Using XY Pads and VU Meter Objects in ASPiK

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:

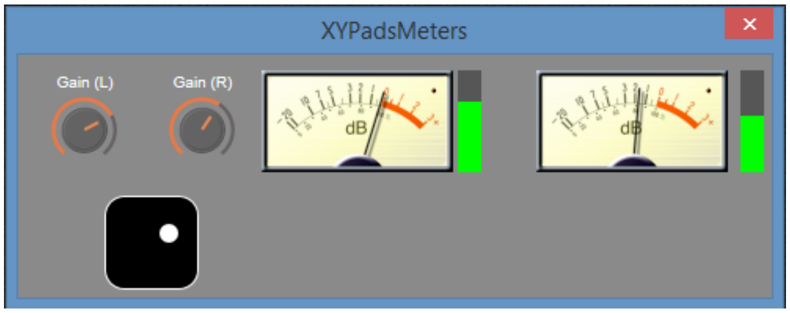


Figure 1: the plugin GUI controls and meters

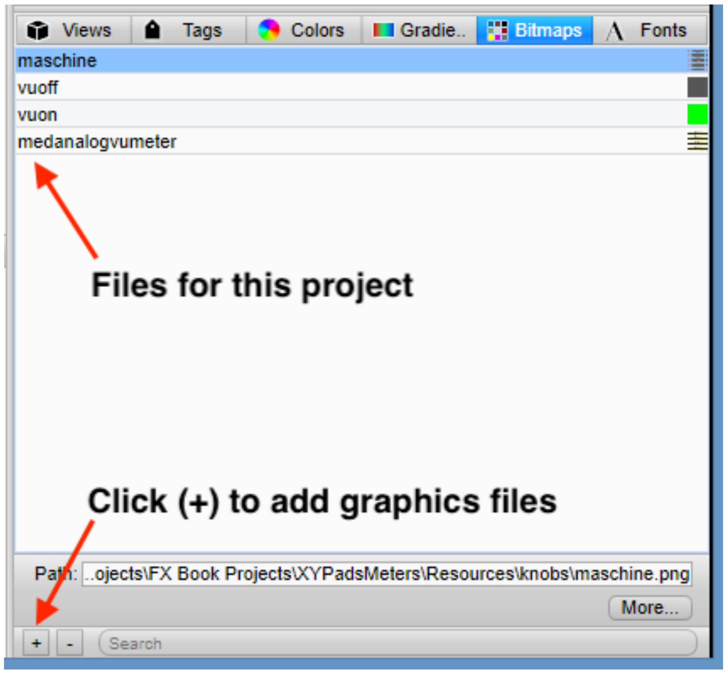
The XY Pad has the Left Gain knob mapped to the X-dimension and the Right Gain knob mapped to the Y-dimension. This document explains how to use each of these controls in the ASPiK GUI Designer. You should already know how to create the plugin and implement the left and right volume (gain) controls, and how to add and connect VU meters to your ASPiK plugin core. You should also know the basics of using the GUI Designer (see tutorial videos at https://ASPiK.com/support/ASPiK7-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 ASPiK or ASPiK project in the ***/Resources/knobs*** folder. The solid VU meters use the vuoff.png and vuon.png files, while the analog meter uses the medanalogvumeter.png file.

Importing Graphics

Once you have two knob and to meter controls defined in your plugin project, build your plugin and load it, open the GUI, and then go to the VSTGUI4 GUI Designer by <shift> Right clicking on the GUI. On the lower right, select the **Bitmaps** tab, click the (+) button, and browse to the */Resources/knobs* folder and select the maschine.png file, then add more graphics by going to */Resources/meters* and grabbing the **vuon.png** and **vuoff.png** files. Finally, use the analog vu meter pack that accompanies this document and import the **medanalogvumeter.png** file. Figure 2 shows how your Bitmaps panel should look after importing these files.



