C API Manual

Phidget21

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: 2.1.1

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. Handlers are then registered, and associated with a Phidget device using one of the `CPhidget_open` functions. After the OnAttach event fires, commands can be issued. If the application needs data from the device, event handlers can be used, or `_get` functions can be used to get specific values. 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 Examples

The most important thing to remember is that open returns immediately. This means that you needs to check the device status yourself before using it – otherwise you will end up tying to read / set value on a unattached / un-initialized device. Following are two methods for doing this. Remember that in real code, you should always check error codes.

This example takes advantage of the fact that open returns immediately to continue doing useful things, while at the same time waiting for an attach event using the attach handler. This is most useful in GUI code.

```
#include "phidget21.h"

int attach_handler(CphidgetHandle phid, void *userPtr)
{
	CPhidgetInterfaceKitHandle ifkit = (CPhidgetInterfaceKitHandle)phid;
	//...use the phidget...
}
int main()
{
	CPhidgetInterfaceKitHandle ifkit;
	CPhidgetInterfaceKit_create(&ifkit);
	CPhidget_set_OnAttach_Handler(ifkit, attach_handler, NULL);
	CPhidget_open(ifkit, -1);

	//...Do useful work not involving the phidget...
}
```

This example blocks until the device is attached. This is most useful in synchronous non-GUI code.

```
int main()
{
	long status = 0;
	CPhidgetInterfaceKitHandle ifkit;
	CPhidgetInterfaceKit_create(&ifkit);
	CPhidget_open(ifkit, -1);

	CPhidget_waitForAttachment((CPhidgetHandle)ifkit, 0)

	//...use the phidget...
}
```

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. Before calling CPhidget_open, we recommend registering the event handlers your application will use.

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.

Open is pervasive. What this means is that you can call open on a device before it is plugged in, and keep the device opened across device dis- and re-connections.

Open is Asynchronous. What this means is that open will return immediately – before the device being opened is actually available. What this means is that you need to either poll CPhidget_getDeviceStatus for an attached status, or handle the attach event, in order to wait for the device to become available before trying to use it.

CPhidget_openRemoteIP (CPhidgetHandle, int SerialNumber, const char *Address, int Port, const char *Password);

Attempts to open a connection to a Phidget Webservice, and waits for a phidget on that connection.

The CPhidgetHandle should be a valid handle previously allocated by CPhidget*(device)*_create. Before calling CPhidget_ openRemoteIP, we recommend registering the event handlers your application will use.

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.

Address is the address of the computer running the Phidget Webservice. This can be either an IP address or a hostname.

Port specifies the port of the Webservice on the remote computer.

Password specifies the password, which is required if authentication is active on the Webservice. If authentication is not active, this can be set to NULL.

OpenRemoteIP will block until it connects to the server, which means that when it returns with `EPHIDGET_OK`, this means that the connection is active. If the server is unavailable, this will return an error code. If the connection to a webservice is disrupted while in use, an error event will be thrown, and it is recommended that an error event listener be registered for this reason.

As with the regular open, openRemoteIP is pervasive and asynchronous – so long as the connection to the Webservice remains active.

CPhidget_openRemote (CPhidgetHandle, int SerialNumber, const char *ServerID, const char *Password);

This method is not yet implemented and will return `EPHIDGET_UNSUPPORTED`.

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, int *);
Returns an integer indicating the status of the device. This method can be passed any Phidget device handle, a Phidget manager handle, or a Phidget dictionary handle. These are the possible codes:

```
PHIDGET_ATTACHED        0x1
PHIDGET_NOTATTACHED     0x0
```

CPhidget_getDeviceType (CPhidgetHandle, char **);
Sets a pointer to a null terminated string describing the type of the Phidget. All PhidgetInterfaceKits will return "PhidgetInterfaceKit", PhidgetRFID returns "PhidgetRFID" and so on.

CPhidget_getDeviceName (CPhidgetHandle, char **);
Sets a pointer to a null terminated string describing the name of the Phidget. For example, “Phidget InterfaceKit 8/8/8”, “Phidget InterfaceKit 0/0/4”, etc.

