A Custom GUI for MIDI Learn Functions

Will Pirkle

A user requested information about making a GUI with MIDI Learn controls. This is an especially good candidate for using the Advanced GUI API features in RackAFX. RackAFX (like AU, AAX and VST) only updates internal variables in your plugin when the audio is streaming through it. This is part of a thread-safety mechanism present in all four APIs. However, with MIDI Learn, the user will be interacting with the GUI when audio may or may not be streaming. So, we need a mechanism to handle these interactions as well as update the GUI as the user requested:

- user turns on the MIDI Learn button
- user presses a key on the connected MIDI keyboard (or built-in RackAFX Piano Control)
- the MIDI Learn Button toggles states and turns off
- the MIDI Note number is stored, and then displayed in a text label for the user to see

This is actually a fairly simple and straightforward use of the Advanced GUI API. And, of course it will also port out to your VST, AU and AAX plugins as well. You could also display information about the control's value, if using CC's rather than note events.

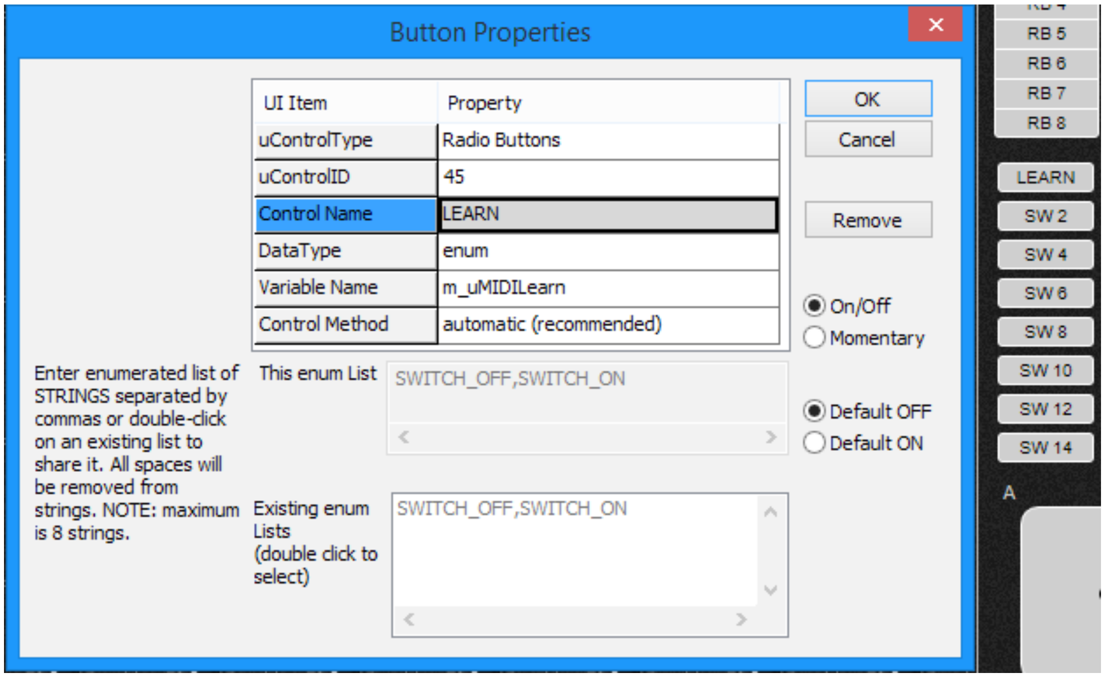
Controls

In order to implement this GUI, we will need to create custom controls for two VSTGUI4 objects:

- COnOffButton: this is the MIDI Learn Button
- CTextLabel: this is the MIDI note "blurb" that we display for the user

In RackAFX, we create a new project and then assign two GUI controls on the main interface. One control is a 2-state switch labeled "LEARN" and the other control is a knob that transmits/receives an integer value (this knob is optional and we won't really be using it for anything other than to create a storage variable for the MIDI Note Number of the learned control). The control setups look like this:

MIDI Learn Button (first in the left column of 2-state switches)


