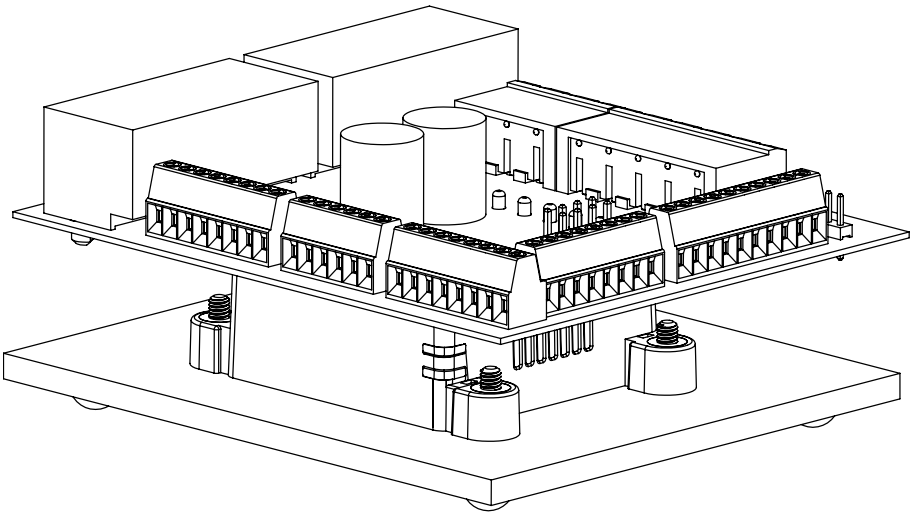
Portescap

A REGAL REXNORD BRAND

PERFORMANCE MOTION DEVICES



Portescap Motor Application Development Kit

User Manual

Revision 0.95 / March 2024

www.portescap.com | www.pmdcorp.com

NOTICE

Warranty

Performance Motion Devices, Inc. warrants that its products shall substantially comply with the specifications applicable at the time of sale, provided that this warranty does not extend to any use of any Performance Motion Devices, Inc. product in an Unauthorized Application (as defined below). Except as specifically provided in this paragraph, each Performance Motion Devices, Inc. product is provided "as is" and without warranty of any type, including without limitation implied warranties of merchantability and fitness for any particular purpose.

Performance Motion Devices, Inc. reserves the right to modify its products, and to discontinue any product or service, without notice and advises customers to obtain the latest version of relevant information (including without limitation product specifications) before placing orders to verify the performance capabilities of the products being purchased. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement and limitation of liability.

Unauthorized Applications

Performance Motion Devices, Inc. products are not designed, approved or warranted for use in any application where failure of the Performance Motion Devices, Inc. product could result in death, personal injury or significant property or environmental damage (each, an "Unauthorized Application"). By way of example and not limitation, a life support system, an aircraft control system and a motor vehicle control system would all be considered "Unauthorized Applications" and use of a Performance Motion Devices, Inc. product in such a system would not be warranted or approved by Performance Motion Devices, Inc.

By using any Performance Motion Devices, Inc. product in connection with an Unauthorized Application, the customer agrees to defend, indemnify and hold harmless Performance Motion Devices, Inc., its officers, directors, employees and agents, from and against any and all claims, losses, liabilities, damages, costs and expenses, including without limitation reasonable attorneys' fees, (collectively, "Damages") arising out of or relating to such use, including without limitation any Damages arising out of the failure of the Performance Motion Devices, Inc. product to conform to specifications.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent procedural hazards.

Disclaimer

Performance Motion Devices, Inc. assumes no liability for applications assistance or customer product design. Performance Motion Devices, Inc. does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of Performance Motion Devices, Inc. covering or relating to any combination, machine, or process in which such products or services might be or are used. Performance Motion Devices, Inc.'s publication of information regarding any third party's products or services does not constitute Performance Motion Devices, Inc.'s approval, warranty or endorsement thereof.

Patents

Performance Motion Devices, Inc. may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

Patents and/or pending patent applications of Performance Motion Devices, Inc. are listed at https://www.pmdcorp.com/company/patents.

Related Documents

ION/CME N-Series Digital Drive User Manual

Complete description of the N-Series ION family of digital drives including complete mechanical, electrical, and software specifications.

Magellan Motion Control IC User Guide

Complete description of the Magellan Motion Control IC features and functions with detailed theory of its operation.

C-Motion Magellan Programming Reference

Descriptions of all Magellan Motion Control IC commands, with coding syntax and examples, listed alphabetically for quick reference.

C-Motion PRP II Programming Reference

Describes C-Motion language function calls and associated PRP-formatted packets along with data types for the ION/CME N-Series ION Digital Drives.

C-Motion Engine Development Tools Manual

Describes the C-Motion Engine Development Tools that allow user application code to be created and compiled on a host PC, then downloaded, executed and monitored on a C-Motion Engine module.

Table of Contents

This page intentionally left blank.

List of Figures

This page intentionally left blank.

1. Introduction

1

In This Chapter

1.1 Overview

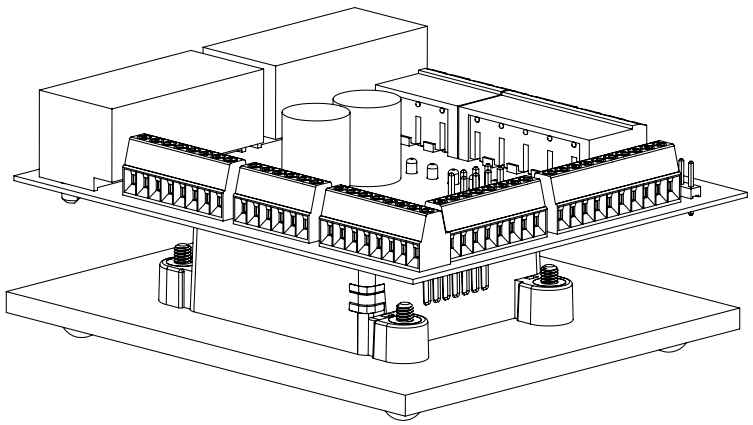


Figure 1-1: Portescap Motor Application Development Kit Controller

The Portescap Motor Application Development Kit is an integrated board/software package that serves as a tool for exercising and building systems with Portescap three-phase Brushless DC motors.

Portescap BLDC Motors provide exceptional torque and speed, higher efficiency and package sizes suited to meet a broad range of application needs. Portescap Brushless DC motors meet even the most stringent requirements. A Brushless DC motor offers excellent speed and position control, long life, and high torque density.

Portescap slotless electric Brushless DC motors use a cylindrical ironless coil made in the same winding technique as ironless DC motors, while the slotted DC Brushless motors are built with stators consisting of the latest technology in lamination and windings. Both are electronically commutated and have 3 winding phases.

Portescap slotted Brushless DC motors are targeted towards performance-critical surgical power devices and also offer sterilization capabilities. For more information refer to https://www.portescap.com.

1.2 ION/CME N-Series Digital Drive

The controller used with the Portescap Motor Application Development Kit is an ION/CME N-Series Digital Drive mounted between a metallic heat sink base and an interconnect board. N-Series IONs are single-axis motion controllers manufactured by Performance Motion Devices Inc. with integrated power electronics and communications. Their compact size, range of power output levels, and high level of connectivity make them an ideal solution for embedded or distributed motion control applications that require high performance in a small envelope.

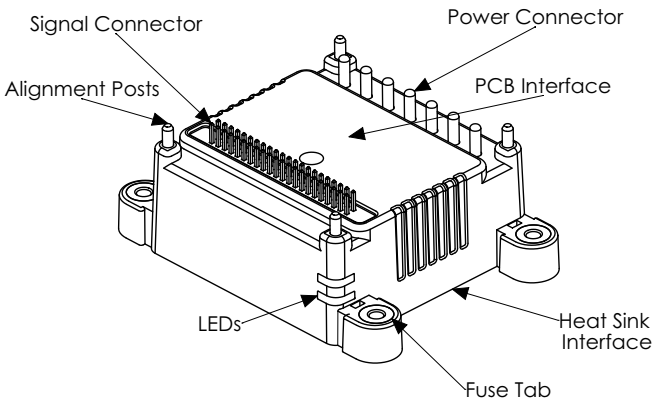


Figure 1-2: N-Series ION Drive

ION N-Series Drives are based on PMD's Magellan Motion Control IC and perform profile generation, encoder position feedback, position servo compensation, step motor stall detection, brushless DC motor commutation, microstep generation, digital current/torque control, and more. All members of the ION family protect from overcurrent, undervoltage, overvoltage, overtemperature, and short-circuit faults.

For a more information on N-Series IONs refer to the *ION/CME N-Series Digital Drive User Manual*.

1.3 Developer Kit Part Numbers

The following Portescap Motor Application Development Kits are available:

DK P/N	ION P/N Installed	Host Interface	Power Level
DK431S0056/06F36	DD431S0056/06F36	Serial	Medium

1.4 Developer Kit Components List

The Portescap Motor Application Development Kit contains the following components:

- Developer Kit interconnect board with N-Series ION Drive soldered in place
- Metal heat sink base with rubber feet
- USB to 3-pin serial programming cable
- USB to 9-pin serial cable

The following software and design materials are also included with each N-Series ION Developer Kit:

- Pro-Motion Windows-based exercisor
- C-Motion Engine Software Developer Kit:
- A complete toolset for the creation of user-specific applications running on ION/CME or host
- An open-source compiler
- C-Motion libraries
- PDFs of all N-Series ION documentation

For a detailed list of components for each N-Series ION DK type refer to Section 4.7, "Developer Kit Hardware Contents."

1.5 Getting Started & Installation Overview

1. Before using the N-Series ION Drive the software must be installed. See Section 1.7, "Software Installation," for instructions on installing the software.
2. Next, connect your system's motor, power, encoder, and sensors to operate the motion hardware. See Section 1.9, "Hardware Setup," for a detailed description of the available connections.
3. Connect the N-Series Drive to the host PC via the 3-pin programming cable. See Section 1.9.5, "Making Connections to the PC," for a description of this procedure.
4. Once this hardware configuration is complete, the final step to finish the installation is to perform a setup procedure and functional test of the connected motor axis. See Section 1.11, "First-Time System Setup & Verification," for a description of this procedure.

Once these steps have been accomplished, the installation is complete, and the DK setup is ready for operation.

1.6 Recommended Hardware

To install an N-Series ION developer kit the following hardware is recommended.

- PC with Intel (or compatible) processor, 1 Gbyte of available disk space and 256 MB of available RAM. The supported PC operating systems are Windows XP, Vista, Windows 7, Windows 8, Windows 10.
- One Portescap Brushless DC motor.
- Cables as required to connect to the motor and associated motion hardware such as encoder signals.
- Power supply. The N-Series ION Drive requires only a single voltage supply. Its internal logic and other circuitry is powered from this input voltage using an internal DC to DC converter. The input voltage range is 12-56V.

1.7 Software Installation

All software applications are designed to work with Microsoft Windows.

To install the software and other DK content on your PC:

1. Go to the Software Downloads www.pmdcorp.com/portescapsw and select download for "Portescap N-Series ION Developer Kit Software."
2. After selecting download you will be prompted to register your DK, providing the serial # for the DK and other information about you and your motion application.
3. After selecting submit the next screen will provide a link to the software download. The software download is a zip file containing various installation programs. Select this link and downloading will begin.
4. Once the download is complete extract the zip file and execute the Pro-Motion install and extract the SDK. Here is more information on Pro-Motion and the SDK used with N-Series ION.
   - **Pro-Motion** – an application for communicating to, and exercising PMD ICs, modules, or boards.
   - **C-Motion PRP II SDK** – an SDK (Software Developer's Kit) for creating PC and downloadable user code for ION/CME N-Series Digital Drive products. Also supports creating motion applications using the .NET (C#, VB) programming languages for all products. Contains PDF versions* of all PMD documentation.

*Adobe Acrobat Reader is required for viewing these files. If the Adobe Acrobat Reader is not installed on your computer, it may be freely downloaded from http://www.adobe.com.

Here is more information on the available software resources.

1.7.1 Pro-Motion

Pro-Motion is a sophisticated, easy-to-use exerciser program which allows all ION unit parameters to be set and/or viewed, and allows all features to be exercised. Pro-Motion features include:

- AxisWizard to automate axis setup and configuration
- Pre-loaded Portescap motor part numbers
- Editable project scripts for saving and restoring your configuration for accessing motion resources and connections
- Distance, time, and electrical units conversion
- Motion oscilloscope graphically displays motion parameters in real-time
- Motor-specific parameter setup
- Auto-run scripting language

1.7.2 C-Motion

For those users who build their application control system using PMD motion control products C-Motion provides a convenient set of callable routines comprising the C language code required for controlling N-Series IONs, whether running on a separate host computer such as a PC, an embedded microcontroller, or running inside the ION on the C-Motion Engine. C-Motion includes the following features:

- Magellan axis virtualization
- Ability to communicate to multiple PMD motion cards or modules
- Ability to communicate via PC/104 bus, serial, CAN, Ethernet, SPI (Serial Peripheral Interface), or 8/16 bit parallel bus
- Provided as source code, allowing easy compilation & porting onto various run-time environments including a PC, microprocessor, embedded card, or C-Motion Engine
- Can be easily linked to any C/C++ application

Three different version of C-Motion SDKs are available; the regular C-Motion SDK is used with PMD products such as Juno ICs, Magellan ICs, and non-CME version ION 500 & 3000 Digital Drives and non-CME version Prodigy Motion Boards. The C-Motion/PRP SDK is used with /CME version products such as ION/CME 500 and Prodigy/CME motion boards. Finally the C-Motion/PRP II SDK is used with ION/CME N-Series Digital Drives.

C-Motion SDK libraries used with Magellan IC products are documented in the *C-Motion Magellan Programming Reference*. C-Motion SDK libraries used with Juno IC products are documented in the *Juno Velocity & Torque Control IC Programming Reference*. C-Motion PRP SDK libraries are documented in the *C-Motion PRP Programming Reference*. C-Motion PRP II SDK libraries are documented in the *C-Motion PRP II Programming Reference*.

1.7.3 .NET Language Support

A complete set of methods and properties is provided for developing applications in Visual Basic and C# using a dynamically loaded library (DLL) containing PMD library software. The DLL may also be used from any language capable of calling C language DLL procedures, such as Labview, but no special software support is provided.

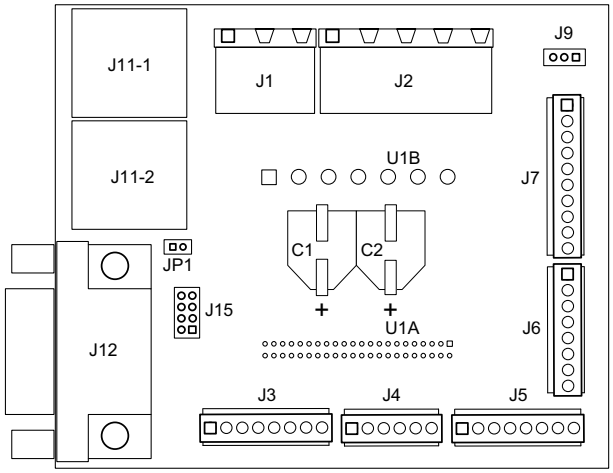
Includes the following features:

- Magellan axis virtualization
- Ability to communicate to multiple PMD motion cards or modules
- Ability to communicate via PC/104 bus, serial, CAN, Ethernet, or SPI
- Provided as a single DLL and Visual Basic .NET source code for easy porting onto various PC environments

1.8 DK Interconnect Board Overview

Figure 1-3 shows the location of various components for the Portescap N-Series ION DK board.

Figure 1-3: Serial Host Interface DK Interconnect Board



The following table identifies these components:

Label	Description
J1	HV Power Connector
J2	Motor Drive Connector
J3	Feedback Connector
J4	Hall Signals Connector
J5	Auxiliary Connector
J6	Motion Signals Connector
J7	Indexer Connector
J9	Programming Port Connector
J11	Expansion CAN Connector
J12	Host Serial Connector (Serial DK version only)
J15	Host Serial Header Connector (Serial DK version only)
JP1	Serial & Ethernet board jumper
U1A	N-Series ION Signal Connector pins
U1B	N-Series ION Power Connector pins
C1, C2	HV Capacitors

1.9 Hardware Setup

Before operating the developer kit the interconnect board must be connected to your motion hardware, a power supply, and to your PC. To assist with this the diagram below summarizes typical connections for a first-time installation, and the tables in the subsequent sections give detailed connection information for connecting Portescap motors.

Each wire connection to the interconnect board is made using a screw terminal. Use a screwdriver to loosen the screw terminal, insert the exposed stripped wire, and tighten until the wire is secure.

A minimal connection setup uses the Power and Motor Drive connections as well as the Hall sensor connections. Portescap motors also sometimes use the motor encoder connections. Other optional motion signal connections include limit switch inputs, a home signal input, and a brake input.

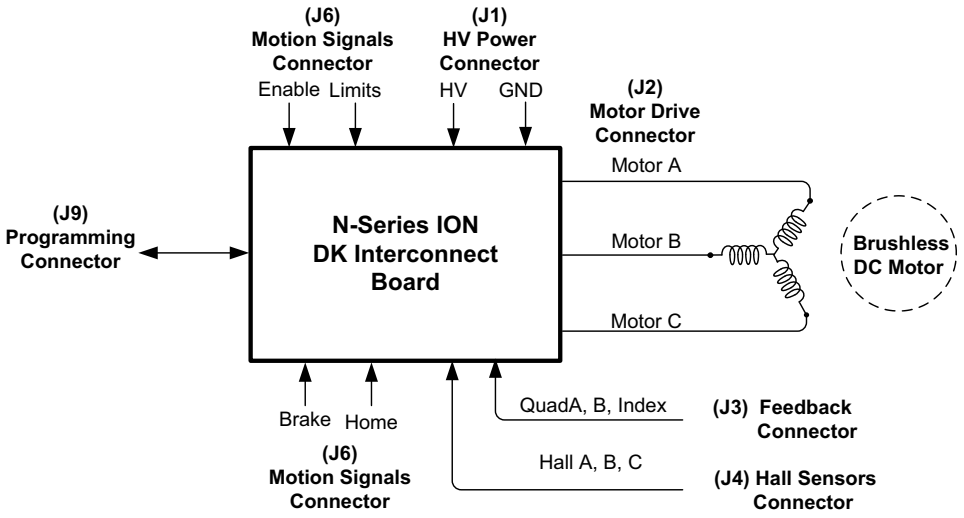


Figure 1-4: Typical DK Connections Overview

1.9.1 Motor Drive & Hall Signal Connections

All of the Portescap BLDC motors have three coil (motor drive) connections and three Hall sensor connections. In addition there is a +5V and a GND connection to provide power to the Hall sensors.

