C 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.

Library Version: 1.0.6

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 was written originally for Windows, but has been ported to MacOS and Linux. Although the library is written in C, the functions can be called from a number of languages including C, C++, Objective-C, Matlab, etc. The source code is freely available for reference/debugging purposes.

Basic Architecture

Phidgets are controlled on this layer using a specific handle for the type of Phidget you are using. The naming convention for these handles is "C <specific Phidget name> handle". For example, PhidgetRFID is controlled using the `CPhidgetRFIDhandle` type. Operations are performed by passing this handle to a set of functions. The handle is allocated using the `_create` function, and associated with a Phidget device using the `CPhidget_open` function. Handlers can then be registered, and commands can be issued. If the application needs data from the device, the `_read` function is polled. Lastly, the `CPhidget_close` function is called to disconnect the handle from the Phidget, and the handle is destroyed using the `CPhidget_delete` function, to free up the memory being used by the underlying structure.

Basic Example

This is a very simple example of how to access and control a PhidgetServo. It opens the servo, sets the position, and then closes and deletes the handle.

```
#include "phidget20.h"

int main(int argc, char* argv[])
{
    int result;
    CPhidgetServoHandle Servo = 0;

    /* open any PhidgetServo */
    CPhidgetServo_create(&Servo);
    if(CPhidget_open(Servo, -1))
    {
          printf("Unable to find a PhidgetServo.\n");
          return 0;
    }

    /* Set position of Servo 0 to 90.0 degrees */
    if(CPhidgetServo_setMotorPosition(Servo, 0, 90.0))
    {
          printf("Unable to set motor position.\n");
          return 0;
    }

    CPhidget_close(Servo);
    CPhidget_delete(Servo);
    return 0;
}
```

Another Example

This example is a little more complicated.   It shows how to read from a Phidgets, as well as how to set up event handlers.

To set up event handlers, you pass a function pointer to a function you want to have run when that event happens.   You need to continuously poll by calling the _read function to receive data, and when his data changes, it is the read function itself that calls your event handlers.

Note that in a real program, you would put the _read functions into some sort of architecture specific thread or timer, rather then creating an infinite loop.   See the examples for how to do this properly.

```
#include "phidget20.h"

int gotSensor(CPhidgetInterfaceKitHandle phid, void *meh, int ind, int val)
{
    printf("Got Sensor Change: %i\n",val);
    return 0;
}

int main (int argc, const char * argv[])
{
    int something;

    CPhidgetInterfaceKitHandle ifkit;
    CPhidgetInterfaceKit_create(&ifkit);
    CPhidget_open(ifkit, -1);
    CPhidgetInterfaceKit_set_OnSensorChange_Handler(ifkit, &gotSensor, 
&something);

    while(1)
    {
          CPhidgetInterfaceKit_read(ifkit);
          CPhidgetInterfaceKit_read(ifkit);
    }
}
```

Phidget Manager

The Phidget Manager is a useful tool for keeping track of Phidgets attached to a system. 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 your system, but even more useful if you need your program to wait for a specific device to be plugged in after the program is running.

This example uses the Phidget Manager to wait for a PhidgetServo to be connected, then moves it and exits. Note that under MacOS X the Phidget Manager is handled differently; see the Mac Read me for details.

```
#include "phidget20.h"

int gotAttach(CPhidgetManagerHandle phidm, void *, CPhidgetHandle phid)
{
    char *id;
    CPhidgetServoHandle Servo = 0;

    CPhidget_getDeviceType((CPhidgetHandle)phid,&id);
    if(!strcmp(id,"PhidgetServo"))
    {
          CPhidgetServo_create(&Servo);
          CPhidget_open(Servo, -1)
          CPhidgetServo_setMotorPosition(Servo, 0, 90.0)
          CPhidget_close(Servo);
          CPhidget_delete(Servo);
          exit(0);
    }

    return 0;
}

int main()
{
    int something;
    CPhidgetManagerHandle phidlist;

    CPhidgetManager_initialize(&phidlist);
    CPhidgetManager_set_AttachHandler(phidlist, &gotAttach, &something);

    while (1)
    {
          CPhidgetManager_poll(phidlist);
    }
}
```

Calls common to all devices

The following calls are common to all Phidget components:

CPhidget_open (CPhidgetHandle, int SerialNumber);

Attempts to locate a Phidget matching your requirements on the local computer.

The CPhidgetHandle should be a valid handle previously allocated by CPhidget*(device)*_create.

SerialNumber specifies the desired serial number, allowing the call to open a specific Phidget. Specifying -1 for the serial number will cause it to open the first available device.

CPhidget_openRemote (CPhidgetHandle, int SerNum, int ServerID, int portNumber, char* password);

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

The CPhidgetHandle should be a valid handle previously allocated by CPhidget*(device)*_create.

SerNum 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 device.

ServerID specifies the server that the Phidget is attached to. Specifying -1 for the Server ID will cause it to open the first available server, even if that is not the server that contains the requested device. Therefore, -1 should not be used where more then one server is running.

portNumber specifies the port that the server 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.

