

# Getting started with Phidgets in Cocoa

## Environment and Libraries

First, 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](http://www.phidgets.com) and get the latest:

- Phidget Framework

You will need the Phidget Framework to use and program with Phidgets. We also recommend that you download the following reference materials from the programming section:

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

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 3.2.4, 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 Cocoa project with a descriptive name such as PhidgetTest.
- Add the Phidget21 Framework (Groups & Files -> Frameworks -> Other Frameworks).
- Create a new Objective-C class with a descriptive name. For the purpose of this guide, the class name will be PhidgetInterfaceKit. A header file (.h) as well as an implementation file (.m) will automatically be created.
- Open the header file for editing
- Add a reference to phidget21.h:

```
#import <Phidget21/phidget21.h>
```
- A text field will be used for the purpose of capturing output. Add a text field outlet in the header file. For example,

```
@interface PhidgetInterfaceKit : NSObject{
    IBOutlet NSTextField *sensorValueTxt;
}
@end
```
- In Groups & Files -> Resources, open up MainMenu.nib to bring up the Interface Builder. Drag a text field from the Library to the Window.
- Now, an instance of the PhidgetInterfaceKit class will need to be created. In the Library, drag an

object to the MainMenu.nib Window. Open up the Identity tab of the Inspector for this object and add the PhidgetInterfaceKit class.

- Connect the PhidgetInterfaceKit class instance to the text field.

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

## Coding For Your Phidget

A Phidget object will need to be declared. For example, we can declare a PhidgetInterfaceKit in the .m implementation 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 through a call to `open()`. `Open` 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 PhidgetInterfaceKit
- (void) awakeFromNib
{
    CPhidgetInterfaceKit_create(&ifkit);
    CPhidget_open((CPhidgetHandle)ifkit, -1);
}
@end
```

The different types of `open` can be used with parameters to try and get the first device it can find, `open` based on its serial number, or even `open` across the network. The API manual lists all of the available modes that `open` provides. 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.

```
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 setStringValue:[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 Cocoa 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 sensorValue;
CPhidgetInterfaceKit_getSensorValue(ifkit, 0, &sensorValue);
[sensorValueTxt setintValue: 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 PhidgetInterfaceKit 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.