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

CPhidget_getErrorDescription (int ErrorCode, char **);
Returns a pointer to a null terminated string describing the ErrorCode passed. These are the error codes, as returned by all CPhidget functions:

```
EPHIDGET_OK                   0
EPHIDGET_NOTFOUND             1
EPHIDGET_NOMEMORY             2
EPHIDGET_UNEXPECTED                 3
EPHIDGET_INVALIDARG                 4
EPHIDGET_NOTATTACHED                5
EPHIDGET_INTERRUPTED                6
EPHIDGET_INVALID                    7
EPHIDGET_NETWORK                    8
EPHIDGET_UNKNOWNVAL                 9
EPHIDGET_BADPASSWORD                10
EPHIDGET_UNSUPPORTED                11
EPHIDGET_DUPLICATE                  12
EPHIDGET_TIMEOUT                    13
EPHIDGET_OUTOFBOUNDS                14
EPHIDGET_EVENT                      15
EPHIDGET_NETWORK_NOTCONNECTED 16
```

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

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

CPhidget_waitForAttachment (CPhidgetHandle, long milliseconds);
Returns with `EPHIDGET_OK` when the device is available, or with `EPHIDGET_TIMEOUT` if the device is not attached before the timeout expires. Timeouts below about 300ms cannot be trusted because of initialization time, and sometimes an even larger timeout is required – ie. the first time a device is

plugged into a windows machine. A timeout of 0 is infinite. This function can be used in conjunction with (or instead of) an attach handler

CPhidget_set_OnAttach_Handler (CPhidgetHandle, int (*fptr) (CPhidgetHandle, void *), void *userPtr);
Sets the attach handler to be run when the device is discovered.

Requires a handle that the callback will be registered with, the function that will be called, and an arbitrary pointer that will be supplied to the callback function.

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.

Requires a handle that the callback will be registered with, the function that will be called, and an arbitrary pointer that will be supplied to the callback function.

CPhidget_set_OnError_Handler (CPhidgetHandle, int (*fptr) (CPhidgetHandle, void *, int, const char *), void *userPtr);
Sets the error handler to be run when an error occurs. This is used for reporting asynchronous errors – mostly related to opening remote Phidgets.

Requires a handle that the callback will be registered with, the function that will be called, and an arbitrary pointer that will be supplied to the callback function.

The callback function returns a reference to the originating Phidget object, the supplied user defined pointer, the error code and a string describing the error.

CPhidget_getDeviceLabel (CPhidgetHandle, char **);
Returns the Label for the Phidget. The Label is a user programmable serial number stored on the Phidget – it can be used to implement a serial numbering scheme, or describe the functionality of the Phidget in a specific application.

CPhidget_setDeviceLabel(CPhidgetHandle, char *);
Writes a Label – a string up to 10 characters long – to the Phidget associated with this handle. **CPhidget_setLabel** may not be available on all operating systems. Currently it is available on MacOS X, Linux, and Windows CE. Calling this on Windows will return `EPHIDGET_UNSUPPORTED`

CPhidget_getServerID(CPhidgetHandle, const char **);
This method is not yet implemented and will return `EPHIDGET_UNSUPPORTED`.

Returns the Server ID for a remote Phidget device. This should only be called on Phidgets that were opened with openRemote or openRemoteIP.

CPhidget_getServerAddress(CPhidgetHandle, char **IPAddr, int Port);
Returns the IP Address and Port of a remote Phidget device. This should only be called on Phidgets that were opened with openRemote or openRemoteIP.

CPhidget_getServerStatus (CPhidgetHandle, int *);
Returns an integer indicating the connection status of a Webservice. This method can be passed any Phidget device handle, a Phidget manager handle, or a Phidget dictionary handle. These are the possible codes:

```
CONNECTED          0x1
NOTCONNECTED       0x0
```

The Phidget Manager

The Phidget manager is an interface that allows for monitoring of all phidgets connected to a system, without opening them.

CPhidgetManager_create (CPhidgetManagerHandle *);
Allocates a CPhidgetManagerHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the variant of _**open** required.

CPhidgetManager_open (CPhidgetManagerHandle);
Opens a connection to the local Phidget Manager.

The CPhidgetManagerHandle should be a valid handle previously allocated by CphidgetManager_create. Before calling CPhidgetManager_open, we recommend registering the event handlers your application will use.

CPhidgetManager_openRemoteIP (CPhidgetManagerHandle, const char *Address, int Port, const char *Password);
Opens a connection to a remote Phidget Manager.

The CPhidgetManagerHandle should be a valid handle previously allocated by CphidgetManager_create. Before calling CPhidgetManager_open, we recommend registering the event handlers your application will use.

