



Element Robot Controller

User's Guide

© 2010-2012 cmRobot
cmRobot is a registered trademark of QuPiron Technology Corporation
www.cmrobot.com | info@cmrobot.com

©2004-2010 Element .NET User's Guide - cmRobot Corporation. All Rights Reserved.
Element .NET and Element WL™ are trademarks of cmRobot Corporation (www.cmrobot.com)
Microsoft™ .NET is a trademark of Microsoft Corporation.

Element .NET User Guide

Table of Contents

| | |
|---|-------------------------------------|
| Table of Contents | 2 |
| Overview | 3 |
| Element .NET Pinout:..... | 8 |
| Applying Power: | 9 |
| Configuring the onboard H-Bridges: | 9 |
| Serial Hardware Configuration: | 11 |
| RS-232 Serial Interface Module: | Error! Bookmark not defined. |
| TLL Voltage Levels:..... | 12 |
| General Purpose, Analog, and I2C I/O lines: | 14 |
| Servo Power Select Jumper: | 14 |
| Protocol Details | 15 |
| Booting Up:..... | 16 |
| Suggested Serial Terminal Applications: | 16 |
| Command Set Summary | 16 |
| fw..... | 16 |
| reset | 17 |
| blink..... | 17 |
| cmpos3 | 17 |
| cfg enc | 18 |
| cfg baud..... | 18 |
| cfg units | 19 |
| getenc..... | 19 |
| clrenc..... | 20 |
| setio | 20 |
| getio..... | 20 |
| maxez1 | 21 |
| mogo | 21 |
| vpid..... | 22 |
| digo..... | 22 |
| dpid | 23 |
| rpid | 24 |
| pids | 24 |
| pwm..... | 25 |
| step..... | 26 |
| sweep | 27 |
| stop..... | 28 |
| sensor | 28 |
| servo | 28 |
| sp03..... | 29 |
| srf04 | 30 |
| srf05 | 30 |
| pping..... | 30 |
| srf08 | 30 |
| srf10 | 31 |
| tpa81 | 31 |
| vel..... | 31 |
| restore | 31 |
| Line[7]..... | Error! Bookmark not defined. |
| i2c currAddr newAddr | 32 |
| i2c <r w> <addr> [data]..... | 32 |
| Upgrading the Firmware:..... | 34 |
| Warranty & Disclaimer Information: | 34 |
| Element Libraries & Documentation: | 34 |
| PID Configuration Examples: | 34 |
| Velocity PID (VPID) | 34 |
| Distance PID (DPID) | 36 |
| Element™ Dimensions: | 36 |
| Element™ Maximum Ratings..... | 37 |
| ASCII Character Set..... | 38 |
| Contact Information: | 39 |

Element .NET User Guide

Overview:

The Element .NET Robot Controller is the fourth generation of the Element Robot Controller. The following improvements have been made:

Upgrade to a 40Mhz cpu (10MIPS).

Doubled the memory (64Kb).

Added 3 GPIO lines.

Updated Bootloader (to Tiny Bootloader).

Updated hardware to support XBee Pro modules.

Updated firmware to support controlling multiple Element Controllers with one COM port on the host computer, using XBee modules.

Added TEST command to firmware.

Added a few useful firmware commands to make debugging and PID configuration easier.

Easy Robot Control using .NET:

The Element Robot Controller provides a ready-to-use solution to interface Microsoft™ .NET Framework and C++ applications to common robotic hardware. Never before has it been this easy to interface DC motors, servos, analog sensors, I2C devices, single and quadrature encoders, switches/relays, and other devices to the .NET Framework. Although the Element Robot Controller was developed for computers running Windows 7, Vista, XP, XP Embedded, or WinCE and the Microsoft .NET Framework, it can also communicate with any controller which contains a free serial port. This means you can communicate with it from other OSes, including Linux and Mac OSX via the use of the Mono .NET framework.

Develop Smarter Applications:

The Element Robot Controller enables .NET developers and Robotics Studio developers interested in developing robotic applications to hit the ground running fast. Why waste time re-implementing bit-twiddling routines when that time could be better spent developing a higher level (and smarter) application that allows your robot to do something meaningful? With the exception of configuring a few parameters, there's no low level programming involved. The Element Robot Controller links both the .NET Framework and C++ apps with low level robotic hardware using a simple serial port.

Open Source .NET library with Full Documentation:

cmRobot has decided to Open Source the Element .NET Library, to allow customers to modify it, improve it, and make it even better than it already is. The Open Source license can be found on the Element product support page and in the downloads section for the Element .NET Library.

Element .NET User Guide

The Element .NET library allows customers to quickly develop applications which can instantly communicate with the Element Robot Controller. Once an application links in the ElementLib.dll, the entire Element interface is available.

The Element .NET library also includes several C# GUI application examples, which contain a user control to invoke every method in the ElementLib.dll assembly. Developers can use it to test out their Element Robot Controller, or as a source for examples for their own application. The Element .NET library Quick Start Guide document provides the exact steps for adding the library to your application. Also, check out the Sample Applications page for more example apps containing source code.

We realize that it's important that developers have easy and quick access to the Element .NET Library interface. Therefore, we provide excellent [MSDN style web pages](#) to document the entire interface, as well as provide links to the supported sensors and components.

All you need is a serial port:

Using our simple serial control protocol, the Element Robot Controller can be controlled by any device featuring a serial port. This could be a Personal Computer, a PDA, a PC104 board, a Single Board Computer, a tablet or a microcontroller. To make developing applications with the .NET Framework faster, we provide the Element .NET library which implements the protocol, and provides an easy to use interface.

Simple Serial Protocol:

The Element Robot Controller Protocol is a simple set of rules defined to allow a program running on a host computer to communicate with the Element board over a serial connection. The protocol commands are made up of 8-bit ASCII characters for ease of use and to reduce bandwidth usage. The use of ASCII characters also allows users to send/receive commands to/from the Element Robot Controller via a simple terminal program (such as Putty), or a dedicated application running on another computer for debugging purposes.

Upgradeable Firmware:

New sensors and components, appropriate for use in robotics, are introduced to the market daily. We realized that the ability to add support for these new sensors and components would make the Element Robot Controller even more useful. Therefore, we include the robust [Tiny Bootloader](#) within the Element Robot Controller. You simply use the Tiny Bootloader application to upload Firmware to the board over the same serial connection you use to communicate with it.

Upgrading the firmware requires three steps:

- 1.) Download the latest firmware from our website

Element .NET User Guide

2.) Download the Tiny Bootloader application (exe) from our website

3.) Upload the firmware to the Element Robot Controller with the help of Tiny Bootloader. You can find the firmware upgrading procedure in the Element resources page @ www.cmrobot.com.

We are also open to requests to add new sensors and components to the current lineup. Customers can send requests to info@cmrobot.com for consideration. If we believe the request is appropriate, we will add support to the firmware, if possible. Since the Element's initial launch, we have added the following functionality to the Element Robot Controller per customer requests:

- Velocity and Distance PID Control and State
- Setting multiple GPIOs simultaneously
- Pre/Post (.NET) events before/after establishing serial communications w/ the Element
- Bipolar Stepper motor control (step and sweep)
- Maxbotix MaxSonar-EZ1 Sonar interface
- Generic I2C interface
- Pwm Ramping
- Factory Restore interface
- Reset interface
- Test interface
- Add four extra servo ports for a total of six available servo ports
- Add backspace, and last command functionality
- Add ability to control multiple Elements via one COM port (requires XBee)

Supports the most popular robotic sensors and components:

The Element Robot Controller provides an interface to query and control some of the most popular robotic components on the market. The current list includes:

I2C Devices:

- Any I2C device (using generic I2C interface)
- Devantech SRF02
- Devantech SRF04
- Devantech SRF05
- Devantech SRF08
- Devantech SRF10

Element .NET User Guide

- Devantech TPA81 Thermopile
- Devantech SP03 Speech Synthesizer
- Devantech CMPS03 Electronic Compass
- Devantech LCD03

Analog Sensors (5 Sensor Inputs)

- Sharp™ GP2D12 infrared distance sensor
- Sharp™ GP2D120 infrared distance sensor
- Onboard battery voltage level monitor– available on analog input 5

Digital I/O Lines (13 I/O Lines + 4 Encoder Input lines):

- Parallax PING)))™ Sonar
- Maxbotix MaxSonar-EZ1 Sonar
- Single and Quadrature Encoder inputs
- NOTE: Six I/O lines are used for Servo control (4,5,6,7,8, & 9). Thus if you're using Servos on any of these pins, you won't be able to use them for any other I/O.

Motors:

- Two DC drive motors up to 4A each
- One Bipolar Stepper Motor
- Six Standard or Digital Hobby Servos
- Gamoto External PID Controller (uses generic I2C interface)
- External H-Bridge control
- Sabertooth 2x5 5A Motor Controller
- Built in Velocity and Distance PID Motor Control Algorithms

LEDs:

- One Green Power LED
- One Green LED is used as a programmable heartbeat
- Two Bi-Color (Green/Red) are used to display the PWM duty cycle percentage (0-100) and direction of current through each motor (and thus motor direction).

