Mobile Application Development with JUCE and Native API's

Adam Wilson

JUCE Summit 2015

Creating mobile apps that mix JUCE components with native iOS or Android UI elements.

Why?

- JUCE is a cross-platform framework
- Write once, run anywhere

JUCE Components were designed for desktop apps, not for touch screen devices.

- Limited support for gestures
- Hard to get the feel right for common UI interactions

E.g. ListBox component

E.g. ListBox component

- Needs adapting to respond to finger gestures

E.g. ListBox component

- Needs adapting to respond to finger gestures
- Hard to get it to scroll smoothly when there is e.g. audio processing happening in the background

Navigation

Navigation

- E.g. swipe to move between pages

Navigation

- E.g. swipe to move between pages
- Problem: Components grab mouse events

Solution: Utilise native API's

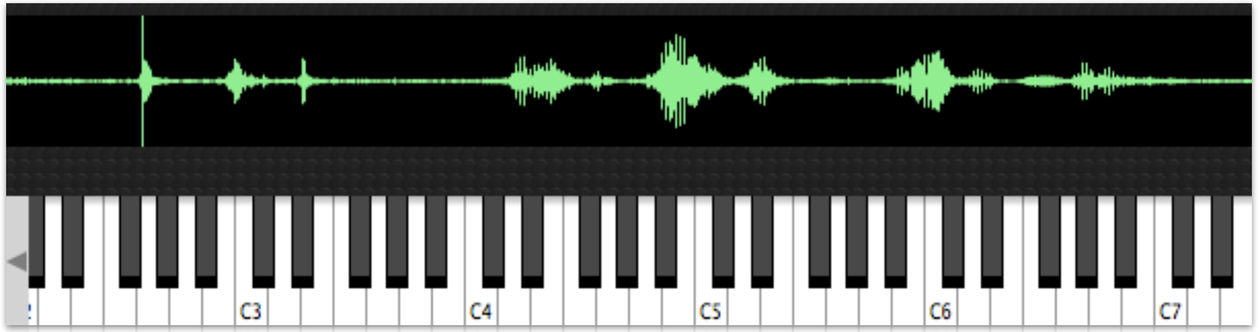
Platform	Language	API
Android	Java	Android API's and Android Support Libraries
iOS	Objective-C / Swift	Cocoa Touch

Use Native API’s for:

- Scrollable Lists
- Navigation
- User input
- Database access

Use JUCE for:

- Audio and data processing
- Custom graphics and animation (OpenGL)
- Custom or specialised GUI elements
  e.g. MidiKeyboardComponent



Advantages of this approach

Advantages of this approach

- Familiar UI elements using each platform’s native idioms

Advantages of this approach

- Familiar UI elements using each platform’s native idioms
- Smooth scrolling, gesture support

Advantages of this approach

- Familiar UI elements using each platform’s native idioms
- Smooth scrolling, gesture support
- Comprehensive toolkits for creating GUI’s: Xcode and Android Studio

Advantages of this approach

- Familiar UI elements using each platform’s native idioms
- Smooth scrolling, gesture support
- Comprehensive toolkits for creating GUI’s: Xcode and Android Studio
- No need to code transitions and other animations - these are provided by the native API’s

Advantages of this approach

- Familiar UI elements using each platform’s native idioms
- Smooth scrolling, gesture support
- Comprehensive toolkits for creating GUI’s: Xcode and Android Studio
- No need to code transitions and other animations - these are provided by the native API’s
- Use a local database e.g. SQLite, Couchbase Mobile, Firebase and connect directly to your UI

Disadvantages of this approach

Disadvantages of this approach

- Designing and coding part of your UI for each platform

Disadvantages of this approach

- Designing and coding part of your UI for each platform
- Potentially complicated communication between languages:

Java ⟷ C++

Objective-C ⟷ C++

Swift ⟷ Objective-C ⟷ C++

How?

There are two things to consider:

How?

There are two things to consider:

1. Mixing JUCE components with native UI

How?

There are two things to consider:

1. Mixing JUCE components with native UI
2. Passing data between C++ and Java, or C++ and Objective-C

Disclaimer:

Assume that most of the time we will only want to call C++ from the native UI code, not the other way round.

Android + JUCE: UI

- We can now use Android Studio!

  (This turns out to be a nice IDE also for C++ development)

Android + JUCE: UI

- Introjucer’s Android exporter generates a Java Activity class

Android + JUCE: UI

- Introjucer's Android exporter generates a Java Activity class
- The default activity is full screen

Android + JUCE: UI

- Introjucer's Android exporter generates a Java Activity class
- The default activity is full screen
- We want our JUCE component as part of a layout

Android + JUCE: UI

- Introjucer's Android exporter generates a Java Activity class
- The default activity is full screen
- We want our JUCE component as part of a layout
- So we need to do some modifications

Android + JUCE: UI

In your default generated Activity class onCreate:

```
viewHolder = new ViewHolder (this);
setContentView (viewHolder);
```

Android + JUCE: UI

In your default generated Activity class onCreate:

```
viewHolder = new ViewHolder (this);
setContentView (viewHolder);
```

