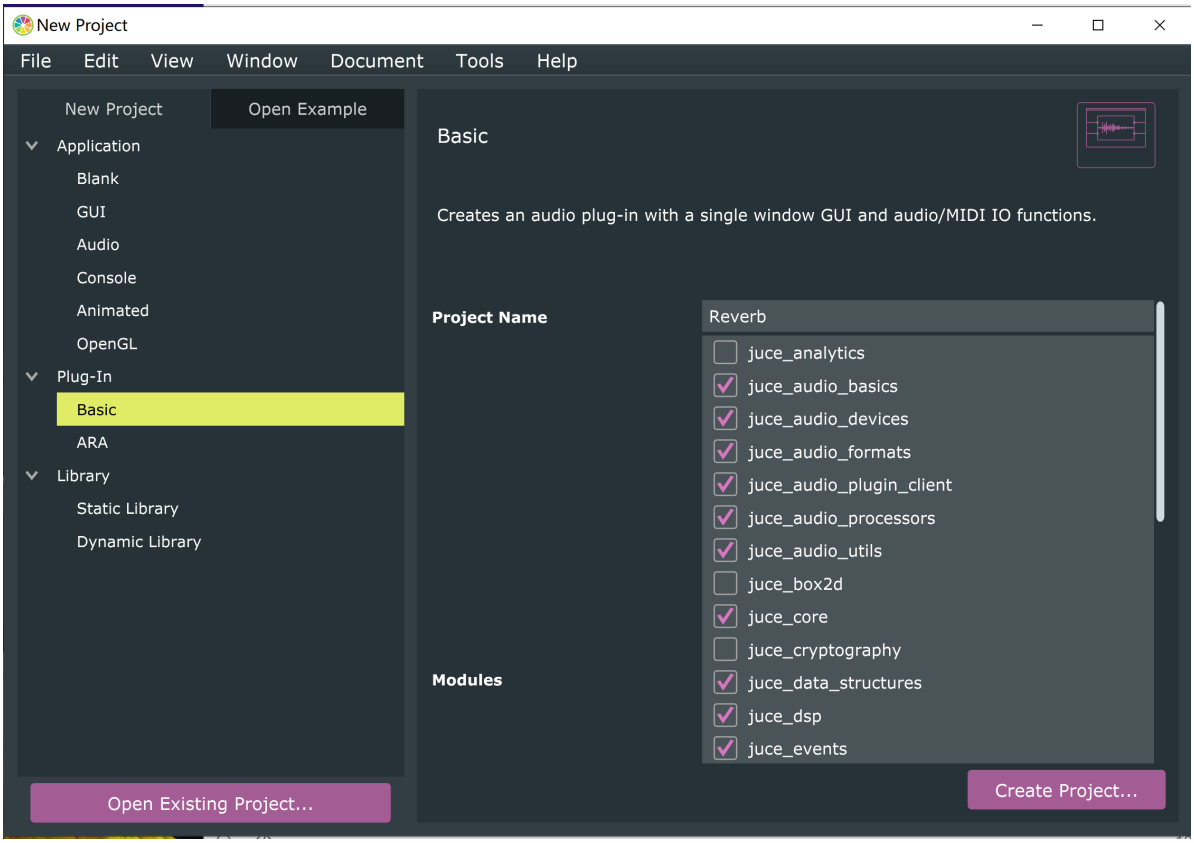
Preview

Start a new project

Open Projucer, and start a new project named "Reverb". Remember to tick juce-dsp in Module so that we are able to use dsp class in JUCE.



DSP

APVTS

I recommend you to use APVTS("AudioProcessorValueTreeState") to control a variety of parameters, because it is much convenient and simple than using the outdated slider or combo box class, for example. First, we need to create a public member for APVTS:

```
class SimpleReverbAudioProcessor : public juce::AudioProcessor
{
public:
. . .
static juce::AudioProcessorValueTreeState::ParameterLayout createParameterLayout
();        juce::AudioProcessorValueTreeState apvts { *this, nullptr,
"Parameters", createParameterLayout()};
```

We can put any parameters we like in createParameter(). In the project, we are designing a reverb plugin, so we are going to use" juce::dsp::Reverb::Parameters(juce::Reverb::Parameters)". Here are the parameters:

- roomSize
- damping
- wetLevel
- dryLevel
- width
- freezeMode

These parameters are matching with different rotary knob in the plugin, but it is inconsistent to divide dry and wet level to two rotary knobs(Just like other Reverb plugin, they are usually represented within one knob), so we can combine them together as a dry/wet rotary knob. Meanwhile, I think it will be better to display their values as percentage. These is what we should write with APVTS in the program code:

```
juce::AudioProcessorValueTreeState::ParameterLayout
SimpleReverbAudioProcessor::createParameterLayout()
{
    juce::AudioProcessorValueTreeState::ParameterLayout layout;

    layout.add (std::make_unique<juce::AudioParameterFloat> ("Room Size",
                                                             "Room Size",

juce::NormalisableRange<float> (0.0f, 1.0f, 0.001f, 1.0f),
                                                            0.5f,
                                                            juce::String(),

juce::AudioProcessorParameter::genericParameter,
                                                            [](float value,
int) {
                                                               if (value * 100
< 10.0f)
                                                                   return
juce::String (value * 100, 0);
                                                               else if (value *
100 < 100.0f)
                                                                   return
juce::String (value * 100, 0);
```

```
                                                                    else
                                                                        return
juce::String (value * 100, 0); },
                                                                nullptr));

    layout.add (std::make_unique<juce::AudioParameterFloat> ("Damping",
                                                                "Damping",
juce::NormalisableRange<float> (0.0f, 1.0f, 0.001f, 1.0f),
                                                                0.5f,
                                                                juce::String(),
juce::AudioProcessorParameter::genericParameter,
                                                                [](float value,
int) {
                                                                    if (value * 100
< 10.0f)
                                                                        return
juce::String (value * 100, 2);
                                                                    else if (value *
100 < 100.0f)
                                                                        return
juce::String (value * 100, 1);
                                                                    else
                                                                        return
juce::String (value * 100, 0); },
                                                                nullptr));


    layout.add (std::make_unique<juce::AudioParameterFloat> ("Width",
                                                                "Width",
juce::NormalisableRange<float> (0.0f, 1.0f, 0.001f, 1.0f),
                                                                0.5f,
                                                                juce::String(),
juce::AudioProcessorParameter::genericParameter,
                                                                [](float value,
int) {
                                                                    if (value * 100
< 10.0f)
                                                                        return
juce::String (value * 100, 2);
                                                                    else if (value *
100 < 100.0f)
                                                                        return
juce::String (value * 100, 1);
                                                                    else
                                                                        return
juce::String (value * 100, 0); },
                                                               nullptr));

    layout.add (std::make_unique<juce::AudioParameterFloat> ("Dry/Wet",
                                                                "Dry/Wet",
```

```
juce::NormalisableRange<float> (0.0f, 1.0f, 0.001f, 1.0f),
                                                                0.5f,
                                                                juce::String(),

juce::AudioProcessorParameter::genericParameter,
                                                                [](float value,
int) {
                                                                    if (value * 100
< 10.0f)
                                                                        return
juce::String (value * 100, 2);
                                                                    else if (value *
100 < 100.0f)
                                                                        return
juce::String (value * 100, 1);
                                                                    else
                                                                        return
juce::String (value * 100, 0); },
                                                                nullptr));

    layout.add (std::make_unique<juce::AudioParameterBool> ("Freeze", "Freeze",
false));

    return layout;
}
```

juce::dsp::Reverb

we have APVTS ready, so we are going to design the Reverb section. First we need to create a member of"juce::dsp::Reverb::Parameters". Be aware, we going to create two members of "juce::dsp::Reverb" to support two-channel stereo:

```
class SimpleReverbAudioProcessor : public juce::AudioProcessor
{
. . .
private:
    juce::dsp::Reverb::Parameters params;
    juce::dsp::Reverb leftReverb, rightReverb;
 //==============================================================================
====
    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (SimpleReverbAudioProcessor)
};
```

Second, we need to create member "ProcessSpec" to store information that is essential to initialize the created members in Reverb:

```
void SimpleReverbAudioProcessor::prepareToPlay (double sampleRate, int
samplesPerBlock)
{
    juce::dsp::ProcessSpec spec;
    spec.sampleRate = sampleRate;
    spec.maximumBlockSize = samplesPerBlock;
    spec.numChannels = 1;

    leftReverb.prepare (spec);
    rightReverb.prepare (spec);
}
```

