COM API Manual

Describes the Component Object Model (COM) Application Program Interface (API) for each Phidget device.

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 COM API library makes Phidgets very natural to use from Visual Basic. Phidgets have an interface very similar to the GUI widgets that you are used to programming with.

Installing the Library

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

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

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

Now, to use Phidgets from Visual Basic open Project menu >> References, select Phidget Library, plug in your hardware, and you are ready to go!

Basic Example

This is a very simple example.

```
' This code shows just how easy it is to connect to a PhidgetServo
' and change the position of a servo motor

' In Visual Basic open a blank form and use this code to connect to
' the PhidgetServo

Dim WithEvents Servo As PhidgetServo
Private Sub Form_Load()                 'When the program opens
    Set Servo = New PhidgetServo        'Define the variable "Servo" as a
PhidgetServo

    Call Servo.Open(True)               'Connect to the PhidgetServo and lock
access to it

    If (Servo.IsAttached = True) Then   'Check to see that a PhidgetServo was found
        Servo.MotorPosition(0) = 90     'This is where the angle of the Servo is set
        MsgBox "Success!"
    Else
        MsgBox "We were not able to find a PhidgetServo"
                                              'When no PhidgetServo is found
    End If
End Sub                                       'That's all
```

Software common to each device

The following software is common to all Phidget components:

PhidgetManager

PhidgetManager will notify an application when new Phidget devices have been plugged into the computer, and when they disappear. A reference to an IPhidget component suitable for that device will be passed back to the application, allowing the programmer to find out what kind of device was attached. 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.

IPhidget

The IPhidget is the generalized Phidget instance that is returned by the PhidgetManager. Its type can be queried, and the programmer can then assign it to a particular Phidget objects that can be used to control the device in an appropriate way.

Properties

Property:	**DeviceType () as String**
Access:	Get By Value
Description:	Returns the name of the device, e.g., "PhidgetEncoder".

Property:	**DeviceVersion() as Single**
Access:	Get By Value
Description:	Returns the version number of the device.

Property:	**InputState(Index as Long) as Boolean**
Access:	Get By Value
Description:	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.

Property:	**IsAttached As Boolean**
Access:	Get By Value

Description:	This property indicates there is a USB Phidget associated with this component.
Property:	**LibraryVersion() as String**
Access:	Get By Value
Description:	Returns the version of the DLL and COM Library installed.
Property:	**NumInputs as Long**
Access:	Get By Value
Description:	The number of inputs on this particular I/O device. Please see the hardware descriptions for the details of device variations.
Property:	**SerialNumber() as Long**
Access:	Get By Value
Description:	Returns a unique serial number, e.g., 1201, which identifies a specific hardware device.
Property:	**Tag as Variant**
Access:	Get, Put By Value
Description:	Returns / Sets a user-defined piece of arbitrary data associated with this Phidget instance. What would you use this for? Your own serial numbering scheme for your application.

Methods

Method:	**Open(Locked As Boolean, [SerialNumber As Long])**
Description:	Open can be called to connect to a specific device without using the PhidgetManager. The Locked parameter, if true, will attempt to open the device exclusively - the Phidget will not be accessible to other applications. The optional SerialNumber parameter allows you to open a unique Phidget, if you know its serial number.

Events

Event:	**OnDetach()**

Description: Occurs when the Phidget is detached from the bus. This allows the application to adapt to the loss of the physical device.

Event: OnError(ByRef Description as BSTR, ByVal SCODE as Long)

Description: Occurs when the Phidget Manager is unable to communicate with the Phidget. This error is used when a call fails unexpectedly, typically when a device is detached but the OnDetach event has not yet been fired. SCODE can take the following values:

SCODE	Error Description
100	Device is not being maintained by the Phidget Manager. This error will occur after the USB Phidget has been detached and the API continues to send commands. Not Serious.
101	Attempt to write to the Phidget failed. This error can occur after the USB Phidget is detached and before the Phidget Manager has noticed. It is not unusual to get several of these errors when a device is detached.
102	Attempt to read the Phidget failed. This error can occur when the device is read after it is detached but before the Phidget Manager noticed. It is not unusual to get several of these errors when a device is detached.
111	Unknown Phidget COM Class.
112	Error Changing Phidget Interface.
113	Unable To Open New Phidget.
114	Unexpected Error: Unknown Phidget Detaching.