Address is the address of the computer running the Phidget Webservice. This can be either an IP address or a hostname.

Port specifies the port of the Webservice on the remote computer.

Password specifies the password, which is required if authentication is active on the Webservice. If authentication is not active, this can be set to NULL.

OpenRemoteIP will block until it connects to the server, which means that when it returns with `EPHIDGET_OK`, this means that the connection is active. If the server is unavailable, this will return an error code. If the connection to a webservice is disrupted while in use, an error event will be thrown, and it is recommended that an error event listener be registered for this reason.

CPhidgetManager_openRemote (CPhidgetManagerHandle, const char *ServerID, const char *Password);
This method is not yet implemented and will return `EPHIDGET_UNSUPPORTED`.

CPhidgetManager_close (CPhidgetManagerHandle);
Closes the file handles for this device. You should always call this when finished with a Manager.

CPhidgetManager_delete (CPhidgetManagerHandle);
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.

CPhidgetManager_set_OnAttach_Handler (CPhidgetManagerHandle, int (*fptr) (CPhidgetHandle, void *), void *userPtr);
Sets the attach handler to be run when and new device is discovered.

Requires a handle that the callback will be registered with, the function that will be called, and an arbitrary pointer that will be supplied to the callback function.

CPhidgetManager_set_OnDetach_Handler (CPhidgetManagerHandle, int (*fptr) (CPhidgetHandle, void *), void *userPtr);

Sets the detach handler to be run when any device is unplugged.

Requires a handle that the callback will be registered with, the function that will be called, and an arbitrary pointer that will be supplied to the callback function.

CPhidgetManager_set_OnError_Handler (CPhidgetManagerHandle, int (*fptr) (CPhidgetManager-Handle, void *, int, const char *), void *userPtr);

Sets the error handler to be run when an error occurs. This is used for reporting asynchronous errors – mostly related to opening remote Phidgets.

Requires a handle that the callback will be registered with, the function that will be called, and an arbitrary pointer that will be supplied to the callback function.

The callback function returns a reference to the originating Phidget object, the supplied user defined pointer, the error code and a string describing the error.

CPhidgetManager_getAttachedDevices (CPhidgetManagerHandle, CPhidgetHandle *phidArray[], int *count);

Allocates an array of CPhidgetHandles big enough to hold all of the devices currently connected, and sets the given pointer to this array. Also sets the count pointer to the size of the array. This array should be freed using free() when it's no longer needed.

The Phidget Dictionary

The Phidget Dictionary is a service provided by the Phidget Webservice. The Webservice maintains a centralized dictionary of key-value pairs that can be accessed and changed from any number of clients through the CPhidgetDictionary interface available in phidget21.

Note that the Webservice uses this dictionary to control access to Phidgets through the openRemote and openRemoteIP interfaces, and as such, you should never add or modify a key that starts with /PSK/ or /PCK/, unless you want to explicitly modify Phidget specific data – and this is highly discouraged, as it's very easy to break things. Listening to these keys is fine if so desired.

The intended use for the dictionary is as a central repository for communication and persistent storage of data between several client applications. As an example - a higher level interface exposed by one application – which controls the Phidgets, for others to access – rather then every client talking directly to the Phidgets themselves.

The dictionary makes use of extended regular expressions for key matching. See the end of this document for the rules of regular expressions.

CPhidgetDictionary_create (CPhidgetDictionaryHandle *);

Allocates a CPhidgetDictionaryHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the variant of _**openRemote** required.

CPhidgetDictionary_openRemoteIP (CPhidgetDictionaryHandle, const char *Address, int Port, const char *Password);

Opens a connection to a remote Phidget Dictionary.

The CPhidgetDictionaryHandle should be a valid handle previously allocated by CPhidgetDictionary_create. Before calling CPhidgetDictionary_open, we recommend registering the event handlers your application will use.

Address is the address of the computer running the Phidget Webservice. This can be either an IP address or a hostname.

Port specifies the port of the Webservice on the remote computer.

Password specifies the password, which is required if authentication is active on the Webservice. If authentication is not active, this can be set to NULL.

OpenRemoteIP will block until it connects to the server, which means that when it returns with `EPHIDGET_OK`, this means that the connection is active. If the server is unavailable, this will return an error code. If the connection to a webservice is disrupted while in use, an error event will be thrown, and it is recommended that an error event listener be registered for this reason.