maschine
vuoff
vuon
medanalogvumeter

Path: ..ojects\FX Book Projects\XYPadsMeters\Resources\knobs\maschine.png

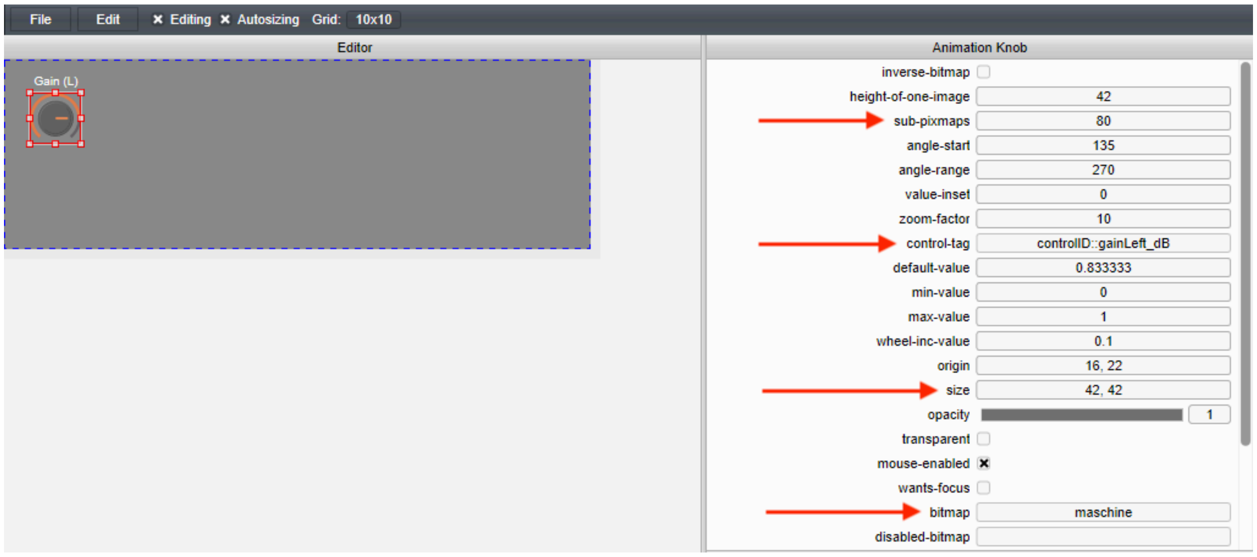
Figure 2: the Bitmaps tab in the VSTGUI4 GUI Designer with the necessary files already imported with the (+) button

CAnimKnobs

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

Place the Controls & Set Attributes

Drag and drop the knob controls and set their attributes as shown in Figure 3 (if this is new, you should watch the video tutorials in the introduction above).



inverse-bitmap	☐
height-of-one-image	42
sub-pixmaps	80
angle-start	135
angle-range	270
value-inset	0
zoom-factor	10
control-tag	controlID::gainLeft_dB
default-value	0.833333
min-value	0
max-value	1
wheel-inc-value	0.1
origin	16, 22
size	42, 42
opacity	1
transparent	☐
mouse-enabled	☒
wants-focus	☐
bitmap	maschine
disabled-bitmap	

