

The Phidgets Manual

Index

Section 1 – Getting Started with Phidgets	Page 3
Section 2 – Creating a project	Page 4
Section 3 – Programming Phidgets	Page 7
Section 3.1 – Phidget Interface Kit	Page 7
Section 3.2 – Phidget RFID	Page 15
Section 3.3 – Phidget Encoder	Page 23
Section 3.4 – Phidget Linear Touch	Page 28
Section 3.5 – Phidget Text LCD	Page 29
Section 3.6– Phidget Touch Rotation/Analog Sensors	Page 40

Section 1. Getting Started with Phidgets

What are Phidgets?

Phidgets are a user-friendly system available for controlling and sensing the environment from your computer. You need no hardware knowledge or experience in order to use them and you can include things like switches, sensors, and low-powered output devices such as LEDs into their projects. It is just a matter of plugging the off the shelf devices into the Interface Kit, which in turn is plugged into the USB port on your computer. After that, you can use the simple to program Phidgets software libraries to access these devices.

Install the Hardware

1. Connect one end of a USB cable to the phidget and the other end to the computer.
2. If the phidget requires power make sure to plug it to a power source. See you specific phidget for more information.

Install the software

Assuming that you will use the Eclipse SDK to develop your Java application for programming the phidgets the first step is

1. Go to www.phidget.com and click on downloads.
2. Click on the link for [Phidget21 Downloads](#)
3. Download the file *PHIDGET.MSI*
4. After downloading, install the *PHIDGET.MSI*. This file allows your OS to easily interact with the Phidgets.
5. Download [Java JNI Library - Phidget21](#) to a known location. This is a .jar file that will be used later for application development.
6. **Optional:** Download the Examples.zip
7. **Optional:** Download the JavaDoc.zip

These files can be found in the following links:

Phidget.MSI

<http://www.phidgets.com/modules.php?op=modload&name=Downloads&file=index&req=getit&lid=27>

Java JNI Library (.jar file)

<http://www.phidgets.com/modules.php?op=modload&name=Downloads&file=index&req=getit&lid=28>

Examples

<http://www.phidgets.com/modules.php?op=modload&name=Downloads&file=index&req=getit&lid=24>

JavaDoc

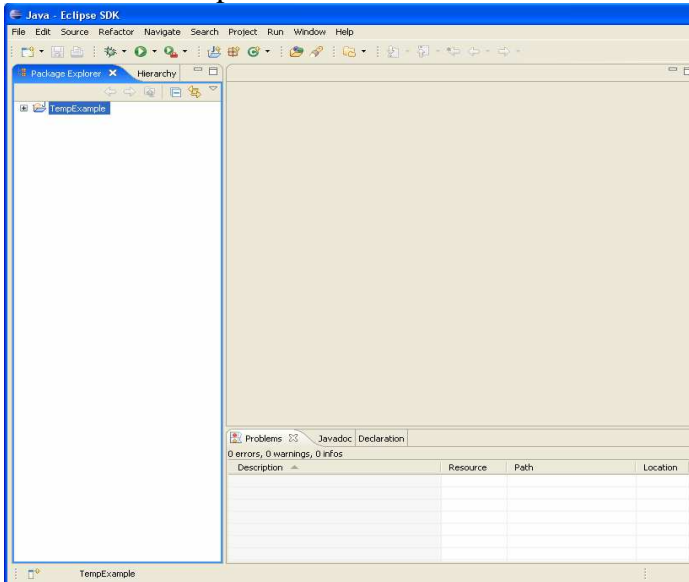
<http://www.phidgets.com/documentation/JavaDoc.zip>

Section 2. Creating a Project

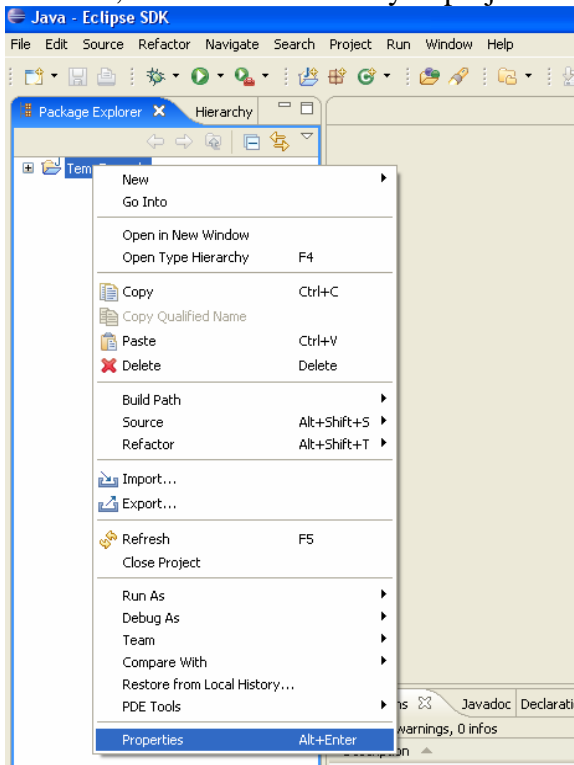
How to create a Project to Program the Phidgets

We will show you how to create a Project that will make use of the phidgets in Eclipse.

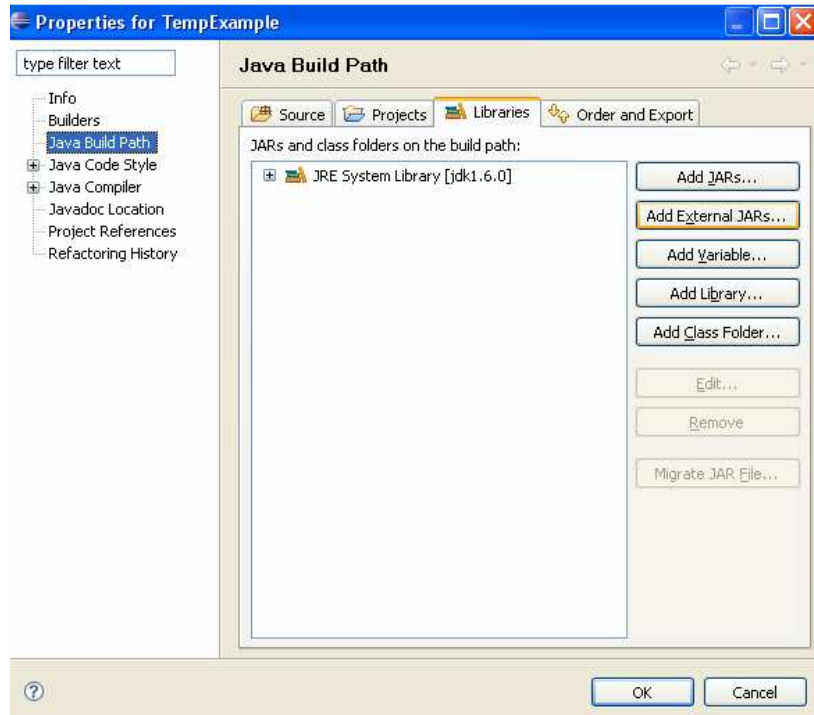
1. Open Eclipse and go to File → New → Project and select Java Project
2. Click Next and give an appropriate name to the project and click Finish. You should end up with a screen that looks like this:



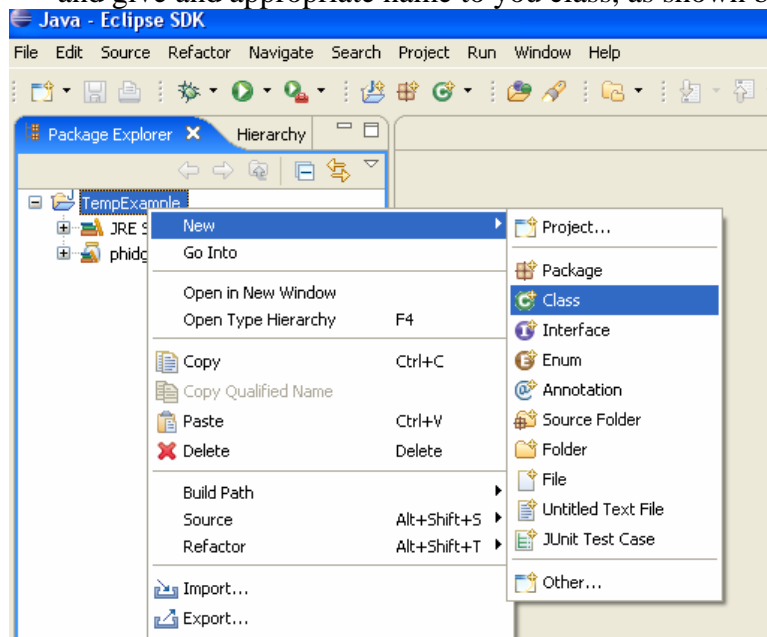
3. Now, RIGHT click under you project name and click on Properties



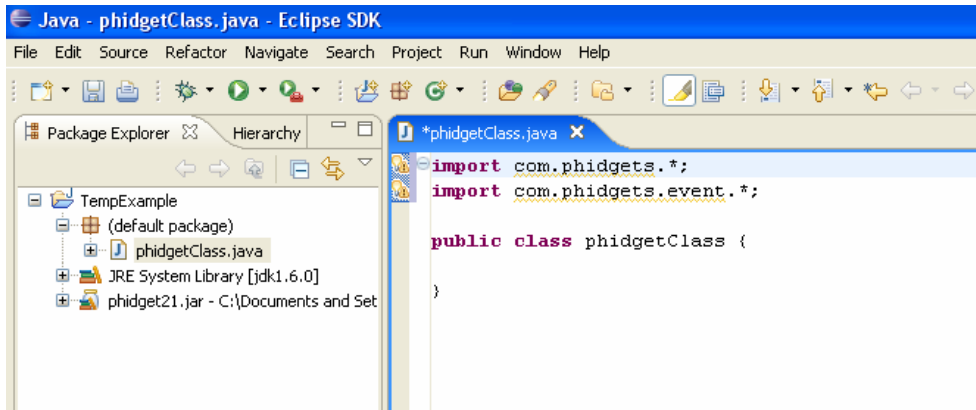
4. On the menu to the left, click on Java Build Path
5. On the Tab Menu click on Libraries
6. You will see several buttons to your right, click on the one that says Add External JARs. You should be in a screen like this one:



7. Now locate the file called phidget21.jar. At this point you are ready to start programming phidgets.
8. To program any phidget, you need to create a class and import two important packages. To do this Right Click on your project name and go to New → Class and give an appropriate name to your class, as shown below



9. In your class make sure that at the beginning you import the following packages: `com.phidget.*` and `com.phidget.event.*`, as show below



10. Now you are ready to start programming your phidgets. Refer to following sections for information on your specific phidget.