CPhidgetDictionary_openRemote (CPhidgetDictionaryHandle, const char *ServerID, const char *Password);

This method is not yet implemented and will return `EPHIDGET_UNSUPPORTED`.

CPhidgetDictionary_close (CPhidgetDictionaryHandle);

Closes the file handles for this device. You should always call this when finished with a Dictionary.

CPhidgetDictionary_delete (CPhidgetDictionaryHandle);

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.

CPhidgetDictionary_set_OnError_Handler (CPhidgetDictionaryHandle, int (*fptr) (CPhidgetDictionaryHandle, void *, int, const char *), void *userPtr);

Sets the error handler to be run when an error occurs. This is used for reporting asynchronous errors – mostly related to opening remote Phidgets.

Requires a handle that the callback will be registered with, the function that will be called, and an arbitrary pointer that will be supplied to the callback function.

The callback function returns a reference to the originating Phidget object, the supplied user defined pointer, the error code and a string describing the error.

CPhidgetDictionary_addKey(CPhidgetDictionaryHandle dict, const char *key, const char *val, int persistent);

Adds a new key to the Dictionary, or modifies the value of an existing key.

The key can only contain numbers, letters, '/', '.', '-', '_', and must begin with a letter, '_' or '/'.

The value can contain any value.

The persistent value controls whether a key will stay in the dictionary after the client that created it disconnects. If persistent == 0, the key is removed when the connection closes. Otherwise the key remains in the dictionary until it is explicitly removed.

CPhidgetDictionary_removeKey(CPhidgetDictionaryHandle dict, const char *pattern);

Removes a key, or set of keys, from the Dictionary.

The key name is a regular expressions pattern, and so care must be taken to only have it match the specific keys you want to remove.

CPhidgetDictionary_set_OnKeyChange_Handler(CPhidgetDictionaryHandle dict, CPhidgetDictionaryListenerHandle *dictlistener, const char *pattern, int(*fptr)(CPhidgetDictionaryHandle dict, void *userPtr, const char *key, const char *val, CPhidgetDictionary_keyChangeReason reason), void *userPtr);

Sets an events handler for a specific pattern of keys. This events will be called on key add, remove, and change. It will also fire once to give an initial key value as soon as it is registered, if the key already exists.

Requires a handle that the callback will be registered with, the function that will be called, and an arbitrary pointer that will be supplied to the callback function.

DictListener is a handle that will allow you to later cancel this listener using CPhidgetDictionary_remove_OnKeyChange_Handler. If you don't intend to cancel the listener, you can set this to NULL.

Pattern is a regular expression that matches the keys you want to listen for.

The key name is a regular expressions pattern, and so care must be taken to only have it match the specific keys you want to remove.

Reason in the callback function contains the reason for the callback – key added, kay removed, key changed, or current value.

CPhidgetDictionary_remove_OnKeyChange_Handler(CPhidgetDictionaryListenerHandle keylistener);

Removes a key listener that has been previously registered. Just pass in the keyListener handle that was set by CPhidgetDictionary_set_OnKeyChange_Handler. No more events will be received.

CPhidgetDictionary_getKey(CPhidgetDictionaryHandle dict, const char *key, const char *val, int valsize);

This method is not yet implemented and will return `EPHIDGET_UNSUPPORTED`.

Note that currently, the only way to get a key value is using the key event listener.

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. The product is available as a dual axis or a 3-axis module. With this component, the programmer can:

- Measure up to 5 Gravity (9.8 m/s2) change per axis, depending on unit purchased.
- Measures both dynamic acceleration (e.g., vibration) and static acceleration (e.g., gravity or tilt) on 2 or 3 axis.

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

CPhidgetAccelerometer_create (CPhidgetAccelerometerHandle *);
Allocates a CPhidgetAccelerometerHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the variant of _**open** required.

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).

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_create (CPhidgetEncoderHandle *);
Allocates a CPhidgetEncoderHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the variant of **_open** required.

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_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_create (CPhidgetInterfaceKitHandle *);
Allocates a CPhidgetInterfaceKitHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the variant of _**open** required.

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. To receive all events, set the SensorChangeTrigger to zero.

