already know how to create the plugin and implement the left and right volume (gain) controls, and how to add and connect VU meters to your RackAFX plugin core. You should also know the basics of using the GUI Designer (see tutorial videos at https://www.willpirkle.com/support/video-tutorials/). This document is only concerned with the GUI controls and explains how you set them up as they are customized versions of VSTGUI4 controls.

Custom Graphics
The two knob controls use the maschine.png file that accompanies this project, however it is also available for you as an added bonus control in your RackAFX or ASPiK project in the ***/Resources/knobs*** folder.

CAnimKnobs
There are two *CAnimKnobs* in the GUI that you need to create and connect. These both use the **maschine.png** graphics file.

Importing Graphics
Once you have two knob and two meter controls defined in your plugin project, build your plugin and load it, and then go to the GUI Designer. On the left, select the PNG tab, click the New button, and browse to the */Resources/knobs* folder and select the file (experiment with the others too), as shown in Figure 2. This graphic file will now be available in all graphics drop-lists in the GUI designer.

Place the Controls & Set Attributes
Using either the Templates or Objects tabs, drag and drop the knob controls into the GUI designer. Here I am using the Knob Group B template. I first set the label text, then work on the knob, changing the graphic and selecting the control ID from the list as the connector as shown in Figure 3. Make sure you set the ***Image Ht*** and ***Images*** attributes correctly. All of the pre-supplied knobs have the same dimensions (42,42), Image Height (42 pixels) and number of images (80).