Section 3. Programming Phidgets

Section 3.1 Phidget Interface Kit

Phidget Interface Kit 8/8/8

Description and Features:

The PhidgetInterfaceKit 8/8/8 board is a very versatile Phidget, designed to be the core of a wide variety of projects. It is a USB based controller with:

- 8 Analog Inputs
- 8 Digital Inputs
- 8 Digital Outputs
- 1 USB input port
- 2 USB output ports

An external power supply is required only to operate the USB output ports. The PhidgetInterfaceKit 8/8/8 board can be controlled from Windows, Linux, and Mac OS X. High-level programming interfaces are available for Visual Basic, C, C++, Flash, .NET, Java, LabVIEW, etc.



What Can the PhidgetInterfaceKit 8/8/8 Do?

Analog inputs can be used to measure continuous quantities, such as temperature, humidity, position, pressure, etc. There are many plug and play sensors in the Phidgets product line that require no assembly. In addition, any sensor that returns a signal between 0 and 5 volts can be easily interfaced.

Digital inputs can be used to convey the state of push buttons, limit switches, relays.

Digital outputs can be used to drive LEDs, solid state relays, transistors; in fact, anything that will accept a CMOS signal.

The two USB output ports can be used to add more Phidgets to your project without using more USB ports on your PC. Phidgets are modular devices; if there is something that the PhidgetInterfaceKit 8/8/8 cannot do – for example motor control or LED dimming – add another Phidget designed for that purpose.

Programming a PhidgetInterfaceKit 8/8/8

In order to program a PhidgetInterfaceKit create a Java project as described in section 2. Next is a sample code that describes some of the capabilities of the PhidgetInterfaceKit. You can also find more details in the Examples file and in the JavaDocs described in the previous sections:

```

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

public class phidgetClass
{
    public static final void main(String args[] throws Exception {
        InterfaceKitPhidget ik;

        System.out.println(Phidget.getLibraryVersion());

        ik = new InterfaceKitPhidget();
        ik.addAttachListener(new AttachListener() {
            public void attached(AttachEvent ae) {
                System.out.println("attachment of " + ae);
            }
        });
        ik.addDetachListener(new DetachListener() {
            public void detached(DetachEvent ae) {
                System.out.println("detachment of " + ae);
            }
        });
        ik.addErrorListener(new ErrorListener() {
            public void error(ErrorEvent ee) {
                System.out.println("error event for " + ee);
            }
        });
        ik.addInputChangeListener(new InputChangeListener() {
            public void inputChanged(InputChangeEvent oe) {
                System.out.println(oe);
            }
        });
        ik.addOutputChangeListener(new OutputChangeListener() {
            public void outputChanged(OutputChangeEvent oe) {
                System.out.println(oe);
            }
        });
        ik.addSensorChangeListener(new SensorChangeListener() {
            public void sensorChanged(SensorChangeEvent se) {
                System.out.println(se);
            }
        });

        ik.openAny();
        System.out.println("waiting for InterfaceKit attachment...");
        ik.waitForAttachment();

        System.out.println(ik.getDeviceName());

        System.in.read();
        ik.close();
        ik = null;
        System.out.println(" ok");
        if (false) {
            System.out.println("wait for finalization...");
            System.gc();
        }
    }
}

```

Detailed Explanation

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

These lines import the necessary packages in order to be able to program the phidgets.

```
InterfaceKitPhidget ik;
System.out.println(Phidget.getLibraryVersion());
ik = new InterfaceKitPhidget();
```

These lines create a new object of type InterfaceKit that will represent the phidget in your program. All the operation for the phidgets will be carried over this object.

```
ik.addAttachListener(new AttachListener() {
    public void attached(AttachEvent ae) {
        System.out.println("attachment of " + ae);
    }
});
ik.addDetachListener(new DetachListener() {
    public void detached(DetachEvent ae) {
        System.out.println("detachment of " + ae);
    }
});
ik.addErrorListener(new ErrorListener() {
    public void error(ErrorEvent ee) {
        System.out.println("error event for " + ee);
    }
});
ik.addInputChangeListener(new InputChangeListener() {
    public void inputChanged(InputChangeEvent oe) {
        System.out.println(oe);
    }
});
ik.addOutputChangeListener(new OutputChangeListener() {
    public void outputChanged(OutputChangeEvent oe) {
        System.out.println(oe);
    }
});
ik.addSensorChangeListener(new SensorChangeListener() {
    public void sensorChanged(SensorChangeEvent se) {
        System.out.println(se);
    }
});
```

This lines attaches listeners to the InterfaceKit. A listener will allow you to read a sensor measure given some event. The names are self-explanatory, for example the `ik.addAttachListener`, given the event that an InterfaceKit gets attached. The code below it will be executed and the variable “ae” will give you the sensor reading. `ik.addDetachListener`, given the event that an InterfaceKit gets detached, the code below it will be executed and the variable ae contain the sensor reading and so on.

```
ik.openAny();
System.out.println("waiting for InterfaceKit attachment...");
ik.waitForAttachment();

...
System.in.read();
```

Ik.OpenAny() will allow the program to interact with any phidget connected to the computer and the System.read() will allow to read the content of the listeners.

This is a sample program to test the basic capabilities of the InterfaceKit.
This is the list of *all the available methods* that you can use for programming the InterfaceKit in JavaDoc format:

Constructor Detail

InterfaceKitPhidget

```
public InterfaceKitPhidget()  
    throws PhidgetException
```

Class Constructor. Calling this opens a connection to the phidget21 C library creates an internal handle for this Phidget, ready to call open on.

Throws:

[PhidgetException](#) - If there was a problem connecting to phidget21 or creating the internal handle.

Method Detail

getOutputCount

```
public int getOutputCount()  
    throws PhidgetException
```

Returns the number of digital outputs on this Interface Kit. Not all interface kits have the same number of digital outputs, and some don't have any digital outputs at all.

Returns:

Number of digital outputs

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

getInputCount

```
public int getInputCount()  
    throws PhidgetException
```

Returns the number of digital inputs on this Interface Kit. Not all interface kits have the same number of digital inputs, and some don't have any digital inputs at all.

Returns:

Number of digital inputs

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

getSensorCount

```
public int getSensorCount()  
    throws PhidgetException
```

Returns the number of analog inputs on the Interface Kit. Not all interface kits have the same number of analog inputs, and some don't have any analog inputs at all.

Returns:

Number of analog inputs

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

getInputState

```
public boolean getInputState(int index)
    throws PhidgetException
```

Returns the state of a digital input. Digital inputs read True where they are activated and false when they are in their default state.

Be sure to check [getInputCount](#) first if you are unsure as to the number of inputs, so as not to set an Index that is out of range.

Parameters:

index - Index of the input

Returns:

State of the input

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is out of range. See [open](#) for information on determining if a device is attached.

getOutputState

```
public boolean getOutputState(int index)
    throws PhidgetException
```

Returns the state of a digital output. Depending on the Phidget, this value may be either the value that you last wrote out to the Phidget, or the value that the Phidget last returned. This is because some Phidgets return their output state and others do not. This means that with some devices, reading the output state of a pin directly after setting it, may not return the value that you just set.

Be sure to check [getOutputCount](#) first if you are unsure as to the number of outputs, so as not to attempt to get an Index that is out of range.

Parameters:

index - Index of the output

Returns:

State of the output

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is out of range. See [open](#) for information on determining if a device is attached.

getSensorValue

```
public int getSensorValue(int index)
    throws PhidgetException
```

Returns the value of an analog input. The analog inputs are where analog sensors are attached on the InterfaceKit 8/8/8. On the Linear and Circular touch sensor Phidgets, analog input 0 represents position on the slider.

The valid range is 0-1000. In the case of a sensor, this value can be converted to an actual sensor value using the formulas provided here: <http://www.phidgets.com/documentation/Sensors.pdf>

Parameters:

index - Index of the sensor

Returns:

Sensor value

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is out of range. See [open](#) for information on determining if a device is attached.

getSensorRawValue

```
public int getSensorRawValue(int index)
    throws PhidgetException
```

Returns the raw value of an analog input. This is a more accurate version of [getSensorValue](#). The valid range is 0-4095. Note however that the analog outputs on the Interface Kit 8/8/8 are only 10-bit values and this value represents an oversampling to 12-bit.

Parameters:

index - Index of the sensor

Returns:

Sensor value

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is out of range. See [open](#) for information on determining if a device is attached.

getSensorChangeTrigger

```
public int getSensorChangeTrigger(int index)
    throws PhidgetException
```

Returns the change trigger for an analog input. This is the amount that an input must change between successive [SensorChangeEvents](#). This is based on the 0-1000 range provided by [getSensorValue](#). This value is by default set to 10 for most Interface Kits with analog inputs.

Parameters:

index - Index of the sensor

Returns:

Trigger value

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is out of range. See [open](#) for information on determining if a device is attached.

setOutputState

```
public void setOutputState(int index,
    boolean newVal)
    throws PhidgetException
```

Sets the state of a digital output. Setting this to true will activate the output, False is the default state.

Parameters:

index - Index of the output

newVal - State to set the output to

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is out of range. See [open](#) for information on determining if a device is attached.

setSensorChangeTrigger

```
public void setSensorChangeTrigger(int index,  
                                   int newVal)  
    throws PhidgetException
```

Sets the change trigger for an analog input. This is the amount that an inputs must change between successive SensorChangeEvents. This is based on the 0-1000 range provided by getSensorValue. This value is by default set to 10 for most Interface Kits with analog inputs.

Parameters:

index - Input
newVal - Value

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is out of range. See [open](#) for information on determining if a device is attached.

addInputChangeListener

```
public final void addInputChangeListener(InputChangeListener l)
```

Adds an input change listener. The input change handler is a method that will be called when an input on this Interface Kit has changed.

There is no limit on the number of input change handlers that can be registered for a particular Phidget.

