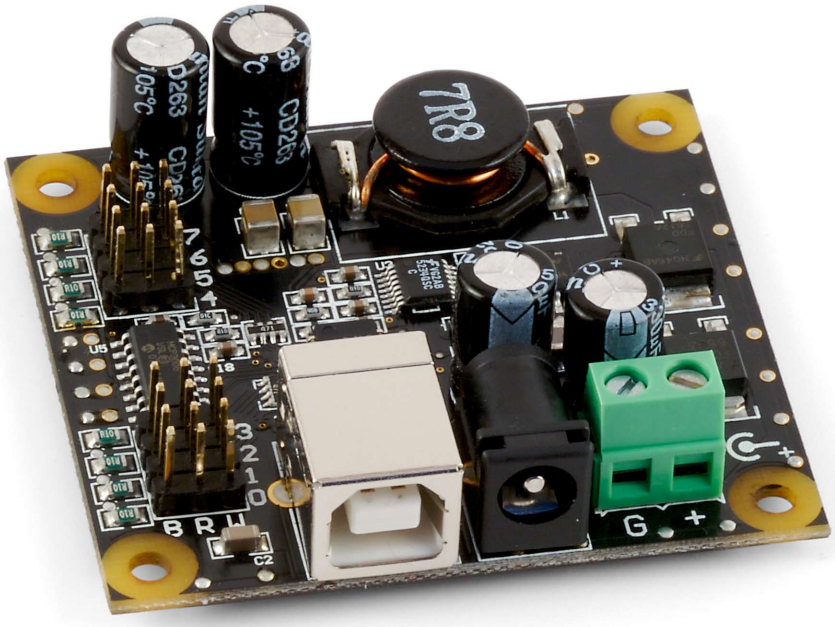
1061 - PhidgetAdvancedServo 8-Motor



Product Features

- Control the position, velocity, and acceleration of up to 8 RC servo motors
- Requires a 6-15VDC external power supply.
- High resolution - 125 steps per degree
- Measures current consumption of each servo.
- A switching power supply allows the AdvancedServo to efficiently operate from 6 to 15 VDC, and be used with a wide range of batteries.
- Switching Regulator protects motors from overvoltage.
- Powers servo motors of up to 1.5amps.
- Connects directly to a computer's USB port.

Programming Environment

Operating Systems: Windows 2000/XP/Vista, Windows CE, Linux, and Mac OS X

Programming Languages (APIs): VB6, VB.NET, C#.NET, C++, Flash 9, Flex, Java, LabVIEW, Python, Max/MSP, and Cocoa.

Examples: Many example applications for all the operating systems and development environments above are available for download at www.phidgets.com.