CPhidgetInterfaceKit_getSensorRawValue (CPhidgetInterfaceKitHandle, int Index, int *pVal);
Returns the actual value of the sensor between 0 - 4095. 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);
Some of the PhidgetInterfaceKits with Analog Inputs have to ability to measure voltage by comparing to a fixed 5 Volt reference, or comparing to the voltage supplied to the sensor. If your sensor operates as a voltage divider, or the output is described as ratiometric, SensorValue will be most accurate if Ratiometric is set to true.

If your sensor is described as non-ratiometric, set Ratiometric to false.

Sensors manufactured by Phidgets Inc. will typically have a small **R** printed on them if they are Ratiometric, or **N** if they are non-ratiometric.

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_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 InterfaceKit, 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_create (CPhidgetLEDHandle *)
Allocates a CPhidgetLEDHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the variant of _**open** required.

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. This property does not return the number of LEDs actually attached.

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_create (CPhidgetMotorControlHandle *);
Allocates a CPhidgetMotorControlHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the variant of _**open** required.

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.

CPhidgetMotorControl_getAcceleration (CPhidgetMotorControlHandle, int Index, double *pVal);
Gets the current acceleration.

CPhidgetMotorControl_getMotorSpeed (CPhidgetMotorControlHandle, int Index, double *pVal);
Gets the current motor speed.

CPhidgetMotorControl_setAcceleration (CPhidgetMotorControlHandle, int Index, double newVal);
Sets the acceleration. Valid values are 0-100.

CPhidgetMotorControl_setMotorSpeed (CPhidgetMotorControlHandle, int Index, double newVal);
Sets the Motor Speed. Valid values are (–100 – 100).

CPhidgetMotorControl_set_OnMotorChange_Handler (CPhidgetMotorControlHandle, int (*fptr) (CPhidgetMotorControlHandle, void *, int, double), void *)
Registers an event handler for changes in motor speed.

CPhidgetMotorControl_set_OnCurrentChange_Handler (CPhidgetMotorControlHandle, int (*fptr) (CPhidgetMotorControlHandle, void *, int, double), void *)
Registers an event handler for changes in motor current consumption. Note that this feature is not supported on all motor controllers.

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.

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 liquid with a pH sensor.

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

CPhidgetPHSensor_create (CPhidgetPHSensorHandle *);
Allocates a CPhidgetPHSensorHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the variant of _**open** required.

CPhidgetPHSensor_getPH (CPhidgetPHSensorHandle, double *pVal);
Returns the current pH.

CPhidgetPHSensor_getPotential (CPhidgetPHSensorHandle, double *pVal);
Returns the current potential in volts. Range is 0-5v, 2.5v corresponds to a pH of 7.0.

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

CPhidgetPHSensor_setPHChangeTrigger (CPhidgetPHSensorHandle, 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.

CPhidgetPHSensor_set_OnPHChange_Handler (CPhidgetPHSensorHandle, int (*fptr) (CPhidgetPHSensorHandle, void *, double), void *)
Registers a callback that will run if the pH changes by more than the PH trigger.

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

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_create (CPhidgetRFIDHandle *);
Allocates a CPhidgetRFIDHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the variant of _**open** required.

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. Please see the hardware description for details on different outputs.

CPhidgetRFID_setAntennaOn (CPhidgetRFIDHandle, int newVal);
Sets the state of the antenna to True or False.

CPhidgetRFID_setLEDOn (CPhidgetRFIDHandle, int newVal);
Sets the state of the LED to True or False.

CPhidgetRFID_getAntennaOn (CPhidgetRFIDHandle, int *pVal);
Sets the state of the antenna.

CPhidgetRFID_getLEDOn (CPhidgetRFIDHandle, int *pVal);
Gets the state of the LED.

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 an RFID 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).

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

Registers a callback that will run when an RFID Tag is removed from the field. If an RFID Tag was recently identified, and no valid tag is found in the next 500 milliseconds, the OnTagLost handler will be called.

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_create (CPhidgetServoHandle *);
Allocates a CPhidgetServoHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the variant of _**open** required.

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_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.

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

CPhidgetTemperatureSensor_create (CPhidgetTemperatureSensorHandle *);
Allocates a CPhidgetTemperatureSensorHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the variant of _**open** required.

CPhidgetTemperatureSensor_getNumTemperatureInputs (CPhidgetTemperatureSensorHandle, int *pVal);
Returns the number of temperature 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).

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_create (CPhidgetTextLCDHandle *);
Allocates a CPhidgetTextLCDHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the variant of _**open** required.

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.