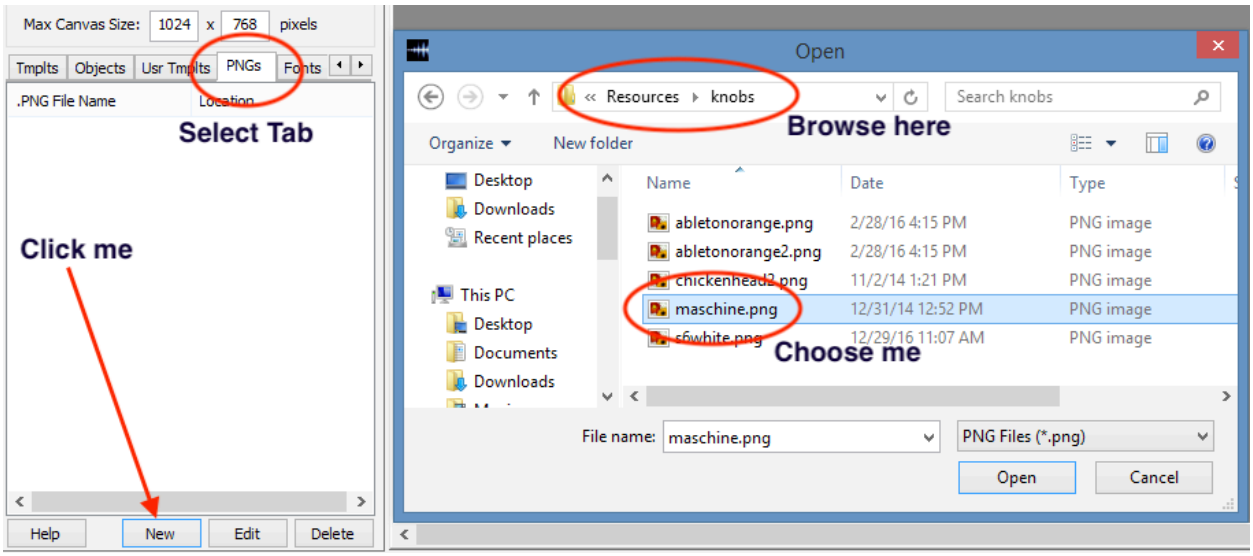

Figure 2: Importing graphics files into your RackAFX plugin project

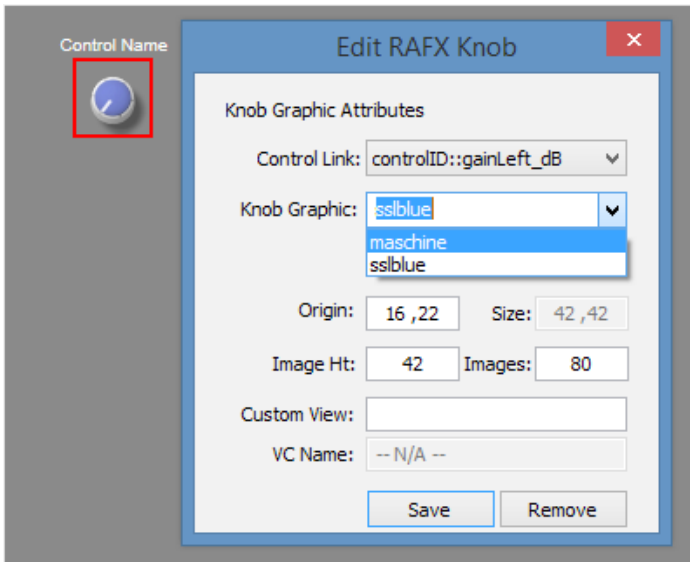

Figure 3: right-clicking the knob and selecting the graphic in RackAFX

XY-Pad

The CXYPad object implements the stock VSTGUI4 control. I have subclassed this control as CXYPadEx that allows for easier customization and operation **without a VSTGUI4 sub-controller**. While the sub-controller is a powerful paradigm in VSTGUI4, it adds some complexity to the normal XY Pad operation and my control gets around that using a simple encoding scheme in the VSTGUI4 custom view name string.

GUI Designer:

In the GUI Designer, choose the Objects tab and drag a XY-Pad control onto the Designer pane. Right click on the control to set it up as shown in Figure 4. Play with the colors and parameter variations. Rounded corners and Show Frame are interesting. Choose the X and Y axis control ID values (tags) from

the drop-lists. The custom view name will be altered for you to encode the X and Y control ID values, as show on the right of Figure 4. See the ASPiK Documentation for the encoding method

http://aspikplugins.com/sdkdocs/html/gui_designer11_d.html

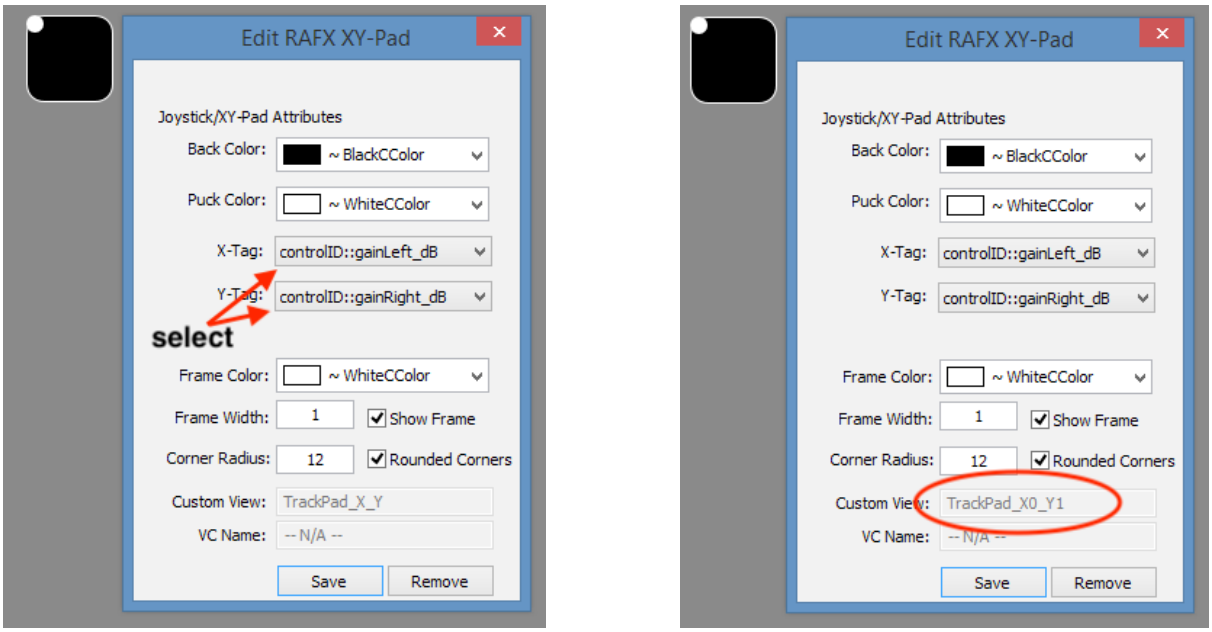

Figure 4: XY Pad setup; choose the X and Y axis tags from the drop-lists; after closing and opening the XY Pad setup dialog, you will see the custom view name has been encoded for you

The custom view name will be decoded and the custom object will be created in the *PluginGUI::createView*( ) function; you can look at that code to see how the custom view string is parsed and the X/Y control IDs are extracted, as well as how the object is instantiated.