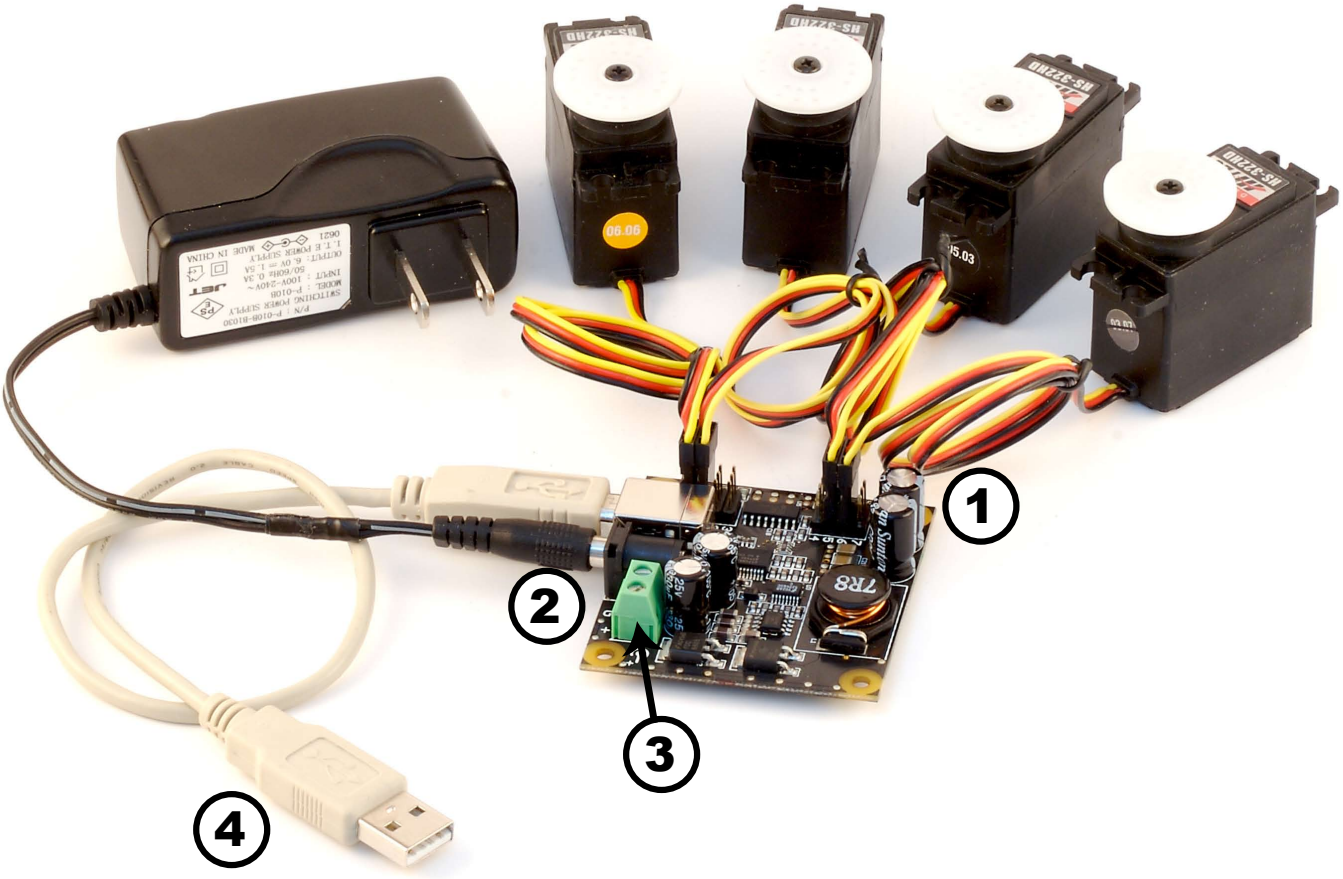
Installing the hardware

The kit contains:

- A PhidgetAdvancedServo 8-Motor
- A USB Cable

You will also need:

- An RC Servo Motor
- A 6 - 15V DC Power Supply



1. Connect the RC Servo Motor to the PhidgetAdvancedServo.
2. Plug in a power supply using the barrel connector.
3. You can also connect a power supply to the Terminal Block for high-current applications. Be sure to observe correct polarity.
4. Connect the PhidgetAdvancedServo to your computer using the USB cable.

Downloading and Installing the software

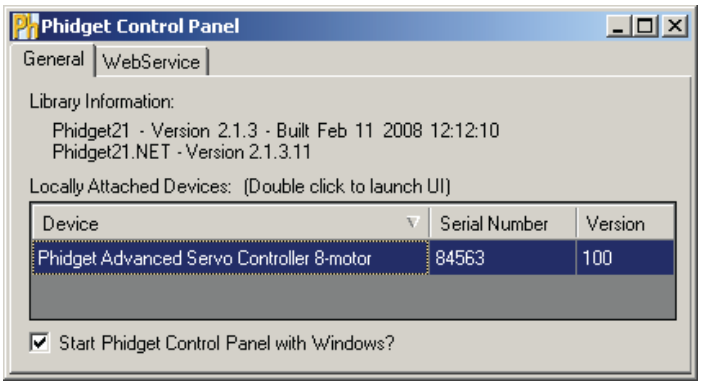
If you are using Windows 2000/XP/Vista

Go to www.phidgets.com >> Downloads >> Windows

Download and run Phidget21.MSI

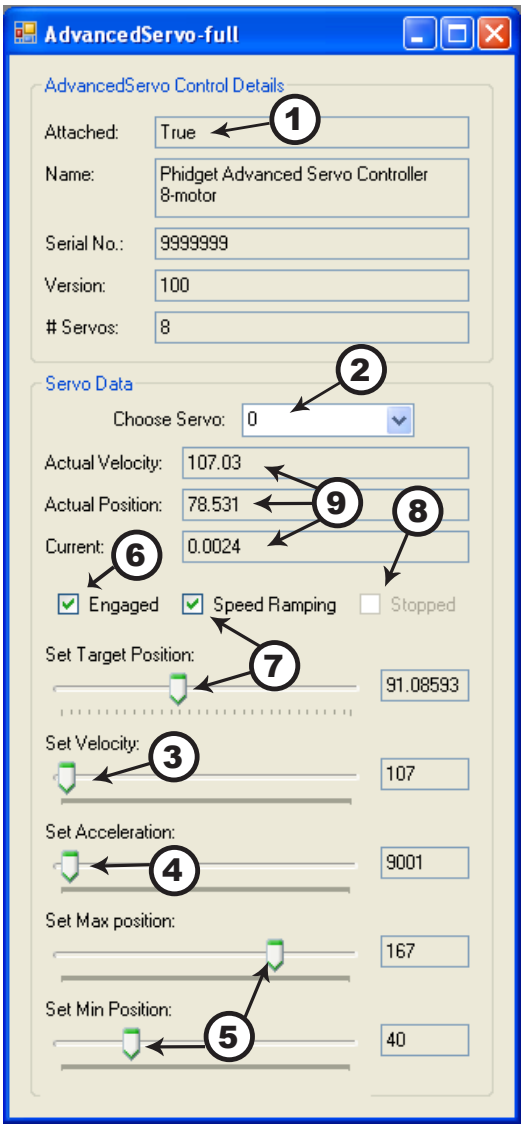
You should see the icon on the right hand corner of the Task Bar.