Event: OnInputChange(ByVal Index as Long, ByVal NewState as Boolean)

Description: This event is triggered when the input's state is changed. The number of the particular input is returned, as well as the state of the input: True is On, False is Off. In the case of a switch (button) implementation, True would correspond to when the switch is pressed.

Specific devices

Each Phidget component is described along with its API.

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.

A reference to the actual device can obtained be through the PhidgetManager, or with the Open method on the software component.

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

Property:	**EncoderPosition() as Long**
Access:	Get, Put By Value
Description:	Returns / Sets the position of the encoder shaft. This property will behave differently for absolute and incremental encoders. An absolute encoder will vary this property from 0 to (CPR - 1), wrapping with each revolution. An incremental encoder will increment or decrement this property based on the direction of rotation.

Property:	**NumEncoders() as Long**
Access:	Get By Value
Description:	Returns the number of encoders supported by this device.

Event:	**OnPositionChange(Index as Long, Time as Long, EncoderDisplacement as Long)**
Description:	Occurs when the encoder shaft is rotated. EncoderDisplacement indicates the change since the last OnPositionChange event was fired, and may be positive (clockwise) or negative. Time is an estimate, in milliseconds, of

the time since the last event was fired. EncoderDisplacement and Time may be used to calculate a rough estimate of velocity.

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.

A reference to the actual device can be obtained through the PhidgetManager, or with the Open method on the software component.

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

Property:	**NumOutputs as Long**
Access:	Get By Value
Description:	The number of outputs on this particular I/O device. Please see the hardware description for the details of device variations.

Property:	**NumSensors as Long**
Access:	Get By Value
Description:	The number of sensors on this particular I/O device. Please see the hardware description for the details of device variations.

Property:	**SensorChangeTrigger(Index as Long) as Long**
Access:	Get, Put By Value
Description:	Sets/Returns the amount of change that should exist between a sensor's last reported value and the current value before an OnSensorChange event is fired.

Property:	**SensorNormalizeMaximum (Index as Long) as Long**
Access:	Get, Put By Value
Description:	Sets/Returns 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 E_INVALIDARG is returned.
Property:	**SensorNormalizeMinimum (Index as Long) as Long**
Access:	Get, Put By Value
Description:	Sets/Returns 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 E_INVALIDARG is returned.
Property:	**SensorRawValue(Index as Long) as Long**
Access:	Get By Value
Description:	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.
Property:	**SensorValue(Index as Long) as Long**
Access:	Get By Value
Description:	Returns the digital value being detected, converted to fit between 0-1000. The raw values within the range of SensorNormalizeMaximum and SensorNormalizeMinimum are normalized to 0 - 1000.
Event:	**OnInputChange(ByVal Index as Long, ByVal NewState as Boolean)**
Description:	This event is triggered when the input's state is changed. The number of the particular input is returned, as well as the state of the input: True is

On, False is Off. In the case of a switch (button) implementation, True would correspond to when the switch is pressed.

Event: **OnOutputChange(ByVal Index as Long, ByVal NewState as Boolean)**

Description: This event is triggered when the output state is changed. The number of the particular output is returned, as well as the state of the output: True is On, False is Off. In the case of an LED implementation, True would correspond to the LED being turned on.

Event: **OnSensorChange(ByVal Index as Long, ByVal SensorValue as Long)**

Description: This event is triggered when the sensor's value is changed from the last reading by plus or minus the SensorChangeTrigger value. The number of the particular sensor is returned, as well as the sensor's value. By default, this value is a number between 0 and 1000. However, different sensors may return values within a subset of that range, i.e., there is no guarantee that the minimum or maximum value returned by a sensor is 0 or 1000.