Parameters:

l - An implementation of the [InputChangeListener](#) interface

removeInputChangeListener

```
public final void removeInputChangeListener(InputChangeListener l)
```

Removes an input change listener. This will remove a previously added input change listener.

addOutputChangeListener

```
public final void addOutputChangeListener(OutputChangeListener l)
```

Adds an output change listener. The output change handler is a method that will be called when an output on this Interface Kit has changed.

There is no limit on the number of output change handlers that can be registered for a particular Phidget.

Parameters:

l - An implementation of the [OutputChangeListener](#) interface

removeOutputChangeListener

```
public final void removeOutputChangeListener(OutputChangeListener l)
```

Removes an output change listener. This will remove a previously added output change listener.

addSensorChangeListener

```
public final void addSensorChangeListener(SensorChangeListener l)
```

Adds a sensor change listener. The sensor change handler is a method that will be called when a sensor on this Interface Kit has changed by at least the [Trigger](#) that has been set for this sensor.

There is no limit on the number of sensor change handlers that can be registered for a particular Phidget.

Parameters:

1 - An implementation of the [SensorChangeListener](#) interface

removeSensorChangeListener

```
public final void removeSensorChangeListener(SensorChangeListener l)
```

Removes a sensor change listener. This will remove a previously added sensor change listener.

Section 3.2 Phidget RFID

1. Installation

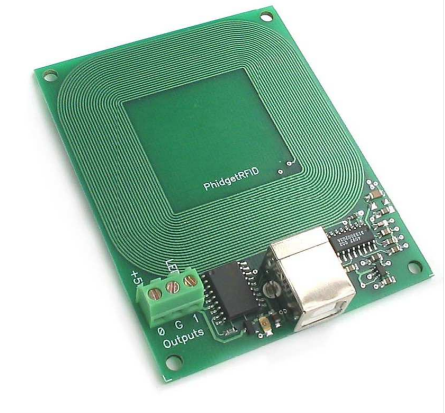
Phidget RFID board

The PhidgetRFID boards are one of our simplest Phidgets to use. They can read any of the three types of RFID tag available from Phidgets that are brought within 3 inches of the reader. When a RFID tag is read, the board returns the unique number contained in the RFID tag.

RFID tags can be attached to things you have lots off and, when read, tell you where that particular thing is.

This device is consist of

- Your PhidgetRFID reader.
- A compatible RFID tag.
- A USB cable.



Programming a PhidgetRFID

In order to program a PhidgetRFID create a Java project as described in the section 2. This is a sample code that describes some of the capabilities of the PhidgetRFID. You can also find more details in the Examples file and in the JavaDocs described in the previous sections:

The example code is in\Phidget21Examples\JavaJNI\RFIDExample.java

Entire Code

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

public class RFIDExample
{
    public static final void main(String args[]) throws Exception {
        RFIDPhidget rfid;

        System.out.println(Phidget.getLibraryVersion());

        rfid = new RFIDPhidget();
        rfid.addAttachListener(new AttachListener() {
            public void attached(AttachEvent ae) {
                System.out.println("attachment of " + ae);
            }
        });
        rfid.addDetachListener(new DetachListener() {
            public void detached(DetachEvent ae) {
                System.out.println("detachment of " + ae);
            }
        });
        rfid.addErrorListener(new ErrorListener() {
            public void error(ErrorEvent ee) {
                System.out.println("error event for " + ee);
            }
        });
    }
}
```

```

    });
    rfid.addTagGainListener(new TagGainListener()
    {
        public void tagGained(TagGainEvent oe)
        {
            System.out.println(oe);
        }
    });
    rfid.addTagLossListener(new TagLossListener()
    {
        public void tagLost(TagLossEvent oe)
        {
            System.out.println(oe);
        }
    });
    rfid.addOutputChangeListener(new OutputChangeListener()
    {
        public void outputChanged(OutputChangeEvent oe)
        {
            System.out.println(oe);
        }
    });

    rfid.openAny();
    System.out.println("waiting for RFID attachment...");
    rfid.waitForAttachment();

    System.out.println("Serial: " + rfid.getSerialNumber());
    System.out.println("Outputs: " + rfid.getOutputCount());

    rfid.setAntennaOn(true);
    rfid.setLEDOOn(true);

    System.out.println("Outputting events.  Input to stop.");
    System.in.read();
    System.out.print("closing...");
    rfid.close();
    rfid = null;
    System.out.println(" ok");
    if (false) {
        System.out.println("wait for finalization...");
        System.gc();
    }
}
}
}

```

Detailed Explanation

```

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

```

It imports the Phidget classes and events from the Java JNI Library (phidget21.jar)

```

RFIDPhidget rfid;

System.out.println(Phidget.getLibraryVersion());

rfid = new RFIDPhidget();

```

It creates an instance of RFIDPhidget as rfid. This class represents a Phidget RFID Reader. All methods to read tags and set outputs on the RFID reader are implemented in this class.

The Phidget RFID reader can read one tag at a time. Both tag and tag loss event handlers are provided, as well as control over the antenna so that multiple readers can exist in close proximity without interference.

```
rfid.addAttachListener(new AttachListener() {
    public void attached(AttachEvent ae) {
        System.out.println("attachment of " + ae);
    }
});
rfid.addDetachListener(new DetachListener() {
    public void detached(DetachEvent ae) {
        System.out.println("detachment of " + ae);
    }
});
```

Attach and Detach are events raised by the Phidget21 library when devices are found or lost. Attach and Detach are generic events covering all Phidgets - it's possible for one Attach event to handle events from many different types of Phidgets. There are also events that are specific to a type of Phidget.

addAttachListener method adds an attach listener. The attach handler is a method that will be called when this Phidget is physically attached to the system, and has gone through its initialization, and so is ready to be used.

AttachListener is an interface that represents an AttachEvent. This event originates from all Phidgets.

Attached method is called with the event data when a new event arrives. AttachEvent is the event data object containing event data.

DetachListener interface represents a DetachEvent. detached method is called with the event data when a new event arrives.

```
rfid.addErrorListener(new ErrorListener() {
    public void error(ErrorEvent ee) {
        System.out.println("error event for " + ee);
    }
});
rfid.addTagGainListener(new TagGainListener() {
    public void tagGained(TagGainEvent oe) {
        System.out.println(oe);
    }
});
rfid.addTagLossListener(new TagLossListener() {
    public void tagLost(TagLossEvent oe)
```

```
    {  
        System.out.println(oe);  
    }  
});
```

Error is event raised when error occurs. TagGain and TagsLoss is raised from Phidget RFID Reader. TagGain event occurs when a tag is placed on a reader. Tag loss events occur when a tag is removed from the RFID reader.

```
rfid.openAny();
```

Open a this Phidget without a serial number. This method is the same as [open](#), except that it specifies no serial number. Therefore, the first available Phidget will be opened. If there are two Phidgets of the same type attached to the system, you should specify a serial number, as there is no guarantee which Phidget will be selected by the call to openAny(). We can use RFID Phidget after calling this method.

```
System.out.println("waiting for RFID attachment...");  
rfid.waitForAttachment();  
System.out.println("Serial: " + rfid.getSerialNumber());  
System.out.println("Outputs: " + rfid.getOutputCount());
```

Waits for this Phidget to become available. This method can be called after open has been called to wait for this Phidget to become available. This is useful because open is asynchronous (and thus returns immediately), and most methods will throw a PhidgetException if they are called before a device is actually ready. This method is synonymous with polling the isAttached method until it returns True, or using the Attach event.

This method blocks indefinitely until the Phidget becomes available. This can be quite some time (forever), if the Phidget is never plugged in.

This method uses the attach handler internally to determine when the Phidget becomes available.

```
rfid.setAntennaOn(true);  
rfid.setLEDOn(true);
```

SetAntennaOn methods sets the state of the antenna. True turns the antenna on, False turns it off. The antenna is by default turned off, and needs to be explicitly activated before tags can be read. Control over the antenna allows multiple readers to be used in close proximity, as multiple readers will interfere with each other if their antenna's are activated simultaneously.

SetLEDon method sets the state of the onboard LED. True turns the LED on, False turns it off. The LED is by default turned off.

```
System.out.println("Outputting events.  Input to stop.");
System.in.read();

System.out.print("closing...");
rfid.close();
rfid = null;
System.out.println(" ok");
if (false) {
    System.out.println("wait for finalization...");
    System.gc();
}
```

If any key input is occurred in the console, this program will end. Before ending, we should close RFID Phidget, because we opened.

Sample Console Output

```
Phidget21 DLL Version 2.1.2 - Built Feb  5 2007 14:34:58
waiting for RFID attachment...
Serial: 11623
Outputs: 2
Outputting events.  Input to stop.
attachment of PhidgetRFID v201 #11623 (attached)
PhidgetRFID v201 #11623 (attached) output 0 changed to false
PhidgetRFID v201 #11623 (attached) output 1 changed to false
PhidgetRFID v201 #11623 (attached) output 2 changed to false
PhidgetRFID v201 #11623 (attached) output 3 changed to true
PhidgetRFID v201 #11623 (attached) output 2 changed to true
PhidgetRFID v201 #11623 (attached) Tag Gained: 0102ac70e5
PhidgetRFID v201 #11623 (attached) Tag lost: 0102ac70e5
PhidgetRFID v201 #11623 (attached) Tag Gained: 0102ac70e5
PhidgetRFID v201 #11623 (attached) Tag lost: 0102ac70e5
PhidgetRFID v201 #11623 (attached) Tag Gained: 0102ac70e5
PhidgetRFID v201 #11623 (attached) Tag lost: 0102ac70e5

closing... ok
```

This is a sample program to test the basic capabilities of the PhidgetRFID. This is the list of *all the available methods* that you can use for programming the PhidgetRFID in JavaDoc format:

Constructor Detail

RFIDPhidget

```
public RFIDPhidget()  
    throws PhidgetException
```

Throws:
[PhidgetException](#)

Method Detail

getOutputCount

```
public int getOutputCount()  
    throws PhidgetException
```

Returns the number of outputs. These are the outputs provided by the terminal block. Older RFID readers do not have these outputs, and this method will return 0.

Returns:
number of outputs

Throws:
[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

getOutputState

```
public boolean getOutputState(int index)  
    throws PhidgetException
```

Returns the state of an output. True indicated activated, False deactivated, which is the default.

Parameters:
index - index of the output

Returns:
state of the output

Throws:
[PhidgetException](#) - If this Phidget is not opened and attached, of the index is out of range. See [open](#) for information on determining if a device is attached.

setOutputState