Testing the PhidgetAdvancedServo 8-Motor Functionality



Double Click on the icon to activate the Phidget Control Panel and make sure that the ***Phidget Advanced Servo Controller 8-Motor*** is properly attached to your PC.

1. Double click on ***Phidget Advanced Servo Controller 8-Motor*** in the Phidget Control Panel to bring up AdvancedServo-full and check that the box labelled Attached contains the word True.
2. Select a connected servo. In this example, a servo is connected at position 0.
3. Use the Velocity slider to set the velocity limit. The servo will try to accelerate to this point during motion.
4. Use the Acceleration slider to set the acceleration.
5. Use the Min/Max Position slider to set the position range. It can prevent the servo from trying to go beyond its actual range of motion.
6. Check the Engaged box to power the servo. If the servo is not already the target position, it should begin to move.
7. Move the Position slider to set a target position. The servo will turn until its actual position equals the target position. If Speed Ramping is enabled, the servo will move using the user set acceleration and velocity.
8. When the servo has reached the target position, a tick mark will appear in the Stopped box.
9. These boxes report the controller's internally calculated position and velocity of the servo, as well as current consumed in amps.

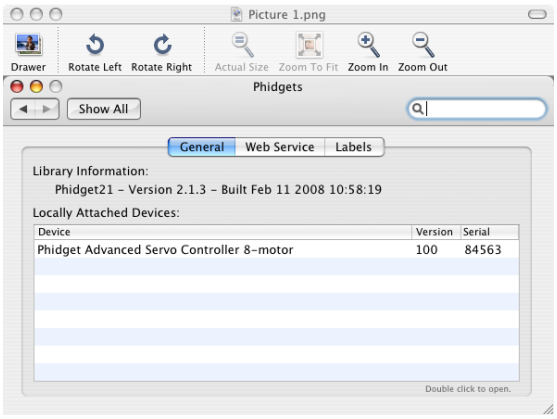


If you are using Mac OS X

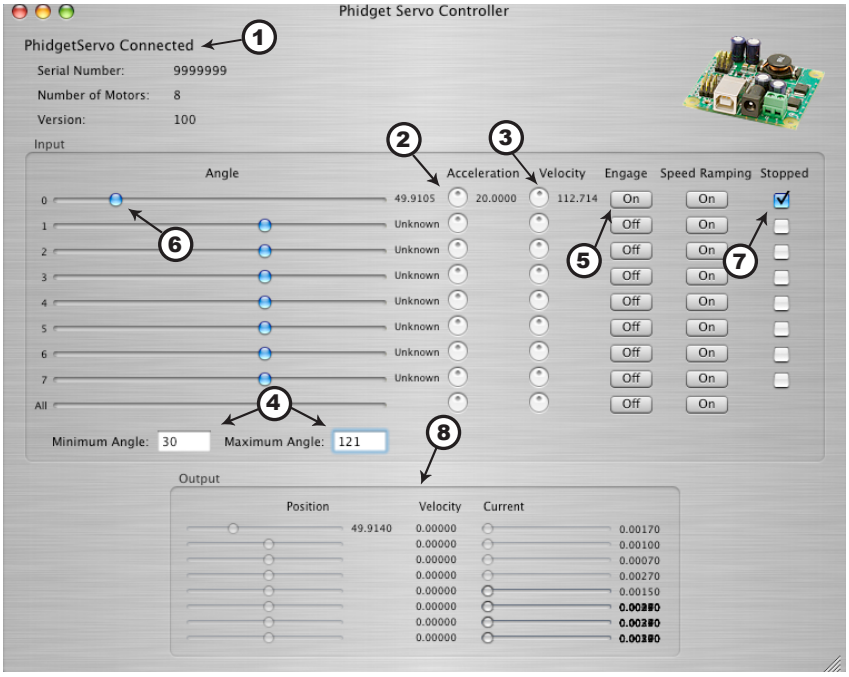
Go to www.phidgets.com >> downloads >> Mac

Download Mac OS X Framework

Testing the PhidgetAdvancedServo 8-Motor functionality



Click on System Preferences >> Phidgets (under Other) to activate the Preference Pane. Make sure that the ***Phidget Advanced Servo Controller 8-Motor*** is properly attached.