Depending on the specific Portescap motor you are using there are a few different wiring schemes that are used. The following sections provide motor drive & hall connections for each different Portescap motor type.

1.9.1.1 ECT & ECP Type Motor

Portescap motor part numbers that contain an ECT or ECP use the wiring scheme in the table below. Examples of part numbers of this type are 12ECP48 8B 21, 22ECP35 8B 184 01, 22ECT82 10B 15 01, and 30ECT64 10B 11 01.

DK Signal	DK Board Connector & Pin #	Portescap Motor Connection & Wire	Comment
Motor Coil			
MotorA	J2, 1	Phase 1, gray	Motor coil phase A
MotorB	J2, 2	Phase 2, violet	Motor coil phase B
MotorC	J2, 3	Phase 3, blue	Motor coil phase C
Hall Sensor			
HallA	J4, 1	Sensor 1, orange	HallA sensor input
HallB	J4, 2	Sensor 2, red	HallB sensor input
HallC	J4, 3	Sensor 3, brown	HallC sensor input
+5V	J3, 1	+5V, green	+5V
GND	J4, 4	GND, yellow	GND

1.9.1.2 20ECF14 Type Motor

Portescap motor part numbers that begin with 20ECF14, for example 20ECF14 8B 32, use the wiring scheme in the table below:

DK Signal	DK Board Connector & Pin #	Portescap Motor Connection & Wire	Comment
Motor Coil			
MotorA	J2, 1	Phase 1, pad* 1	Motor coil phase A
MotorB	J2, 2	Phase 2, pad 2	Motor coil phase B

DK Signal	DK Board Connector & Pin #	Portescap Motor Connection & Wire	Comment
MotorC	J2, 3	Phase 3, pad 3	Motor coil phase C
Hall Sensor			
HallA	J4, 1	Sensor 1, pad 6	HallA sensor input
HallB	J4, 2	Sensor 2, pad 7	HallB sensor input
HallC	J4, 3	Sensor 3, pad 8	HallC sensor input
+5V	J3, 1	+5V, pad 4	+5V
GND	J4, 4	GND, pad 5	GND

** pad #s refer to the connector socket location. Refer to the motor datasheet for details*

1.9.1.3 32ECF17 Type Motor

Portescap motor part numbers that begin with 32ECF17, for example 32ECF17 8B 190, use the wiring scheme in the table below:

DK Signal	DK Board Connector & Pin #	Portescap Motor Connection & Wire	Comment
Motor Coil			
MotorA	J2, 1	Phase 1, pad* 8	Motor coil phase A
MotorB	J2, 2	Phase 2, pad 7	Motor coil phase B
MotorC	J2, 3	Phase 3, pad 6	Motor coil phase C
Hall Sensor			
HallA	J4, 1	Sensor 1, pad 3	HallA sensor input
HallB	J4, 2	Sensor 2, pad 4	HallB sensor input
HallC	J4, 3	Sensor 3, pad 2	HallC sensor input
+5V	J3, 1	+5V, pad 1	+5V
GND	J4, 4	GND, pad 5	GND

** pad #s refer to the connector socket pin location. Refer to the motor datasheet for details*

1.9.1.4 45ECF and 90ECF Type Motor

Portescap motor part numbers that begin with 45ECF and 90ECF, for example 45ECF2 8B 25 and 90ECF40 8B 13, use the wiring scheme in the table below:

DK Signal	DK Board Connector & Pin #	Portescap Motor Connection & Wire	Comment
Motor Coil			
MotorA	J2, 1	Phase 1, pad* 7	Motor coil phase A
MotorB	J2, 2	Phase 2, pad 8	Motor coil phase B
MotorC	J2, 3	Phase 3, pad 4	Motor coil phase C
Hall Sensor			
HallA	J4, 1	Sensor 1, pad 1	HallA sensor input
HallB	J4, 2	Sensor 2, pad 2	HallB sensor input
HallC	J4, 3	Sensor 3, pad 5	HallC sensor input
+5V	J3, 1	+5V, pad 3	+5V
GND	J4, 4	GND, pad 6	GND

** pad #s refer to the connector socket pin location. Refer to the motor datasheet for details*

1.9.1.5 B0512A1, B0614A4, B1112A4, B2010A4, B0512A4, and B0912N1 Type Motor

Portescap motor part numbers that begin with B0512A1, B0614A4, B1112A4, B2010A4, B0512A4, or B0912N1 use the wiring scheme in the table below. Examples of part numbers of this type are B0512A1000, B0614A4001. and B0912N1010.

DK Signal	DK Board Connector & Pin #	Portescap Motor Connection & Wire	Comment
Motor Coil			
MotorA	J2, 1	Phase A, blue	Motor coil phase A
MotorB	J2, 2	Phase B, brown	Motor coil phase B
MotorC	J2, 3	Phase C, violet	Motor coil phase C
Hall Sensor			
HallA	J4, 1	Hall 1, yellow	HallA sensor input
HallB	J4, 2	Hall 2, orange	HallB sensor input
HallC	J4, 3	Hall 3, white	HallC sensor input
+5V	J3, 1	VDC, red	+5V
GND	J4, 4	Supply RTN, black	GND

1.9.2 Encoder Signal Connections

Some Portescap motors have a quadrature encoder installed on them, and if this is the case the encoder connections should be connected to the developer kit.

The encoder provided with the motor is determined by the motor diameter. 12mm diameter motors use the M-Sense12A encoder, 16 mm diameter motors use the M-Sense16 encoder, and 22 mm diameter motors use the M-Sense22 encoder . The next two sections detail the connections for the encoder part number you are using.

1.9.2.1 M-Sense12A Encoder

The table below shows the connections that should be made to connect the M-Sense12A encoder with the motor application DK. For information on connector pin identification and related info refer to the Portescap data sheet for this encoder type.

DK Signal	DK Board Connector & Pin #	Portescap Encoder Signal Name & Connection	DK Signal
QuadA+	J3, 3	A, Conn. 2 pos. # 6	A+ input
QuadA-	J3, 4	NA, Conn. 2 pos. # 5	A- input
QuadB+	J3, 5	B, Conn. 2 pos. # 4	B+ input
QuadB-	J3, 6	NB, Conn. 2 pos. # 3	B- input
Index+	J3, 7	Z, Conn. 2 pos. # 2	I+ input
Index-	J3, 8	NZ, Conn. 2, pos. # 2	I- input
+5V	J3, 1	VCC, Conn. 2 pos. # 8	+5V
GND	J3, 2	GND, Conn. 2 pos. # 6	GND

1.9.2.2 M-Sense16 & M-Sense22 Encoders

The table below shows the connections that should be made to connect M-Sense16 and M-Sense22 encoders with the motor application DK. For information on connector pin identification and related info refer to the Portescap data sheet for these encoder types.

DK Signal	DK Board Connector & Pin #	Portescap Encoder Signal Name & Connection	Comment
QuadA+	J3, 3	Channel A, pin #6	A+ input
QuadA-	J3, 4	Channel A/, pin #5	A- input
QuadB+	J3, 5	Channel B, pin #8	B+ input
QuadB-	J3, 6	Channel B/, pin #7	B- input
Index+	J3, 7	Channel Z, pin #10	I+ input
Index-	J3, 8	Channel Z/, pin #9	I- input
+5V	J3, 1	VCC, pin #2	+5V
GND	J3, 2	GND, pin #3	GND

1.9.3 Power Connections

The Portescap developer kit requires DC voltage power within a range of 12V to 56V. A laboratory type power supply with current limiting is recommended.

In addition to power inputs N-Series ION supports an optional shunt function which can be used to control overvoltage conditions. For more information refer to Section 3.2, "Shunt Operation."

The table below shows the power-related connections to the developer kit:

DK Signal	DK Board Connector & Pin #	Comment
HV Aux	J1, 1	Connect to positive DC voltage output
HV	J1, 2	Connect to positive DC voltage output
HV GND	J1, 3	Connect to negative DC voltage output
Shunt	J2, 4	Optional shunt output

1.9.4 Other Motion & Control Signals

Depending on the application other motion signals may be connected including a home signal input, positive and negative limit switch inputs, a brake signal input, and an enable input. Use of all of these signals is optional except the Enable signal which must be tied low for the unit to operate. Refer to Section 1.9.6, "Enabling the N-Series ION," for details.

The table below shows DK connection information for these signals:

DK Signal	DK Board Connector & Pin #	Comment
Home	J6, 3	Home signal input
PosLim	J6, 1	Positive limit switch input
NegLim	J6, 2	Negative limit switch input
Brake	J6, 7	Brake signal input
Enable	J6, 4	Enable signal input
GND	J6, 8	Ground

For detailed electrical interfacing information on these signals refer to the *ION/CME N-Series ION Digital Drive User Manual*.

1.9.5 Making Connections to the PC

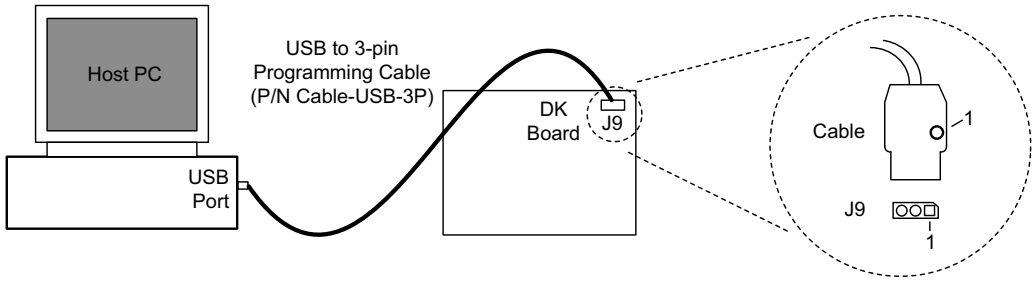


Figure 1-5: PC to N-Series ION DK Board Connection

The next step in the hardware setup process is to connect the PC to the developer kit.

This is accomplished with the 3-pin serial programming interface (J9) and the included 3-pin programming cable. This serial cable (PMD p/n Cable-USB-3P) should be connected to the board's J9 Programming Connector, while the opposite end of the cable should be connected to one of your computer's USB ports. Take special care when connecting to the 3-pin connector on the board that pin #1 on the cable (marked with a dot on the cable) aligns with pin #1 on the connector (the pin closest to the right side of the board).

1.9.6 Enabling the N-Series ION

The next step is to establish an active ***Enable*** signal to the N-Series ION drive. This is usually accomplished via a short jumper wire connecting the ***Enable*** signal on the Motion Signals Connector (J6, pin 4) to ground connect (J6, pin 8).

1.9.7 Applying Power

Once you have made your motion hardware, communication, and power connections, hardware installation is complete and the developer kit is ready for operation. The allowed HV power input range is 12V to 56V. When power is applied, the N-Series ION's green power LED should be in a steady on condition and the red LED should be off.

If the LED output is displaying a different status some of these may be non-serious correctable conditions. The most common are a solid green LED in combination with a fast blinking red LED. This may indicate that the HVAux is not connected or that there is an over or under voltage supply from the DC power supply. Another common condition is a slow blinking green LED with the red LED off indicating that the ***Enable*** signal is not active. An LED display of solid red with no green indicates a serious problem. If this occurs you should immediately turn off power to the unit and re-check power and motor coil connections.

For more information on the N-Series ION's LEDs and additional display output combinations see Section 4.13, "LEDs."

After power up no output drive to the motor will be applied. Therefore the motor should remain stationary. If the motor moves or jumps, power down and check the motor and encoder connections.

1.10 Complete Connector Reference

The following sections provide complete and detailed connector information for the DK Interconnect board. This may be useful as you go beyond exercising the motor and begin to develop your complete motion application.

1.10.1 Feedback Connector (J3)

The following table details the Feedback Connector (J3), which is an 8-pin terminal screw style connector.

Pin #	Signal Name	Description
1	+5V	+5V power output which may be used to power the motor's encoder circuitry
2	GND	This is the preferred ground connection for the quadrature and Index signal inputs
3	QuadA1+/Cos+*	Differential A+ or Cos+ encoder input for primary axis.
4	QuadA1-/Cos-*	Differential A- or Cos- encoder input for primary axis.
5	QuadB1+/Sin+*	Differential B+ or Sin+ encoder input for primary axis.
6	QuadB1-/Sin-*	Differential B- or Sin- encoder input for primary axis.
7	Index1+/BiSSClock+	Differential Index+ or BiSSClock+ encoder connection for primary axis.
8	Index1-/BiSSClock-	Differential Index- or BiSSClock- encoder connection for primary axis.

** Quadrature encoder type is the default setting. For information on how to change encoder types refer to the ION/CME N-Series Digital Drive User Manual.*

1.10.1.1 Single-ended Encoder Connections

Encoder inputs may be connected differentially, with two wires per signal (as shown in the table above), or with just one wire per signal. If single-ended encoders are used, connect encoder signals to the positive encoder input only. The negative input may remain unconnected.

1.10.2 Motion Signals Connector (J6)

The following table details the Motion Signals Connector (J6). This connector is an 8-pin terminal screw style connector.

Pin #	Signal Name	Description
1	PosLim	Positive position limit input (optional)
2	Neglim	Negative position limit input (optional)
3	Home	Home signal input (optional)
4	Enable	Enable input signal
5	FaultOut	FaultOut signal output
6	Reset	Reset signal input (optional)
7	Brake	Brake signal input (optional)
8	GND	Digital ground

1.10.3 Power Connector (J1)

The following table details the Power Connector (J1).

The Power Connector is a Würth Elektronik 3 Position Terminal Block Header P/N 691313710003.

Pin #	Signal Name	Description
1	HV Aux	Positive motor voltage power used to drive N-Series ION's internal logic
2	HV	Positive motor voltage power used to drive the motor
3	GND	Motor voltage power ground

The HV Aux and HV pins are normally tied together but may be kept separate.

1.10.4 Motor Drive Connector (J2)

The following table details the motor drive connector (J2). There are four motor drive connections and a shield connection. Not every motor type uses all four drive connections.

The J2 Motor Drive Connector is a Würth Elektronik 5 Position Terminal Block Header P/N 691313710005.

Pin #	Signal Name	Description
1	Motor A	A motor drive lead.
2	Motor B	B motor drive lead.
3	Motor C	C motor drive lead.
4	Motor D/Shunt	D motor drive lead.
5	Case/shield	Connection to motor case/shield. A shield connection is recommended for most motor setups. This pin is connected to the N-Series ION's Power Connector GND signal.

1.10.5 Hall Signals Connector (J4)

The following table details the Hall Signals Connector (J4). Hall signals are used with Brushless DC motors. This connector is a 6-pin terminal screw style connector.

Pin #	Signal Name	Description
1	HallA	Hall Sensor A Input
2	HallB	Hall Sensor B Input
3	HallC	Hall Sensor C Input
4	GND	Digital ground
5	AnalogIn+	Positive differential signal of the general purpose analog input
6	AnalogIn-	Negative differential signal of the general purpose analog input

1.10.6 Auxiliary Connector (J5)

The following table details the Auxiliary Connector (J5). This connector is an 8-pin terminal screw style connector.

Pin #	Signal Name	Description
1	+5V	+5V power output which may be used to power the encoder
2	GND	Ground
3	QuadA2+	Auxiliary encoder differential A+ quadrature input
4	QuadA2-	Auxiliary encoder differential A- quadrature input
5	QuadB2+	Auxiliary encoder differential B+ quadrature input
6	QuadB2-	Auxiliary encoder differential B- quadrature input
7	Index2+/BiSSData+*	Differential Index+ or BiSSData+ encoder connection.
8	Index2-/BiSSData-*	Differential Index- or BiSSData- encoder connection.

** Quadrature encoder type is the default setting for both the primary and auxiliary axis. For information on how to change encoder types refer to the ION/CME N-Series Digital Drive User Manual.*

1.10.7 Indexer Connector (J7)

The following table details the Indexer Connector (J7). This connector is an 10-pin terminal screw style connector.

Pin #	Signal Name	Description
1	GND	Ground
2	DigitalIO1	Digital I/O bit 1
3	DigitalIO2	Digital I/O bit 2
4	DigitalIO3/AxisIn*	Digital I/O bit 3 or AxisIn signal depending on pin MUX setting. AxisIn is the default functionality.
5	DigitalIO4/SynchIn*	Digital I/O bit 4 or SynchIn signal depending on pin MUX setting. SynchIn is the default functionality.
6	DigitalIO5/AxisOut*	Digital I/O bit 5 or AxisOut signal depending on pin MUX setting. AxisOut is the default functionality.
7	DigitalIO6/HostInterrupt*	Digital I/O bit 6 or HostInterrupt signal depending on pin MUX setting. HostInterrupt is the default functionality.
8	DigitalIO7/SynchOut*	Digital I/O bit 7 or SynchOut signal depending on pin MUX setting. SynchOut is the default functionality.
9	DigitalIO8	Digital I/O bit 1
10	GND	Ground

** To access the desired function the pin MUX setting may need to be changed. Refer to the ION/CME N-Series Digital Drive User Manual for more information.*

1.10.8 Expansion CAN Connector (J11-1, J11-2)

The following table details the Expansion CAN Connectors (J11-1 and J11-2), which consist of two 8-pin RJ45 connectors wired such that each J11-1 pin connects to the matching J11-2 connector pin (daisy chain configuration).

J11-1 Pin #	J11-2 Pin #	Signal Name	Description
1	1	ExpCAN+	Expansion CAN+
2	2	ExpCAN-	Expansion CAN-
3	3	GND	Ground
4	4	N.C.	
5	5	N.C.	
6	6	N.C.	
7	7	GND	Ground
8	8	N.C.	

1.10.9 Host Serial Connector (J12)

The following table details the Host Serial Connector (J12), which is provided with serial host interface N-Series ION DKs. This connector supports two RS232 ports or a single RS485 port operating in either RS485 full duplex, or RS485 half duplex mode. This connector is a female DB-9.