Button Properties

UI Item	Property
uControlType	Radio Buttons
uControlID	45
Control Name	LEARN
DataType	enum
Variable Name	m_uMIDILearn
Control Method	automatic (recommended)

[✓ On/Off] Momentary

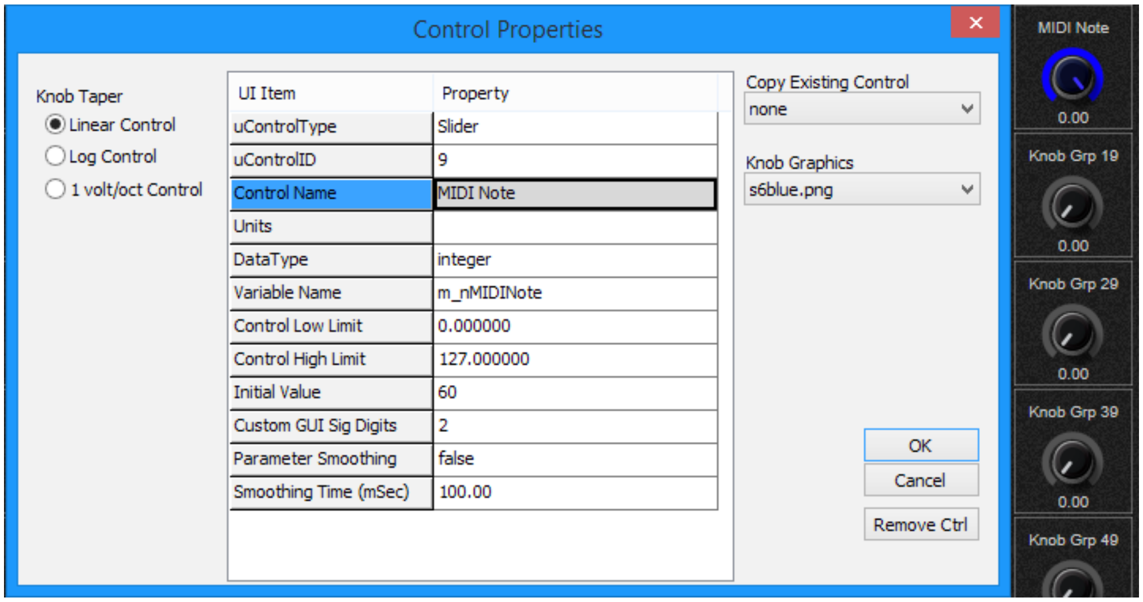
[✓ Default OFF] Default ON

Enter enumerated list of STRINGS separated by commas or double-click on an existing list to share it. All spaces will be removed from strings. NOTE: maximum is 8 strings.

This enum List: SWITCH_OFF,SWITCH_ON

Existing enum Lists (double click to select): SWITCH_OFF,SWITCH_ON

MIDI Note Knob



Knob Taper
[✓ Linear Control]
Log Control
1 volt/oct Control

UI Item	Property
uControlType	Slider
uControlID	9
Control Name	MIDI Note
Units	
DataType	integer
Variable Name	m_nMIDINote
Control Low Limit	0.000000
Control High Limit	127.000000
Initial Value	60
Custom GUI Sig Digits	2
Parameter Smoothing	false
Smoothing Time (mSec)	100.00

