JUCE Seaboard Tutorial Notes

Introduction

This folder contains three tutorials that a Jules Utility Class Extensions (JUCE), an all-encompassing C++ class library for developing cross-platform software, to create a data visualiser and synthesiser controlled by the Seaboard. The tutorials, along with the Seaboard C++ Library Files they use, will get any C++ programmer up and running, and hopefully inspire some hacks.

Tutorial 1: Visual Feedback

- Navigate to JUCE/SeaboardVisuals and open up `SeaboardVisuals.jucer` in the Introjucer app. If you've never used JUCE before, you can find the executable in JUCE/Introjucer.
- Click on 'Config' in the upper left and then click 'Save Project and Open in Xcode…' at the bottom.
- In Xcode, expand the SeaboardVisuals/Source folder and click on `SeaboardComponent.h`.

When our project is compiled, a SeaboardComponent will be created and presented as a simple window. The SeaboardComponent class relies on two main components: 1) a Seaboard object which manages all the message callbacks from the Seaboard hardware, and 2) a Seaboard Visualiser object with handles the MIDI visualisation that is rendered to the SeaboardComponent Window

Take a closer look at the Seaboard class. It is a simple MidiInputCallback subclass that decodes messages from the Seaboard and triggers one of 5 callback methods. To respond to these callbacks in a class of your own, you simple need to subclass the Seabaord::Listener class, and implement whichever methods your would like to listen to. You then need to call Seaboard::addListener() to register your class to the Listener list.

A great example of a Seaboard::Listener subclass is the SeaboardVisualiser. This only listens to note on and note off messages, and stores the data from these callbacks in a ValueTree object. The view is then re-drawn using this updated data to visualise the messages coming from the Seaboard.

Value Trees

The `ValueTree::Listener` protocol provides methods to respond to changes to Value Tree objects. ValueTree objects are tree-like data structures provided by the JUCE library. Value Trees can contain any number of properties and any number of child trees. In our implementation, we add a value tree called `theSeaboardData`, to which we will add child value trees to represent each current note being played on the Seaboard. For example, the tree might be represented visually as such:

```
theSeaboardData
|
|
    Note - Channel 1
    |
    -MIDI Note Number = 57
|
|
    Note - Channel 2
    |
    -MIDI Note Number = 100
|
|
```

The tree above shows a parent tree (`theSeaboardData`) with two children that represent the notes currently being played. Each of the notes has a MIDI Note Number property.

Click on `SeaboardVisualiser.cpp`. In our implementation, we add the `SeaboardVisualiser` as a listener to the seaboard data value tree object (`theSeaboardData.addListener(this)` on line 30). This allows us to respond to changes to the Seaboard data value tree. This is implemented at lines 110 - 133. We simply repaint whenever there is a change to the Seaboard data tree.

`Seaboard::Listener`

The `Seaboard::Listener` class provides methods to respond to MIDI messages coming from the Seaboard. The two methods we overwrite from the `Seaboard::Listener` class are `seaboardDidGetNoteOn()` and `seaboardDidGetNoteOff()`. In each of these methods we simply add or remove new value tree objects from the seaboard data value tree.

`Component`

The `paint()` method is the main drawing loop for a JUCE component object. It will be called continuously. In our implementation, we loop through each child tree in the Seaboard data tree (each note) and draw circles that represent the position of each note on the Seaboard.

Tutorial 2: Simple Audio

- Navigate to JUCE/SeaboardSimpleSynth and open up `SeaboardSimpleSynth.jucer`.
- Click on ‘Config’ in the upper left and then click ‘Save Project and Open in Xcode…’ at the bottom.
- In Xcode, expand the SeaboardSimpleSynth/Source folder and click on `SeaboardComponent.cpp`

Voice Setup

- Look at the constructor method `SeaboardComponent::SeaboardComponent()`.

Notice at line 28 that we need to add voice objects to our `SeaboardPlayer` object. These voice objects will determine the kind of sound that is produced in response to Seaboard MIDI messages. We add nine instances of voice objects so that we can handle polyphonic information.

Sound Generation

- Navigate to `SineWaveVoice.cpp`.

This simple class produces sinusoidal tones in response to incoming Seaboard messages. The `startNote()` and `stopNote()` functions implement Note On and Note Off functionality. The main sound generation loop happens in `renderNextBlock()`. In this example we simply fill in sound buffers of a sinusoidal signal.

Try running the application and playing some notes. Notice that we get simple Note On/Note Off sound response, but this is fairly limited. For example, we have nothing happening in response to pitch bend or aftertouch gestures. This will be implemented in the SeaboardSynth tutorial.

Tutorial 3: SeaboardSynth

This example expands on the previous two by adding pitch bend and aftertouch response to the voice objects, and by adding a visualiser to visualise the pitch bend and aftertouch values of each note, independently.

- Navigate to JUCE/SeaboardSynth and open up `SeaboardSynth.jucer`.
- Click on ‘Config’ in the upper left
- Click ‘Save Project and open in Xcode…’ at the bottom.

In Xcode, expand the SeaboardSynth/Source folder and click on `SeaboardVisualiser.cpp`. This implementation is similar to our visualisation implementation from Tutorial 1, however we have added response methods for pitch bend and aftertouch. We have also added functionality to change circle size depending on aftertouch value. Pitch bend is visualised by drawing lines from the edges of the circles that point straight up for 0 pitch bend and roll round to the left for pitch bend down, and roll to the right for pitch bend up.

Click on `SineWaveVoice.cpp`. The only difference between this implementation and the implementation from Tutorial 2 is the addition of response methods for pitch bend and aftertouch (lines ~67-83).

Try compiling and running. We have implemented a simple polyphonic synthesiser with both audio and visual feedback.

Thanks for trying our tutorials! If you have feedback, we would love to hear from you
support@roli.com