Pin #	Signal Name	RS232	RS485 Full Duplex	RS485 Half Duplex
1	No connect	no connect	no connect	no connect
2	Srl1Xmt	Serial 1 transmit output	no connect	no connect
3	Srl1Rcv	Serial 1 receive input	no connect	no connect
4	No connect	no connect	no connect	no connect
5	GND	Ground	Ground	Ground
6	$RS485Rcv^{+}$	no connect	Positive (non-inverting) receive input	no connect

7	$Srl2Rcv/RS485Rcv^-$	Serial 2 receive input	Negative (inverting) receive input	no connect
8	$Srl2Xmt/RS485Xmt^-$	Serial 2 transmit output	Negative (inverting) transmit output	Negative transmit/receive
9	$RS485Xmt^+$	no connect	Positive (non-inverting) transmit output	Positive transmit/receive

1.10.10 Host Serial Header Connector (J15)

The following table details the Host Serial Header Connector (J15), which is provided with serial host interface N-Series ION DKs. This connector provides a subset of the signals provided with the Host Serial Connector (J12). This connector is an 8-pin 100 mil header.

Pin #	Signal Name	Description
1	Srl2Xmt	Serial2 Transmit
2	Srl2Rcv	Serial2 Receive
3	Srl1Xmt	Serial1 Transmit
4	Srl1Rcv	Serial1 Receive
5	DigitalIO6	DigitalIO6 signal
6	RS485Sel	RS232/RS485 select pin
7	GND	Ground
8	GND	Ground

1.11 First-Time System Setup & Verification

The first time system verification procedure summarized below has three overall goals. The first is to connect the N-Series ION with your PC so that they are communicating properly. The second is to set various control and operating parameters for your specific motor setup. The third is to initialize the axis and bring it under stable control capable of making controlled moves. While there are many additional capabilities that Pro-Motion and N-Series IONs provide, the first time verification will create a foundation for further successful exploration and development.

Here is a summary of the steps that will be used during first time system verification. Each of these steps will be described further in the sections below.

1. Initiate Pro-Motion and establish communication between the PC and the N-Series ION using the serial 3-pin programming cable.
2. Run Pro-Motion's Axis Wizard to initialize safety parameters, your specific application connections, and more.
3. Save your application setup parameters into a setup script file.
4. Restore your setup settings using this script file.
5. Execute motion commands demonstrating that the axis is operating correctly and under stable control.

During this first time system setup you may find it useful to refer to the *Magellan Motion Control IC User Guide*. and the *ION/CME N-Series ION Digital Drive User Manual.*

1.11.1 Establishing Communications

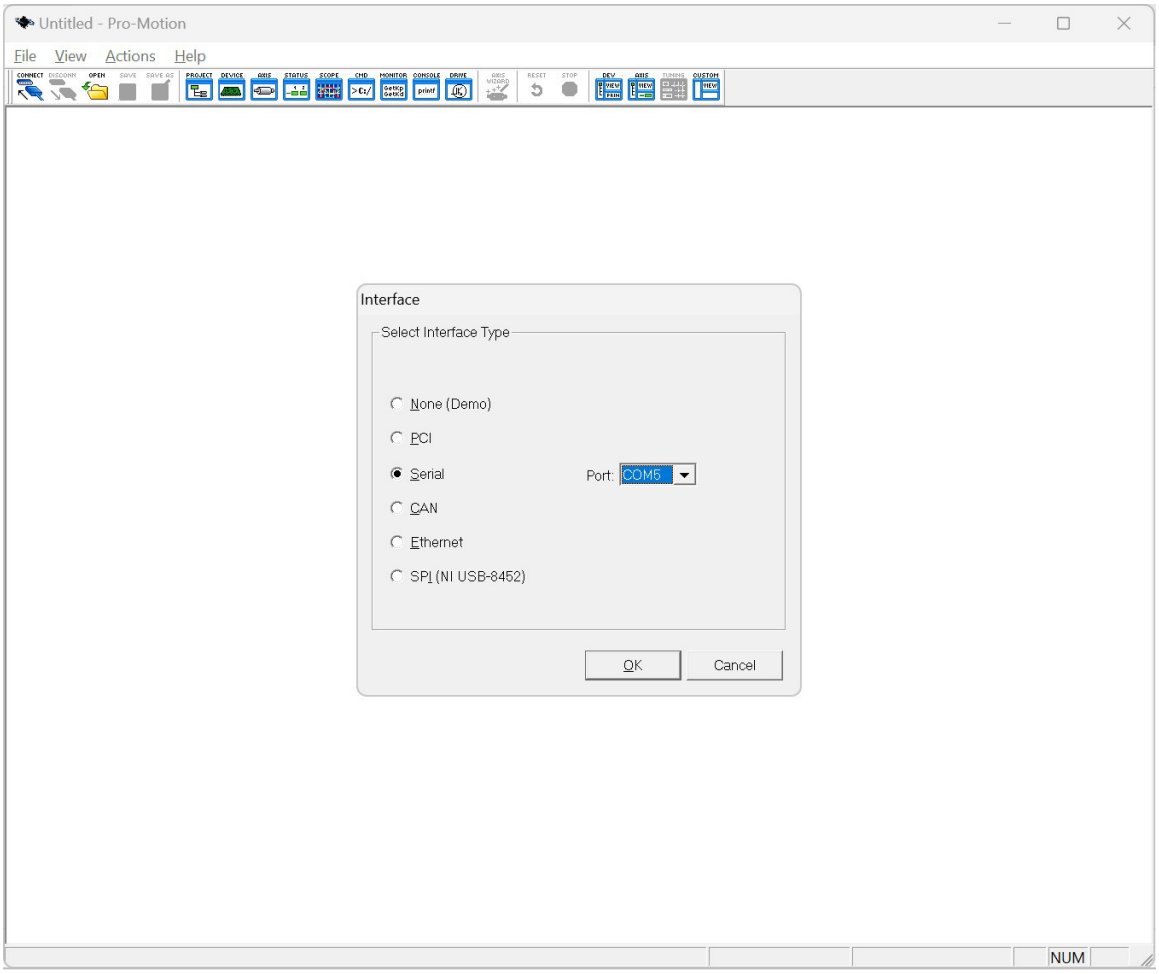
To establish serial communications:

1. Make sure the N-Series ION DK interconnect board is powered and connected to the PC via the 3-pin programming cable as shown in Figure 1-5.

2 Launch the Pro-Motion application.

When Pro-Motion is launched you will be prompted with an Interface selection window. A typical screen view when first launching Pro-Motion appears below.

The purpose of the Interface dialog box is to indicate to Pro-Motion how your N-Series ION is connected to the PC. It provides various selectable communication options such as serial, CAN, Ethernet.



3 Click Serial and view the available COM ports listed in the Port field. If you know which COM port your 3-pin programming cable is connected to, select it.

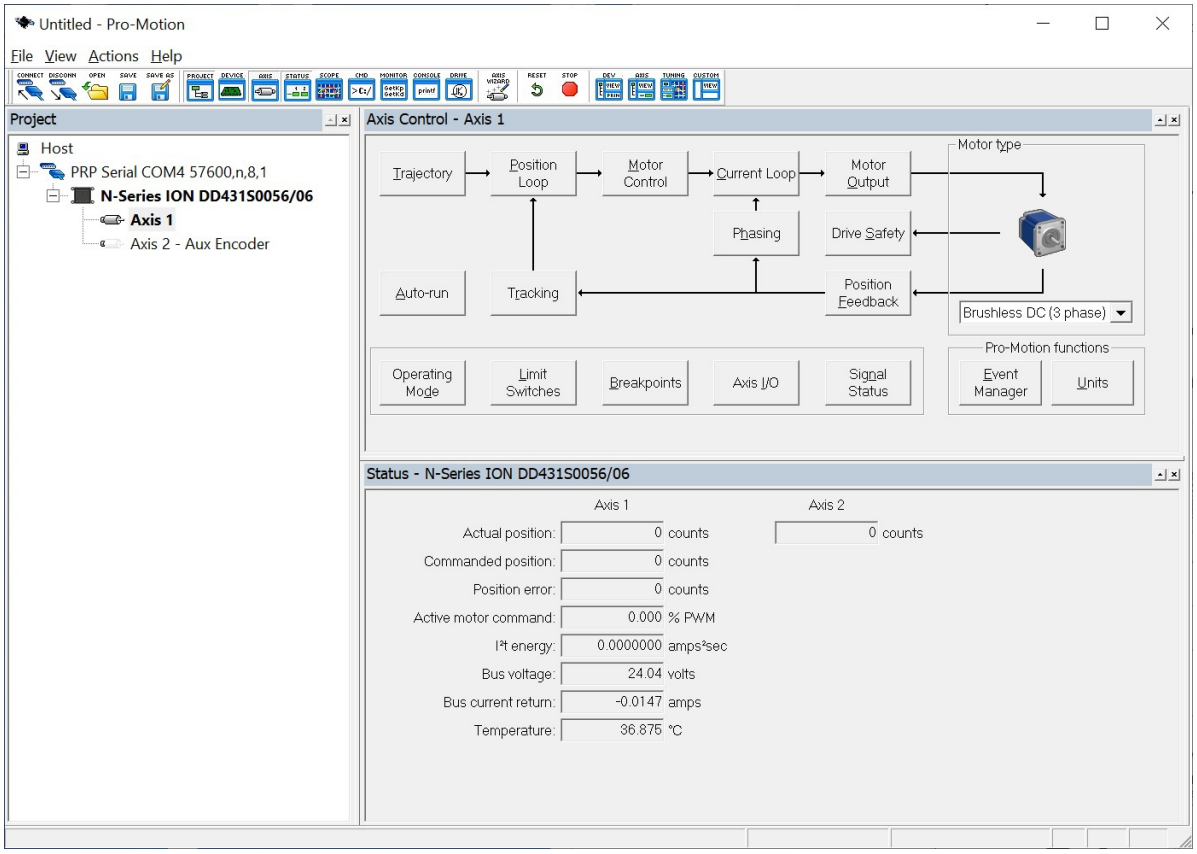
If you are not sure which of the listed COM ports is the correct one, you may use the following procedure:

- First, unplug the 3-pin cable from your USB port and exit the Interface dialog box. Now re-enter the Interface dialog box, select Serial, and view the COM port list. Record the list of COM ports.
- Next plug the 3-pin cable back into the USB port and once again exit and re-enter the Interface dialog box. Select Serial and now when you view the COM port list you should see a new COM port listed. This is the COM port that is connected via the 3-pin cable provided with the DK.
- Select this COM port and hit the OK button.

The Serial Port dialog box displays with default communication values of 57,600 baud, no parity, 1 stop bit, and point to point protocol.

4 Click OK without changing any of these settings.

If serial communication is correctly established, after a brief pause a set of object graphics loads into the Project window to the left, as shown in the following figure.



If serial communications are not correctly established, a message appears indicating that an error has occurred. If this is the case, recheck your connections and repeat from step 1.

1.11.2 Running the Axis Wizard

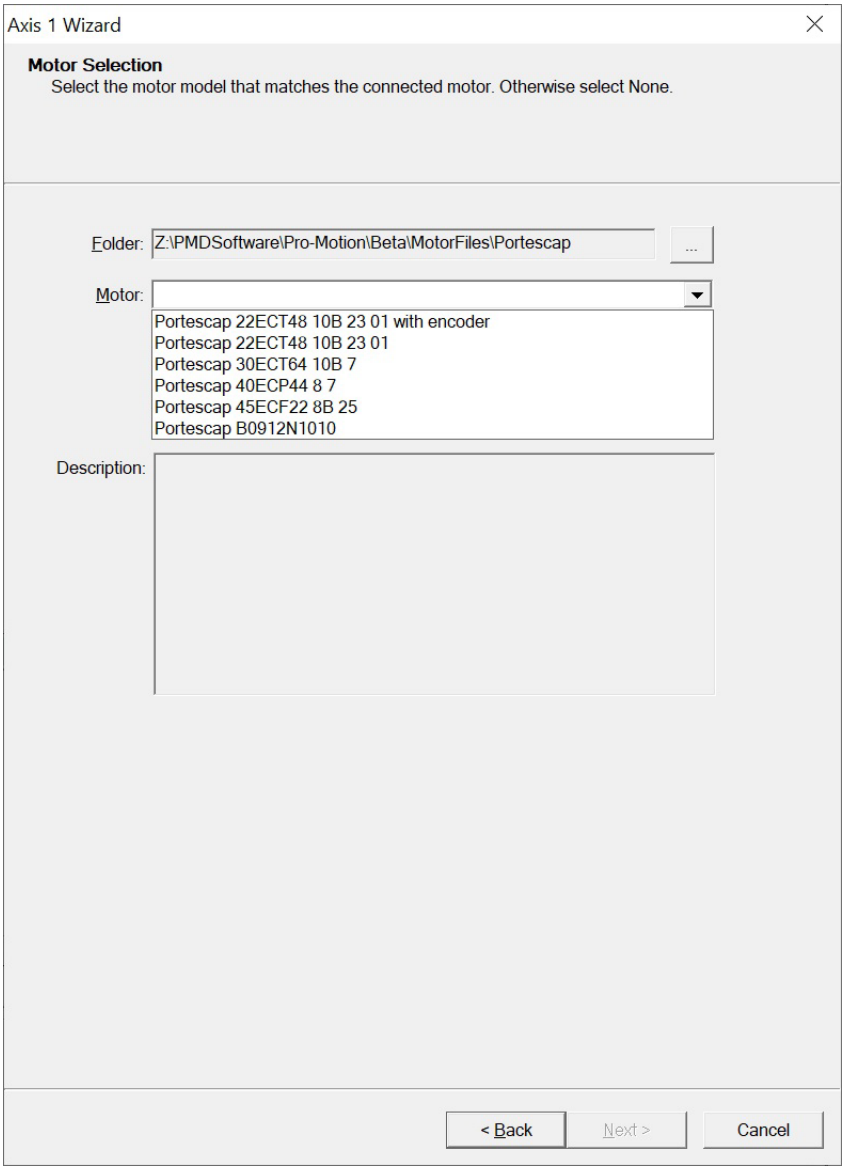
The next step is to initialize the axis, thereby verifying correct encoder feedback connections (if an encoder is used), and other functions. All of this can be conveniently accomplished using Pro-Motion's Axis Wizard function.

To operate the Axis Wizard:

1 Click the Axis Wizard toolbar button which is located right of center in the row of icons toward the top of the window.

The Axis Wizard initialization window appears.

2 Click Next. The screen that will appear is shown below.

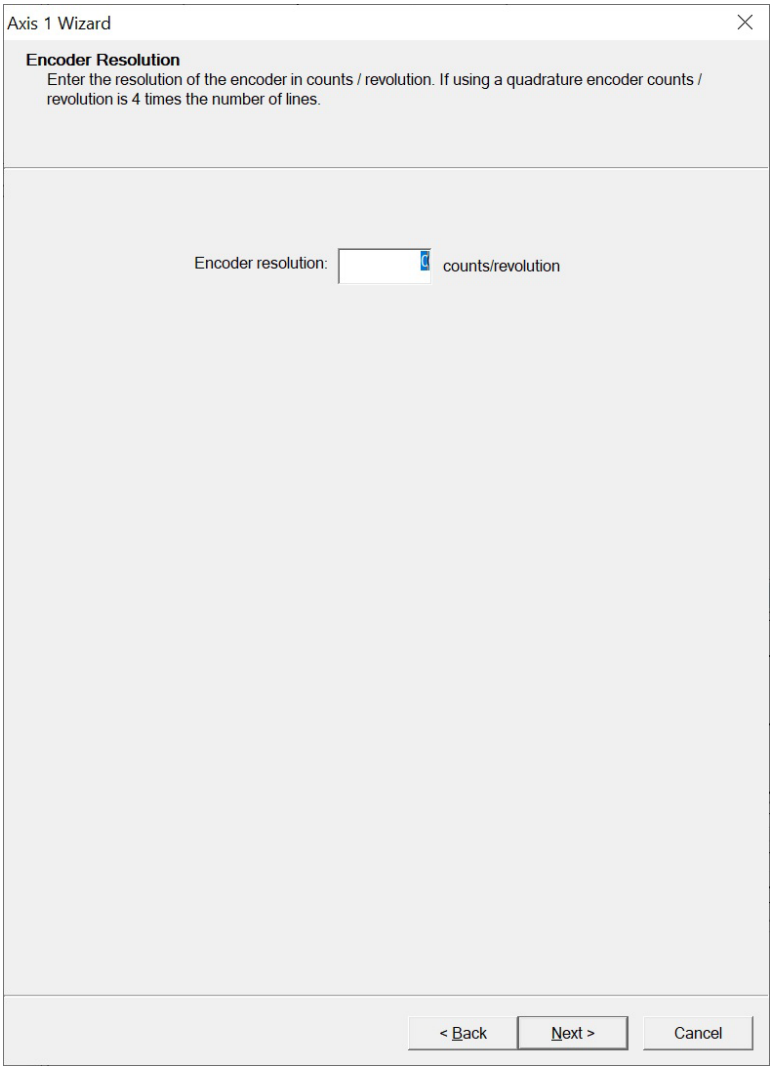


3 The Folder listed shows a directory containing motor configuration script files, and the Motor field allows you to browse the available motor types. Some motors have two different scripts, one if an encoder is attached, and one if there is no encoder. Select the script that matches the motor configuration you are using. Upon selecting a motor script additional information will be displayed about the contents of this specific motor setup script file in the Description field.

4 When you are satisfied with the motor selection click the Next button at the bottom of the dialog box. This will install the pre-programmed motor parameters for the selected motor.

5 Depending on whether the motor script file you selected was for a Halls only motor or a motor with an encoder one of two screens will come up next. For motors with an encoder step 6 below begins the description for the next screen in the Axis Wizard process. For Halls only motors skip to Section 1.11.3, "Axis Wizard Next Steps," to view the next screen description.