```
public void setOutputState(int index,  
    boolean state)  
    throws PhidgetException
```

Sets the state of a digital output. True indicated activated, False deactivated, which is the default.

Parameters:
index - index of the output
state - desired state

Throws:
[PhidgetException](#) - If this Phidget is not opened and attached, of the index is out of range. See [open](#) for information on determining if a device is attached.

getAntennaOn

```
public boolean getAntennaOn()  
    throws PhidgetException
```

Returns the state of the antenna. True indicated that the antenna is active, False indicated inactive.

Returns:
state of the antenna

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

setAntennaOn

```
public void setAntennaOn(boolean state)
    throws PhidgetException
```

Sets the state of the antenna. True turns the antenna on, False turns it off. The antenna is by default turned off, and needs to be explicitly activated before tags can be read. Control over the antenna allows multiple readers to be used in close proximity, as multiple readers will interfere with each other if their antenna's are activated simultaneously.

Parameters:

state - new state for the antenna

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

getLEDOn

```
public boolean getLEDOn()
    throws PhidgetException
```

Returns the state of the onboard LED. This LED is by default turned off.

Returns:

state of the LED

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

setLEDOn

```
public void setLEDOn(boolean state)
    throws PhidgetException
```

Sets the state of the onboard LED. True turns the LED on, False turns it off. The LED is by default turned off.

Parameters:

state - new state for the LED

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

getLastTag

```
public java.lang.String getLastTag()
    throws PhidgetException
```

Returns the last tag read. This method will only return a valid tag after a tag has been seen. This method can be used even after a tag has been removed from the reader

Returns:

tag

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

addTagGainListener

```
public final void addTagGainListener(TagGainListener l)
```

Adds a tag gained listener. The tag gained handler is a method that will be called when a new tag is seen by the reader. The event is only fired one time for a new tag, so the tag has to be removed and then replaced before another tag gained event will fire.

There is no limit on the number of tag gained change handlers that can be registered for a particular Phidget.

Parameters:

l - An implementation of the [TagGainListener](#) interface

removeTagGainListener

```
public final void removeTagGainListener(TagGainListener l)
```

addTagLossListener

```
public final void addTagLossListener(TagLossListener l)
```

Adds a tag lost listener. The tag lost handler is a method that will be called when a tag is removed from the reader.

There is no limit on the number of tag lost change handlers that can be registered for a particular Phidget.

Parameters:

l - An implementation of the [TagLossListener](#) interface

removeTagLossListener

```
public final void removeTagLossListener(TagLossListener l)
```

addOutputChangeListener

```
public final void addOutputChangeListener(OutputChangeListener l)
```

Adds an output change listener. The output change handler is a method that will be called when an output has changed.

There is no limit on the number of output change handlers that can be registered for a particular Phidget.

Parameters:

l - An implementation of the [OutputChangeListener](#) interface

removeOutputChangeListener

```
public final void removeOutputChangeListener(OutputChangeListener l)
```

Section 3.3 Tutorial for Encoder

1. Introduction

Phidget Encoder board

It is just a matter of plugging your PhidgetEncoder into the USB port on your computer. After that, you can use the simple to program Phidgets software libraries to access these devices.

The PhidgetEncoder uses a typical two-bit mechanical encoder with a built-in momentary-action pushbutton switch. It returns 80 counts for 360 degrees of rotation. With It you can:

- Detect changes in incremental and absolute position.
- Easily track these changes with respect to time.



What Can the PhidgetEncoder Do?

The PhidgetEncoder is intended to be used as a human interface, not as a device to measure shaft speed. Use it as a volume knob, and you can turn up your volume without limit!

2. Programming a PhidgetEncoder

In order to program a PhidgetEncoder create a Java project as described in the section 2 above. This is a sample code that describes some of the capabilities of the PhidgetEncoder you can also find more details in the Examples file and in the JavaDocs described in the previous sections:

The example code is in ...\\Phidget21Examples\\JavaJNI\\EncoderExample.java

Entire Code

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

public class EncoderExample
{
    public static final void main(String args[]) throws Exception {
        EncoderPhidget enc;

        System.out.println(Phidget.getLibraryVersion());
    }
}
```

```

enc = new EncoderPhidget();
enc.addAttachListener(new AttachListener() {
    public void attached(AttachEvent ae) {
        System.out.println("attachment of " + ae);
    }
});
enc.addDetachListener(new DetachListener() {
    public void detached(DetachEvent ae) {
        System.out.println("detachment of " + ae);
    }
});
enc.addErrorListener(new ErrorListener() {
    public void error(ErrorEvent ee) {
        System.out.println("error event for " + ee);
    }
});
enc.addInputChangeListener(new InputChangeListener() {
    {
        public void inputChanged(InputChangeEvent oe) {
            System.out.println(oe);
        }
    }
});
enc.addEncoderPositionChangeListener(new
EncoderPositionChangeListener()
{
    public void
encoderPositionChanged(EncoderPositionChangeEvent oe)
    {
        System.out.println(oe);
    }
});

enc.openAny();
System.out.println("waiting for Encoder attachment...");
enc.waitForAttachment();

System.out.println("Serial: " + enc.getSerialNumber());
System.out.println("Encoders: " + enc.getEncoderCount());
System.out.println("Inputs: " + enc.getInputCount());

System.out.println("Outputting events. Input to stop.");
System.in.read();
System.out.print("closing...");
enc.close();
enc = null;
System.out.println(" ok");
if (false) {
    System.out.println("wait for finalization...");
    System.gc();
}
}
}

```

Console Output

```
Phidget21 DLL Version 2.1.2 - Built Feb 5 2007 14:34:58
waiting for Encoder attachment...
attachment of PhidgetEncoder v103 #13755 (attached)
Serial: 13755
Encoders: 1
Inputs: 1
Outputting events. Input to stop.
```

This is a sample program to test the basic capabilities of the PhidgetEncoder.

3. Java Doc

This class represents a Phidget Encoder. All methods to read encoder data from an encoder are implemented in this class.

Phidget Encoder boards generally support 1 or more encoders with 0 or more digital inputs. Both high speed optical and low speed mechanical encoders are supported with this API.

Constructor Detail

EncoderPhidget

public **EncoderPhidget**()
throws [PhidgetException](#)

Throws:
[PhidgetException](#)

Method Detail

getEncoderCount

public int **getEncoderCount**()
throws [PhidgetException](#)
Returns number of encoders. All current encoder boards support one encoder.

Returns:
number of encoders
Throws:
[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

getInputCount

public int **getInputCount**()
throws [PhidgetException](#)
Returns number of digital inputs. On the mechanical encoder this refers to the pushbutton. The high speed encoder does not have any digital inputs.

Returns:
number of inputs

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

getEncoderPosition

public int **getEncoderPosition**(int index)
throws [PhidgetException](#)

Returns the position of an encoder. This is an absolute position as calculated since the encoder was plugged in. This value can be reset to anything using [setEncoderPosition](#).

Parameters:

index - index of the encoder

Returns:

position of the encoder

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is out of range. See [open](#) for information on determining if a device is attached.

setEncoderPosition

public void **setEncoderPosition**(int index,
int position)
throws [PhidgetException](#)

Sets the position of a specific encoder. This resets the internal position count for an encoder. This call in no way actually sends information to the device, as an absolute position is maintained only in the library. After this call, position changes from the encoder will use the new value to calculate absolute position as reported by `getEncoderPosition`.

Parameters:

index - index of the encoder

position - new position for this encoder.

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is out of range. See [open](#) for information on determining if a device is attached.

getInputState

public boolean **getInputState**(int index)
throws [PhidgetException](#)

Returns the state of a digital input. On the mechanical encoder this refers to the pushbutton. The high speed encoder does not have any digital inputs. A value of true means that the input is active(the button is pushed).

Parameters:

index - index of the input

Returns:

state of the input

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is out of range. See [open](#) for information on determining if a device is attached.

addEncoderPositionChangeListener

public final void **addEncoderPositionChangeListener**([EncoderPositionChangeListener](#) l)

Adds a position change listener. The position change handler is a method that will be called when the position of an encoder changes. The position change event provides data about how many ticks

have occurred, and how much time has passed since the last position change event, but does not contain an absolute position. This can be obtained from `getEncoderPosition`.

There is no limit on the number of position change handlers that can be registered for a particular Phidget.

Parameters:

1 - An implementation of the [EncoderPositionChangeListener](#) interface

removeEncoderPositionChangeListener

```
public final void removeEncoderPositionChangeListener(EncoderPositionChangeListener l)
```

addInputChangeListener

```
public final void addInputChangeListener(InputChangeListener l)
```

Adds an input change listener. The input change handler is a method that will be called when an input on this Encoder board has changed.

There is no limit on the number of input change handlers that can be registered for a particular Phidget.

Parameters:

1 - An implementation of the [InputChangeListener](#) interface

removeInputChangeListener

```
public final void removeInputChangeListener(InputChangeListener l)
```

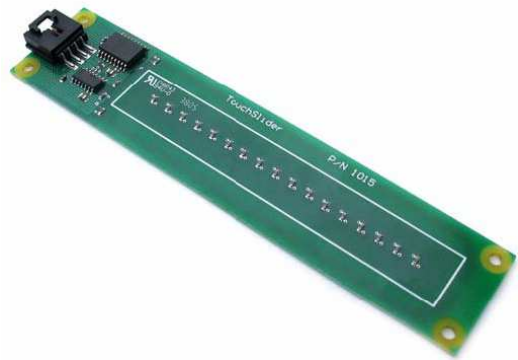
Section 3.4 Tutorial for Linear Touch

1. Introduction

PhidgetLinearTouch

It is just a matter of plugging the Phidget-LinearTouch into the USB port on your computer. After that, you can use the simple to program Phidgets software libraries to access these devices.

The PhidgetLinearTouch changes value when it is touched. It can detect approximately 125 steps. The PhidgetLinearTouch will work through ¼ inch of glass, and appears to the Phidget software libraries as an Interface Kit.



This device is consist of

- Your PhidgetLinearTouch.
- A USB cable.

What Can the PhidgetLinearTouchDo?