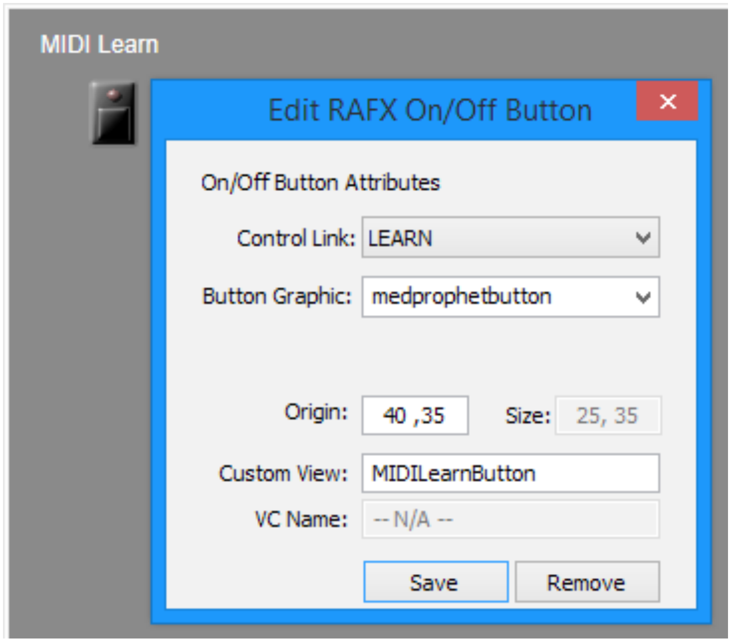
The connected variables show up in the .h file:

```
// ADDED BY RACKAFX -- DO NOT EDIT THIS CODE!!! ----------------------------------- //
//  **--0x07FD--**

int m_nMIDINote;
UINT m_uMIDILearn;
enum{SWITCH_OFF,SWITCH_ON};

// **--0x1A7F--**
// ------------------------------------------------------------------------------- //
```

RackAFX GUI

In the GUI Designer, we need to add an on-off button (here I use the Medium Prophet button graphic) and some text labels to display information. We will need to fill-in or alter the CustomView attribute for the two controls of interest: the MIDI Learn Button and the text label to display the note information. For the button, I named the CustomView "MIDILearnButton" and for the text label, the CustomView is changed to "MIDINoteLabel."

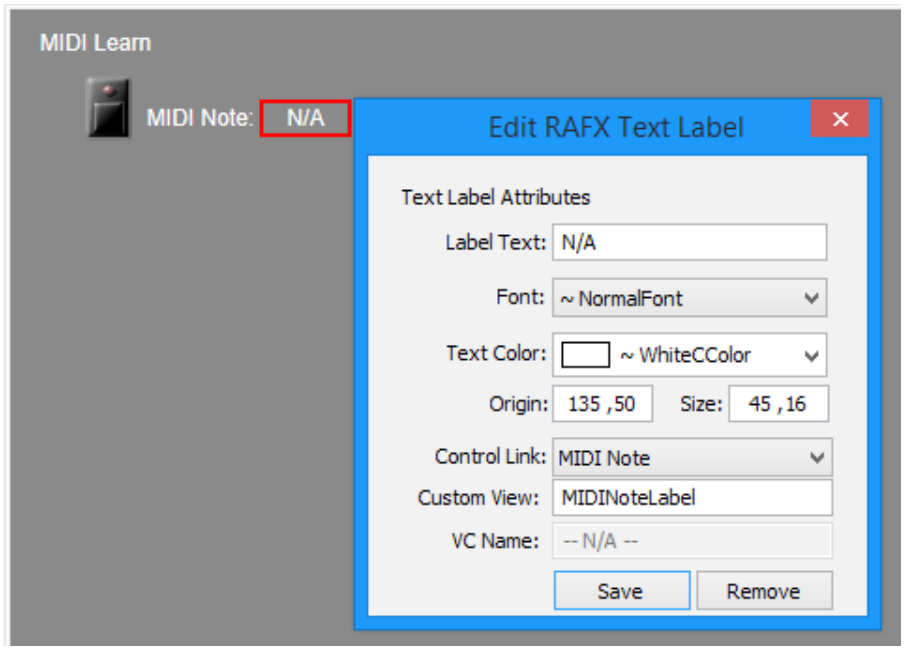
First the button; note that the control is linked to the LEARN control:



On/Off Button Attributes

Control Link: LEARN
Button Graphic: medprophetbutton
Origin: 40 ,35
Size: 25, 35
Custom View: MIDILearnButton
VC Name: -- N/A --

Next is the text label, which I've initialized to "N/A"; note that the control is linked to the MIDI Note control:



Text Label Attributes

Label Text: N/A
Font: ~ NormalFont
Text Color: ~ WhiteCColor
Origin: 135 ,50
Size: 45 ,16
Control Link: MIDI Note
Custom View: MIDINoteLabel
VC Name: -- N/A --