CPhidget_openRemoteIP (CPhidgetHandle, int SerNum, char *bufferIPAddress, int portNumber, char* password);

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

The CPhidgetHandle should be a valid handle, previously allocated by CPhidget(device)_create.

SerNum 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 device.

bufferIPAddress specifies the server that the Phidget is attached to. This can be the hostname, or the IP address.

portNumber 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.

CPhidget_close (CPhidgetHandle);

Closes the file handles for this device. You can call this while reads and writes are still outstanding; they will fail quickly.

CPhidget_delete (CPhidgetHandle);

Supply CPhidget_delete with a valid CPhidgetHandle, and the memory will be freed. After a device has been deleted, it is no longer safe to make calls on the handle.

CPhidget_getDeviceStatus (CPhidgetHandle, long *);
Returns an integer indicating the status of the device.

CPhidget_getDeviceType (CPhidgetHandle, char **);
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.

CPhidget_getDeviceVersion (CPhidgetHandle, long *);
Returns the device version of this Phidget.

CPhidget_getErrorDescription (int ErrorCode, char **);
Sets the detach handler to be run when the device is unplugged, or closed from software.

0 OK
1 NOTFOUND
2 NOMEMORY
3 UNEXPECTED
4 INVALIDARG
5 NOTATTACHED
6 INTERRUPTED
7 INVALIDERROR
8 NETWORKERROR
9 UNKNOWNVAL

CPhidget_getLibraryVersion (char ** buffer);
Sets a pointer to a null terminated string providing the version number of the API library.

CPhidget_getSerialNumber (CPhidgetHandle, long *);
Returns the unique serial number of this Phidget. This number is set during manufacturing, and is unique across all Phidgets.

CPhidget_set_OnDetach_Handler (CPhidgetHandle, int (*fptr) (CPhidgetHandle, void *), void *userPtr);
Sets the detach handler to be run when the device is unplugged, or closed from software.

CPhidget_getServerID(CPhidgetHandle, long *);
Returns the Server ID for a remote Phidget device.

CPhidget_getServerAddress(CPhidgetHandle, char **IPAddr, int Port);
Returns the IP Address and Port of a remote Phidget device.

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:

CPhidgetAccelerometer_close (CPhidgetAccelerometerHandle);

Deprecated. *As of version 1.0.6, replaced by CPhidget_close(CPhidgetHandle)*

Closes the file handles for this device. You can call this while reads and writes are still outstanding; they will fail quickly.

CPhidgetAccelerometer_create (CPhidgetAccelerometerHandle *);

Allocates a CPhidgetAccelerometerHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the CPhidgetAccelerometer_openRemote or _openRemoteIP function required.

CPhidgetAccelerometer_open (CPhidgetAccelerometerHandle *, int Reserved, int SerialNumber);

Deprecated. *As of version 1.0.6, replaced by CPhidget_open(CPhidgetHandle, int SerialNumber)*

CPhidgetAccelerometer_getAcceleration (CPhidgetAccelerometerHandle, int Index, double *pVal);

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

CPhidgetAccelerometer_getAccelerationChangeTrigger (CPhidgetAccelerometerHandle, int Index, double *pVal);

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

CPhidgetAccelerometer_getNumAxis (CPhidgetAccelerometerHandle, int *pVal);

Get the number of axes available from the PhidgetAccelerometer.

CPhidgetAccelerometer_setAccelerationChangeTrigger (CPhidgetAccelerometerHandle, 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.

CPhidgetAccelerometer_set_OnAccelerationChange_Handler (CPhidgetAccelerometerHandle, int (*fptr) (CPhidgetAccelerometerHandle, void *, int, double), void *)

Registers a callback that will run if the acceleration changes by more than the Acceleration trigger.

Requires the handle for the Accelerometer, the function that will be called, and an arbitrary pointer that will be supplied to the callback function (may be NULL).

CPhidgetAccelerometer_read (CPhidgetAccelerometerHandle);

The Read function is available on Phidgets that can be polled for data. Read is a blocking function that completes after a read against the Phidget completes, or returns with an error. Before completing, any appropriate event handlers will be run in this context. For most applications, a thread is allocated to continuously call this function. Assume that data passed to event handlers will be valid until the application event handler exits.

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:

CPhidgetEncoder_close (CPhidgetEncoderHandle);
Deprecated. *As of version 1.0.6, replaced by CPhidget_close(CPhidgetHandle)*

Closes the file handles for this device. You can call this while reads and writes are still outstanding; they will fail quickly.

CPhidgetEncoder_create (CPhidgetEncoderHandle *);
Allocates a CPhidgetEncoderHandle. Typically this is the first call on a handle, followed by registering the eventhandlers, and calling the CPhidgetEncoder_openRemote or _openRemoteIP function required.

CPhidgetEncoder_open (CPhidgetEncoderHandle *, int Reserved, int SerialNumber);
Deprecated. *As of version 1.0.6, replaced by CPhidget_open(CPhidgetHandle, int SerialNumber)*

Attempts to locate a Phidget matching your requirements on the local computer.

Open is supplied with a CPhidgetEncoderHandle. If the CPhidgetEncoderHandle is NULL it will be allocated, but it is recommended to use CPhidgetEncoder_create to first allocate the handle.

SerialNumber specifies the desired serial number, allowing the call to open a specific Encoder. Specifying -1 for the serial number will cause it to open the first available Encoder.

Reserved - Should be set to zero.

CPhidgetEncoder_getEncoderPosition (CPhidgetEncoderHandle, int Index, int *pVal);
Gets the last position value received from the encoder for the particular encoder selected.

CPhidgetEncoder_getNumEncoders (CPhidgetEncoderHandle, int *pVal);
Gets the number of encoders available on the Phidget.

CPhidgetEncoder_setEncoderPosition (CPhidgetEncoderHandle, int Index, int newVal);
Specifies a new value for the current position of the encoder.

CPhidgetEncoder_getInputState (CPhidgetEncoderHandle, int Index, int *pVal);
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.

CPhidgetEncoder_getNumInputs (CPhidgetEncoderHandle, int *pVal);
Returns the number of inputs on this particular Phidget device. Please see the hardware description for the details of device variations.

CPhidgetEncoder_read (CPhidgetEncoderHandle);
The read function is available on Phidgets that can be polled for data. Read is a blocking function that completes after a read against the Phidget completes, or returns with an error. Before completing, any appropriate event handlers will be run in this context. For most applications, a thread is allocated

to continuously call this function. Assume that data passed to event handlers will be valid until the application event handler exits.

CPhidgetEncoder_set_OnPositionChange_Handler (CPhidgetEncoderHandle, int (*fptr) (CPhidgetEncoderHandle, void *, int, int, int), void *)
Registers a callback that will run if the encoder changes.

Requires the handle for the Encoder, the function that will be called, and an arbitrary pointer that will be supplied to the callback function (may be NULL).

CPhidgetEncoder_set_OnInputChange_Handler (CPhidgetEncoderHandle, int (*fptr) (CPhidgetEncoderHandle, void *, int, int), void *)
Registers a callback that will run if an input changes.

Requires the handle for the Phidget, the function that will be called, and an arbitrary pointer that will be supplied to the callback function (may be NULL).

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:

CPhidgetInterfaceKit_close (CPhidgetInterfaceKitHandle);
Deprecated. *As of version 1.0.6, replaced by CPhidget_close(CPhidgetHandle)*

Closes the file handles for this device. You can call this while reads and writes are still outstanding; they will fail quickly.

CPhidgetInterfaceKit_create (CPhidgetInterfaceKitHandle *);
Allocates a CPhidgetInterfaceKitHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the CPhidgetInterfaceKit_openRemote or _openRemoteIP function required.

CPhidgetInterfaceKit_open (CPhidgetInterfaceKitHandle *, int Reserved, int SerialNumber);
Deprecated. *As of version 1.0.6, replaced by CPhidget_open(CPhidgetHandle, int SerialNumber)*

Attempts to locate a Phidget matching your requirements on the local computer.

Open is supplied with a CPhidgetInterfaceKitHandle. If the CPhidgetInterfaceKitHandle is NULL it will be allocated, but it is recommended to use CPhidgetInterfaceKit_create to first allocate the handle.

SerialNumber specifies the desired serial number, allowing the call to open a specific InterfaceKit. Specifying -1 for the serial number will cause it to open the first available InterfaceKit.

Reserved - Should be set to zero.

CPhidgetInterfaceKit_read (CPhidgetInterfaceKitHandle);
The Read function is available on Phidgets that can be polled for data. Read is a blocking function that completes after a read against the Phidget completes, or returns with an error. Before completing, any appropriate event handlers will be run in this context. For most applications, a thread is allocated to continuously call this function. Assume that data passed to event handlers will be valid until the application event handler exits.

CPhidgetInterfaceKit_getNumSensors (CPhidgetInterfaceKitHandle, int *pVal);
Gets the number of analog inputs available on the given PhidgetInterface Kit.

CPhidgetInterfaceKit_getRatiometric (CPhidgetInterfaceKitHandle, int *pVal);
desc

CPhidgetInterfaceKit_getSensorChangeTrigger (CPhidgetInterfaceKitHandle, int Index, int *pVal);
Gets the amount of change that should exist between the last reported value and the current value before an OnSensorChange event is fired.

CPhidgetInterfaceKit_getSensorNormalizeMaximum (CPhidgetInterfaceKitHandle, int Index, int *pVal);
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 lease a difference of 1 between this property and SensorNormalizeMinimum, otherwise an INVALIDARG error is returned.

CPhidgetInterfaceKit_getSensorNormalizeMinimum (CPhidgetInterfaceKitHandle, int Index, int *pVal);
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 lease a difference of 1 between this property and SensorNormalizeMaximum, otherwise INVALIDARG is returned.

CPhidgetInterfaceKit_getSensorRawValue (CPhidgetInterfaceKitHandle, int Index, int *pVal);
Returns the actual value of the sensor between 0 - 4095, regardless of how SensorNormalizeMaximum or SensorNormalizeMinimum have been set. This is directly proportional to the analog input, ranging from 0-5V.

CPhidgetInterfaceKit_getSensorValue (CPhidgetInterfaceKitHandle, int Index, int *pVal);
Gets the last reported sensor value for the given index.