Touch sensors like being touched. When you need human beings to input values into your project then getting them to touch a sensor is quick and intuitive. The PhidgetLinearTouch will detect when you place your finger on it, and returns a value depending on where you place your finger. Sliding your finger over the sensor continuously changes the value returned. The PhidgetLinearTouch can be used to set a value; any value.

2. Programming a PhidgetLinearTouch

This part is exactly same as **Tutorial For PhidgetInterfaceKit (Section 3.2)**.

3. Java Doc

This part is exactly same as **Tutorial For PhidgetInterfaceKit (Section 3.2)**.

Section 3.5 Tutorial for Text LCD

1. Introduction

PhidgetTextLCD

A PhidgetTextLCD is a two line by 20-character Liquid Crystal Display (LCD) text screen; green, blue, and white backlit versions are available. In addition, on the back, the PhidgetTextLCD includes a PhidgetInterfaceKit 8/8/8. This Interface Kit is equipped with:

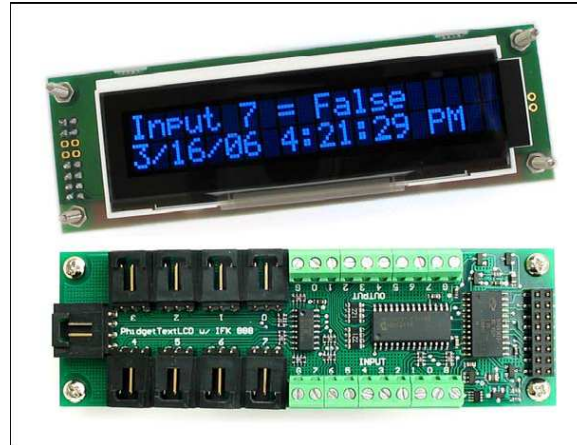
- 8 Digital Inputs
- 8 Analog Inputs
- 8 Digital Outputs

This device is consist of

- Your PhidgetTextLCD.
- A USB cable.

What Can the PhidgetTextLCD Do?

On the LCD you can display results, messages, anything that will fit in two lines of twenty characters. Digital inputs can be used to convey the state of push buttons, limit switches, relays. Any sensor that returns a signal between 0 and 5 volts can be easily interfaced to the Analog Inputs. Digital outputs can be used to drive LCDs, solid-state relays, transistors; anything that will accept a CMOS signal.



2. Programming a PhidgetTextLCD

In order to program a PhidgetTextLCD create a Java project as described in the section 2. This is a sample code that describes some of the capabilities of the PhidgetTextLCD you can also find more details in the Examples file and in the JavaDocs described in the previous sections:

The example code is in ...\\Phidget21Examples\\JavaJNI\\TextLCDExample.java

Entire Code

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

public class TextLCDExample
{
    public static final void main(String args[]) throws Exception {
        TextLCDPhidget lcd;

        System.out.println(Phidget.getLibraryVersion());

        lcd = new TextLCDPhidget();
        lcd.attachListener(new AttachListener() {
            public void attached(AttachEvent ae) {
                System.out.println("attachment of " + ae);
            }
        });
        lcd.detachListener(new DetachListener() {
            public void detached(DetachEvent ae) {
                System.out.println("detachment of " + ae);
            }
        });
        lcd.errorListener(new ErrorListener() {
            public void error(ErrorEvent ee) {
                System.out.println("error event for " + ee);
            }
        });

        lcd.openAny();
        System.out.println("waiting for TextLCD attachment...");
        lcd.waitForAttachment();

        System.out.println("Serial: " + lcd.getSerialNumber());
        System.out.println("rows: " + lcd.getRowCount());
        System.out.println("columns: " + lcd.getColumnCount());

        lcd.setDisplayString(0, "Hi There...");

        for (int i = 0; i < 100; i++)
        {
            lcd.setContrast(i);
            Thread.sleep(10);
        }

        lcd.setCustomCharacter(8, 0x0, 0xF8000);
        lcd.setCustomCharacter(9, 0x0, 0xFFC00);
        lcd.setCustomCharacter(10, 0x0, 0xFFFE0);
        lcd.setCustomCharacter(11, 0x0, 0xFFFFF);
        lcd.setCustomCharacter(12, 0xF8000, 0xFFFFF);
    }
}
```

```

        lcd.setCustomCharacter(13, 0xFFC00, 0xFFFFF);
        lcd.setCustomCharacter(14, 0xFFFE0, 0xFFFFF);
        lcd.setCustomCharacter(15, 0xFFFFF, 0xFFFFF);

        lcd.setDisplayString(1, "\010\011\012\013\014\015\016\017");
        //Note: representation is octal
        lcd.setBacklight(true);
        lcd.setCursorBlink(true);
        lcd.setCursor(true);

        System.out.print("closing...");
        lcd.close();
        lcd = null;
        System.out.println(" ok");
        if (false) {
            System.out.println("wait for finalization...");
            System.gc();
        }
    }
}

```

Detailed LCD Explanation

```

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

```

It imports the Phidget classes and events from the Java JNI Library (phidget21.jar)

```

TextLCDPhidget lcd;
System.out.println(Phidget.getLibraryVersion());

lcd = new TextLCDPhidget();

```

These lines create a new object of type *TextLCDPhidget* as *lcd* that will represent the a Phidget Text LCD in your program. All methods to control the Text LCD are implemented in this class.

The TextLCD Phidget consists of a Vacuum Fluorescent display that is capable of displaying Standard as well as custom characters in multiple rows.

```

lcd.addAttachListener(new AttachListener() {
    public void attached(AttachEvent ae) {
        System.out.println("attachment of " + ae);
    }
});
lcd.addDetachListener(new DetachListener() {
    public void detached(DetachEvent ae) {
        System.out.println("detachment of " + ae);
    }
});
lcd.addErrorListener(new ErrorListener() {

```

```

    public void error(ErrorEvent ee) {
        System.out.println("error event for " + ee);
    }
});

```

Attach and Detach are events raised by the Phidget21 library when devices are found or lost. Attach and Detach are generic events covering all Phidgets - it's possible for one Attach event to handle events from many different types of Phidgets. There are also events that are specific to a type of Phidget.

addAttachListener method adds an attach listener. The attach handler is a method that will be called when this Phidget is physically attached to the system, and has gone through its initialization, and so is ready to be used.

AttachListener is an interface that represents an AttachEvent. This event originates from all Phidgets.

Attached method is called with the event data when a new event arrives. AttachEvent is the event data object containing event data.

DetachListener interface represents a DetachEvent. detached method is called with the event data when a new event arrives. And Error is event raised when error occurs.

```

lcd.setDisplayString(0, "Hi There...");

for (int i = 0; i < 100; i++)
{
    lcd.setContrast(i);
    Thread.sleep(10);
}

lcd.setCustomCharacter(8, 0x0, 0xF8000);
lcd.setCustomCharacter(9, 0x0, 0xFFC00);
lcd.setCustomCharacter(10, 0x0, 0xFFFE0);
lcd.setCustomCharacter(11, 0x0, 0xFFFFF);
lcd.setCustomCharacter(12, 0xF8000, 0xFFFFF);
lcd.setCustomCharacter(13, 0xFFC00, 0xFFFFF);
lcd.setCustomCharacter(14, 0xFFFE0, 0xFFFFF);
lcd.setCustomCharacter(15, 0xFFFFF, 0xFFFFF);

lcd.setDisplayString(1, "\010\011\012\013\014\015\016\017");
//Note: representation is octal
lcd.setBacklight(true);
lcd.setCursorBlink(true);
lcd.setCursor(true);

```

openAny() will allow the program to interact with any phidget connected to the computer without a serial number. This method is the same as *open*, except that it specifies no serial number. Therefore, the first available Phidget will be opened. If there are two Phidgets of the same type attached to the system, you should specify a serial number, as there is no guarantee which Phidget will be selected by the call to *openAny()*.

waitForAttachment() Waits for this Phidget to become available. This method can be called after *open* has been called to wait for this Phidget to become available. This is

useful because open is asynchronous (and thus returns immediately), and most methods will throw a PhidgetException if they are called before a device is actually ready. This method is synonymous with polling the isAttached method until it returns True, or using the Attach event. This method blocks indefinitely until the Phidget becomes available. This can be quite some time (forever), if the Phidget is never plugged in. This method uses the attach handler internally to determine when the Phidget becomes available.

```
lcd.setDisplayString(0, "Hi There...");

for (int i = 0; i < 100; i++)
{
    lcd.setContrast(i);
    Thread.sleep(10);
}

lcd.setCustomCharacter(8, 0x0, 0xF8000);
lcd.setCustomCharacter(9, 0x0, 0xFFC00);
lcd.setCustomCharacter(10, 0x0, 0xFFFE0);
lcd.setCustomCharacter(11, 0x0, 0xFFFFF);
lcd.setCustomCharacter(12, 0xF8000, 0xFFFFF);
lcd.setCustomCharacter(13, 0xFFC00, 0xFFFFF);
lcd.setCustomCharacter(14, 0xFFFE0, 0xFFFFF);
lcd.setCustomCharacter(15, 0xFFFFF, 0xFFFFF);

lcd.setDisplayString(1, "\010\011\012\013\014\015\016\017");
//Note: representation is octal
lcd.setBacklight(true);
lcd.setCursorBlink(true);
lcd.setCursor(true);
```

setDisplayString() sets the display string ('Hi There...') of a certain row(0). If the string is longer than the row, it will be truncated.

setContrast() sets the contrast of the display. The valid range is 0-255. Changing the contrast can increase the readability of the display in certain viewing situation, such as at an odd angle. In the example, this methods can be used with *for* statement to slightly change the contrast.

setCustomCharacter() sets a custom character. You can set up to 8 custom characters, each one is completely defined by two integers, and gets stored in the character display until power is removed, whence they must be re-programmed. The characters lie in positions 8-15, and can be displayed by sending these codes to *setDisplayString* in amongst standard ASCII characters. The first parameter is index and the second parameter (param1) is first half of the character code. The last parameter (param2) is second half of the character code.