1. Double click on ***Phidget Advanced Servo Controller 8-Motor*** in the Phidget Control Panel to bring up the PhidgetServo Controller and check that the servo is connected.
2. Use the Acceleration dial to set the acceleration for a particular servo.
3. Use the Velocity dial to set the velocity limit. The servo will try to accelerate to this point during motion.
4. Use the Min/Max Position boxes to set the range for position. It can prevent the servo from trying to go beyond its actual range of motion.
5. Check the Engaged box to power the servo. If the servo is not already the target position, it should begin to move.
6. Move the Position slider to set a target position. The servo will turn until its actual position equals the target position. If Speed Ramping is On, the servo will move using the previously set acceleration and velocity.
7. When the servo has reached the position target, a tick mark will appear in the Stopped box.
8. This area reports the controller's internally calculated position and velocity of the servo, as well as current consumed in amps.

If you are using Linux

Go to www.phidgets.com >> Downloads >> Linux

- Download Linux Source
- Have a look at the readme file
- Build Phidget21

The most popular programming languages in Linux are C/C++ and Java.

Note: Many Linux systems are now built with unsupported third party drivers. It may be necessary to uninstall these drivers for our libraries to work properly.

Note: Phidget21 for Linux is a user-space library. Applications typically have to be run as root, or udev/hotplug must be configured to give permissions when the Phidget is plugged in.

If you are using Windows Mobile/CE 5.0 or 6.0

Go to www.phidgets.com >> Downloads >> Windows Mobile/CE

Download x86 or ARMV4I, depending on the platform you are using. Mini-itx and ICOP systems will be x86, and most mobile devices, including XScale based systems will run the ARMV4I.

The CE libraries are distributed in .CAB format. Windows Mobile/CE is able to directly install .CAB files.

The most popular languages are C/C++, .NET Compact Framework (VB.NET and C#). A desktop version of Visual Studio can usually be configured to target your Windows Mobile Platform, whether you are compiling to machine code or the .NET Compact Framework.

Programming a Phidget

Phidgets' philosophy is that you do not have to be an electrical engineer in order to do projects that use devices like sensors, motors, motor controllers, and interface boards. All you need to know is how to program. We have developed a complete set of Application Programming Interfaces (API) that are supported for Windows, Mac OS X, and Linux. When it comes to languages, we support VB6, VB.NET, C#.NET, C, C++, Flash 9, Flex, Java, LabVIEW, Python, Max/MSP, and Cocoa.

Architecture

We have designed our libraries to give you the maximum amount of freedom. We do not impose our own programming model on you.

To achieve this goal we have implemented the libraries as a series of layers with the C API at the core surrounded by other language wrappers.

Libraries

The lowest level library is the C API. The C API can be programmed against on Windows, CE, OS X and Linux. With the C API, C/C++, you can write cross-platform code. For systems with minimal resources (small computers), the C API may be the only choice.

The Java API is built into the C API Library. Java, by default is cross-platform - but your particular platform may not support it (CE).

The .NET API also relies on the C API. Our default .NET API is for .NET 2.0 Framework, but we also have .NET libraries for .NET 1.1 and .NET Compact Framework (CE).

The COM API relies on the C API. The COM API is programmed against when coding in VB6, VBScript, Excel (VBA), Delphi and Labview.

The ActionScript 3.0 Library relies on a communication link with a PhidgetWebService (see below). ActionScript 3.0 is used in Flex and Flash 9.

Programming Hints

- Every phidget has a unique serial number - this allows you to sort out which device is which at runtime. Unlike USB devices which model themselves as a COM port, you don't have to worry about where in the USB bus you plug your phidget in. If you have more than one phidget, even of the same type, their serial numbers enable you to sort them out at runtime.
- Each phidget you have plugged in is controlled from your application using an object/handle specific to that phidget. This link between the phidget and the software object is created when you call the .OPEN group of commands. This association will stay, even if the phidget is disconnected/reattached, until .CLOSE is called.
- The Phidget APIs are designed to be used in an event-driven architecture. While it is possible to poll them, we don't recommend it. Please familiarize yourself with event programming.

Networking Phidgets

The PhidgetWebService is an application written by Phidgets Inc. which acts as a network proxy on a computer. The PhidgetWebService will allow other computers on the network to communicate with the Phidgets connected to that computer. ALL of our APIs

have the capability to communicate with Phidgets on another computers that has the PhidgetWebService running.

The PhidgetWebService also makes it possible to communicate with other applications that you wrote and that are connected to the PhidgetWebService, through the PhidgetDictionary object.

