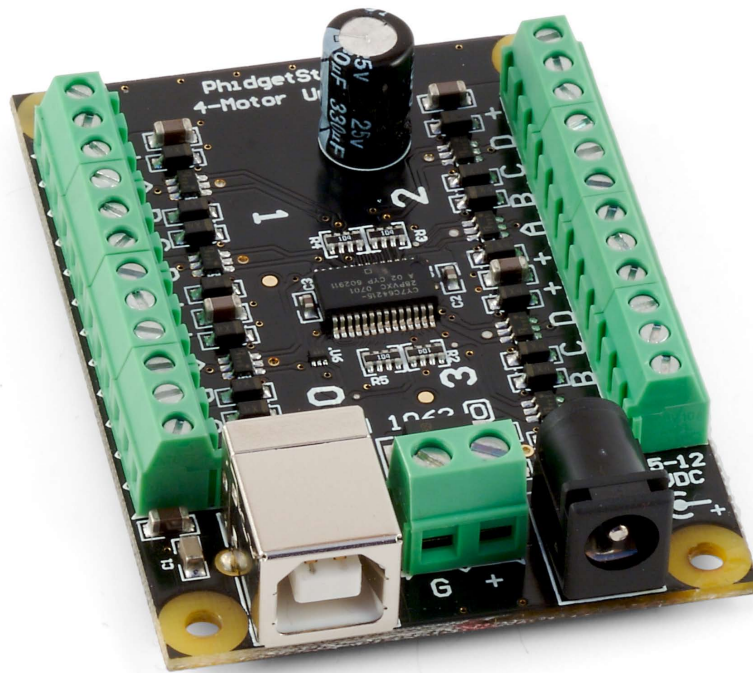


1062 - PhidgetStepper Unipolar 4-Motor



Product Features

- The PhidgetStepper Unipolar 4-Motor allows you to control the position, velocity, and acceleration of up to 4 unipolar stepper motors.
- The 1062 can be used in applications that require precise positioning and continuous rotation, at low cost.
- Requires an external 5 to 12VDC power supply.
- 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.