Would become something like:

```
viewHolder = new ViewHolder (this);
setContentView(R.layout.main_activity);
LinearLayout juceViewContainer = (LinearLayout) findViewById(R.id.juce_view_container);
juceViewContainer.addView(viewHolder);
```

Android + JUCE: UI

In your default generated Activity class onCreate:

```
viewHolder = new ViewHolder (this);
setContentView (viewHolder);
```

Would become something like:

main_activity.xml layout file

```
viewHolder = new ViewHolder (this);
setContentView(R.layout.main_activity);
LinearLayout juceViewContainer = (LinearLayout) findViewById(R.id.juce_view_container);
juceViewContainer.addView(viewHolder);
```

Android + JUCE: UI

In your default generated Activity class onCreate:

```
viewHolder = new ViewHolder (this);
setContentView (viewHolder);
```

Would become something like:

main_activity.xml layout file

juce window within layout

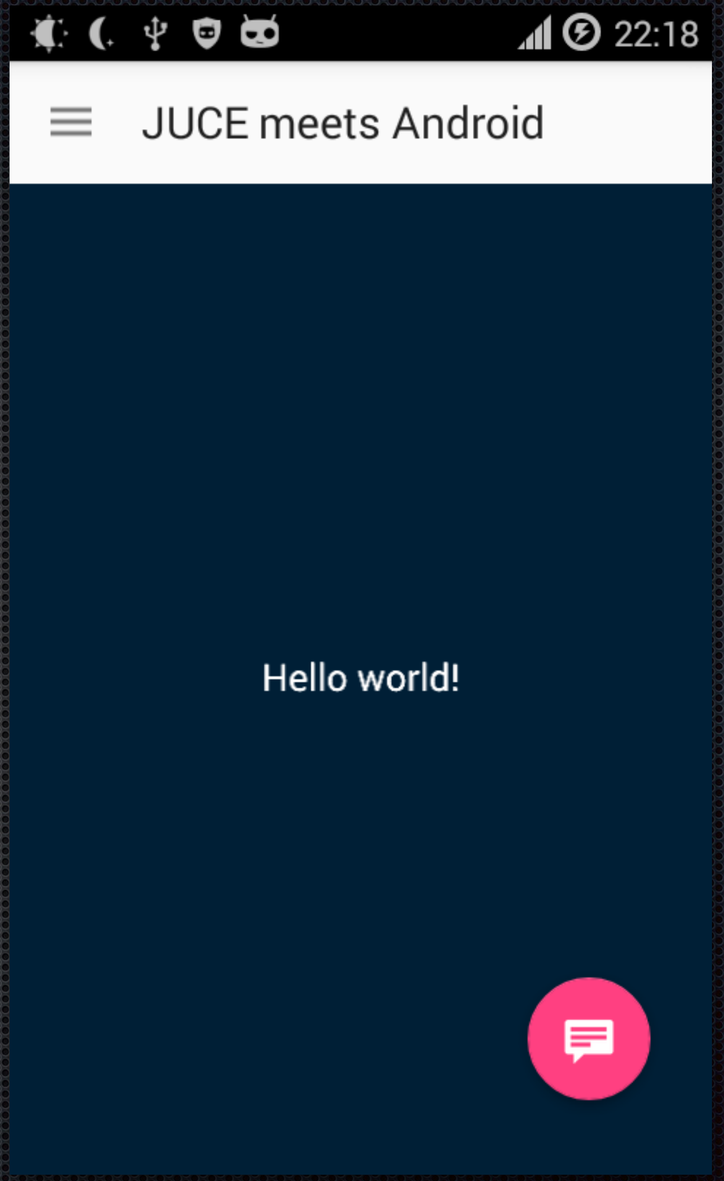
```
viewHolder = new ViewHolder (this);
setContentView(R.layout.main_activity);
LinearLayout juceViewContainer = (LinearLayout) findViewById(R.id.juce_view_container);
juceViewContainer.addView(viewHolder);
```

Android + JUCE: UI

If we want to use the recent Android Support Libraries, which are necessary for Material Design style apps then we also need to change the base class (superclass) from Activity to AppCompatActivity:

```
import android.support.v7.app.AppCompatActivity;
//…

public class JuceActivity extends AppCompatActivity
{
    //…
}
```

Android + JUCE



Android + JUCE: Data

- Declare our JNI functions in Java and implement them in C++
- The C++ functions could be getters, setters, actions etc.

Android + JUCE: Data

Sending data to C++:

Android + JUCE: Data

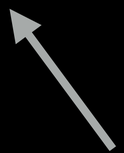
Sending data to C++:

```
#if JUCE_ANDROID

JUCE_JNI_CALLBACK (JUCE_ANDROID_ACTIVITY_CLASSNAME, setMessage, void, (JNIEnv* env,
jclass, jstring message))
{
    SharedResourcePointer<MainContentComponent> mainComponent;

    mainComponent->message = juceString (env, message);
    mainComponent->repaint();
}

#endif
```

Android + JUCE: Data

Sending data to C++:

```
#if JUCE_ANDROID

JUCE_JNI_CALLBACK (JUCE_ANDROID_ACTIVITY_CLASSNAME, setMessage, void, (JNIEnv* env,
jclass, jstring message))
{
    SharedResourcePointer<MainContentComponent> mainComponent;

    mainComponent->message = juceString (env, message);
    mainComponent->repaint();
}

#endif
```