```
lcd.setDisplayString(1, "\010\011\012\013\014\015\016\017");
```

This code shows custom characters.

setBacklight() sets the status of the backlight. True turns the backlight on, False turns it off. The backlight is by default turned on.

setCursorBlink() sets the state of the cursor blink. True indicates that the cursor blink is on, False indicates that it is off. The cursor blink is an flashing box which appears directly to the right of the last entered character on the display, in the same spot as the cursor if it is enabled. The cursor blink is by default disabled.

setCursor() sets the state of the cursor. True indicates that the cursor on, False indicates that it is off. The cursor is an underscore which appears directly to the right of the last entered character on the display. The cursor is by default disabled.

Console Output

```
Phidget21 DLL Version 2.1.2 - Built Feb  5 2007 14:34:58
waiting for TextLCD attachment...
Serial: 14903
attachment of PhidgetTextLCD v113 #14903 (attached)
rows: 2
columns: 20
closing... ok
```

This is a sample program to test the basic capabilities of the PhidgetTextLCD. The rest of the functions for the PhidgetTextLCD are given in the form of JavaDocs and are the following.

3. Java Doc

This is the list of *all the available methods* that you can use for programming the TextLCDPhidget:

Constructor Detail

TextLCDPhidget

```
public TextLCDPhidget()
    throws PhidgetException
```

Throws:

[PhidgetException](#)

Method Detail

getRowCount

```
public int getRowCount()
    throws PhidgetException
```

Rwturns the number of rows available on the display.

Returns:

Number of rows

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

getColumnCount

```
public int getColumnCount()  
        throws PhidgetException
```

Returns the number of columns (characters per row) available on the display. This value is the same for every row.

Returns:

Number of columns

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

getContrast

```
public int getContrast()  
        throws PhidgetException
```

Returns the contrast of the display. This is the contrast of the entire display.

Returns:

Current contrast

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

setContrast

```
public void setContrast(int contrast)  
        throws PhidgetException
```

Sets the contrast of the display. The valid range is 0-255. Changing the contrast can increase the readability of the display in certain viewing situation, such as at an odd angle.

Parameters:

contrast - New contrast to set

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or the contrast value is out of range. See [open](#) for information on determining if a device is attached.

getBacklight

```
public boolean getBacklight()  
        throws PhidgetException
```

Returns the status of the backlight. True indicated that the backlight is on, False indicated that it is off. The backlight is by default turned on.

Returns:

Status of the backlight

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

setBacklight

```
public void setBacklight(boolean backlight)  
        throws PhidgetException
```

Sets the status of the backlight. True turns the backlight on, False turns it off. The backlight is by default turned on.

Parameters:

backlight - New backlight state

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

getCursor

```
public boolean getCursor()  
    throws PhidgetException
```

Returns the status of the cursor. True turns the cursor is on, False turns it off. The cursor is an underscore which appears directly to the right of the last entered character on the display. The cursor is by default disabLCD.

Returns:

Status of the cursor

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

setCursor

```
public void setCursor(boolean cursor)  
    throws PhidgetException
```

Sets the state of the cursor. True indicates that the cursor on, False indicates that it is off. The cursor is an underscore which appears directly to the right of the last entered character on the display. The cursor is by default disabLCD.

Parameters:

cursor - New cursor state

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

getCursorBlink

```
public boolean getCursorBlink()  
    throws PhidgetException
```

Returns the status of the cursor blink. True turns the cursor blink on, False turns it off. The cursor blink is an flashing box which appears directly to the right of the last entered character on the display, in the same spot as the cursor if it is enabLCD. The cursor blink is by default disabLCD.

Returns:

Status of the cursor blink

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

setCursorBlink

```
public void setCursorBlink(boolean cursorblink)  
    throws PhidgetException
```

Sets the state of the cursor blink. True indicates that the cursor blink is on, False indicates that it is off. The cursor blink is an flashing box which appears directly to the right of the last entered

character on the display, in the same spot as the cursor if it is enabLCD. The cursor blink is by default disablLCD.

Parameters:

cursorblink - New cursor blink state

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

setDisplayString

```
public void setDisplayString(int index,  
                             java.lang.String text)  
    throws PhidgetException
```

Sets the display string of a certain row. If the string is longer then the row, it will be truncated.

Parameters:

index - row

text - String

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the row is invalid. See [open](#) for information on determining if a device is attached.

setCustomCharacter

