# Mobile Instruments Made Easy: Creating Musical Mobile Apps with LIBPD and iOS, No Experience Necessary

**Danny Holmes**
Louisiana State University
Center for Computation and Technology
`dholm13@tigers.lsu.edu`

## ABSTRACT

The quirks of programming native iOS applications can be daunting, especially to someone not already familiar with other text based programming languages. Fortunately, recent developments in Apple's Xcode IDE, along with the open-source Pure Data wrapper, LIBPD, enable the process of creating an iOS native, standalone, musical mobile app to be quite accessible. Basic implementation of LIBPD for iOS can be reduced to a series of 10 simple and accessible steps requiring no actual knowledge of objective-c or the LIBPD library itself, and requires no previous coding experience, text-based or otherwise. In addition, the drag-and-drop feature of Xcode's Storyboards allows the design and programming of a native user interface to also be reduced to simple and accessible step-by-step instructions. This is not just limited to buttons and sliders, but also includes the drag-and-drop creation of typical touch screen gestures. Presently, this process will be outlined and explained, and its current and potential uses described.

## 1. INTRODUCTION

The development tools and modern languages available today have made mobile programming more accessible than ever before [1]. While at least some experience and knowledge are still necessary to achieve all but the most rudimentary goals, it is entirely possible to engage those with limited or even no prior experience or knowledge and enable them to create their own musically capable mobile applications with little time investment. Initiatives such as the EMDM Academy (an Experimental Music and Digital Media Outreach) and the iOS Musical App-a-thon at Louisiana State University are designing and implementing curriculum, as well as identifying and exploring potential benefits to music education.

One technique being explored is the rapid-prototyping of standalone musical mobile applications by high school and college students. The effective length of our classes and workshops can be measured in hours (as opposed to days, weeks, or months), and many of the students who have been introduced to this process had no prior programming experience. Thus far, the workshops have featured either web-based applications or native iOS applications.

During an iOS Musical App-a-thon, students are introduced to mobile development using Apple's Xcode IDE and Pure Data, and they design and create a new application all within a single 8-10 hour period. In order to eliminate the need for prior knowledge and reduce setup time, it was necessary to stream-line the process for creating a LIBPD enabled working environment for prototyping musical mobile apps with Xcode.

## 2. LIBPD IN 10 STEPS

The following step-by-step process outlines a basic method for setting up LIBPD in an Xcode project. The code provided can simply be copied directly into the appropriate portion of your own project, and only a few specific changes to the code will be required. [1]

This process assumes you have already created a free Apple developer account, have installed Xcode, and have downloaded the LIBPD project from Github.

```
http://bit.ly/dev_register
http://bit.ly/dl_xcode
http://bit.ly/libpd_github
```

When the process is finished, the project will be set up for testing your app using Xcode's built-in iOS simulator. In order to put the app on a physical device, it will be necessary to join an existing development team or upgrade to a paid developer account. Once your account is included in an appropriate development profile, and your device is provisioned, only one extra step is required to set up your project for testing on an actual device.

### 2.1 Create an Xcode Project

Creating an Xcode project is fairly self-explanatory, but a few options must be considered. First, open Xcode and choose to "Create a New Xcode Project." Then select "Single View Application" (ensuring iOS is selected in the left-side menu) and fill out the project information. [2] Cur-

[1] These instructions were derived from various resources (including personal experience), but they began with the great information in Peter Brinkman's *Making Musical Apps* [2].

[2] Xcode 6 is due for public release in the third quarter of 2014. With the new version, it may be necessary to also choose the language (objective-c) and to deselect the option to use Core Data (unless it is essential to the app!).

rently, the prefix field and devices menu are the most important. The actual prefix is inconsequential but will be referenced in the next few steps. The device choice will affect the application's behavior on different devices. Setting the choice as "Universal" will automatically set up the project to work on all iOS devices, but requires a separate UI be created for phones and tablets. The "iPhone" option will also work on all devices, but iPads will emulate and roughly scale the UI designed for phones. The "iPad" option will not work on phones or iPods.

### 2.2 Copy LIBPD to the Project Folder

Once the project is created, simply copy the entire (unzipped) LIBPD project folder directly into the root of the Xcode project folder. This step is completed with a typical Finder window, not with Xcode. No changes should be made to the file structure inside the LIBPD project folder. Once this is done, return to the Xcode window with the project open.

### 2.3 Add the LIBPD Xcode Project

Steps 3-6 will require the main project file be selected in the navigator on the left. Notice the navigator can be set to show varying information with the buttons at the top. If the project is not visible, make sure the "Project Navigator" is selected (the folder icon). Now, choose "File - Add Files to..." in the top menu bar. Navigate to the LIBPD project folder just copied into the main project folder, and select the file libpd.xcodeproj. Before adding, ensure the "Copy items..." option is checked and that the project is selected under "Add to targets."

### 2.4 Set the User Header Search Path

In the center view, choose the "Build Settings" tab, and search for "User Header Search Paths." Double-click on the empty field and add the search path ../libpd-master/.. while also ensuring it is set to recursive. Please note that the path should simply reflect the name of the LIBPD project folder previously copied into the project folder.

### 2.5 Set up the Build Phases

Choose the "Build Phases" tab in the center view. Select "Target Dependencies" and add "libpd-ios." Then, under "Link Binary with Libraries," add:

libpd-ios.a [3]
AudioToolbox.framework
AVFoundation.framework

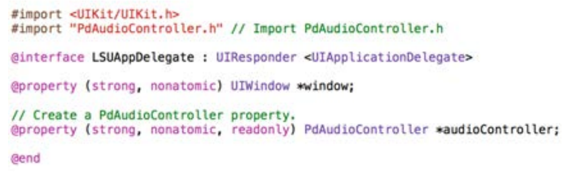
### 2.6 Add a PD Patch

Once again, choose "File - Add Files to..." in the top menu bar. This time, locate and add the PD patch that the app will use. Check that the same settings from step 3 are still selected.

[3] "libpd-ios.a" may show up in red text once added. This is a known bug and should not interfere with the project.

### 2.7 AppDelegate.h

Choose the AppDelegate.h file in the project navigator. Please note that the actual file name will reflect the project prefix chosen during setup (e.g. LSUAppDelegate.h). During this step, import the PdAudioController.h file and add a property for PdAudioController. It will only require two lines of code be added, and the file should look similar to Figure 1. The lines with comments next to or directly above them are the only ones added to the default project.

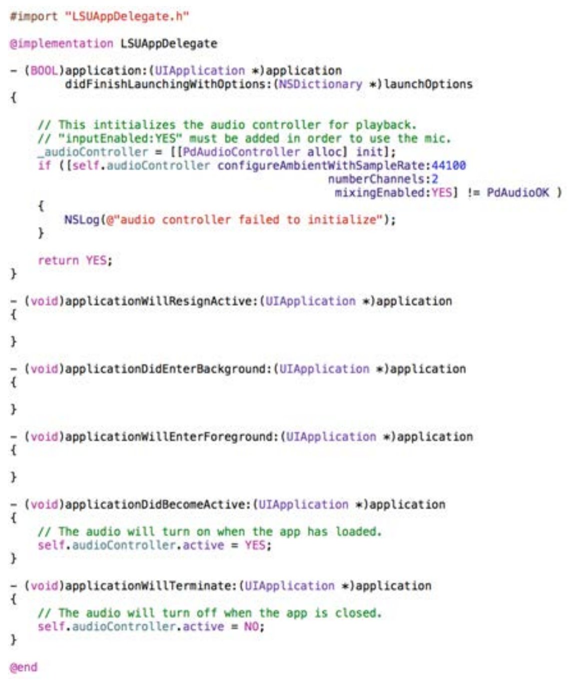


```
#import <UIKit/UIKit.h>
#import "PdAudioController.h" // Import PdAudioController.h

@interface LSUAppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;

// Create a PdAudioController property.
@property (strong, nonatomic, readonly) PdAudioController *audioController;

@end
```

**Figure 1**. AppDelegate.h

### 2.8 AppDelegate.m

Now, select the AppDelegate.m file. Here the audio controller is initialized and its life cycle determined. Locate the "didFinishLaunchingWithOptions," "applicationDidBecomeActive," and "applicationWillTerminate" sections and update them as shown in Figure 2. [4]



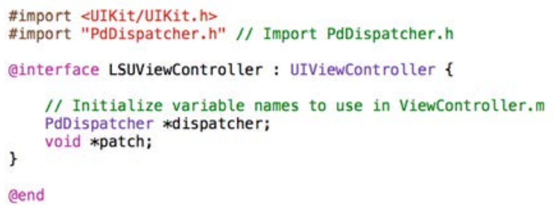
```
#import "LSUAppDelegate.h"

@implementation LSUAppDelegate

- (BOOL)application:(UIApplication *)application
        didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{

    // This intitializes the audio controller for playback.
    // "inputEnabled:YES" must be added in order to use the mic.
    _audioController = [[PdAudioController alloc] init];
    if ([self.audioController configureAmbientWithSampleRate:44100
                                            numberChannels:2
                                             mixingEnabled:YES] != PdAudioOK )
    {
        NSLog(@"audio controller failed to initialize");
    }

    return YES;
}

- (void)applicationWillResignActive:(UIApplication *)application
{

}

- (void)applicationDidEnterBackground:(UIApplication *)application
{

}

- (void)applicationWillEnterForeground:(UIApplication *)application
{

}

- (void)applicationDidBecomeActive:(UIApplication *)application
{
    // The audio will turn on when the app has loaded.
    self.audioController.active = YES;
}

- (void)applicationWillTerminate:(UIApplication *)application
{
    // The audio will turn off when the app is closed.
    self.audioController.active = NO;
}

@end
```

**Figure 2**. AppDelegate.m

### 2.9 ViewController.h

In ViewController.h (Figure 3), import PdDispatcher.h and declare variable names for PdDispatcher and the PD patch. These names will be referenced in ViewController.m. Also, please notice the curly braces that are also added.

[4] There may be several default comments in AppDelegate.m. Those have already been deleted in Figure 2.

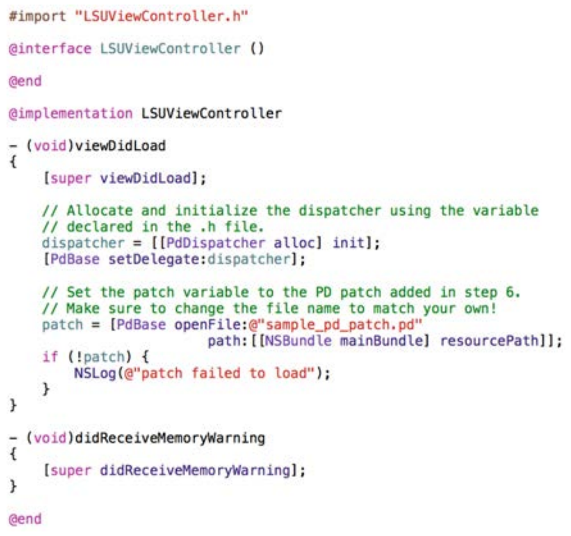


```
#import <UIKit/UIKit.h>
#import "PdDispatcher.h" // Import PdDispatcher.h

@interface LSUViewController : UIViewController {

    // Initialize variable names to use in ViewController.m
    PdDispatcher *dispatcher;
    void *patch;
}

@end
```

**Figure 3**. ViewController.h

## 2.10 ViewController.m

Finally, ViewContoller.m (Figure 4) is where we initialize the dispatcher and set our variables.



```
#import "LSUViewController.h"

@interface LSUViewController ()

@end

@implementation LSUViewController

- (void)viewDidLoad
{
    [super viewDidLoad];

    // Allocate and initialize the dispatcher using the variable
    // declared in the .h file.
    dispatcher = [[PdDispatcher alloc] init];
    [PdBase setDelegate:dispatcher];

    // Set the patch variable to the PD patch added in step 6.
    // Make sure to change the file name to match your own!
    patch = [PdBase openFile:@"sample_pd_patch.pd"
                        path:[[NSBundle mainBundle] resourcePath]];
    if (!patch) {
        NSLog(@"patch failed to load");
    }
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
}

@end
```

**Figure 4**. ViewController.m

## 3. USING STORYBOARDS IN XCODE

Storyboards is an Xcode feature enabling the drag-and-drop creation of working UIs. Once the project is set up with LIBPD, it only requires a few more steps to have a working UI interfacing with the PD patch already added to the project.

When using these instructions to introduce someone new to audio programming languages, prior to this point it would be necessary to cover the rudiments of using PD and creating a patch that will correctly receive messages. Presently, that discussion will be checked, but please consider that simple synthesis techniques with PD can be taught to young adults quickly and efficiently.

Concerning prepping a patch for LIBPD, any patch that could be controlled using a UI will work. Simply replace any connections to UI objects with a typical [receive ...] object. The names assigned to the receive objects will be referenced in Xcode.

### 3.1 Drag-and-Drop UI

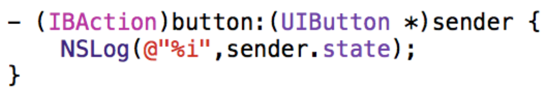
In the Xcode project navigator, select the appropriate ".storyboard" file (likely Main.storyboard). On the bottom right of the storyboard view is the resource menu. For this simple example of how to use storyboards, search for "button" and drag the button option into the graphical view of the iOS UI.

Now, look at the icons in the top right corner, and locate and turn on the "Assistant editor." It looks a bit like the front of a tuxedo, and it will split the center view into two windows. One should remain the storyboard view, but the other should show the ViewController.m file. If it doesn't, simply locate it using the drop down menus at the top of the new window. Now, control-click (not right-click!) on the button just placed in the storyboard view, and drag a connection to the ViewController.m file. Drop the connection anywhere between the @implementation and @end tags, but not inside an existing method (Figure 7).

In the pop-up menu that will appear, name the button, change the type to UIButton (using the drop down menu), and change the event to Touch Down. Then select "connect." This will automatically create code for interacting with that button via storyboards. When hovering over the tiny dot next to the new code, the button in the storyboard view should highlight and confirm the connection. Similar processes can be used to create sliders, toggles, and even touch-screen gestures.

### 3.2 Sending Messages to PD

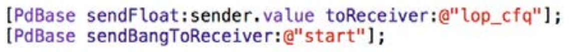
The final stage is to enable the button to send to messages to PD. As it is, the simplest method of obtaining data when the button is clicked is via the "sender.state" property. Update the project with the code in Figure 5.



```
- (IBAction)button:(UIButton *)sender {
    NSLog(@"%i",sender.state);
}
```

**Figure 5**. NSLog logs to the console.

In the top left corner, locate the start/stop icons. They are followed by the current project name and a drop down list of simulators and devices.[5] Choose the appropriate simulator and hit the play button (technically the "Build and run..." button). The iOS simulator will launch, and the UI button is clickable! Clicking down will log the "sender.state" value in the console, which happens to be a 1. LIBPD enables us to send the UI values from iOS directly to a receive object.

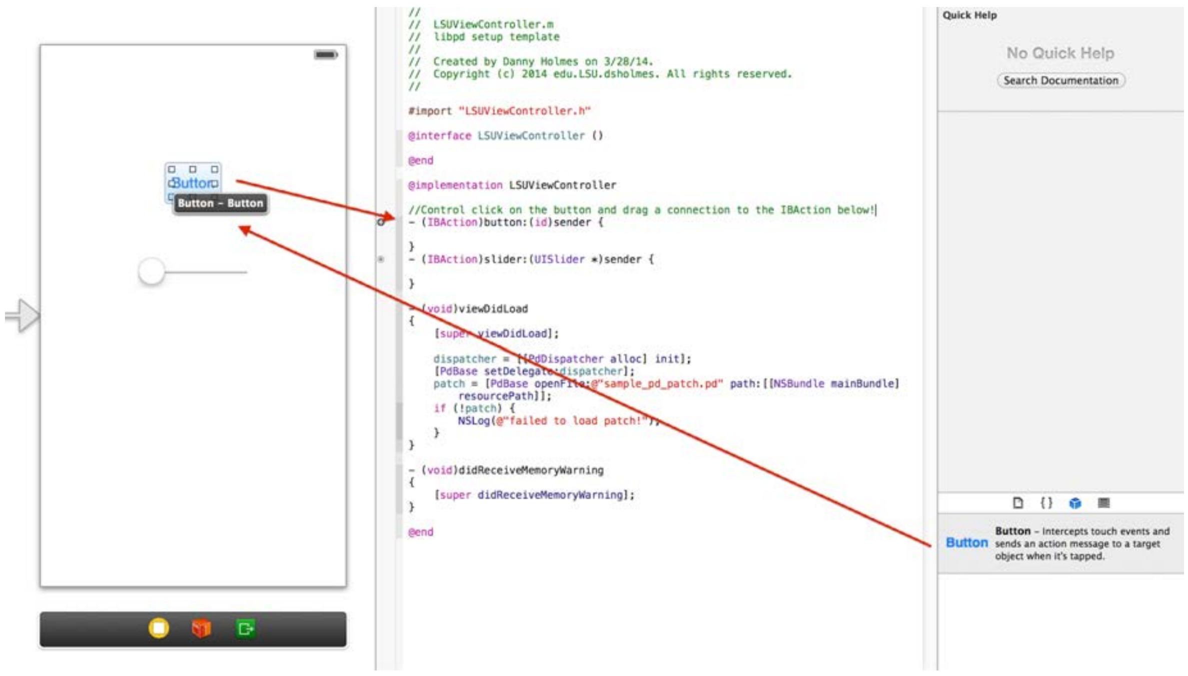


```
[PdBase sendFloat:sender.value toReceiver:@"lop_cfq"];
[PdBase sendBangToReceiver:@"start"];
```

**Figure 6**. Examples of using PdBase.

Figure 6 shows examples of the last bit of code necessary to actually send the message to the PD patch. Either line would simply be added inside the button method's curly braces along with the NSLog line.

[5] It may be necessary to download the appropriate simulator. Each major device has its own standalone simulator. Packages can be found under File - Preferences - Downloads.



```
//
//  LSUViewController.m
//  libpd setup template
//
//  Created by Danny Holmes on 3/28/14.
//  Copyright (c) 2014 edu.LSU.dsholmes. All rights reserved.
//

#import "LSUViewController.h"

@interface LSUViewController ()

@end

@implementation LSUViewController

//Control click on the button and drag a connection to the IBAction below!
- (IBAction)button:(id)sender {

}
- (IBAction)slider:(UISlider *)sender {

}

- (void)viewDidLoad
{
    [super viewDidLoad];

    dispatcher = [[PdDispatcher alloc] init];
    [PdBase setDelegate:dispatcher];
    patch = [PdBase openFile:@"sample_pd_patch.pd" path:[[NSBundle mainBundle]
        resourcePath]];
    if (!patch) {
        NSLog(@"failed to load patch!");
    }
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
}

@end
```

Button – Intercepts touch events and sends an action message to a target object when it's tapped.

**Figure 7**. Use Storyboard's Assistant editor to automatically create code for UI elements.

In the first PdBase example, it would be necessary to change "sender.value" to "sender.state" in order to receive the correct data from the button. It would also be necessary to change the receiver to the appropriate name. This would then send a 1 to that receiver each time the button is pressed down, and dealing with that data would be up to PD at that point. Similarly, the second example would simply send a bang instead.

## 4. USING THIS PROCESS TO MAKE AN APP

This particular method of implementing LIBPD serves two purposes: to better enable rapid-prototyping musical iOS apps, and to make the entire package more accessible to anyone not familiar with objective-c. Yet, these instructions alone will not resolve into a terribly useful application in either case. No matter how simple or complex, it is beneficial to create the sounds in PD prior to beginning work with Xcode. Once the UI controlled data types are known and receive objects are in place, a competent UI can be designed. Then, this process bridges the gap from concept to working app.

## 5. CONCLUSIONS

This entire process can actually be accomplished in minutes. This is an attractive prospect given how accessible mobile devices already are. Many teenagers and young adults already have phones in their pockets, and they already know how basic interactions work. The stream-lined process described here gives students of all ages tools that can extend their existing knowledge of mobile interactions into the realms of designing and creating musical mobile apps. Reducing time commitments to setup and boilerplate code enables us to focus more attention on instrument design and musicality [3]. While cost, availability, and computing power increase the general accessibility of mobile devices, tools such as these increase the general accessibility of becoming a mobile developer.

**Acknowledgments**

Special thanks are given to Jesse Allison for creating and running LSU's Musical App-a-thons (along with the many other reasons he should be thanked). Thanks to EMDM Academy for being the platform from which we launch many different ventures into music education reform, especially pertaining to electronic music and music technology. Also, thanks to Best Buy Community Grants for providing the initial funding for EMDM Academy's outreach efforts.

## 6. REFERENCES

[1] M. Firtman, *Programming the Mobile Web, Second Edition*. Sebastopol, CA: O'Reilly Media, Inc., 2014.

[2] P. Brinkman, *Making Musical Apps*. Sebastopol, CA: O'Reilly Media, Inc., 2012.

[3] S. P. Anderson, *Seductive Interaction Design: Creating Playful, Fun, and Effective User Experiences*. Berkeley, CA: PoetPainter, LLC., 2011.