Coding workshop: Implement a table component

In this worksheet, we will develop a basic table component. This Component can form the basis of a playlist component which allows the user to manage a persistent list of audio files.

The TableListBox

JUCE provides a ready-made component that can be used to display tables of information. It is called TableListBox.

TableListBox is somewhat complicated to use. It requires a helper class called a TableListBoxModel which provides the TableListBox with access to data to display and helper functions which paint the individual cells in the table.

We are going to wrap another class around TableListBox so that we can provide a simple public class interface to the main application. This class will be called PlaylistComponent.

Create a new class in Projucer

Load up your project in Projucer. Add a new Component class in the file explorer inside Projucer. Call it PlaylistComponent.

Save the project and open in your IDE.

Add the class to MainComponent

In MainComponent.h, include the PlaylistComponent.h header file and add a PlaylistComponent object to the private section of the class:

```
#include "PlaylistComponent.h"
...
private:
    PlaylistComponent playlistComponent;
...
```

Now addAndMakeVisible and implement resized. In MainComponent::MainComponent:

```
...
addAndMakeVisible(playlistComponent);
...
```

In MainComponent::resized, work out how you want to lay out your application. I have given the playlist half of the window here (note that your DeckGUI

variables might be called deck1 and deck2, not deckGUI1 and deckGUI2. Adjust your code as appropriate):

```
deckGUI1.setBounds(0, 0, getWidth()/2, getHeight() / 2);
deckGUI2.setBounds(getWidth()/2, 0, getWidth()/2, getHeight() / 2);
playlistComponent.setBounds(0, getHeight()/2, getWidth(), getHeight()/2);
```

Now compile and run. Verify that you can see the playlistComponent displaying under the two DeckGUIs.

Add a TableListBox Component to the PlaylistComponent

Now we have a new Component up and running, we will add the TableListBox Component to PlaylistComponent.

In PlaylistComponent.h, add this to the private section of the class:

```
TableListBox tableComponent;
```

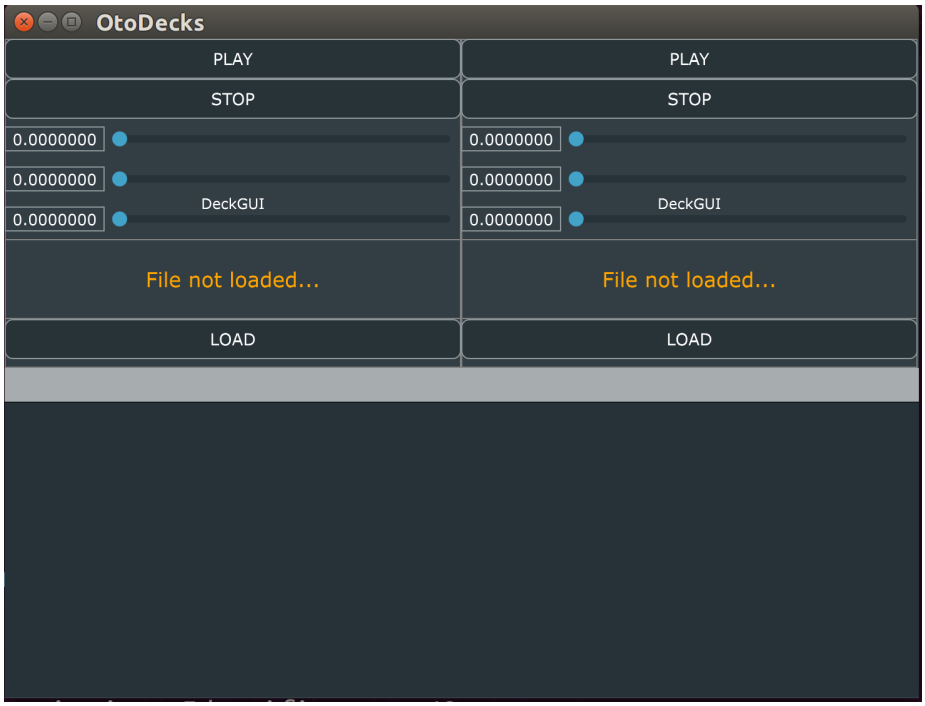
Take the usual step of calling addAndMakeVisible in PlaylistComponent::PlaylistComponent:

```
addAndMakeVisible(tableComponent);
```