CPhidgetTextLCD_setDisplayCharacter (CPhidgetTextLCDHandle, int Row, int Column, char displayCharacter);
Sets the character to display at a particular location on the display.

CPhidgetTextLCD_setCustomCharacter (CPhidgetTextLCDHandle, int Index, char *characterCode);
Sets a custom character. With existing hardware, there is space for 8 custom characters : Index can range from 8 to 15. Each character is described by a set of integers. For more information, have a look at the TextLCD example in the examples.zip for Phidget21.

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_create (CPhidgetTextLEDHandle *);
Allocates a CPhidgetTextLEDHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the variant of _**open** required.

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_create (CPhidgetWeightSensorHandle *);
Allocates a CPhidgetWeightSensorHandle. Typically this is the first call on a handle, followed by registering the event handlers, and calling the variant of _**open** required.

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).

Regular Expressions:

. Matches any single character. Into [ ] this character has its habitual meaning.

[ ] Matches a single character that is contained within the brackets. For example, [abc] matches "a", "b", or "c". [a-z] matches any lowercase letter. These can be mixed: [abcq-z] matches a, b, c, q, r, s, t, u, v, w, x, y, z, and so does [a-cq-z].

The '-' character should be literal only if it is the last or the first character within the brackets: [abc-] or [-abc]. To match an '[' or ']' character, the easiest way is to make sure the closing bracket is first in the enclosing square brackets: [][ab] matches ']', '[', 'a' or 'b'.

[^] Matches a single character that is not contained within the brackets. For example, [^abc] matches any character other than "a", "b", or "c". [^a-z] matches any single character that is not a lowercase letter. As above, these can be mixed.

^ Matches the start of the line (or any line, when applied in multiline mode)

$ Matches the end of the line (or any line, when applied in multiline mode)

( ) Defines a "marked subexpression". What the enclosed expression matched can be recalled later. See the next entry, *n*. Note that a "marked subexpression" is also a "block". Note that this is not found in some instances of regex.

n Where *n* is a digit from 1 to 9; matches what the *n*th marked subexpression matched. This construct is theoretically **irregular** and has not been adopted in the extended regular expression syntax.

* A single character expression followed by "*" matches zero or more copies of the expression. For example, "[xyz]*" matches "", "x", "y", "zx", "zyx", and so on.

*n**, where *n* is a digit from 1 to 9, matches zero or more iterations of what the *n*th marked subexpression matched. For example, "(a.)c\1*" matches "abcab" and "abcabab" but not "abcac".

An expression enclosed in "\(" and "\)" followed by "*" is deemed to be invalid. In some cases (e.g. /usr/bin/xpg4/grep of SunOS 5.8), it matches zero or more iterations of the string that the enclosed expression matches. In other cases (e.g. /usr/bin/grep of SunOS 5.8), it matches what the enclosed expression matches, followed by a literal "*".

+ A single character expression followed by "+" matches one or more copies of the expression. For example, "[xyz]+" matches "x", "y", "zx", "zyx", and so on.

n+, where *n* is a digit from 1 to 9, matches one or more iterations of what the *n*th marked subexpression matched.

An expression enclosed in "\(" and "\)" followed by "+" is deemed to be invalid.

{*x*,*y*} Match the last "block" at least *x* and not more than *y* times. For example, "a\{3,5\}" matches "aaa", "aaaa" or "aaaaa". Note that this is not found in some instances of regex.

Extended Regular Expressions:

+	Match the last "block" one or more times - "ba+" matches "ba", "baa", "baaa" and so on
?	Match the last "block" zero or one times - "ba?" matches "b" or "ba"

| The choice (or set union) operator: match either the expression before or the expression after the operator - "abc|def" matches "abc" or "def".

Also, backslashes are removed: \{...\} becomes {...} and \(...\) becomes (...). Examples:

"[hc]+at" matches with "hat", "cat", "hhat", "chat", "hcat", "ccchat" etc.

"[hc]?at" matches "hat", "cat" and "at"

"([cC]at)|([dD]og)" matches "cat", "Cat", "dog" and "Dog"

Since the characters '(', ')', '[', ']', '.', '*', '?', '+', '^' and '$' are used as special symbols they have to be escaped if they are meant literally. This is done by preceding them with '\' which therefore also has to be escaped this way if meant literally. Examples:

"a\.(\(|\))" matches with the string "a.)" or "a.("