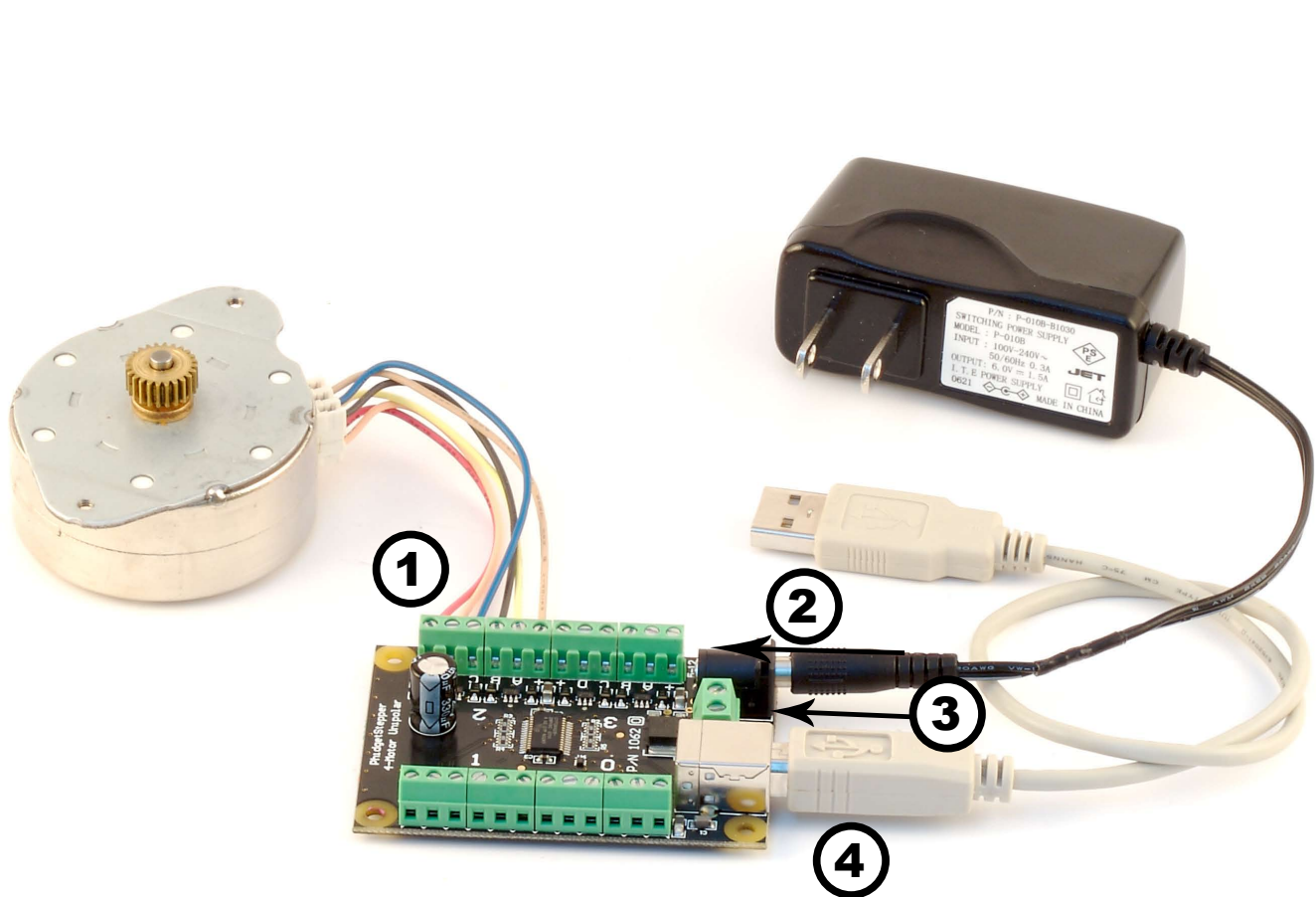
Installing the hardware

The kit contains:

- A PhidgetStepper Unipolar 4-Motor
- A USB Cable

You will also need:

- A 5 to 12V DC Power Supply
- A stepper motor (5, 6, or 8 wire)




1. Connect the motor to the PhidgetStepper board. If you are having difficulty connecting your motor, refer to the Technical Section in this manual.
2. Connect the power supply to the board using the barrel connector.
3. Power supplies with higher current (more than 2.5 Amps) should be wired directly to the terminal block.
4. Connect the PhidgetMotorControl board to your PC using the USB cable.

Downloading and Installing the software

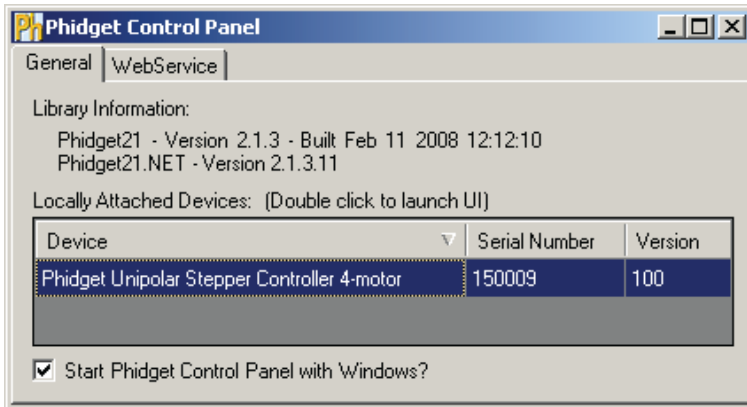
If you are using Windows 2000/XP/Vista


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

Download and run Phidget21.MSI

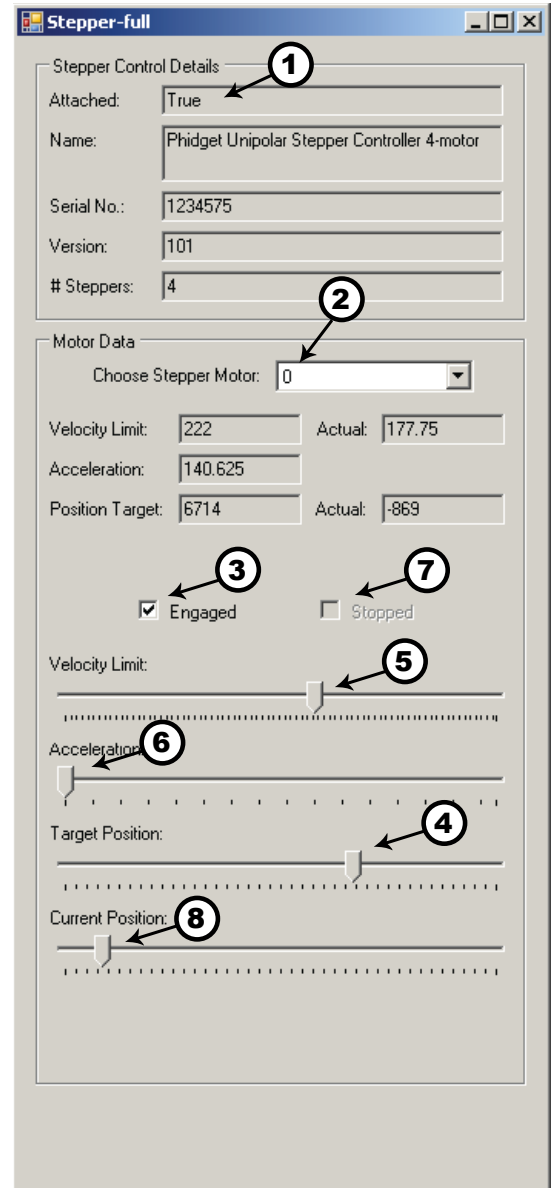
You should see the  icon on the right hand corner of the Task Bar.

