

Mar 10, 2024 5:04	stdin	Page 1/31
<pre> #include "DJAudioPlayer.h"  DJAudioPlayer::DJAudioPlayer(AudioFormatManager&amp; formatManager)     : AudioAppComponent(),       formatManager(formatManager),       resampleSource(&amp;transportSource, false, 2) {     startTimer(500); }  DJAudioPlayer::~DJAudioPlayer() {     stopTimer(); }  void DJAudioPlayer::prepareToPlay(int samplesPerBlockExpected, double sampleRate) {     transportSource.prepareToPlay(samplesPerBlockExpected, sampleRate);     resampleSource.prepareToPlay(samplesPerBlockExpected, sampleRate); }  void DJAudioPlayer::getNextAudioBlock(const AudioSourceChannelInfo&amp; bufferToFill) {     resampleSource.getNextAudioBlock(bufferToFill); }  void DJAudioPlayer::releaseResources() {     transportSource.releaseResources();     resampleSource.releaseResources(); }  void DJAudioPlayer::loadURL(const URL&amp; audioURL) {     auto* reader = formatManager.createReaderFor(audioURL.createInputStream(false));     if (reader != nullptr) {         auto newSource = std::make_unique&lt;AudioFormatReaderSource&gt;(reader, true);         transportSource.setSource(newSource.get(), 0, nullptr, reader-&gt;sampleRate);         readerSource.reset(newSource.release());         justLoaded = true;     } else {         DBG("DJAudioPlayer::loadURL: Bad audio file!");     } }  void DJAudioPlayer::setGain(double gain) {     gain = std::clamp(gain / 100, 0.0, 1.0);     transportSource.setGain(gain); }  void DJAudioPlayer::setSpeed(double ratio) {     ratio = std::clamp(ratio, 0.0, MAX_SPEED_RATIO);     resampleSource.setResamplingRatio(ratio); }  void DJAudioPlayer::setPosition(double posInSecs) {     transportSource.setPosition(posInSecs); }  void DJAudioPlayer::setPositionRelative(double pos) {     pos = std::clamp(pos, 0.0, 1.0);     double posInSecs = transportSource.getLengthInSeconds() * pos;     setPosition(posInSecs); </pre>		

Mar 10, 2024 5:04	stdin	Page
<pre> }  void DJAudioPlayer::setLoop(double seconds) {     if (seconds &gt; 0 &amp;&amp; seconds &lt;= MAX_LOOP_SECONDS) {         loopIsActive = true;         loopSeconds = seconds;         loopEnd = transportSource.getCurrentPosition();         loopStart = loopEnd - seconds;     } else {         loopIsActive = false;     } }  void DJAudioPlayer::timerCallback() {     if (loopIsActive &amp;&amp; (transportSource.getCurrentPosition() &gt;= loopEnd    loopSeconds == 0)) {         transportSource.setPosition(loopStart);     } }  void DJAudioPlayer::start() {     transportSource.start();     justLoaded = false; }  void DJAudioPlayer::pause() {     transportSource.stop(); }  void DJAudioPlayer::stop() {     transportSource.stop();     transportSource.setPosition(0); }  double DJAudioPlayer::getPosInTrack() const {     return transportSource.getCurrentPosition(); }  double DJAudioPlayer::getPositionRelative() const {     auto lengthInSeconds = transportSource.getLengthInSeconds();     return (lengthInSeconds &gt; 0) ? (transportSource.getCurrentPosition() / lengthInSeconds) : 0.0; }  bool DJAudioPlayer::fileJustLoaded() {     bool wasJustLoaded = justLoaded;     justLoaded = false;     return wasJustLoaded; }  #pragma once  #include &lt;JuceHeader.h&gt;  class DJAudioPlayer : public AudioAppComponent, public Timer { public:     explicit DJAudioPlayer(AudioFormatManager&amp; formatManager);     virtual ~DJAudioPlayer();      void prepareToPlay(int samplesPerBlockExpected, double sampleRate) override;     void getNextAudioBlock(const AudioSourceChannelInfo&amp; bufferToFill) override;     void releaseResources() override; </pre>		

Mar 10, 2024 5:04

stdin

Page 3/31

```

void loadURL(const URL& audioURL);
void setGain(double gain);
void setSpeed(double ratio);
void setPosition(double posInSecs);
void setPositionRelative(double pos);
void setLoop(double seconds);
void timerCallback() override;
void start();
void pause();
void stop();

double getPosInTrack() const;
double getPositionRelative() const;
bool fileJustLoaded();

private:
    AudioFormatManager& formatManager;
    std::unique_ptr<AudioFormatReaderSource> readerSource;
    AudioTransportSource transportSource;
    ResamplingAudioSource resampleSource;

    double loopStart{0.0};
    double loopEnd{0.0};
    double loopSeconds{0.0};
    bool justLoaded{false};
    bool loopIsActive{false};

    static constexpr double MAX_SPEED_RATIO = 100.0;
    static constexpr double MAX_LOOP_SECONDS = 16.0;

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(DJAudioPlayer)
};
#include <JuceHeader.h>
#include "DJPanel.h"
#include "Initialise.h"

//=====
DJPanel::DJPanel(DJAudioPlayer* _player,
    TooltipWindow* _tooltipWindow)
    : player(_player),
      tooltipWindow(_tooltipWindow),
      hcPos1(-1.0),
      hcPos2(-1.0),
      hcPos3(-1.0),
      hcPos4(-1.0),
      experienceLevel(0)
{
    Initialise::sliderOptions(this, &volSlider, this, Slider::LinearVertical, Slider::TextBoxBelow, false, 50, 10, 0.0, 100.0, 1.0, &v1, volSlider.textBoxOutlineColourId, Colours::transparentWhite);
    Initialise::sliderOptions(this, &speedSlider, this, Slider::LinearVertical, Slider::TextBoxBelow, false, 50, 10, 0.01, 2.0, 0.1, &v1, speedSlider.textBoxOutlineColourId, Colours::transparentWhite);
    Initialise::sliderOptions(this, &loopSlider, this, Slider::IncDecButtons, Slider::TextBoxAbove, false, 50, 20, 0.0, 16.0, 1.0, &v1, loopSlider.textBoxOutlineColourId, Colours::transparentWhite);

    Initialise::labelOptions(this, &volume, "VOL", dontSendNotification, Justification::centred, 14.0f, label.textColourId, Colour::fromRGBA(11, 24, 98, 255));
    Initialise::labelOptions(this, &speed, "SPEED", dontSendNotification, Justification::centred, 14.0f, label.textColourId, Colour::fromRGBA(79, 0, 16, 255));