No need for serial logic level conversion:

The Element Robot Controller can be interfaced using any of the following serial interfaces modules:

- USB (optional module)
- XBee/XBeePro (optional module)
- TTL port (header located on Element Robot Controller, not on modules).

Element .NET User Guide

Figure 2: Multiple Supported Wired and Wireless Serial Interfaces



USB



XBee

If you want to communicate with the Element from your PC, simply plug in one of the serial interface modules mentioned above into the Element Robot Controller. While only one module will work at a time, you can swap the modules out as needed. If you have a microcontroller or embedded device which needs to talk serially at TTL voltage levels, simply use the 0.100" header on the Element base board, and remove one jumper.

Flexible H-Bridge Control:

Not only can you use the onboard h-bridges to drive two DC motors (up to 4A each), but you can also drive external h-bridges (with a higher or lower capacity) with the change of a jumper. If that's not enough, you can drive the onboard h-bridges from an external controller. This flexibility allows the customer to use the Element Robot Controller in many applications.

Using the Generic I2C command, you can ALSO easily interface up to 8 **Gamoto PID Motor Driver boards**, without tying up the onboard h-bridges. This allows you to use the onboard h-bridges to control two motors (or a bipolar stepper motor), and use the Gamoto PID drivers to control extra motors.



Figure 3: Gamoto PID Motor Controller Board

Single and Quadrature Wheel Encoder and PID Control Algorithm support included:

Since it's important to know where your robot has been, where it is now, and where it's going, the Element Robot Controller also includes the capability to query two single or two quadrature encoders. Additionally, it includes built-in fully configurable PID motor control. You can guarantee that your robot travels the exact distance you command it, at the specified speed.

Element .NET User Guide

Each Proportional, Integral, Derivative, and Loop parameter can be configured for your personal drive train (with some exceptions). Robot velocity, distance, and direction can be extracted from the encoded inputs. The current PID state can also be queried, so you can determine when the latest PID distance command has finished. Please see the PID configuration section at the end of this document for information regarding PID configuration.

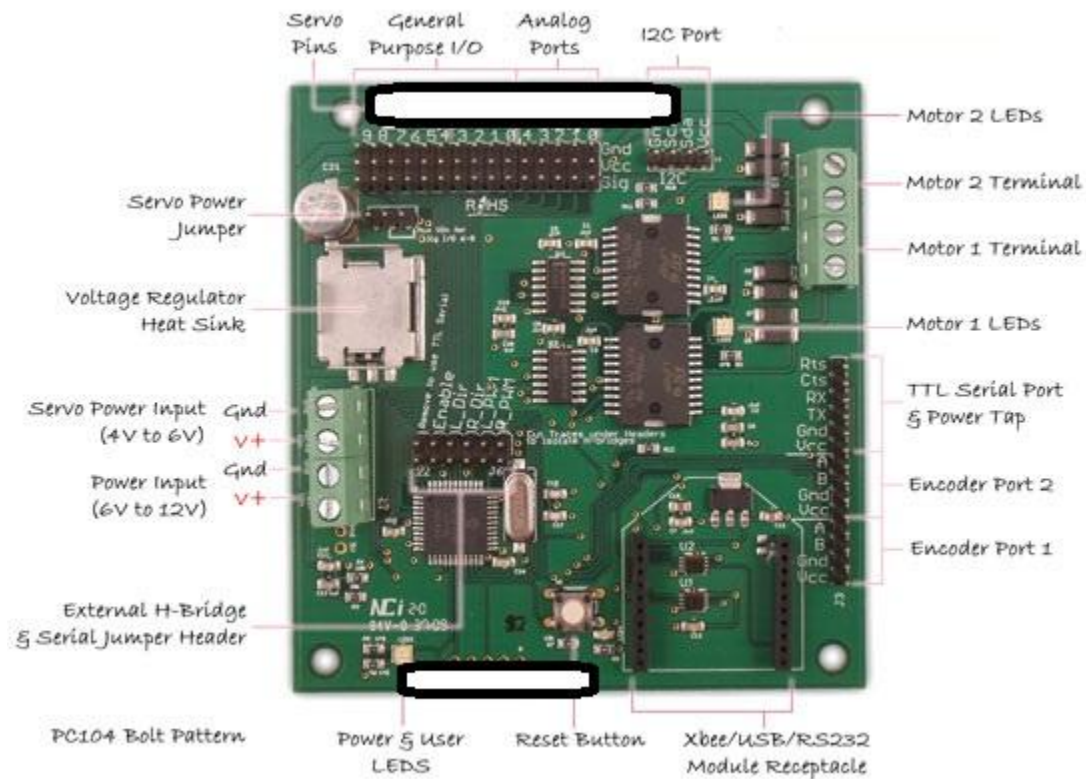


Figure 4: Element Robot Controller Pinout

Element Robot Controller Pinout:

Before you can interface any components to the Element Robot Controller, you need to know how they interface. The picture above shows all of the I/O lines and connectors on the Element Robot Controller, which includes:

- One Regulated Power Input Terminal (6v to12v)
- One Unregulated Servo Power Input Terminal (you supply regulated 4v to 6v)
- One Servo input power selection jumper

Element .NET User Guide

- One Serial Interface Module Header for USB, and XBee (please note location of Pin1)
- One TTL Serial I/O Communication Port
- Two Encoder Ports – Single and Dual Quadrature Channels supported
- Two DC Motor Terminals – Handles motor current draw up to 4A, each terminal
- One I2C Port – SDA, SCL, 5V, Gnd
- One Analog Port with six analog inputs – one input is tied internally to the supply voltage
- One General Purpose I/O Port with 13 I/O lines – 4,5,6,7,8, & 9 can be used to control six servos
- One Reset Button
- Jumper bank to configure H-Bridges and Serial I/O communication
- One Green User programmable LED, and one Green Power LED
- One Green & one Red LED per DC Motor Port to show current flow direction through motors

Applying Power:

This is one of the most important steps in getting the Element Robot Controller up and communicating with your host controller. You **MUST** make sure that you apply power to the Power Terminal using the correct polarity. **Reverse Polarity will damage the Element Robot Controller.** *cmRobot is not responsible for such damage, nor do we warranty against such damage.* Make sure you take time to apply power correctly.

To power the Element Robot Controller, simply connect the ground wire from your supply to the screw terminal labeled "Gnd", and then connect the positive wire from your supply to the screw terminal labeled "+".

NOTE: Maximum supply voltage cannot exceed 12V DC.

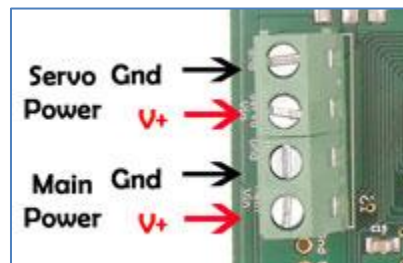


Figure 5 – Power Supply Terminal

Once power is supplied, you should see the Red Power LED light up, and the two User definable Green LEDs should be twiddling back and forth. Also make sure you have a power supply which can supply at least 1 amp of power out or more, especially if you have many components connected (e.g. sensors, etc.). In fact, **if you have components attached, we suggest you use a power supply rated to supply a minimum current output of 1.5A to 2A.**

Configuring the onboard H-Bridges:

Element .NET User Guide

Not only can you use the onboard h-bridges to drive two DC motors (up to 4A each), but you can also drive external h-bridges (with a higher or lower capacity) with the change of a jumper. If that's not enough, you can drive the onboard h-bridges from an external controller.

There are four jumpers used to configure the H-Bridges (pictured below). Note: The leftmost jumper is used for Serial Communication configuration, and doesn't affect h-bridge operation. If you want to drive external h-bridges, the h-bridges can be connected via jumper wires to the pins at this header.

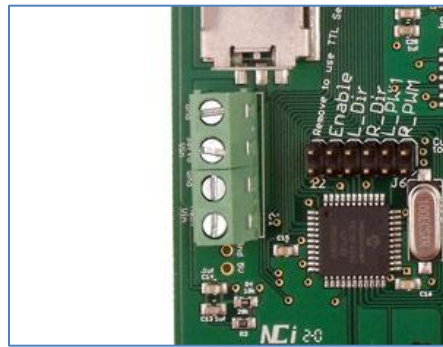


Figure 6 – H-Bridge Jumpers

The internal H-Bridges (rated at 4A each) are configured to be driven by the onboard controller from the factory. In this configuration the traces on the bottom of the board connect the pins are not needed.