Android JNI Helper method
converts jstring to JUCE String

Android + JUCE: Data

Sending data to C++:

```
#if JUCE_ANDROID

JUCE_JNI_CALLBACK (JUCE_ANDROID_ACTIVITY_CLASSNAME, setMessage, void, (JNIEnv* env,
jclass, jstring message))
{
    SharedResourcePointer<MainContentComponent> mainComponent;

    mainComponent->message = juceString (env, message);
    mainComponent->repaint();
}

#endif
```

This is a global function

Android + JUCE: Data

Declare it in your Activity class:

```
public static native void setMessage (String message);
```

Android + JUCE: Data

Declare it in your Activity class:

```
public static native void setMessage (String message);
```

and then wherever you want to call that function, e.g.

```
JuceActivity.setMessage("My new message");
```

Android + JUCE: Data

We might also want to pull data from our C++ code - e.g. a list of menu items

Android + JUCE: Data

Pulling data from C++ takes a little more work, e.g. in the case of a string:

```
JUCE_JNI_CALLBACK (JUCE_ANDROID_ACTIVITY_CLASSNAME, getJsonDataBytes, jbyteArray, (JNIEnv* env,
jclass))
{
    SharedResourcePointer<MainContentComponent> mainComponent;
    String jsonData = mainComponent->data.toJson();

    int byteCount = jsonData.length();
    const jbyte* nativeString = reinterpret_cast<const jbyte*> ((const char *) jsonData.toUTF8());
    jbyteArray bytes = env->NewByteArray (byteCount);
    env->SetByteArrayRegion (bytes, 0, byteCount, nativeString);

    return bytes;
}
```

Android + JUCE: Data

In your Activity class:

```
public static String getJsonData()
{
    return new String(getJsonDataBytes(), Charset.forName("UTF-8"));
}

private static native byte[] getJsonDataBytes();
```

Android + JUCE: Data

- Thats it!
- If you do need to call Java from C++, you need to add your function definition to juce_android_JNIHelpers.h

  (I’ve not had to do this yet)

iOS + JUCE: UI

- JUCE Window is a UIView
- Therefore we can add it as a subview to any other view in an iOS app
  E.g. a UIViewController

iOS + JUCE: UI

- We need to modify the MainWindow class and make it an Objective-C++ file (.mm)

iOS + JUCE: UI

- We need to modify the MainWindow class and make it an Objective-C++ file (.mm)
- Here we:
    1. Create a UIWindow

iOS + JUCE: UI

```
MainWindow::MainWindow (String name)  : DocumentWindow (name,
                                                        Colours::lightgrey,
                                                        DocumentWindow::allButtons)
{
    UIWindow* window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
```

iOS + JUCE: UI

- We need to modify the MainWindow class and make it an Objective-C++ file (.mm)
- Here we:
    1. Create a UIWindow
    2. Create a UIView

iOS + JUCE: UI

```
MainWindow::MainWindow (String name)  : DocumentWindow (name,
                                                        Colours::lightgrey,
                                                        DocumentWindow::allButtons)
{
    UIWindow* window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    UIView* juceView = [[UIView alloc] initWithFrame: [[UIScreen mainScreen] bounds]];
```

iOS + JUCE: UI

- We need to modify the MainWindow class and make it an Objective-C++ file (.mm)
- Here we:
    1. Create a UIWindow
    2. Create a UIView
    3. Add our Component to the UIView

iOS + JUCE: UI

```
MainWindow::MainWindow (String name)  : DocumentWindow (name,
                                                        Colours::lightgrey,
                                                        DocumentWindow::allButtons)
{
    UIWindow* window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    UIView* juceView = [[UIView alloc] initWithFrame: [[UIScreen mainScreen] bounds]];
    MainWindow::addComponentToUIView (mainComponent.get(), juceView);
```

iOS + JUCE: UI

- We need to modify the MainWindow class and make it an Objective-C++ file (.mm)
- Here we:
    1. Create a UIWindow
    2. Create a UIView
    3. Add our Component to the UIView
    4. E.g. add our JUCE UIView as a subview of a UIViewController