Then in PlaylistComponent::resized, let the TableListBox take up the whole space:

```
void PlaylistComponent::resized()
{
    tableComponent.setBounds(0, 0, getWidth(), getHeight());
}
```

Now compile and run. You should see a single row table, something like this:



Add some columns

Let's add some columns to the table. In PlaylistComponent::PlaylistComponent:

```
tableComponent.getHeader().addColumn("Track title", 1, 400);
tableComponent.getHeader().addColumn("", 2, 200);
```

The first paramter is the column title, the second is the id and the third is the width in pixels. Note that the ids have to start at 1 as it does not accept zero ids on some systems.

If you build and run, you should see there are columns at the top of the table now.

Provide some data and drawing code for the table

Now we need some data to display in the table. This is where things get a bit tricky. To display data in a TableListBox, we need to inherit from the TableListBoxModel class and register with the table as a model. The idea is that the TableListBox needs a helper object which provides a standardised set of functions. The TableListBox calls these functions to get information about the data for the table and the way that data should be drawn.

Inherit from the TableListBoxModel

TableListBoxModel is an abstract class - it has the following pure virtual functions which we must implement:

```
virtual int getNumRows ()=0

virtual void paintRowBackground (Graphics &, int rowNumber, int width, int height, bool rowI

virtual void paintCell (Graphics &, int rowNumber, int columnId, int width, int height, bool
```

We are going to implement these on the PlaylistComponent class. In PlaylistComponent.h, update the inheritance relationship:

```
class PlaylistComponent    : public Component,
                             public TableListBoxModel
```

Add the pure virtual functions

Add the pure virtual functions in PlaylistComponent.h, in the public section of the class definition:

```
int getNumRows () override;

void paintRowBackground (Graphics &,
                         int rowNumber,
                         int width,
                         int height,
                         bool rowIsSelected) override;

void paintCell (Graphics &,
                         int rowNumber,
                         int columnId,
                         int width,
                         int height,
                         bool rowIsSelected) override;
```

The names of the functions are somewhat self-explanatory, but this is what each is for:

- getNumRows returns the number of rows in the table
- paintRowBackground provides graphical code which draws the background of a row in the table
- paintCell contains graphical code which draws the contents of individual cells

Now put in basic implementations to PlaylistComponent.cpp:

```
int PlaylistComponent::getNumRows ()
```

