

Preliminary

.NET API Manual

Describes the Application Program Interface (API) for each Phidget device. The API can be used by a number of languages; this manual discusses use via C# and the code examples reflect this.

How to use Phidgets

Phidgets are an easy to use set of building blocks for low cost sensing and control from your PC. Using the Universal Serial Bus (USB) as the basis for all Phidgets, the complexity is managed behind this easy to use and robust Application Program Interface (API) library.

The library is a network library – the computer with the phidgets attached must be running the PhidgetWebService; even if the computer with the phidgets is also running the .NET application. The computer running the .NET application does not require a USB Host – just network access. This allows this library to be used on .NET devices such as PDAs and cell phones.

Installing the Library

To access the Phidget .NET software components, you must first install the library on your computer:

- To install the library go to www.Phidgets.com >> Downloads >> Release
- Select the PHIDGET.msi file.

A dialog box will appear, asking if you would like to open the file or save it to your computer. You can do either, but if you are unsure just select Open and follow the instructions.

Now, to use Phidgets from the Visual Studio IDE, open Project >> Add Reference, and browse for PhidgetsNET.DLL. This installs by default into C:\Program Files\Phidgets. Plug in your hardware, and you are ready to go!

Running the PhidgetWebService

The PhidgetWebService must be running on the computer that has the phidgets attached. PhidgetWebService.exe is installed into C:\Program Files\Phidgets by Phidget.MSI, so it should already be on your computer. PhidgetWebService requires you, at minimum, to specify a password that will be used to authenticate any clients; such as your .NET application. Create a shortcut to PhidgetWebService.exe, specifying PhidgetWebService.exe pass as the Target. Run the shortcut; and PhidgetWebService should be running. This instance of PhidgetWebService is running on Port 5001, with a Server SerialNumber of 0. The Server SerialNumber is used to find a unique instance of PhidgetWebService on local network without worrying about IP addresses.

Basic Example

This is a very simple example of how to access and control a PhidgetServo. It attempts to open a PhidgetServo, and if successful, will set servo motor #0 to position 90. (approximately 90 degrees).

Note that since the open functions will return before a device is actually ready, you should either wait for the attach event to fire or poll the IsAttached() function for true, before trying to access the device.

```
namespace ServoTest
{
    public partial class Form1 : Form
    {
        PhidgetServo servo;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            servo = new PhidgetServo();
            servo.Attach += new AttachEventHandler(Attach);
            servo.OpenRemote(0, 5001, -1, "pass");
        }
        void Attach(object sender, AttachEventArgs e)
        {
            servo.SetMotorPosition(0, 90);
        }
    }
}
```

Phidget Manager

The Phidget Manager is a useful tool for keeping track of Phidgets attached to a PhidgetWebService. It can also be used to fire attach and detach events for all Phidgets. This is useful if you just want to know which Phidgets are attached to the computer.

Phidget Manager derives from Phidget, so you can use all of the common Phidget functions with a Phidget Manager, even though some don't really make a lot of sense (such as `GetDeviceType()`). You'll want to add Event handlers to the Attach and Detach events, and use these to keep track of Phidgets being plugged in and removed. There is also a `getPhidgets()` function available that returns an `ArrayList` of `Phidget` objects representing the current list of attached devices.

This example uses the Phidget Manager to wait for a PhidgetServo to be connected, then moves it and exits.

```
namespace ManagerExample
{
    public partial class Form1 : Form
    {
        PhidgetManager Manager;
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            Manager = new PhidgetManager();
            Manager.Attach += new AttachEventHandler(Attach);
            Manager.OpenRemote(0, 5001, "pass");
        }
        void Attach(object sender, AttachEventArgs e)
        {
            if (e.getPhidget().GetDeviceType().Equals("PhidgetServo"))
            {
                PhidgetServo servo = new PhidgetServo();
                servo.OpenRemote(0, 5001,
                    e.getPhidget().GetSerialNumber(), "pass");
                servo.SetMotorPosition(0, 90);
                servo.Close();
            }
        }
    }
}
```

Events

All devices share three common events: Attach, Detach and Error. There are also many device specific events. These events have been implemented in a standard way for .NET. Essentially, you need to create a new event handler delegate that points to a specific function that you want run when the event occurs, and then add that to the event to register the callback. You can see an example of this in the Phidget Manager section.