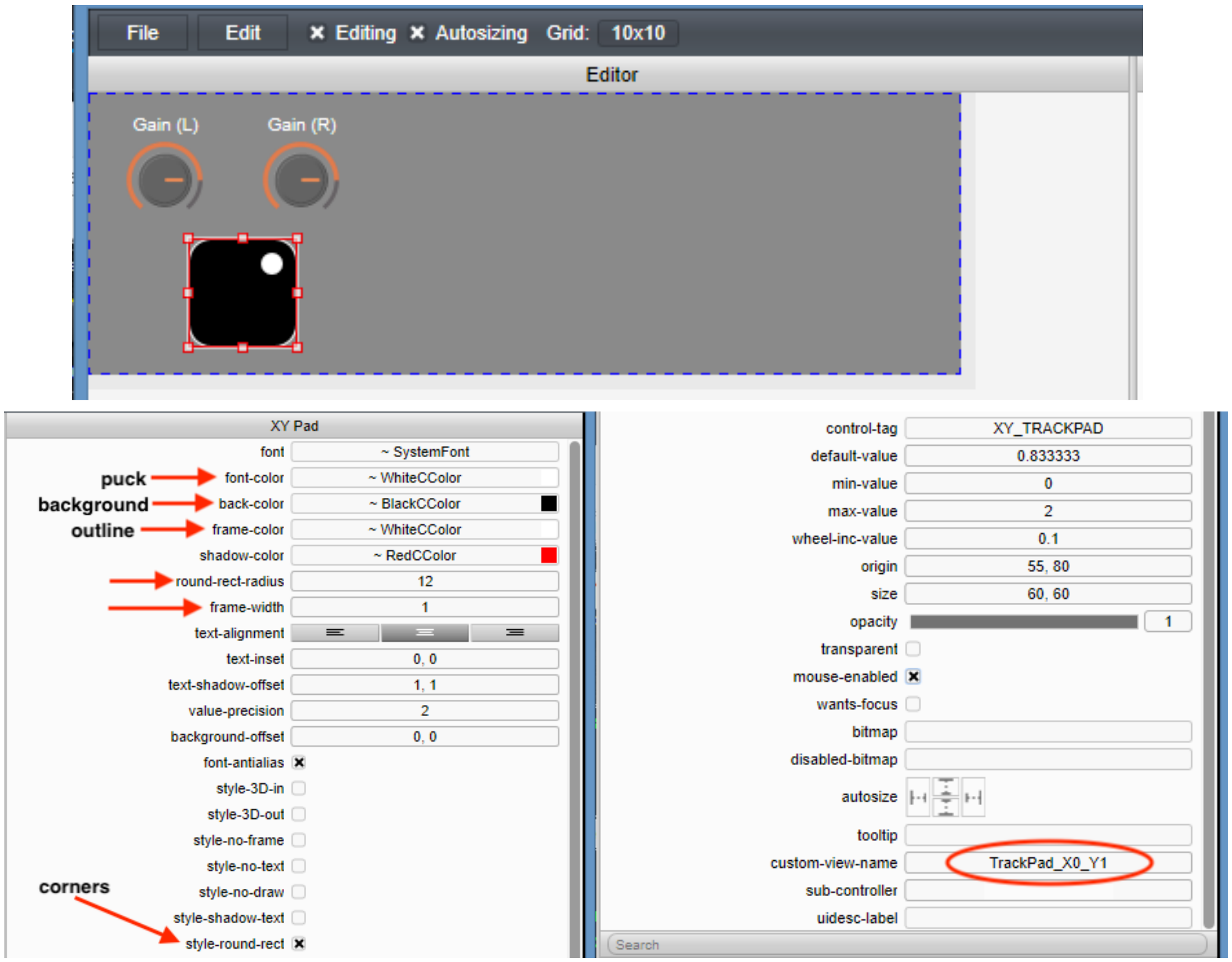
Figure 3: the attributes for the first *CAnimKnob* control; the red arrows show the values that need to be set for proper operation

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 drag a XY-Pad control onto the Designer pane. Set the attributes as shown in Figure 4. Notice the way the X and Y control values are encoded into the custom view name. This is documented in the ASPiK documentation: http://aspikplugins.com/sdkdocs/html/gui_designer11_d.html



font	~ SystemFont
font-color	~ WhiteCColor
back-color	~ BlackCColor
frame-color	~ WhiteCColor
shadow-color	~ RedCColor
round-rect-radius	12
frame-width	1
text-alignment	
text-inset	0, 0
text-shadow-offset	1, 1
value-precision	2
background-offset	0, 0
font-antialias	✓
style-3D-in	
style-3D-out	
style-no-frame	
style-no-text	
style-no-draw	
style-shadow-text	
style-round-rect	✓

control-tag	XY_TRACKPAD
default-value	0.833333
min-value	0
max-value	2
wheel-inc-value	0.1
origin	55, 80
size	60, 60
opacity	1
transparent	
mouse-enabled	✓
wants-focus	
bitmap	
disabled-bitmap	
autosize	
tooltip	
custom-view-name	TrackPad_X0_Y1
sub-controller	
uidesc-label	

