```
/*
  ==============================================================================

    DeckGUI.cpp

  ==============================================================================
*/

#include "../JuceLibraryCode/JuceHeader.h"
#include "DeckGUI.h"
#include <cmath>
#include "PlaylistComponent.h"


DeckGUI::DeckGUI(DJAudioPlayer* _player,
                 PlaylistComponent* _playlistComponent,
                 AudioFormatManager& formatManagerToUse,
                 AudioThumbnailCache& cacheToUse,
                 int channelToUse
                 ) : player(_player),
                     playlistComponent(_playlistComponent),
                     waveformDisplay(formatManagerToUse,cacheToUse),
                     channel(channelToUse)
{
```

```
//BUTTONS AND LISTENERS
addAndMakeVisible(playButton);
addAndMakeVisible(stopButton);
addAndMakeVisible(nextButton);
playButton.addListener(this);
stopButton.addListener(this);
nextButton.addListener(this);

//GUI SLIDERS, LISTENERS, COLORS
addAndMakeVisible(posSlider);
posSlider.addListener(this);
posSlider.setSliderStyle(Slider::SliderStyle::LinearHorizontal);
posSlider.setRange(0.0, 1.0);
posSlider.setTextBoxStyle(Slider::NoTextBox, false, 0, 0);

addAndMakeVisible(volSlider);
volSlider.addListener(this);
volSlider.setRange(0.0, 1.0);
volSlider.setValue(0.5);
volSlider.setSliderStyle(Slider::SliderStyle::LinearBarVertical);
volSlider.setTextBoxStyle(Slider::NoTextBox, false, 0, 0);
addAndMakeVisible(volLabel);
volLabel.setText("Volume", juce::dontSendNotification);
volLabel.attachToComponent(&volSlider, false);
```

```
volLabel.setJustificationType(juce::Justification::centred);


addAndMakeVisible(speedSlider);
speedSlider.addListener(this);
speedSlider.setRange(0.5, 2, 0); // Min half speed, max speed 2x
speedSlider.setValue(1); // Default speed at 1x

// Set the slider style to LinearHorizontal
speedSlider.setSliderStyle(Slider::SliderStyle::LinearHorizontal);

speedSlider.setTextBoxStyle(Slider::NoTextBox, false, 0, 0);
addAndMakeVisible(speedLabel);
speedLabel.setText("Speed", juce::dontSendNotification);
speedLabel.attachToComponent(&speedSlider, false);
speedLabel.setJustificationType(juce::Justification::centred);



//SLIDERS COLORS
getLookAndFeel().setColour(juce::Slider::thumbColourId, juce::Colours::orange);
getLookAndFeel().setColour(juce::Slider::trackColourId, juce::Colours::orange);
getLookAndFeel().setColour(juce::Slider::rotarySliderFillColourId, juce::Colours::orange);



//LIST SONGS
```

```
    upNext.getHeader().addColumn("Next Playing", 1, 100);
    upNext.setModel(this);
    addAndMakeVisible(upNext);

    //WAVEFORM
    addAndMakeVisible(waveformDisplay);



    startTimer(100);
}

DeckGUI::~DeckGUI()
{

    stopTimer();
}

//==============================================================================
void DeckGUI::paint(Graphics & g)
{

}

void DeckGUI::resized()
{
```

```
    double rowH = getHeight() / 6;
    double colW = getWidth() / 4;




    posSlider.setBounds(0, rowH * 2, getWidth(), rowH);

    volSlider.setBounds(0, rowH * 3 +20, colW, rowH*3 -30);
    speedSlider.setBounds(colW, rowH * 3 +20, colW*1.5, rowH*2 - 30);
    upNext.setBounds(colW * 2.5, rowH * 3, colW * 1.5 - 20, rowH * 2);

    playButton.setBounds(colW+10, rowH * 5 + 10, colW-20, rowH-20);
    stopButton.setBounds(colW*2+10, rowH * 5 + 10, colW-20, rowH-20);
    nextButton.setBounds(colW * 3 + 10, rowH * 5 + 10, colW - 20, rowH - 20);
    posSlider.setBounds(0, rowH * 2, getWidth(), rowH);
    waveformDisplay.setBounds(0, 0, getWidth(), rowH * 2);

}

//==============================================================================
void DeckGUI::buttonClicked(Button* button)
{
    if (button == &playButton)
    {
        player->start(); //need to press load (nextbutton) first to work
```

```
}
if (button == &stopButton)
{
   player->stop();
}
if (button == &nextButton)
{