```

Sunday March 10, 2024

Mar 10, 2024 5:04

stdin

Page

```

    Initialise::labelOptions(this, &loop, "Loop", dontSendNotification, Justification::centred, 16.0f, label.textColourId, Colour::fromRGBA(11, 24, 98, 255));
    Initialise::labelOptions(this, &hotCue, "HotCue", dontSendNotification, Justification::centred, 16.0f, label.textColourId, Colour::fromRGBA(79, 0, 16, 255));

    volume.attachToComponent(&volSlider, false);
    speed.attachToComponent(&speedSlider, false);
    loop.attachToComponent(&loopSlider, false);
    hotCue.attachToComponent(&hcBtn1, false);

    addAndMakeVisible(hcBtn1);
    addAndMakeVisible(hcBtn2);
    addAndMakeVisible(hcBtn3);
    addAndMakeVisible(hcBtn4);

    volSlider.setValue(50);
    speedSlider.setValue(1.0);

    volSlider.setTextValueSuffix("%");
    speedSlider.setTextValueSuffix(" x");
    loopSlider.setTextValueSuffix(" s");

    hcBtn1.addListener(this);
    hcBtn2.addListener(this);
    hcBtn3.addListener(this);
    hcBtn4.addListener(this);

    if (experienceLevel <= 2)
    {
        loopSlider.setTooltip("Select how many seconds from the \ncurrent position backwards to play repeatedly.");
        hcBtn1.setTooltip("Click to save the current position for easy calling\n\n CTRL + click to cancel previously saved position.");
    }
}

DJPanel::~DJPanel()
{
}

void DJPanel::paint(Graphics& g)
{
    double rowH = getHeight() / 4;
    double rowW = getWidth() / 4;

    // Fill background.
    g.fillAll(Colour::fromRGBA(255, 183, 197, 255));

    g.setColour(Colours::white);
    g.setFont(16.0f);

    if (player->fileJustLoaded())
    {
        // Reset all controls.
        hcPos1 = -1.0;
        hcBtn1.setToggleState(false, NotificationType::dontSendNotification);
        hcPos2 = -1.0;
        hcBtn2.setToggleState(false, NotificationType::dontSendNotification);
        hcPos3 = -1.0;
        hcBtn3.setToggleState(false, NotificationType::dontSendNotification);

```

stdin

Mar 10, 2024 5:04	stdin	Page 5/31
	<pre>         hcPos4 = -1.0;         hcBtn4.setToggleState(false, NotificationType::dontSendNotification);          volSlider.setValue(50);         speedSlider.setValue(1.0);         loopSlider.setValue(0);          experienceLevel++;          if (experienceLevel &gt;= 3)         {             // Remove tooltips.             loopSlider.setTooltip("");             hcBtn1.setTooltip("");              // Reset loop tooltip to appear after 1 hour             tooltipWindow-&gt;setMillisecondsBeforeTipAppears(3666666);         }     }      repaint(); }  void DJPanel::resized() {     const double rowH = getHeight() / 5;     const double colW = getWidth() / 8;     const double colH = getHeight() / 9;     const double sliderW = getWidth() / 3;     const double sliderH = getHeight() / 10;     const double buttonW = sliderW * 1.2;     const double buttonH = sliderH * 1.2;      const Colour buttonOnColour1 = Colour::fromRGB(102, 157, 246);     const Colour buttonOnColour2 = Colour::fromRGB(1, 30, 254);      // Volume Slider     volSlider.setNumDecimalPlacesToDisplay(0);     volSlider.setBounds(colW, getHeight() / 15, sliderW, rowH * 1.8 );      // Speed Slider     speedSlider.setNumDecimalPlacesToDisplay(1);     speedSlider.setBounds(colW * 4.5, getHeight() / 15, sliderW, rowH * 1.8);      // Loop Slider     loopSlider.setBounds(colW * 4.7, rowH * 3, buttonW, sliderH * 2.8);      // Hot Cue Buttons     setButtonProperties(hcBtn1, colW, colH * 4.615, buttonW, buttonH, buttonOnCo lour1);     setButtonProperties(hcBtn2, colW, colH * 5.7, buttonW, buttonH, buttonOnColo ur2);     setButtonProperties(hcBtn3, colW, colH * 6.8, buttonW, buttonH, buttonOnColo ur1);     setButtonProperties(hcBtn4, colW, colH * 7.9, buttonW, buttonH, buttonOnColo ur2); }  void DJPanel::setButtonProperties(TextButton&amp; button, double x, double y, double width, double height, Colour colour) {     button.setBounds(x, y, width, height); </pre>	

Mar 10, 2024 5:04	stdin	Page
	<pre>         button.setColour(TextButton::ColourIds::buttonOnColourId, colour);         button.setColour(TextButton::ColourIds::textColourOnId, Colours::black);         button.setCursor(MouseCursor::PointingHandCursor);     }      void DJPanel::sliderValueChanged(Slider* slider)     {         if (slider == &amp;volSlider)         {             player-&gt;setGain(slider-&gt;getValue());         }          if (slider == &amp;speedSlider)         {             player-&gt;setSpeed(slider-&gt;getValue());         }          if (slider == &amp;loopSlider)         {             player-&gt;setLoop(slider-&gt;getValue());         }     }      void DJPanel::handleButtonClick(Button* button, double&amp; hcPos)     {         if (!isCommandDown())         {             if (hcPos == -1.0)             {                 // Set cue                 hcPos = player-&gt;getPosInTrack();                 button-&gt;setToggleState(true, NotificationType::dontSendNotificat ion);             }             else             {                 // Recall cue                 player-&gt;setPosition(hcPos);             }         }         else         {             // Reset when user press CTRL &amp; click the button             hcPos = -1.0;             button-&gt;setToggleState(false, NotificationType::dontSendNotificatio n);         }     }      void DJPanel::buttonClicked(Button* button)     {         if (button == &amp;hcBtn1)         {             handleButtonClick(button, hcPos1);         }         else if (button == &amp;hcBtn2)         {             handleButtonClick(button, hcPos2);         }         else if (button == &amp;hcBtn3)         {             handleButtonClick(button, hcPos3);         }     } </pre>	

Mar 10, 2024 5:04

stdin

Page 7/31

```

        else if (button == &hcBtn4)
        {
            handleClick(button, hcPos4);
        }
    }

bool DJPanel::isCommandDown() const noexcept
{
    // Check if CTRL key is pressed
    if (currentModifiers == ModifierKeys::Flags::commandModifier)
    {
        return true;
    }

    return false;
}#pragma once

#include <JuceHeader.h>
#include "DJAudioplayer.h"
#include "Visuals.h"

//=====
/*
*/
class DJPanel : public Component,
    public Slider::Listener,
    public Button::Listener,
    public ModifierKeys
{
public:
    DJPanel(DJAudioplayer* _player, TooltipWindow* _tooltipWindow);
    ~DJPanel() override;

    void paint(Graphics& g) override;
    void resized() override;

    void setButtonProperties(TextButton& button, double x, double y, double width, double height, Colour colour);
    void sliderValueChanged(Slider* slider) override;
    void buttonClicked(Button* button) override;

    void handleClick(Button* button, double& hcPos);

    // Check if CTRL key is pressed
    bool isCommandDown() const noexcept;

private:
    Label volume, speed, loop, hotCue;

    Visuals v1;

    Label label;

    // Sliders
    Slider volSlider;
    Slider speedSlider;
    Slider loopSlider;

    //Hot Cue Buttons
    TextButton hcBtn1{ "1" };
    TextButton hcBtn2{ "2" };