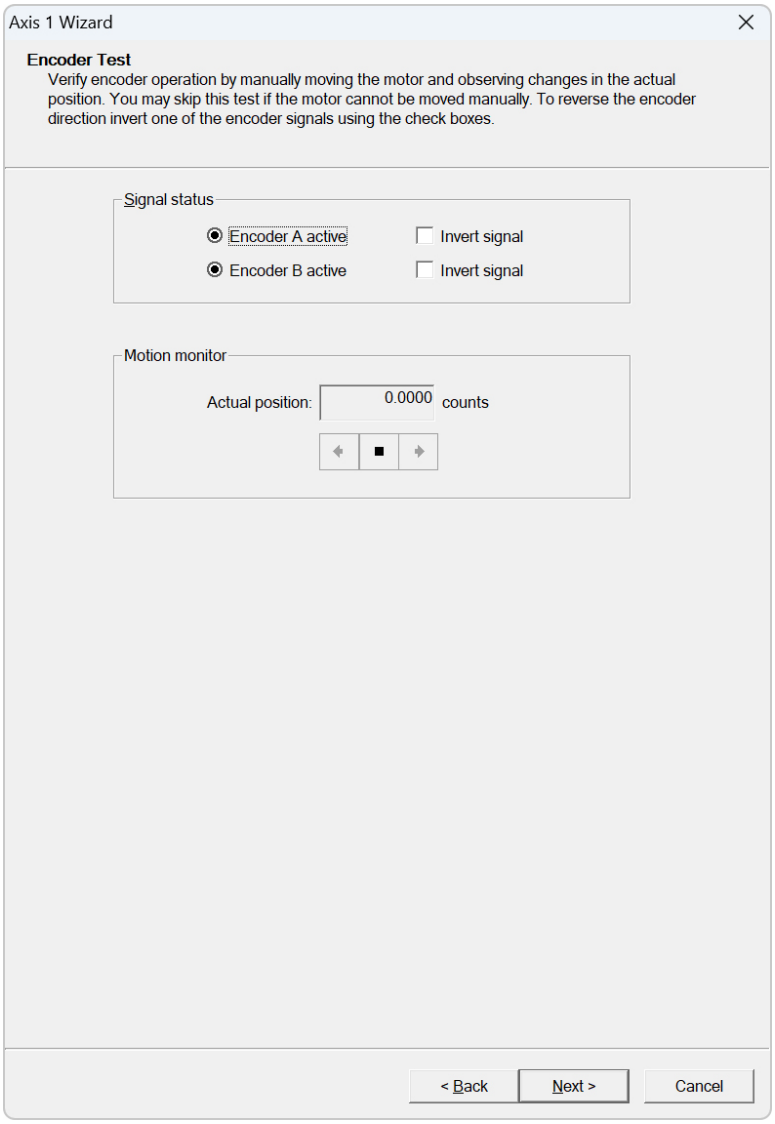
6 For motors with encoders the next screen that appears lets you specify the encoder resolution.



7 Enter the number of quadrature encoder counts your encoder has per mechanical rotation. The number of counts is four times the number of lines. For example if the motor is indicated as having 512 lines, the resolution in counts would be 2,048 counts per revolution.

8 When you have entered the encoder resolution click Next to see the next screen

The next screen that appears lets you test that the motor encoder is connected correctly.

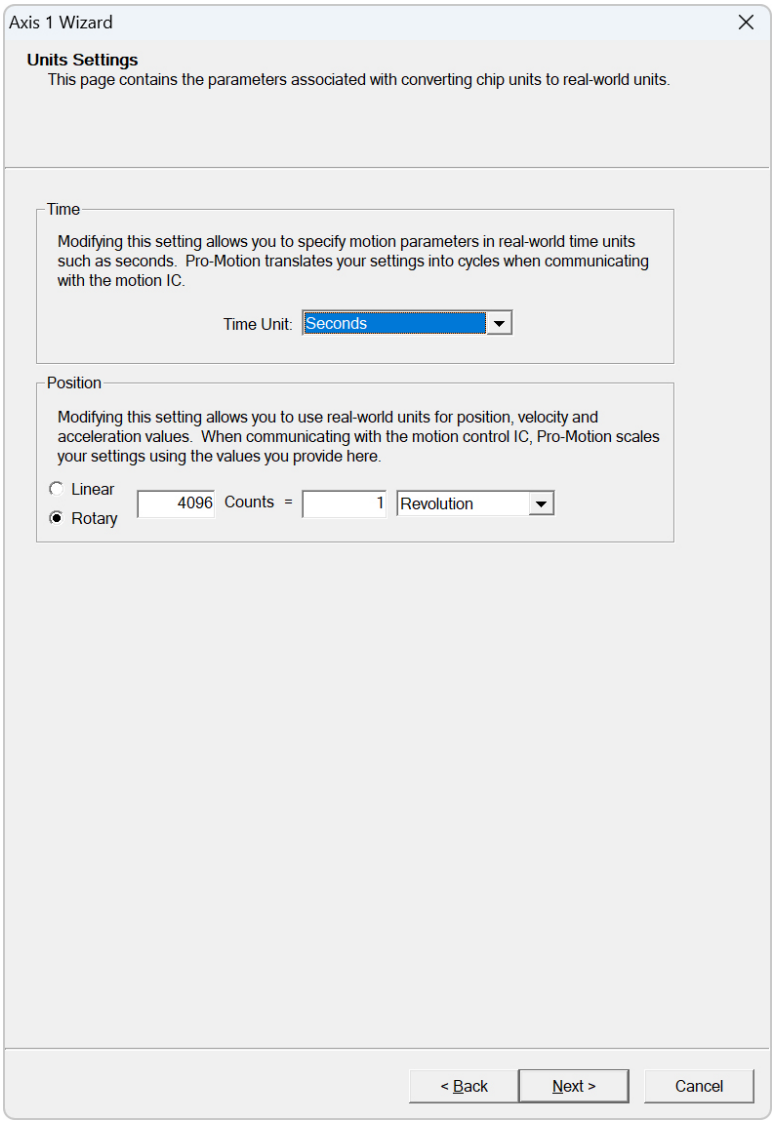


9 Manually rotate the motor, checking that the encoder position increases or decreases in correspondence with the direction you manually rotate the motor. If the encoder position changes in the opposite way you would like you can invert either the A or B encoder signal to reverse.

If the reported encoder position does not change at all, power down the developer kit and recheck connections. When ready hit Next to view the next screen.

1.11.3 Axis Wizard Next Steps

The next screen that appears allows you to select units. Native position units are counts, which are quadrature encoder counts for motors with an encoder, and Hall states for motors without an encoder.



1 It is possible to operate Pro-Motion in native N-Series ION units but many users will prefer to use real world units such as seconds, revolutions, or centimeters. The time unit options are cycles, seconds, milliseconds, and minutes. The position unit options are counts, millimeters, inch, foot, and meter if linear motion is selected, and counts, degrees, radians, and revolutions if rotary motion is selected.

Upon entering this screen the position units box should display the correct rotary settings to convert counts to motor revolutions. If no encoder is present counts are defined as Hall state transitions, so this will be a multiple of six depending on the number of pole pairs the motor has. For example if the motor has two pole pairs the number of counts per revolution will be 12.

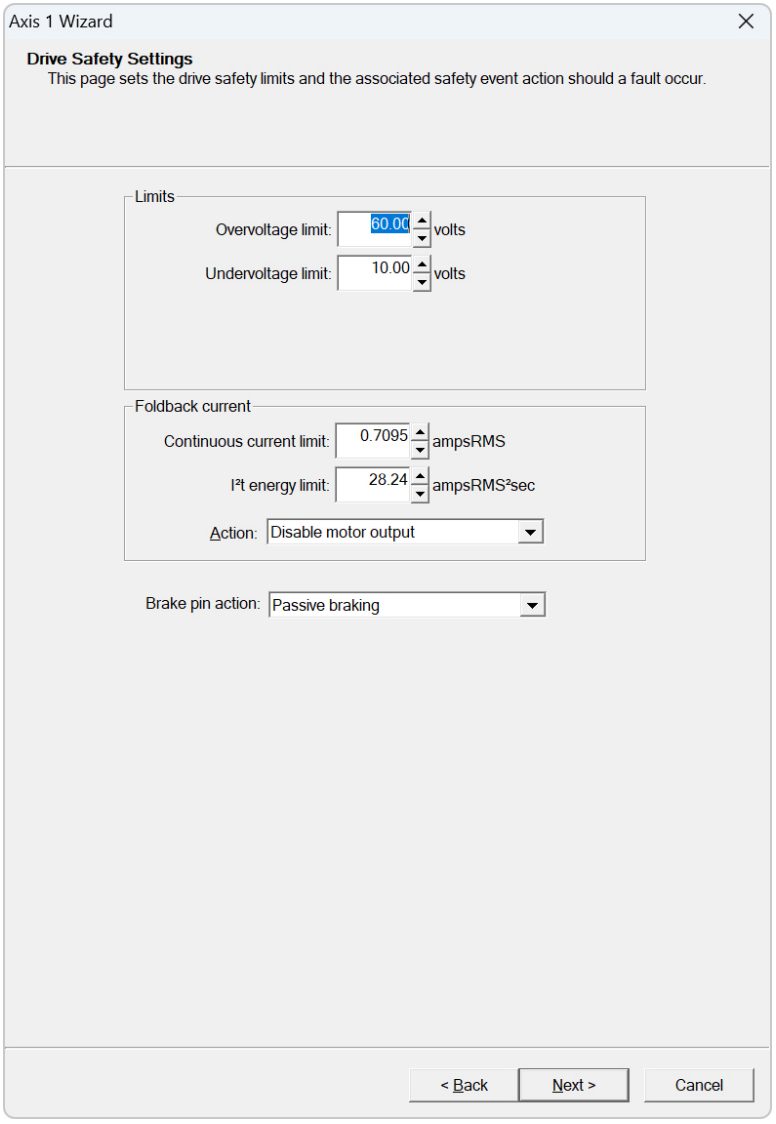
For motors with a gearhead multiply the counts per rotation by the gearhead ratio to set the physical units ratio. For linear motion you will need to enter the # of distance units corresponding to a specific number of counts.

For this ratio parameter setting the number of counts must be an integer but the equivalent distance unit can be a floating point number. For example if a linear stage traverses 10,000 counts for every 1.5 millimeters of travel the two numbers entered could be 10000 and 1.5, or any equivalent ratiometric values.

When the settings for this screen have been set up as desired select Next at the bottom of the dialog box.

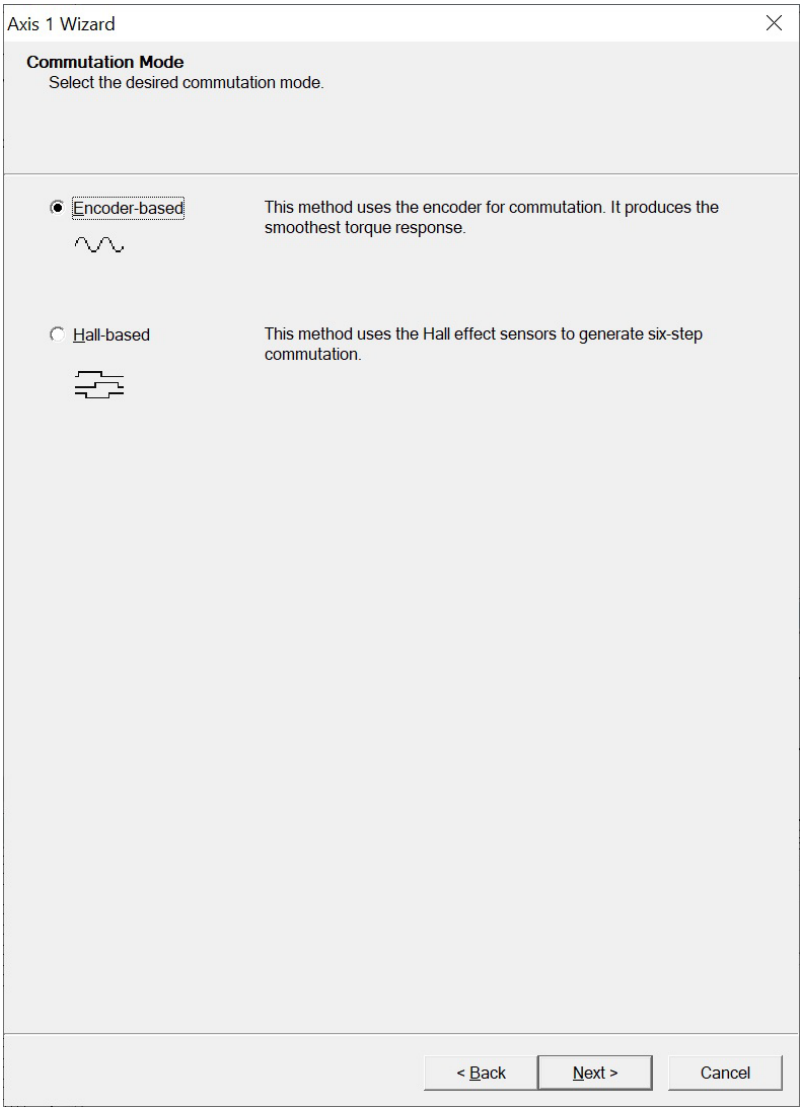
2 The next screen, shown below, allows you to set various safety parameters. Over and undervoltage settings should be based on the DC Bus supply voltage. To avoid overstressing the power supply a general recommendation is to set each of these safety parameters 20% above or below the supply voltage being used. For example if your DC supply voltage is 24V, you might set the overvoltage limit to 30V and the undervoltage limit to 18V.

Foldback current settings are motor specific and can not be increased but can be decreased if desired.



When ready click Next to view the next screen. For motors with encoders this is described beginning at step 3 below. For motors with Halls only skip to Section 1.11.4, "Axis Wizard Next Steps," for descriptions of the next screen.

3 For motors with encoders the next screen that appears lets you specify the commutation mode. This screen is shown below.



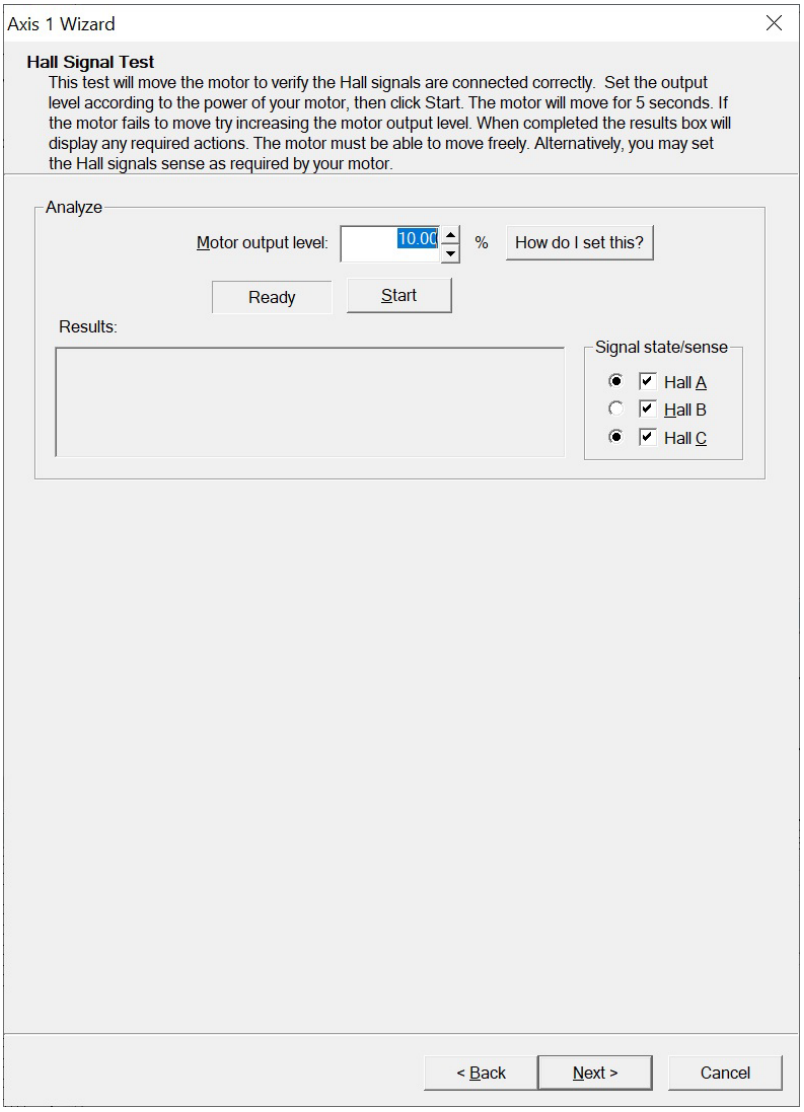
4 Encoder-based commutation generally provides smoother rotation because the commutation angle is determined by the encoder, which has a high resolution compared to Hall sensors. The resulting control waveforms are sinusodial and therefore have no discontinuities.

Hall-based commutation uses the hall sensors to determine the commutation angle. Within a 360 degree electrical cycle the Halls define 6 separate states, each 60 degrees wide. So with this scheme the commutation angle changes only every 60 degrees. Nevertheless some motors are designed specifically to operate with Hall based (also sometimes called trapezoidal) waveforms rather than sinusoidal waveforms.

5 When ready select the desired commutation mode and hit Next.

1.11.4 Axis Wizard Next Steps

The next screen that appears, shown below, allows you to test the wiring of the Hall sensors.