Solid/Segmented VU Meter

The VSTGUI4 *CVuMeter* object uses two PNG files: one for the meter in a fully OFF state and the other for the meter in the fully ON state. The RackAFX meter graphics are designed to work with a built-in extended object called *CVuMeterEx* that is available in all RackAFX and ASPiK projects in the files customviews.h and customviews.cpp. This object is subclassed from *CVuMeter* with one big difference: my version includes the ability to use PNG strip-animation files just like the knobs use IN ADDITION to the normal on/off pair of graphic files that allows you to implement an analog view meter with needle and background. You can also use any other strip animation from KnobMan as a VU meter. You supply two graphics that you set in the GUI Designer (RackAFX or VSTGUI4). You also set the graphics size as X,Y in pixels, and the "number of LEDs." For solid meters, the number of LEDs is the number of Y-pixels for vertical, and X-pixels for horizontal meters respectively. For segmented meters, this refers to the number of discrete LEDs that light as the meter moves.

Importing Graphics & Adding the Meters

Use the same mechanism to import the two included solid VU meter files called ***vuon.png*** and ***vuoff.png*** that indicate a vertical meter (tall and narrow). In the GUI designer, drag a **LED Meter** object from the Objects tab into the designer pane. Right click to set it up. Choose your ON and OFF meter graphics from the drop-lists. Be careful to set the ON graphic to the ON file and vice versa for OFF. Select the control ID of the meter to connect it. Figure 5 shows how the VU meter setup panel looks before and after you

select the graphics. The RackAFX GUI Designer will automatically calculate the dimensions of your file; make sure that the Horizontal box is NOT checked. After you select the graphics files, the graphic display will change as show on the right side of Figure 5. The OFF graphic will always be shown in the GUI designer. If you see the ON graphic instead, it means you have chosen the wrong ON/OFF pair. Note that this Size control is greyed out as it will be set automatically for you (both files must be identical in size).