Testing the PhidgetStepper Unipolar 4-Motor Functionality



Double Click on the  icon to activate the Phidget Control Panel and make sure that the **Phidget Unipolar Stepper Controller 4-motor** is properly attached to your PC.

1. Double Click on **Phidget Unipolar Stepper Controller 4-motor** in the Phidget Control Panel to bring up Stepper-full and check that the box labelled Attached contains the word True.
2. Select the connected motor. If you have connected your motor at the same place as the one in the picture on page 3, it should be at position 2.
3. Check the Engaged box to power up the motor.
4. Move the Target Position slider to the right or the left. The target motor position will be displayed in the Position Target box and the motor will start turning until the Actual position is the same as the target.
5. Use the Velocity Limit slider to set the maximum velocity. The motor will accelerate until the Actual velocity is equal to the Velocity Limit.
6. Use the Acceleration slider to increase or decrease the acceleration.
7. When the motor has reached the position target, a tick mark will appear in the Stopped box.
8. When the motor is stopped, you can reset the current motor Position by using the Current Position slider.

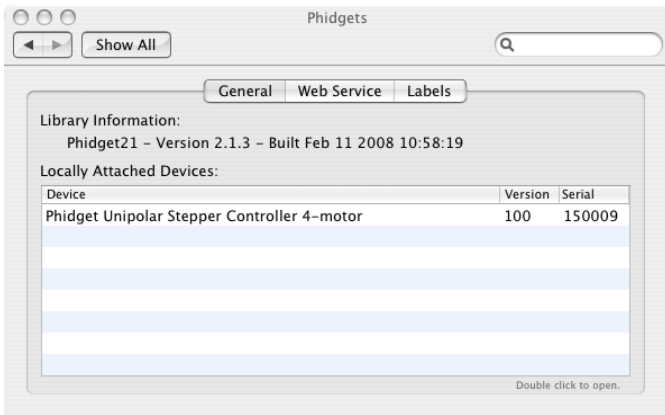


If you are using Mac OS X

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

Download Mac OS X Framework

Testing the PhidgetStepper Unipolar 4-Motor functionality



Click on System Preferences >> Phidgets (under Other) to activate the Preference Pane. Make sure that the **Phidget Unipolar Stepper Controller 4-motor** is properly attached.

1. Double Click on **Phidget Unipolar Stepper Controller 4-motor** in the Phidget Preference Pane to bring up PhidgetStepper Unipolar Example and check that the PhidgetStepper Unipolar 4-motor is properly connected.

2. Select the connected motor. If you have connected your motor at the same place as the one in the picture on page 3, it should be at position 2.

3. Check the Enabled box to power up the motor.

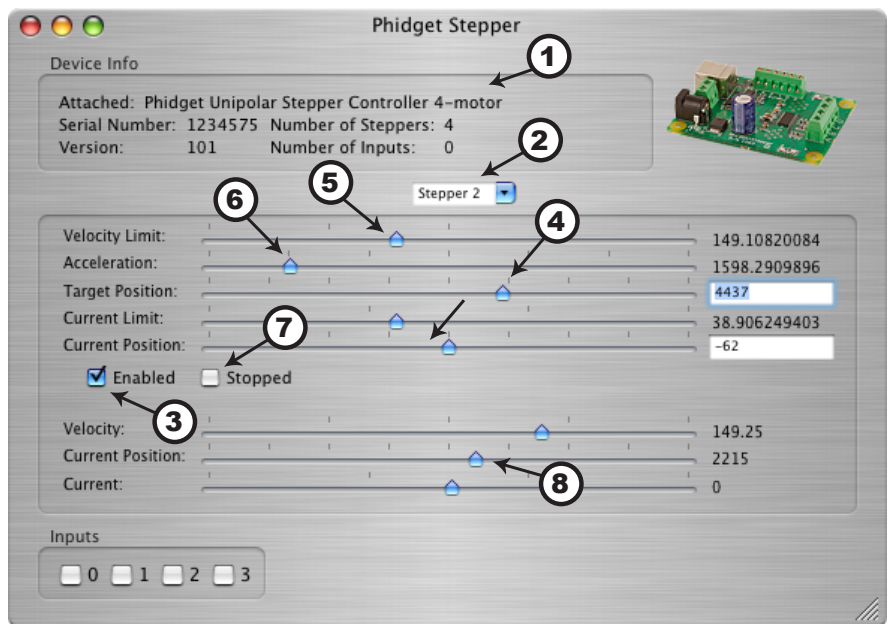
4. Move the Target Position slider pointer to the right or the left. The target motor position will be displayed in the Target Position box and the motor will start turning until the Current position is the same as the target.