If you wish to surpass the internal H-Bridges, and drive external H-Bridges w/ a higher rating, you can cut the traces on the bottom of the board, and connect the h-bridges to the pins w/ jumper wires as shown (in orange) below:

If you cut the traces and then decide to go back to using the onboard H-bridges, simply make sure all jumpers are installed across the five H-Bridge pins (as shown above).

Element .NET User Guide

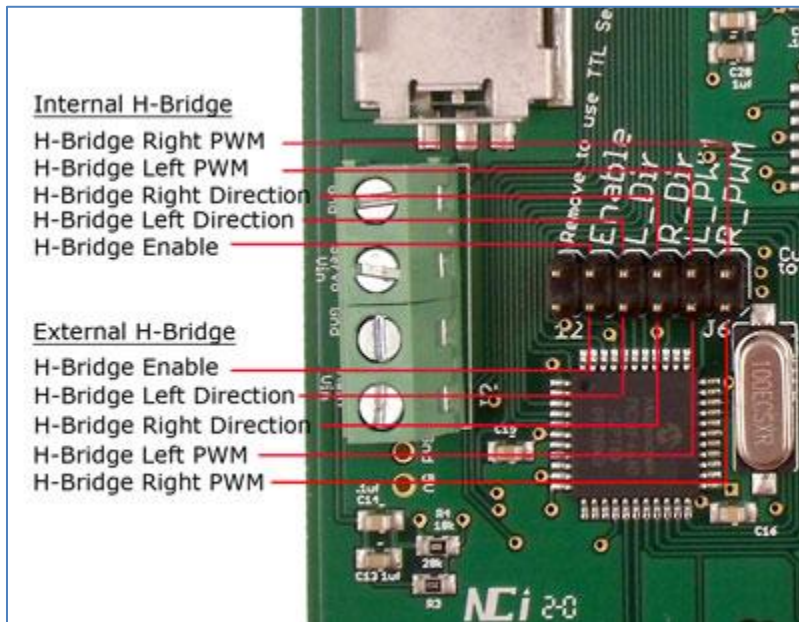


Figure 7 – H-Bridge Jumper Configuration

If you wish to drive the internal H-Bridges from an external controller, you can simply connect the pins (labeled in blue) to the appropriate step and direction pins as shown above with jumper wires.

You also have the ability to interface Gamoto PID Motor Controller boards via the I2C port using the Generic I2C command.

Serial Hardware Configuration:

You probably have a good idea at this point how you're planning on interfacing the Element Robot Controller to your computer. As you have seen, the Element can communicate over five different serial interfaces, but care **MUST BE TAKEN** to ensure that you don't damage the internal Element Robot Controller communications hardware or serial modules. The serial interface modules should be inserted as shown in the diagrams below.

USB Serial Interface Module:

If you are communicating with the Element Robot Controller using a USB connection from any computer, then you'll connect the USB serial interface module as shown below:

Element .NET User Guide



Figure 9: Element Robot Controller with USB Module

XBee Serial Interface Module:

If you are communicating with the Element Robot Controller using an XBee connection from any computer, then you'll connect the XBee or XBee Pro module as shown below.

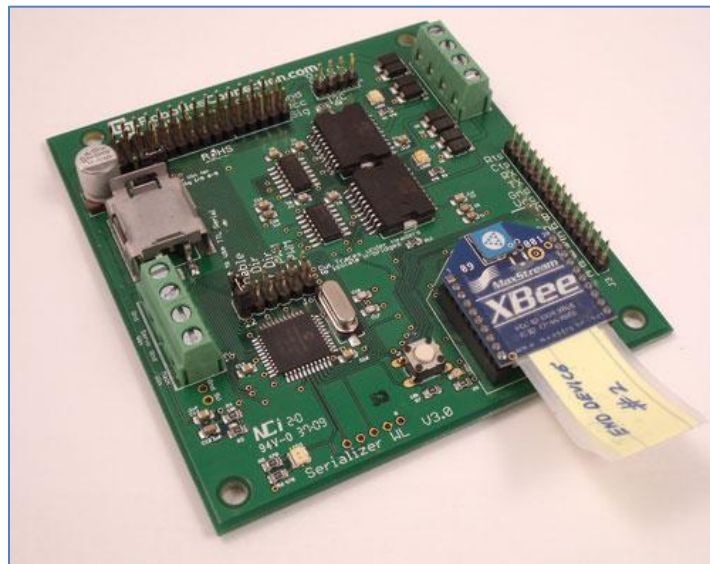


Figure 11: Element Robot Controller with XBee Module

TLL Voltage Levels:

If you are communicating with the Element Robot Controller using a device which uses TTL voltage levels (such as a microcontroller, or wireless device) then you'll need to perform two steps.

Element .NET User Guide

First, remove the jumper in the figure below labeled "Remove to use TTL Serial". This basically enables the TTL_Serial port in the figure above, disables the serial interface modules on the Element Robot Controller (if plugged in).



Figure 12: RS-232/TTL Select Jumper

Next, connect the pins on the TTL Serial port (see image below) with correct pins on the device communicating with the Element Robot Controller. This will include RX, TX, Gnd, and Vcc. Vcc is used as a +5V supply power (if needed) to the host device. Ground should ALWAYS be connected between the

Element Robot Controller and Host device. Otherwise, serial communications will most assuredly not work, since the voltage levels won't have a common ground from which to measure voltage levels from.



Figure 13: TTL Serial I/O Pins

You can also leverage RTS and CTS for flow control if need be. RTS and CTS are tied to internal pins.

Element .NET User Guide

General Purpose, Analog, and I2C I/O lines:

The Element Robot Controller contains 10 General Purpose I/O (GPIO) lines, 6 10-bit Analog input lines, and an I2C port.

Below is a diagram showing the detailed pin out of the I/O lines. The topmost row of pins for the GPIO and Analog port is Gnd (Ground). The middle row of pins is Vcc (+5V) power supply to devices connected to the ports. The lower row are the input/output signals. **Make sure you connect the appropriate pins between the ports and your device to ensure that damage doesn't ensue. Also do not connect a sensor or device which is capable of producing more than 5.3V to any of the I/O or Analog lines (damage will ensue).**

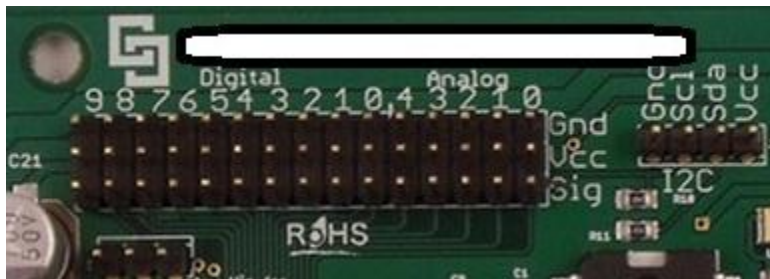


Figure 15: General Purpose, Analog and I2C I/O Line Pinout

The leftmost port is the General Purpose I/O port (labeled Digital). Pins are numbered 0-9. Pins 4,5,6,7,8, & 9 can be used to control up to six standard or digital hobby servos. **If you are controlling Six servos**, make sure you are using external power to drive I/O lines 4,5,6,7,8, and 9 (see Servo Power Select section below). NOTE: When you use an I/O line to control a servo, it obviously can't be used for

any other functionality. There are also three additional GPIO lines tied to H-Bridge enabling/disabled, I2C SCL, and I2C SDA pins.

The middle port is the 10-bit Analog port (labeled Analog). Pins are numbered 0-4, and provide 5 analog inputs. However, there is an extra analog pin which is internally tied to 1/3 of the supply voltage level to allow users to read the supply voltage to the board.

The rightmost port is the I2C port (labeled I2C). Pins are labeled Gnd (Ground), N/C (no connection), Scl (Serial Clock), Sda (Serial Data), and Vcc (+5V supply). Again, SDA and SCL serve as GPIO pins when not used for I2C functionality.

Servo Power Select Jumper:

Element .NET User Guide

Power is supplied to Servo/GPIO lines 4,5,6,7,8, and9 via the external servo power terminal by default. You can use the main input power source instead to provide internal regulated power to Servo/GPIO lines 4,5,6,7,8, and 9 to drive the servos. **However, understand that servos can draw >1A! This can cause the main regulator to overheat, and possibly cause the Element to reset if too much current is drawn by a servo.** So, you're better off supplying unregulated external voltage to the servo power terminal. You can use the onboard power (coming through the main regulator) by moving the Servo Power Select Jumper from the right two pins (factory setting) to the left two pins (see jumper diagram below).



Figure 16: Servo Voltage Regulator Select

NOTE: If you are using a total of six servos under moderate to heavy loads, make sure you provide external unregulated 4v to 6v power. Failure to do so will cause the main voltage regulator to overheat, and cause permanent damage to your Element! Servos can draw a lot of current, especially 6 of them!