iOS + JUCE: UI

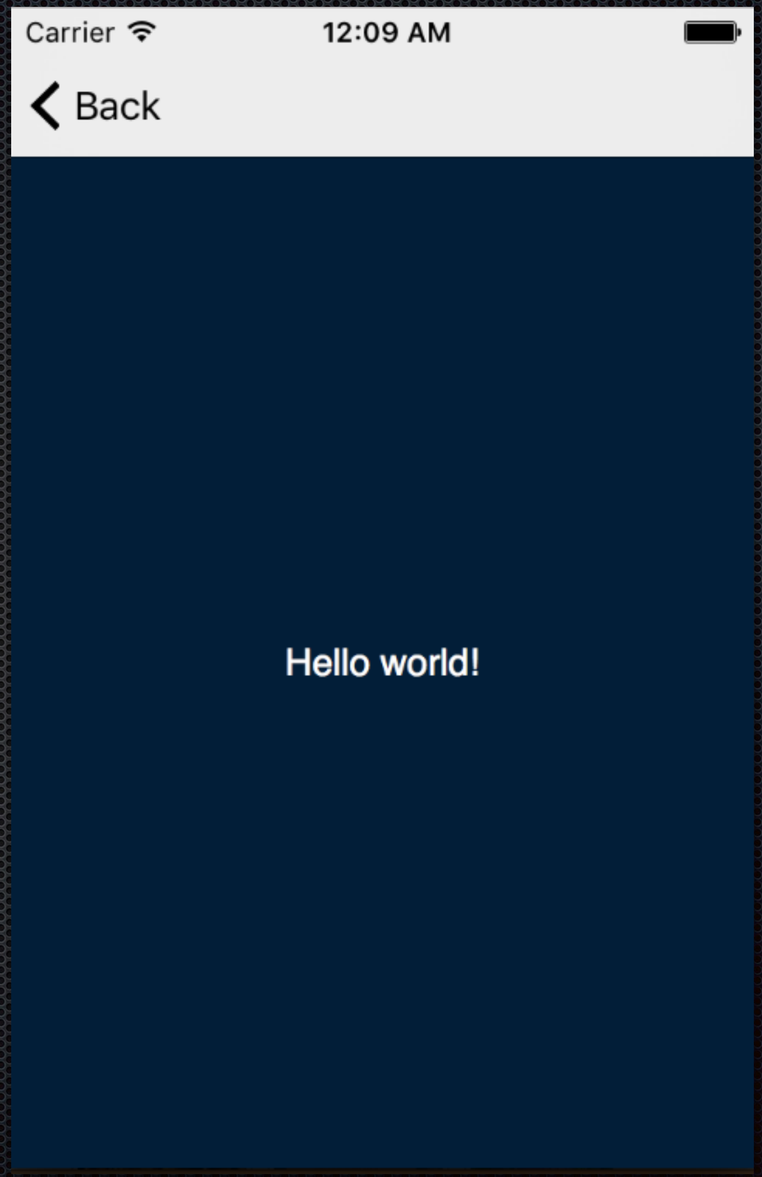
```
MainWindow::MainWindow (String name)  : DocumentWindow (name,
                                                        Colours::lightgrey,
                                                        DocumentWindow::allButtons)
{
    UIWindow* window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    UIView* juceView = [[UIView alloc] initWithFrame: [[UIScreen mainScreen] bounds]];
    MainWindow::addComponentToUIView (mainComponent.get(), juceView);

    JuceViewController* juceViewController = // Subclass of UIViewController
        [[JuceViewController alloc] initWithContentView: juceView];

    juceViewController.contentView = juceView;

    //…
```

iOS + JUCE: UI

Handy function:

```
void MainWindow::addComponentToUIView (Component& component, void* uiView)
{
    component.addToDesktop (0, uiView);
    UIView* view = (UIView*) uiView;
    component.setVisible (true);
    component.setBounds (view.bounds.origin.x, view.bounds.origin.y,
                         view.bounds.size.width, view.bounds.size.height);
}
```