5. Use the Velocity Limit slider to set the maximum velocity. The motor will accelerate until the Velocity is equal to the Velocity Limit.

6. Use the Acceleration slider to increase or decrease the acceleration.

7. When the motor has reached the position target, a tick mark will appear in the Stopped box.

8. When the motor is stopped, you can change the current motor position by using the Current Position pointer.



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#

```
/* /* - Stepper simple -
*****
* This simple example sets up a Stepper object, hooks the event handlers and opens it
* for device connections. Once a Stepper is attached with a motor in motor 0 it will
* reset the motor to position 0 and then move the motor to position 2000, displaying the
* event details to the console. For a more detailed example, see the Stepper-full
* example.
*
* Please note that this example was designed to work with only one Phidget Stepper
* connected. For an example using multiple Phidget Steppers, please see a "multiple"
* example in the Steppers Examples folder.
*
* Copyright 2007 Phidgets Inc. All rights reserved.
* This work is licensed under the Creative Commons Attribution 2.5 Canada License.
* To view a copy of this license, visit http://creativecommons.org/licenses/by/2.5/ca/
*/

using System;
using System.Collections.Generic;
using System.Text;
using Phidgets; //For the Stepper class and the exceptions class
using Phidgets.Events; //For the event handling classes

namespace Stepper_simple
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                //Declare a Stepper object
                Stepper stepper = new Stepper();

                //Hook the basic event handlers
                stepper.Attach += new AttachEventHandler(stepper_Attach);
                stepper.Detach += new DetachEventHandler(stepper_Detach);
                stepper.Error += new ErrorEventHandler(stepper_Error);

                //Hook the Stepper specific event handlers
                stepper.CurrentChange += new CurrentChangeEventHandler
                    (stepper_CurrentChange);
                stepper.InputChange += new InputChangeEventHandler
                    (stepper_InputChange);
                stepper.PositionChange += new StepperPositionChangeEventHandler
                    (stepper_PositionChange);
                stepper.VelocityChange += new VelocityChangeEventHandler
                    (stepper_VelocityChange);

                //Open the Stepper object for device connections
                stepper.open();

                //get the program to wait for a Stepper device to be attached
                Console.WriteLine("Waiting for a Stepper to be attached....");
                stepper.waitForAttachment();
            }
        }
    }
}
```



```

        Console.WriteLine("Now the velocity will be set");
        Console.WriteLine
            ("The motor will run until it reaches the set goal position");
        Console.ReadLine();

        //Set the max velocity to start the stepper motor moving until it hits
        //the goal position
        stepper.steps[0].Velocity = 100.00; //Max velocity
        Console.WriteLine("Stepper motor velocity set to: {0}",
            stepper.steps[0].Velocity.ToString());
        stepper.steps[0].Acceleration = 100.00; //Max Acceleration
        Console.WriteLine("Stepper motor acceleration set to: {0}",
            stepper.steps[0].Acceleration.ToString());
        stepper.steps[0].Position = 0;

        Console.ReadLine();
        Console.WriteLine("Will now move to position 2000.");
        Console.WriteLine("Please wait until the motor stops.");
        Console.ReadLine();

        //Set the goal position to 2000
        stepper.steps[0].Position = 2000;

        Console.ReadLine();
        Console.WriteLine("Press any key to end....");
        Console.ReadLine();

        //close the Stepper object
        stepper.Close();

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

        //if no exceptions were thrown at this point it is safe to terminate
        Console.WriteLine("ok");
    }
    catch (PhidgetException ex)
    {
        Console.WriteLine(ex.Description);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

//Attach event handler....Display the name and serial number of the attached
//Stepper device
static void stepper_Attach(object sender, AttachEventArgs e)
{
    Console.WriteLine("{0} {1} attached!", e.Device.Name,
        e.Device.SerialNumber.ToString());
}

//Detach event handler....Display the name and serial number of the detached
//Stepper device
static void stepper_Detach(object sender, DetachEventArgs e)