Protocol Details

All commands and parameters are separated by spaces. There are two types of parameters: Simple and Complex. Simple parameters are basically simple strings, such as "hello" or "1". Complex parameters are

simple strings separated by colons ":", such as "hello:world" or "2:1". Think of a complex parameter as a key:value pair. Complex parameters are used in commands where multiple ids are used in a single command. This allows multiple objects to be interrogated on the Element Robot Controller in a single atomic command. The number of required parameters depends on the command being issued.

All commands (and associated parameters) are terminated by an ASCII carriage-return character "\r" (0x0D), which is denoted by a <CR> in the command set below. The <CR> in the command examples below simply means that each command needs to have a "\r" appended to it before it will be processed. If you are using Hyperterm to send commands by hand, it simply means that you hit the 'return' key.

All responses will be appended with a "<CR><LF>", followed by a prompt ">" string, which will signify that the Element™ has processed the command, and is ready to receive another command. If an error condition occurs, then a "NACK<CR><LF>" followed by a prompt ">" string will be returned. For commands which do not return a value, an "ACK<CR><LF>" followed by a prompt ">" will be returned if

Element .NET User Guide

the command was successfully executed. NOTE: The light grey portion of the command example below depicts the characters sent by the Element.

```
>command param1 param2<CR>
```

```
ACK|NACK<CR><LF>
```

```
>
```

Booting Up:

Once the Robot Controller boots up, the following information is printed out (NOTE The example below is what you would see if you were using a terminal application like PuTTY):

```
Element, firmware 1.0.0 Copyright 2010-2010, cmRobot Corporation
```

```
>
```

If you're not using a terminal application, then this information isn't really useful. **Thus remember to ignore this sequence the first time you read your serial port.** We basically added it to aid in debugging communication troubles. If you do not see the information above appear during power up, the Baud Rate, Data Bits, Stop Bits, and/or Parity are probably not configured correctly on the host computer which is communicating w/ the Element. The Element Robot Controller ships communicating at 19200 baud, 8 data bits, 1 stop bit, No Parity, and no Flow Control.

Suggested Serial Terminal Applications:

We suggest the use of [PuTTY](#) as a terminal application. PuTTY will work across all Windows versions.

Command Reference Set

fw

The *fw* command returns the current firmware version

Example:

```
>fw<CR>
```

```
1.0.0
```

```
>
```

Element .NET User Guide

reset

The *reset* command resets the Element Robot Controller board and reboots it. You will see the welcome screen appear after a short delay. Once the welcome string appears, the Element Robot Controller is ready to accept commands.

Example:

```
>reset<CR>
```

```
Element, Firmware 1.0.0 Copyright 2010-2012, cmrobot.com
```

```
>
```

blink ledId:rate [ledId:rate]

The *blink* command can blink one of the two onboard green LEDs simultaneously, or individually. Each complex parameter is comprised of an <ledId:blinkRate> pair. The ledId specifies which of the two green LEDs to blink, and blinkRate specifies the delay between blinks. The minimum blink rate is 1, and the largest is 127. A value of 0 turns the led off. Each complex parameter must be separated by one or more spaces.

Example 1: Blink LED 1 at a rate of 50:

```
>blink 1:50<CR>
```

```
ACK
```

```
>
```

Example 2: Blink LED 1 at a rate of 50, and LED 2 at a rate of 100:

```
>blink 1:50 2:100<CR>
```

```
ACK
```

```
>
```

Example 3: Turn off both LEDs:

```
>blink 1:0 2:0<CR>
```

```
ACK
```

```
>
```

cmeps03 [i2c addr]

The *cmeps03* command queries a Devantech CMPS03 Electronic compass module attached to the Element's I2C port. The current heading is returned in Binary Radians, or BRADS. To convert BRADS to DEGREES, multiply BRADS by 360/255 (~1.41). The default I2C address is 0xC1, however another I2C address can be

Element .NET User Guide

supplied as an optional parameter.

Example: Query a CMPS03 for the current heading:

```
>cmps03<CR>  
176  
>
```

cfg enc [encoderType]

The *cfg enc* command configures the internal encoder type to be either single (0) or quadrature (1) type. This information is saved in the EEPROM, so that the configuration will be retained after a reboot. If you are using a quadrature encoder (dual channels), and the Element is configured for single encoder operation, then the second quadrature channel will be ignored. Thus make sure the correct encoder type is configured according to your setup. The *cfg enc* command without a parameter returns the value currently stored in EEPROM.

Example: Configure internal encoder type to be of quadrature type:

```
>cfg enc 1<CR>  
ACK  
>
```

Example: Query internal encoder type stored in EEPROM:

```
>cfg enc<CR>  
1  
>
```

cfg baud [baudRate]

The *cfg baud* command configures the serial baud rate on the Element™. Values can be 0=2400, 1=4800, 2=9600, 3=19200, 4=57600, or 5=115200. You can also type in the actual baud rate string as well (e.g. "19200"). The default baud rate used to communicate with the Element is 19200. The *cfg baud* command without a parameter returns the value currently stored in EEPROM.

Example: Set the baud rate of the Element to 115200:

```
>cfg baud 5<CR>  
ACK  
>
```

Example: Set the baud rate of the Element to 19200:

```
>cfg baud 19200<CR>  
ACK
```

Element .NET User Guide

>

Example: Query value of baud rate stored in EEPROM:

```
>cfg baud<CR>
```

```
19200
```

>

cfg units [unit type]

The `cfg units` command sets the internal units used for sensor readings. Values are 0 for metric mode, 1 for English mode, and 2 for raw mode. In raw mode, `srf04`, `srf05`, `pping`, and `maxez1` return reading in units of 0.4us. `srf08` and `srf10` return readings of 1us. The `cfg units` command without a parameter returns the value currently stored in EEPROM.

Example 1: Set internal units to Metric system

```
>cfg units 0
```

```
ACK
```

>

Example 2: Set internal units to English system:

```
>cfg units 1
```

```
ACK
```

>

Example 3: Query internal unit configuration stored in EEPROM:

```
>cfg units
```

```
1
```

>

getenc encoderId [encoderId]

The `getenc` command returns the values of the encoder count (channel B) for the specified encoder Id(s). NOTE: The encoder counts for channel A are used for internal VPID and DPID algorithms.

Example 1: Query encoder input 1 and input 2 for their current count:

```
>getenc 1 2<CR>
```

```
2400 2388
```

>

Example 2: Query encoder input 1 for its current count:

```
>getenc 1<CR>
```

```
2388
```

>

Element .NET User Guide

clrenc encoderId [encoderId]

The *clrenc* command clears the values of the encoder count (channel B) for the specified encoder Id.

Example 1: Clear encoder count for encoder input 1 and input 2:

```
>clrenc 1 2<CR>
```

```
ACK
```

```
>
```

Example 2: Clear encoder count for encoder input 1:

```
>clrenc 1<CR>
```

```
ACK
```

```
>
```

setio pinId:value [pinId:value]

The *setio* command sets the specified General Purpose I/O line *pinId* (range 0-12) on the Element™ to the specified *value*. Each complex parameter is a <pinId:value> pair, where the valid range of pinId is 0 thru 12, and value can be 0 or 1 which corresponds to 0v or +5V respectively. Also I/O lines 4,5,6,7, 8 and 9 cannot be used if you have servos connected to them. Pin 10, 11, and 12 correspond to the internal h-bridge enable, SCL, and SDA respectively.

Example 1: Set general purpose I/O pins 1 and 2 to 1(+5V) and pins 6 and 8 to 0 (0V)

```
>setio 1:1 2:1 6:0 8:0<CR>
```

```
ACK
```

```
>
```

Example 2: Disable the H-Bridges

```
>setio 10:0<CR>
```

```
ACK
```

```
>
```

Example 3: Set the I2C SCL and SDA pins to 1 (+5V)

```
>setio 11:0 12:0v<CR>
```

```
ACK
```

```
>
```

getio pinId [pinId]

The *getio* command changes the pin, pinId (range 0-12), to an input (if it was an output), and gets the value of the specified General Purpose I/O lines on the Element™. The valid range of I/O pin Ids is 0 thru 12. Pins need to be separated by one or more spaces.

Element .NET User Guide

Example 1: Query the values of the general purpose I/O pins 1, 2, 6, and 8:

```
>getio 1 2 6 8<CR>
```

```
1 1 0 0
```

```
>
```

maxez1 triggerPin outputPin

The maxez1 command queries a Maxbotix MaxSonar-EZ1 sonar sensor connected to the General Purpose I/O lines, triggerPin, and outputPin, for a distance, and returns it in Centimeters. **NOTE: MAKE SURE there's nothing directly in front of the MaxSonar-EZ1 upon power up, otherwise it won't range correctly for object less than 6 inches away!** The sensor reading defaults to use English units (inches). The sensor distance resolution is integer based. **Also, the maxsonar trigger pin is RX, and the echo pin is PW.**

Example 1: Query MaxSonar-EZ1 using pin 3 as the trigger pin, and pin 4 as the output pin.

```
>maxez1 3 4<CR>
```

```
10
```

```
>
```

mogo motorId:vel [motorId:vel]

The *mogo* command sets motor speed using one or more complex parameters containing a <motorId:spd> value pair.