API documentation

We maintain API manuals for COM (Windows), C (Windows/Mac OSX/Linux), Action Script, .Net and Java. Look at the section that corresponds to the Phidget you are using. These manuals can be accessed in different ways:

Using Downloads on main menu

Click on Downloads >> Operating System (i.e. Windows) >> Platform (i.e. C#) >> API Document (i.e. Net API Manual)

Using Products on Home Page

Click on InterfaceKits (under Products) >> 1018 PhidgetInterfaceKit 8/8/8 >> API Manual (Under Software Information)

Using Information on Home Page

Click on Information (under Main Menu) >> Your API Manual (under Phidgets API Manuals)

Examples

We have written examples to illustrate how the APIs are used. Examples for the C#.NET programming language Include .exe files for each of the examples in the directory root.

Due to the large number of languages and devices we support, we cannot provide examples in every language for every phidget. Some of the examples are very minimal, and other examples will have a full-featured GUI allowing all the functionality of the device to be explored. Most developers start by modifying existing examples until they have an understanding of the architecture.

To get the examples, go to www.phidgets.com and click on Downloads. Under Step 2: click on your platform of choice and click on the File Name besides Examples.

Support

- Click on Live Support on www.phidgets.com to chat with our support desk experts
- Call the support desk at 1.403.282.7335 8:00 AM to 5:00 PM Mountain Time (US & Canada) - GMT-07:00
- E-mail sales@phidgets.com

Simple example written in C#

```
/*- AdvancedServo simple -
 ********************************************************************************
 * This example simply displays the Phidget AdvancedServo serial number when it is
 * attached and sets the servo positions, velocity, and acceleration to 0.  Then we will
 * do a simple simulation of a basic movement of a servo motor at 100.00 velocity and
 * 100.00 Acceleration.  I decided to leave out the current change event handler for
 * easier readability. For a more detailed example, see the Servo-full example.
 *
 * Please note that this example was designed to work with only one Phidget AdvanceServo
 * connected.
 * For an example using multiple Phidget AdvancedServos, please see a "multiple" example
 * in the AdvancedServos Examples folder.

 * Copyright 2007 Phidgets Inc.  All rights reserved.*/

using System;
using System.Collections.Generic;
using System.Text;
//Needed for the AdvancedServo class, phidget base classes, and PhidgetException class
using Phidgets;
//Needed for the event handling classes
using Phidgets.Events;
//Using this simply for the sleep() method so that the for loop will wail for the motor
//to finish moving to the previous new position before setting a new position
using System.Threading;

namespace AdvancedServo_simple
{
    class Program
    {
        static void Main(string[] args)
        {
            //Declare an Advanced Servo object
            AdvancedServo advServo = new AdvancedServo();

            //Hook the basic event handlers
            advServo.Attach += new AttachEventHandler(advServo_Attach);
            advServo.Detach += new DetachEventHandler(advServo_Detach);
            advServo.Error += new ErrorEventHandler(advServo_Error);

            //hook the phidget specific event handlers
            //I decided to leave out the current change event handler for readability.
            advServo.PositionChange += new PositionChangeEventHandler
                (advServo_PositionChange);
            advServo.VelocityChange += new VelocityChangeEventHandler
                (advServo_VelocityChange);

            //open the Servo object for device connections
            advServo.open();

            //Get the program to wait for an Advanced Servo to be attached
            Console.WriteLine("Waiting for Advanced Servo to be attached...");
            advServo.waitForAttachment();

            //Set the initial position for the servo motor.  I set it to 15 here just
            //since I am going to cycle through the positive values to show a basic
```

```
        //full movement
        if (advServo.Attached)
        {
            advServo.servos[0].Acceleration = 1000.00; //max 4590, min 0
            advServo.servos[0].Velocity = 1000.00; //max 1500, min 0
            advServo.servos[0].Position = 30.00; //range is -23 to 232
        }

        //Wait for the motor to finish getting to position and let user continue
        Console.WriteLine("Press any key to continue...");
        Console.Read();

        //Move the motor from position value 30 to 200, we sleep for 500
        //milliseconds to give the motor enough time to move to the set position
        if (advServo.Attached)
        {
            advServo.servos[0].Position = 200; //range is -23 to 232
            Thread.Sleep(500);
        }

        //Wait for the events to fire and display, user input will continue the
        //program
        Console.WriteLine("Press any key to end...");
        Console.Read();


        //user input was read so we can terminate the program now, close the
        //AdvancedServo object. Left out the current change event handler for
        //readability.
        advServo.PositionChange -= new PositionChangeEventHandler
            (advServo_PositionChange);
        advServo.VelocityChange -= new VelocityChangeEventHandler
            (advServo_VelocityChange);

        //user input was read so we can terminate the program now, close the
        //AdvancedServo object
        advServo.close();

        //set the object to null to get it out of memory
        advServo = null;

        Console.WriteLine("ok");
    }

    //Attach event handler. Display serial number of the attached servo device
    static void advServo_Attach(object sender, AttachEventArgs e)
    {
        Console.WriteLine("AdvancedServo {0} attached!",
                                e.Device.SerialNumber.ToString());
        AdvancedServo attached = (AdvancedServo)sender;

        for (int i = 0; i < attached.servos.Count; i++)
        {
            attached.servos[i].Position = 30.00; //range is -23 to 232
            attached.servos[i].Velocity = 1000.00; //max 1500, min 0
            attached.servos[i].Acceleration = 1000.00; //max 4590, min 0
        }
    }
```

```
        //Detach event handler...Display the serial number of the detached servo device
        static void advServo_Detach(object sender, DetachEventArgs e)
        {
            Console.WriteLine("AdvancedServo {0} detached!",
                                    e.Device.SerialNumber.ToString());
        }

        //Error event handler....Display the error description to the console
        static void advServo_Error(object sender, ErrorEventArgs e)
        {
            Console.WriteLine("AdvancedServo Error: {0}", e.Description);
        }

        //Position CHange event handler...display which motor changed position and its
        //new position value to the console
        static void advServo_PositionChange(object sender, PositionChangeEventArgs e)
        {
            Console.WriteLine("Motor #{0} - Actual Position: {1}",
                                    e.Index, e.Position.ToString());
        }

        //Velocity CHange event handler....display which motor's velocity is changing and
        //its new value to the console
        static void advServo_VelocityChange(object sender, VelocityChangeEventArgs e)
        {
            Console.WriteLine("Motor #{0} - Actual Velocity: {1}", e.Index,
                                    e.Velocity.ToString());
        }
    }
}
```