Each event has an associated delegate and argument class for passing data to event handlers. For example, the Attach event uses the AttachEventHandler delegate and the AttachEventArgs class. So to add a callback to the Attach event:

```
Attach += new AttachEventHandler(attachEvent);
```

Where you have created the attachEvent function:

```
void attachEvent(object sender, AttachEventArgs e)
{
    //do something on an attach...
}
```

Each argument class defines getter functions for accessing data passed to the event handler:

```
ie: e.getPhidget();
```

You can add as many event handling functions as you like to each event, and also add each event handling function to as many events as you like.

Calls common to each device

The following calls and events are common to all Phidget components:

int OpenRemote (int ServerSerialNumber, int Port, int SerialNumber, String password);

Attempts to locate a Phidget matching your requirements on a PhidgetWebService found on the local network.

SerialNumber specifies the desired Phidget serial number, allowing the call to open a specific Phidget. Specifying -1 for the serial number will cause it to open the first available Phidget for the specific software component that you are using.

ServerSerialNumber specifies the PhidgetWebService that the Phidget is found on.

Port specifies the port that the PhidgetWebService is listening on. By default, this port is 5001.

Password is used to authenticate with the server. A server will only accept requests from clients that send the valid password.

int OpenRemoteIP (String IPAddress, int Port, int SerialNumber, String password);

Attempts to locate a Phidget matching your requirements on a specific PhidgetWebService on the network. OpenRemoteIP is recommended to be used with servers that have a static IP Address.

SerialNumber specifies the desired Phidget serial number, allowing the call to open a specific Phidget. Specifying -1 for the serial number will cause it to open the first available Phidget for the specific software component that you are using.

IPAddress specifies the server that the Phidget is found on. This can be the hostname, or the IP address.

Port specifies the port that the server is listening on. By default, this port is 5001, unless the server is run with a different port specified.

Password is used to authenticate with the server. A server will only accept requests from clients that send the valid password.

int Close ();

Closes this software object.

bool IsAttached ();

Returns a bool indicating the status of the device.

String GetDeviceType ();

Sets a pointer to a null terminated string describing the name of the Phidget. All PhidgetInterfaceKits will return "PhidgetInterfaceKit", PhidgetRFID returns "PhidgetRFID" and so on.

int GetDeviceVersion ();

Returns the device version of this Phidget.

int GetSerialNumber ();

Returns the unique serial number of this Phidget. This number is set during manufacturing, and is unique across all Phidgets.

String GetServerAddress ();

Returns the IP address of the Phidget WebService hosting the device.

int GetServerPort ();

Returns the Port of the Phidget WebService hosting the device.

int GetServerID ();

Returns the Server ID of the Phidget WebService hosting the device.

event DetachEventHandler Detach;

Occurs when the device is unplugged, or closed from software.

event AttachEventHandler Attach;

Occurs when the device is plugged in, or discovered from software.

event ErrorHandler Error;

Occurs whenever an unexpected condition occurs.

Specific devices

Each Phidget component is described along with its API.

PhidgetAccelerometer

The PhidgetAccelerometer is a component that provides a high-level programmer interface to control a PhidgetAccelerometer device connected through a USB port. With this component, the programmer can:

- Measure +-2 times Gravity (9.8 m/s²) change per axis.
- Measure both dynamic acceleration (e.g., vibration) and static acceleration (e.g., gravity or tilt).

In addition to the common calls described above, the following calls are specific to this device:

int GetNumAxis ();

Get the number of axes available from the PhidgetAccelerometer.

double GetAcceleration (int Index);

Gets the last acceleration value received from the PhidgetAccelerometer for a particular axis.

double GetAccelerationChangeTrigger (int Index);

Gets the amount of change that should exist between the last reported value and the current value before an OnAccelerationChange event is fired.

void SetAccelerationChangeTrigger (int Index, double newVal);

Specifies the amount of change that should exist between the last reported value and the current value before an OnAccelerationChange event is fired.

event AccelerationChangeEventHandler AccelerationChange;

Occurs when the acceleration changes by more than the Acceleration trigger.