```

Sunday March 10, 2024

Mar 10, 2024 5:04

stdin

Page

```

    TextButton hcBtn3{ "3" };
    TextButton hcBtn4{ "4" };

    double hcPos1;
    double hcPos2;
    double hcPos3;
    double hcPos4;
    int experienceLevel;

    DJAudioPlayer* player;
    TooltipWindow* tooltipWindow;

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(DJPanel)
};#include <JuceHeader.h>
#include "DeckGUI.h"
#include "Initialise.h"

DeckGUI::DeckGUI(DJAudioplayer* _player,
    AudioFormatManager& formatManagerToUse,
    AudioThumbnailCache& cacheToUse,
    Colour& colourToUse,
    TooltipWindow* _tooltipWindow
) : player(_player),
    formatManager(formatManagerToUse),
    waveformDisplay(formatManagerToUse, cacheToUse, colourToUse),
    accentColour(colourToUse),
    tooltipWindow(_tooltipWindow),
    isLoading(false),
    userExp(0)
{
    disc = ImageCache::getFromMemory(BinaryData::dj_9_png, BinaryData::dj_9_png_size);

    loadImageButtonImage(playButton, BinaryData::play_png, BinaryData::play_png_size);
    loadImageButtonImage(pauseButton, BinaryData::pause_png, BinaryData::pause_png_size);
    loadImageButtonImage(stopButton, BinaryData::stop_png, BinaryData::stop_png_size);
    loadImageButtonImage(loadButton, BinaryData::load_png, BinaryData::load_png_size);

    Initialise::buttonOptions(this, &playButton, this, true, "", 0.8f);
    Initialise::buttonOptions(this, &pauseButton, this, true, "", 0.8f);
    Initialise::buttonOptions(this, &stopButton, this, true, "", 0.8f);
    Initialise::buttonOptions(this, &loadButton, this, true, "", 0.8f);

    addAndMakeVisible(posSlider);
    addAndMakeVisible(waveformDisplay);

    posSlider.addListener(this);
    posSlider.setRange(0.0, 1.0);

    startTimer(500);
}

DeckGUI::~DeckGUI()
{
    stopTimer();
}

```

stdin

Mar 10, 2024 5:04

stdin

Page 9/31

```

}

void DeckGUI::loadImageButtonImage(ImageButton& button, const void* imageData, s
ize_t imageSize) {
    auto image = ImageCache::getFromMemory(imageData, imageSize);
    button.setImages(true, true, true, image, 1, Colours::transparentWhite, Imag
e(nullptr), 1, Colours::transparentWhite, Image(nullptr), 1, Colours::transparen
tBlack);
}

void DeckGUI::paint(Graphics& g)
{
    // Background colour
    g.fillAll(Colour::fromRGBA(255, 183, 197, 255));

    // Set position, transform the disc
    g.setOrigin(getWidth() / 2, getHeight() / 1.8);

    // Set origin of disc rotation
    AffineTransform transform(AffineTransform::translation((float) (disc.getWidth
() / -2),
        (float) (disc.getHeight() / -2)));

    // Draw disc rotation on file load & play
    transform = transform.followedBy(getTransform());

    // Draw disc img transformation
    g.drawImageTransformed(disc, transform, false);

    repaint();
}

void DeckGUI::resized()
{
    double rowH = (double) (getHeight() / 8);
    double rowW = (double) (getWidth() / 8);
    double buttonW = (double) (getWidth() / 10);
    double layoutH = (double) (getHeight() / 3);

    posBtn(playButton, buttonW - 8);
    posBtn(pauseButton, buttonW * 3);
    posBtn(stopButton, buttonW * 5 + 8);
    posBtn(loadButton, buttonW * 7 + 16);

    posSlider.setBounds(rowW, rowH * 2, rowW * 6, rowH * 5);
    posSlider.setSliderStyle(Slider::SliderStyle::RotaryHorizontalVerticalDrag);
    posSlider.setCursor(MouseCursor::DraggingHandCursor);
    posSlider.setTextBoxStyle(Slider::TextEntryBoxPosition::NoTextBox, true, 0,
0);
    posSlider.setColour(Slider::ColourIds::thumbColourId, accentColour);
    posSlider.setColour(Slider::ColourIds::rotarySliderFillColourId, Colour::fro
mRGBA(255, 134, 123, 255));
    posSlider.setRotaryParameters(MathConstants<float>::pi,
        MathConstants<float>::twoPi + MathConstants<float>::pi,
        true);

    waveformDisplay.setBounds(0, 0, getWidth(), layoutH - rowH / 2);
}

void DeckGUI::buttonClicked(Button* button)

```

Sunday March 10, 2024

Mar 10, 2024 5:04

stdin

Page 1

```

{
    if (button == &playButton)
    {
        player->start();

        if (isLoading)
        {
            playButton.setToggleState(true, NotificationType::dontSendNotifi
cation);
            pauseButton.setToggleState(false, NotificationType::dontSendNotifi
cation);
            stopButton.setToggleState(false, NotificationType::dontSendNotifi
cation);
            loadButton.setToggleState(false, NotificationType::dontSendNotifi
cation);
        }

        if (button == &pauseButton)
        {
            player->pause();

            if (isLoading)
            {
                playButton.setToggleState(false, NotificationType::dontSendNotifi
cation);
                pauseButton.setToggleState(true, NotificationType::dontSendNotifi
cation);
                stopButton.setToggleState(false, NotificationType::dontSendNotifi
cation);
                loadButton.setToggleState(false, NotificationType::dontSendNotifi
cation);
            }
        }

        if (button == &stopButton)
        {
            player->stop();

            if (isLoading)
            {
                playButton.setToggleState(false, NotificationType::dontSendNotifi
cation);
                pauseButton.setToggleState(false, NotificationType::dontSendNotifi
cation);
                stopButton.setToggleState(true, NotificationType::dontSendNotifi
cation);
                loadButton.setToggleState(false, NotificationType::dontSendNotifi
cation);
            }
        }

        if (button == &loadButton)
        {
            if (isLoading)
            {
                playButton.setToggleState(false, NotificationType::dontSendNotifi
cation);
                pauseButton.setToggleState(false, NotificationType::dontSendNotifi
cation);
                stopButton.setToggleState(false, NotificationType::dontSendNotifi
cation);
            }
        }
    }
}

```

stdin

Mar 10, 2024 5:04

stdin

Page 11/31

```

        loadButton.setToggleState(true, NotificationType::dontSendNotificati
on);
    }

    FileChooser chooser{ "Select a music file..." };

    if (chooser.browseForFileToOpen())
    {
        File trackFile = chooser.getResult();
        String trackName = getSongTitle(trackFile);
        String trackDuration = getSongDuration(trackFile);
        URL trackPath = URL{ trackFile };

        player->loadURL(trackPath);
        waveformDisplay.loadFile(trackName, trackDuration, trackPath);

        isLoading = true;
        playButton.setToggleState(false, NotificationType::dontSendNotificat
ion);
        pauseButton.setToggleState(false, NotificationType::dontSendNotifica
tion);
        stopButton.setToggleState(false, NotificationType::dontSendNotificat
ion);
        loadButton.setToggleState(false, NotificationType::dontSendNotificat
ion);

        userExp++;
    }
}

void DeckGUI::sliderValueChanged(Slider* slider)
{
    if (slider == &posSlider)
    {
        double sliderValue = slider->getValue();

        if (sliderValue >= 1.0)
        {
            sliderValue = 0.0;
            player->setPositionRelative(sliderValue);
            player->start();
        }
        else
        {
            player->setPositionRelative(sliderValue);
        }
    }
}

bool DeckGUI::isInterestedInFileDrag(const StringArray& files)
{
    return true;
}

void DeckGUI::filesDropped(const StringArray& files, int x, int y)
{
    if (files.size() == 1)
    {
        String trackName = getSongTitle(File{ files[0] });
        String trackDuration = getSongDuration(File{ files[0] });
        URL trackPath = URL{ File{files[0]} };