Figure 4: XY Pad setup; note the encoding of the X and Y control ID values in the Custom View name; also, the font-color is actually the control-puck color

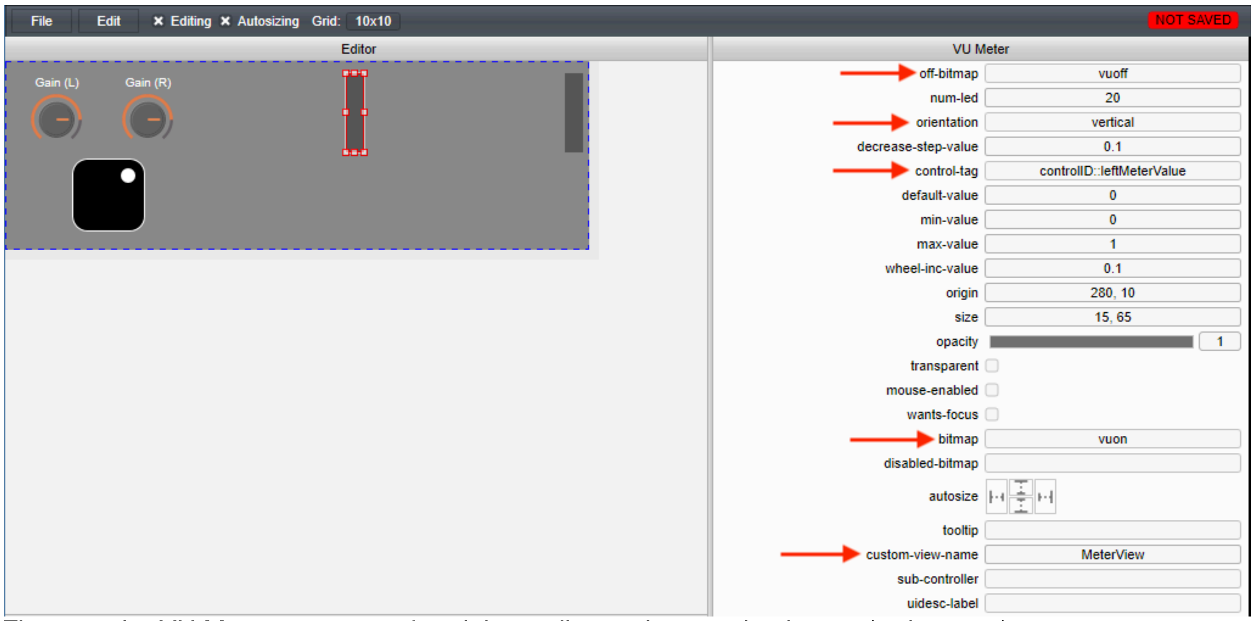
The custom view name will be decoded and the custom object will be created in the *PluginGUI::createView*( ) function; you can look at that code to see how the custom view string is parsed and the X/Y control IDs are extracted, as well as how the object is instantiated.

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 ASPiK meter graphics are designed to work with a built-in extended object called *CVuMeterEx* that is available in all ASPiK 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. 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.

Adding the Meters

In the GUI designer, drag a **VU Meter** object into the designer pane. 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 tag of the meter to connect it. Figure 5 shows how the VU meter setup panel will look for one of the controls.



