Case Study

An enhanced user experience

Using VSTGUI's Custom Views and Sub-Controllers

Steve Dwyer

After having completed my Certificate of Higher Education in Mathematical Sciences in the summer of 2021, I've been working through Will Pirkle's book *Designing Audio Effect Plugins in C++*. I've since completed a number of effects pedals.

I would like to invite you to visit my GitHub repository https://github.com/DoomyDwyer/ASPiKProjects where my work is available for review and the plugins, available as VST3 and Audio Units, as well as presets and user guides, can be downloaded. The README.md file displays a screenshot of each of the effects currently available. Releases available to download here:

https://github.com/DoomyDwyer/ASPiKProjects/releases

All GUI design was done in the open-source graphics design tools GIMP & Inkscape, and the knobs were created using JKnobMan. The plugins were built in C++ using Will Pirkle's open-source ASPiK framework[1]

1. Getting a kick switch to behave like its counterpart on a hardware effects pedal

After completing my first set of plugins, I felt that the pedal GUIs missed some important characteristics. In particular, the kick switch didn't behave like a real kick switch on an effects pedal:

- The kick switch on a guitar effects pedal changes the state of the pedal (between Bypass and On, or vice versa) when pressed down *only*
- The state of the pedal is unaffected when the kick switch is released
- A status LED displays the state of the pedal, i.e. On or in Bypass mode. As in a hardware pedal however, touching or clicking this LED would not be expected to change the state of the pedal. In other words, this control needs to be read-only.

The standard components offered in the VSTGUI 4 SDK didn't offer the required behaviour:

- The CKickButton control acts as a momentary On/Off switch – the linked control tag changes state on both mouse down and mouse up events. Although this control displayed the desired animation, whereby the kick button travels upwards again on releasing the mouse button, using this control would leave the effect on *only* when the mouse is held down.
- The COnOffButton reacts to click events, i.e. the mouse down only, and so could be used to change the state of the pedal on each click, however using this control didn't produce the desired animation, as the kick switch would remain 'down' as long as the pedal was On.

I opted to encapsulate both the kick switch, defined as a CKickButton, and the status LED, a COnOffButton, with its *mouse-enabled* attribute set to "false", into a custom view and to implement the desired behaviour in a custom sub-controller. The custom view, defined as a template, is

[1] http://www.aspikplugins.com/

configured to use the sub-controller in the plugin's `PluginGUI.uidesc`. The status LED's ability to respond to mouse events is also disabled in this same file.

The `KickSwitchController` class, an implementation of the VSTGUI `IController` interface, keeps a reference to an instance of `CKickButton` (the 'activator'), as well as an instance of `COnOffButton` (the 'indicator'). Each of these controls is linked to a control tag on the plugin, `fx_OnOff_Toggle` and `fx_On`, respectively. The control tag `fx_On` is used in the plugin's audio processing thread, to determine whether the effect is On or in Bypass mode. `fx_OnOff_Toggle` Is used simply to determine in the sub-controller what the state is of the kick switch after its value has changed – down or up.

Once the sub-controller is registered with the gui, events will invoke callback methods on the sub-controller, as follows:

- `verifyView` will be called, when the plugin is loaded, once for each member of the custom view. Using dynamic casts, it can be determined which of the controls is being passed, and these are then assigned to the member objects of the `KickSwitchController` object accordingly
- The `valueChanged` method is called each time the value of one of the controls contained within the custom view is changed by the user.

The `valueChanged` method needed to implement the following behaviour:

- Whenever `fx_OnOff_Toggle`'s value is 1, the kick switch has been depressed, in which case the state of the `fx_On` control, represented in the GUI by the read-only status LED, needs to be inverted.

This was achieved with the following code in the sub-controller's `valueChanged` method:

```
void valueChanged(CControl* control) override
{
    if (control == activator)
    {
        // On Mouse Down *only*, hence when control's value normalized = 1
        if (isEqual(control->getValueNormalized(), 1.0f))
        {
            // set the control visually and flip its value (boolean negate using arithmetic)
            indicator->setValueNormalized(indicator->getValueNormalized() * -1 + 1);

            // --- do the value change at parent level, to set on plugin
            parentController->valueChanged(indicator);
            // Force a repaint
            indicator->invalid();
        }
    }

    // --- do the value change at parent level, to set on plugin
    parentController->valueChanged(control);
}
```

Despite the added complexity, I believe the effort required to implement this to be worthwhile, since the user experience is central to the success of any application, including audio software & plugins. Providing the user with visual confirmation of their actions, such as having a kick switch control which behaves in a similar fashion to how a guitarist or other musician would expect a kick switch to behave, gives the user confidence that they are in control of things, and that they have a satisfactory understanding of how to use the software.

2. Using tooltips to display a knob's current value

A second enhancement to my plugins was actually a divergence from standard hardware effects pedals. In this case it adds additional behaviour, which one would only expect to see in a software product. Nevertheless, I felt it would add value to the user experience.

I've authored User Guides for each of my pedals, in which, amongst other things, the range and units of the pedal's knobs are mentioned. In an analog hardware pedal, the actual values of the pots under the knobs aren't displayed to the user. I felt however, that since I'd already made mention of the units & ranges of each knob, it would help the user if they were able to see the value to which a knob is currently set.

I felt that a musician would accept this enhancement as a typical behaviour of a software product, which wouldn't be possible on a hardware pedal, while keeping the appearance of the plugin's GUI in line with what one would expect to find on an analog effects pedal, which could have been built at any point from the 1960s onwards.

In the VSTGUI 4 SDK by default, a tooltip text for a control can only be set at design time and to update this value at runtime requires a custom implementation. This was also achieved using custom views and sub-controllers.

The custom implementations of the VSTGUI `IController` interface, used for the kick switch and tooltip examples discussed above, are contained in the C++ header file `gui/custompedalviews.h`. This header file is maintained in a second repository, which contains some common classes used across all of the effects pedals, and is available here: https://github.com/DoomyDwyer/ASPiKCommon.

Here is a screenshot of one of my plugins, "Memento", a ducking delay: