

Getting started with Phidgets in Ruby (Mac OS X and Linux)

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 and get the latest Phidget driver for your system. You will also need to install the phidgets-ffi gem from RubyGems. More information will be provided in the "Installing the phidgets-ffi gem" section of this document.

We also recommend that you refer to the following reference materials:

- RubyDoc API Documentation(<http://rubydoc.info/gems/phidgets-ffi/frames>)
- Programming Manual(http://phidgets.com/programming_resources.php)
- The Product Manual for your device(<http://phidgets.com>)
- Ruby code samples(<https://github.com/kreynolds/phidgets-ffi/tree/master/examples>)

The RubyDoc API manual contains 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

Installing the phidgets-ffi gem

To install:

```
gem install phidgets-ffi
```

phidgets-ffi is dependent on the ffi(>=1.0.9) gem. If the above command does not automatically install the gem for you, please enter:

```
gem install ffi
```

.

Coding for Your Phidget

Before you can use the Phidget, you must include a reference to the phidgets-ffi gem:

```
require `rubygems`  
require `phidgets-ffi`
```

Afterwards, the Phidget constructor method will need to be called to create the Phidget object. There are two methods of programming with Phidgets in Ruby: with a block and without a block. For the most part, the two methods behave the same, but there are some subtle differences, which will be explained in the next section.

Without a block:

```
ifkit = Phidgets::InterfaceKit.new
```

With a block:

```
Phidgets::InterfaceKit.new do |ifkit|  
  ...  
end
```

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. In the above example, InterfaceKit represents the PhidgetInterfaceKit. For simplicity in explaining the programming concepts, the remainder of this document will use the InterfaceKit object.

Connecting to the Phidget

The program needs to try and connect to the Phidget. Options can be added to the InterfaceKit constructor to connect to the first Phidget it finds, based on its serial number, label, or even connect across the network. Rubydocs API Documentation lists all of the available modes that the constructor provides.

For example, the following will try to connect to the first Phidget it finds:

```
ifkit = Phidgets::InterfaceKit.new
```

The following will try to connect to a Phidget over the Phidget WebService with a serial number of 99999, and a server id of `myserver`:

```
options = {:serial_number => 99999, :server_id => `myserver`, :password => nil}  
ifkit = Phidgets::InterfaceKit.new(options)
```

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

Phidgets::InterfaceKit.new will tell the program to continuously try to connect to a Phidget, based on the options given(if any), even trying to reconnect if it gets disconnected.

Non Block Programming

For the non block programming method, simply calling the constructor does not guarantee you can use the Phidget immediately. We can handle this by using event driven programming and tracking the on_attach and on_detach events, or by calling: wait_for_attachment. wait_for_attachment will block indefinitely until a connection is made to the Phidget, or an optional timeout is exceeded.

```
ifkit = Phidgets::InterfaceKit.new
```

```
ifkit.wait_for_attachment 2000 #halt the program for up 2000 milliseconds or until the Phidget is connected
```

Please also remember to call close at the end of the program to free any locks on the Phidget

```
ifkit.close
```

Block Programming

If you are programming inside a block, wait_for_attachment is automatically called. By default, it will halt the program and try to connect to the Phidget for up to 1000 milliseconds. Afterwards, the block will yield. Finally, close is not needed as it is automatically called once the block has yield.

```
options = {:timeout => 2000)
Phidgets::InterfaceKit.new(options) do |ifkit|
  ...
end
```

Event Driven Programming

We recommend the use of event driven programming when working with Phidgets. In Ruby, we can

hook an event handler at loading with the following code:

```
ifkit.on_sensor_change do |device, input, value, obj|
  puts "Sensor #{input.index}'s value has changed to #{value}"
end
```

With this method, the block inside `on_sensor_change` will get executed every time the `PhidgetInterfaceKit` reports a change on one of its analog inputs. Some events such as `on_attach` and `on_detach` belong to the base `Phidget` object and thus are common to all types of `Phidgets`. Please refer to the [RubyDoc API documentation](#) for a full 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 instance members and properties. This is also how you would set properties on the `Phidget` such as the output state or sensor sensitivity. These functions can be used inside a polling loop as an alternative to event driven programming.

```
puts "Serial number: #{ifkit.serial_number}"
puts "Sensor value[0]: #{ifkit.sensors[0].value}"
```

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, the constructor can be called with the serial number or label 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 a `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.

Raw FFI

As an alternative to programming with the method as outlined in this document, you can also program making straight C calls through FFI. Please refer to the files in `<Ruby Gems directory>/phidgets-ffi-x.x.x/lib/phidgets-ffi/ffi` to see a list of available methods, and the C API for usage. There are raw ffi examples in `<Ruby Gems directory>/phidgets-ffi-x.x.x/examples/raw-ffi`.