```

```

{
    Console.WriteLine("{0} {1} detached!", e.Device.Name,
                      e.Device.SerialNumber.ToString());
}

//Error event handler....Display the description of the error
static void stepper_Error(object sender, EventArgs e)
{
    Console.WriteLine(e.Description);
}

//Current Change event handler....Display the motor index and new value of
//the current
static void stepper_CurrentChange(object sender, CurrentChangeEventArgs e)
{
    Console.WriteLine("Stepper Motor {0} current changed! New Value: {1}",
                      e.Index.ToString(), e.Current.ToString());
}

//Input Change event handler....Display the Input index and the new input value
static void stepper_InputChange(object sender, InputChangeEventArgs e)
{
    Console.WriteLine("Input {0} changed! New Value: {1}", e.Index.ToString(),
                      e.Value.ToString());
}

//Position Change event handler. Display the motor index and the position value
static void stepper_PositionChange(object sender,
                                   StepperPositionChangeEventArgs e)
{
    Console.WriteLine("Stepper Motor {0} position: {1}", e.Index,
                      e.Position.ToString());
}

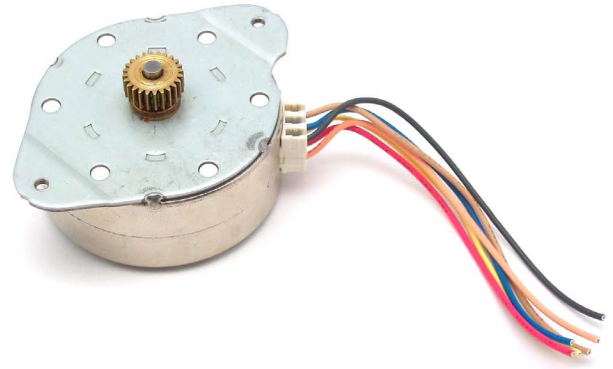
//Velocity Change event handler. Display the motor index and the velocity value
static void stepper_VelocityChange(object sender, VelocityChangeEventArgs e)
{
    Console.WriteLine("Stepper Motor {0} velocity: {1}", e.Index,
                      e.Velocity.ToString());
}
}
}

```

Technical Section

Introduction to Stepper Motors

Stepper motors are broadly available motors commonly used for positioning. DC Motors are controlled by simply applying power which sends them blindly spinning. Steppers Motors are controlled in a series of discrete steps, allowing them to be sent to a very precise position. By repeating the set of steps over and over, the motor can be made to rotate while your system tracks the position.



Some steppers will require hundreds of steps to do a full rotation, while others, like the air core motor, can do a full rotation with only 4 steps.

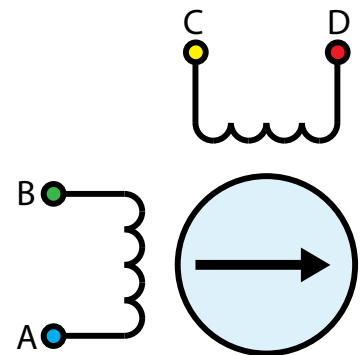
The vast majority of steppers are built using two coils of wire, whose magnetism pulls or repels a rotating magnet attached to an exposed shaft. Depending on how these two coils are available to wire up, we get 4, 5, 6 or 8 wire stepper motors. Many steppers can be controlled both in a bipolar or unipolar configuration.

Bipolar Stepper Motors

Bipolar Steppers motors are available in 4, 6 or 8 wire configurations. Driving a motor in the Bipolar Configuration produces maximum torque. The word Bipolar means that the controlling electronics have to be able to produce a current flow in either direction in each coil, to produce magnetic fields in opposite directions.

4 Wire Stepper Motors - Bipolar Only

A 4 wire Stepper motor can only be controlled as a bipolar. There are two coils, with the ends of each coil brought out as wires typically. By applying current to the two coils in sequence, and chaining the direction of current flow, the motor can be made to rotate.



Unipolar Stepper Motors

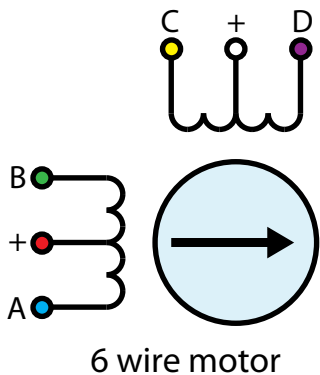
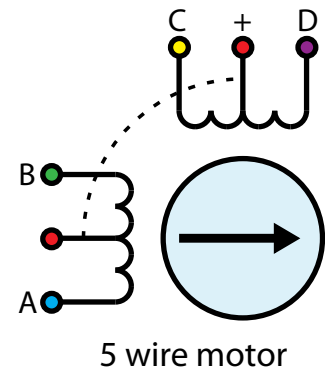
Unipolar Stepper motors are available in 5, 6 or 8 wire configurations. By making the middle of the coils available for connection, the cost of the controlling electronics can be reduced. Driving a motor in the Unipolar configuration reduces the maximum torque the motor can produce – something which you should keep in mind.