   //Next button in the left side
   if (channel == 0 && playlistComponent->playListL.size() > 0) //handle only if there are songs added
   {
      //FIRST SONGS URL
      URL fileURL = URL{ File{playlistComponent->playListL[0]} };
      //LOAD URL AND WAVEFORM
      player->loadURL(fileURL);
      waveformDisplay.loadURL(fileURL);
      //POP FIRST URL
      playlistComponent->playListL.erase(playlistComponent->playListL.begin());
   }

   //Next button in the right side
   if (channel == 1 && playlistComponent->playListR.size() > 0)
   {
      //FIRST SONGS URL
      URL fileURL = URL{ File{playlistComponent->playListR[0]} };
      //LOAD URL AND WAVEFORM
```

```
            player->loadURL(fileURL);
            waveformDisplay.loadURL(fileURL);
            //POP FIRST URL
            playlistComponent->playListR.erase(playlistComponent->playListR.begin());
        }

        //WAY TO CHANGE 1ST SONG
        if (nextButton.getButtonText() == "LOAD")
        {
            nextButton.setButtonText("NEXT");
        }
        else
        {
            player->start();
        }
    }



    upNext.updateContent();
}

void DeckGUI::sliderValueChanged(Slider* slider)
{
    if (slider == &volSlider)
    {
```

```
        player->setGain(slider->getValue());
    }

    if (slider == &speedSlider)
    {
        player->setSpeed(slider->getValue());
    }
    if (slider == &posSlider)
    {
        player->setRelativePosition(slider->getValue());
    }

}

//==============================================================================
int DeckGUI::getNumRows()
{

    if (channel == 0) //LEFT
    {
        return playlistComponent->playListL.size();
    }
    if (channel == 1) //RIGHT
    {
```

```
        return playlistComponent->playListR.size();
    }
}

void DeckGUI::paintRowBackground(Graphics& g,
    int rowNumber,
    int width,
    int height,
    bool rowIsSelected)
{

    g.fillAll(juce::Colours::transparentBlack);
}

void DeckGUI::paintCell(Graphics& g,
    int rowNumber,
    int columnId,
    int width,
    int height,
    bool rowIsSelected)
{
    std::string filepath = "";

    //FILE PATH DEPENDING IN CHANNEL
    if (channel == 0) //left
```

```
    {
        filepath = playlistComponent->playListL[rowNumber];
    }
    if (channel == 1) //right
    {
        filepath = playlistComponent->playListR[rowNumber];
    }

    // FILE NAME EXTRACTION FROM PATH
    std::size_t startFilePos = filepath.find_last_of("\\");
    std::size_t startExtPos = filepath.find_last_of(".");
    std::string extn = filepath.substr(startExtPos + 1, filepath.length() - startExtPos);
    std::string file = filepath.substr(startFilePos + 1, filepath.length() - startFilePos - extn.size() - 2);
    //CELL NAME DRAW
    g.drawText(file,
        1, rowNumber,
        width - 4, height,
        Justification::centredLeft,
        true);
}



void DeckGUI::timerCallback()
{
    waveformDisplay.setRelativePosition(
```

```
        player->getRelativePosition());
}

/*
  ==============================================================================
    DJAudioPlayer.cpp
  ==============================================================================
*/

#include "DJAudioPlayer.h"

DJAudioPlayer::DJAudioPlayer(AudioFormatManager& _formatManager)
    :formatManager(_formatManager)
{}

DJAudioPlayer::~DJAudioPlayer()
{}

//==============================================================================
void DJAudioPlayer::prepareToPlay(int samplesPerBlockExpected, double sampleRate)
{
    transportSource.prepareToPlay(
        samplesPerBlockExpected,
        sampleRate);
    resampleSource.prepareToPlay(
```

```
        samplesPerBlockExpected,
        sampleRate);
}

void DJAudioPlayer::getNextAudioBlock(const AudioSourceChannelInfo& bufferToFill)
{
    resampleSource.getNextAudioBlock(bufferToFill);
}

void DJAudioPlayer::releaseResources()
{