Third, we design how the sound will be processed. Like I mentioned before, we need to combine dry/wet level as a rotary knob, so we will use (1-value of wet) to set up the value of dry. This is how is works:

```
void SimpleReverbAudioProcessor::processBlock (juce::AudioBuffer<float>& buffer,
 juce::MidiBuffer& midiMessages)
{
    juce::ScopedNoDenormals noDenormals;
    auto totalNumInputChannels = getTotalNumInputChannels();
    auto totalNumOutputChannels = getTotalNumOutputChannels();
    for (auto i = totalNumInputChannels; i < totalNumOutputChannels; ++i)
    buffer.clear (i, 0, buffer.getNumSamples());
    params.roomSize = *apvts.getRawParameterValue ("Room Size");
    params.damping = *apvts.getRawParameterValue ("Damping");
    params.width = *apvts.getRawParameterValue ("Width");
    params.wetLevel = *apvts.getRawParameterValue ("Dry/Wet");
    params.dryLevel = 1.0f - *apvts.getRawParameterValue ("Dry/Wet");
    params.freezeMode = *apvts.getRawParameterValue ("Freeze");

    leftReverb.setParameters (params);
    rightReverb.setParameters (params)
    juce::dsp::AudioBlock<float> block (buffer);

    auto leftBlock = block.getSingleChannelBlock (0);
    auto rightBlock = block.getSingleChannelBlock (1);

    juce::dsp::ProcessContextReplacing<float> leftContext (leftBlock);
    juce::dsp::ProcessContextReplacing<float> rightContext (rightBlock);

    leftReverb.process (leftContext);
    rightReverb.process (rightContext);
}
```

Finally, we have finished the dsp section of the Reverb

UI

CustomLookAndFeel

First, we need to customize LookAndFeel, which is the fundamental elements of our UI design. Open projucer, create a new header and cpp file.

Here is what we should do to the header file:

```
class CustomLookAndFeel : public juce::LookAndFeel_V4
{
public:
    CustomLookAndFeel();
    ~CustomLookAndFeel();

    juce::Slider::SliderLayout getSliderLayout (juce::Slider& slider) override;

    void drawRotarySlider (juce::Graphics&, int x, int y, int width, int height,
                           float sliderPosProportional, float rotaryStartAngle,
                           float rotaryEndAngle, juce::Slider&) override;

    juce::Label* createSliderTextBox (juce::Slider& slider) override;

    juce::Font getTextButtonFont (juce::TextButton&, int buttonHeight) override;

    void drawButtonBackground (juce::Graphics& g, juce::Button& button,
                               const juce::Colour& backgroundColour,
                               bool shouldDrawButtonAsHighlighted,
                               bool shouldDrawButtonAsDown) override;

private:
    juce::Colour blue      = juce::Colour::fromFloatRGBA (0.43f, 0.83f, 1.0f,
 1.0f);
    juce::Colour offWhite  = juce::Colour::fromFloatRGBA (0.83f, 0.84f, 0.9f,
 1.0f);
    juce::Colour grey      = juce::Colour::fromFloatRGBA (0.42f, 0.42f, 0.42f,
1.0f);
    juce::Colour blackGrey = juce::Colour::fromFloatRGBA (0.2f,  0.2f,  0.2f,
 1.0f);

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (CustomLookAndFeel);
};
```

More detailed explanation of each function is down below:

getSliderLayout

function "getSliderLayout()" is used to set up the look of sliders. In another word, it define slider's size, where it should be located, and where the corresponding text brackets should be located.

In this project, RotarySlider's text bracket should be defined in the middle of the knob.

```
juce::Slider::SliderLayout CustomLookAndFeel::getSliderLayout (juce::Slider&
slider)
{
 auto localBounds = slider.getLocalBounds();
```

```
 juce::Slider::SliderLayout layout;
 layout.textBoxBounds = localBounds;
 layout.sliderBounds = localBounds;
 return layout;
}
drawRotarySlider()
RotarySlider::RotarySlider()
{
    setSliderStyle (juce::Slider::SliderStyle::RotaryVerticalDrag);
    setTextBoxStyle (juce::Slider::TextBoxBelow, true, 0, 0);
    setLookAndFeel (&customLookAndFeel);
    setColour (juce::Slider::rotarySliderFillColourId, blue);
    setColour (juce::Slider::textBoxTextColourId, blackGrey);
    setColour (juce::Slider::textBoxOutlineColourId, grey);
    setVelocityBasedMode (true);
    setVelocityModeParameters (0.5, 1, 0.09, false);
    setRange (0.0, 100.0, 0.01);
    setRotaryParameters (juce::MathConstants<float>::pi *
1.25f,juce::MathConstants<float>::pi * 2.75f,
 true);
    setWantsKeyboardFocus (true);
    setTextValueSuffix (" %");
    onValueChange = [&]()
    {
        if (getValue() < 10)
            setNumDecimalPlacesToDisplay (2);
        else if (getValue() >= 10 && getValue() < 100)
            setNumDecimalPlacesToDisplay (1);
        else
            setNumDecimalPlacesToDisplay (0);
    };
}
```