The word Unipolar means that the coils only have current passing in one direction. When the center tap of each coil is connected to the power supply, the ends of each coil can be grounded or left floating (ungrounded) in sequence to generate the magnetic fields to rotate the motor.

5 Wire Stepper Motors - Unipolar Only

In a 5 wire motor, the center taps of the coils are connected together. This scheme prevents this motor from being controlled as a bipolar motor.

To use a 5 wire motor as a unipolar, the center tap wire, is connected to the power supply.



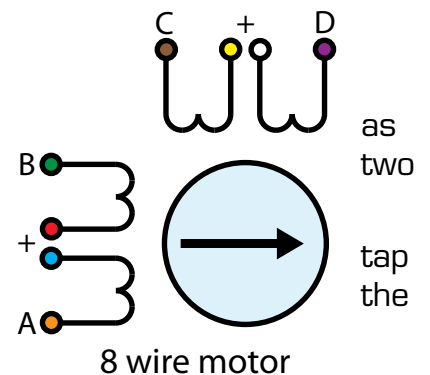
6 Wire Stepper Motors - Unipolar or Bipolar

In a 6 wire motor, the center taps of the coils are brought out, but not connected. If a 6 wire motor is to be controlled as a bipolar, the centers taps should be left unconnected.

8 Wire Stepper Motors - Unipolar or Bipolar

In an 8 wire motor, the coils are split. To use an 8 wire motor as a bipolar, the coils need to be explicitly connected by wiring the wires together for each coil.

To use a 6, or 8 wire motor as a unipolar, the center wires, whether there are 1, 2, or 4, are connected to power supply.



Controlling Steppers - Open and Closed Loop

Because stepper motors do not have the inherent ability to sense their actual shaft position, they are considered open loop systems. This means that the value contained in the current position property is merely a count of the number of steps that have occurred towards the target value; it can not be relied upon as a measure of the actual shaft angle, as external forces may also be affecting the motor.

There are several ways of overcoming this drawback. The simplest is to allow the motor load to depress a limit switch located at a known position. This can be used to fire an event in software to recalibrate the shaft position values. A more elegant solution might involve the mounting of an optical encoder on the shaft and the development of a control system.

Using the 1062 - PhidgetStepper Unipolar with a Stepper Motor

Attach a unipolar motor to the PhidgetStepper Unipolar by connecting the coil-end wires to the terminals labelled A, B, C, D, and the center taps to the terminals labelled with a (+). If the wires are not labelled, or the motor manufacturer does not specify a color-code for the wires, the center taps can be identified with an ohmmeter. This is done by measuring the electrical resistance between the wires to understand the relationship.

- On a 5 wire motor, there will be one motor which has about the same resistance when measured with any of the other 4 wires. This is the center tap and should be connected to the supply terminal.
- On a 6 wire motor, there should be two sets of 3 wires, which both sets electrically insulated from each other. Within each set, there will be one wire which has approximately the same resistance when measured against the other 2 wires in the set. This is the common wire for that set and should be connected to the supply (+) terminal.
- On an 8 wire motor, there should be 4 sets of 2 wires, with each set electrically insulated from the others. One wire from each set should be connected to the supply terminal.

Once as much as possible is known about your motor, the remaining combinations can be enumerated, wired up, and tested until a functional combination is found.

The 1062 PhidgetStepper Unipolar controls stepper motors in half-step increments. A Position increment of one corresponds to one half-step. A stepper motor with 15 degree step increments will rotate in 7.5 degree steps. The 1062 accomplishes this by alternating the number of powered coils between one and two, always at least one coil powered. In this way, the rotor is positioned at both full steps and half steps. The table below describes the order in which coils are powered to achieve this.

After step number 8 in the table, the order the coils are powered in simply repeats from the beginning. As the motor approaches the requested position, it is decelerated according to the value of the acceleration property. When the desired position has been reached, the 1062 stops the motor and holds it at that position

Step Number	Coil(s) Powered				Shaft Angle
	A	B	C	D	
1	1	0	1	0	0°
2	1	0	0	0	7.5°
3	1	0	0	1	15°
4	0	0	0	1	22.5°
5	0	1	0	1	30°
6	0	1	0	0	37.5°
7	0	1	1	0	45°
8	0	0	1	0	52.5°

example stepper motor with 15° step angle

Continuous Rotation

A stepper motor can be caused to rotate continuously by simply setting the motor position property to an extremely large number. The valid range of values for the motor position property is large enough to be able to cause the motor to continuously turn at maximum velocity for 45 years.