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.

A reference to the actual device can obtained through the PhidgetManager, or with the Open method on the software component.

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

Property:	**MotorPosition(Index as Long) as Single**
Access:	Get, Put By Value
Description:	Sets/Returns 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. MotorPosition will return E_INVALIDARG if the value is not between -23 and 231 or if the index is not a valid servo. Note that motor position requests are not cumulative. That is, let us say the current actual motor position is 0 and you do the following: Servo.MotorPosition(1) = 180 Servo.MotorPostion(1) = 0 The motor will not rotate all the way to 180 before returning back to 0. Rather, it will likely just begin rotating based on the first call, but will immediately return to 0 from its current position when it gets the second call. All you would see (perhaps) is a very slight twitch. If you wanted a cascading effect, you would likely have to use a timer to guarantee that the previous rotation was completed.
Property:	**NumMotors () as Long**

Access: Get By Value

Description: The maximum number of motors on this particular Phidget device. 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.

Event: OnPositionChanged(Index as Long, Position as Long)

Description: Occurs when the servo motor is changing its position.

Note that this event is raised as a side effect of programmatically setting the MotorPosition property, i.e., it is the final resting position requested. Thus it may not match the actual current position of the actual motor (e.g., because of cascading requests as described in the MotorPosition property, or because the motor is still turning).

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.

A reference to the actual device can be obtained through the PhidgetManager, or with the Open method on the software component.

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

Property:	**OutputState(Index as Long) as Boolean**
Access:	Get, Put By Value
Description:	Sets/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 the details of device variations.

Event:	**OnTag(Index As Long, TagNumber As String)**
Description:	Occurs when a valid tag is read by a reader. Index indicates which reader recognized the tag. TagNumber is the unique ID read from the tag.

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.

A reference to the actual device can obtained through the PhidgetManager, or with the Open method on the software component.

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

Property:	**Backlight as Boolean**
Access:	Get, Put By Value
Description:	Turns the backlight for this LCD on or off.
Property:	**CursorBlink as Boolean**
Access:	Get, Put By Value
Description:	Sets the cursor's blinking on or off.
Property:	**CursorOn as Boolean**
Access:	Get, Put By Value
Description:	Turns the cursor on or off. This cursor is displayed at the last location that was changed.
Property:	**DisplayString(Index as Long) as String**
Access:	Put By Value
Description:	Sets the text to display on a particular row of the display. Text will be clipped at the right edge of the display.
Property:	**NumColumns as Long**
Access:	Get By Value
Description:	Returns number of columns of text that may be used on the display.
Property:	**NumRows as Long**

Access:	Get By Value
Description:	Returns number of rows of text that may be presented on the display.

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.

A reference to the actual device can be obtained through the PhidgetManager, or with the Open method on the software component.

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

Property:	**DiscreteLED(Index As Long) As Long**
Access:	Get, Put By Value
Description:	Set the brightness of an individual LED. Range of brightness is 0-100.

Property:	**NumLEDs As Long**
Access:	Get By Value
Description:	Indicates how many LEDs are supported by this controller.

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.

A reference to the actual device can be obtained through the PhidgetManager, or with the Open method on the software component.

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

Property:	**Brightness As Long**
Access:	Get, Put By Value
Description:	Controls the brightness of the LED display. Varying this property will control the brightness uniformly across all digits in the display.
Property:	**DisplayRawData() as String**
Access:	Put By Value
Description:	Sets raw binary data, with each bit corresponding to an LED. This property will be replaced.
Property:	**DisplayString(Index as Long) as String**
Access:	Put By Value
Description:	Sets the text to display on a particular row of the display. Text will be clipped at the right edge of the display.
Property:	**NumColumns as Long**
Access:	Get By Value
Description:	Returns number of columns of text that may be presented on the display.
Property:	**NumRows as Long**

Access:	Get By Value
Description:	Returns number of rows of text that may be presented on the display.