Technical Section

How RC Servo Motors Work

RC Servos are used for positioning applications. They were originally designed to control Remote Control airplanes and their low cost and high torque makes them very useful as an actuator in prototyping applications.

An RC Servo can be instructed to move to a desired position by the controller. Internally, it monitors the current position, and drives the motor as fast as it can until it reaches the desired position.

This is a very cheap and simple way to control a motor. It has some limitations - there is no way for the controller to know the current position and speed of the motor. Applications that want smooth movement suffer from the aggressive acceleration.

The PhidgetAdvancedServo

The PhidgetAdvancedServo is able to address some of these limitations. Instead of sending the desired position immediately, the PhidgetAdvancedServo sends a series of progressive positions according to acceleration and velocity parameters. In most applications, this dramatically smooths the operation of the servo, and allows reasonably precise control of position, velocity and acceleration.

The PhidgetAdvancedServo has a built in switching regulator - this allows it to efficiently operate from a wide voltage range (6-15VDC), and maintain proper power to the servo motors even if the power supply is varying. This built in voltage regulator will not operate if your power supply is undersized.

Current Sense

The PhidgetAdvancedServo continuously measures the current consumed by each motor. The current roughly corresponds to torque, making it possible to detect several scenarios.

- By monitoring for no current, it's possible to determine if the servo is not connected. It may not be possible to distinguish between a servo at rest and a servo not attached.
- Stalled motors can be detected, by monitoring for the maximum current possible with your motor.
- The position limits of the servo can be programmatically determined by moving the servo until it stalls against the internal or external stops.

Limitations

The PhidgetAdvancedServo does not know the current position of the motor on its own. If your motor is free to move, and is not being driven beyond the physical limitations of the

motor, the position returned to your application will be very close to the position of the motor.

Degree Abstraction

The PhidgetAdvancedServo software component uses degrees to specify position, velocity, and acceleration. The degree unit is translated into a pulse sent to the servo, but it's up to the servo to translate this signal into a particular position. This translation varies between servo models and manufacturers so our degree abstraction will not be exactly the same as the servo you are using. Our degree abstraction is based on the HS322-HD servo, which is typically 10.4uS per degree.

PWM(ms)= [(degrees + 23) * 4/375]

Using the PhidgetAdvancedServo with a Servo Motor

The PhidgetAdvancedServo has been designed to be used with a variety of RC servo motors independent of the motor-specific position, velocity and torque limits. Select a motor that suits your application and falls within the PhidgetAdvancedServo device specifications.

To use a servo motor, first select (in software) which attached motor the PhidgetAdvancedServo should affect. Position, velocity and acceleration can be controlled for each individual motor. The software can also display a readout of the electrical current flowing through each motor.