Paint()

The function is used for when we place our mouse on the rotary knob, the function will be activate. shown down below:

```
void RotarySlider::paint (juce::Graphics& g)
{
    juce::Slider::paint (g);
     if (hasKeyboardFocus (false))
    {
        auto length = getHeight() > 15 ? 5.0f : 4.0f;
        auto thick = getHeight() > 15 ? 3.0f : 2.5f;

        g.setColour (findColour (juce::Slider::textBoxOutlineColourId));
        // fromX fromY toX toY
        g.drawLine (0, 0, 0, length, thick);
        g.drawLine (0, 0, length, 0, thick);
        g.drawLine (0, getHeight(), 0, getHeight() length, thick);
        g.drawLine (0, getHeight(), length, getHeight(), thick);
        g.drawLine (getWidth(), getHeight(), getWidth() - length, getHeight(),
thick);
```

```
        g.drawLine (getWidth(), getHeight(), getWidth(), getHeight() length,
thick);
        g.drawLine (getWidth(), 0, getWidth() - length, 0, thick);
        g.drawLine (getWidth(), 0, getWidth(), length, thick);
    }
}
```

mouseDown()/mouseUp()

The function is used to make the mouse disappear when dragging the rotary knob, and appear when it is released:

```
void RotarySlider::mouseDown (const juce::MouseEvent& event)
{
    juce::Slider::mouseDown (event);
    setMouseCursor (juce::MouseCursor::NoCursor);
}
void RotarySlider::mouseUp (const juce::MouseEvent& event)
{
    juce::Slider::mouseUp (event);
    juce::Desktop::getInstance().getMainMouseSource().
setScreenPosition(event.source.getLastMouseDownPosition());
    setMouseCursor (juce::MouseCursor::NormalCursor);
}
```

NameLabel

The class is used to set up tags for each rotary knob. If we apply the class, we will repeat the same codes many time, which is not easy to read, so we cab create a new header file if projucer like the codes down below:

```
#pragma once
#include <JuceHeader.h>
class NameLabel : public juce::Label
{
public:
    NameLabel()
    {
        setFont (20.f);
        setColour (juce::Label::textColourId, grey);
        setJustificationType (juce::Justification::centred);
    }
    ~NameLabel(){}
private:
    juce::Colour grey = juce::Colour::fromFloatRGBA (0.42f, 0.42f, 0.42f, 1.0f);
};
```

PluginEditor

Now we have finished the basic elements of UI design, we can start the PluginEditor section.

```
#pragma once
```

```
#include <JuceHeader.h>
#include "PluginProcessor.h"
#include "CustomLookAndFeel.h"
#include "RotarySlider.h"
#include "NameLabel.h"

//==============================================================================
/**
*/
class SimpleReverbAudioProcessorEditor  : public juce::AudioProcessorEditor
{
public:
    SimpleReverbAudioProcessorEditor (SimpleReverbAudioProcessor&);
    ~SimpleReverbAudioProcessorEditor() override;


 //==============================================================================
    void paint (juce::Graphics&) override;
    void resized() override;

private:
    SimpleReverbAudioProcessor& audioProcessor;

    NameLabel sizeLabel,
              dampLabel,
              widthLabel,
              dwLabel;

    RotarySlider sizeSlider,
                 dampSlider,
                 widthSlider,
                 dwSlider;

    juce::TextButton freezeButton;

    juce::AudioProcessorValueTreeState::SliderAttachment sizeSliderAttachment,
                                                         dampSliderAttachment,
                                                         widthSliderAttachment,
                                                         dwSliderAttachment;

    juce::AudioProcessorValueTreeState::ButtonAttachment freezeAttachment;

    CustomLookAndFeel customLookAndFeel;

    juce::Colour blue      = juce::Colour::fromFloatRGBA (0.43f, 0.83f, 1.0f,
1.0f);
    juce::Colour offWhite  = juce::Colour::fromFloatRGBA (0.83f, 0.84f, 0.9f,
1.0f);
    juce::Colour grey      = juce::Colour::fromFloatRGBA (0.42f, 0.42f, 0.42f,
1.0f);
    juce::Colour black     = juce::Colour::fromFloatRGBA (0.08f, 0.08f, 0.08f,
1.0f);
```

```
    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR
(SimpleReverbAudioProcessorEditor)
};
```

