Getting started with Phidgets in Python

Environment and Libraries

First, we need to set up the proper environment and get the necessary files off the Phidgets web site. Visit the drivers and programming section at www.phidgets.com and get the latest:

- Phidget Framework
- Python Module

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

- Python Examples
- Python API Manual
- Programming Manual
- The Product Manual for your device

The Python 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 and there are working samples for each type of Phidget in the Python Examples. 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 and examples open while working through these instructions.

Setting up a Phidgets Project

The Phidget examples were written in Python 3.0 and this tutorial assumes its use. However, they should still be compatible with Python 2.6. To run the examples using Python 2.5, you will need to modify the example code in the exception handling to read "except RuntimeError, e:", instead of "except RuntimeError as e:".

Please ensure you have extracted the "Phidgets" directory from the Python Module into your project directory or into the lib\site-packages\ directory in your python install.

Coding For Your Phidget

Before you can use the Phidget, you must include a reference in the code to the library. In Python:

```
from Phidgets.PhidgetException import *
from Phidgets.Events.Events import *
from Phidgets.Devices.InterfaceKit import *
```

Afterwards, the Phidget object will need to be declared and then initialized. For example, we can declare a PhidgetInterfaceKit with:

```
try:
    interfaceKit = InterfaceKit()
except RuntimeError as e:
    print("Runtime Error: %s" % e.message)
```

The initialization of the Phidget as well as calls using the Phidget object should be surrounded by a try catch block to handle any errors thrown by the library. Calls to the Phidget object will throw a PhidgetException on an error.

```
try:
    #Your program Code here
except PhidgetException as e:
    print ("Phidget Exception %i: %s" % (e.code, e.detail))
    exit(1)
```

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

The program can try to connect to the Phidget through an open call. Open will 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 checking isAttached().

```
interfaceKit.openPhidget()
interfaceKit.waitForAttach(10000)
print ("%d attached!" % (interfaceKit.getSerialNum()))
```

The parameters can be used to open the first Phidget of a type 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.

```
interfaceKit.closePhidget()
```

Event Driven Programming

We recommend the use of event driven programming when working with Phidgets. In Python, we hook an event handler by defining the callback function and then calling a set handler function on it.

```
def interfaceKitSensorChanged(e):
    print ("Sensor %i: %i" % (e.index, e.value))
    return 0

interfaceKit.setOnSensorChangeHandler(interfaceKitSensorChanged)
```

With this, the code inside interfaceKitSensorChanged will get executed every time the PhidgetInterfaceKit reports a change on one of its analog inputs. The values from the report can be accessed from the PhidgetDataEvent object properties.

Certain 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 for a full list of events and their usage.

Working directly with the Phidget

Some values can be directly read and set on the Phidget and used as an alternative to event driven programming. Simply use the instance properties or call member functions such as getSensorValue(index) or setOutputState(index, state) for PhidgetInterfaceKits.

```
interfaceKit.setOutputState(0, 1)
```

Working with multiple Phidgets

Multiple Phidgets of the same type can easily be run inside the same program. In our case, it requires another instance of a PhidgetInterfaceKit 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 a PhidgetInterfacekit, you would declare an RFID instead of an InterfaceKit. The functions and events available would change but they can be accessed in a similar manner.