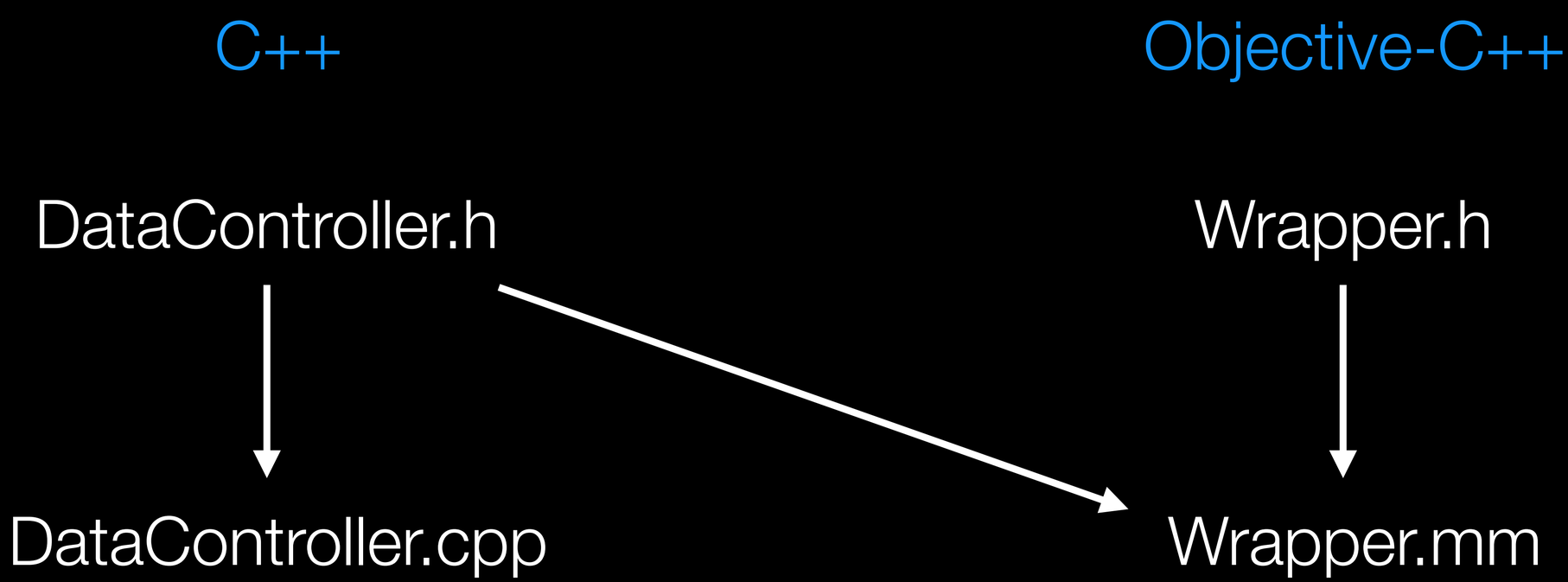
iOS + JUCE



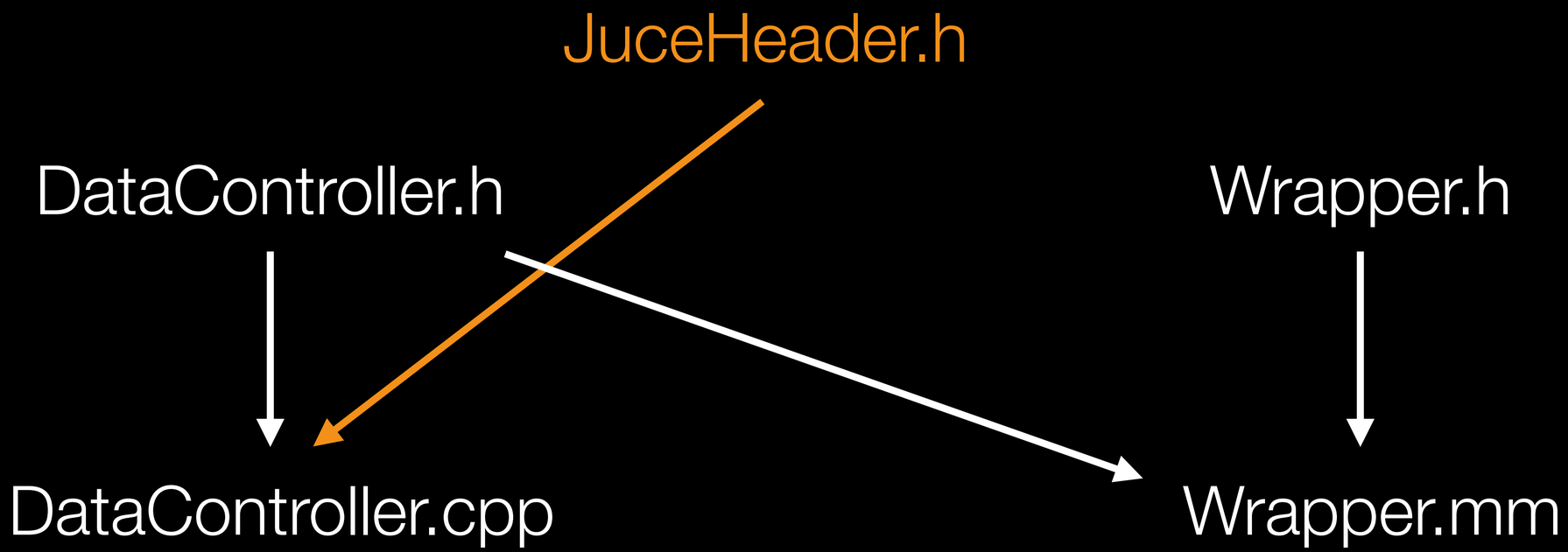
iOS + JUCE: Data

- We need to two layers
- One that is visible to C++, and the other that is visible to Objective-C.
- Lets call these DataController and Wrapper