Using the PhidgetAdvancedServo with Continuous Rotation Servos

A continuous rotation servo is a servo motor that has had its headgear-stop removed and potentiometer replaced by two matched-value resistors. This has the effect of allowing the motor to rotate freely through a full range of motion, but disables the motor's ability to control it's position.

When using the PhidgetAdvancedServo with a servo motor modified in this way, the position control in software becomes the motor's speed control. Because the two resistors that replace the motor's potentiometer are matched in value, the motor will always think its shaft is at center position. If the target position in software is set to center, the motor will believe it has achieved the target and will therefore not rotate. The further away from center the target position is set to, the faster the motor will rotate (trying to reach that position, but never doing so). Changing the value above or below center changes the direction of rotation.

Using the PhidgetAdvancedServo with Electronic Speed Controllers (ESCs)

Some DC motor controllers accept a servo motor PCM signal as valid input, and use the signal to control the speed of a DC motor. Examples of these include Victor and Thor series motor controllers from IFI Robotics. Operation of these are similar to the way the PhidgetAdvancedServo is used to control continuous rotation servos, however DC motors with much higher voltage/current ratings can be driven. Note: a buffer on the control line is

sometimes required when interfacing to these types of motor controllers, and can typically be purchased from the motor controller manufacturer.

RC Servo Motors

The PhidgetAdvancedServo will work with the majority of 3-wire servo motors. A few motors are listed below.

Manufacturer	Part Number	Description
Hitec	HS-55	Feather Series RC Servo Motor
Hitec	HS-322HD	Deluxe Series RC Servo Motor (shown)
Hitec	HS-805BB	Mega Quarter Scale RC Servo Motor

The Hitec HS-322HD is available for purchase at www.phidgets.com. Many RC servo motors are available directly from manufacturers like Hitec or at local distributors.

Consumer Quality - not for industrial use

Most RC Servo Motors are designed for use in toys. Even expensive RC Servo motors with metal gearing will have a very short lifetime compared to industrial quality motors.

API Section

We document API calls specific to the 1061. Functions common to all Phidgets are not covered here. This documentation is deliberately generic. For calling conventions in a specific language, refer to that languages' API manual.

Functions

int Count() [get]

Returns the number of servos this PhidgetAdvancedServo can control. In the case of the 1061, this will always return 8. This call does not return the number of servos actually connected.

double Acceleration(int ServoIndex) [get,set]

Acceleration is the maximum change in velocity the PhidgetAdvancedServo uses when speeding up / slowing down a servo.

- The range of valid Acceleration is bounded by AccelerationMax/AccelerationMin.
- There is a practical limit on how fast your servo can accelerate, based on load and the physical design of the motor.
- This property should always be set by the user as part of initialization. The value does not initialize to the value last set on the device.

double AccelerationMax(int ServoIndex) [get] : Constant

AccelerationMax is the upper limit to which Acceleration can be set. For the 1061, this will always return 320000.

double AccelerationMin(int ServoIndex) [get] : Constant

AccelerationMin is the lower limit to which Acceleration can be set. For the 1061, this will always return 19.53125.

double Velocity(int ServoIndex) [get]

Velocity returns the actual velocity that a particular servo is being driven at. A negative value means it is moving towards a lower position. This call does not return the actual physical velocity of the connected motor.

double VelocityLimit(int ServoIndex) [get, set]

Gets or sets the maximum absolute velocity that the PhidgetAdvancedServo controller will drive the servo. If it's changed mid-movement, the controller will accelerate accordingly. If the target position of the controller is near enough, then the VelocityLimit may never be reached.

- This property should always be set by the user as part of initialization.
- There is a practical limit on how fast your servo can rotate, based on the physical design of the motor.
- The range of VelocityLimit is bounded by VelocityMax/VelocityMin
- Note that when VelocityLimit is set to 0, the servo will not move.

double VelocityMax(int ServoIndex) [get] : Constant

VelocityMax is the absolute upper limit to which Velocity can be set. For the1061, this will always return 6400.

double VelocityMin(int ServoIndex) [get] : Constant

VelocityMin is the absolute lower limit to which Velocity can be set. For the 1061, this will always return 0.

double Position(int ServoIndex) [get,set]

Position is used for both the target and actual position for a particular servo. If the servo is currently engaged and a new value is set, then the controller will continuously try to move to this position. Otherwise, this call will return the current position of the servo. This call does not return the actual physical position of the servo.