    transportSource.releaseResources();
    resampleSource.releaseResources();
}


//==============================================================================
void DJAudioPlayer::loadURL(URL audioURL)
{
    auto* reader = formatManager.createReaderFor(audioURL.createInputStream(false));
    if (reader != nullptr)
    {
        std::unique_ptr<AudioFormatReaderSource> newSource(new AudioFormatReaderSource(reader,
            true));
        transportSource.setSource(newSource.get(), 0, nullptr, reader->sampleRate);
```

```
        readerSource.reset(newSource.release());
    }
}

void DJAudioPlayer::setGain(double gain)
{
    if (gain < 0 || gain >1) {}
    else
    {
        transportSource.setGain(gain);
    }
}

void DJAudioPlayer::setSpeed(double ratio)
{
    if (ratio < 0) {}
    else
    {
        resampleSource.setResamplingRatio(ratio);
    }
}

void DJAudioPlayer::setRelativePosition(double pos)
{
    if (pos < 0 || pos >1) {}
```

```
    else {
        double posInSecs = transportSource.getLengthInSeconds() * pos;
        setPosition(posInSecs);
    }
}
void DJAudioPlayer::setPosition(double posInSecs)
{
    transportSource.setPosition(posInSecs);
}

void DJAudioPlayer::start()
{
    transportSource.start();
}

void DJAudioPlayer::stop()
{
    transportSource.stop();
}

double DJAudioPlayer::getRelativePosition()
{
    return transportSource.getCurrentPosition() / transportSource.getLengthInSeconds();
}
```

```
/*
  ==============================================================================

MAIN.cpp

   This file was auto-generated!


   It contains the basic startup code for a JUCE application.


  ==============================================================================
*/


#include "../JuceLibraryCode/JuceHeader.h"
#include "MainComponent.h"


//==============================================================================
class OtoDecksApplication  : public JUCEApplication
{
public:
   //==============================================================================
   OtoDecksApplication() {}
   const String getApplicationName() override      { return ProjectInfo::projectName; }
   const String getApplicationVersion() override    { return ProjectInfo::versionString; }
   bool moreThanOneInstanceAllowed() override      { return true; }


   //==============================================================================
   void initialise (const String& commandLine) override
```

```
{
    // This method is where you should put your application's initialisation 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 ignore this
    // request and let the app carry on running, or call quit() to allow the app to close.
    quit();
}

void anotherInstanceStarted (const String& commandLine) override
{
    // When another instance of the app is launched while this one is running,
    // this method is invoked, and the commandLine parameter tells you what
```

```
    // the other instance's command-line arguments were.
}

//==============================================================================
/*
    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().getDefaultLookAndFeel()
                                            .findColour (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'll just
        // ask the app to quit when this happens, but you can change this to do
        // whatever you need.
        JUCEApplication::getInstance()->systemRequestedQuit();
    }

    /* Note: Be careful if you override any DocumentWindow methods - the base
       class uses a lot of them, so by overriding you might break its functionality.
       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 your
       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.
START_JUCE_APPLICATION (OtoDecksApplication)

/*
  ==============================================================================

    This file was auto-generated!