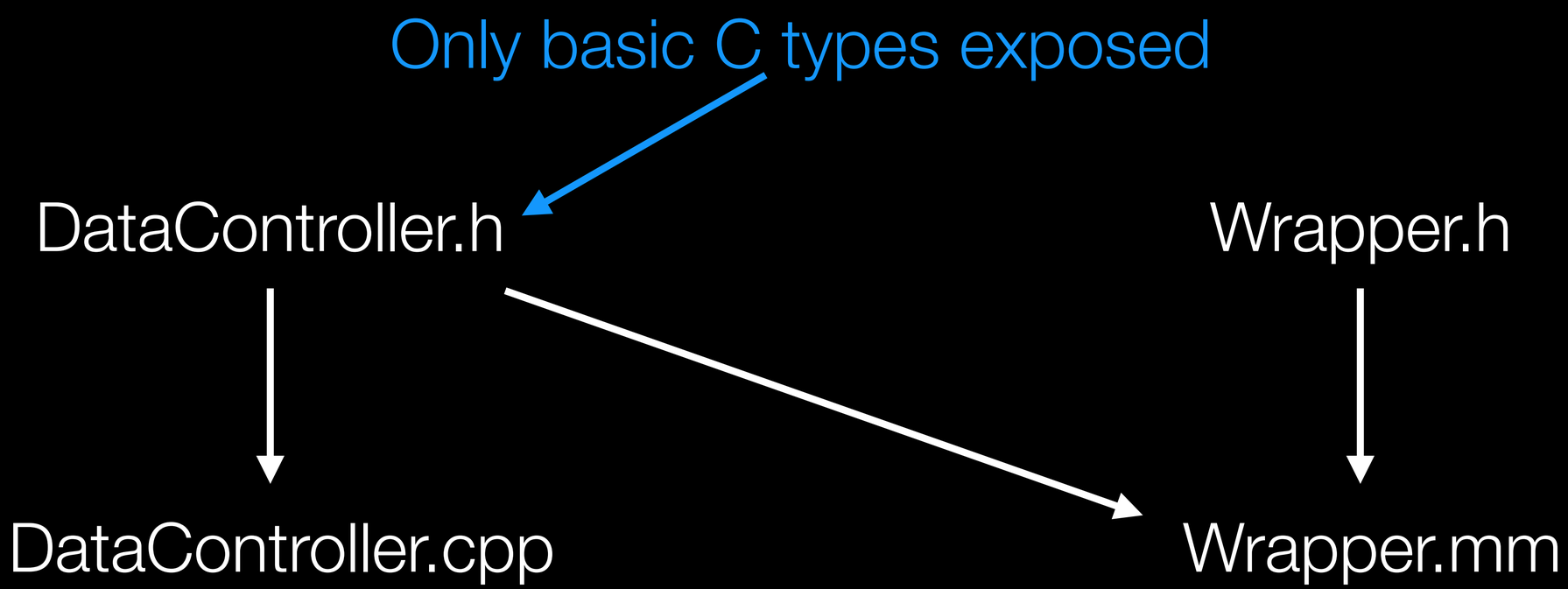
iOS + JUCE: Data



iOS + JUCE: Data



iOS + JUCE: Data

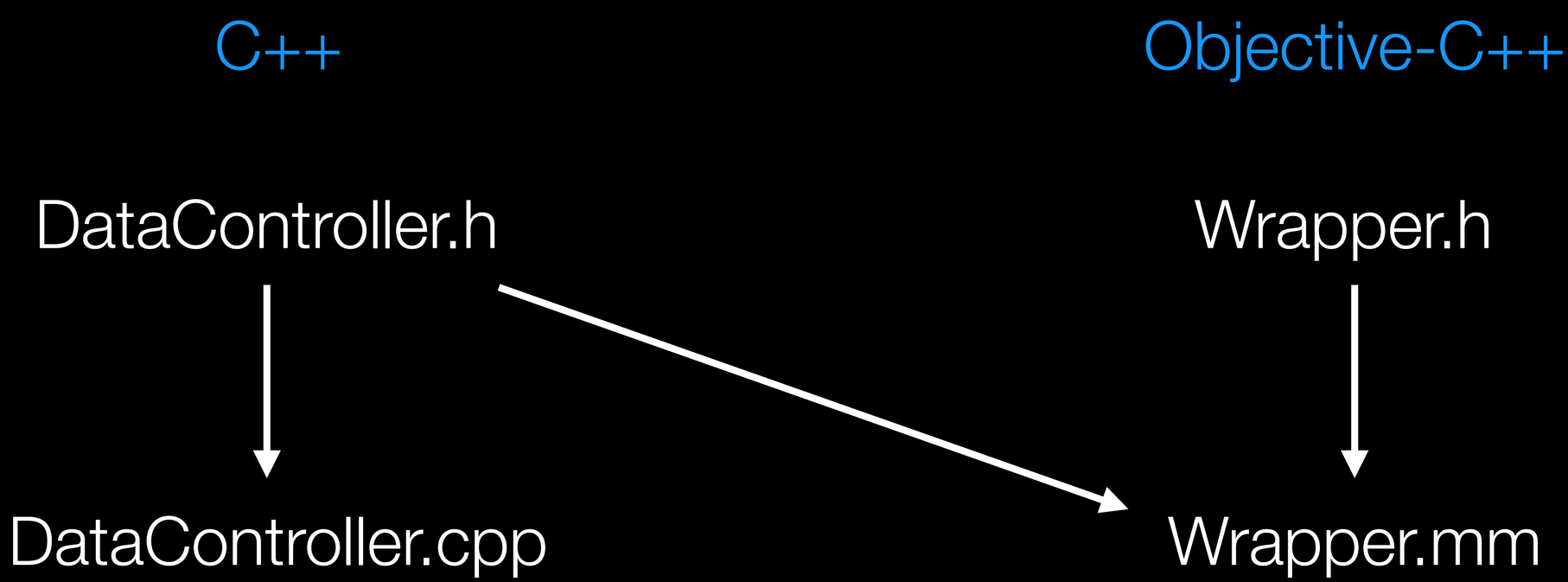


iOS + JUCE: Data

DataController.h

```
/**
 * Methods to expose to Objective-C
 * Note that only basic C types are exposed
 */

struct DataController
{
    DataController();

    /** Get JSON string of all message data */
    const char* getJsonData();

    /** Set message and repaint MainContentComponent */
    void setMessage (const char *title, const char* message);
};
```