CPhidgetInterfaceKit_setRatiometric (CPhidgetInterfaceKitHandle, int newVal);
desc

CPhidgetInterfaceKit_setSensorChangeTrigger (CPhidgetInterfaceKitHandle, 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.

CPhidgetInterfaceKit_setSensorNormalizeMaximum (CPhidgetInterfaceKitHandle, int Index, int newVal);
Specifies the maximum setting of a sensor's range.

CPhidgetInterfaceKit_setSensorNormalizeMinimum (CPhidgetInterfaceKitHandle, int Index, int newVal);
Specifies the minimum setting of a sensor's range.

CPhidgetInterfaceKit_getInputState (CPhidgetInterfaceKitHandle, int Index, int *pVal);
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.

CPhidgetInterfaceKit_getNumInputs (CPhidgetInterfaceKitHandle, int *pVal);
Returns the number of inputs on this particular Phidget device. Please see the hardware description for the details of device variations.

CPhidgetInterfaceKit_getNumOutputs (CPhidgetInterfaceKitHandle, int *pVal);
Returns the number of outputs on this particular Phidget device. Please see the hardware description for the details of device variations.

CPhidgetInterfaceKit_getOutputState (CPhidgetInterfaceKitHandle, int Index, int *pVal);
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.

CPhidgetInterfaceKit_setOutputState (CPhidgetInterfaceKitHandle, int Index, int 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.

CPhidgetInterfaceKit_set_OnInputChange_Handler (CPhidgetInterfaceKitHandle, int (*fptr) (CPhidgetInterfaceKitHandle, void *, int, int), void *)
Registers a callback that will run if an input changes.

Requires the handle for the Phidget, the function that will be called, and an arbitrary pointer that will be supplied to the callback function (may be NULL).

CPhidgetInterfaceKit_set_OnSensorChange_Handler (CPhidgetInterfaceKitHandle, int (*fptr) (CPhidgetInterfaceKitHandle, void *, int, int), void *)
Registers a callback that will run if the sensor value changes by more than the OnSensorChange trigger.

Requires the handle for the IntefaceKit, the function that will be called, and an arbitrary pointer that will be supplied to the callback function (may be NULL).

CPhidgetInterfaceKit_set_OnOutputChange_Handler (CPhidgetInterfaceKitHandle, int (*fptr) (CPhidgetInterfaceKitHandle, void *, int, int), void *)
Registers a callback that will run if an output changes.

Requires the handle for the Phidget, the function that will be called, and an arbitrary pointer that will be supplied to the callback function (may be NULL).

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:

CPhidgetLED_close (CPhidgetLEDHandle);
Deprecated. *As of version 1.0.6, replaced by CPhidget_close(CPhidgetHandle)*

Closes the file handles for this device. You can call this while reads and writes are still outstanding; they will fail quickly.

CPhidgetLED_create (CPhidgetLEDHandle *)
Allocates a CPhidgetLEDHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the CPhidgetLED_openRemote or _openRemoteIP function required.

CPhidgetLED_open (CPhidgetLEDHandle *, int Reserved, int SerialNumber);
Deprecated. *As of version 1.0.6, replaced by CPhidget_open(CPhidgetHandle, int SerialNumber)*

Attempts to locate a Phidget matching your requirements on the local computer.

Open is supplied with a CPhidgetLEDHandle. If the CPhidgetLEDHandle is NULL it will be allocated, but it is recommended to use CPhidgetLED_create to first allocate the handle.

SerialNumber specifies the desired serial number, allowing the call to open a specific LED. Specifying -1 for the serial number will cause it to open the first available LED.

Reserved - Should be set to zero.

CPhidgetLED_getDiscreteLED (CPhidgetLEDHandle, int Index, int *pVal);
Gets the brightness of an individual LED. Range of brightness is 0-100.

CPhidgetLED_getNumLEDs (CPhidgetLEDHandle, int *pVal);
Returns the number of LED positions available in this Phidget.

CPhidgetLED_setDiscreteLED (CPhidgetLEDHandle, 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:

CPhidgetMotorControl_close (CPhidgetMotorControlHandle);
Deprecated. *As of version 1.0.6, replaced by CPhidget_close(CPhidgetHandle)*

Closes the file handles for this device. You can call this while reads and writes are still outstanding; they will fail quickly.

CPhidgetMotorControl_create (CPhidgetMotorControlHandle *);
Allocates a CPhidgetMotorControlHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the CPhidgetMotorControl_openRemote or _openRemoteIP function required.

CPhidgetMotorControl_open (CPhidgetMotorControlHandle *, int Reserved, int SerialNumber);
Deprecated. *As of version 1.0.6, replaced by CPhidget_open(CPhidgetHandle, int SerialNumber)*

Attempts to locate a Phidget matching your requirements on the local computer.

Open is supplied with a CPhidgetMotorControlHandle. If the CPhidgetMotorControlHandle is NULL it will be allocated, but it is recommended to use CPhidgetMotorControl_create to first allocate the handle.

SerialNumber specifies the desired serial number, allowing the call to open a specific MotorControl. Specifying -1 for the serial number will cause it to open the first available MotorControl.