VU Meter

off-bitmap	vuoff
num-led	20
orientation	vertical
decrease-step-value	0.1
control-tag	controlID::leftMeterValue
default-value	0
min-value	0
max-value	1
wheel-inc-value	0.1
origin	280, 10
size	15, 65
opacity	1
transparent	
mouse-enabled	
wants-focus	
bitmap	vuon
disabled-bitmap	
autosize	
tooltip	
custom-view-name	MeterView
sub-controller	
uidesc-label	

Figure 5: the VU Meter setup panel and the attributes that need to be set (red arrows)

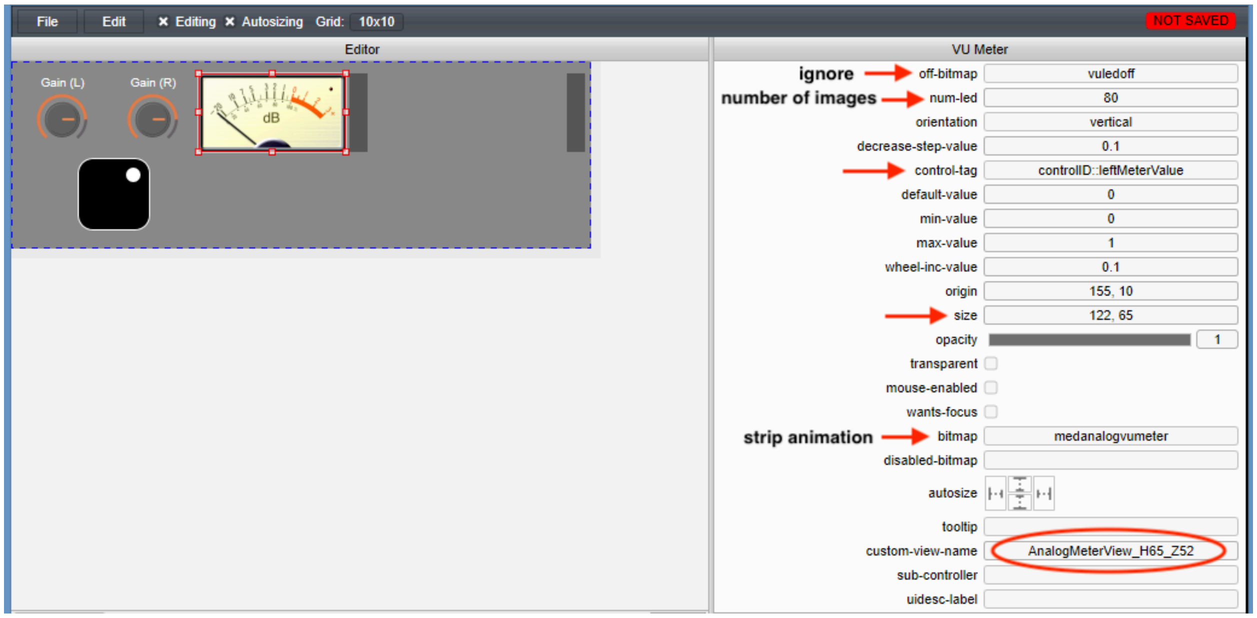
Notice that the Custom View name is set to *MeterView* which is the specific name for the normal LED meter. If you setup your meter in the ASPiK prototyping panel as inverted, then the custom view will change to *InvertedMeterView*. This will happen automatically for you when you select a new control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

Adding the Meters

In the ASPiK GUI Designer, drag another **VU Meter** into the Designer pane. Assign the *medanalogvumeter.png* file as the bitmap.



ignore → off-bitmap	vuledoff
number of images → num-led	80
orientation	vertical
decrease-step-value	0.1
control-tag	controlID::leftMeterValue
default-value	0
min-value	0
max-value	1
wheel-inc-value	0.1
origin	155, 10
size	122, 65
opacity	1
transparent	
mouse-enabled	
wants-focus	
strip animation → bitmap	medanalogvumeter
disabled-bitmap	
autosize	
tooltip	
custom-view-name	AnalogMeterView_H65_Z52
sub-controller	
uidesc-label	

Figure 6: the VU meter setup need some additional parameters; the **num-led** attribute is the number of images in the strip animation, while the custom view encodes the height of one image, and the zero dB frame

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

Size: as with the normal VU meters, or the CAnimKnob this parameter sets the dimensions of a <u>single frame or sub-image</u>. 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 122,65 for the medanalogvumeter.png 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, <u>the 0dB frame is number 52</u> (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). ASPiK 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