Plugin-Side Code

So, setting up the CustomView attributes is very easy. Now, we need to create the custom view controls from within the plugin and save (cache) the pointers to these controls. You should already be familiar with the Advanced GUI API tutorials in Modules 2 - 4 which demonstrate how to setup the project for this kind of operation. After installing the VSTGUI4.3 library and adding the ViewAttributes files to your project (this is covered in the tutorials), we need to un-comment the *#includes* at the top:

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

and then add our own pointer variables in the user-variable section, one for each custom control. Note also the addition of the GUI Helper object that will greatly simplify decoding information from the host about the control we are creating.

```
        // Add your code here: --------------------------------------------------------- //
        // --- helper for custom view stuff
        CVSTGUIHelper m_GUIHelper;

        // --- the MIDI Learn Button
        COnOffButton* m_pMIDILearnButton;

        // --- MIDI Text blurb
        CTextLabel* m_pMIDIBlurb;


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

In the .cpp file, we initialize the pointers to NULL in the constructor:

```
        // --- custom view stuff
        m_pMIDILearnButton = NULL;
        m_pMIDIBlurb = NULL;
```

In our *showGUI( )* function, we need to un-comment the customization code and add our own stuff to it. First, for the **GUI_CUSTOMVIEW** message, we decode the *customViewName* string and then create the needed controls (see the tutorials for much more information on this). We create and store the pointers for the two controls, based on their CustomView attributes, then return the pointers cloaked as NULL. Notice the customization of the text label (transparency, and text alignment):

```
case GUI_CUSTOMVIEW:
{
        if(strcmp(info->customViewName, "MIDILearnButton") == 0)
        {
                // --- get the needed attributes with the helper
                const CRect rect = m_GUIHelper.getRectWithVSTGUIRECT(info->customViewRect);
                const CPoint offsetPt = m_GUIHelper.getPointWithVSTGUIPOINT(
                                                                        info->customViewOffset);
                CBitmap* pBitmap = m_GUIHelper.loadBitmap(info);

                // --- create COnOffButton and cache pointer
                //     For more info, see the class definition for COnOffButton and the VSTGUI4 docs
                m_pMIDILearnButton = new COnOffButton(rect, (IControlListener*)info->listener,
                                                                info->customViewTag, pBitmap);
        // --- decrement ref count
        if(pBitmap)
            pBitmap->forget();

        // --- return control cloaked as a void*
        return (void*)m_pMIDILearnButton;
    }
    if(strcmp(info->customViewName, "MIDINoteLabel") == 0)
    {
        // --- get the needed attributes with the helper
        const CRect rect = m_GUIHelper.getRectWithVSTGUIRECT(info->customViewRect);

        // --- create CTextLabel and chache pointer
        //     For more info, see the class definition for CTextLabel and the VSTGUI4 docs
        m_pMIDIBlurb = new CTextLabel(rect);

        // --- make background transparent (can play with this for your own customization)
        m_pMIDIBlurb->setTransparency(true);

        // --- align text to left
        m_pMIDIBlurb->setHoriAlign(kLeftText);

        // --- return control cloaked as a void*
        return (void*)m_pMIDIBlurb;
    }
    return NULL;
}
```

Next, for the **GUI_DID_OPEN** message, we initialize the values of these controls:

```
case GUI_DID_OPEN:
{
    // --- initialize any cached controls
    if(m_pMIDILearnButton)
    {
        // --- turn off button
        m_pMIDILearnButton->setValue(0.0);
    }
    if(m_pMIDIBlurb)
    {
        // --- set text to Not Available
        m_pMIDIBlurb->setText("N/A");
    }