```

Sunday March 10, 2024

Mar 10, 2024 5:04

stdin

Page 1

```

        player->loadURL(trackPath);
        waveformDisplay.loadFile(trackName, trackDuration, trackPath);

        isLoading = true;
        playButton.setToggleState(false, NotificationType::dontSendNotificat
;
        pauseButton.setToggleState(false, NotificationType::dontSendNotifica
);
        stopButton.setToggleState(false, NotificationType::dontSendNotificat
;
        loadButton.setToggleState(false, NotificationType::dontSendNotificat
;

        userExp++;
    }
}

void DeckGUI::timerCallback()
{
    double relativePosition = player->getPositionRelative();

    if (!isnan(relativePosition))
    {
        waveformDisplay.setPositionRelative(relativePosition);
        waveformDisplay.updateTrackDuration(player->getPosInTrack());
        posSlider.setValue(relativePosition);
    }

    if (userExp <= 2)
    {
        posSlider.setTooltip("Click and drag disc to the right/left or \nunp
hange the current position.");
    }
    else
    {
        posSlider.setTooltip("");
    }
}

void DeckGUI::loadTrack(String trackName, String trackDuration, URL trackPat
{
    player->loadURL(trackPath);
    waveformDisplay.loadFile(trackName, trackDuration, trackPath);

    isLoading = true;
    playButton.setToggleState(false, NotificationType::dontSendNotification);
    pauseButton.setToggleState(false, NotificationType::dontSendNotification);
    stopButton.setToggleState(false, NotificationType::dontSendNotification);
    loadButton.setToggleState(false, NotificationType::dontSendNotification);

    userExp++;
}

String DeckGUI::getSongTitle(File songFile)
{
    String songTitle{ songFile.getFileNameWithoutExtension() };
    return songTitle;
}

String DeckGUI::getSongDuration(File songFile)
{
    int numSamples{ 0 };

```

stdin

Mar 10, 2024 5:04

stdin

Page 13/31

```

double sampleRate{ 0.0 };
double lengthInSecs{ 0.0 };
auto* reader = formatManager.createReaderFor(URL{ songFile }.createInputStream(false));

if (reader != nullptr) // If file is valid
{
    numSamples = reader->lengthInSamples;
    sampleRate = reader->sampleRate;
    lengthInSecs = numSamples / sampleRate;
}

// Prevent memory leak
delete reader;

int minutes = floor(lengthInSecs / 60.0);
int seconds = floor(lengthInSecs - (minutes * 60));

String songLength;

if (seconds >= 0 && seconds < 10)
{
    songLength = String{ minutes } + ":0" + String{ seconds };
}
else
{
    songLength = String{ minutes } + ":" + String{ seconds };
}

return songLength;
}

void DeckGUI::posBtn(ImageButton& button, double x)
{
    double rowH = (double)(getHeight() / 8);
    double buttonW = (double)(getWidth() / 10);

    button.setBounds(x, rowH * 7.1 - 2, buttonW * 2, rowH / 1.2);
    button.setCursor(MouseCursor::PointingHandCursor);
}

AffineTransform DeckGUI::getTransform()
{
    AffineTransform t;
    t = t.rotated(fmod(0.15 * player->getPosInTrack(), 2.0) * MathConstants<float>::twoPi);

    return t;
}

#pragma once

#include <JuceHeader.h>
#include "DJAudioplayer.h"
#include "WaveformDisplay.h"
#include "Initialise.h"

//=====
/*
    This class represents a GUI component for a deck in a DJ application.
    It includes controls for loading, playing, pausing, and stopping tracks,
    as well as a waveform display and a position slider.
*/

```

Sunday March 10, 2024

Mar 10, 2024 5:04

stdin

Page 1

```

class DeckGUI : public Component,
    public Button::Listener,
    public Slider::Listener,
    public FileDragAndDropTarget,
    public Timer
{
public:
    DeckGUI(DJAudioplayer* _player,
        AudioFormatManager& formatManagerToUse,
        AudioThumbnailCache& cacheToUse,
        Colour& colourToUse,
        TooltipWindow* _tooltipWindow);
    ~DeckGUI();

    void loadImageButtonImage(ImageButton& button, const void* imageData, size_t imageSize);
    void paint(Graphics& g) override;
    void resized() override;

    void buttonClicked(Button* button) override;
    void sliderValueChanged(Slider* slider) override;

    bool isInterestedInFileDrag(const StringArray& files) override;
    void filesDropped(const StringArray& files, int x, int y) override;
    void timerCallback() override;
    void loadTrack(String trackName, String trackDuration, URL trackPath);
    static String getSongTitle(File songFile);
    String getSongDuration(File songFile);

private:

    void posBtn(ImageButton& button, double x);
    AffineTransform getTransform();
    ImageButton playButton;
    ImageButton pauseButton;
    ImageButton stopButton;
    ImageButton loadButton;
    Slider posSlider;
    Image disc;
    Colour accentColour;

    bool isLoading;
    int userExp;

    AudioFormatManager& formatManager;
    WaveformDisplay waveformDisplay;
    DJAudioplayer* player;
    TooltipWindow* tooltipWindow;

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(DeckGUI)
};

#include <JuceHeader.h>
#include "Initialise.h"

Initialise::Initialise()
{
}

void Initialise::sliderOptions(Component* component, Slider* slider, Slider::Listener* listener,
    Slider::SliderStyle style, Slider::TextEntry*)

```

stdin

Mar 10, 2024 5:04	stdin	Page 15/31
osition textPos,	bool readOnly, int textBoxW, int textBoxH, doubl	
e rangeStart,	double rangeEnd, double increment, LookAndFeel*	
Visuals,	Slider::ColourIds colourid, Colour colour)	
{	component->addAndMakeVisible(slider);	
	slider->addListener(listener);	
	slider->setSliderStyle(style);	
	slider->setTextBoxStyle(textPos, readOnly, textBoxW, textBoxH);	
	slider->setColour(slider->textBoxOutlineColourId, Colours::transparentWhite)	
;	slider->setRange(rangeStart, rangeEnd, increment);	
	slider->setLookAndFeel(Visuals);	
	slider->setColour(colourid, colour);	
}		
void Initialise::labelOptions(Component* component, Label* label, String title,	NotificationType notiType, Justification justific	
ation,	float fontSize, Label::ColourIds colourid, Colour	
colour)		
{	component->addAndMakeVisible(label);	
	label->setText(title, notiType);	
	label->setJustificationType(justification);	
	label->setFont(fontSize);	
	label->setColour(colourid, colour);	
}		
void Initialise::buttonOptions(Component* component, Button* button,	Button::Listener* listener, bool toggleOn,	
	String tooltip, float alpha)	
{	component->addAndMakeVisible(button);	
	button->addListener(listener);	
	button->setClickingTogglesState(toggleOn);	
	button->setTooltip(tooltip);	
	button->setAlpha(alpha);	
}		
#pragma once		
#include <JuceHeader.h>		
#include "Visuals.h"		
class Initialise : public Component,		
	public Slider::Listener,	
	public Button::Listener,	
	public ComboBox::Listener	
{		
public:	Initialise();	
	static void sliderOptions(Component* component, Slider* slider, Slider::List	
ener* listener,	Slider::SliderStyle style, Slider::TextEntryBoxPos	
ition textPos,	bool readOnly, int textBoxW, int textBoxH, double	
rangeStart,	double rangeEnd, double increment, LookAndFeel* Vi	
suals,		