```
public void setCustomCharacter(int index,  
                               int param1,  
                               int param2)  
    throws PhidgetException
```

Sets a custom character. You can set up to 8 custom characters, each one is completely defined by two integers, and gets stored in the character display until power is removed, whence they must be re-programmed. The characters lie in positions 8-15, and can be displayed by sending these codes to setDisplayString in amongst standard ASCII characters. See the TextLCD java example for more information.

Parameters:

index - position (8-15)

param1 - first half of the character code

param2 - second half of the character code

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is invalid. See [open](#) for information on determining if a device is attached.

4. ASCII Character Codes

		Higher 4-bit (D4 to D7) of Character Code (Hexadecimal)																		
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F			
Lower 4-bit (D0 to D3) of Character Code (Hexadecimal)	0	CG RAM (1)			0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	1	CG RAM (2)		.	!	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G
	2	CG RAM (3)		"	#	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I
	3	CG RAM (4)		*	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H
	4	CG RAM (5)		*	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H
	5	CG RAM (6)		%	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K
	6	CG RAM (7)		8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	7	CG RAM (8)		'	7	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M
	8	CG RAM (1)		(9	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	9	CG RAM (2))	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	A	CG RAM (3)		*	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	B	CG RAM (4)		+	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	C	CG RAM (5)		.	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	D	CG RAM (6)		-	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	E	CG RAM (7)		.	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	F	CG RAM (8)		/	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N

		Higher 4-bit (D4 to D7) of Character Code (Hexadecimal)															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Lower 4 bit (D0 to D3) of Character Code (Hexadecimal)	0	CG RAM (1)	+		0	P	'	P	G	á		'	R	B	T		
	1	CG RAM (2)	=	!	!	A	a	a	a	a	i		J	+	y	v	
	2	CG RAM (3)	7	"	2	R	b	r	e	f	é		o	o	o	x	
	3	CG RAM (4)	L	#	3	S	s	s	s	s	ó		'	T	e	w	
	4	CG RAM (5)	7	*	4	T	t	t	t	t	ä		'	4	n	z	o
	5	CG RAM (6)	7	K	E	E	e	e	e	e	e		'	t	a	n	T
	6	CG RAM (7)	7	o	E	F	f	f	v	á	ó	*	'	w	o	o	*
	7	CG RAM (8)	7	'	7	E	w	w	S	O	R	X	'	+	Á	L	7
	8	CG RAM (1)	7	(E	X	H	x	e	ó	'	'	'	'	E	K	R
	9	CG RAM (2)	7)	9	Y	y	w	e	o	i	z	'	T	A	A	
	A	CG RAM (3)	*	*	#	J	Z	z	z	é	ó	á	z	'	z	v	T
	B	CG RAM (4)	J	+	*	K	k	C	I	R	á	k	'	L	T	v	*
	C	CG RAM (5)	=	,	<	L	\	l	i	n	o	*	'	J	o	z	o
	D	CG RAM (6)	w	-	-	M	m	D	i	o	*	'	'	w	T	-	-
	E	CG RAM (7)	E	.	>	N	n	n	á	o	o	T	'	o	o	o	o
	F	CG RAM (8)	E	/	?	O	_	o	á	á	z	'	'	o	o	o	o

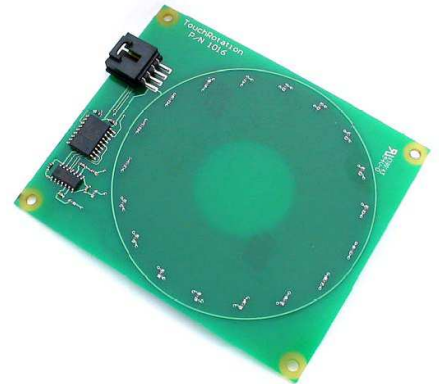
Section 3.6 Tutorial for Touch Rotation/Linear Touch/Analog Sensors

1. Introduction

PhidgetCircularTouch

It is just a matter of plugging the PhidgetCircularTouch into the USB port on your computer. After that, you can use the simple to program Phidgets software libraries to access these devices.

The PhidgetCircularTouch changes value when it is touched. It can detect approximately 125 steps. The PhidgetCircularTouch will work through ¼ inch of glass, and appears to the Phidget software libraries as an Interface Kit.



This device is consist of

- Your PhidgetCircularTouch.
- A USB cable.

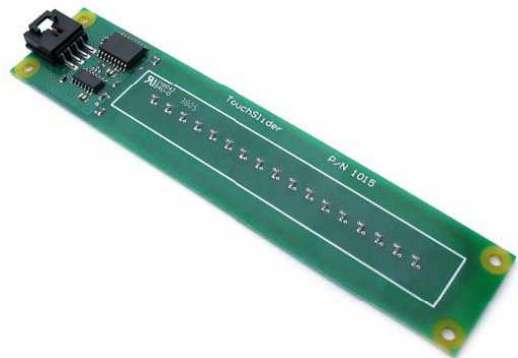
What Can the PhidgetCircularTouch Do?

Touch sensors like being touched. When you need human beings to input values into your project then getting them to touch a sensor is quick and intuitive. The PhidgetCircularTouch will detect when you place your finger on it, and returns a value depending on where you place your finger. Sliding your finger over the sensor continuously changes the value returned. The PhidgetCircularTouch can be used to set a direction, heading, or angle of rotation.

PhidgetLinearTouch

It is just a matter of plugging the Phidget-LinearTouch into the USB port on your computer. After that, you can use the simple to program Phidgets software libraries to access these devices.

The PhidgetLinearTouch changes value when it is touched. It can detect approximately 125 steps. The PhidgetLinearTouch will work through ¼ inch of glass, and appears to the Phidget software libraries as an Interface Kit.



This device is consist of

- Your PhidgetLinearTouch.
- A USB cable.

What Can the PhidgetLinearTouchDo?

Touch sensors like being touched. When you need human beings to input values into your project then getting them to touch a sensor is quick and intuitive. The PhidgetLinearTouch will detect when you place your finger on it, and returns a value depending on where you place your finger. Sliding your finger over the sensor continuously changes the value returned. The PhidgetLinearTouch can be used to set a value; any value.

PhidgetAnalogSensors

It is just a matter of plugging our off the shelf sensors into the PhidgetInterfaceKit 8/8/8, which in turn is plugged into the USB port on your computer. After that, you can use the simple to program Phidgets software libraries to access these devices.

A wide range of analog sensors are offered, all designed to plug into the PhidgetInterfaceKit 8/8/8 board.

What Can Analog Sensors Do?

Analog sensors measure continuous quantities, such as voltage, current, gas pressure, temperature, humidity, light, position, force, magnetic field, vibration, etc. There are many plug and play sensors in the Phidgets product line that require no assembly. Each sensor, which plugs into the PhidgetInterfaceKit 8/8/8, is described below.

Motion Sensors

This motion sensor is a raw form of the on/off sensors used to trigger lights in security systems. They work by concentrating on a 15 degree cone in front of the sensor, and splitting this in two sections. The difference in infrared radiation of these two sections is amplified, and eventually cancelled out after several seconds. This differential allows you to get a sense of the size of the object (as viewed from the sensor), and the direction of travel.

The sensor quickly filters out noise and tracks to about 500 if nothing is moving. As it turns out, you can simulate movement by moving the sensor!

Type: Active

Board Dimensions: 4.2 cm (diameter) x 6.0 cm



Sensor: Glolab PIR325

Rotation Sensor

This sensor rotates 300 degrees.

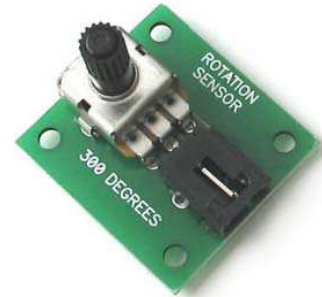
Type: Resistive 3-Pin

Board Dimensions: 3.1 x 3.3 cm

Mounting Holes: 2.3 x 2.5 cm

Formula: At fully clockwise the sensor reads zero, and at fully counter clockwise the sensor reads 1000. The maximum resistance of the potentiometer is 10 k ohm.

Sensor: CTS Series 296



Force Sensor

With no force applied this sensor will read zero. As force increases on the circular button the value increases towards 1000. It is not accurate enough to be used as a weight measurement device. The sensor is not designed to have force applied constantly over time.

Type: Resistive 2-Pin

Board Dimensions: 3.1 x 2.8 cm

Mounting Holes: 2.3 x 2.1 cm

Sensor: CUI IESP-12



Slider 60

This device is a variable resistor similar to a potentiometer.

Type: Resistive 3-Pin

Board Dimensions: 3.1 x 9.1 cm

Mounting Holes: 2.3 x 6.8 cm

Formula: When the slider is at one side it will read zero and 1000 when the slider is at the other end. The maximum resistance of the slider is 10 k ohm.

Sensor: Panasonic EWAQ1



Light Sensor

In the dark, the value produced is approximately zero. As the amount of light increases, the value increases towards 1000.

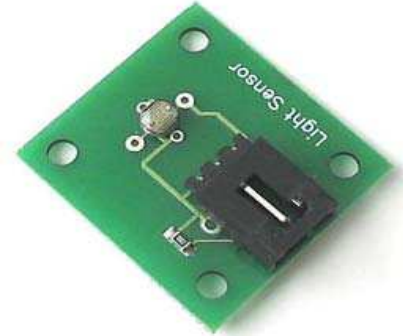
Type: Resistive 2-Pin

Board Dimensions: 3.1 x 2.8 cm

Mounting Holes: 2.3 x 2.1 cm

Formula: With no light the resistance of this sensor is 500 k ohm. At 10 lux the resistance falls to between 10 k and 5 k ohm. This resistance is in a voltage divider with a 7.5 k ohm resistor.

Sensor: A standard CdS (Cadmium Sulfide) photoresistor



Temperature Sensor

This sensor measures ambient temperature from -40 to +125 degrees Celsius. This device is a precision temperature to voltage converter that outputs a voltage that is directly proportional to temperature.

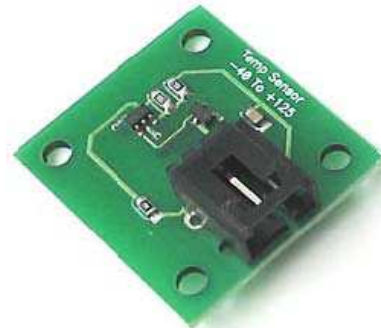
Type: Active

Board Dimensions: 3.1 x 2.8 cm

Mounting Holes: 2.3 x 2.0 cm

Formula: $\text{Temperature (in degrees Celsius)} = (\text{SensorValue} - 200) / 4$

Sensor: Microchip TC1047A

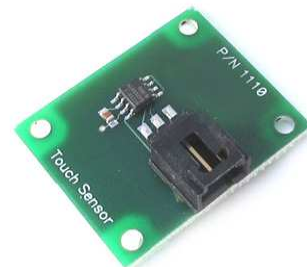


Touch Sensor

This sensor changes value from 1000 to 0 when it is touched. More specifically, this sensor is actually a capacitive change sensor. When the capacitance changes the sensor goes to zero. It will work through 1/4 inch of glass.

Type: Active

Board Dimensions: 3.1 x 3.8 cm



Mounting Holes: 2.3 x 3.0 cm

Sensor: QProx QT110

Humidity Sensor

This sensor measures the relative humidity of the environment around the sensor. Built in temperature compensation produces a linear output ranging from 10% to 95% relative humidity.

Type: Active

Board Dimensions: 3.1 x 5.0 cm

Mounting Holes: 2.3 x 4.3 cm

Formula: RH (in %) =
(SensorValue x 0.1946) – 41.98

Sensor: Humirel HTM1735



2. Programming

In order to program a PhidgetRFID create a Java project as described in the section 2. This is a sample code that describes some of the capabilities of the PhidgetRFID you can also find more details in the Examples file and in the JavaDocs described in the previous sections:

The example code is in\Phidget21Examples\JavaJNI\ OpenIFKitExample.java
Note : PhidgetCircularTouch uses same code as Phidget Interface Kit.

Entire Code

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

public class OpenIFKitExample
{
    public static final void main(String args[]) throws Exception {
        InterfaceKitPhidget ik;

        System.out.println(Phidget.getLibraryVersion());

        ik = new InterfaceKitPhidget();
        ik.addAttachListener(new AttachListener() {
            public void attached(AttachEvent ae) {
                System.out.println("attachment of " + ae);
            }
        });
        ik.addDetachListener(new DetachListener() {
            public void detached(DetachEvent ae) {
```

```

        System.out.println("detachment of " + ae);
    }
});
ik.addErrorListener(new ErrorListener() {
    public void error(ErrorEvent ee) {
        System.out.println("error event for " + ee);
    }
});
ik.addInputChangeListener(new InputChangeListener() {
    public void inputChanged(InputChangeEvent oe) {
        System.out.println(oe);
    }
});
ik.addOutputChangeListener(new OutputChangeListener() {
    public void outputChanged(OutputChangeEvent oe) {
        System.out.println(oe);
    }
});
ik.addSensorChangeListener(new SensorChangeListener() {
    public void sensorChanged(SensorChangeEvent se) {
        System.out.println(se);
    }
});

ik.openAny();
System.out.println("waiting for InterfaceKit attachment...");
ik.waitForAttachment();

System.out.println(ik.getDeviceName());

Thread.sleep(500);

System.out.println("input(7,8) = (" +
    ik.getInputState(7) + "," +
    ik.getInputState(8) + ")");

if (false) {
    System.out.print("closing...");
    System.out.flush();
    ik.close();
    System.out.println(" ok");
    System.out.print("opening...");
    ik.openAny();
    System.out.println(" ok");
    ik.waitForAttachment();
}
if (ik.getInputCount() > 8)
    System.out.println("input(7,8) = (" +
        ik.getInputState(7) + "," +
        ik.getInputState(8) + ")");
if (false) {
    System.out.print("turn on outputs (slowly)...");
    for (int i = 0; i < ik.getOutputCount(); i++) {
        ik.setOutputState(i, true);
        try {
            Thread.sleep(1000);
        } catch (Exception e) {
        }
    }
    System.out.println(" ok");
}

if (false)
    for (;;) {
        try {
            Thread.sleep(1000);
        } catch (Exception e) {
        }
    }
for (int j = 0; j < 1000; j++) {
    for (int i = 0; i < ik.getOutputCount(); i++) {

```

```

        ik.setOutputState(i, true);
    }
    for (int i = 0; i < ik.getOutputCount(); i++)
        ik.setOutputState(i, false);
}
if (false) {
    System.out.println("toggling outputs like crazy");
    boolean o[] = new boolean[ik.getOutputCount()];
    for (int i = 0; i < ik.getOutputCount(); i++)
        o[i] = ik.getOutputState(i);
    for (int i = 0; i < 100000; i++) {
        int n = (int)(Math.random() * ik.getOutputCount());
        ik.setOutputState(n, !o[n]);
        System.out.println("setOutputState " + n +
            ": " + !o[n]);
        o[n] = !o[n];
        try {
            Thread.sleep(1);
        } catch (Exception e) {
        }
    }
}

System.out.println("Outputting events.  Input to stop.");
System.in.read();
System.out.print("closing...");
ik.close();
ik = null;
System.out.println(" ok");
if (false) {
    System.out.println("wait for finalization...");
    System.gc();
}
}
}
}

```

Detailed Explanation

```

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

```

It imports the Phidget classes and events from the Java JNI Library (phidget21.jar)

```

InterfaceKitPhidget ik;

System.out.println(Phidget.getLibraryVersion());

ik = new InterfaceKitPhidget();

```

These lines create a new object of type *InterfaceKit* as *ik*. that will represent the phidget in your program. All the operation for the phidgets will be against this object.

This class represents a Phidget Interface Kit. All methods to read and write data to and from an Interface Kit are implemented in this class.

There are many types of Interface Kits, but each is simply a collection of 0 or more digital inputs, digital outputs and analog sensors. Inputs can be read and outputs can be set, and event handlers can be set for each of these.

```
ik.addAttachListener(new AttachListener() {
    public void attached(AttachEvent ae) {
        System.out.println("attachment of " + ae);
    }
});
ik.addDetachListener(new DetachListener() {
    public void detached(DetachEvent ae) {
        System.out.println("detachment of " + ae);
    }
});
ik.addErrorListener(new ErrorListener() {
    public void error(ErrorEvent ee) {
        System.out.println("error event for " + ee);
    }
});
ik.addInputChangeListener(new InputChangeListener() {
    public void inputChanged(InputChangeEvent oe) {
        System.out.println(oe);
    }
});
ik.addOutputChangeListener(new OutputChangeListener() {
    public void outputChanged(OutputChangeEvent oe) {
        System.out.println(oe);
    }
});
ik.addSensorChangeListener(new SensorChangeListener() {
    public void sensorChanged(SensorChangeEvent se) {
        System.out.println(se);
    }
});
```

Attach and Detach are events raised by the Phidget21 library when devices are found or lost. Attach and Detach are generic events covering all Phidgets - it's possible for one Attach event to handle events from many different types of Phidgets. There are also events that are specific to a type of Phidget.

addAttachListener method adds an attach listener. The attach handler is a method that will be called when this Phidget is physically attached to the system, and has gone through its initialization, and so is ready to be used.