Reserved - Should be set to zero.

CPhidgetMotorControl_getInputState (CPhidgetMotorControlHandle, int Index, int *pVal);
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.

CPhidgetMotorControl_getNumInputs (CPhidgetMotorControlHandle, int *pVal);
Returns the number of inputs on this particular Phidget device. Please see the hardware description for the details of device variations.

CPhidgetMotorControl_set_OnInputChange_Handler (CPhidgetMotorControlHandle, int (*fptr)(CPhidgetMotorControlHandle, void *, int, int), void *)
Registers a callback that will run if an input changes.

Requires the handle for the Phidget, the function that will be called, and a arbitrary pointer that will be supplied to the callback function (may be NULL).

CPhidgetMotorControl_getAcceleration (CPhidgetMotorControlHandle, int Index, int *pVal);
desc

CPhidgetMotorControl_getMotorSpeed (CPhidgetMotorControlHandle, int Index, int *pVal);
desc

CPhidgetMotorControl_getTorque (CPhidgetMotorControlHandle, int Index, int *pVal);
desc

CPhidgetMotorControl_setAcceleration (CPhidgetMotorControlHandle, int Index, int newVal);
desc

CPhidgetMotorControl_setMotorSpeed (CPhidgetMotorControlHandle, int Index, int newVal);
desc

CPhidgetMotorControl_set_OnMotorChange_Handler (CPhidgetMotorControlHandle, int (*fptr) (CPhidgetMotorControlHandle, void *, int, int), void *)
desc

CPhidgetMotorControl_set_OnTorqueChange_Handler (CPhidgetMotorControlHandle, int (*fptr) (CPhidgetMotorControlHandle, void *, int, int), void *)
desc

CPhidgetMotorControl_getNumMotors (CPhidgetMotorControlHandle, int *pVal);
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.

CPhidgetMotorControl_read (CPhidgetMotorControlHandle);
The Read function is available on Phidgets that can be polled for data. Read is a blocking function that completes after a read against the Phidget completes, or returns with an error. Before completing, any appropriate event handlers will be run in this context. For most applications, a thread is allocated to continuously call this function. Assume that data passed to event handlers will be valid until the application event handler exits.

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:

CPhidgetRFID_close (CPhidgetRFIDHandle);

Deprecated. *As of version 1.0.6, replaced by CPhidget_close(CPhidgetHandle)*

Closes the file handles for this device. You can call this while reads and writes are still outstanding; they will fail quickly.

CPhidgetRFID_create (CPhidgetRFIDHandle *);

Allocates a CPhidgetRFIDHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the CPhidgetRFID_openRemote or _openRemoteIP function required.

CPhidgetRFID_open (CPhidgetRFIDHandle *, int Reserved, int SerialNumber);

Deprecated. *As of version 1.0.6, replaced by CPhidget_open(CPhidgetHandle, int SerialNumber)*

Attempts to locate a Phidget matching your requirements on the local computer.

Open is supplied with a CPhidgetRFIDHandle. If the CPhidgetRFIDHandle is NULL it will be allocated, but it is recommended to use CPhidgetRFID_create to first allocate the handle.

SerialNumber specifies the desired serial number, allowing the call to open a specific RFID. Specifying -1 for the serial number will cause it to open the first available RFID.

Reserved - Should be set to zero.

CPhidgetRFID_getNumOutputs (CPhidgetRFIDHandle, int *pVal);

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

CPhidgetRFID_getOutputState (CPhidgetRFIDHandle, int Index, int *pVal);

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.

CPhidgetRFID_setOutputState (CPhidgetRFIDHandle, int Index, int 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.

CPhidgetRFID_read (CPhidgetRFIDHandle);

The Read function is available on Phidgets that can be polled for data. Read is a blocking function that completes after a read against the Phidget completes, or returns with an error. Before completing, any appropriate event handlers will be run in this context. For most applications, a thread is allocated to continuously call this function. Assume that data passed to event handlers will be valid until the application event handler exits.

CPhidgetRFID_set_OnOutputChange_Handler (CPhidgetRFIDHandle, int (*fptr) (CPhidgetRFID-Handle, void *, int, int), void *)

Registers a callback that will run if an output changes.

Requires the handle for the Phidget, the function that will be called, and an arbitrary pointer that will be supplied to the callback function (may be NULL).

CPhidgetRFID_set_OnTag_Handler (CPhidgetRFIDHandle, int (*fptr) (CPhidgetRFIDHandle, void *, unsigned char *), void *)

Registers a callback that will run when a Tag is read.

Requires the handle for the PhidgetRFID, the function that will be called, and an arbitrary pointer that will be supplied to the callback function (may be NULL).

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:

CPhidgetServo_close (CPhidgetServoHandle);

Deprecated. *As of version 1.0.6, replaced by CPhidget_close(CPhidgetHandle)*

Closes the file handles for this device. You can call this while reads and writes are still outstanding; they will fail quickly.

CPhidgetServo_create (CPhidgetServoHandle *);

Allocates a CPhidgetServoHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the CPhidgetServo_openRemote or _openRemoteIP function required.