The motorId can be either 1 or 2, which corresponds to the Motor Terminal port.

The vel value specifies the motor velocity, and it's range depends on your VPID settings. See the [VPID parameters](#) section below to determine your MAX velocity. A positive value rotates the motors in one direction, which a negative value rotates the motors in the opposite direction.

You will have to determine which direction is positive for your motors, and connect the motors wires to the terminals on the Element board in the appropriate configuration.

Element .NET User Guide

Example 1: Set motor speed to a velocity of 45 in positive direction for both motor 1 and motor 2. Again your VPID settings determine what the MAX values are for the velocity.

```
>mogo 1:45 2:45<CR>  
ACK  
>
```

Example 2: Stop motor 1 and motor 2.

```
>stop<CR>  
ACK  
>
```

Example 3: Turn motor 1 and motor 2 in opposite directions at a velocity of 55.

```
>mogo 1:55 2:-55<CR>  
ACK  
>
```

vpid prop integ deriv loop

The vpid command gets/sets the PIDL (Proportional, Integral, Derivative, and Loop) parameters for the Velocity PID control on the Element™. If the PIDL parameters are absent, the PIDL values are returned. Otherwise the PIDL parameters are parsed, and saved (in eeprom). For more information on PIDL control, see the PIDL configuration section below.

Example 1: Get the PIDL parameter values:

```
>vpid<CR>  
P:10 I:0 D:0 L:25  
>
```

Example 2: Set the PIDL parameter values on the Element™ to 10, 0, 0, and 700, respectively:

```
>vpid 10:0:0:700<CR>  
ACK  
>
```

digo id:distance:vel [id:distance:vel]

Simply put, the digo command allows you to command your robot to travel a specified distance, at a specified speed. This command uses the internal VPID and DPID algorithms to control velocity and distance.

Element .NET User Guide

Therefore, you must have dual motors, and dual wheel encoders connected to the Element motor ports and encoder inputs.

You must also configure the VPID and DPID parameters so match the physical configuration of your robot. Please see the VPID and DPID configuration sections below to configure the parameters for your robot.

Id specifies the motor id, and can be either 1 or 2, which corresponds to the Motor Terminal port.

Distance specifies the distance (in encoder ticks) which you want your robot to travel.

Vel specifies the motor velocity. **NOTE!!!** [The MAX velocity that can be used for this command depends on your VPID settings.](#)

[If you use a velocity higher than your max velocity, the the PID algorithms will NOT WORK!!!](#) Please see the PID section below to determine your MAX velocity value.

The best way to disable the digo command is to issue a 'stop' command.

Please note that you can query the state of the PID distance algorithm to determine when it has completed the last command by using the 'pids' command (see below).

Example 1: Tell the Element to command your robot to go a distance of 2500 encoder ticks, using a velocity of 25.

```
>digo 1 : 2500 : 28 2 : 2500 : 28<CR>  
ACK  
>
```

dpid prop integ deriv accel

The dpid command gets/sets the PIDA (Proportional, Integral, Derivative, and Acceleration) parameters for the distance PID control on the Element™. If the PIDA parameters are absent, the PIDA values are returned. Otherwise the PIDA parameters are parsed, and saved (in eeprom). For more information on PIDA control, see the DPID configuration section below

Element .NET User Guide

Example 1: Get the PIDA parameter values:

```
>vpid<CR>
P:10 I:0 D:0 A:25
>
```

Example 2: Set the PIDA parameter values on the Element™ to 1, 0, 0, and 10, respectively:

```
>vpid 1:0:0:10<CR>
ACK
>
```

rpids <s|t>

The rpid command sets the default PID params known to work with either the Stinger or Traxster Robotic Kits in the firmware. This makes it quick and easy to set up the PID params for both robots.

Example 1: Set the default Stinger PID params:

```
>rpids s<CR>
ACK
>
```

Example 2: Set the default Traxster PID params:

```
>rpids t<CR>
ACK
>
```

pids

Once a digo command is issued, an internal state variable within the firmware is set to '1', and it stays in that state until the algorithm has completed. Upon completion, the state is set to '0'. The 'pids' command simply returns the value of the internal variable to determine if the algorithm is currently busy, or if it has finished, thus allowing subsequent digo commands to be issued w/o clobbering previous ones.

Example 1: Query PID state: (PID algorithm busy)

```
>pids<CR>
1
>
```

Example 2: Query PID state: (PID algorithm completed)

```
>pids<CR>
0
```

Element .NET User Guide

>

pwm [r:rate] id:pwm [id:pwm] The pwm command sets the Pulse Width Modulation value for Motor 1 & Motor 2. Each complex parameter is a motor <motorId:pwm value> pair, where the motor id can be 1 or 2, and the pwm value can be -100 to 100. Each complex parameter pair is separated by one or more spaces.

The optional r:rate complex parameter is a switch that allows the motor(s) speed(s) to be ramped up or down to the specified speed from the current motor speed. **NOTE: the r:rate parameter MUST occur before any id:pwm parameter for proper operation, and it is optional!** A ramping value of 0 produces no ramping, where a value equal to the pwm value produces the slowest amount of ramping. A ramping loop is executed every 51 milliseconds. Therefore, if you specify a ramp value of 100, for a pwm of 100, the motor will reach full velocity within 5.1 seconds. If your current motor speed is 0, and you want to ramp to 100% velocity within 2 seconds, then:

Ramp = 2000 msec / 51 msec

Ramp = 39

PWM values can be ramped from any negative value to any positive value, and vice versa, so long as the pwm values are within range (-100 to 100).

NOTE: The firmware caps the ramping value to the pwm value to ensure calculations are performed correctly. So if you specify a ramping value of 85 for a pwm value of 75, the ramping value will be limited to 75.

The pwm value does not take the PID parameters into account. So, if you want to control motor speed without the use of PID, control it with the pwm command.

Example 1: Set motor 1 to -50% pwm:

```
>pwm 1:-50<CR>
```

```
ACK
```

```
>
```

Example 2: Set motor 1 and 2 to 100% pwm:

Element .NET User Guide

```
>pwm 1:100 2:100<CR>
```

```
ACK
```

```
>
```

Example 3: Set motor 1 and 2 to 100% pwm, and ramp up to full speed within 3 seconds (assumes motors are stopped):

```
>pwm r:59 1:100 2:100<CR>
```

```
ACK
```

```
>
```

Example 3: Ramp both motors up to 100% pwm in the positive range, then ramp them down to full speed in the opposite direction. This will ramp their pwm values from 100 to 0, then from 0 to -100 in one step. This would be useful if you wanted to stop and reverse your robot in a very smooth fashion.

```
>pwm r:100 1:100 2:100<CR>
```

```
ACK
```

```
>pwm r:100 1:-100 2:-100<CR>
```

```
ACK
```

```
>
```

Example 4: Ramp motor 1 and 2 from their current speed (could be any speed), down to 0 slowly.

```
>pwm r:100 1:0 2:0<CR>
```

```
ACK
```

```
>
```

Example 5: Stop motor 1 and 2:

```
>stop<CR>
```

```
ACK
```

```
>
```

step dir:speed:steps

The `step` command is used to step a bipolar stepper motor in direction `dir`, at the specified speed, for the specified number of steps.

The `dir` parameter specifies a CW or CCW rotational direction, and its value can be either 0 (CCW) or 1(CW). Your specific direction is based on the way that you have your bipolar motor connected to the Element. The `speed` parameter can be a value from 0 to 100.

Element .NET User Guide

The *steps* parameter specifies the maximum number of steps to take. A value of 0 means step infinitely. Internally, this number is stored in an unsigned 32 bit variable, so the user can specify a larger number of steps.

You may stop the step by either issuing a step command w/ a 0 speed, or simply sending a 'stop' command.

Example 1: Step a bipolar direction 1 at 100% speed for an infinite number of steps

```
>step 1 100 0<CR>
ACK
>
```

Example 2: Stop the bipolar stepper motor:

```
>stop<CR>
ACK
>
```

sweep speed:steps

The sweep command is used to sweep a bipolar motor, for *step* number of steps, at *speed* (0-100), thus providing a sweeping motion. The initial rotational direction of sweep is in the CW direction.

Upon initial receipt of the command, the firmware will sweep the motor for ½ of the number of *steps* specified, starting in a CW direction. Once that number of steps has occurred, the sweep direction will change, and subsequent sweeps will rotate for the full amount of steps. Thus, the starting point for the motor is in the middle of each sweep.

You may stop the sweep by either issuing a sweep command w/ a 0 speed, or simply sending a 'stop' command.