  ==============================================================================
*/

#include "MainComponent.h"

//==============================================================================
MainComponent::MainComponent()
{
    setSize (800, 600);

    // Some platforms require permissions to open input channels so request that here
    if (RuntimePermissions::isRequired (RuntimePermissions::recordAudio)
        && ! RuntimePermissions::isGranted (RuntimePermissions::recordAudio))
    {
```

```
    RuntimePermissions::request (RuntimePermissions::recordAudio,
                                 [&] (bool granted) { if (granted)  setAudioChannels (2, 2); });
}
else
{
    // Specify the number of input and output channels that we want to open
    setAudioChannels (0, 2);
}

// JUCE FILE FORMATS
formatManager.registerBasicFormats();

// APPLICATION COMPONMENTS
addAndMakeVisible(deckGUILeft);
addAndMakeVisible(deckGUIRight);
addAndMakeVisible(playlistComponent);

// LABEL CUSTOMIZATION
addAndMakeVisible(waveformLabel);
waveformLabel.setText("Output Signal", juce::dontSendNotification);
waveformLabel.setColour(juce::Label::textColourId, juce::Colours::whitesmoke);
waveformLabel.setJustificationType(juce::Justification::centred);
addAndMakeVisible(posLabel);
posLabel.setText("Playback", juce::dontSendNotification);
posLabel.setColour(juce::Label::textColourId, juce::Colours::whitesmoke);
```

```
    posLabel.setJustificationType(juce::Justification::centred);
    addAndMakeVisible(widgetLabel);
    widgetLabel.setText("Control Panel", juce::dontSendNotification);
    widgetLabel.setColour(juce::Label::textColourId, juce::Colours::whitesmoke);
    widgetLabel.setJustificationType(juce::Justification::centred);
    addAndMakeVisible(playlistLabel);
    playlistLabel.setText("Drag Files H E R E", juce::dontSendNotification);
    playlistLabel.setColour(juce::Label::textColourId, juce::Colours::whitesmoke);
    playlistLabel.setJustificationType(juce::Justification::centred);

}

MainComponent::~MainComponent()
{
    // This shuts down the audio device and clears the audio source.
    shutdownAudio();
}



//==============================================================================
void MainComponent::prepareToPlay (int samplesPerBlockExpected, double sampleRate)
{
    playlistComponent.prepareToPlay(samplesPerBlockExpected, sampleRate);

    playerLeft.prepareToPlay(samplesPerBlockExpected, sampleRate);
```

```
    playerRight.prepareToPlay(samplesPerBlockExpected, sampleRate);

    mixerSource.prepareToPlay(samplesPerBlockExpected, sampleRate);

    mixerSource.addInputSource(&playerLeft, false);
    mixerSource.addInputSource(&playerRight, false);

 }

void MainComponent::getNextAudioBlock (const AudioSourceChannelInfo& bufferToFill)
{
    mixerSource.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.

    playlistComponent.releaseResources();

    playerLeft.releaseResources();
    playerRight.releaseResources();
}

//==============================================================================
```

```
void MainComponent::paint (Graphics& g)
{
    g.fillAll (getLookAndFeel().findColour (ResizableWindow::backgroundColourId));
}

void MainComponent::resized()
{
    double rowH = getHeight() / 10;
    double colW = getWidth() / 7;

    //LABEL POSITION
    waveformLabel.setBounds(0, 0, colW, rowH*2);
    posLabel.setBounds(0, rowH*2, colW, rowH);
    widgetLabel.setBounds(0, rowH*3, colW, rowH*3);
    playlistLabel.setBounds(0, rowH*6, colW, rowH*3);

    //FGUI ADD
    deckGUILeft.setBounds(colW, 0, colW * 3, rowH*6);
    deckGUIRight.setBounds(colW * 4, 0, colW * 3, rowH * 6);

    //PLAYLIST ADD
    playlistComponent.setBounds(colW, rowH *6, colW * 6, rowH * 4);

}
```

```
/*
  ==============================================================================

    PlaylistComponent.cpp

  ==============================================================================
*/

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

//==============================================================================
PlaylistComponent::PlaylistComponent(AudioFormatManager& _formatManager)
    : formatManager(_formatManager)
{
    //LIBRARY SETUP
    tableComponent.getHeader().addColumn("Title",1, 250);
    tableComponent.getHeader().addColumn("Song Length", 2, 100);
    tableComponent.getHeader().addColumn("Add to Left", 3, 100);
    tableComponent.getHeader().addColumn("Add to Right", 4, 100);
    tableComponent.setModel(this);