Mar 10, 2024 5:04	stdin	Page 1
	Slider::ColourIds colourid, Colour colour);	
	static void labelOptions(Component* component, Label* label, String tit	
	NotificationType notiType, Justification justifi	
ion,	float fontSize, Label::ColourIds colourid, Col	
colour);		
	static void buttonOptions(Component* component, Button* button,	
	Button::Listener* listener, bool toggleOn,	
	String tooltip, float alpha);	
};		
#include "../JuceLibraryCode/JuceHeader.h"		
#include "MainComponent.h"		
//=====		
class OtoDecksApplication : public JUCEApplication		
{		
public:	//=====	
====		
OtoDecksApplication() {}		
const String getApplicationName() override	{ return ProjectInfo::p	
ctName; }		
const String getApplicationVersion() override	{ return ProjectInfo::p	
onString; }		
bool moreThanOneInstanceAllowed() override	{ return true; }	
	//=====	
====		
void initialise (const String& commandLine) override		
{	// This method is where you should put your application's initialis	
code..		
	mainWindow.reset (new MainWindow (getApplicationName()));	
}		
void shutdown() override		
{	// Add your application's shutdown code here..	
	mainWindow = nullptr; // (deletes our window)	
}		
	//=====	
====		
void systemRequestedQuit() override		
{	// This is called when the app is being asked to quit: you can ignor	
is	// request and let the app carry on running, or call quit() to allow	
app to close.	quit();	
	}	
	void anotherInstanceStarted (const String& commandLine) override	
{	// When another instance of the app is launched while this one is ru	
g,	// this method is invoked, and the commandLine parameter tells you w	
	// the other instance's command-line arguments were.	



Mar 10, 2024 5:04

stdin

Page 17/31

```

    }

    //=====
    /*
       This class implements the desktop window that contains an instance of
       our MainComponent class.
    */
    class MainWindow      : public DocumentWindow
    {
    public:
        MainWindow (String name)      : DocumentWindow (name,
                                                    Desktop::getInstance().getDe
faultLookAndFeel(),
                                                    .findC
olour (ResizableWindow::backgroundColourId),
                                                    DocumentWindow::allButtons)
        {
            setUsingNativeTitleBar (true);
            setContentOwned (new MainComponent(), true);

            #if JUCE_IOS || JUCE_ANDROID
                setFullScreen (true);
            #else
                setResizable (true, true);
                centreWithSize (getWidth(), getHeight());
            #endif

            setVisible (true);
        }

        void closeButtonPressed() override
        {
            // This is called when the user tries to close this window. Here, we
            // ask the app to quit when this happens, but you can change this to
            // whatever you need.
            JUCEApplication::getInstance()->systemRequestedQuit();
        }

        /* Note: Be careful if you override any DocumentWindow methods - the bas
e
        class uses a lot of them, so by overriding you might break its functi
onality.
        It's best to do all your work in your content component instead, but
        if
        you really have to override any DocumentWindow methods, make sure you
        r
        subclass also calls the superclass's method.
        */

    private:
        JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (MainWindow)
    };

private:
    std::unique_ptr<MainWindow> mainWindow;
};

//=====
// This macro generates the main() routine that launches the app.

```

Sunday March 10, 2024

Mar 10, 2024 5:04

stdin

Page 1

```

START_JUCE_APPLICATION (OtoDecksApplication)
#include "MainComponent.h"

//=====
MainComponent::MainComponent()
{
    // Make sure you set the size of the component after
    // you add any child components.
    setSize (800, 600);

    if (RuntimePermissions::isRequired (RuntimePermissions::recordAudio)
        && ! RuntimePermissions::isGranted (RuntimePermissions::recordAudio))
    {
        RuntimePermissions::request (RuntimePermissions::recordAudio,
                                      [&] (bool granted) { if (granted) setA
Channels (2, 2); });
    }
    else
    {
        // Specify number of input and output channels to be opened
        setAudioChannels (2, 2);
    }

    addAndMakeVisible(deckGUI1);
    addAndMakeVisible(deckGUI2);
    addAndMakeVisible(DJPanel1);
    addAndMakeVisible(DJPanel2);
    addAndMakeVisible(stereoReverbGUI);
    // Format the audio files
    formatManager.registerBasicFormats();
}

MainComponent::~MainComponent()
{
    // Shuts down audio device & clears the audio source
    shutdownAudio();
}

//=====
void MainComponent::prepareToPlay (int samplesPerBlockExpected, double samp
e)
{
    mixerSource.addInputSource(&player1, false);
    mixerSource.addInputSource(&player2, false);

    stereoReverb.prepareToPlay(samplesPerBlockExpected, sampleRate);
}

void MainComponent::getNextAudioBlock (const AudioSourceChannelInfo& buffer
1)
{
    stereoReverb.getNextAudioBlock(bufferToFill);
}

void MainComponent::releaseResources()
{
    // This will be called when the audio device stops, or when it is being
    // restarted due to a setting change.

    // For more details, see the help for AudioProcessor::releaseResources()
    player1.releaseResources();
    player2.releaseResources();
}

```

stdin

Mar 10, 2024 5:04

stdin

Page 19/31

```

    mixerSource.releaseResources();
    stereoReverb.releaseResources();
}

//=====
void MainComponent::paint (Graphics& g)
{
    // (Our component is opaque, so we must completely fill the background with
    a solid colour)
    g.fillAll (getLookAndFeel().findColour (ResizableWindow::backgroundColourId)
);
    // You can add your drawing code here!
}

void MainComponent::resized()
{
    // This is called when the MainContentComponent is resized.
    // If you add any child components, this is where you should
    // update their positions.
    deckGUI1.setBounds(0, getHeight() * 0.35, getWidth() * 0.35, getHeight() * 0
.65);
    deckGUI2.setBounds(getWidth() * 0.65, getHeight() * 0.35, getWidth() * 0.35,
getHeight() * 0.65);
    DJPanel1.setBounds(getWidth() * 0.35, getHeight() * 0.35, getWidth() * 0.15,
getHeight() * 0.65);
    DJPanel2.setBounds(getWidth() * 0.50, getHeight() * 0.35, getWidth() * 0.15,
getHeight() * 0.65);
    stereoReverbGUI.setBounds(0, 0, getWidth() * 1, getHeight() * 0.35);
}

#pragma once

#include "../JuceLibraryCode/JuceHeader.h"
#include "DJAudioplayer.h"
#include "DeckGUI.h"
#include "DJPanel.h"
#include "StereoGUI.h"

class MainComponent    : public AudioAppComponent
{
public:
    MainComponent();
    ~MainComponent();

    void prepareToPlay (int samplesPerBlockExpected, double sampleRate) override
;
    void getNextAudioBlock (const AudioSourceChannelInfo& bufferToFill) override
;
    void releaseResources() override;

    void paint (Graphics& g) override;
    void resized() override;

private:
    AudioFormatManager formatManager;
    AudioThumbnailCache thumbCache{ 100 };

    // Left side of the GUI
    DJAudioPlayer player1{ formatManager };