Example 1: Sweep a bipolar stepper motor 125 steps in a CW motion, the 125 steps at 75% speed, starting in the CW direction.

```
>sweep 1 75 125<CR>
ACK
>
```

Example 2: Stop the bipolar stepper motor from sweeping:

```
>stop<CR>
```

Element .NET User Guide

ACK
>

stop

The stop command immediately stops motor 1 and 2 connected to the motor ports. This will override any pwm, mogo, or digo commands which are currently active.

Example 1: Stop all motors:

```
>stop<CR>  
ACK  
>
```

sensor id [idN]

The sensor command returns the raw A/D (8 bit) reading from the analog sensor ports 0-5. Multiple values can be read at a time by specifying multiple pins as a parameters (range 0-5). Pin 5 is 1/3 of the voltage of the power supply for the Element™. To calculate the battery voltage, simply multiply the value returned by Sensor 5 by 15/1028.

Example: Read the value of analog port 3

```
>sensor 3<CR>  
768  
>
```

Example: Read multiple analog values. NOTE: Readings will be returned in the same order as the pins which they represent.

```
>sensor 0 1 2<CR>  
768 522 242  
>
```

servo id:pos [id:pos]

The servo command sets a servo connected to General Purpose I/O port the specified position. The value of the position can range from – 99 to 100, where 0 is the center position. Setting the position to -100 will disable the servo, allowing it to turn freely by hand.

Each parameter is a <servo id:position> pair, where the servo id can be 1,2,3,4,5, or 6.

Element .NET User Guide

Below is the mapping table depicting servo Id to GPIO pin relationship:

| Servo Id (firmware) | 1 | 2 | 3 | 4 | 5 | 6 |
|----------------------|---|---|---|---|---|---|
| GPIO Pin Id (header) | 8 | 9 | 6 | 7 | 4 | 5 |

Servo ID to GPIO Pin Mapping

Each complex parameter is separated by one or more spaces.

Example 1: Set servo 1 and 2 to position 35:

```
>servo 1:35 2:35<CR>
```

```
ACK
```

```
>
```

Example 2: Center servo 4 and 5:

```
>servo 4:0 5:0<CR>
```

```
ACK
```

```
>
```

Example 3: Set servo 2 to position -85:

```
>servo 2:-85<CR>
```

```
ACK
```

```
>
```

sp03 [i2cAddr]

The sp03 command instructs a Devantech SP03 Speech Synthesizer to speak the appropriate phrase. If a character representing a number in the range of 0 to 30, then the SP03 will speak previously programmed canned phrases. If a phrase is sent, then it will speak the phrase. An optional I2C address can also be specified. Otherwise, the default I2C address of 0xC4.

Example 1: Speak canned phrase 12:

```
>sp03 12<CR>
```

```
ACK
```

```
>
```

Example 2: Speak the following phrase:

```
>sp03 Hello World
```

```
<CR>
```

```
ACK
```

```
>
```

Element .NET User Guide

srf04 triggerPin outputPin

The srf04 command queries an SRF04 sonar sensor connected to the General Purpose I/O lines triggerPin and outputPin, for a distance and returns it in the units configured (default is English – inches).

If the Element units are configured (using “cfg units”) for raw mode, srf04 returns readings in units of 0.4us, and the max distance returned is 65000 (out of range). When configured for English units, max distance returned is 100 inches (out of range), and when configured

for Metric units, max distance returned is 255 (out of range). NOTE: Sonar distance resolution is integer based.

Example 1: Query SRF04

```
>srf04 5 6<CR>
```

```
7
```

```
>
```

srf05 pinId

pping pinId

The srf05/Ping command queries an SRF05/Ping sonar sensor connected to the General Purpose I/O line pinId for a distance, and returns it in the units configured (default is English – inches).

If the Element units are configured (using “cfg units”) for raw mode, pping and srf05 return readings in units of 0.4us, and the max distance returned is 65000 (out of range). When configured for English units, max distance returned is 100 inches (out of range), and when configured for Metric units, max distance returned is 255 (out of range). Sonar distance resolution is integer based.

Example 1: Query SRF05

```
>srf05 3<CR>
```

```
14
```

```
>
```

Example 3: Query Ping

```
>pping 2<CR>
```

```
8
```

```
>
```

srf08 [i2cAddr]

Element .NET User Guide

srf10 [i2cAddr]

The srf08/srf10 command queries a Devantech SRF08/SRF10 sonar sensor at address i2cAddr for a distance reading in the units configured (default is English – inches). The i2cAddr parameter is optional, and defaults to 0xE0 for both sensors. The i2c address can be changed for any i2c module using the i2cp command. Sonar distance resolution is integer based.

If the Element units are configured (using “cfg units”) for raw mode, srf08 and srf10 return readings in units of 1us.

Example 1: Query an SRF08 for distance:

```
>srf08<CR>
32
>
```

Example 2: Query an SRF10 for distance:

```
>srf10<CR>
40
>
```

tpa81 [i2cAddr]

The tpa81 command queries a Devantech TPA81 thermopile sensor for temperature values. It returns 8 temperature values.

Example:

```
>tpa81<CR>
11 11 12 17 19 21 16 13
>
```

vel

The vel command returns the left and right wheel velocities. The velocity returned is based on the PIDL parameter configuration.

Example:

```
>vel<CR>
200 203
>
```

restore

Restores the factory default settings, and resets the board. NOTE: This will erase any configurations you have saved to EEPROM,

Element .NET User Guide

including VPID, DPID, and baud rate settings.

Example:

```
>restore<CR>
> Element, firmware v1.5.0
Copyright 2010-2012, cmRobot Corporation
>
```

```
>i2c r 80 1<CR>
28
>
```

i2cp currAddr newAddr

The `i2cp` command programs an I2C device connected to the I2C port on the Element™ from the current I2C address specified (`currAddr`) to the new I2C address specified (`newAddr`). NOTE: no other devices

can be connected to the I2C port/bus while programming a single device. This command is **ONLY** used for the Devantech SRF08, SRF10, and TPA81 I2C Devices.

Example: Set an I2C device with the current address 240 to the new I2C address 250.

```
>i2cp 240 250<CR>
ACK
>
```

i2c <r|w> <addr> [data]

The flexible `i2c` command allows you to execute a generic `i2c read`, or `write` command to the specified device at address `addr`. Depending on whether you issue a read or write, additional parameters vary.

If you are issuing a read, then the following attributes should be used:

```
>i2c r <device address> <numBytesToRead>
```

If you are issuing a write, then the following attributes should be used:

```
>i2c w <device address> <byte2write 0> <byte2write 1>...
<byte2write N>
```

An `i2c read` returns a value queried from the device at the specified address. An `i2c write` returns an ACK if the write was successful. Both command flavors return a NACK if the command fails.

Element .NET User Guide

If you have a device which communicates over the I2C bus, you should be able to communicate with that device from the Element using the i2c command. A good example of a device which uses I2C that can be interfaced using the i2c commands is the Gamoto PID Motor Controller. We have included many examples below to show you how to use this flexible command.

Example 1: Read the firmware version of a Gamoto PID Motor Controller:

```
>i2c w 158 178<CR>
ACK
>i2c r 158 1<CR>
16
>
```

Example 2: Set the speed of the motor, connected to a Gamoto PID Motor Controller to 50%:

```
>i2c w 158 43 17<CR>
ACK
>i2c w 158 60 64<CR>
ACK
>
```

Example 3: Stop the motor, connected to a Gamoto PID Motor Controller:

```
>i2c w 158 43 17<CR>
ACK
>i2c w 158 60 0<CR>
ACK
>
```

Example 4: Query an SRF08 sonar module (at address 244) for distance and light reading, using English units:

```
>i2c w 244 0 80<CR>
ACK
>i2c r 244 3<CR>
20 0 63
>
```

Example 5: Query the temperatures on a TPA81, located at address 214:

```
>i2c w 214 1<CR>
ACK
>i2c r 214 9<CR>
38 35 36 38 38 37 36 36 35
>
```

Example 7: Command an SP03 to speak the phrase 'hello' on the fly.

Element .NET User Guide

```
>i2c w 196 0 0 0 5 3 104 101 108 108 111 0<CR>  
ACK  
>i2c w 196 0 64<CR>  
ACK  
>
```

// NOTE: The characters in the word 'hello' were spelled out in the command using their ASCII decimal equivalents (highlighted in bold).

Upgrading the Firmware:

The Element's firmware can easily be upgraded using the Tiny Bootloader program using a few easy steps.

Warranty & Disclaimer Information:

The Element Robot Controller is guaranteed to be free of defects in hardware and craftsmanship upon delivery. The Element Robot Controller is not warranted against damage caused by user error, such as reverse polarity damage, over voltage, or simple abuse. Please handle and use the Element Robot Controller responsibly. Please note that we perform a software test on every Element Robot Controller before it leaves our facility, so we know for a fact that they arrive in working order.