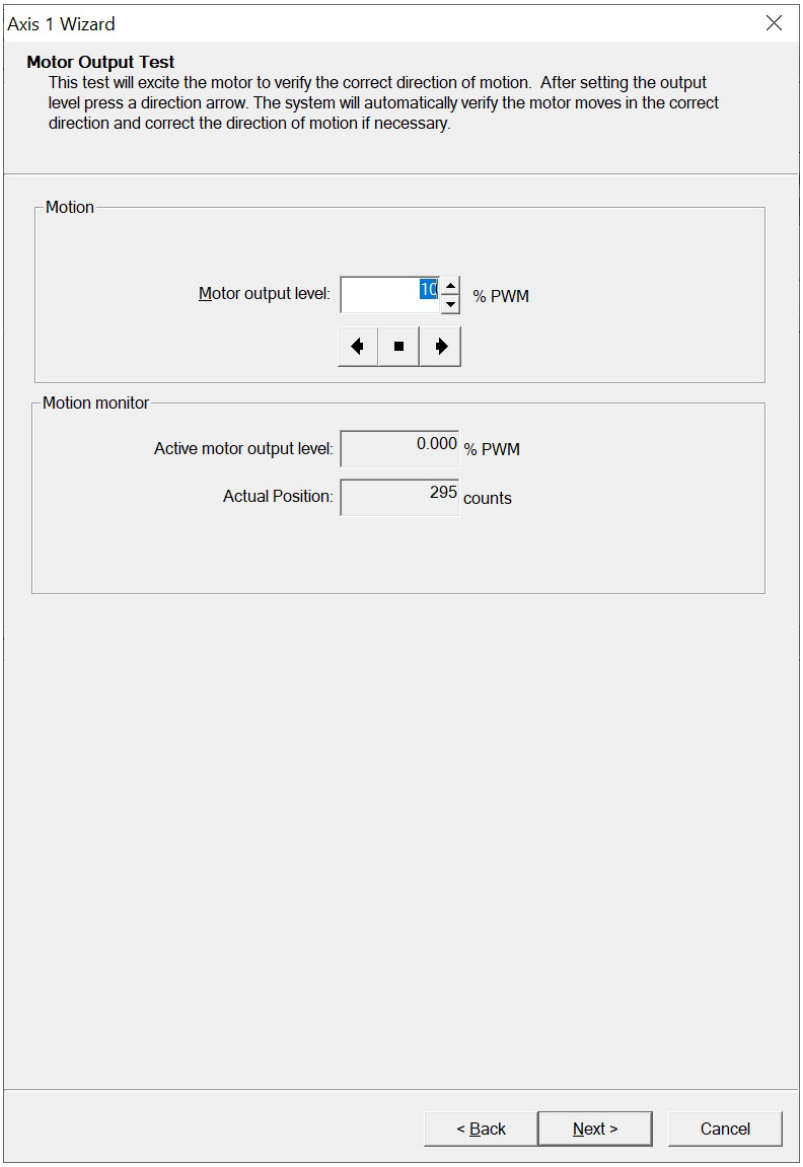
1 Start by selecting a motor command level. The specified percentage is the percentage of the DC supply voltage that will be briefly applied to the motor. Typical values are 5 to 15% depending on the supply voltage. With higher voltages such as 36V or 48V a value of 5% may be a good starting value, and with lower voltages such as 12V or 24V 10% or 15% may be a good value.

2 When ready select start. You will see the motor rotate in small increments forwards or backwards and after a few seconds the results will be indicated. The results may indicate instructions to rewire the motor Hall signals, may indicate that the Hall signals should be inverted, or may indicate a problem such as no motion being observed.

   If the motor halls have been wired correctly any required inversion will be handled automatically. If the motor does not move go back to step 1 increasing the PWM level. If the PWM level is 15% or higher and there is still no motion recheck the motor coil connections.

3 When ready hit Next to advance to the next screen. For motors that have encoders the next screen is described below beginning at step 8. For motors with Halls only skip to Section 1.10.5, "Hall Signals Connector (J4)," for the description of the next screen.

4 The next screen in the Axis Wizard sequence lets you test correct rotation of the motor. This screen is shown below:



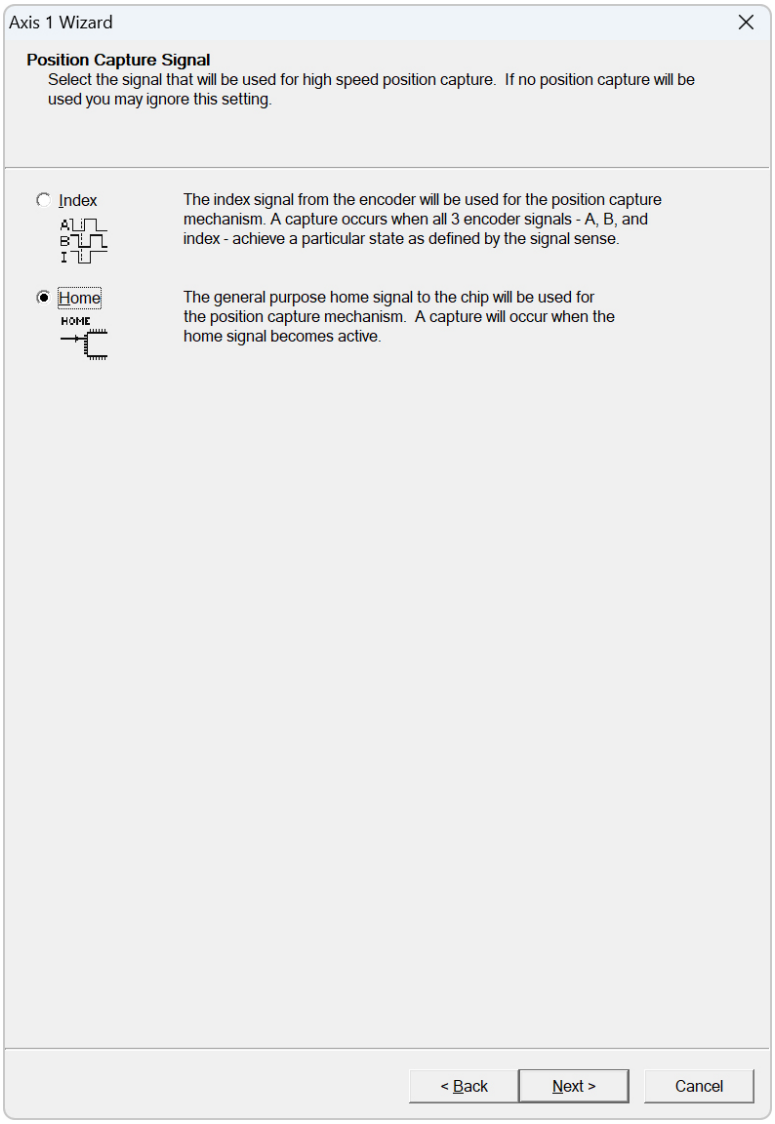
5 Start by selecting a motor command level. The specified percentage is the percentage of the DC supply voltage that will applied to the motor. The default value will be same value as used for the Hall signal test and most users will continue to use this same motor command level.

6 Press the right arrow or the left arrow key to start the motor moving. Confirm that the motor rotates, and that it rotates in the correct direction—meaning that the right arrow results in an increase in Actual Position (position as measured by the motor encoder or Hall sensor counts), and the left arrow results in a decrease in Actual Position.

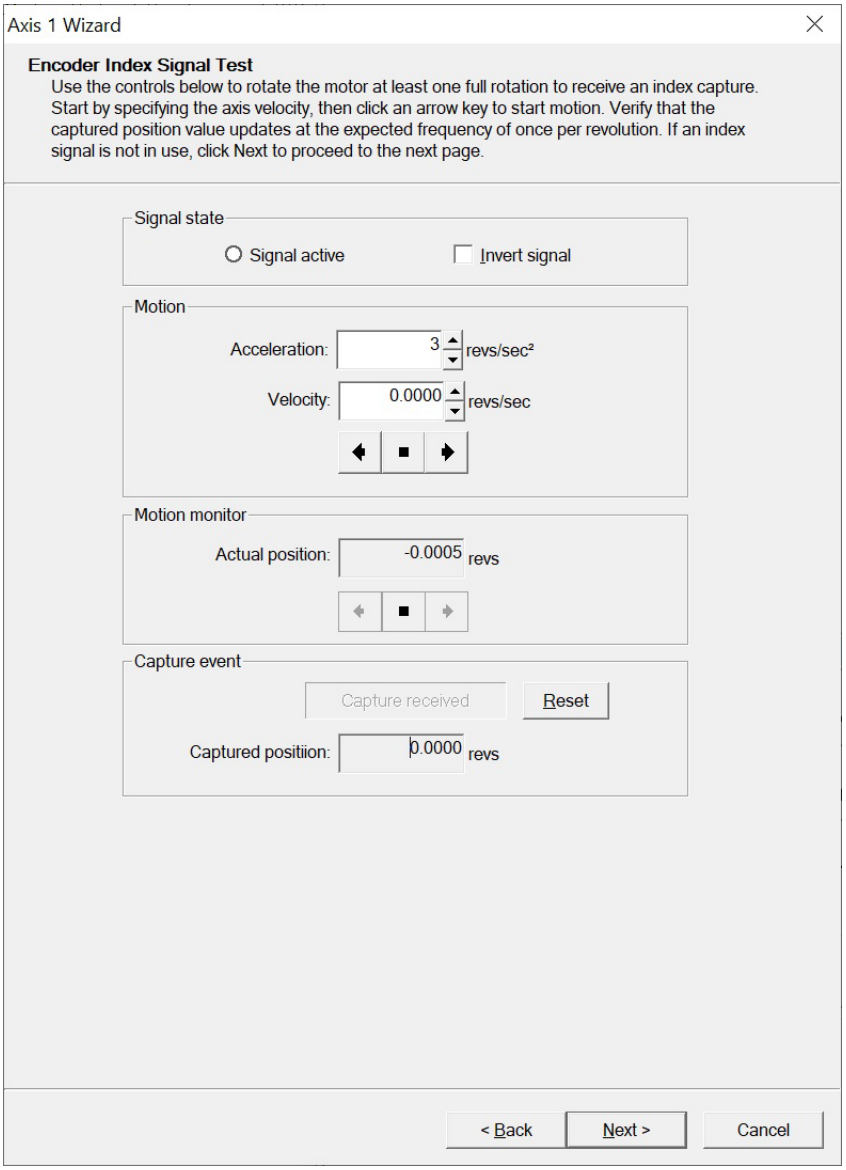
If the motor moves in the wrong direction the system should automatically correct for this. If the motor does not move, or moves in an unexpected way check Hall signal and coil signal connections and re-run the Axis Wizard from the beginning.

7 When ready select Next to advance to the next screen. For motors that have encoders the next screen is described below beginning at step 8. For motors with Halls only skip to Section 1.11.5, "Axis Wizard Next Steps," for the description of the next screen.

For motors with encoders the next screen that appears, shown below, allows you to select the position capture source.



8 This screen lets you select either the Index signal or a Home signal for use as the high speed position capture. If neither of these signals are used in your system you may skip this screen and the Capture Input Signal screen by hitting Next twice. Note that if you select Home you must wire a home sensor according to the connections detailed in Section 1.10.2, "Motion Signals Connector (J6)." Once ready click Next and the following screen will display:

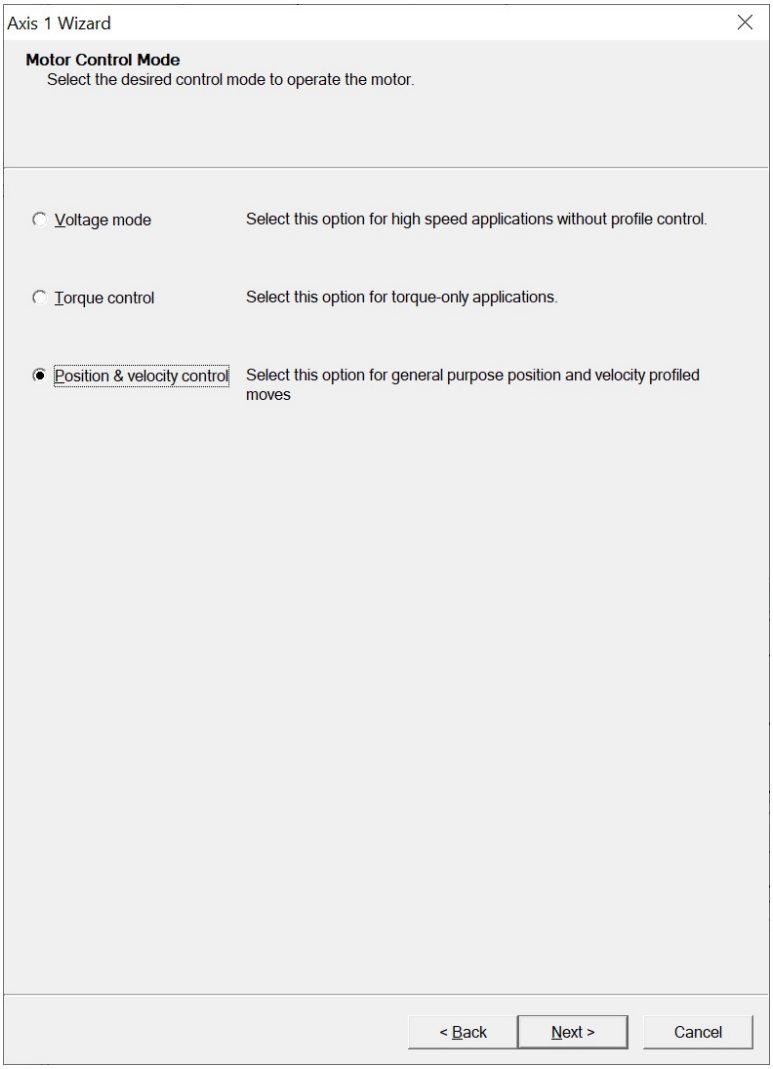


9 To check that the capture is working set a test velocity value and hit either the right or left arrow key to start motion. If Index was chosen for the capture signal a capture should occur within one full rotation. If Home was chosen, depending on the nature of your setup, more motor movement may be required before the capture occurs.

10 If the capture occurs before you expect you may need to invert the capture signal sense. When testing your settings remember that a received capture must be reset to re-enable further captures. This is done by hitting the Reset key in the Capture event box. When the capture is functioning correctly hit Next.

1.11.5 Axis Wizard Next Steps

The next screen, shown below, allows you to select the motor control mode.



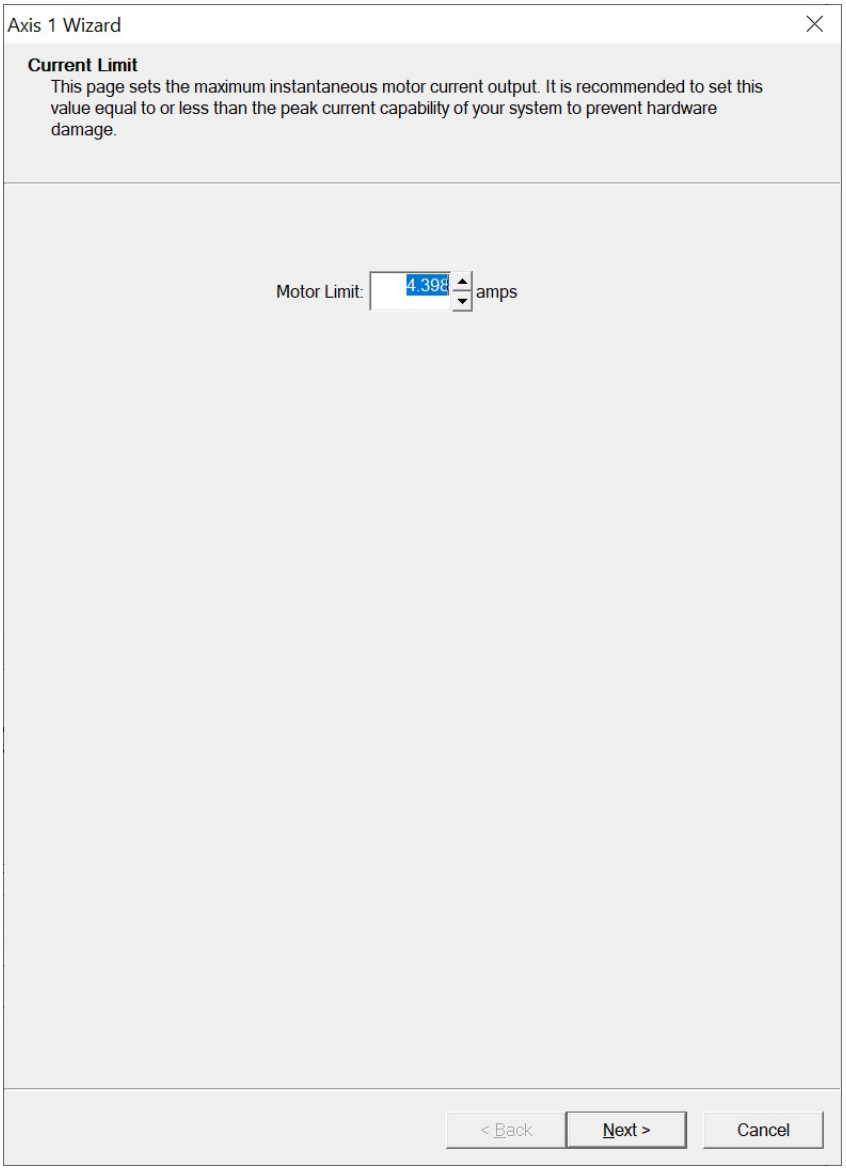
1 Three different control modes are available for operating your motor. Many users will choose the third option, "Position and Velocity Control". This control mode uses trajectory profile generation and a position PID servo control loop. Applications which require precise control of the motor's position or velocity should choose this motor control mode.

Users interested in controlling only the motor's torque output should select the second option, "Torque Control". This motor control mode does not control the position or velocity of the motor. It only commands a specified torque. When operating in this mode you should be aware that if there is nothing to impede the motor movement even a small torque (current) command may result in a very high spin rate. This control mode is often used for applications such as web control and other torque control applications.

The first option operates the motor in voltage mode. This control mode is typically used for open-lop spinning of the motor at high speed. Like current loop mode the motor will spin at a very high speed even with a fairly small voltage command. Unlike Torque Control Mode this is the most common application of voltage mode and nothing should impede the motor motion because doing so may result in an excessive amount of current being delivered to the motor.

For more information on how each of these three different control modes are implemented in the Magellan Motion Control IC refer to Section 4.4, "Motor Control Mode Operation."

