Coding workshop: completing the waveform display class

Introduction

In this worksheet, we will complete the WaveformDisplay functionality by enabling it to load and display a file, and to trigger this when the user loads a file into one of the decks.

Add a loadURL function

In WaveformDisplay.h, add this public function member:

```
public:
...
    void loadURL(URL audioURL);
...
```

Add this private data member to WaveformDisplay:

```
private:
...
    bool fileLoaded;
...
```

In WaveformDisplay.cpp, put in this implementation:

```
void WaveformDisplay::loadURL(URL audioURL)
{
  audioThumbnail.clear();
  fileLoaded = audioThumbnail.setSource(new URLInputSource(audioURL));
}
```

Also, initialise the fileLoaded variable to false in the WaveformDisplay constructor:

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

Notice that we send in a pointer to the setSource function. How do we know it is a pointer? It says new. That means it is a dynamically allocated object.

If you see the new keyword, you should typically start worrying about memory leaks. Where is the call to delete to clear up the pointer? Consulting the JUCE documentation for AudioThumbnail.setSource, you will see this statement: 'The source that is passed in will be deleted by this object when it is no longer needed'.

setSource returns true if the file is loaded, false if something goes wrong. We can use that to add two drawing modes to the WaveformDisplay component.

Update the paint function, so it uses the thumbnail to paint

We are now going to actually ask the audioThumb to draw the waveform. In Waveform.cpp's paint function:

```
if (fileLoaded)
{
    g.setColour(Colours::lightgreen);
    audioThumbnail.drawChannel(g, getLocalBounds(), 0, audioThumbnail.getTotalLength(), 0, 1

}
else
{
    // put the original paint code here
}
```

Now hook it up to the load event on the DeckGUI

In DeckGUI::buttonClicked in DeckGUI.cpp tell the WaveformDisplay when we load a file. Noting that the DJAudioPlayer variable will either be called player or djAudioPlaer. Use your IDE code completion to figure it out:

```
if (button == &loadButton)
{
    auto fileChooserFlags =
    FileBrowserComponent::canSelectFiles;
    fChooser.launchAsync(fileChooserFlags, [this](const FileChooser& chooser)
    {
        djAudioPlayer->loadURL(URL{chooser.getResult()});
        // and now the waveformDisplay as well
        waveformDisplay.loadURL(URL{chooser.getResult()});
    });
}
```

Now compile and try to load a file. Does your waveform display show a waveform? No? There is a reason. If you read the blurb in the documentation for the AudioThumbnailCache, you might remember that it said 'The cache runs a single background thread that is shared by all the thumbnails that need it'. The idea is that the audio waveforms are drawn in a background thread, not on the main thread so the app can keep running while the waveforms are rendered in the background. This adds complexity to the code as we have to redraw the waveform once it has been generated. You can force a call to paint by resizing the window after loading a file. Try it - you should see the waveform appear when you resize the window as paint is called at this point. A bit like this:

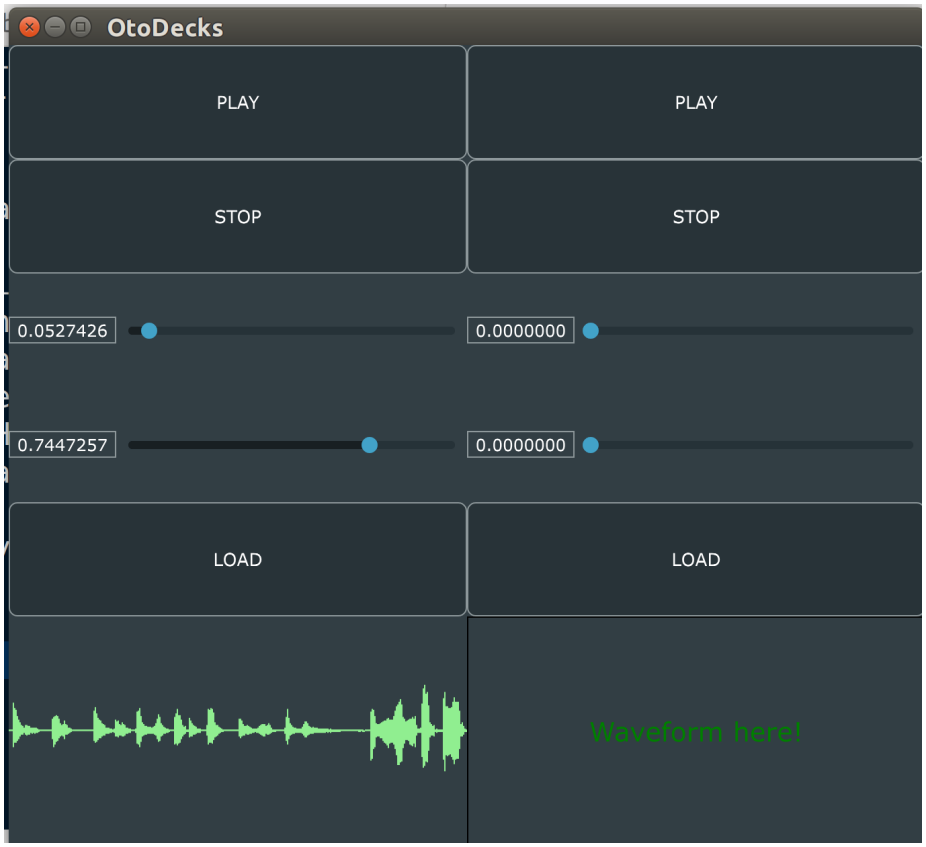

Figure 1: Waveform display after resizing window

Implement the ChangeListener interface

The solution to the 'no automatic drawing of waveform' problem is to implement another listener interface, the ChangeListener. If you check the JUCE documentation for the AudioThumbnail class, you will find that it inherits from the ChangeBroadcaster class. ChangeBroadcasters hold a list of ChangeListeners and send them messages when changes in the ChangeBroadcaster take place. One change that the AudioThumbnail will broadcast is when its display is updated, e.g. because of a file loading. Let's go ahead and implement the ChangeListener in our WaveformDisplay class then register to receive changes from the AudioThumbnail.

In WaveformDisplay.h, inherit from ChangeListener:

```
class WaveformDisplay    : public Component,
                           public ChangeListener
```

Then add the pure virtual function from ChangeListener to the public section:

```
void changeListenerCallback (ChangeBroadcaster *source) override;
```

In WaveformDisplay.cpp, define the listener callback:

```
void WaveformDisplay::changeListenerCallback (ChangeBroadcaster *source)
{
    std::cout << "wfd: change received! " << std::endl;
    repaint();
}
```

Finally, register to receive changes in WaveformDisplay::WaveformDisplay:

```
audioThumbnail.addChangeListener(this);
```

Now compile and try to load a file. Does the waveform display?

Here is a screenshot showing two files loaded into two players:

Also, try loading an invalid file (like a doc or other non-audio file). It should correctly deal with it.

Finally! We should have a working waveform plotter.

Final classes

Since we have made edits to multiple classes, it would be a bit messy to provide them all here. I recommend that you check out the starter code for Topic 10 for reference.

Epilogue: tidy up DJAudioPlayer

After doing all that work to pass the AudioFormatManager around, we are still independently creating AudioFormatManager objects in the DJAudioPlayer class.

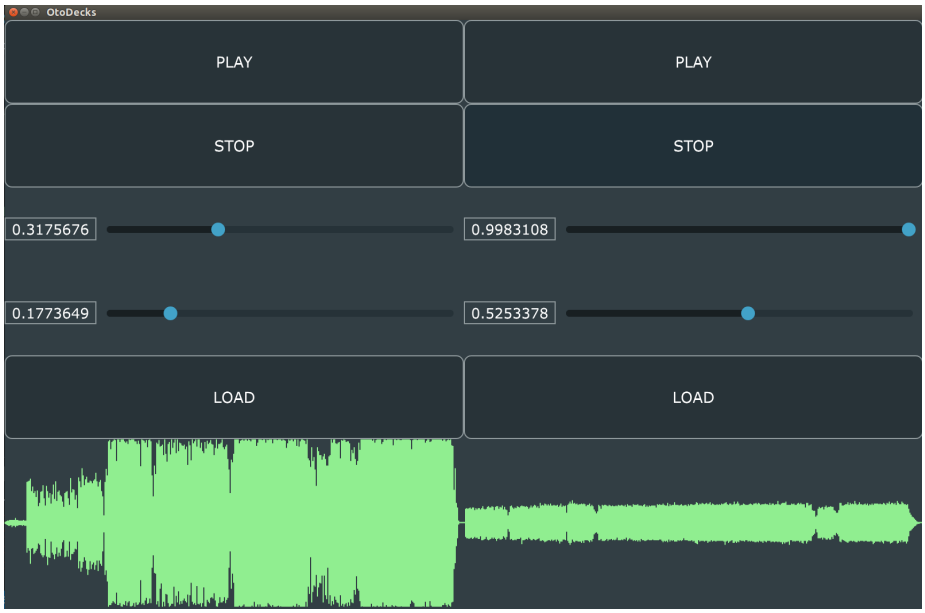

Figure 2: Two waveform displays

Let's fix that by passing the AudioFormatManager from the MainComponent into the DJAudioPlayer.

In DJAudioPlayer.h, update the constructor:

```
DJAudioPlayer(AudioFormatManager& formatManager);
```

and in DJAudioPlayer.h, change the type of the formatManager to a reference member type:

```
AudioFormatManager& formatManager;
```

This means it does not store its own copy, it stores a reference to the one you sent in. If you miss the ampersand on the variable declaration, even though the constructor receives a reference, it will be made into a copy when you assign.

In DJAudioPlayer.cpp, update the constructor:

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

}
```

You might also find that your DJAudioPlayer::prepareToPlay function calls formatManager.registerBasicFormats(). You can remove that line.

Now in MainComponent.h, send the formatManager to the updated DJAudioPlayer constructors:

```
DJAudioPlayer player1{formatManager};
DJAudioPlayer player2{formatManager};
```

That's it! Compile and test. It should behave the same, but now you only have one application-wide AudioFormatManager.

Conclusion

We have completed the implementation of a waveform display using the AudioThumbnail class.