We do not guarantee the firmware to be free of defects. If you do find a bug please report it to us, and we will fix it as soon as possible, and provide an updated firmware version. Please send bugs and or questions to our support email:

support@cmrobot.com

The Element Robot Controller should not be used in any form of life support devices used in hospitals or doctors offices under any circumstances. The Element Robot Controller should only be used for robotics development and research where there is no risk to life.

Element Libraries & Documentation:

The Element Robot Controller comes with several full featured libraries, including .NET.

The Element .Net Library, QuickStart Guide & Documentation can be found here;

<http://www.cmrobot.com/#!element-.net/vstc2=resources>

PID Configuration Examples:

Velocity PID (VPID)

This section provides some help on the PID configuration. Before using the mogo command, the PID settings should be set depending on your max motor velocity, drivetrain gearing, encoder resolution, etc.

Element .NET User Guide

It is important to understand that the PID algorithm is already implemented in the Element's firmware, so all you have to do is figure out your drivetrain specifics so that the PID algorithm works.

What are the PIDL parameters? Well, the P is for Proportional, I is for Integral, D is for Derivative, and L is for Loop multiplier. There are many good tutorials on PID loops available on the web, reading some of these should help.

- http://www.seattlerobotics.org/encoder/200108/using_a_pid.html
- <http://www.engin.umich.edu/group/ctm/PID/PID.html>

The only thing not covered in these tutorials would be the L parameter. The basic loop for the PID calculations occurs approximately every 1.6ms. This is too fast for many wheel-encoder combinations so the L is used to multiply the number of basic loops before the PID calculations are done. For example if the L is set to 10 then the PID loop will occur every 16ms.

So, how are the PIDL parameters determined for your robot? Follow either example below.

Configuration by Experimentation:

The way to determine the proper setting for L is to set the PIDL settings to a known value like 10 0 0 100, then use the pwm command to set the value to 100. This will cause the motor to run at its maximum velocity. Remember, pwm is not affected by the PIDL parameters. Only the mogo and digo commands use the PIDL parameters.

You can then use the vel command to get the motor velocities in ticks/loop. If the number is in the 50-250 range then it is good, otherwise change the L setting until this range is reached. If the VEL is too high you will not be able to reach 100% power, if it is too low the velocity will "Hunt" for the correct speed.

After the proper L is determined the rest of the PID parameters can be set. When setting the P it is important to realize that the larger the number the faster the response will be, but if it is too high then again the motor may overshoot the desired velocity then undershoot when it tries to correct and therefore again "Hunt" for the correct speed.

A D value of approximately 1/2 the P value will help to dampen the oscillations. The I parameter is not used that often in Velocity PID loops but is included for completeness in the algorithm.

Configuration by calculation:

Let's say your robot has the following measurements and specifications:

- Wheel diameter: 2.5" Wheel Circumference: 7.854"
- Max Motor RPM: 175 RPM
- Gear Reduction Ratio: 50:1
- Encoder Resolution (ticks per rev): 8
- Distance per encoder tick: $7.854" / 8 \times 50 = 0.020"$
- Ticks per unit distance: 51.02 ticks/inch of linear travel
- Ticks per second: 1166.67 ticks/sec (calculated using motor RPM, gear reduction, encoder resolution, and wheel circumference)

$1166.67 \text{ ticks per second} \times 1.6 \text{ msec per pid loop} = 1.867 \text{ ticks every time the pid calculation runs}$. Let's say that we choose a target velocity of 130. Thus, $130/1.867 \text{ ticks per pid loop execution}$, we would arrive at an L value of 69.6, or basically 70.

From here, you can customize the PID parameters as outlined above.

Element .NET User Guide

Distance PID (DPID)

For the DIGO command there is another PID configuration that oversees the VPID controls to allow the robot to travel a specific distance and then stop.

The VPID parameters above should be tuned first so that the MOGO command works smoothly. The DIGO command simply modifies the velocity on-the-fly that the mogo function is trying to maintain. By doing this all VPID parameters are still valid and must work correctly before trying to use DIGO.

The DPID parameters are similar to the Vpid parameters except that the last parameter is an Acceleration value. This is the value that determines how long it will take the motors to reach the full speed set by the DIGO command. A smaller value will allow a smoother ramp up of speed, while a larger value will get you to maximum speed faster (but may lurch at the beginning).

The default value is 1. The acceleration value is added to the speed every time a motor control loop is executed (Based on the L parameter). Thus, if L is set to 50, the loop is performed every $50 * 1.6\text{ms} = 80\text{ms}$, then, the speed is increased by A. If you set $A = 1$, and use digo to go a speed of 25, then it will take $25 * 80\text{ms} = 2$ seconds to reach full speed. This is a nice smooth takeoff speed for most robots.

The default Values for the DPID is 1 0 0 1, More discussions on the PID portion of the DPID parameters will be coming shortly.

Querying PID Status:

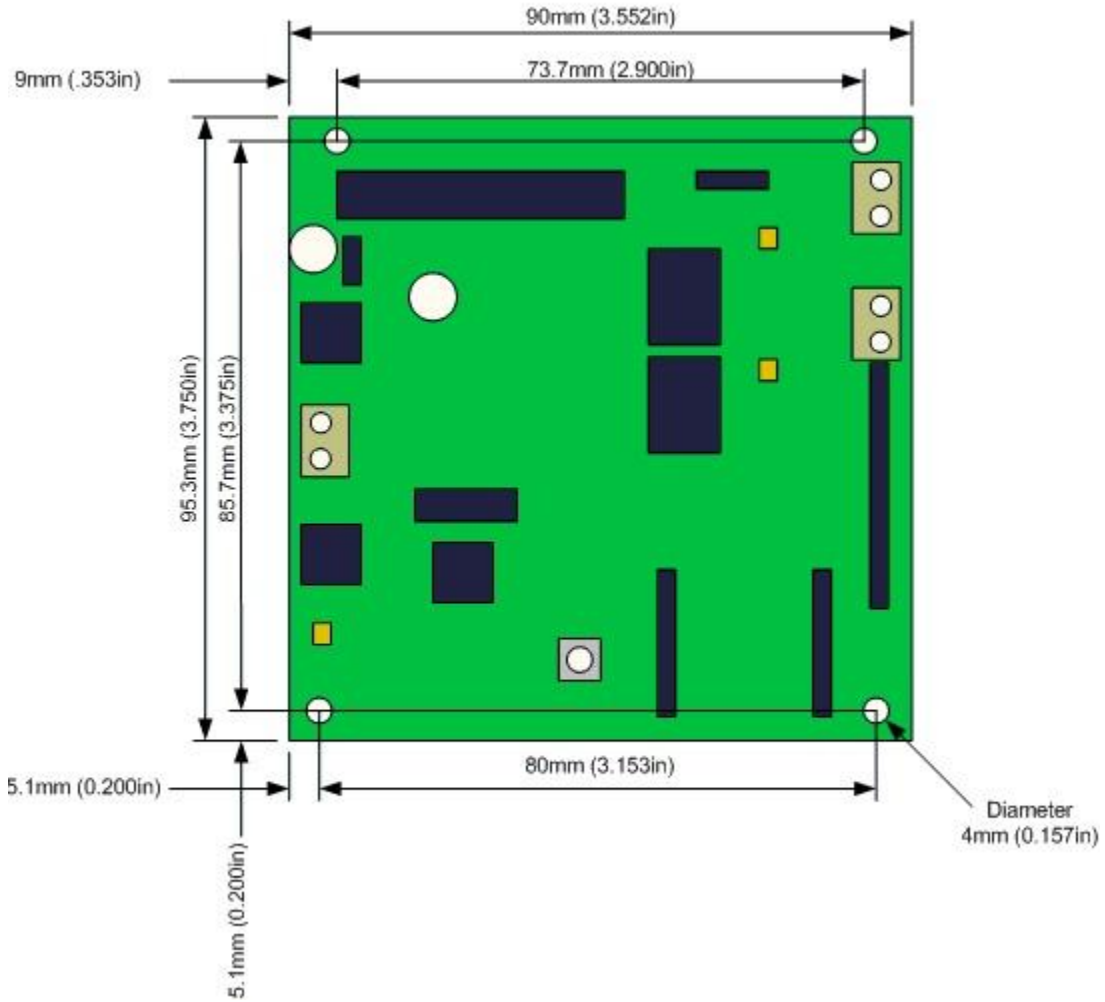
The current state of the PID algorithms can be queried using the 'pids' command. This is useful if you want to poll the state of the PID algorithm to determine when it has completed it's latest command.

For instance, if you issue a 'digo' command to make your motors rotate for 20,000 ticks, it may take up to 30 seconds to complete. If you want to issue a command to rotate the motors a specified distance after the initial command completes, simply use the 'pids' command to query the state until it reads '0', then issue your next 'digo' command. The state will be '1' while the PID algorithm is busy, and '0' when it has completed.

