

Getting started with Phidgets in iOS

Environment and Libraries

The Phidget21 iOS library supports devices with a iOS version of 3.0 or later. However, it is recommended that your device have the latest iOS version that is supported. We need to set up the proper environment and get the necessary files off the Phidgets website. Visit the drivers section at www.phidgets.com and get the latest:

- Phidget21 library(Windows, Max OS X, or Linux)
- Phidget21 iOS library

You will need the Phidget21 library and the iOS library to use and program with Phidgets. As iOS devices do not have USB ports, support for Phidgets is made available through the WebService. The system that is hosting the Phidgets will need to have the Phidget21 library installed. After the framework is installed, the WebService must be started. For more information on the WebService, please see the "Working with Phidget WebService" guide from the programming section at www.phidgets.com. The computer that is used for iOS development will need the Phidget21 iOS library. We also recommend that you download the following reference materials from the programming section at www.phidgets.com:

- C API Manual
- Programming Manual
- The Product Manual for your device
- Example Programs written for iOS

The C API manual lists calls and events for every type of Phidget and can be used as a reference. You can find a high level discussion about programming with Phidgets in general in the Programming Manual. The Product manual for your device also contains an API section that describes limitations, defaults, and implementation details specific to your Phidget. You may want to have these manuals open while working through these instructions.

Setting up a Phidgets Project

The Phidget examples were written in Objective-C and Xcode 4.0, and this tutorial assumes their use. Other versions of Xcode should work as well and would be set up in a similar manner. In Xcode:

- Generate a new iOS Windows-based Application project with a descriptive name such as PhidgetTest.
- Set up the Provisioning Profiles and Code Signing settings, if necessary.
- Download the Phidget21 iOS library, extract the file, and move the iphoneos and the iphonesimulator folders, as well as the phidget21.h into the same directory as the the .xcodeproj file.
- In xCode, open up the Project Settings > Build Settings.
- In Linking > Other Linker Flags, follow these steps for both Debug, and Release:

» Select "Any iOS Simulator SDK" and enter:

```
$(SRCROOT)/iphonesimulator/libPhidget21.a
```

» Select "Any iOS SDK" and enter:

```
$(SRCROOT)/iphoneos/libPhidget21.a
```

- In Search Path > Header Search Path, enter:

```
$(SRCROOT)
```

- In the header file, add a reference to phidget21.h:

```
#import "phidget21.h"
```

- A text field will be used for the purpose of capturing output. Open MainMenu.nib to bring up the Interface Builder. Drag a text field from the Library to the Window.
- Add a text field outlet in the header file. For example,

```
@interface PhidgetTestAppDelegate : NSObject <UIApplicationDelegate>{
    IBOutlet UITextField *sensorValueTxt;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet UITextField *sensorValueTxt;
@end
```

- In the implementation file, add the following line to the implementation section:

```
@synthesize sensorValueTxt;
```

- Also, be sure release the object in the dealloc method

```
[sensorValueTxt release];
```

- Connect the PhidgetTestAppDelegate class instance to sensorValueTxt.

The project now has access to Phidgets and we are ready to begin coding.

Please note that the iPhone library also contains a sample skeleton xCode project for iOS. This project contain the necessary project settings for Phidgets development. Alternatively, you can use it to start developing with Phidgets.

Coding For Your Phidget

A Phidget object will need to be declared. For example, we can declare a PhidgetInterfaceKit in the .h header file with:

```
CPhidgetInterfaceKitHandle ifkit
```

The object name for any type of Phidget is listed in the API manual. Every type of Phidget also inherits functionality from the Phidget base class.