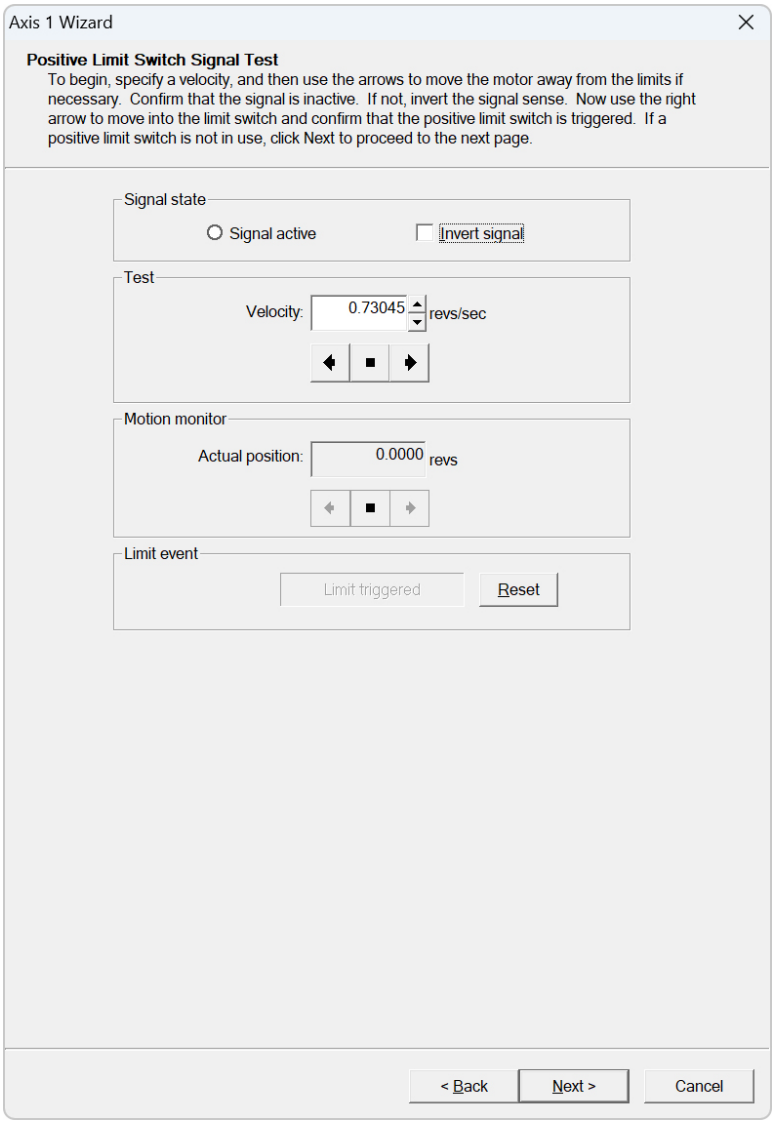
2 After making your selection and hitting Next one of two screens will show. If you selected Position and Velocity Control mode the next screen is described at step 3 below. If you selected Torque Control mode or Voltage mode skip to Section 1.11.6, "Axis Wizard Next Steps," for a description of the next screen that will appear.



3 This screen allows you to set the maximum instantaneous current output to the motor. The default value shown can be decreased, but can not be increased for protection of the motor.

4 When ready select Next to advance to the next screen.

After clicking Next the following screen will be displayed.

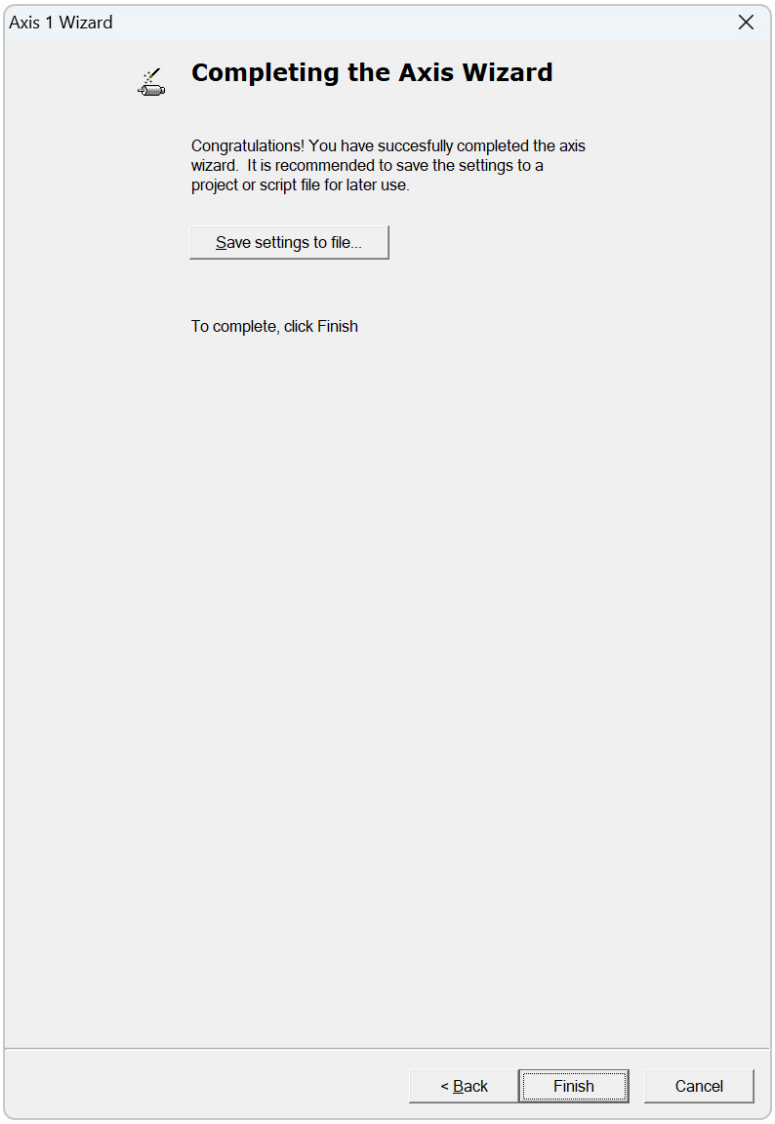


5 If you have wired limit switches to your system the above screen and the next screen (one for the positive limit input, one for the negative limit input) will let you test that they are wired correctly. The settings function in the same way as the previous Capture Input Signal Test screen. You can start a test move in the positive or negative direction and invert the limit signal sense if needed. If you are not using limit switches you may select Next to skip past these two screens.

1.11.6 Axis Wizard Next Steps

The last screen allows you to save the various control parameters you have specified while in the Axis Wizard to a setup script file. Setup script files are editable files which can be called up during a regular Pro-Motion's session via the File/Project Open function. This is convenient because it allows you to quickly load your control settings and make

them active without the need to re-run the Axis Wizard. We will present more information on Project Open and Project Save functions in a later section.



1 To save the Axis Wizard settings select "Save settings to file..." and specify a destination directory and file name. The Axis Wizard will create the setup script with your parameters loaded into it.

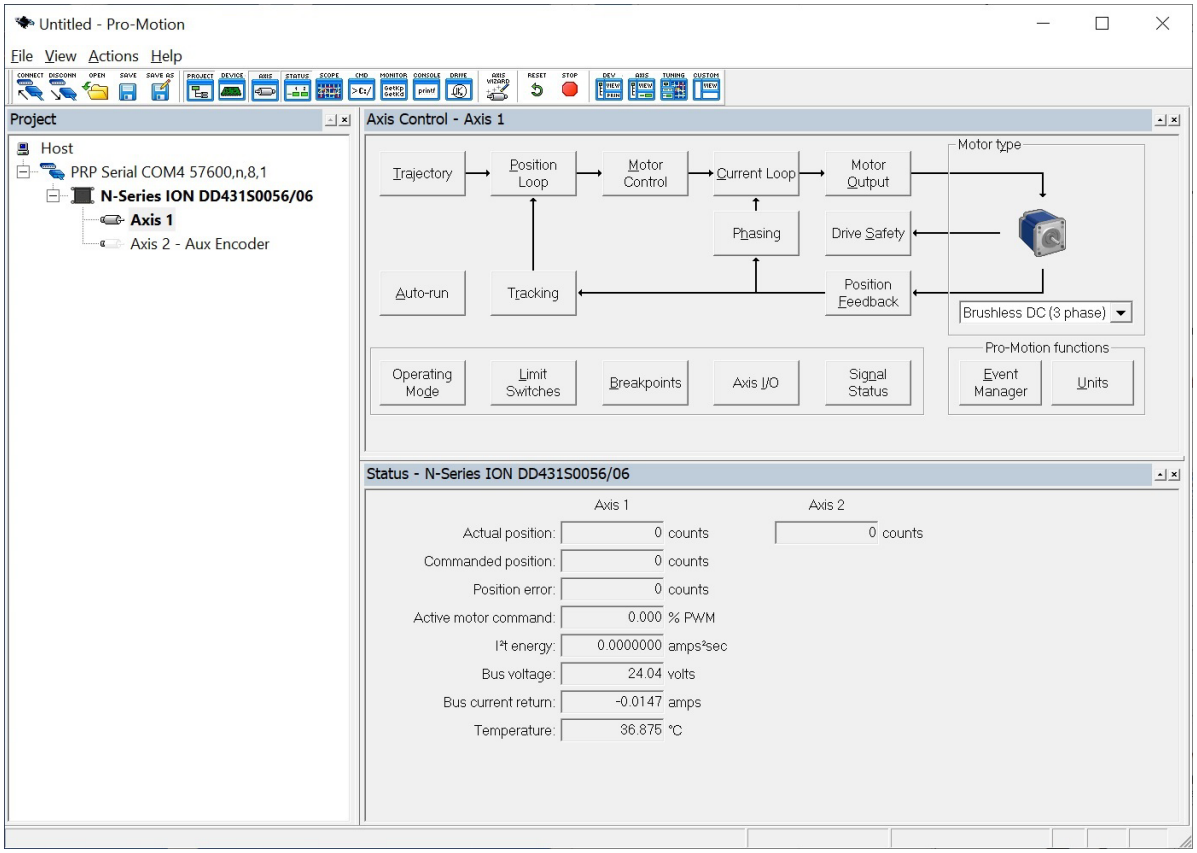
2 When you have specified a setup script file name and saved your settings select Finish at the bottom of the screen. You will now be returned to the main Pro-Motion screen.

1.11.7 Starting A Regular Pro-Motion Session

The next overall step to our 'getting started' process is to reset the N-Series ION to its default configuration, close Pro-Motion, and then restart Pro-Motion calling up the saved setup script to confirm that the motor settings were saved correctly via the Axis Wizard and that motor motion can be stably commanded. Here are the steps will we use to achieve this:

1 The screen image below shows Pro-Motion after completing the Axis Wizard setup. From this screen, to bring the N-Series ION back to its default condition select the Reset function from the top bar of icons. When the Reset dialog box appears select OK. A new dialog box should pop up within a few seconds indicating that a reset has occurred. Select "X" in the upper right to clear this dialog box.

Note that after resetting, power will no longer be applied to the motor and all N-Series ION control parameters will be set to their default values.



2 Select File in the very upper left of the Pro-Motion window and then Exit. After doing so the Pro-Motion screen should disappear.

3 Once exited from Pro-Motion, restart Pro-Motion. After a brief pause the Interface dialog box will appear. As described earlier in this manual, you should select Serial and the COM port # that the developer kit is connected to. Once connected the Pro-Motion screen should once again look like the above screen image, with the N-Series ION icon and product part numbers showing in the Project window to the upper left.

4 Next, select File and Open Project. You will be asked to specify a setup script, which should be the same setup script you saved at the end of the Axis Wizard. Once you specify a file and select Open in the dialog box the script will be input by Pro-Motion and the content sent to the N-Series ION unit. In the event that there are errors during script input, or errors that occur as the N-Series ION receives and processes the script content, a dialog box will appear indicating this.

5 Once the script has processed a dialog box will appear asking whether you are ready to energize and initialize the motor. In the case of a Brushless DC motor this means the commutation is initialized and depending on the motor control mode setting the current loop and the position loop may be enabled. Once initialized, the motor is ready to execute commands provided by the user.

1.11.8 Exercising The Motor

The next several sections will provide instructions on exercising your motor and verifying that it is reacting correctly to programmed commands.

There are three sets of instructions depending on the choice of motor control mode selected in the Axis Wizard:

If you selected Position and Velocity Control mode then you should continue to the next section, Section 1.11.9.

If you selected Torque Control mode then you should skip to Section 1.11.10.

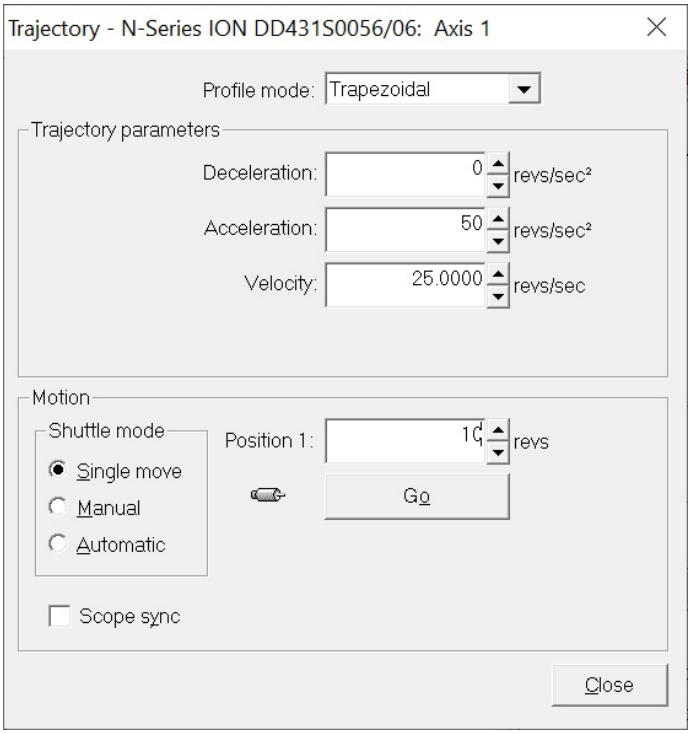
If you selected Voltage mode then you should skip to Section 1.11.11.

1.11.9 Performing a Simple Trajectory Move

If you selected Position and Velocity Control mode during Axis Wizard setup, the last step in first time system verification is to perform a simple profiled trajectory move.

To perform a simple move:

1 Click the Trajectory button in the Axis Control window. The Trajectory dialog box appears, shown below.



2 In the Profile mode list, select Trapezoidal, and in the Motion box select Single move.

3 Enter motion profiles for deceleration, acceleration, velocity, and destination position (Position 1) that are safe for your system and will demonstrate proper motion. The units of these parameters should match the units you requested earlier in the Axis Wizard Setup process. If you would like to change the units you can do this by going to the Axis Control Window and clicking the Units box which is to the far lower right on that window. You should then exit and re-enter the Trajectory dialog box for the units change to be visible.

4 Click Go and confirm that the motion occurred in a stable and controlled fashion.

Congratulations! First-time system verification for this axis is now complete.

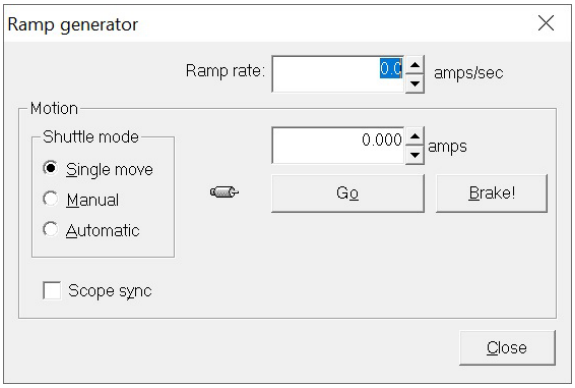
Section 1.12 of this chapter provides a guided tour of frequently used features of Pro-Motion. You may find this helpful if you are new to Pro-Motion and PMD's Magellan Motion Control IC, but taking the guided tour is optional and not required to initialize your motor or your setup. Having completed the above sequence your motor is fully initialized and ready for operation.

1.11.10 Specifying a Motor Torque Command

If you selected torque control motor control mode during Axis Wizard setup, the last step in first time system verification is to output torque commands to the motor.

To set motor output torque levels:

1 Click the Ramp Generator button in the Axis Control window. The Ramp generator dialog box appears, shown below.



2 Enter a desired torque command in amps and enter a ramp rate in amps/second that is safe for your system. When controlling a motor which is not attached to a load a very small torque output should be used, perhaps 0.1 amps. Since there is nothing to impede motion of the motor even very small torque commands will generate high spin rates.

If your motor is installed in a mechanism that can impede the motor spinning in an uncontrolled manner then the output command can be set to a higher value up to the continuous current rating of the motor.

3 Click Go and confirm that the motors reacts as expected to the programmed torque command. If desired, and depending on your setup, you can try other torque values including negative torque command values. This will result in the motor spinning in a backwards (counts decreasing) direction.

Congratulations! First-time system verification for this axis is now complete.

Section 1.12, "Going Further with Pro-Motion," provides a guided tour of frequently used features of Pro-Motion. You may find this helpful if you are new to Pro-Motion and PMD's Magellan Motion Control IC, but taking the guided tour is optional and not required to initialize your motor or your setup. Having completed the above sequence your motor is fully initialized and ready for operation.

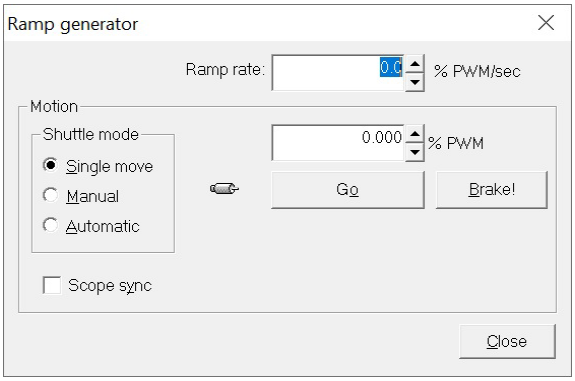
Both torque control mode and voltage control mode do not explicitly control the position or spin rate of the motor. It is up to the user to determine that operating in these modes is safe for the motor and the mechanical setup that the motor is connected to.

1.11.11 Specifying a Motor Voltage Command

If you selected voltage control motor control mode during Axis Wizard setup, the last step in first time system verification is to output voltage commands to the motor.

To set motor output voltage commands:

1 Click the Ramp Generator button in the Axis Control window. The Ramp generator dialog box appears, shown below.



2 Enter a desired voltage command in % PWM output and enter a ramp rate in %PWM/sec that is safe for your system. PWM is short for Pulse Width Modulation which is the method by which the N-Series ION drive controls its amplifier. To set the % level use the DC Bus voltage and multiply by the % PWM setting. For example if your DC bus supply is 18V and you want to ramp up to a voltage of 9V you would enter a setting of 50 % PWM.

To protect the motors against excessive heating the voltage setting as well as the ramp rate is limited on a motor-specific basis. If you enter a % PWM command or ramp rate that exceeds the limits for your motor the maximum allowed value will instead be inserted and used.