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/s^2) change per axis.
- Measure both dynamic acceleration (e.g., vibration) and static acceleration (e.g., gravity or tilt).

A reference to the actual device can be obtained through the PhidgetManager, or with the Open method on the software component.

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

Property:	**Acceleration(Index as Long) as Double**
Access:	Get By Value
Description:	Gets the last reported acceleration for this axis, selected by Index.
Property:	**AccelerationChangeTrigger(Index as Long) as Double**
Access:	Get, Put By Value
Description:	Sets/gets the change in acceleration required before the OnAccelerationChange event will fire. If this property is set to zero, the event will fire every time there is an update from the device.
Property:	**NumAxis As Long**
Access:	Get by Value
Description:	Returns the number of axis that this accelerometer measures acceleration on. Each axis is measured and reported independently.
Event:	**OnAccelerationChange(Index as Long, Acceleration as Double)**
Description:	Fires when an axis is changing.

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.

A reference to the actual device can be obtained through the PhidgetManager, or with the Open method on the software component.

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

Property:	**UseImperial As Boolean**
Access:	Get, Put By Value
Description:	Changes from Metric (SI units) to Imperial (USA, Myanmar, Liberia).
Property:	**Weight As Double**
Access:	Get By Value
Description:	Returns the current weight in Kilograms or Pounds (depending on UseImperial property).
Property:	**WeightChangeTrigger as Double**
Access:	Get/Put By Value
Description:	Sets/Returns the amount of change that should exist between the last reported weight and the current weight before an OnWeightChange event is fired.
Event:	**OnWeightChange(Weight As Double)**
Description:	Fires when the value received from the PhidgetWeightSensor has changed by an amount greater than it's WeightChangeTrigger property.

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.

A reference to the actual device can be obtained through the PhidgetManager, or with the Open method on the software component.

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

Property:	**NumTemperatureInputs() As Long**
Access:	Get By Value
Description:	Indicates how many thermocouple inputs are supported on this controller.

Property:	**Temperature(Index As Long) As Double**
Access:	Get By Value
Description:	Returns the current temperature in Celsius or Fahrenheit (depending on UseImperial property). Index = 0 returns the temperature of the cold junction. Index = 1 returns the temperature of the thermocouple.

Property:	**TemperatureChangeTrigger**
Access:	Get, Put By Value
Description:	Sets/Returns the amount of change that should exist between the last reported temperature and the current temperature before an OnTemperatureChange event is fired.

Property:	**UseImperial As Boolean**
Access:	Get, Put By Value
Description:	Changes from Metric (SI units) to Imperial (USA, Myanmar, Liberia).

Event:	**OnTemperatureChange(Index As Long, Temperature As Double)**
Description:	Fires when a temperature input has changed by an amount greater than its TemperatureChangeTrigger property.

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.

A reference to the actual device can be obtained through the PhidgetManager, or with the Open method on the software component.

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

Property:	**Acceleration(Index As Long) As Long**
Access:	Get, Put By Value
Description:	This controls the rate of ramping.
Property:	**MotorSpeed(Index as Long) As Long**
Access:	Get, Put By Value
Description:	Sets the amount of power to apply to the motor, scaled from 0 to 100. There is no true feedback from the motor, but increasing the MotorSpeed property will cause the motor to develop more torque, and turn at a higher speed.
Property:	**NumInputs as Long**
Access:	Get By Value
Description:	The number of inputs on this particular I/O device. Please see the hardware description for the details of device variations.
Property:	**NumMotors () as Long**
Access:	Get By Value
Description:	The maximum number of motors on this particular Phidget device. 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.

Event:	**OnInputChange(ByVal Index as Long, ByVal NewState as Boolean)**
Description:	This event is triggered when the input's state is changed. The number of the particular input is returned, as well as the state of the input: True is On, False is Off. In the case of a switch (button) implementation, True would correspond to when the switch is pressed.
Event:	**OnMotorChange(Index as Long, Speed as Long)**
Description:	As the power being applied to the motor is ramped up or ramped down, this event will be fired periodically.