iOS + JUCE: Data

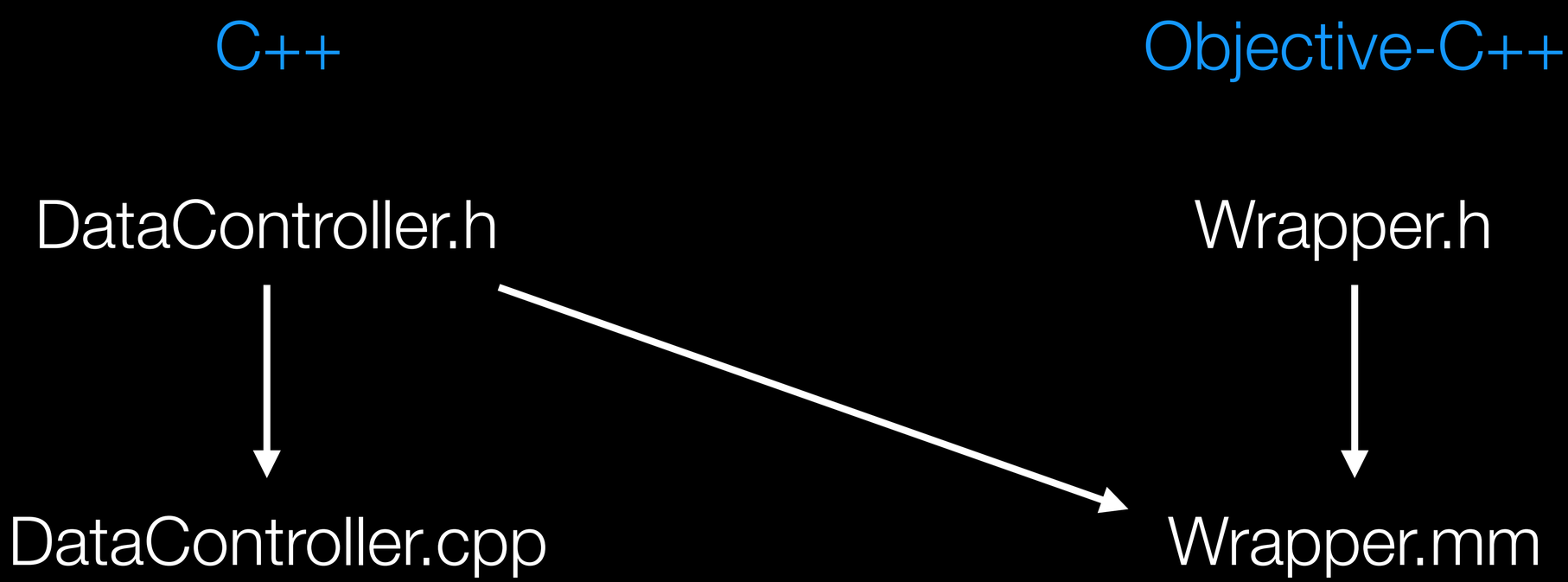


iOS + JUCE: Data

DataController.cpp

```
#include "JuceHeader.h"
#include "DataController.h"
#include "MainComponent.h"
#include "Data.h"

void DataController::setMessage (const char *title, const char* message)
{
    SharedResourcePointer<MainContentComponent> mainComponent;
    mainComponent->title = (String) title;
    mainComponent->message = (String) message;
    mainComponent->repaint();
}

const char* DataController::getJsonData()
{
    SharedResourcePointer<MainContentComponent> mainComponent;
    return mainComponent->data.toJson().toRawUTF8();
}
```