    return NULL;
}
```

And, in the **GUI_WILL_CLOSE** message handler, we just NULL the variables (they are self-deleting so we don't need to destroy them - see the tutorials for more information):

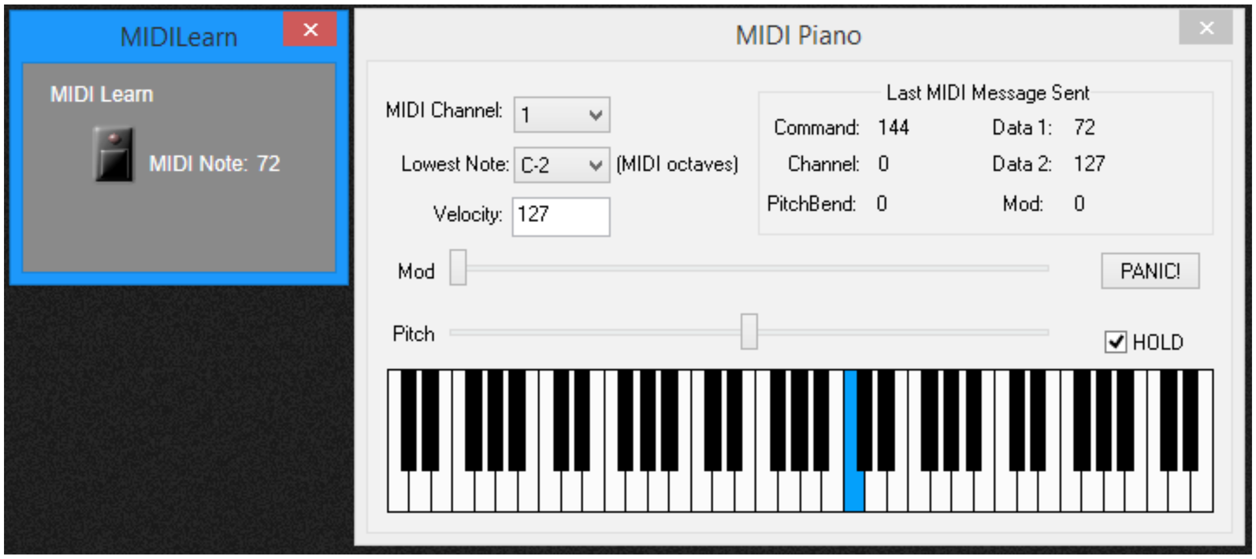
```
case GUI_WILL_CLOSE:
{
	if(m_pMIDILearnButton)
		// --- reset to NULL
		m_pMIDILearnButton = NULL;

	if(m_pMIDIBlurb)
		// --- reset to NULL
		m_pMIDIBlurb = NULL;

	return NULL; // all OK
}
```

The only thing left to do is modify the *midiNoteOn( )* method in the plugin to pick up the learned note number and manipulate the GUI. This is straightforward if you have worked through the first few tutorials. We also use the built-in helper function *uintToString( )* to convert the MIDI note number into a *char** for use in the GUI - it is important to delete this pointer when done as shown:

```
// --- handle note on for learning
bool __stdcall CMIDILearn::midiNoteOn(UINT uChannel, UINT uMIDINote, UINT uVelocity)
{
	// --- if GUI is visible, manipulate it
	if(m_pMIDILearnButton)
	{
		// --- is Learn Button on?
		if(m_pMIDILearnButton->getValue() == 1.0)
		{
			// --- clear the variable in case this happens while audio is flowing
			m_uMIDILearn = SWITCH_OFF;

			// --- clear the switch on the GUI
			m_pMIDILearnButton->setValue(0.0);

			// --- save the note number
			m_nMIDINote = uMIDINote;

			// --- show it in text; use helper to convert UNIT to string
			char* pNote = uintToString(uMIDINote);

			// --- change the text blurb
			if(m_pMIDIBlurb)
				m_pMIDIBlurb->setText(pNote);

			// --- free memory
			delete [] pNote;
		}
	}
	return true;
}
```

Build and Test

Thats it! Now we just build and test the plugin; the custom GUI is used for the MIDI learn operation and works as the user prescribed. The screen shot here shows the GUI after the C above middle-C has been pressed, with the MIDI Learn switch turned on - the switch has been turned off programmatically, and the MIDI Note number has been set in the GUI text label:



MIDI Note: 72

MIDI Channel: 1
Lowest Note: C-2
Velocity: 127

Command: 144
Data 1: 72
Channel: 0
Data 2: 127
PitchBend: 0
Mod: 0

Finally, notice that the GUI does not contain the Knob control we setup earlier, though it does use the control's variable to store the MIDI note number. You might optionally get rid of this control, and simply declare your own variable to store the data.