PhidgetStepper

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.

A reference to the actual device can be obtained through the PhidgetManager, or with the Open method on the software component.

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

Property:	**Acceleration(Index As Long) As Long**
Access:	Get, Put By Value
Description:	The controls the rate of ramping.

Property:	**MotorSpeed(Index as Long) As Long**
Access:	Get, Put By Value
Description:	Sets the maximum speed of the motor, measured in steps per second.

Property:	**NumInputs as Long**
Access:	Get By Value
Description:	The number of inputs on this particular I/O device. Please see the hardware description for the details of device variations.

Property:	**NumMotors () as Long**
Access:	Get By Value
Description:	The maximum number of motors on this particular Phidget device. 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.
Property:	**Torque(Index As Long) As Long**
Access:	Get, Put By Value
Description:	Measures the current being consumed by the motor. Setting this property changes the amount of current that will be applied. Higher settings will increase torque, and power consumption. Please note that MotorSpeed sets the ultimate speed of a stepper. Torque increases the ability of the motor to do work.
Event:	**OnInputChange(ByVal Index as Long, ByVal NewState as Boolean)**
Description:	This event is triggered when the input's state is changed. The number of the particular input is returned, as well as the state of the input: True is On, False is Off. In the case of a switch (button) implementation, True would correspond to when the switch is pressed.
Event:	**OnPositionChange(Index As Long, Position As Double)**
Description:	As the stepper moves, this event will fire periodically, with the new position.

PhidgetAdvancedServo — Under Revision

The PhidgetAdvancedServo is a component that provides a high-level programmer interface to control a PhidgetAdvancedServo 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.
- Control the velocity and acceleration of each servo motor.
- Track the predicted position of the servo motor.

A reference to the actual device is obtained through the PhidgetManager, after which the PhidgetAdvancedServo can be used to control the attached device until it is disconnected. After disconnection, the component will continue to function but with its IsAttached field set to false.

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

Property: **NumMotors as Long**

Access: Get By Value

Description: The maximum number of servo motors on this particular servo device. 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 USB device.

Property: **MotorPosition(Index as Long) as Single**

Access: Get, Put By Value

Description: Sets/Returns the desired servo motor position for a particular servo motor. This value may range from 0 - 180, corresponding to degrees from the farthest counter-clockwise position. ServoPosition will return E_INVALIDARG if the value is not between 0 and 180 or if the Index is not a valid servo.

Note that motor position requests are not cumulative. That is, let us say the current actual motor position is 0 and you do the following:

```
Servo.MotorPosition(1) = 180
Servo.MotorPostion(1) = 0
```

The motor will not rotate all the way to 180 before returning back to 0. Rather, it will likely just begin rotating based on the first call, but will immediately return to 0 from its current position when it gets the second call. All you would see (perhaps) is a very slight twitch. If you wanted a cascading effect, you would likely have to use a timer to guarantee that the previous rotation was completed.

Property:	**MotorEnabled(Index as Long) as Boolean**
Access:	Get, Put by Value
Description:	Sets / Returns a state that effectively relaxes the motor. Note that changing the MotorPosition while MotorEnabled is false will set MotorEnabled to true.

Property:	**MaxVelocity(Index as Long) as Single**
Access:	Get, Put by Value
Description:	Sets / Returns the maximum velocity that a particular motor will be rotated at, measured in degrees per second.

Property:	**Acceleration(Index as Long) as Single**
Access:	Get, Put by Value
Description:	Sets / Returns the acceleration/deceleration which will be used while calculating the velocity, measured in degrees per second squared.

Event:	**OnPositionChanged(Index as Long, Position as Long)**
Description:	Occurs when the servo motor is changing its position.

Note that unlike the PhidgetServo device, this will typically reflect the actual position of the servo motor in question. If this is not accurate, you have stalled the motor, or have the MaxVelocity property set to too high a value for your particular servo.