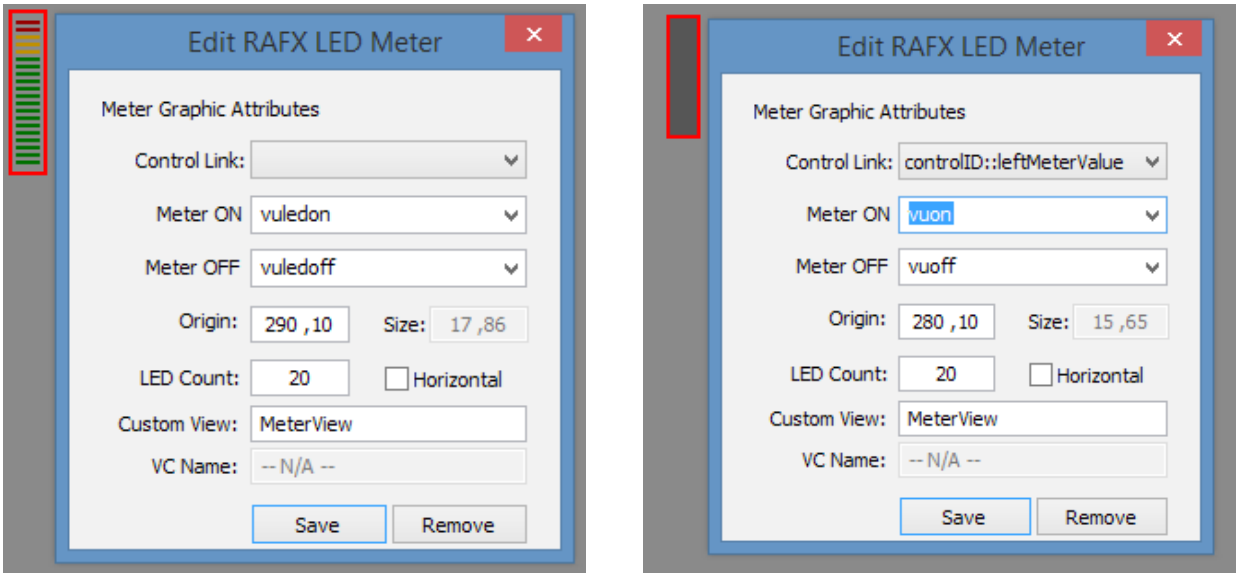

Figure 5: the VU Meter setup panel in RackAFX makes it easy to connect the control and set the on/off graphics files

Notice that the Custom View name is set to *MeterView* which is the specific name for the normal LED meter. If you setup your meter in the RackAFX prototyping panel as inverted, then the custom view will change to *InvertedMeterView*. This will happen automatically for you when you select a new control ID value.

Analog VU Meter

Analog meters use the strip animation just like the *CAnimKnob* control for the graphics. However, there are several more parameters that must be set for this control to work properly.

Importing Graphics & Adding the Meters

Use the same mechanism to import the four analog VU meter files included in this sample pack and code. The *readme.txt* file in the accompanying folder contains some of the text from this document so it is available there as well.

In the RackAFX GUI Designer, drag an **Analog VU Meter** from the Objects tab on the left into the Designer pane. It will use a built-in graphic that is the same as the *medanalogvumeter.png* file that accompanies this document. Right clicking the control shows the additional parameters to set as shown in Figure 6.

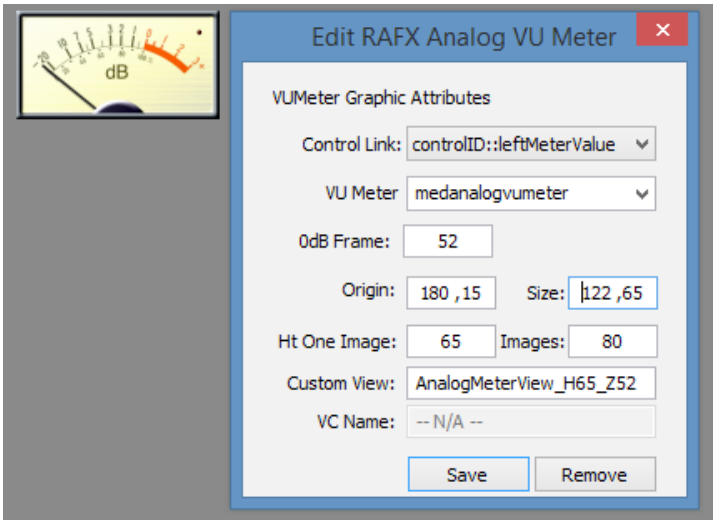

Figure 6: the analog VU meter setup panel is not the same as the LED meter panel and includes extra parameters for the strip animation; note the Custom View name will be set for you based on you attribute settings

Images: the number of frames that you set in KnobMan, or the number of individual images that makeup the strip animation.

Ht One Image: as with the *CAnimKnob* control, this sets the height of one frame in the strip animation.

Size: as with the normal VU meters, or the CAnimKnob this parameter sets the dimensions of a single frame or sub-image. For the meters in this pack the values are:

biganalogvumeter.png	177 x 94
analogvumeter.png	147 x 78
medanalogvumeter.png	122 x 65
smallanalogvumeter.png	60 x 32

Enter the size as X,Y or 177,94 for the biganalogvumeter file.

Notice that the second parameter (94 in this case) is the height of just one image in the strip.

0dB Frame: this is specific to the custom control. The analog VU meter images show the VU needle in various locations from minimum to maximum values. The PNG files included here have a 0dB marking, and then a section of RED that is above it. This is a sort of calibrated meter that shows the *nominal* level as 0dBVU, and values above that as +dBVU in the RED zone on the meter background. With these meters, the nominal value is around -9dBFS.

For all of the analog meters in this pack, the 0dB frame is number 52 (the top image is number 0, the next image down is number 1, etc...).