```

Sunday March 10, 2024

Mar 10, 2024 5:04

stdin

Page 2

```

    Colour blueDeckColour{ Colour::fromRGBA(1, 30, 254, 255) };
    DeckGUI deckGUI1{ &player1, formatManager, thumbCache, blueDeckColour, f
};
    DJPanel DJPanel1{ &player1, &tip };

    // Right side of the GUI
    DJAudioPlayer player2{ formatManager };
    Colour purpleDeckColour{ Colour::fromRGBA(201, 0, 255, 255) };
    DeckGUI deckGUI2{ &player2, formatManager, thumbCache, purpleDeckColour, f
p };
    DJPanel DJPanel2{ &player2, &tip};

    // Take in 2 audio sources
    MixerAudioSource mixerSource;

    TooltipWindow tip{ this, 700 };

    // Stereo Reverb GUI
    StereoGUI stereoReverbGUI{&stereoReverb};
    // Stereo Reverb
    Stereo stereoReverb{&mixerSource};

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (MainComponent)
};
#include <JuceHeader.h>
#include "Stereo.h"

//=====
Stereo::Stereo(MixerAudioSource* mixerSource) : reverbSource(mixerSource, f
{
    // Set bypassed
    reverbSource.setBypassed(true);
    // Set default parameters to 0s
    defaultParameters.roomSize = 0;
    defaultParameters.damping = 0;
    defaultParameters.wetLevel = 0;
    defaultParameters.dryLevel = 0;
    defaultParameters.width = 0;
    defaultParameters.freezeMode = 0;
    reverbSource.setParameters(defaultParameters);
}

Stereo::~Stereo()
{
}

void Stereo::prepareToPlay(int samplesPerBlockExpected, double sampleRate)
{
    reverbSource.prepareToPlay(samplesPerBlockExpected, sampleRate);
}

void Stereo::releaseResources()
{
    reverbSource.releaseResources();
}

void Stereo::getNextAudioBlock(const AudioSourceChannelInfo& bufferToFill)
{
    // Set parameters during callback
    reverbSource.setParameters(parameters);
    reverbSource.getNextAudioBlock(bufferToFill);
}

```

stdin

Mar 10, 2024 5:04

stdin

Page 21/31

```

}

void Stereo::setBypass()
{
    reverbSource.setBypassed(false);
}

#pragma once

#include <JuceHeader.h>

//=====
/*
*/
class Stereo : public AudioSource
{
public:
    Stereo(MixerAudioSource* mixerSource);
    ~Stereo();

    void prepareToPlay(int samplesPerBlockExpected, double sampleRate) override;
    void releaseResources() override;
    void getNextAudioBlock(const AudioSourceChannelInfo& bufferToFill) override;
    void setBypass();

    // Parameters for reverb audio filters
    Reverb::Parameters parameters;
    Reverb::Parameters defaultParameters;
private:
    // Reverb audio source apply -> audio source
    // &mixerSource -> MainComponent
    ReverbAudioSource reverbSource;
};
#include <JuceHeader.h>
#include "StereoGUI.h"
#include "Initialise.h"

//=====
StereoGUI::StereoGUI(Stereo* _stereoReverb) : stereoReverb(_stereoReverb)
{
    // Sliders
    Initialise::sliderOptions(this, &roomSizeSlider, this, Slider::LinearVertical,
1, Slider::TextBoxBelow, false, 50, 10, 0.0, 1.0, 0.01, &v1, roomSizeSlider.textBoxOutlineColourId, Colours::transparentWhite);
    Initialise::sliderOptions(this, &dampingSlider, this, Slider::LinearVertical,
1, Slider::TextBoxBelow, false, 50, 10, 0.0, 1.0, 0.01, &v1, dampingSlider.textBoxOutlineColourId, Colours::transparentWhite);
    Initialise::sliderOptions(this, &wetLevelSlider, this, Slider::LinearVertical,
1, Slider::TextBoxBelow, false, 50, 10, 0.0, 1.0, 0.01, &v1, wetLevelSlider.textBoxOutlineColourId, Colours::transparentWhite);
    Initialise::sliderOptions(this, &dryLevelSlider, this, Slider::LinearVertical,
1, Slider::TextBoxBelow, false, 50, 10, 0.0, 1.0, 0.01, &v1, dryLevelSlider.textBoxOutlineColourId, Colours::transparentWhite);
    Initialise::sliderOptions(this, &widthSlider, this, Slider::LinearVertical,
1, Slider::TextBoxBelow, false, 50, 10, 0.0, 1.0, 0.01, &v1, widthSlider.textBoxOutlineColourId, Colours::transparentWhite);
    Initialise::sliderOptions(this, &freezeModeSlider, this, Slider::LinearVertical,
1, Slider::TextBoxBelow, false, 50, 10, 0.0, 1.0, 0.01, &v1, freezeModeSlider.textBoxOutlineColourId, Colours::transparentWhite);

```

Sunday March 10, 2024

Mar 10, 2024 5:04

stdin

Page 2

```

// Labels
    Initialise::labelOptions(this, &label, "Stereo Reverb", dontSendNotification, Justification::centred, 14.0f, label.textColourId, Colour::fromRGBA(11, 24, 98, 255));
    Initialise::labelOptions(this, &roomSize, "RS", dontSendNotification, Justification::centred, 14.0f, label.textColourId, Colour::fromRGBA(79, 0, 16, 255));
    Initialise::labelOptions(this, &damping, "DMP", dontSendNotification, Justification::centred, 14.0f, label.textColourId, Colour::fromRGBA(11, 24, 98, 255));
    Initialise::labelOptions(this, &wetLevel, "WL", dontSendNotification, Justification::centred, 14.0f, label.textColourId, Colour::fromRGBA(79, 0, 16, 255));
    Initialise::labelOptions(this, &dryLevel, "DL", dontSendNotification, Justification::centred, 14.0f, label.textColourId, Colour::fromRGBA(11, 24, 98, 255));
    Initialise::labelOptions(this, &width, "W", dontSendNotification, Justification::centred, 14.0f, label.textColourId, Colour::fromRGBA(79, 0, 16, 255));
    Initialise::labelOptions(this, &freezeMode, "FM", dontSendNotification, Justification::centred, 14.0f, label.textColourId, Colour::fromRGBA(11, 24, 98, 255));
};

// Slider labels
    roomSize.attachToComponent(&roomSizeSlider, false);
    damping.attachToComponent(&dampingSlider, false);
    wetLevel.attachToComponent(&wetLevelSlider, false);
    dryLevel.attachToComponent(&dryLevelSlider, false);
    width.attachToComponent(&widthSlider, false);
    freezeMode.attachToComponent(&freezeModeSlider, false);

// Info button
    auto infoImage = ImageCache::getFromMemory(BinaryData::info_png, BinaryData::info_pngSize);
    infoBtn.setImages(true, true, true, infoImage, 1, Colours::mediumpurple, 1, Colour::fromRGBA(11, 24, 98, 255), Image(nullptr), 1, Colour::transparentBlack);
    addAndMakeVisible(infoBtn);
    infoBtn.setTooltip("R: Room Size, DMP: Damping, WL: Wet Level, DL: Dry Level, W: Width, FM: Freeze Mode");
}

StereoGUI::~StereoGUI()
{
    // Remove sliders lookandfeel
    roomSizeSlider.setLookAndFeel(nullptr);
    dampingSlider.setLookAndFeel(nullptr);
    wetLevelSlider.setLookAndFeel(nullptr);
    dryLevelSlider.setLookAndFeel(nullptr);
    widthSlider.setLookAndFeel(nullptr);
    freezeModeSlider.setLookAndFeel(nullptr);
}

void StereoGUI::paint(Graphics& g)
{
    // Fill background
    g.fillAll(Colour::fromRGBA(255, 183, 197, 255));
    // Outline component
    g.setColour(Colour::fromRGBA(255, 183, 197, 255));
    g.drawRect(getLocalBounds(), 1);
    // Outline on header
    g.setColour(Colours::mediumpurple);
    g.drawRect(label.getBounds());

    // Placeholder text
    g.setColour(Colours::white);
    g.setFont(14.0f);