3 Click Go and confirm that the motor reacts as expected to the programmed voltage. If desired, and depending on your setup, you can try other voltage values including negative voltage command values. This will result in the motor spinning in a backwards (counts decreasing) direction.

Congratulations! First-time system verification for this axis is now complete.

The next section provides a guided tour of frequently used features of Pro-Motion. You may find this helpful if you are new to Pro-Motion and PMD's Magellan Motion Control IC, but taking the guided tour is optional and not required to initialize your motor or your setup. Having completed the above sequence your motor is fully initialized and ready for operation.

PWM % limits are calculated by measuring the voltage of the DC supply during the Axis Wizard configuration setup. Therefore the DC supply voltage should not be changed without re-running the Axis Wizard setup sequence so that correct % PWM limits can be re-calculated and stored.

Both torque control mode and voltage control only mode do not explicitly control the position or spin rate of the motor. It is up to the user to determine that operating in these modes is safe for the motor and the mechanical setup that the motor is connected to.

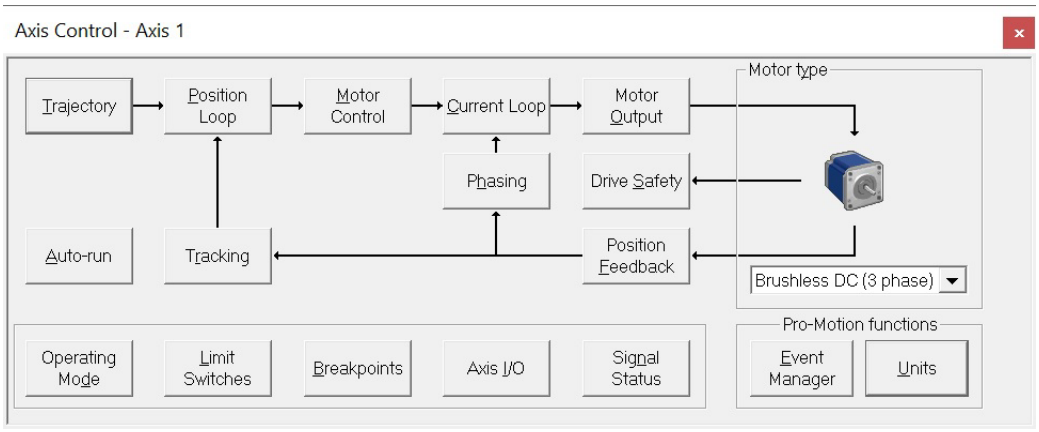
1.12 Going Further with Pro-Motion

In this section we provide more information on Pro-Motion to help familiarize you with its most commonly used features, and information that may be helpful as you start developing your motion application. We will cover the following areas:

- Axis Control Window
- Auto-Run Scripts
- Trace & Motion Oscilloscope
- Adjusting The Position Loop Parameters
- Troubleshooting Suggestions

1.12.1 Axis Control Window

We have already used the Axis Control Window in our getting started procedure. Each selectable box (technically these are Windows buttons) in the Axis Control Window results in a dialog box being opened letting you access a sub-set of the functions provided by the Magellan Control IC. For example there are boxes that access the Magellan IC's trajectory generator (labeled 'Trajectory', the position servo loop (labeled 'Position Loop'), drive safety settings (labeled 'Drive Safety') etc.



Visually the Axis Window presents these selectable boxes such that the overall control flow for the motor control mode selected is evident. For example if the motor is operated in position control mode an arrow exits the trajectory generator and enters the position loop box, which in turn connects via an arrow to a box called motor control. The boxes and arrows reflect the actual control flow in the Magellan IC. Whether or not specific boxes or connections exist depends on the control mode selected. For example if the motor is operated in voltage mode a ramp generator box appears and there is no Trajectory or Position Loop box.

Some of the boxes at the bottom of the Axis Control Window are not connected via the control flow arrows because they set or view the status of system-wide functions. These include access to settings for Limit Switches, Breakpoints, Axis I/O, Signal Status, and others.

The box labeled 'Operating Mode' provides control of whether major axis control functions are active. These control functions are trajectory, position loop, current loop, and motor output. In Position and Velocity Control mode all of the operating mode functions are active, but there may be circumstances, for example if a motion error occurs, where some will get disabled by the Magellan IC for safety reasons and may need to be manually re-enabled. For the Torque Control and Voltage modes the control diagram and operating mode settings are somewhat different. For details refer to Section 4.4, "Motor Control Mode Operation."

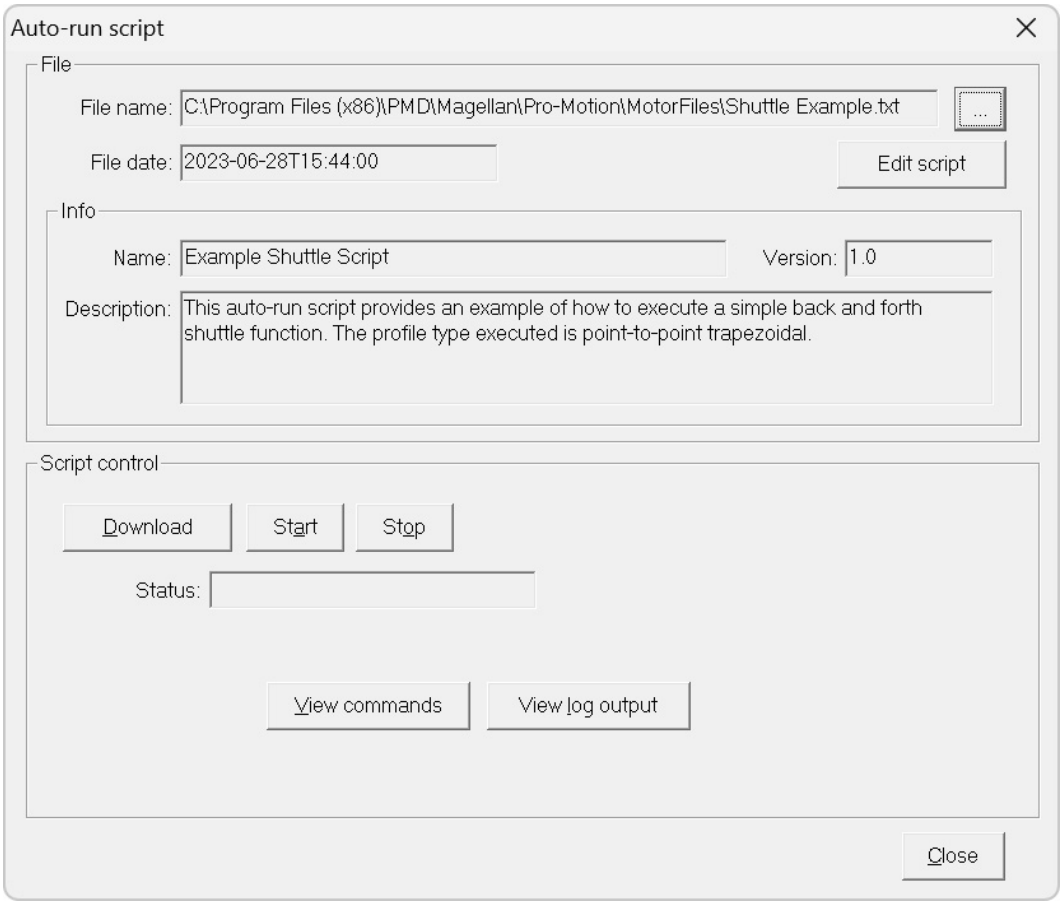
Underpinning the control flow arrows and selectable boxes in the Axis Control Window is the control architecture of PMD's Magellan Motion Control IC. The reference manual that describes this is the *Magellan Motion Control IC User Guide*. For example if you want to know what Trajectory profile modes are available, and exactly how they function and what parameter settings they require this manual contains this information. The same applies for the other selectable boxes and associated functions within the Axis Control Window. As you become familiar with how the Magellan IC controls the motor axis it will be easier and more intuitive for you to access those motion control functions via Pro-Motion.

1.12.2 Auto-Run Scripts

On the left hand side of the Axis Control Window you will notice a selectable box labeled "Auto-run". Auto-run refers to scripts which provide a way to program your own application-specific motion sequences.

As part of our Pro-Motion guided tour we will load an example auto-run script, in this case a script that implements a shuttle function, edit it, and execute it. For a general introduction to PMD's scripting system refer to Section 4.2, "Introduction to Auto-run and Setup Scripts."

1 Start by selecting the Auto-run box from the Axis Window.



2 Next click the file browse button in the upper right of the box, view the available list of scripts, and select the one titled "Example Shuttle". After doing so the Name, Version, and Description fields in the dialog box will fill-in with content as shown in the image above.

3 Select the Edit Script button in the upper right, and an ASCII text file will display using Microsoft Notepad. This is the content of the script file that encodes the example shuttle function. You are welcome to read the comments at the top of the file. Comments begin with a single quote character and may be located at the beginning of the line or to the right of a command line. Comments are not

processed by the script compiler, they are strictly to help the viewer of the script understand the script's content and function.

In addition you can also view the :CN, :DESC, and :CVER entries below the initial comment block at the top of the file. You will notice that the content in the script file matches what is displayed in the auto-run dialog box for the Name, Description, and Version fields. These entries which begin with ":" are special fields that Pro-Motion can read and make visible.

4. Because we are going to edit the file, to preserve the original file it will be safest to first make a copy of the file using Notepad's File/Save As menu function. Feel free to create a new file choosing any name different from the original.

5. Once you have created a copy of the original file with a new name locate the line which contains the comment "Cycle count set to 100" and which contains the script command 'Copy R2 100'. Edit this line so that 50 back and forth shuttle cycles are executed rather than 100 by changing Copy R2 100 to Copy R2 50. If desired you can change the comment to "Cycle count set to 50" so that it correctly describes the command line function after your edit.

6. Make a change in the :DESC entry which is located toward the top of the script file to let you confirm that your script file edits were recognized by Pro-Motion. For example the existing :DESC entry is: "This auto-run script provides an example of how to execute a simple back and forth shuttle function. The profile type executed is point-to-point trapezoidal." You could change this to "Modified version of auto-run example shuttle script with 50 instead of 100 shuttle cycles."

7. Now save the file using the Notepad File/Save menu function and return to the auto-run dialog box. Locate the new file name you have created using the browse button and select the file, thereby loading the content of the new file you have just created. You should now see your modified :DESC entry in the Description field box.

8. Select Download which is one of the buttons in the 'Script Control' portion of the dialog box. If a small dialog box appears indicating an error re-check your edits.

9. With the download successful click on View Log Output. Opening this window will allow print statements embedded in the example script to be displayed.

10. Finally, click Start. You should immediately notice your motor moving back and forth by a distance of 5,000 counts or microsteps (depending on whether you are operating the motor in closed loop stepper mode or microstepping mode). Simultaneously you should see a series of print messages outputting to the log window indicating the loop # being executed.

The trajectory parameters in the Example Shuttle script move the motor back and forth rather slowly, at approximately 1 Hz. You can increase or decrease this rate by editing the script and altering the distance traveled, the acceleration command, or the maximum velocity command. To do this search for the comment "'// Specify profile type, velocity, accel, and decel" and edit the trajectory setting commands below this comment.

Although for safety reasons the default shuttle trajectory parameters are conservative, the auto-run system is capable of high speed shuttle frequencies. Depending on the motor, the distance traveled, and the load attached to the motor shuttles can operate at 100 Hz or higher.

To learn more about the Magellan commands that are used to command and monitor trajectory moves as well as any other aspect of how the motor is controlled refer to the *Magellan Motion Control IC User Guide*. To determine the correct ASCII mnemonic syntax and required arguments for specific Magellan commands refer to the *C-Motion Magellan Programming Reference*.

For a general introduction to scripts refer to Section 4.2, "Introduction to Auto-run and Setup Scripts." and for a complete reference on Auto-run scripts and its language features such as branch instructions, mathematical operations, and printing, refer to Chapter 2, Auto-Run Scripts.

1.12.3 Trace & Motion Oscilloscope

Many users of the Portescap Motor Application Development Kit will have a specific performance goal in mind, or will want to characterize the performance of the motor when connected to the hardware in their application.

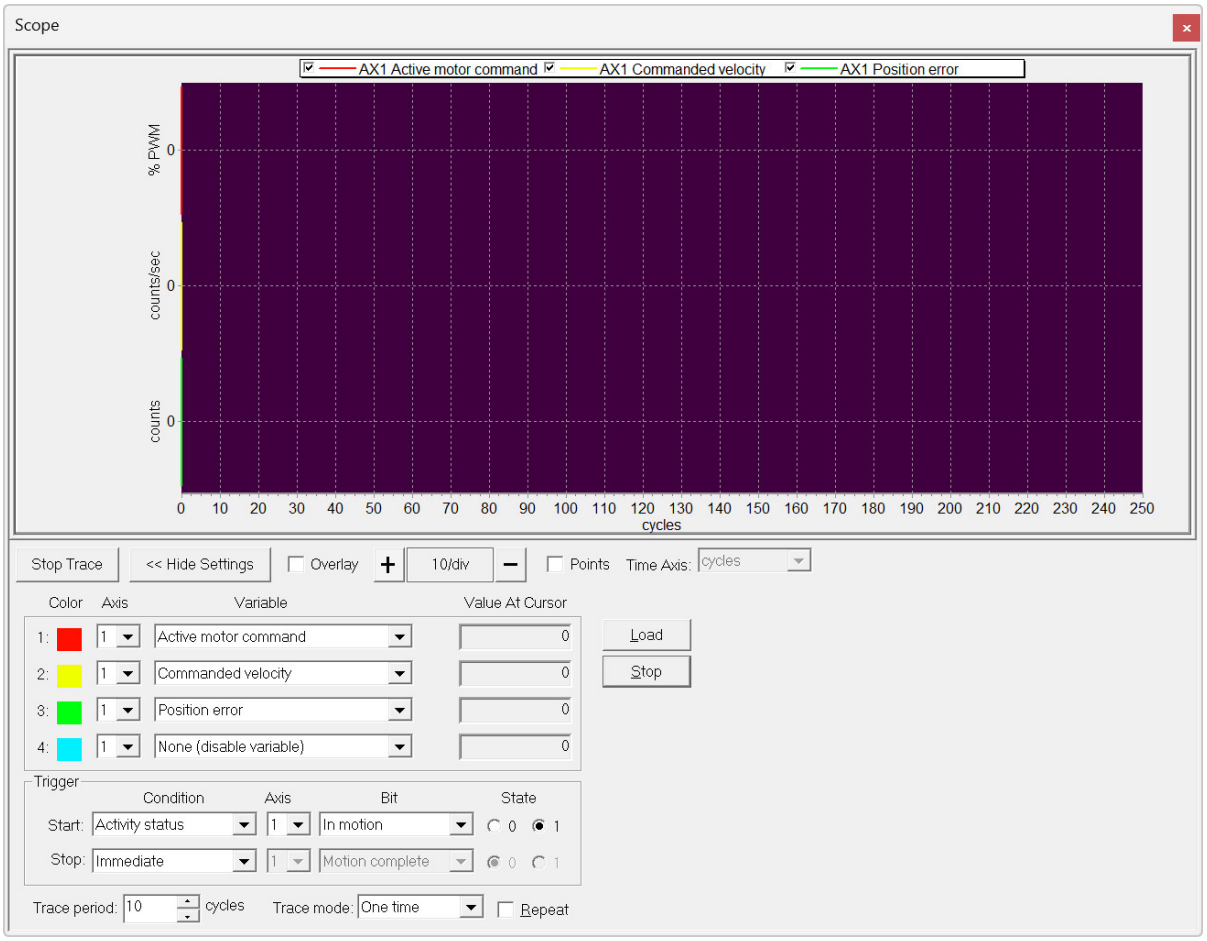
Both optimizing and measuring performance can be accomplished using Pro-Motion's Scope function. These Oscilloscope functions tie directly to a very powerful feature of the Magellan Motion Control IC called trace, which provides the ability to capture and store in hardware memory up to four motion registers simultaneously, at up to 20 kHz speed.

Once a trace operation is specified the Magellan IC captures the motion values at a programmed time interval and stores these values in its local RAM memory. Once the data has been captured Pro-Motion sends commands to retrieve the data from the Magellan IC and then display this data graphically. In addition to being displayed, traced data can be captured to a file for import to spreadsheets or other graphing and analysis software.

There are over 100 different trace variables selectable, but common traced variables include the motor output command, servo position error, commanded position, commanded velocity, and the actual position.

1.12.3.1 Using Pro-Motion Scope Function

To access the Pro-Motion scope function click the icon at the top bar labeled "Scope". A window will open up which you can resize and move if desired.



Here are some of the key settable fields of the Scope Window:

Trace Variables – The core of the trace and scope function is the list of motion registers that will be traced and graphed. These are shown as Variables 1 through 4. For each trace variable click the down arrow which in turn displays a list of trace categories such as Commanded Trajectory, Feedback, Position Loop, etc... Selecting one of these categories then shows the specific available traceable motion variables.

Data Graph – The top portion of the scope window graphs captured data. Up to four variables can be graphed at the same time. The horizontal scale is time, with selectable units via the Time Axis field of cycles, milliseconds, and seconds.

Trigger Controls – Similar to a regular oscilloscope the conditions by which trace data collection can start or end is settable. Use the down arrow key to see the list of available trigger registers, and the associated bit or signal for each register. Popular bits to start trace on are the In Motion bit of the Activity Status word, specific external signals in the Signal Status Register, and one of the two breakpoint bits of the Event Status register. For each trigger bit selected the specific state, low or high also needs to be set.

Trace Control – Two other fields in the scope window are especially useful. Trace Period and Trace mode. The trace period is expressed in cycles, which are 51.2 µSec each in length. So specifying a period of 1 cycle captures data at 20 kHz. Trace mode can be set to One-time or Rolling Buffer Mode. Rolling buffer is useful for monitoring events at slower speeds (higher trace periods). One-time capture fills the trace buffer only once when the trigger conditions are satisfied.

When using Rolling buffer mode caution should be exercised to avoid overflowing the buffer. This can occur if the rate of data being put into the trace buffer by the Magellan IC is faster then the rate at which Pro-Motion can request it. When a buffer overflow occurs a box in the scope window indicates this, and the data being graphed may show a discontinuity. In general selecting One time capture for the trace mode is recommended unless a particular reason for selecting Rolling buffer exists.