iOS + JUCE: Data

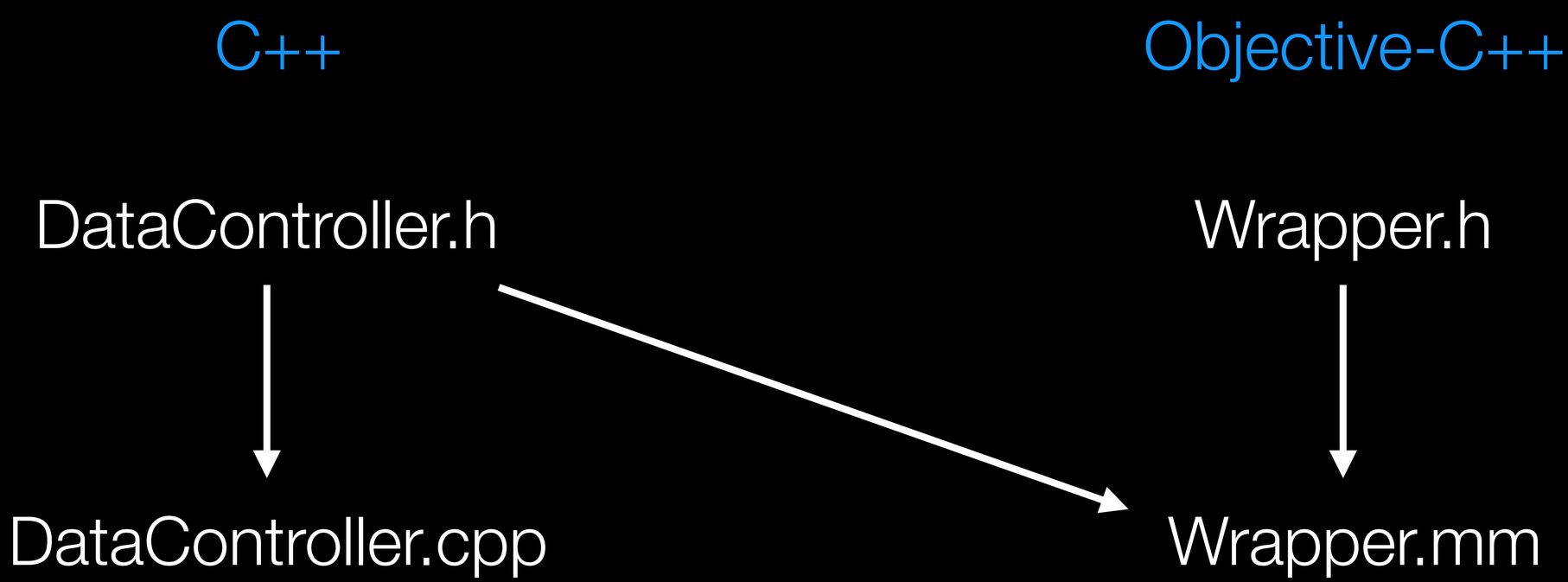


iOS + JUCE: Data

Wrapper.h

```objc
#import <Foundation/Foundation.h>

@interface Wrapper : NSObject

/** Set message and repaint JUCE MainContentComponent */
- (void)setMessage:(NSString*) message;

/** Get Data as JSON String */
- (NSString*)getJsonData;

@end
```

iOS + JUCE: Data



iOS + JUCE: Data

Wrapper.mm

```
#import <Foundation/Foundation.h>
#import "Wrapper.h"
#import "DataController.h"

@interface Wrapper ()
{
    DataController* wrapped;
}
@end

@implementation Wrapper

- (id)init
{
    self = [super init];
    if (self)
    {
        wrapped = new DataController();
        if (!wrapped) self = nil;
    }
    return self;
}
```

http://philjordan.eu/article/mixing-objective-c-c++-and-objective-c++

iOS + JUCE: Data

Wrapper.mm

```
#import <Foundation/Foundation.h>
#import "Wrapper.h"
#import "DataController.h"

@interface Wrapper ()
{
    DataController* wrapped;
}
@end

@implementation Wrapper

- (id)init
{
    self = [super init];
    if (self)
    {
        wrapped = new DataController();
        if (!wrapped) self = nil;
    }
    return self;
}
```

```
- (void)setMessage: (NSString*) message
{
    const char* messageChar = [message UTF8String];
    wrapped->setMessage(messageChar);
}

- (NSString*)getJsonData
{
    const char* jsonCharPointer = wrapped->getJsonData();
    NSString* jsonDataString = [NSString stringWithFormat:@"%s"
                                jsonCharPointer];
    return jsonDataString;
}

- (void)dealloc
{
    delete wrapped;
    [super dealloc];
}

@end
```

http://philjordan.eu/article/mixing-objective-c-c++-and-objective-c++

iOS + JUCE: Data

Wrapper.mm

```
#import <Foundation/Foundation.h>
#import "Wrapper.h"
#import "DataController.h"

@interface Wrapper ()
{
    DataController* wrapped;
}
@end

@implementation Wrapper

- (id)init
{
    self = [super init];
    if (self)
    {
        wrapped = new DataController();
        if (!wrapped) self = nil;
    }
    return self;
}
```

```
- (void)setMessage: (NSString*) message
{
    const char* messageChar = [message UTF8String];
    wrapped->setMessage(messageChar);
}

- (NSString*)getJsonData
{
    const char* jsonCharPointer = wrapped->getJsonData();
    NSString* jsonDataString = [NSString stringWithFormat:@"%s"
                                jsonCharPointer];
    return jsonDataString;
}

- (void)dealloc
{
    delete wrapped;
    [super dealloc];
}

@end
```

http://philjordan.eu/article/mixing-objective-c-c++-and-objective-c++

iOS + JUCE: Data

- Any Objective-C class can now call the exposed methods via

```
#import "DataControllerWrapper.h"
```

iOS + JUCE: Data

- Any Objective-C class can now call the exposed methods via

```
#import "DataControllerWrapper.h"
```

- This includes **Swift** - in that case it would be in e.g. MyJuceApp-Header.h (This is a special header file that allows us to expose Objective-C methods to Swift)

- I would recommend using Swift if possible

iOS + JUCE: Data

- If we want to call Objective-C (or Swift!) from C++ then we need to employ the PIMPL idiom
- This is the subject of another talk!

  See http://philjordan.eu/article/mixing-objective-c-c++-and-objective-c++

Tying things together

- Seperate MainWindow class from Main.cpp
- Have an Objective-C++ version (e.g. iosWindow.h and iosWindow.mm)
- Then in your Main.cpp:

```
#if JUCE_IOS
#include "iosMainWindow.h"
#else
#include "MainWindow.h"
#endif
```

Thanks for listening!

Example project source code available at:

github.com/adamski/juce-native-navigation

adam@codegarden.co.uk