    addAndMakeVisible(tableComponent);

    //SEARCH BAR
```

```
    addAndMakeVisible(searchBar);
    searchBar.addListener(this);
    //SEARCH BAR LABEL

    addAndMakeVisible(searchLabel);
    searchLabel.setText("Please choose track:", juce::dontSendNotification);

}

PlaylistComponent::~PlaylistComponent()
{
}

//==============================================================================
void PlaylistComponent::paint (juce::Graphics& g)
{
}

void PlaylistComponent::resized()
{
    double rowH = getHeight() / 8;
    double colW = getWidth() / 6;

    //SEARCH BAR POSITION
```

```
    searchLabel.setBounds(0, 0, colW, rowH);
    searchBar.setBounds(colW, 0, colW * 5, rowH);
    tableComponent.setBounds(0, rowH, getWidth(), rowH*7);

}



//==============================================================================
int PlaylistComponent::getNumRows()
{
    return interestedTitle.size();
}

void PlaylistComponent::paintRowBackground(Graphics& g,
    int rowNumber,
    int width,
    int height,
    bool rowIsSelected)
{
    if (rowIsSelected)
    {
        g.fillAll(juce::Colours::orange);
    }
    else {
        g.fillAll(juce::Colours::grey);
```

```
}

void PlaylistComponent::paintCell(Graphics& g,
    int rowNumber,
    int columnId,
    int width,
    int height,
    bool rowIsSelected)
{
    //SONG TITLE
    if (columnId == 1)
    {
        g.drawText(interestedTitle[rowNumber],
            1, rowNumber,
            width - 4, height,
            Justification::centredLeft,
            true);
    }
    //SONG DURATION
    if (columnId == 2)
    {
        g.drawText(std::to_string(interestedDuration[rowNumber]) + "s",
            1, rowNumber,
            width - 4, height,
            Justification::centredLeft,
```

```
            true);
    }
}

Component* PlaylistComponent::refreshComponentForCell(int rowNumber,
    int columnId,
    bool isRowSelected,
    Component* existingComponentToUpdate)
{
    //BUTTON FOR CONTROLS LEFT SIDE
    if (columnId == 3)
    {
        if (existingComponentToUpdate == nullptr)
        {
            TextButton* btn = new TextButton{ "Add to L" };
            String id{ std::to_string(rowNumber) };
            btn->setComponentID(id);
            btn->addListener(this);
            existingComponentToUpdate = btn;
            btn->setColour(TextButton::buttonColourId, juce::Colours::darkslategrey);
        }
    }
    //BUTTON FOR CONTROLS RIGHT SIDE
    if (columnId == 4)
    {
```

```
        if (existingComponentToUpdate == nullptr)
        {
            TextButton* btn = new TextButton{ "Add to R" };
            String id{ std::to_string(rowNumber + 1000) };
            btn->setComponentID(id);
            btn->addListener(this);
            existingComponentToUpdate = btn;
            btn->setColour(TextButton::buttonColourId, juce::Colours::darkslategrey);
        }
    }
    return existingComponentToUpdate;
}



//==============================================================================
//FUNCTIONS
void PlaylistComponent::prepareToPlay(int samplesPerBlockExpected, double sampleRate){}
void PlaylistComponent::getNextAudioBlock(const AudioSourceChannelInfo& bufferToFill){}
void PlaylistComponent::releaseResources(){}



//==============================================================================
//BUTTONS
void PlaylistComponent::buttonClicked(Button* button)
{
```

```
    int id = std::stoi(button->getComponentID().toStdString());
    if (id < 1000)
    {
        addToChannelList(interestedFiles[id], 0);
    }

    else
    {
        addToChannelList(interestedFiles[id - 1000], 1);
    }
}


//==============================================================================
bool PlaylistComponent::isInterestedInFileDrag(const StringArray& files)
{
    return true;
}
//FILE MANAGEMENT
void PlaylistComponent::filesDropped(const StringArray& files, int x, int y)
{

    for (String filename : files)
    {
```

```
        std::string filepath = String(filename).toStdString();
        std::size_t startFilePos = filepath.find_last_of("\\");
        std::size_t startExtPos = filepath.find_last_of(".");
        std::string extn = filepath.substr(startExtPos + 1, filepath.length() - startExtPos);
        std::string file = filepath.substr(startFilePos + 1, filepath.length() - startFilePos - extn.size() - 2);


        inputFiles.push_back(filepath);
        trackTitles.push_back(file);


        getAudioLength(URL{ File{filepath} });

    }