CPhidgetServo_open (CPhidgetServoHandle *, int Reserved, int SerialNumber);

Deprecated. *As of version 1.0.6, replaced by CPhidget_open(CPhidgetHandle, int SerialNumber)*

Attempts to locate a Phidget matching your requirements on the local computer.

Open is supplied with a CPhidgetServoHandle. If the CPhidgetServoHandle is NULL it will be allocated, but it is recommended to use CPhidgetServo_create to first allocate the handle.

SerialNumber specifies the desired serial number, allowing the call to open a specific Servo. Specifying -1 for the serial number will cause it to open the first available Servo.

Reserved - Should be set to zero.

CPhidgetServo_getMotorPosition (CPhidgetServoHandle, int Index, double *pVal);

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

CPhidgetServo_setMotorPosition (CPhidgetServoHandle, 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.

CPhidgetServo_getNumMotors (CPhidgetServoHandle, int *pVal);

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.

CPhidgetServo_read (CPhidgetServoHandle);

The Read function is available on Phidgets that can be polled for data. Read is a blocking function that completes after a read against the Phidget completes, or returns with an error. Before completing, any appropriate event handlers will be run in this context. For most applications, a thread is allocated to continuously call this function. Assume that data passed to event handlers will be valid until the application event handler exits.

CPhidgetServo_set_OnMotorPositionChange_Handler (CPhidgetServoHandle, int (*fptr) (CPhidgetServoHandle, void *, int, double), void *)

Registers a callback that will run when the motor position is changed.

Requires the handle for the Phidget, the function that will be called, and an arbitrary pointer that will be supplied to the callback function (may be NULL).

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:

CPhidgetTemperatureSensor_close (CPhidgetTemperatureSensorHandle);

Deprecated. *As of version 1.0.6, replaced by CPhidget_close(CPhidgetHandle)*

Closes the file handles for this device. You can call this while reads and writes are still outstanding; they will fail quickly.

CPhidgetTemperatureSensor_create (CPhidgetTemperatureSensorHandle *);

Allocates a CPhidgetTemperatureSensorHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the CPhidgetTemperatureSensor_openRemote or _openRemoteIP function required.

CPhidgetTemperatureSensor_open (CPhidgetTemperatureSensorHandle *, int Reserved, int SerialNumber);

Deprecated. *As of version 1.0.6, replaced by CPhidget_open(CPhidgetHandle, int SerialNumber)*

Attempts to locate a Phidget matching your requirements on the local computer.

Open is supplied with a CPhidgetTemperatureSensorHandle. If the CPhidgetTemperatureSensorHandle is NULL it will be allocated, but it is recommended to use CPhidgetTemperatureSensor_create to first allocate the handle.

SerialNumber specifies the desired serial number, allowing the call to open a specific TemperatureSensor. Specifying -1 for the serial number will cause it to open the first available TemperatureSensor.

Reserved - Should be set to zero.

CPhidgetTemperatureSensor_read (CPhidgetTemperatureSensorHandle);

The Read function is available on Phidgets that can be polled for data. Read is a blocking function that completes after a read against the Phidget completes, or returns with an error. Before completing, any appropriate event handlers will be run in this context. For most applications, a thread is allocated to continuously call this function. Assume that data passed to event handlers will be valid until the application event handler exits.

CPhidgetTemperatureSensor_getNumTemperatureInputs (CPhidgetTemperatureSensorHandle, int *pVal);

Returns the integer value of the number of thermocouple inputs.

CPhidgetTemperatureSensor_getTemperature (CPhidgetTemperatureSensorHandle, int Index, double *pVal);
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 temperatrue of the thermocouple.

CPhidgetTemperatureSensor_getTemperatureChangeTrigger (CPhidgetTemperatureSensorHandle, int Index, double *pVal);
Gets the amount of change that should exist between the last reported value and the current value before an OnTemperatureChange event is fired.

CPhidgetTemperatureSensor_setTemperatureChangeTrigger (CPhidgetTemperatureSensorHandle, 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.

CPhidgetTemperatureSensor_set_OnTemperatureChange_Handler (CPhidgetTemperatureSensorHandle, int (*fptr) (CPhidgetTemperatureSensorHandle, void *, int, double), void *)
Registers a callback that will run if the Temperature changes by more than the Temperature trigger.

Requires the handle for the Temperature Sensor, the function that will be called, and a arbitrary pointer that will be supplied to the callback function (may be NULL).

CPhidgetTemperatureSensor_getUseImperial (CPhidgetTemperatureSensorHandle, int *pVal);
Returns the state indicating if Metric (SI units) to Imperial (USA, Myanmar, Liberia) units are being used.

CPhidgetTemperatureSensor_setUseImperial (CPhidgetTemperatureSensorHandle, int newVal);
Specifies the state indicating if Metric (SI units) to Imperial (USA, Myanmar, Liberia) units are being used.

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:

CPhidgetTextLCD_close (CPhidgetTextLCDHandle);
Deprecated. *As of version 1.0.6, replaced by CPhidget_close(CPhidgetHandle)*

Closes the file handles for this device. You can call this while reads and writes are still outstanding; they will fail quickly.

