

Getting started with Phidgets in Java

Environment and Libraries

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

- Phidget Framework
- Phidget21.jar

You will need the Phidget21.jar from the Java section to program with Phidgets, and the Phidget Framework to use them. We also recommend that you download from the programming section the following reference materials:

- Javadoc API Manual
- Programming Manual
- The Product Manual for your device
- Example Programs written in Java

The Javadoc 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.

Setting up a Phidgets Project

The Phidget examples were written using NetBeans and this tutorial assumes its use. Other environments such as Eclipse work as well and would be set up in a similar manner. For Java command line compilers, include the phidget21.jar during compilation. For example, you can use “javac -classpath phidget21.jar MyPhidgetProgram.java” from the Java SDK.

In NetBeans, the Phidget .jar library can be added to the project from the project explorer window. Simply right click the “Libraries” item in the project explorer and then select “Add JAR/Folder”. Navigate to the location where the phidget21.jar was extracted and then add it to the project. You are now ready to begin coding with Phidgets.

Coding For Your Phidget

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

```
import com.phidgets.*;
import com.phidgets.event.*;
```

Now in the main body of code, the Phidget object will need to be declared. For example, we can declare a PhidgetInterfaceKit with:

```
static InterfaceKitPhidget ik;
```

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.

All Phidgets can throw PhidgetExceptions if something unexpected happens during operation. Make sure to catch or declare the exception, even in a generic way. This is what the main function looks like where we initialize the Phidget object without any constructors:

```
public static final void main(String args[]) throws Exception
{
    ik = new InterfaceKitPhidget();
}
```

Connecting to the Phidget

Next, the program needs to try and connect to the Phidget through an open call. The 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.

```
ik.openAny();
ik.waitForAttachment();
```

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.

```
ik.close();
ik = null;
```

Event Driven Programming

We recommend the use of event driven programming when working with Phidgets. In Java, we hook an event handler with the following code:

```
ik.addSensorChangeListener(new SensorChangeListener()
{
    public void sensorChanged(SensorChangeEvent se)
    {
        //Insert your code here
        System.out.println(se.getValue());
    }
});
```

With this method, the code inside `sensorChanged` will get executed every time the `InterfaceKit` reports a change on one of its analog inputs. The items from the event, such as the index or reported value, can be accessed from the `SensorChangeEvent` object properties.

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 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`. Simply use the instance's properties or call member functions such as `getSensorValue(int index)` or `setOutputState(int index, boolean newVal)` for `InterfaceKits`. These methods 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 `InterfaceKit`, you would declare an `RFIDPhidget` instead of an `InterfaceKitPhidget`. The methods and events available would change but they can be accessed in a similar manner.

Compiling a .jar File

Finally, when the project is completed it is recommend to compile the project as a `.jar`. This will reduce the number of extra files created into a single package that is easier to manage. In `NetBeans` a `.jar` file is automatically created during compilation. Under the command line, you can use the `jar` utility from the `Java SDK` to package the `.class` files. For example: `"jar -cf MyProgram.jar source.class source$1.class source$2.class"`.

You will also need to provide a `Manifest` file for the `jar` program to indicate the program entry point via the `Main-Class` and the `Class-Path` which will point to the `phidget21.jar`.

Simply create a file, for example MyProgram.mf, and enter the following lines:

Manifest-Version: 1.0

Class-Path: phidget21.jar

Main-Class: MyProgram

Now run the jar utility to package the files.

For example: "jar -cmf MyProgram.mf

MyProgram.jar source.class source\$1.class source\$2.class".