PhidgetEncoder

The PhidgetEncoder is a component that provides a high-level programmer interface to control a PhidgetEncoder device connected through a USB port. With this component, the programmer can:

- Detect changes in position of incremental and absolute encoders.
- Easily track the changes with respect to time.

In addition to the common calls described above, the following calls are specific to this device:

int GetNumEncoders ();

Gets the number of encoders available on the Phidget.

int GetNumInputs ();

Returns the number of inputs on this particular Phidget device. Please see the hardware description for the details of device variations.

int GetEncoderPosition (int Index);

Gets the last position value received from the encoder for the particular encoder selected.

bool GetInputState (int Index);

Returns the state of the designated input. In the case of a switch or button which is normally open, True would correspond to when the switch is pressed.

void SetEncoderPosition (int Index);

Specifies a new value for the current position of the encoder.

event InputChangeEventHandler InputChange;

Occurs when an input changes.

event PositionChangeEventHandler PositionChange;

Occurs when the encoder changes position.

PhidgetInterfaceKit

The PhidgetInterfaceKit is a component that provides a high-level programmer interface to control a PhidgetInterfaceKit device connected through a USB port. With this component, the programmer can:

- Turn particular outputs on and off.
- Get notified of changes of state of the inputs as events.
- Configure events to fire when the analog inputs change.

The PhidgetInterfaceKit devices provide a combination of

- Digital outputs.
- Digital inputs.
- Analog inputs.

In addition to the common calls described above, the following calls are specific to this device:

int GetNumOutputs ();

Returns the number of outputs on this particular Phidget device. Please see the hardware description for the details of device variations.

int GetNumInputs ();

Returns the number of inputs on this particular Phidget device. Please see the hardware description for the details of device variations.

int GetNumSensors ();

Gets the number of analog inputs available on the given PhidgetInterface Kit.

bool GetInputState (int Index);

Returns the state of the designated input. In the case of a switch or button which is normally open. True would correspond to when the switch is pressed.

bool GetOutputState (int Index);

Returns the state of the designated output to True or False. Depending on the type of output available the specified output goes to a high value or completes a connection. Please see the hardware description for details on different outputs.

int GetSensorValue (int Index);

Gets the last reported sensor value for the given index.

int GetSensorNormalizeMinimum (int Index);

Gets the minimum setting of a sensor's range. This is usually 0, but if one sets it to a larger value, e.g., 200, then the scale is adjusted so that the real range of 200 - 1000 are normalized to 0 - 1000. Actual readings less than this number are always reported as 0. There must be at least a difference of 1 between this property and SensorNormalizeMaximum, otherwise INVALIDARG is returned.

int GetSensorNormalizeMaximum (int Index);

Gets the maximum setting of a sensor's range. This is usually 1000, but if one sets it to a smaller value, e.g., 700, then the scale is adjusted so that the real range of 0 - 700 are normalized to 0 - 1000. Actual readings greater than this number are always reported as 1000. There must be at least a difference of 1 between this property and SensorNormalizeMinimum, otherwise an INVALIDARG error is returned.

int GetSensorChangeTrigger (int Index);

Gets the amount of change that should exist between the last reported value and the current value before an OnSensorChange event is fired.

void SetOutputState (int Index, bool newVal);

Sets the state of the designated output to True or False. Depending on the type of output available the specified output goes to a high value or completes a connection. Please see the hardware description for details on different outputs.

void SetSensorNormalizeMinimum (int Index, int newVal);

Specifies the minimum setting of a sensor's range.

void SetSensorNormalizeMaximum (int Index, int newVal);

Specifies the maximum setting of a sensor's range.

void SetSensorChangeTrigger (int Index, int newVal);

Specifies the amount of change that should exist between the last reported value and the current value before an OnSensorChange event is fired.

bool GetRatiometric (InterfaceKit, int);

desc

void SetRatiometric (bool newVal);

desc

event InputChangeEventHandler InputChange;

Occurs when an input changes.

event SensorChangeEventHandler SensorChange;

Occurs when a sensor value changes by more than the OnSensorChange trigger.

PhidgetLED

The PhidgetLED is a component that provides a high-level programmer interface to control a PhidgetLED device connected through a USB port. With this component, the programmer can:

- Control each led individually, On/Off and Brightness.