    interestedTitle = trackTitles;
    interestedFiles = inputFiles;


    tableComponent.updateContent();
}


//==============================================================================
```

```
void PlaylistComponent::textEditorTextChanged(TextEditor& textEditor)
{

    interestedTitle.clear();
    interestedDuration.clear();
    interestedFiles.clear();

    /
    int pos = 0;
    for (std::string track : trackTitles)
    {

        if (track.find(searchBar.getText().toStdString()) != std::string::npos)
        {
            interestedTitle.push_back(trackTitles[pos]);
            interestedDuration.push_back(trackDurations[pos]);
            interestedFiles.push_back(inputFiles[pos]);
        }
        ++pos;
    }

    tableComponent.updateContent();
}
```

```
//==============================================================================
//ADD SONG FILE
void PlaylistComponent::addToChannelList(std::string filepath, int channel)
{
    if (channel == 0)
    {
        playListL.push_back(filepath);
    }
    if (channel == 1)
    {
        playListR.push_back(filepath);
    }
}

//AUDIO DURATION
void PlaylistComponent::getAudioLength(URL audioURL)
{
    double trackLen = 0.0;

    auto* reader = formatManager.createReaderFor(audioURL.createInputStream(false));

    if (reader != nullptr)
    {
        std::unique_ptr<AudioFormatReaderSource> newSource(new AudioFormatReaderSource(reader,
            true));
```

```
        transportSource.setSource(newSource.get(), 0, nullptr, reader->sampleRate);
        readerSource.reset(newSource.release());
        double trackLen = transportSource.getLengthInSeconds();
    }


    interestedDuration = trackDurations;
}

/*
  ==============================================================================

    WaveformDisplay.cpp

  ==============================================================================
*/

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

//==============================================================================
WaveformDisplay::WaveformDisplay(AudioFormatManager & formatManagerToUse,
                                 AudioThumbnailCache & cacheToUse) :
                                 audioThumb(1000, formatManagerToUse, cacheToUse),
                                 fileLoaded(false),
```

```
                    position(0)
{

    audioThumb.addChangeListener(this);
}

WaveformDisplay::~WaveformDisplay(){}



//==============================================================================
void WaveformDisplay::paint(juce::Graphics& g)
{
    g.fillAll(getLookAndFeel().findColour(juce::ResizableWindow::backgroundColourId));



    g.setColour(juce::Colours::grey);
    g.drawRect(getLocalBounds(), 1);



    if (fileLoaded)
    {
        //WAVEFORM RED
        g.setColour(juce::Colours::crimson);
        audioThumb.drawChannel(g,
            getLocalBounds(),
```

```
            0,
            audioThumb.getTotalLength(),
            0,
            1.0f
        );

        //PLAYHEAD COLOUR
        g.setColour(juce::Colours::orange);
        g.fillRect(position * getWidth(), 0, 2, getHeight());

        //SONG NAME DISPLAY
        g.setColour(juce::Colours::floralwhite);
        g.setFont(16.0f);
        g.drawText(nowPlaying, getLocalBounds(),
            juce::Justification::centred, true);
    }
    else
    {

        g.setColour(juce::Colours::orange);
        g.setFont(20.0f);
        g.drawText("CLICK UPLOAD TO SEE WAVEFORM", getLocalBounds(),
            juce::Justification::centred, true);
    }
}
```

```
void WaveformDisplay::resized()
{

}

//==============================================================================
void WaveformDisplay::loadURL(URL audioURL)
{
   audioThumb.clear();
   fileLoaded = audioThumb.setSource(new URLInputSource(audioURL));
   if (fileLoaded)
   {
      std::string justFile = audioURL.toString(false).toStdString();
      std::size_t startFilePos = justFile.find_last_of("/");
      std::size_t startExtPos = justFile.find_last_of(".");
      std::string extn = justFile.substr(startExtPos + 1, justFile.length() - startExtPos);
      std::string file = justFile.substr(startFilePos + 1, justFile.length() - startFilePos - extn.size() - 2);

      nowPlaying = file;
      repaint();
   }
   else
   {

   }
```

```
}

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

void WaveformDisplay::setRelativePosition(double pos)
{
    if (pos != position)
    {
        position = pos;
        repaint();
    }
}
```