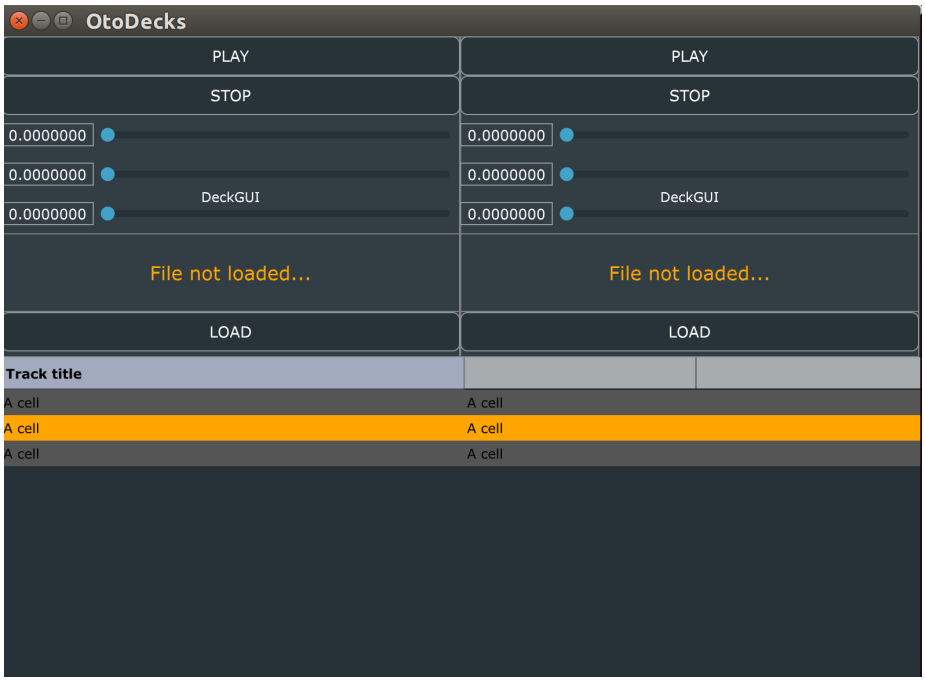
```
{
    return 3; // we'll have three rows for now
}

void PlaylistComponent::paintRowBackground (Graphics & g,
                    int rowNumber,
                    int width,
                    int height,
                    bool rowIsSelected)
{
    // just highlight selected rows
    if (rowIsSelected)
    {
        g.fillAll(Colours::orange);
    }
    else{
        g.fillAll(Colours::darkgrey);
    }
}

void PlaylistComponent::paintCell (Graphics & g,
                    int rowNumber,
                    int columnId,
                    int width,
                    int height,
                    bool rowIsSelected)
{
    g.drawText ("A cell", // we will change this later
                2, 0,
                width - 4, height,
                Justification::centredLeft,
                true);
}
```

Register as a model with the TableListBox

Now we need to register the PlaylistComponent with the TableListBox as a TableListBoxModel. This will tell the table to call the functions we just implemented. In PlaylistComponent.cpp, PlaylistComponent::PlaylistComponent:

```
tableComponent.setModel(this);
```

Build and run. You should see a three-row table. If you click on a row, it should highlight in orange, something like this:



Add some data to PlaylistComponent

Now we have the table and data working, we need to provide some proper data. Remember that we are building a playlist here, so the data should be information about audio files. For now, we will put our data into a vector. In PlaylistComponent.h, private section:

```
std::vector<std::string> trackTitles;
```

Put some data into trackTitles in PlaylistComponent::PlaylistComponent:

```
trackTitles.push_back("Track 1");
trackTitles.push_back("Track 2");
trackTitles.push_back("Track 3");
trackTitles.push_back("Track 4");
```