The 0dB frame is only important for inverted analog meters, as you might find on an analog compressor/limiter -- when the needle is at 0dBVU, there is no gain reduction, and as the device compresses, the needle moves backwards from the 0dBVU point. This value sets that point.

In addition, the strip animation includes a small LED in the upper right corner that will light when you go above the 0dB point. The further above 0dBVU, the brighter the LED becomes (on the small meter, it is very difficult to see as it is so tiny).

CUSTOM VIEW NAME: Normal Analog Meters
For this control, you need to supply a custom view name, that will be the same for all similar meters (based on size). RackAFX will do this for you, Here is the encoding information:

The custom view name also encodes the height of one image and the 0dB frame because these parameters are not natively available in the VSTGUI4 base class *CVuMeter*. Encoding these values is similar to the XY-Pad's X and Y tag encoding. The form of the custom view name is:

AnalogViewMeter_Haa_Zbb where aa is the height of one image in pixels, and bb is the zero dB frame number. For the BIG analog meter graphic, the Custom View name would be:

AnalogViewMeter_H94_Z52

CUSTOM VIEW NAME: Inverted Analog Meters
For inverted meters, the only difference is in the custom view name. For the inverted meters, you use **InvertedAnalogViewMeter_Haa_Zbb** so for the same meter above you would use

InvertedAnalogMeter_H94_Z52

Audio Processing Code

The audio processing code is simple and shown below. Note that the calculation from dB to a raw multiplier should be moved to the *preProcessAudioBuffers* function to reduce CPU usage, but I am leaving it here for simplicity of coding. Connecting the meters is a simple task: just write the audio sample value into the meter variable directly. The C++ object will handle rectifying and detecting the value.

```
bool PluginCore::processAudioFrame(ProcessFrameInfo& processFrameInfo)
{
    // --- fire any MIDI events for this sample interval
    processFrameInfo.midiEventQueue->fireMidiEvents(processFrameInfo.currentFrame);

       // --- do per-frame updates; VST automation and parameter smoothing
       doSampleAccurateParameterUpdates();

       // --- convert gain values ~NOTE~ this should be done once per buffer
       //     for better CPU usage; this is the simplest example
       double gainLeft = pow(10.0, gainLeft_dB / 20.0);
       double gainRight = pow(10.0, gainRight_dB / 20.0);

       // --- process left channel
       double outLeft = processFrameInfo.audioInputFrame[0] * gainLeft;

       double outRight = processFrameInfo.numAudioInChannels > 1 ?
                         processFrameInfo.audioInputFrame[1] * gainRight :
                         outLeft;

       // --- meters watch the output value; just write them out
       //     the meter's envelope detector will supply the ballistics
       leftMeterValue = outLeft;
```

```
        rightMeterValue = outRight;

    // --- FX Plugin:
    if(processFrameInfo.channelIOConfig.inputChannelFormat == kCFMono &&
       processFrameInfo.channelIOConfig.outputChannelFormat == kCFMono)
    {
            // --- out left
        processFrameInfo.audioOutputFrame[0] = outLeft;

        return true; /// processed
    }

    // --- Mono-In/Stereo-Out
    else if(processFrameInfo.channelIOConfig.inputChannelFormat == kCFMono &&
       processFrameInfo.channelIOConfig.outputChannelFormat == kCFStereo)
    {
            // --- repeat left output
        processFrameInfo.audioOutputFrame[0] = outLeft;
        processFrameInfo.audioOutputFrame[1] = outLeft;

        return true; /// processed
    }

    // --- Stereo-In/Stereo-Out
    else if(processFrameInfo.channelIOConfig.inputChannelFormat == kCFStereo &&
       processFrameInfo.channelIOConfig.outputChannelFormat == kCFStereo)
    {
            // --- output stereo values
        processFrameInfo.audioOutputFrame[0] = outLeft;
        processFrameInfo.audioOutputFrame[1] = outRight;

        return true; /// processed
    }

    return false; /// NOT processed
}
```

Compile and Test

Add the second analog and LED meters and rebuild the code. You should get the same GUI as in Figure 1.

References:

ASPiK Documentation: : http://aspikplugins.com/sdkdocs/html/index.html