CPhidgetTextLCD_create (CPhidgetTextLCDHandle *);
Allocates a CPhidgetTextLCDHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the CPhidgetTextLCD_openRemote or _openRemoteIP function required.

CPhidgetTextLCD_open (CPhidgetTextLCDHandle *, int Reserved, int SerialNumber);
Deprecated. *As of version 1.0.6, replaced by CPhidget_open(CPhidgetHandle, int SerialNumber)*

Attempts to locate a Phidget matching your requirements on the local computer.

Open is supplied with a CPhidgetTextLCDHandle. If the CPhidgetTextLCDHandle is NULL it will be allocated, but it is recommended to use CPhidgetTextLCD_create to first allocate the handle.

SerialNumber specifies the desired serial number, allowing the call to open a specific TextLCD. Specifying -1 for the serial number will cause it to open the first available TextLCD.

Reserved - Should be set to zero.

CPhidgetTextLCD_getBacklight (CPhidgetTextLCDHandle, int *pVal);
Determines if the backlight for this LCD is on or off.

CPhidgetTextLCD_getCursorBlink (CPhidgetTextLCDHandle, int *pVal);
Determines if the cursor's blinking is on or off.

CPhidgetTextLCD_getCursorOn (CPhidgetTextLCDHandle, int *pVal);
Determines if the cursor is on or off.

CPhidgetTextLCD_setBacklight (CPhidgetTextLCDHandle, int newVal);
Sets the backlight for this LCD on or off.

CPhidgetTextLCD_setCursorBlink (CPhidgetTextLCDHandle, int newVal);
Sets the cursor's blinking on or off.

CPhidgetTextLCD_setCursorOn (CPhidgetTextLCDHandle, int newVal);
Sets the cursor on or off. This cursor is displayed at the last location that was changed.

CPhidgetTextLCD_getNumColumns (CPhidgetTextLCDHandle, int *pVal);
Returns number of columns of text that may be used on the display.

CPhidgetTextLCD_getNumRows (CPhidgetTextLCDHandle, int *pVal);
Returns number of rows of text that may be presented on the display.

CPhidgetTextLCD_setDisplayString (CPhidgetTextLCDHandle, int Row, char *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:

CPhidgetTextLED_close (CPhidgetTextLEDHandle);
Deprecated. *As of version 1.0.6, replaced by CPhidget_close(CPhidgetHandle)*

Closes the file handles for this device. You can call this while reads and writes are still outstanding. They will fail quickly.

CPhidgetTextLED_create (CPhidgetTextLEDHandle *);
Allocates a CPhidgetTextLEDHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the CPhidgetTextLED_openRemote or _openRemoteIP function required.

CPhidgetTextLED_open (CPhidgetTextLEDHandle *, int Reserved, int SerialNumber);
Deprecated. *As of version 1.0.6, replaced by CPhidget_open(CPhidgetHandle, int SerialNumber)*

Attempts to locate a Phidget matching your requirements on the local computer.

Open is supplied with a CPhidgetTextLEDHandle. If the CPhidgetTextLEDHandle is NULL it will be allocated, but it is recommended to use CPhidgetTextLED_create to first allocate the handle.

SerialNumber specifies the desired serial number, allowing the call to open a specific TextLED. Specifying -1 for the serial number will cause it to open the first available TextLED.

Reserved - Should be set to zero.

CPhidgetTextLED_getBrightness (CPhidgetTextLEDHandle, int *pVal);
Returns the brightness value of the LED display.

CPhidgetTextLED_setBrightness (CPhidgetTextLEDHandle, int newVal);
Sets the brightness of the LED display. Varying this property will control the brightness uniformly across all digits in the display.

CPhidgetTextLED_getNumColumns (CPhidgetTextLEDHandle, int *pVal);
Returns number of columns of text that may be used on the display.

CPhidgetTextLED_getNumRows (CPhidgetTextLEDHandle, int *pVal);
Returns number of rows of text that may be presented on the display.

CPhidgetTextLED_setDisplayString (CPhidgetTextLEDHandle, int Row, char *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:

CPhidgetWeightSensor_close (CPhidgetWeightSensorHandle);
Deprecated. *As of version 1.0.6, replaced by CPhidget_close(CPhidgetHandle)*

Closes the file handles for this device. You can call this while reads and writes are still outstanding; they will fail quickly.

CPhidgetWeightSensor_create (CPhidgetWeightSensorHandle *);
Allocates a CPhidgetWeightSensorHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the CPhidgetWeightSensor_openRemote or _openRemoteIP function required.

CPhidgetWeightSensor_open (CPhidgetWeightSensorHandle *, int Reserved, int SerialNumber);
Deprecated. *As of version 1.0.6, replaced by CPhidget_open(CPhidgetHandle, int SerialNumber)*

Attempts to locate a Phidget matching your requirements on the local computer.

Open is supplied with a CPhidgetWeightSensorHandle. If the CPhidgetWeightSensorHandle is NULL it will be allocated, but it is recommended to use CPhidgetWeightSensor_create to first allocate the handle.

SerialNumber specifies the desired serial number, allowing the call to open a specific WeightSensor. Specifying -1 for the serial number will cause it to open the first available WeightSensor.