Element Robot Controller Dimensions:

The Element Robot Controller dimensions and bolt hole locations are based on the PC104 spec. Exact dimensions are shown below.

Element .NET User Guide



Element Robot Controller Maximum Ratings

DC Characteristics:

| Symbol | Characteristic/Device | Min | Typ† | Max | Units | Conditions |
|--------|-----------------------|-------|------|------|-------|--------------------------------------|
| VDD | Supply Voltage | 7.5 | 9.6 | 12.0 | V | Using on-board H-bridge |
| VDD | Supply Voltage | 6.5 | 9.6 | 12.0 | V | Using off-board H-bridge |
| IDD | Supply Current | ----- | 100 | 200 | mA | Excluding H-bridge and servo current |
| IDD | Supply Current | ----- | 100 | 2200 | mA | Excluding H-bridge current |

Element .NET User Guide

| | | | | | | |
|-----|----------------|-------|------------|--------------|-----------|---|
| IDD | Supply Current | ----- | 100 | 10200 | mA | Including H-bridge and Servo current |
|-----|----------------|-------|------------|--------------|-----------|---|

Absolute Maximum Ratings †

| | |
|---|-----------------------|
| Ambient temperature under bias..... | -55 to +125°C |
| Storage temperature | -65°C to +150°C |
| Voltage on any pin with respect to VSS..... | -0.3V to (VDD + 0.3V) |
| Voltage on VDD with respect to VSS | -0.3 to +7.5V |
| Total power dissipation on digital I/O pins..... | 1.0W |
| Maximum current out of VSS pin..... | 300 mA |
| Maximum current into VDD pin | 250 mA |
| Input clamp current, I _{IK} (V _I < 0 or V _I > VDD)..... | 20 mA |
| Output clamp current, I _{OK} (V _O < 0 or V _O > VDD)..... | 20 mA |
| Maximum output current sunk by any I/O | 25 mA |
| Maximum output current sourced by any I/O pin..... | 25 mA |

† NOTICE: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

ASCII Character Set

We have provided the ASCII character set below as a quick reference:

| Char | Dec | Oct | Hex | | Char | Dec | Oct | Hex | | Char | Dec | Oct | Hex | | Char | Dec | Oct | Hex |
|-------------|-----------|-------------|-------------|--|------|-----|------|------|--|------|-----|------|------|--|------|-----|------|------|
| (nul) | 0 | 0000 | 0x00 | | (sp) | 32 | 0040 | 0x20 | | @ | 64 | 0100 | 0x40 | | ` | 96 | 0140 | 0x60 |
| (soh) | 1 | 0001 | 0x01 | | ! | 33 | 0041 | 0x21 | | A | 65 | 0101 | 0x41 | | a | 97 | 0141 | 0x61 |
| (stx) | 2 | 0002 | 0x02 | | " | 34 | 0042 | 0x22 | | B | 66 | 0102 | 0x42 | | b | 98 | 0142 | 0x62 |
| (etx) | 3 | 0003 | 0x03 | | # | 35 | 0043 | 0x23 | | C | 67 | 0103 | 0x43 | | c | 99 | 0143 | 0x63 |
| (eot) | 4 | 0004 | 0x04 | | \$ | 36 | 0044 | 0x24 | | D | 68 | 0104 | 0x44 | | d | 100 | 0144 | 0x64 |
| (enq) | 5 | 0005 | 0x05 | | % | 37 | 0045 | 0x25 | | E | 69 | 0105 | 0x45 | | e | 101 | 0145 | 0x65 |
| (ack) | 6 | 0006 | 0x06 | | & | 38 | 0046 | 0x26 | | F | 70 | 0106 | 0x46 | | f | 102 | 0146 | 0x66 |
| (bel) | 7 | 0007 | 0x07 | | ' | 39 | 0047 | 0x27 | | G | 71 | 0107 | 0x47 | | g | 103 | 0147 | 0x67 |
| (bs) | 8 | 0010 | 0x08 | | (| 40 | 0050 | 0x28 | | H | 72 | 0110 | 0x48 | | h | 104 | 0150 | 0x68 |
| (ht) | 9 | 0011 | 0x09 | |) | 41 | 0051 | 0x29 | | I | 73 | 0111 | 0x49 | | i | 105 | 0151 | 0x69 |
| (nl) | 10 | 0012 | 0x0a | | * | 42 | 0052 | 0x2a | | J | 74 | 0112 | 0x4a | | j | 106 | 0152 | 0x6a |
| (vt) | 11 | 0013 | 0x0b | | + | 43 | 0053 | 0x2b | | K | 75 | 0113 | 0x4b | | k | 107 | 0153 | 0x6b |
| (np) | 12 | 0014 | 0x0c | | , | 44 | 0054 | 0x2c | | L | 76 | 0114 | 0x4c | | l | 108 | 0154 | 0x6c |
| (cr) | 13 | 0015 | 0x0d | | - | 45 | 0055 | 0x2d | | M | 77 | 0115 | 0x4d | | m | 109 | 0155 | 0x6d |
| (so) | 14 | 0016 | 0x0e | | . | 46 | 0056 | 0x2e | | N | 78 | 0116 | 0x4e | | n | 110 | 0156 | 0x6e |
| (si) | 15 | 0017 | 0x0f | | / | 47 | 0057 | 0x2f | | O | 79 | 0117 | 0x4f | | o | 111 | 0157 | 0x6f |
| (dle) | 16 | 0020 | 0x10 | | 0 | 48 | 0060 | 0x30 | | P | 80 | 0120 | 0x50 | | p | 112 | 0160 | 0x70 |
| (dc1) | 17 | 0021 | 0x11 | | 1 | 49 | 0061 | 0x31 | | Q | 81 | 0121 | 0x51 | | q | 113 | 0161 | 0x71 |
| (dc2) | 18 | 0022 | 0x12 | | 2 | 50 | 0062 | 0x32 | | R | 82 | 0122 | 0x52 | | r | 114 | 0162 | 0x72 |
| (dc3) | 19 | 0023 | 0x13 | | 3 | 51 | 0063 | 0x33 | | S | 83 | 0123 | 0x53 | | s | 115 | 0163 | 0x73 |
| (dc4) | 20 | 0024 | 0x14 | | 4 | 52 | 0064 | 0x34 | | T | 84 | 0124 | 0x54 | | t | 116 | 0164 | 0x74 |
| (nak) | 21 | 0025 | 0x15 | | 5 | 53 | 0065 | 0x35 | | U | 85 | 0125 | 0x55 | | u | 117 | 0165 | 0x75 |
| (syn) | 22 | 0026 | 0x16 | | 6 | 54 | 0066 | 0x36 | | V | 86 | 0126 | 0x56 | | v | 118 | 0166 | 0x76 |
| (etb) | 23 | 0027 | 0x17 | | 7 | 55 | 0067 | 0x37 | | W | 87 | 0127 | 0x57 | | w | 119 | 0167 | 0x77 |
| (can) | 24 | 0030 | 0x18 | | 8 | 56 | 0070 | 0x38 | | X | 88 | 0130 | 0x58 | | x | 120 | 0170 | 0x78 |

Element .NET User Guide

| | | | | | | | | | | | | | | | | | | |
|-------|----|------|------|--|---|----|------|------|--|---|----|------|------|--|-------|-----|------|------|
| (em) | 25 | 0031 | 0x19 | | 9 | 57 | 0071 | 0x39 | | Y | 89 | 0131 | 0x59 | | y | 121 | 0171 | 0x79 |
| (sub) | 26 | 0032 | 0x1a | | : | 58 | 0072 | 0x3a | | Z | 90 | 0132 | 0x5a | | z | 122 | 0172 | 0x7a |
| (esc) | 27 | 0033 | 0x1b | | ; | 59 | 0073 | 0x3b | | [| 91 | 0133 | 0x5b | | { | 123 | 0173 | 0x7b |
| (fs) | 28 | 0034 | 0x1c | | < | 60 | 0074 | 0x3c | | \ | 92 | 0134 | 0x5c | | | 124 | 0174 | 0x7c |
| (gs) | 29 | 0035 | 0x1d | | = | 61 | 0075 | 0x3d | |] | 93 | 0135 | 0x5d | | } | 125 | 0175 | 0x7d |
| (rs) | 30 | 0036 | 0x1e | | > | 62 | 0076 | 0x3e | | ^ | 94 | 0136 | 0x5e | | ~ | 126 | 0176 | 0x7e |
| (us) | 31 | 0037 | 0x1f | | ? | 63 | 0077 | 0x3f | | _ | 95 | 0137 | 0x5f | | (del) | 127 | 0177 | 0x7f |

Contact Information:

© 2010-2012 cmRobot a registered trademark of QuPiron Technology Corporation

1 Tara Boulevard Suite 200
Nashua, NH 03062
1.888.762.2503 - Phone
1.888.762.2513 – Fax
www.cmrobot.com
info@cmrobot.com