Disabling the PhidgetStepper Unipolar

When the stepper motor has rotated a requested number of half steps, and is stopped, the coils will remain energized to hold it in position. This is necessary to allow the motor to support a load on it's shaft without rotating to an unknown position.

The amount of current required to hold a motor shaft in place is called the holding current, and is based mainly on the resistance of the motor coils and the load being supported by the motor. The current required to produce the holding torque can often be large enough to cause the motor to generate heat from the power dissipated in the coils. If the motor is not supporting a load or is not required to maintain a specific angle, it is recommended to set the Enable property to false. This will allow the motor shaft to rotate freely, but the present angle may be lost if forces on the motor-shaft are greater than can be resisted by the detent torque of the unpowered motor.

Starting the motor

When the steppers are first engaged from software, the stepper motor likely will not be at the same state as the default output state of the controller. This will cause the stepper to 'snap' to the position asserted by the controller - potentially moving by 2 full steps.

High precision applications

Stepper motors precision are limited by the manufacturing process used to build them. Errors in the rotor and coils will cause some degree of inaccuracy. In our experience, inexpensive stepper motors will often have positioning errors approaching a half-step.

Synchronization of multiple motors

Many applications call for several steppers motors operating in unison - for example, operating a CNC table, or a robot arm. Highly precise synchronization of steppers using the PhidgetStepper is not possible, as the sequencing will be affected by the real-time performance of your operating system. Each stepper is controlled as a independent unit, so there is no way of arranging for a particular action to happen to all motors at the same time. Typical jitter can be 10-30mS.

Stepper Motors

The PhidgetStepper Unipolar will work with a variety of stepper motors. Some examples are given below.

Manufacturer	Part Number	Description
Danaher Motion	42M100B2U	12VDC 3.6° (100 step/rev) Unipolar Stepper
Danaher Motion	42M048C2U	12VDC 7.5° (48 step/rev) Geared Unipolar Stepper
Danaher Motion	20DAM10D2U	12VDC 0.001" Linear Actuator Unipolar Stepper

These and many other unipolar motors are available from www.digikey.com

API (Software Technical)

We document API Calls specific to the 1062. Functions common to all Phidgets are not covered here. The PhidgetStepper object is a generic object - not all functions will be useful to users of the 1062. (PhidgetStepper is used for other products that may have functionality not in the 1062 - we don't document those calls here) This documentation is deliberately generic. For calling conventions in a specific language, refer to that languages' API manual.

Functions

int MotorCount() [get]

Returns the number of motors this PhidgetStepper can control. In the case of the 1062, this will always return 4. This call does not return the number of motors actually connected - on the 1062, there is no way of programmatically finding out if motors are connected.

double Acceleration(int MotorIndex) [get,set]

Acceleration is the maximum change in velocity the PhidgetStepper uses when speeding up/slowing down the motor. This is specified in the same units used for MotorPosition - in the case of the 1062, half-steps.

- If your motor is heavily loaded, or not supplied with a high enough voltage, there will be a practical limit on how fast it can accelerate.
- The range of valid Acceleration permitted is bounded by the software properties AccelerationMax/AccelerationMin.
- This property should be set by the user as part of initialization. If not set, this value will remain unknown, and could be any of: Minimum acceleration, mid-point acceleration, or any value previously set by another application.

double AccelerationMax(int MotorIndex) [get] : Constant

AccelerationMax is the Maximum Acceleration the 1062 can accept, and apply to the motor. That does not mean that your motor can accelerate that fast!

double AccelerationMin(int MotorIndex) [get] : Constant

AccelerationMin is the Minimum Acceleration the 1062 can accept, and apply to the motor.

double Velocity(int MotorIndex) [get]

Velocity returns the current speed that a particular motor is being driven at. In the case of the 1062, the unit is half-steps per second. With the PhidgetStepper, there is no way of directly controlling the velocity of a motor, because of acceleration curves, however the maximum velocity (VelocityLimit) can be set. The Velocity is returned from the 1062 - so there will be a delay - typically 30-50ms.

double VelocityLimit(int MotorIndex) [get,set]

Sets the maximum velocity that the stepper controller will move the motor. Please note that this is not necessarily the speed that the motor is being turned at. The motor is accelerated to the VelocityLimit, and then decelerated as it approaches the target. If the target is close enough, you may never reach the VelocityLimit.