Reserved - Should be set to zero.

CPhidgetWeightSensor_read (CPhidgetWeightSensorHandle);
The Read function is available on Phidgets that can be polled for data. Read is a blocking function that completes after a read against the Phidget completes, or returns with an error. Before completing, any appropriate event handlers will be run in this context. For most applications, a thread is allocated to continuously call this function. Assume that data passed to event handlers will be valid until the application event handler exits.

CPhidgetWeightSensor_getUseImperial (CPhidgetTemperatureSensorHandle, int *pVal);
Returns the state indicating if Metric (SI units) to Imperial (USA, Myanmar, Liberia) units are being used.

CPhidgetWeightSensor_setUseImperial (CPhidgetWeightSensorHandle, int newVal);
Specifies the state indicating if Metric (SI units) to Imperial (USA, Myanmar, Liberia) units are being used.

CPhidgetWeightSensor_getWeight (CPhidgetWeightSensorHandle, double *pVal);
Returns the current weight in Kilograms or Pounds (depending on UseImperial property).

CPhidgetWeightSensor_getWeightChangeTrigger (CPhidgetWeightSensorHandle, double *pVal);
Gets the amount of change that should exist between the last reported value and the current value before an OnWeightChange event is fired.

CPhidgetWeightSensor_setWeightChangeTrigger (CPhidgetWeightSensorHandle, 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.

CPhidgetWeightSensor_set_OnWeightChange_Handler (CPhidgetWeightSensorHandle, int (*fptr)(CPhidgetWeightSensorHandle, void *, double), void *)

Registers a callback that will run if the Weight changes by more than the Weight trigger.

Requires the handle for the Weight Sensor, the function that will be called, and a arbitrary pointer that will be supplied to the callback function (may be NULL).

PhidgetStepper — Under Revision

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

- Set the desired position for stepper motors.
- Control the velocity and acceleration of each stepper motor.
- Track the predicted position of the stepper motor.

Stepper motors can be rotated continuously, moving in discrete steps.

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

CPhidgetStepper_close (CPhidgetStepperHandle);
Deprecated. *As of version 1.0.6, replaced by CPhidget_close(CPhidgetHandle)*

Closes the file handles for this device. You can call this while reads and writes are still outstanding; they will fail quickly.

CPhidgetStepper_create (CPhidgetStepperHandle *);
Allocates a CPhidgetStepperHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the CPhidgetStepper_openRemote or _openRemoteIP function required.

CPhidgetStepper_open (CPhidgetStepperHandle *, int Reserved, int SerialNumber);
Deprecated. *As of version 1.0.6, replaced by CPhidget_open(CPhidgetHandle, int SerialNumber)*

Attempts to locate a Phidget matching your requirements on the local computer.

Open is supplied with a CPhidgetStepperHandle. If the CPhidgetStepperHandle is NULL it will be allocated, but it is recommended to use CPhidgetStepper_create to first allocate the handle.

SerialNumber specifies the desired serial number, allowing the call to open a specific Stepper. Specifying -1 for the serial number will cause it to open the first available Stepper.

Reserved - Should be set to zero.

CPhidgetStepper_read (CPhidgetStepperHandle);
The Read function is available on Phidgets that can be polled for data. Read is a blocking function that completes after a read against the Phidget completes, or returns with an error. Before completing, any appropriate event handlers will be run in this context. For most applications, a thread is allocated to continuously call this function. Assume that data passed to event handlers will be valid until the application event handler exits.

PhidgetAdvancedServo — Under Revision

The PhidgetAdvancedServo 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:

CPhidgetAdvancedServo_close (CPhidgetAdvancedServoHandle);

Deprecated. *As of version 1.0.6, replaced by CPhidget_close(CPhidgetHandle)*

Closes the file handles for this device. You can call this while reads and writes are still outstanding; they will fail quickly.

CPhidgetAdvancedServo_create (CPhidgetAdvancedServoHandle *);

Allocates a CPhidgetAdvancedServoHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the CPhidgetAdvancedServo_openRemote or _openRemoteIP function required.

CPhidgetAdvancedServo_open (CPhidgetAdvancedServoHandle *, int Reserved, int SerialNumber);

Deprecated. *As of version 1.0.6, replaced by CPhidget_open(CPhidgetHandle, int SerialNumber)*

Attempts to locate a Phidget matching your requirements on the local computer.

Open is supplied with a CPhidgetAdvancedServoHandle. If the CPhidgetAdvancedServoHandle is NULL it will be allocated, but it is recommended to use CPhidgetAdvancedServo_create to first allocate the handle.

SerialNumber specifies the desired serial number, allowing the call to open a specific AdvancedServo. Specifying -1 for the serial number will cause it to open the first available AdvancedServo.

Reserved - Should be set to zero.

CPhidgetAdvancedServo_read (CPhidgetAdvancedServoHandle);

The Read function is available on Phidgets that can be polled for data. Read is a blocking function that completes after a read against the Phidget completes, or returns with an error. Before completing, any appropriate event handlers will be run in this context. For most applications, a thread is allocated to continuously call this function. Assume that data passed to event handlers will be valid until the application event handler exits.