SimpleReverbAudioProcessorEditor()

Here's code for the constructors:

```
SimpleReverbAudioProcessorEditor::SimpleReverbAudioProcessorEditor
(SimpleReverbAudioProcessor& p)
    : AudioProcessorEditor (&p), audioProcessor (p),
      sizeSliderAttachment (audioProcessor.apvts, "Room Size", sizeSlider),
      dampSliderAttachment (audioProcessor.apvts, "Damping", dampSlider),
      widthSliderAttachment (audioProcessor.apvts, "Width", widthSlider),
      dwSliderAttachment (audioProcessor.apvts, "Dry/Wet", dwSlider),
      freezeAttachment (audioProcessor.apvts, "Freeze", freezeButton)
{
    juce::LookAndFeel::getDefaultLookAndFeel().setDefaultSansSerifTypefaceName
("Avenir Next Medium");

    setSize (500, 250);
    setWantsKeyboardFocus (true);

    sizeLabel.setText ("Size", juce::NotificationType::dontSendNotification);
    sizeLabel.attachToComponent (&sizeSlider, false);

    dampLabel.setText ("Damp", juce::NotificationType::dontSendNotification);
    dampLabel.attachToComponent (&dampSlider, false);

    widthLabel.setText ("Width", juce::NotificationType::dontSendNotification);
    widthLabel.attachToComponent (&widthSlider, false);

    dwLabel.setText ("Dry/Wet", juce::NotificationType::dontSendNotification);
    dwLabel.attachToComponent (&dwSlider, false);

    addAndMakeVisible (sizeSlider);
    addAndMakeVisible (dampSlider);
    addAndMakeVisible (widthSlider);
    addAndMakeVisible (dwSlider);
}
```

paint()

To fill background with color of gray, and show "Reverb":

```
void SimpleReverbAudioProcessorEditor::paint (juce::Graphics& g)
{
    g.fillAll (black);

    g.setFont (30);
    g.setColour (offWhite);
    g.drawText ("Simple Reverb", 150, 0, 200, 75, juce::Justification::centred);
}
```

resized()

To set up location and size of location:

```
void SimpleReverbAudioProcessorEditor::resized()
{
    sizeSlider.setBounds (30, 120, 60, 60);
    dampSlider.setBounds (125, 120, 60, 60);
    widthSlider.setBounds (315, 120, 60, 60);
    dwSlider.setBounds (410, 120, 60, 60);
    freezeButton.setBounds (210, 120, 80, 40);
}
```

Try to run the program!

Conclusion

In this tutorial, I introduce how to use JUCE DSP module to design a simple reverb plugin. I recommend people to use the module because it can save you time to create a plugin. Welcome for any comments and improvements, and thank you for reading!

User guide

Introduction or rotary bottoms

Size:

Size is used as a scalar for some/all of the delay lengths that make up the digital reverb network. A smaller Size value will reduce the length of the delays in the reverb algorithm, while larger Size values increase the length of the delays.

Damping:

damping is the absorption of high frequencies in the reverb. Low damping values yield less high-frequency absorption, whereas high damping values produce more absorption of high frequencies. Lower the damping to allow high frequencies to decay for longer to create a brighter reverb sound, or raise the damping to choke the high frequencies and make a darker sound.

Width:

Width is how your sounds move from one side of the mix (or speakers) to the other. Width is achievable in many ways, but one of the most important keys for width is panning. Increasing this value applies more variation between left and right channels, creating a more "spacious" effect. When set at zero, the effect is applied independently to left and right channels.

Wet/Dry level:

Dry simply means without any effects. Wet means with effects. Reverb is usually the effect/plugin that is used to control the front to back placement in a mix. The dryer the track (less reverb) the more forward it will be in the mix.

Reference for User manual:

https://valhalladsp.com/2018/07/09/reverb-101-size/#:~:text=How%20does%20Size%20work%3F,the%20length%20of%20the%20delays.

https://www.izotope.com/en/learn/reflecting-on-reverb-what-it-is-and-how-to-use-it.html

https://manual.audacityteam.org/man/reverb.html

https://www.musicgateway.com/blog/how-to/reverb#:~:text=Dry%20vs%20Wet,will%20be%20in%20the%20mix.