Connecting to the Phidget

Next, the Phidget object needs to be initialized and the program needs to try and connect to the Phidget over the Webservice through a call to `openRemoteIP()` or `openRemote()`. These calls will tell the program to continuously try to connect to a Phidget, based on the parameters given, even trying to reconnect if it gets disconnected. This means that simply calling `open` does not guarantee you can use the Phidget immediately. We can handle this by using event driven programming and tracking the `AttachEvents` and `DetachEvents`, or by calling `waitForAttachment`. `waitForAttachment` will block indefinitely until a connection is made to the Phidget, or an optional timeout is exceeded. For example, we can connect to a `PhidgetInterfaceKit` in the `.m` implementation file with:

```
@implementation PhidgetTest
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    CPhidgetInterfaceKit_create(&ifkit);
    CPhidget_openRemoteIP((CPhidgetHandle)ifkit, -1, "192.168.2.163", 5001, NULL);
}
@end
```

The `openRemoteIP()` and `openRemote()` calls can be used with parameters to try and get the first device it can find over the network, and open it based on a serial number. The API manual provides more information on `openRemoteIP()` and `openRemote()`. One important thing to remember is that when working with Phidgets, a local connection will reserve the device until closed. This prevents any other instances from retrieving data from the Phidget, including other programs. The one connection per device limit does not apply when exclusively using the Phidget Webservice.

At the end of your program, don't forget to call `close` to free any locks on the Phidget.

```
- (void)applicationWillTerminate:(UIApplication *)application
{
    CPhidget_close((CPhidgetHandle)ifkit);
    CPhidget_delete((CPhidgetHandle)ifkit);
}
```

Event Driven Programming

We recommend the use of event driven programming when working with Phidgets. We can hook an event handler at loading with the following code:

```
CPhidgetInterfaceKit_set_OnSensorChange_Handler(ifkit, gotSensorChange, self);
```

Next, the callback method needs to be set up before it can be used. For example,

```
int gotSensorChange(CPhidgetInterfaceKitHandle phid, void *context, int ind, int val)
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    [(id)context performSelectorOnMainThread:@selector(SensorChange:)
withObject:[NSArray arrayWithObjects:[NSNumber numberWithInt:ind], [NSNumber
```

```

numberWithInt:val], nil] waitUntilDone:NO];
    [pool release];
    return 0;
}

```

Above, the `SensorChange` method is invoked on the main thread. Event data is stored in a `NSArray`, which in turn is sent as a single argument to the `SensorChange` method. The `NSAutoreleasePool` object is created to clean up released objects on the event thread, and is released at the end of the method.

The `SensorChange` method is defined as follows:

```

- (void)SensorChange:(NSArray *)sensorChangeData
{
    int sensorIndex, sensorValue;
    sensorIndex = [[sensorChangeData objectAtIndex:0] intValue];
    sensorValue = [[sensorChangeData objectAtIndex:1] intValue];
    _sensorValueTxt.text = [NSString stringWithFormat:@"Sensor: %d, Value: %d",
sensorIndex, sensorValue]];
}

```

With this function, the code inside `SensorChange` will get executed every time the `PhidgetInterfaceKit` reports a change on one of its analog inputs. Some events such as `Attach` and `Detach` belong to the base `Phidget` object and thus are common to all types of `Phidgets`. Please refer to the API manual and the iOS examples for a list of events and their usage.

Working directly with the Phidget

Some values can be read and sent directly to the `Phidget`, simply use the C API functions such as `CPhidgetInterfaceKit_getSensorValue` for `PhidgetInterfaceKits`.

```

int sensorIndex = 0;
int sensorValue;
CPhidgetInterfaceKit_getSensorValue(ifkit, sensorIndex, &sensorValue);
_sensorValueTxt.text = [NSString stringWithFormat:@"Sensor: %d, Value: %d",
sensorIndex, sensorValue]];

```

These functions can be used inside a polling loop as an alternative to event driven programming.

Working with multiple Phidgets

Multiple `Phidgets` of the same type can easily be run inside the same program. In our case, it requires another `InterfaceKit` instance to be defined and initialized. The new instance can then be set up, opened and used in the same process as the previous one.

If the application needs to distinguish between the devices, `open` can be called with the serial number of a specific `Phidget`.

Other Phidgets

The design given in this document can also be followed for almost all `Phidgets`. For example, if

you were using a PhidgetRFID instead of an PhidgetInterfaceKit, you would declare an RFID object instead of an InterfaceKit. The methods and events available would change but they can be accessed in a similar manner.