In addition to the common calls described above, the following calls are specific to this device:

int GetNumLEDs ();

Returns the number of LED positions available in this Phidget.

int GetDiscreteLED (int Index);

Gets the brightness of an individual LED. Range of brightness is 0-100.

void SetDiscreteLED (int Index, int newVal);

Sets the brightness of an individual LED. Range of brightness is 0-100.

PhidgetMotorControl

The PhidgetMotorControl is a component that provides a high-level programmer interface to control a PhidgetMotorControl device connected through a USB port. With this component, the programmer can:

- Control direction, and start and stop DC motors.
- Control the velocity and acceleration of each DC motor.
- Read the limit switch.

In addition to the common calls described above, the following calls are specific to this device:

int GetNumMotors ();

Returns the maximum number of motors on this particular Phidget device. This depends on the actual hardware. Please see the hardware description for the details of device variations.

Note: that there is no way of programmatically determining how many motors are actually plugged into the hardware.

int GetNumInputs ();

Returns the number of inputs on this particular Phidget device. Please see the hardware description for the details of device variations.

bool GetInputState (int Index);

Returns the state of the designated input. In the case of a switch or button which is normally open, True would correspond to when the switch is pressed.

Int GetAcceleration (int Index);

desc

int GetMotorSpeed (int Index);

desc

void SetAcceleration (int Index, int newVal);

desc

void SetMotorSpeed (int Index, int newVal);

desc

event InputChangeEventHandler InputChange;

Occurs when an input changes.

event MotorChangeEventHandler MotorChange;

Occurs when the motor speed changes.

PhidgetPHSensor

The PhidgetPHSensor is a component that provides a high-level programmer interface to control a PhidgetPHSensor device connected through a USB port. With this component, the programmer can:

- Read the PH of a solution with a PH probe.

In addition to the common calls described above, the following calls are specific to this device:

double GetPH ();

Returns the current ph.

double GetPHChangeTrigger ();

Gets the amount of change that should exist between the last reported value and the current value before an OnPHChange event is fired.

void SetPHChangeTrigger (double newVal);

Specifies the amount of change that should exist between the last reported value and the current value before an OnPHChange event is fired.

event PHChangeEventHandler PHChange;

Occurs when the PH changes by more than the PH trigger.

PhidgetRFID

The PhidgetRFID is a component that provides a high-level programmer interface to control a PhidgetRFID device connected through a USB port. With this component, the programmer can:

- Read Radio Frequency Identification tags.

Radio Frequency Identification or RFID, is a non-contact identification technology which uses a reader to read data stored on low cost tags. The particular instance of the technology we use stores a 40-bit number on the tag. Every tag that is purchased from Phidgets Inc. is guaranteed unique.

When a RFID tag is read, the component returns the unique number contained in the RFID tag.

In addition to the common calls described above, the following calls are specific to this device:

int GetNumOutputs ();

Returns the number of outputs on this particular Phidget device. Please see the hardware description for the details of device variations.

bool GetOutputState (int Index);

Returns the state of the designated output to True or False. Depending on the type of output available the specified output goes to a high value or completes a connection. Please see the hardware description for details on different outputs.

void SetOutputState (int Index, bool newVal);

Sets the state of the designated output to True or False. Depending on the type of output available the specified output goes to a high value or completes a connection. Please see the hardware description for details on different outputs.

String GetLastTag ();

Returns the last tag that the reader has registered.

event TagEventHandler Tag;

Occurs when a Tag is read.

PhidgetServo

The PhidgetServo is a component that provides a high-level programmer interface to control a PhidgetServo device connected through a USB port. With this component, the programmer can:

- Set the desired position for a servo motor, ranging from 0 to 180 degrees.

In addition to the common calls described above, the following calls are specific to this device:

double GetMotorPosition (int Index);

Gets the desired servo motor position for a particular servo motor.

void SetMotorPosition (int Index, double newVal);

Sets the desired servo motor position for a particular servo motor. This value may range from -23 to 231, corresponding to time width of the control pulse, the angle that the Servo motor moves to depends on the characteristic of individual motors. Please read the PhidgetServo documentation to understand what behaviour you can expect from your motors.

int GetNumMotors ();

Returns the maximum number of motors on this particular Phidget device. This depends on the actual hardware. Please see the hardware description for the details of device variations.