- VelocityLimit is bounded by VelocityMax/VelocityMin.
- This property should be set by the user as part of initialization. If not set, this value will remain unknown, and could be any of: 0 (motor won't move), mid-point velocity, or any value previously set by another application.
- Note that when VelocityLimit is set to 0, the motor will not move.

double VelocityMax(int MotorIndex) [get] : Constant

VelocityMax is the Maximum VelocityLimit the 1062 can accept. Functionally, this is the maximum speed that the 1062 can drive your motors at.

double VelocityMin(int MotorIndex) [get] : Constant

VelocityMin is the Minimum Velocity Limit the 1062 can accept.

int64 CurrentMotorPosition(int MotorIndex) [get,set]

Returns the current position of a motor. Note that there will be some lag (typical 30-50ms) between the PhidgetStepper reporting a position, and that position being read by your application. CurrentMotorPosition is fixed-point - an increment of one is one half-step - the smallest step that the 1062 can move the motor.

Sets the position that the PhidgetStepper is at right now. This is useful for zeroing the position when a limit switch is reached, for example. To keep accurate track of position, CurrentMotorPosition should only be set when the MotorStopped property is true, because if this property is set while the motor is moving, the motor will have to decelerate to stop moving, before setting the current position.

int64 TargetMotorPosition(int MotorIndex) [get,set]

Sets the desired motor position. Note that setting TargetMotorPosition will override a previous set TargetMotorPosition, and the motor will begin tracking to the new position immediately. The velocity of the motor will be ramped appropriately. TargetMotorPosition is bounded by MotorPositionMin, and MotorPositionMax.

Returns the last set TargetMotorPosition.

int64 MotorPositionMax(int MotorIndex) [get] : Constant

MotorPositionMax is the Maximum MotorPosition the 1062 can accept. Functionally, this is the largest position that the 1062 can move your motors toward. The initial MotorPosition is halfway between Min and Max. Behaviour is undefined if MotorPosition is driven past Min or Max.

int64 MotorPositionMin(int MotorIndex) [get] : Constant

MotorPositionMin is the Minimum MotorPosition the 1062 can move your motors toward.

bool Engaged(int MotorIndex) [get,set]

Enables a particular stepper to be positioned. If this property is false, no power is applied to the motor. Note that when the motor is first Enabled, the coils may not be exactly aligned, and the motor will snap to position.

MotorOn is useful to reduce the power consumed by a motor once it's reached a given position. If you are concerned about keeping accurate track of position, MotorOn should not be disabled until MotorStopped = True.

bool Stopped(int MotorIndex) [get]

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

Events

VelocityChange(int MotorIndex, double Velocity) [event]

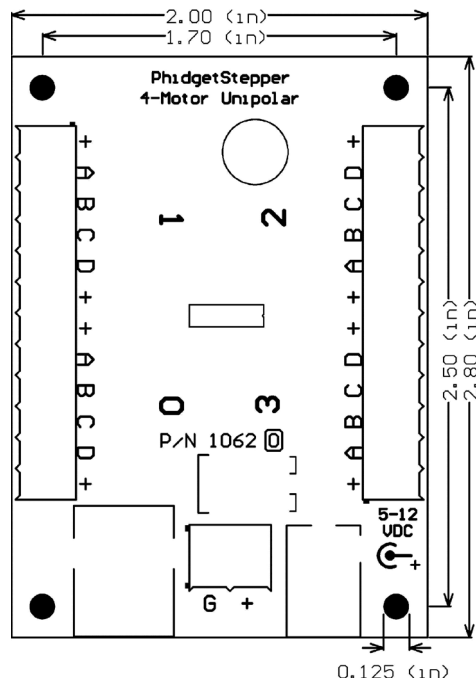
An event issued when the velocity changes on a motor.

PositionChange(int MotorIndex, int64 Velocity) [event]

An event issued when the position changes on a motor. You are not guaranteed to receive events for every motor position - updates are throttled at approximately 16ms.

Mechanical Drawing

1:1 scale



Device Specifications

Output Controller Update Rate	62.5 updates/second/motor
Position Resolution	1/2 step (40-bit signed)
Upper Position Limit	$2^{39} - 1$ 1/2 steps
Lower Position Limit	$-(2^{39} - 1)$ 1/2 steps
Velocity Resolution	0.75 1/2 steps/second (9-bit)
Velocity Limit	383.25 1/2 steps/second
Acceleration Resolution	140.625 1/2 steps/second ² (6-bit)
Acceleration Limit	8859.375 1/2 steps/second ²
Minimum Power Supply Voltage	5V
Maximum Power Supply Voltage	12V
Max Current Per Coil	1A
USB-Power Current Specification	100mA max
Device Quiescent Current Consumption	23mA

Note: current from USB supply is not available for motors

Product History

Date	Revisions	Comment
April 2008	Firmware: 101 Libraries: 2.1.4 Board: 0	Product Release