- The range of Position is bounded by PositionMin/PositionMax
- If the servo is not engaged, then the position cannot be read.
- The position can still be set while the servo is not engaged. Once engaged, the servo will snap to position if it is not there already.
- This property should be set by the user as part of initialization. If not, it will report the last value set on the device (unless the 1061 has been power-cycled).
- Get will return the last value as reported by the device. This means sets to this value will take a small amount of time to propagate.

double PositionMax(int ServoIndex) [get,set]

PositionMax is the upper limit to which Position can be set, and is initialized to 233. It can be used to prevent the controller from going beyond a servo's range of motion. A PhidgetException will be thrown if this is set above 233 or below PositionMin.

double PositionMin(int ServoIndex) [get,set]

PositionMin is the lower limit to which Position can be set, and is initialized to -22.9921875. It can be used to prevent the controller from going beyond a servo's range of motion. A PhidgetException will be thrown if this is set below -22.9921875 or above PositionMax.

double Current(int ServoIndex) [get]

Current returns the power consumption in amps for a particular servo. The value returned for a disconnected or idle servo will be slightly above zero due to noise.

bool SpeedRamping(int ServoIndex) [get,set]

SpeedRamping enables or disables whether the PhidgetAdvancedServo tries to smoothly control the motion of a particular servo. If enabled, then the 1061 will progressively send commands based on velocity, acceleration and position.

- This property should be set by the user as part of initialization. If not, it will report the last value set on the device (unless the 1061 has been power-cycled).
- Get will return the last value as reported by the device. This means sets to this value will take a small amount of time to propagate.

bool Engaged(int ServoIndex) [get,set]

Enables a particular servo to be positioned. If this property is false, no power is applied to the motors. Note that when it is first enabled, the servo will snap to position, if it is not physically positioned at the same point.

Engaged is useful for relaxing a servo once it's reached a given position. If you are concerned about keeping accurate track of position, Engaged should not be disabled until Stopped = True.

- This property should be set by the user as part of initialization. If not, it will report the last value set on the device (unless the 1061 has been power-cycled).
- Get will return the last value as reported by the device. This means sets to this value will take a small amount of time to propagate.

bool Stopped(int ServoIndex) [get]

Stopped returns false if the servo is currently in motion. It guarantees that the servo is not moving (unless you are moving it by hand), and that there are no commands in the pipeline to the servo. Note that virtually any API calls will cause Stopped to be temporarily false, even changing Acceleration or VelocityLimit on a stopped servo.

Events

VelocityChange(int ServoIndex, double Velocity) [event]

An event issued when the velocity changes on a motor.

PositionChange(int ServoIndex, double Position) [event]

An event issued when the position changes on a motor.

CurrentChange(int ServoIndex, double Current) [event]

An event issued when the current consumed changes on a servo.

Device Specifications	
Pulse Code Period	Typical: 20ms - Maximum: 25ms
Minimum Pulse Width	83.3ns
Maximum Pulse Width	2.7307ms
Output Controller Update Rate	Typical: 31 updates/second
Output Impedance (control)	600 Ohms
Position Resolution	0.0078125° (15-bit)
Lower Position Limit	-22.9921875°
Upper Position Limit	233°
Velocity Resolution	0.390625°/s (14-bit)
Velocity Limit	6400°/s
Acceleration Resolution	19.53125°/s^2 (14-bit)
Acceleration Limit	320000°/s^2
Time Resolution	83.3ns
Minimum Power Supply Voltage	6V
Maximum Power Supply Voltage	15V
Max Motor Current Continuous(individual)	1.6A
Max Motor Current (Surge)	3 A
Motor Overcurrent Trigger (combined)	12A
Operating Motor Voltage	5.0V
Device Current Consumption	36mA max

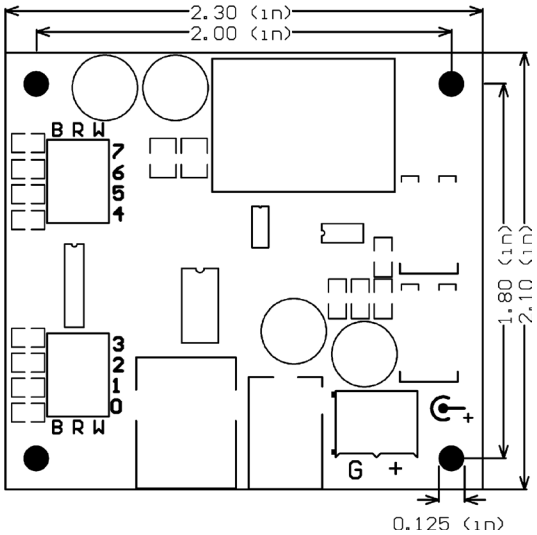
Note: Current from USB supply is not available for motors

Hitech HS-322HD RC Servo Specifications

Torque @ 4.8V	41.66 oz.in
Speed @ 4.8V	190ms/60°
Size L x W x H	1.57" x 0.78" x 1.43"
Weight	1.51 oz.

Mechanical Drawing

1:1 scale



Product History

Date	Product Revision	Comment
July 2008		Product Release