Use the data in the TableListBoxModel functions

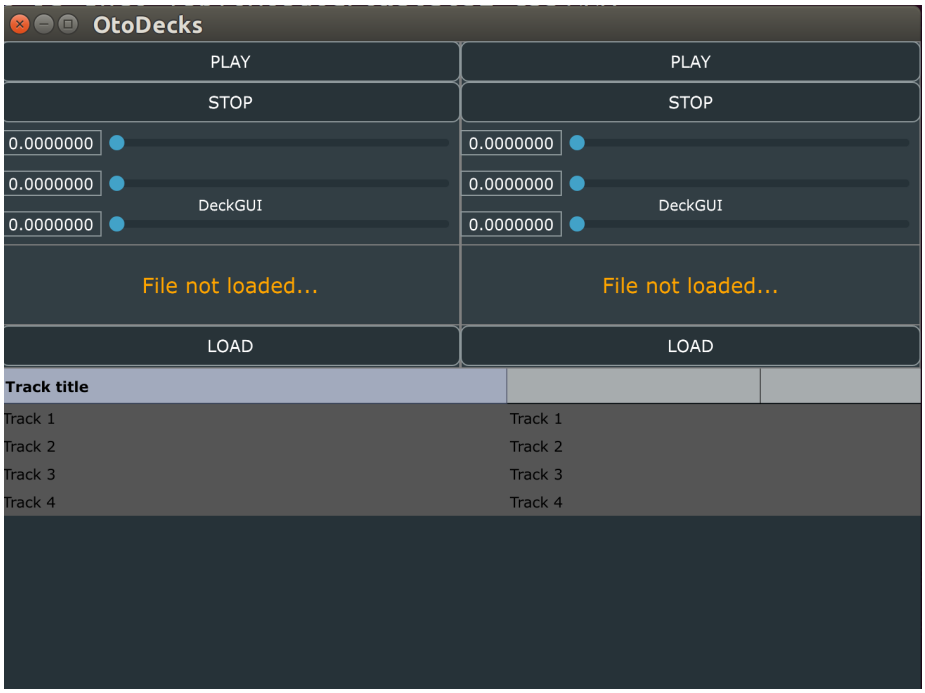
In PlaylistComponent.cpp, change getNumRows and paintCell:

```
int PlaylistComponent::getNumRows ()
{
    return trackTitles.size();
}
```

```
void PlaylistComponent::paintCell (Graphics & g,
                     int rowNumber,
                     int columnId,
                     int width,
                     int height,
                     bool rowIsSelected)
{
    g.drawText (trackTitles[rowNumber], // the important bit
                2, 0,
                width - 4, height,
                Justification::centredLeft,
                true);
}
```

Notes: * The number of rows now responds to the size of the vector * We pull the track title from the vector using the rowNumber

Build and run. You should see the appropriate track titles displayed in the table, like this:



Adding buttons to the table

Now we have the playlist working with data, we need a way to trigger events when the user clicks on a row.

The simplest option is to implement another function from the TableListBox-Model class. Here is the signature for that function from the TableListBoxModel class:

```
virtual void cellClicked (int rowNumber, int columnId, const MouseEvent &)
```

This is not pure virtual, so we do not have to implement it, but we can override it with our own implementation.

Can you add the function signature to PlaylistComponent.h then write an implementation in PlaylistComponent.cpp? See if you can print out the row and column ids.

Embedding Components to a table cell

It would be neat if we could embed Components such as Buttons in the cell. This would allow us to create more familiar controls for the playlist. Let's add a Button to the second column.

To do this, we need to override another function from TableListBoxModel. Add this to PlaylistComponent.h, in the public section of the class definition:

```
 Component* refreshComponentForCell (int rowNumber,
                                     int columnId,
                                     bool isRowSelected,
                                     Component *existingComponentToUpdate) override;
```

Our job with this function is to create and return a Component. That Component will then be drawn in the table cell. If we don't want to make a custom Component, we can just return the Component sent into the function, called existingComponentToUpdate.

Write an implementat in PlaylistComponent.cpp like this:

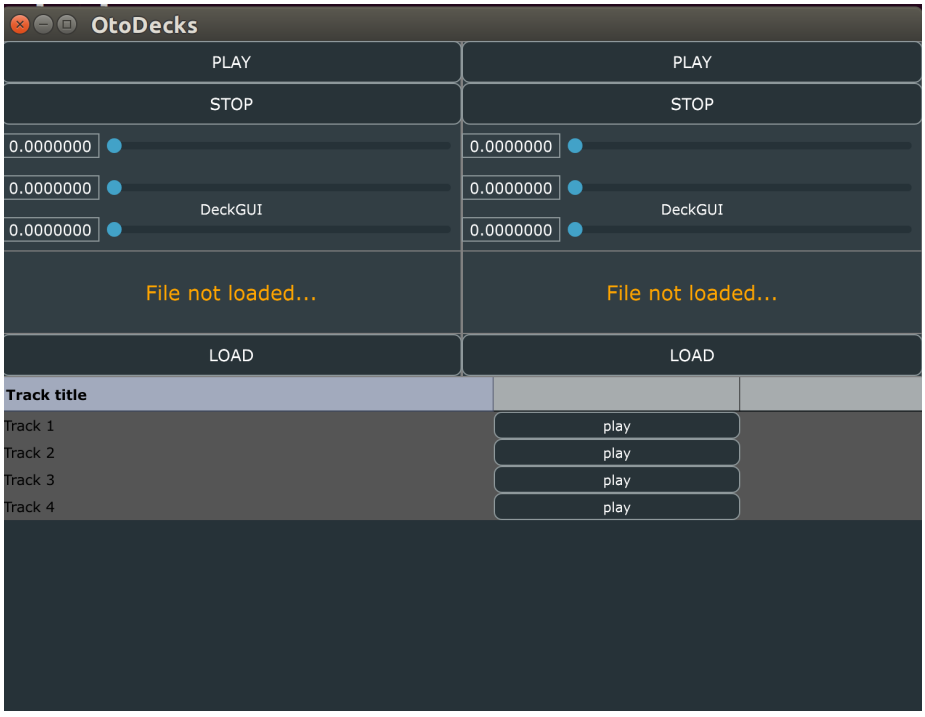
```
Component* PlaylistComponent::refreshComponentForCell (
                    int rowNumber,
                    int columnId,
                    bool isRowSelected,
                    Component *existingComponentToUpdate)
{
    if (columnId == 1)
    {
        if (existingComponentToUpdate == nullptr)
        {
```

```
            TextButton* btn = new TextButton("play");
            existingComponentToUpdate = btn;
        }
    }
    return  existingComponentToUpdate;
}
```

There is quite a lot to unpack there.

- This function will be called for every cell in the table
- If the cell is *not* in the column with id 1, we just return the sent Component, which will actually be nullptr. This will only draw the table cell as usual
- If the cell is in column 1 which is the second column, we want to draw a Button
- We check if the existingComponent coming in is nullptr. It will be nullptr if we have not yet created the Button
- If it is nullptr, we create a TextButton and store it into existingComponent
- At the end, we return existingComponent

If it all went according to plan, you should see something like this:



Add a Button Listener

Now we have buttons on the interface, we need a button listener to respond to them.

In the Playlist.h file, update the inheritance relationships:

```
class PlaylistComponent    : public Component,
                             public TableListBoxModel,
                             public Button::Listener
```

Then add the buttonClicked function to the public section of the class definition in PlaylistComponent.h:

```
void buttonClicked(Button* button) override;
```

Now add a simple implementation to PlaylistComponent.cpp:

```
void PlaylistComponent::buttonClicked(Button* button)
{
DBG("PlaylistComponent::buttonClicked ");
}
```

Finally, register with the buttons for events. In PlaylistComponent::refreshComponentForCell:

```
...
btn->addListener(this);
...
```

Build and run. Verify that you are receiving events in the PlaylistComponent.

Which Button?

In DeckGUI, all the buttons were class data members, and we could check the memory address to find out which button the user clicked on. We can't do that here because we do not have class data members storing the Buttons, we are creating them dynamically, then losing access to them.

The solution is to set ids on the buttons when we create them, then check those ids in the event listener. In PlaylistComponent::refreshComponentForCell:

```
String id{std::to_string(rowNumber)};
btn->setComponentID(id);
```

In PlaylistComponent::buttonClicked(Button* button), something like this:

```
void PlaylistComponent::buttonClicked(Button* button)
{
    int id = std::stoi(button->getComponentID().toStdString());
    DBG("PlaylistComponent::buttonClicked " << trackTitles[id]);
}
```

Build and run. Check that you now have the row number coming back into your event listener so you can correctly print out the track title.

Conclusion

In this worksheet, we have built a Component that can display tabular data. The Component is a suitable basis upon which to build a fully functional playlist for the DJ application.