```

stdin

Mar 10, 2024 5:04

stdin

Page 23/31

```

    g.drawText("", getLocalBounds(), Justification::centred, true);
}

void StereoGUI::resized()
{
    double w = getWidth() / 6;
    double h = getHeight() * 0.2;
    double h2 = getHeight() * 0.3;

    // Sliders
    roomSizeSlider.setBounds(0, h, w, h * 3.25);
    dampingSlider.setBounds(w, h2, w, h * 3.25);
    wetLevelSlider.setBounds(w * 2, h, w, h * 3.25);
    dryLevelSlider.setBounds(w * 3, h2, w, h * 3.25);
    widthSlider.setBounds(w * 4, h, w, h * 3.25);
    freezeModeSlider.setBounds(w * 5, h2, w, h * 3.25);

    // Label
    label.setBounds(0, 0, getWidth(), getHeight() * 0.11);

    // Info button
    infoBtn.setBounds(w * 5.5, getHeight() * 0.02, w * 0.5, getHeight() * 0.06);
}

// Value of parameters of sliders
void StereoGUI::sliderValueChanged(Slider* slider)
{
    if (slider == &roomSizeSlider)
    {
        stereoReverb->parameters.roomSize = slider->getValue();
        stereoReverb->setBypass();
    }

    if (slider == &dampingSlider)
    {
        stereoReverb->parameters.damping = slider->getValue();
        stereoReverb->setBypass();
    }

    if (slider == &wetLevelSlider)
    {
        stereoReverb->parameters.wetLevel = slider->getValue();
        stereoReverb->setBypass();
    }

    if (slider == &dryLevelSlider)
    {
        stereoReverb->parameters.dryLevel = slider->getValue();
        stereoReverb->setBypass();
    }

    if (slider == &widthSlider)
    {
        stereoReverb->parameters.width = slider->getValue();
        stereoReverb->setBypass();
    }

    if (slider == &freezeModeSlider)
    {
        stereoReverb->parameters.freezeMode = slider->getValue();
        stereoReverb->setBypass();
    }
}

```

Sunday March 10, 2024

Mar 10, 2024 5:04

stdin

Page 2

```

};#pragma once

#include <JuceHeader.h>
#include "Stereo.h"
#include "Visuals.h"

//=====
/*
*/
class StereoGUI : public Component,
                  public Slider::Listener
{
public:
    StereoGUI(Stereo* _stereoReverb);
    ~StereoGUI() override;

    void paint(Graphics&) override;
    void resized() override;

    // Get parameters value from respective sliders -> implement Slider::Listener
    void sliderValueChanged(Slider* slider) override;

private:
    // 6 slider values for 6 reverb parameters
    Slider roomSizeSlider;
    Slider dampingSlider;
    Slider wetLevelSlider;
    Slider dryLevelSlider;
    Slider widthSlider;
    Slider freezeModeSlider;

    // Labels for all the parameters
    Label roomSize, damping, wetLevel, dryLevel, width, freezeMode;

    // Customise slider appearance
    Visuals v1;

    // Label for StereoGUI
    Label label;

    // Info button
    ImageButton infoBtn;

    // Stereo reverb filter
    Stereo* stereoReverb;

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(StereoGUI)
};
#include <JuceHeader.h>
#include "Visuals.h"

// Reference: https://github.com/juce-framework/JUCE/blob/master/examples/GUI/okAndFeelDemo.h

// =====
=
Visuals::Visuals()
{
    // In your constructor, you should add any child components, and
    // initialise any special settings that your component needs.
}

```

stdin

Mar 10, 2024 5:04	stdin	Page 25/31
<pre> }  // Slider // Customise slider thumb appearance void Visuals::drawRoundThumb(Graphics&amp; g, float x, float y, float diameter, Colour colour, float outlineThickness) {     auto halfThickness = outlineThickness * 0.5f;      Path p;     p.addRoundedRectangle(x + halfThickness, y + halfThickness, diameter - outlineThickness, diameter - outlineThickness, 5.0f);      DropShadow(Colours::mediumpurple, 1, {}).drawForPath(g, p);      g.setColour(Colour::fromRGB(76, 104, 215));     g.fillPath(p);     g.strokePath(p, PathStrokeType(outlineThickness)); }  void Visuals::drawLinearSliderThumb(Graphics&amp; g, int x, int y, int width, int height,                                      float sliderPos, float minSliderPos, float maxSliderPos,                                      const Slider::SliderStyle style, Slider&amp; slider) {     auto sliderRadius = (float)(getSliderThumbRadius(slider) - 2);      auto isDownOrDragging = slider.isEnabled() &amp;&amp; (slider.isMouseOverOrDragging()    slider.isMouseButtonDown());      auto knobColour = slider.findColour(Slider::thumbColourId)         .withMultipliedSaturation((slider.hasKeyboardFocus(false)    isDownOrDragging) ? 1.3f : 0.9f)         .withMultipliedAlpha(slider.isEnabled() ? 1.0f : 0.7f);      if (style == Slider::LinearHorizontal    style == Slider::LinearVertical)     {         float kx, ky;          if (style == Slider::LinearVertical)         {             kx = (float)x + (float)width * 0.5f;             ky = sliderPos;         }         else         {             kx = sliderPos;             ky = (float)y + (float)height * 0.5f;         }          auto outlineThickness = slider.isEnabled() ? 0.8f : 0.3f;          drawRoundThumb(g,             kx - sliderRadius,             ky - sliderRadius,             sliderRadius * 2.0f,             knobColour, outlineThickness);     }     else </pre>		

Mar 10, 2024 5:04	stdin	Page 2
<pre> {     // Calling base class for demo     LookAndFeel_V2::drawLinearSliderThumb(g, x, y, width, height, sliderPos, minSliderPos, maxSliderPos, style, slider); }  // Customise slider appearance void Visuals::drawLinearSlider(Graphics&amp; g, int x, int y, int width, int height,                                 float sliderPos, float minSliderPos, float maxSliderPos,                                 const Slider::SliderStyle style, Slider&amp; slider) {     g.fillAll(slider.findColour(Slider::backgroundColourId).withMultipliedBrightness(0.8f));      if (style == Slider::LinearBar    style == Slider::LinearBarVertical)     {         Path p;          if (style == Slider::LinearBarVertical)         {             p.addRectangle((float)x, sliderPos, (float)width, 1.0f + (float)height - sliderPos);         }         else         {             p.addRectangle((float)x, (float)y, sliderPos - (float)x, (float)height);         }         auto baseColour = slider.findColour(Slider::rotarySliderFillColourId)             .withMultipliedSaturation(slider.isEnabled() ? 1.0f : 0.5f)             .withMultipliedAlpha(0.8f);          g.setColour(baseColour);         g.fillPath(p);          auto lineThickness = jmin(15.0f, (float)jmin(width, height) * 0.45f);         g.drawRect(slider.getLocalBounds().toFloat(), lineThickness);     }     else     {         drawLinearSliderBackground(g, x, y, width, height, sliderPos, minSliderPos, maxSliderPos, style, slider);         drawLinearSliderThumb(g, x, y, width, height, sliderPos, minSliderPos, maxSliderPos, style, slider);     } }  // Customise slider background void Visuals::drawLinearSliderBackground(Graphics&amp; g, int x, int y, int width, int height,  float, // sliderPos  float, // minSliderPos  float, // maxSliderPos  const Slider::SliderStyle, // style  Slider&amp; slider) {     auto sliderRadius = (float)getSliderThumbRadius(slider) - 5.0f; </pre>		