Note that there is no way of programmatically determining how many motors are actually plugged into the hardware.

PhidgetTemperatureSensor

The PhidgetTemperatureSensor is a component that provides a high-level programmer interface to control a PhidgetTemperatureSensor device connected through a USB port. With this component, the programmer can:

- Read the temperature of Thermocouple device.
- Read cold junction temperature.
- Get notification of temperature change.
- Use metric or imperial units.

In addition to the common calls described above, the following calls are specific to this device:

int GetNumTemperatureInputs ();

Returns the integer value of the number of thermocouple inputs.

double GetTemperature (int Index);

Returns the current temperature in Celsius or Fahrenheit (depending on UseImperial property). Index = 0 returns the temperature of the cold junction. Index = 1 returns the temperature of the thermocouple.

double GetTemperatureChangeTrigger (int Index);

Gets the amount of change that should exist between the last reported value and the current value before an OnTemperatureChange event is fired.

void SetTemperatureChangeTrigger (int Index, double newVal);

Specifies the amount of change that should exist between the last reported value and the current value before an OnTemperatureChange event is fired.

void SetUseImperial (bool newVal);

Specifies the state indicating if Metric (SI units) or Imperial (USA, Myanmar, Liberia) units are being used.

event TemperatureChangeEventHandler TemperatureChange;

Occurs when the Temperature changes by more than the Temperature trigger.

PhidgetTextLCD

The PhidgetTextLCD is a component that provides a high-level programmer interface to control a PhidgetTextLCD device connected through a USB port. With this component, the programmer can:

- Display text on a PhidgetTextLCD module.

In addition to the common calls described above, the following calls are specific to this device:

int GetNumRows ();

Returns number of rows of text that may be presented on the display.

int GetNumColumns ();

Returns number of columns of text that may be used on the display.

bool GetBacklight ();

Determines if the backlight for this LCD is on or off.

bool GetCursorOn ();

Determines if the cursor is on or off.

bool GetCursorBlink ();

Determines if the cursor's blinking is on or off.

void SetBacklight (bool newVal);

Sets the backlight for this LCD on or off.

void SetCursorOn (bool newVal);

Sets the cursor on or off. This cursor is displayed at the last location that was changed.

void SetCursorBlink (bool newVal);

Sets the cursor's blinking on or off.

void SetDisplayString (int Row, String displayString);

Sets the text to display on a particular row of the display. The text will be clipped at the right edge of the display.

PhidgetTextLED

The PhidgetTextLED is a component that provides a high-level programmer interface to control a PhidgetTextLED device connected through a USB port. With this component, the programmer can:

- Display text and numbers on segment type LED modules.
- Brightness can be controlled for the entire display.

In the case of 7-segment LED characters numbers are displayed easily and text can be displayed with some restrictions.

In addition to the common calls described above, the following calls are specific to this device:

int GetNumRows ();

Returns number of rows of text that may be presented on the display.

int GetNumColumns ();

Returns number of columns of text that may be used on the display.

int GetBrightness ();

Returns the brightness value of the LED display.

void SetBrightness (int newVal);

Sets the brightness of the LED display. Varying this property will control the brightness uniformly across all digits in the display.

void SetDisplayString (int Row, String displayString);

Sets the text to display on a particular row of the display. Will clip the text at the right edge of the display.

PhidgetWeightSensor

The PhidgetWeightSensor is a component that provides a high-level programmer interface to control a PhidgetWeightSensor device connected through a USB port. With this component, the programmer can:

- Read the weight of an item or person on the weight scale.

In addition to the common calls described above, the following calls are specific to this device:

double GetWeight ();

Returns the current weight in Kilograms or Pounds (depending on UseImperial property).

double GetWeightChangeTrigger ();

Gets the amount of change that should exist between the last reported value and the current value before an OnWeightChange event is fired.

void SetWeightChangeTrigger (double newVal);

Specifies the amount of change that should exist between the last reported value and the current value before an OnWeightChange event is fired.

void SetUseImperial (bool newVal);

Specifies the state indicating if Metric (SI units) or Imperial (USA, Myanmar, Liberia) units are being used.

event WeightChangeEventHandler WeightChange;

Occurs when the Weight changes by more than the Weight trigger.