AttachListener is an interface represents an AttachEvent. This event originates from all Phidgets.

Attached method is called with the event data when a new event arrives. AttachEvents is the event data object containing event data.

DetachListener interface represents a DetachEvent. detached method is called with the event data when a new event arrives.

Error is event raised when error occurs. . The output change handler is a method that will be called when an output on this Interface Kit has changed. The sensor change handler is a method that will be called when a sensor on this Interface Kit has changed by at least the Trigger that has been set for this sensor.

```
ik.openAny();
System.out.println("waiting for InterfaceKit attachment...");
ik.waitForAttachment();

...
System.in.read();
```

openAny() will allow the program to interact with any phidget connected to the computer without a serial number. This method is the same as [open](#), except that it specifies no serial number. Therefore, the first available Phidget will be opened. If there are two Phidgets of the same type attached to the system, you should specify a serial number, as there is no guarantee which Phidget will be selected by the call to *openAny()*.

waitForAttachment() Waits for this Phidget to become available. This method can be called after open has been called to wait for this Phidget to become available. This is useful because open is asynchronous (and thus returns immediately), and most methods will throw a PhidgetException if they are called before a device is actually ready. This method is synonymous with polling the *isAttached* method until it returns True, or using the Attach event. This method blocks indefinitely until the Phidget becomes available. This can be quite some time (forever), if the Phidget is never plugged in. This method uses the attach handler internally to determine when the Phidget becomes available.

System.in.read() - If any key input is occurred in the console, this program will end. Before ending, we should close Phidget, because we opened.

Console Output

```
Phidget21 DLL Version 2.1.2 - Built Feb 5 2007 14:34:58
waiting for InterfaceKit attachment...
attachment of PhidgetInterfaceKit v102 #28847 (attached)
Phidget Touch Rotation
PhidgetInterfaceKit v102 #28847 (attached) input 0 changed to false
PhidgetInterfaceKit v102 #28847 (attached) input 1 changed to false
input(7,8) = (false,false)
Outputting events. Input to stop.
PhidgetInterfaceKit v102 #28847 (attached) input 1 changed to true
PhidgetInterfaceKit v102 #28847 (attached) input 0 changed to true
PhidgetInterfaceKit v102 #28847 (attached) sensor 0 changed to 803
PhidgetInterfaceKit v102 #28847 (attached) sensor 0 changed to 787
PhidgetInterfaceKit v102 #28847 (attached) sensor 0 changed to 842
PhidgetInterfaceKit v102 #28847 (attached) sensor 0 changed to 882
```

```
...
PhidgetInterfaceKit v821 #17494 (attached) sensor 1 changed to 307
PhidgetInterfaceKit v821 #17494 (attached) sensor 1 changed to 291
PhidgetInterfaceKit v821 #17494 (attached) sensor 1 changed to 305
PhidgetInterfaceKit v821 #17494 (attached) sensor 1 changed to 321
PhidgetInterfaceKit v821 #17494 (attached) sensor 1 changed to 310
PhidgetInterfaceKit v821 #17494 (attached) sensor 1 changed to 256
PhidgetInterfaceKit v821 #17494 (attached) sensor 1 changed to 162
PhidgetInterfaceKit v821 #17494 (attached) sensor 1 changed to 76
PhidgetInterfaceKit v821 #17494 (attached) sensor 1 changed to 42
PhidgetInterfaceKit v821 #17494 (attached) sensor 1 changed to 81

closing... ok
```

This is a sample program to test the basic capabilities of the PhidgetCircularTouch. The rest of the functions for the PhidgetCircularTouch are the following, given in JavaDoc format

3. Java Doc

This is the list of *all the available methods* that you can use for programming the InterfaceKit:

Constructor Detail

InterfaceKitPhidget

```
public InterfaceKitPhidget()
```

```
    throws PhidgetException
```

Class Constructor. Calling this opens a connection to the phidget21 C library creates an internal handle for this Phidget, ready to call open on.

Throws:

[PhidgetException](#) - If there was a problem connecting to phidget21 or creating the internal handle.

Method Detail

getOutputCount

```
public int getOutputCount()
```

```
    throws PhidgetException
```

Returns the number of digital outputs on this Interface Kit. Not all interface kits have the same number of digital outputs, and some don't have any digital outputs at all.

Returns:

Number of digital outputs

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

getInputCount

```
public int getInputCount()  
        throws PhidgetException
```

Returns the number of digital inputs on this Interface Kit. Not all interface kits have the same number of digital inputs, and some don't have any digital inputs at all.

Returns:

Number of digital inputs

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

getSensorCount

```
public int getSensorCount()  
        throws PhidgetException
```

Returns the number of analog inputs on the Interface Kit. Not all interface kits have the same number of analog inputs, and some don't have any analog inputs at all.

Returns:

Number of analog inputs

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached. See [open](#) for information on determining if a device is attached.

getInputState

```
public boolean getInputState(int index)  
        throws PhidgetException
```

Returns the state of a digital input. Digital inputs read True where they are activated and false when they are in their default state.

Be sure to check [getInputCount](#) first if you are unsure as to the number of inputs, so as not to set an Index that is out of range.

Parameters:

index - Index of the input

Returns:

State of the input

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is out of range. See [open](#) for information on determining if a device is attached.

getOutputState

```
public boolean getOutputState(int index)  
        throws PhidgetException
```

Returns the state of a digital output. Depending on the Phidget, this value may be either the value that you last wrote out to the Phidget, or the value that the Phidget last returned. This is because some Phidgets return their output state and others do not. This means that with some devices, reading the output state of a pin directly after setting it, may not return the value that you just set.

Be sure to check [getOutputCount](#) first if you are unsure as to the number of outputs, so as not to attempt to get an Index that is out of range.

Parameters:

index - Index of the output

Returns:

State of the output

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is out of range. See [open](#) for information on determining if a device is attached.

getSensorValue

```
public int getSensorValue(int index)
    throws PhidgetException
```

Returns the value of a analog input. The analog inputs are where analog sensors are attached on the InterfaceKit 8/8/8. On the Linear and Circular touch sensor Phidgets, analog input 0 represents position on the slider.

The valid range is 0-1000. In the case of a sensor, this value can be converted to an actual sensor value using the formulas provided here: <http://www.phidgets.com/documentation/Sensors.pdf>

Parameters:

index - Index of the sensor

Returns:

Sensor value

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is out of range. See [open](#) for information on determining if a device is attached.

getSensorRawValue

```
public int getSensorRawValue(int index)
    throws PhidgetException
```

Returns the raw value of a analog input. This is a more accurate version of [getSensorValue](#). The valid range is 0-4095. Note however that the analog outputs on the Interface Kit 8/8/8 are only 10-bit values and this value represents an oversampling to 12-bit.

Parameters:

index - Index of the sensor

Returns:

Sensor value

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is out of range. See [open](#) for information on determining if a device is attached.

getSensorChangeTrigger

```
public int getSensorChangeTrigger(int index)
    throws PhidgetException
```

Returns the change trigger for an analog input. This is the ammount that an inputs must change between successive SensorChangeEvents. This is based on the 0-1000 range provided by [getSensorValue](#). This value is by default set to 10 for most Interface Kits with analog inputs.

Parameters:

index - Index of the sensor

Returns:

Trigger value

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is out of range. See [open](#) for information on determining if a device is attached.

setOutputState

```
public void setOutputState(int index,  
                           boolean newVal)  
    throws PhidgetException
```

Sets the state of a digital output. Setting this to true will activate the output, False is the default state.

Parameters:

index - Index of the output
newVal - State to set the output to

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is out of range. See [open](#) for information on determining if a device is attached.

setSensorChangeTrigger

```
public void setSensorChangeTrigger(int index,  
                                    int newVal)  
    throws PhidgetException
```

Sets the change trigger for an analog input. This is the amount that an inputs must change between successive SensorChangeEvents. This is based on the 0-1000 range provided by `getSensorValue`. This value is by default set to 10 for most Interface Kits with analog inputs.

Parameters:

index - Input
newVal - Value

Throws:

[PhidgetException](#) - If this Phidget is not opened and attached, or if the index is out of range. See [open](#) for information on determining if a device is attached.

addInputChangeListener

```
public final void addInputChangeListener(InputChangeListener l)
```

Adds an input change listener. The input change handler is a method that will be called when an input on this Interface Kit has changed.

There is no limit on the number of input change handlers that can be registered for a particular Phidget.

Parameters:

l - An implementation of the [InputChangeListener](#) interface

removeInputChangeListener

```
public final void removeInputChangeListener(InputChangeListener l)
```

Removes an input change listener. This will remove a previously added input change listener.

addOutputChangeListener

```
public final void addOutputChangeListener(OutputChangeListener l)
```

Adds an output change listener. The output change handler is a method that will be called when an output on this Interface Kit has changed.

There is no limit on the number of output change handlers that can be registered for a particular Phidget.

Parameters:

1 - An implementation of the [OutputChangeListener](#) interface

removeOutputChangeListener

```
public final void removeOutputChangeListener(OutputChangeListener l)
```

Removes an output change listener. This will remove a previously added output change listener.

addSensorChangeListener

```
public final void addSensorChangeListener(SensorChangeListener l)
```

Adds a sensor change listener. The sensor change handler is a method that will be called when a sensor on this Interface Kit has changed by at least the [Trigger](#) that has been set for this sensor.

There is no limit on the number of sensor change handlers that can be registered for a particular Phidget.

Parameters:

1 - An implementation of the [SensorChangeListener](#) interface

removeSensorChangeListener

```
public final void removeSensorChangeListener(SensorChangeListener l)
```

Removes a sensor change listener. This will remove a previously added sensor change listener.

References:

[1] www.phidgets.com