Mar 10, 2024 5:04

stdin

Page 27/31

```

    Path on, off;

    if (slider.isHorizontal())
    {
        auto iy = (float)y + (float)height * 0.5f - sliderRadius * 0.5f;
        Rectangle<float> r((float)x - sliderRadius * 0.5f, iy, (float)width + sliderRadius, sliderRadius);
        auto onWidth = r.getWidth() * ((float)slider.valueToProportionOfLength(slider.getValue()));

        on.addRectangle(r.removeFromLeft(onWidth));
        off.addRectangle(r);
    }
    else
    {
        auto ix = (float)x + (float)width * 0.5f - sliderRadius * 0.5f;
        Rectangle<float> r(ix, (float)y - sliderRadius * 0.5f, sliderRadius, (float)height + sliderRadius);
        auto onH = r.getHeight() * ((float)slider.valueToProportionOfLength(slider.getValue()));

        on.addRectangle(r.removeFromBottom(onH));
        off.addRectangle(r);
    }
    // Set slider background colour
    g.fillAll(Colour::fromRGBA(255, 183, 197, 255));
    // Set slider colour -> included portion
    g.setColour(Colour(0xff001dab));
    g.fillPath(on);

    g.setColour(slider.findColour(Slider::trackColourId));
    g.fillPath(off);
}
#pragma once

#include <JuceHeader.h>

//=====
/*
*/
class Visuals : public LookAndFeel_V4
{
public:

    Visuals();

    // Customise slider thumb appearance
    void drawRoundThumb(Graphics& g, float x, float y, float diameter, Colour colour, float outlineThickness);

    void drawLinearSliderThumb(Graphics& g, int x, int y, int width, int height, float sliderPos, float minSliderPos, float maxSliderPos, const Slider::SliderStyle style, Slider& slider) override;

    // Customise slider appearance
    void drawLinearSlider(Graphics& g, int x, int y, int width, int height, float sliderPos, float minSliderPos, float maxSliderPos, const Slider::SliderStyle style, Slider& slider) override;

```

Sunday March 10, 2024

Mar 10, 2024 5:04

stdin

Page 2

```

    // Customise slider background
    void drawLinearSliderBackground(Graphics& g, int x, int y, int width, int height, float, // sliderPos
                                     float, // minSliderPos
                                     float, // maxSliderPos
                                     const Slider::SliderStyle, //style
                                     Slider& slider) override;

};

#include <JuceHeader.h>
#include "WaveformDisplay.h"

//=====
WaveformDisplay::WaveformDisplay(AudioFormatManager& formatManagerToUse, AudioThumbnailCache& cacheToUse, Colour& colourToUse) : audioThumb(1000, formatManagerToUse, cacheToUse), fileLoaded(false), position(0.0), posInSecs(0.0), trackName(""), trackDuration(""), formatManager(formatManagerToUse), accentColour(colourToUse)

{
    audioThumb.addChangeListener(this);
}

WaveformDisplay::~WaveformDisplay()
{
}

void WaveformDisplay::paint(Graphics& g)
{
    // Fill background
    g.fillAll(Colour::fromRGBA(255, 183, 197, 255));

    g.setColour(Colours::orangered);
    if (fileLoaded)
    {
        double rowW = getWidth() / 4.0;

        g.setFont(20.0f);
        g.setColour(Colours::white);

        // Draw song title.
        Rectangle<float> titleArea(10, 5, (float)(rowW * 3) - 5, 20);
        g.drawText(trackName, titleArea, Justification::centredLeft, true);

        // Draw song length.
        Rectangle<float> durationArea((float)(rowW * 2.8), 5, (float)rowW, 20);
        g.drawText(trackDuration, durationArea, Justification::centredRight, true);
    }

    // Draw left channel.
    g.setColour(accentColour);
    Rectangle<int> waveformArea(0, 30, getWidth(), getHeight() - 30);
    audioThumb.drawChannel(g, waveformArea, 0.0,

```

stdin

Mar 10, 2024 5:04

stdin

Page 29/31

```

        audioThumb.getTotalLength(),
        0,
        1.0f);
g.setColour(Colour::fromRGBA(76, 104, 215, 255));

// Draw playhead.
double playheadPos = position * getWidth();
Line<float> arrowLine(Point<float>((float)playheadPos, 30),
    Point<float>((float)playheadPos, 40));
g.drawArrow(arrowLine, 20, 20, 55);
g.drawRect((int)playheadPos, 35, 2, getHeight() - 5);
}
else
{
    g.setFont(20.0f);
    g.setColour(Colour::fromRGBA(102, 157, 246, 255));
    g.drawText("File not loaded...", getLocalBounds(),
        Justification::centred, true); // draw some placeholder text
}
}

void WaveformDisplay::resized()
{
}

void WaveformDisplay::loadFile(String fileName, String fileLength, URL audioURL)
{
    trackName = fileName;
    trackDuration = fileLength;

    audioThumb.clear();
    fileLoaded = audioThumb.setSource(new URLInputSource(audioURL));
    if (fileLoaded)
    {
        repaint();
    }
    else {
        DBG("WaveformDisplay: not loaded!");
    }
}

void WaveformDisplay::changeListenerCallback(ChangeBroadcaster* source)
{
    repaint();
}

void WaveformDisplay::setPositionRelative(double pos)
{
    posInSecs = pos;

    if (pos != position && !isnan(pos))
    {
        position = pos; // goes from 0 to 1;
        repaint();
    }
}

String WaveformDisplay::getSongDuration(double lengthInSecs)
{
    int minutes = (int)floor(lengthInSecs / 60.0);
    int seconds = (int)floor(lengthInSecs - (minutes * 60.0));

```

Sunday March 10, 2024

Mar 10, 2024 5:04

stdin

Page 3

```

    String songLength;

    if (seconds >= 0 && seconds < 10)
    {
        songLength = String{ minutes } + ":0" + String{ seconds };
    }
    else
    {
        songLength = String{ minutes } + ":" + String{ seconds };
    }

    return songLength;
}

void WaveformDisplay::updateTrackDuration(double currentPos)
{
    double remainingDuration;
    double totalDuration = audioThumb.getTotalLength();

    if (currentPos >= totalDuration)
    {
        remainingDuration = 0.0;
    }
    else
    {
        remainingDuration = totalDuration - currentPos;
    }

    trackDuration = getSongDuration(remainingDuration);
}#pragma once

#include <JuceHeader.h>

//=====
/*
*/
class WaveformDisplay : public Component,
    public ChangeListener
{
public:

    WaveformDisplay(AudioFormatManager& formatManagerToUse,
        AudioThumbnailCache& cacheToUse,
        Colour& colourToUse);
    ~WaveformDisplay();

    void paint(Graphics& g) override;
    void resized() override;

    void changeListenerCallback(ChangeBroadcaster* source) override;

    void loadFile(String fileName, String fileLength, URL audioURL);

    // Set position of playhead relative to the track's length and between 0
1    void setPositionRelative(double pos);

    // Calculate remaining duration of the track currently playing
    void updateTrackDuration(double currentPos);

private:

```

stdin

Mar 10, 2024 5:04

stdin

Page 31/31

```
// Calculate track length, inputing length in seconds and converts it to a string
static String getSongDuration(double lengthInSecs);

AudioThumbnail audioThumb;
AudioFormatManager& formatManager;

bool fileLoaded;
double position;
double posInSecs;

String trackName;
String trackDuration;
Colour accentColour;

JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(WaveformDisplay)
};
```