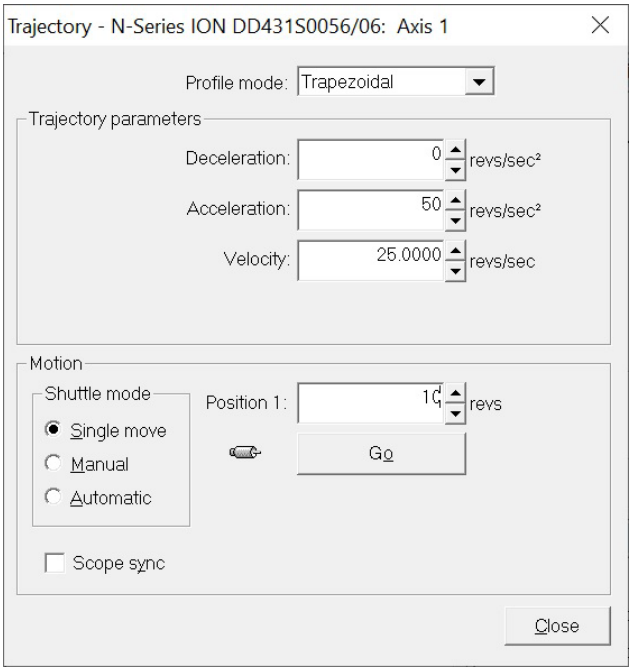
Trace Length – The total number of trace samples can be specified via this setting. Setting this value may help you better-manage how your data displays or how it exports to a file. The programmed value represents the number of data set captures. For example if this value is set to 500 and one variable is traced a total of 500 data values will be captured, if two variables are traced a total of 1,000 data values will be captured etc...

1.12.3.2 Example Performance Optimization Session

Many different optimized control parameter settings can be explored using the trace function's extensive list of traceable variables. In the sequence below we will provide an example that shows how to iteratively find the profile parameters to maximize acceleration speed.

Our goal is to determine what maximum acceleration the motor and attached mechanisms can achieve without exceeding the safe current limit specification for the motor. In this example we will assume motor is driven in position control mode.

1 Start by opening up the Trajectory dialog box, which can be accessed from the Axis Control Window. Specify Trapezoidal profile mode.



2 In the Shuttle mode box at the lower left select Single move.

3 Enter an acceleration value, and a velocity value, and a destination position. A deceleration of 0 commands the deceleration rate to match the acceleration rate.

4 Next open up the scope window and enter the following settings:

Trace variable 1 - Active motor command

Trace variable 2 - Commanded velocity

Trace variable 3 - Position error

Trace Period: 10 cycle

Trace mode: One-time

Start trigger condition Activity Status register, In motion bit, State = 1

Stop trigger condition - Immediate

5 Select the Start Trace button in the upper left of the trace window. The data graph display should not change. If it immediately begins to graph data wait till graph display completes and select Start Trace again. When in the proper state the graph area should go blank and although the Start Trace has been activated trace capture (or graphing) does not begin.

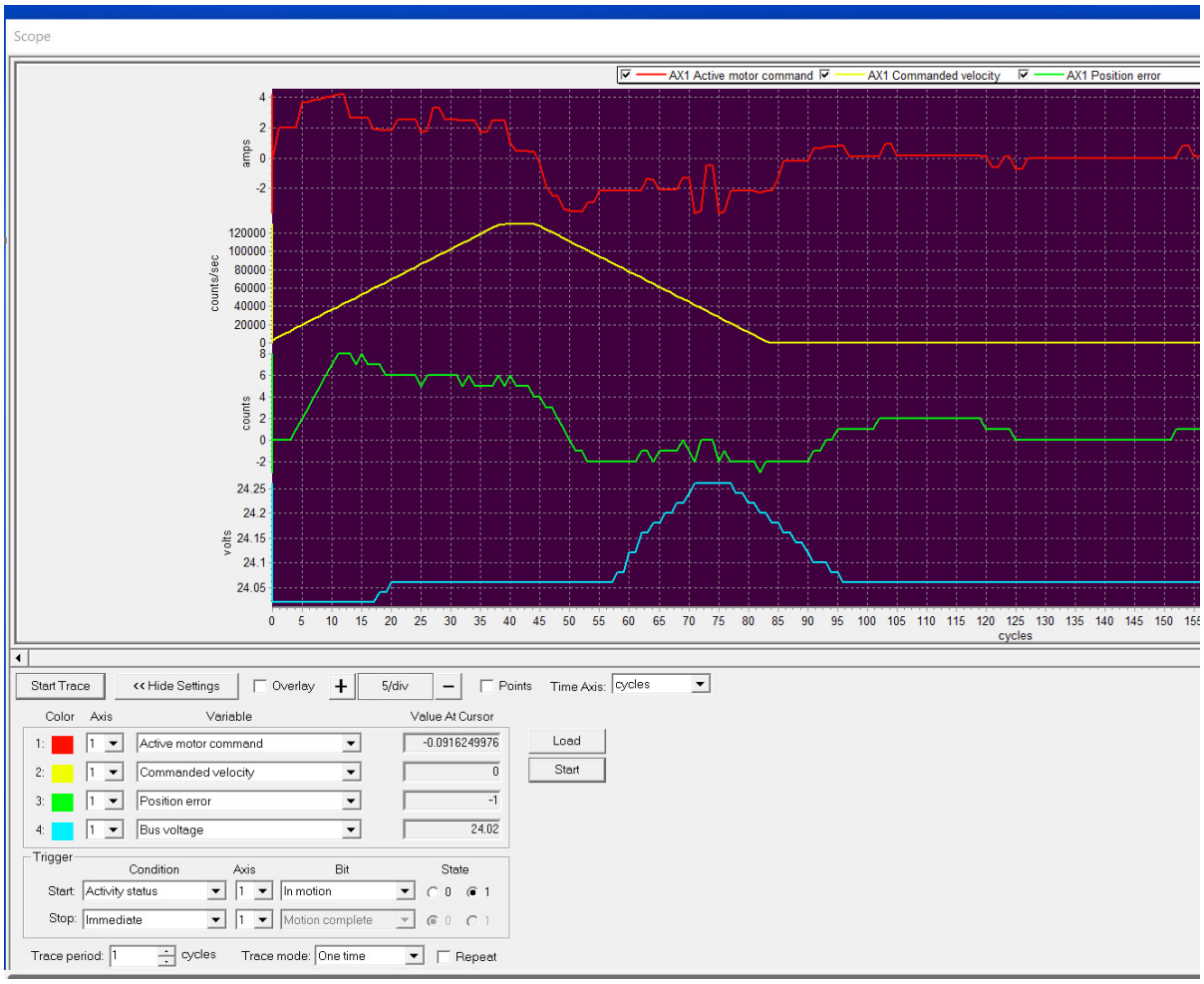
6 Review the trace length field located in the middle bottom of the screen and increase or decrease as desired. 250 to 500 is a typical number because it generally displays one oscilloscope screen-full without 'scrolling'. But the N-Series ION allows up to 8,000 data sets to be collected. This may be useful for larger traces of data that will be exported for analysis in a spreadsheet or other analysis tool. To export traced data to a file use the Pro-Motion File/Export Trace menu function.

7 From the Trajectory dialog box select Go. You should see the trace data graphing area immediately begin to display captured data and continue till a full buffer has been captured. The reason this is the case is

that by pressing Go the Magellan trajectory generator is activated and the Activity Status Register In Motion bit goes from 0 (false) to 1 (true).

To manage the data being displayed you can use the scope's **-** or **+** buttons to expand or reduce the horizontal scale. Alternatively if you would like to speed up the trace buffer update, see Section 1.12.3.3, "Trace Buffer Display Optimization," for suggestions.

8 The image below shows an example of what a trace may look like when optimizing the acceleration:



This screen capture shows data from a motor with a current drive limit of 4 amps, optimized to accelerate as fast as possible without exceeding the current drive limit. The resultant point-to-point trapezoidal move traverses 300 counts (~ 25 motor shaft degrees) in approximately 4 milliseconds. Note that despite this aggressive move the on-the-fly position error never exceeds 8 encoder counts, and the motor settles to within 2 counts error after 85 cycles, which corresponds to 4.25 mSec.

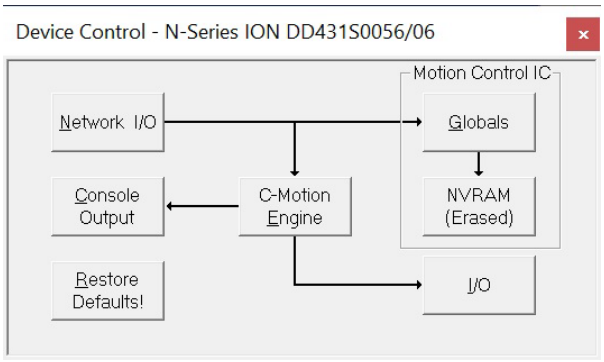
In a typical session a series of profile moves would be made exploring the effect of different accelerations. The limit of the acceleration command is determined by the maximum current rating of the motor. If too high the motor output limit is exceeded and the motor command value results in rapid increases in the position error.

Other parameters that are commonly optimized to develop the highest possible performance of the motion hardware include the position loop gain settings, the acceleration feed forward values, the maximum velocity, and the servo settling time.

1.12.3.3 Trace Buffer Display Optimization

The speed of the scope display update is related to the size of the buffer that fills with data in the Magellan IC, and the speed with which the Windows PC can retrieve this data from the N-Series ION drive. For higher speed interfaces such as CAN or Ethernet optimizing these settings is usually not needed. However for serial interface N-Series IONs, or when communicating via the 3-pin programming port connection, improving the communication speed may be useful.

To increase the serial baud rate thereby increasing throughput select the Device icon from the left side of Pro-Motion's top bar of icons. The following dialog box will display:



Click the Network I/O box and select the Serial3 tab. You will see a list of available baud rates with the current setting at 57,600 baud. Note that selecting higher baud rates may result in an increased rate of communication errors, particularly if driving the motor with high currents. If you observe this you should lower the baud rate.

The selected baud rate is stored into the ION's NVRAM but does not take effect until the N-Series ION unit is re-powered. So after setting this baud rate and selecting OK you should select the Disconnect icon in the upper left of the top icon bar and power down and then re-power the N-Series ION unit. When you connect again be sure to choose the newly-programmed baud rate.

Another important way you can speed up communications to the 3-pin programming port is to reduce the latency between message packet sends. This is done via the following sequence; First, open the Windows Device Manager by typing "Device Manager" in the Windows search box. Next, find Ports (COM & LPT) from the list and click on it, then right-click the USB Serial Port (COMn) of the serial port you are using and then select Properties. Click on the Port Settings tab and then select the Advanced button. Change the field for Latency Timer (msec) to a value of 1. Click OK on both windows and close the Device Manager. The driver is now optimized for use with the N-Series ION drive.

Alternatively, you can use the USB to 9-pin serial cable included with your developer kit which uses an RS-232 scheme. This is more robust than the 3-pin programming cable serial (UART) scheme and usually allowing higher baud rate settings without noise-related communication errors. In addition, when using the RS-232 link the latency is automatically set to the minimum value. For information on communicating via the RS-232 cable refer to Section 3.1.2, "Setting Up the ION for RS232 Communications."

1.12.4 Adjusting Position Loop Parameters

This section applies only for those applications using the controller in position loop mode.

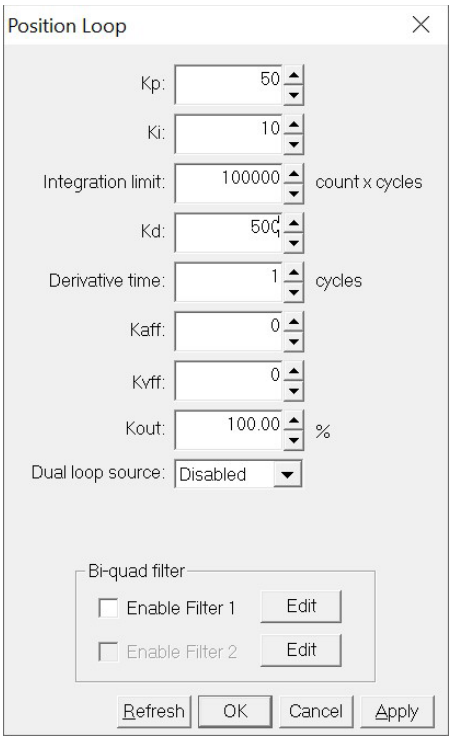
There are a number of reasons why, as you work with the Portescap Motor Application Development Kit, you may be interested in re-tuning the position loop parameters. The most common is that you have installed the motor into an actual hardware system and the inertia driven by the motor has changed. When this happens the position loop settings stored in the motor script file will very likely not work at all, or will not provide fully stable control.

The next few sections provides a simple and straightforward approach to developing position loop gain settings. The results you obtain may not be completely optimum, but should work well in a large majority of cases.

Re-tuning the position loop parameters is strongly recommended if hardware has been attached to the motor resulting in a change of inertia, or if for any reason the motor motion is altered by being incorporated/installed into the machine hardware.

1.12.4.1 Setup

1 To set up a tuning session the motor should be energized and operating with existing position loop parameters. Open the position loop dialog box by clicking the Position Loop box in the Axis Control window. The diagram below shows what this dialog box looks like:



2 Next, click the Tuning button on the right side of the top icon bar of Pro-Motion. After doing so the trace scope will be displayed along with a Step Response Dialog box. You may want to adjust the location and size of the scope window to conveniently fit your monitor size.

3 In the scope window select Active Motor Command for the first variable to trace, select Commanded Position as the second variable, and select Actual Position as the third variable. If desired you can select a fourth variable to trace but for the purpose of this description this is not necesssary.

4 In the Step Response Dialog box specify a move distance. A typical move distance for motors with an encoder is 50 or 100 counts. For motors using Halls to track the position a typical move distance is 5 to 10 counts.

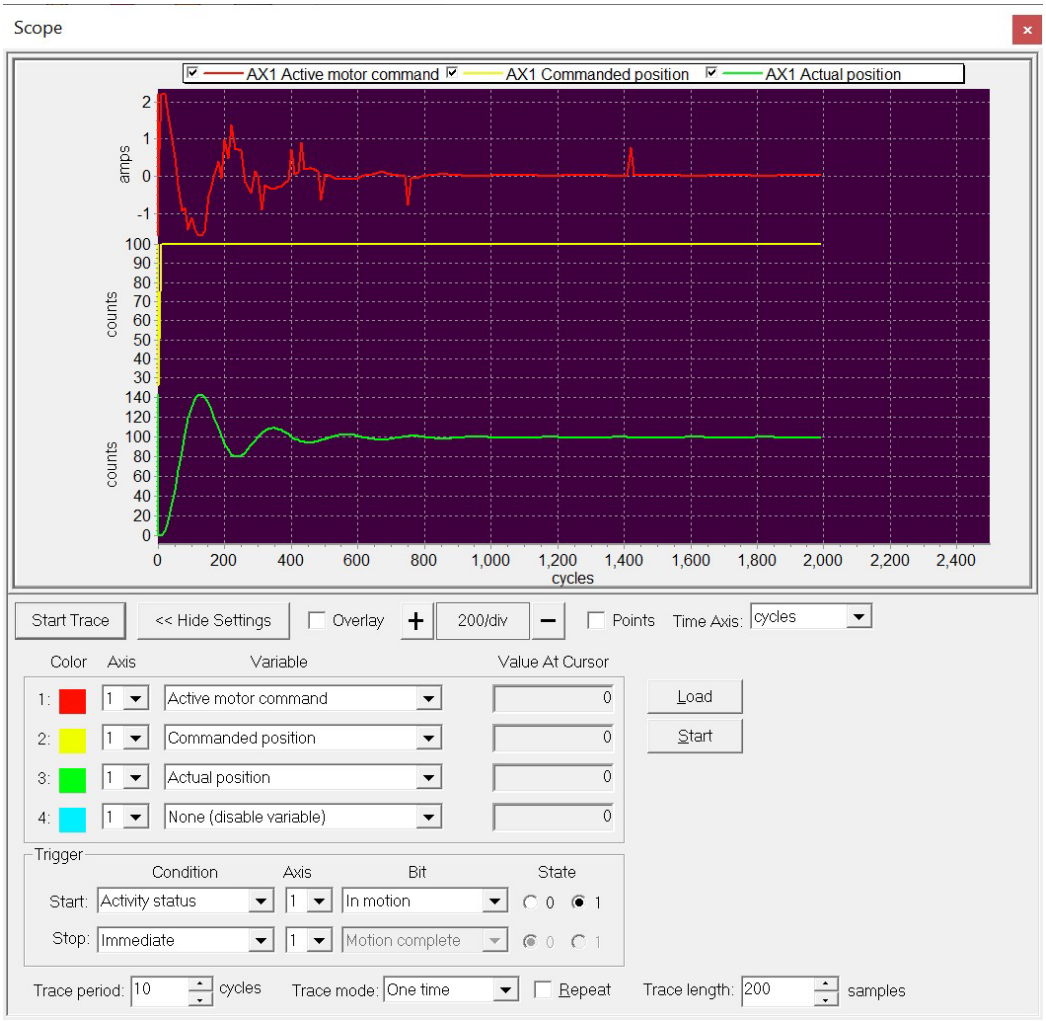
5 As confirmation of a correct setup click the right arrow key in the Step response dialog box. This should result in the motor quickly (in a single instantaneous step) moving by the programmed amount, and shortly thereafter the trace scope window should begin to display data. If this doesn't happen review the instructions above and try again.

6 To develop new tuning parameters set Ki, Integration limit, Kaff, and Kvff to zero. Set Kp to a small value, typically 10 to 50 for motors with encoders and 100 to 500 for motors with Halls only. Set the derivative value to a value 10 times greater than the Kp value you entered. Set the derivative time to 1 and set KOut to 100%.

7 Hit the Apply button and observe the physical motor. If it oscillates or vibrates set Kp to a lower value and re-apply this setting. The scope trace should display the results of the settings you entered. To tune the Kp and Kd you will look at the display results and determine whether the response is underdamped, critically damped, or overdamped and increase. These characterizations will guide you to how to change the gain settings as described in the following three sections.

1.12.4.2 Underdamped Response

The screen capture below shows a typical underdamped response.

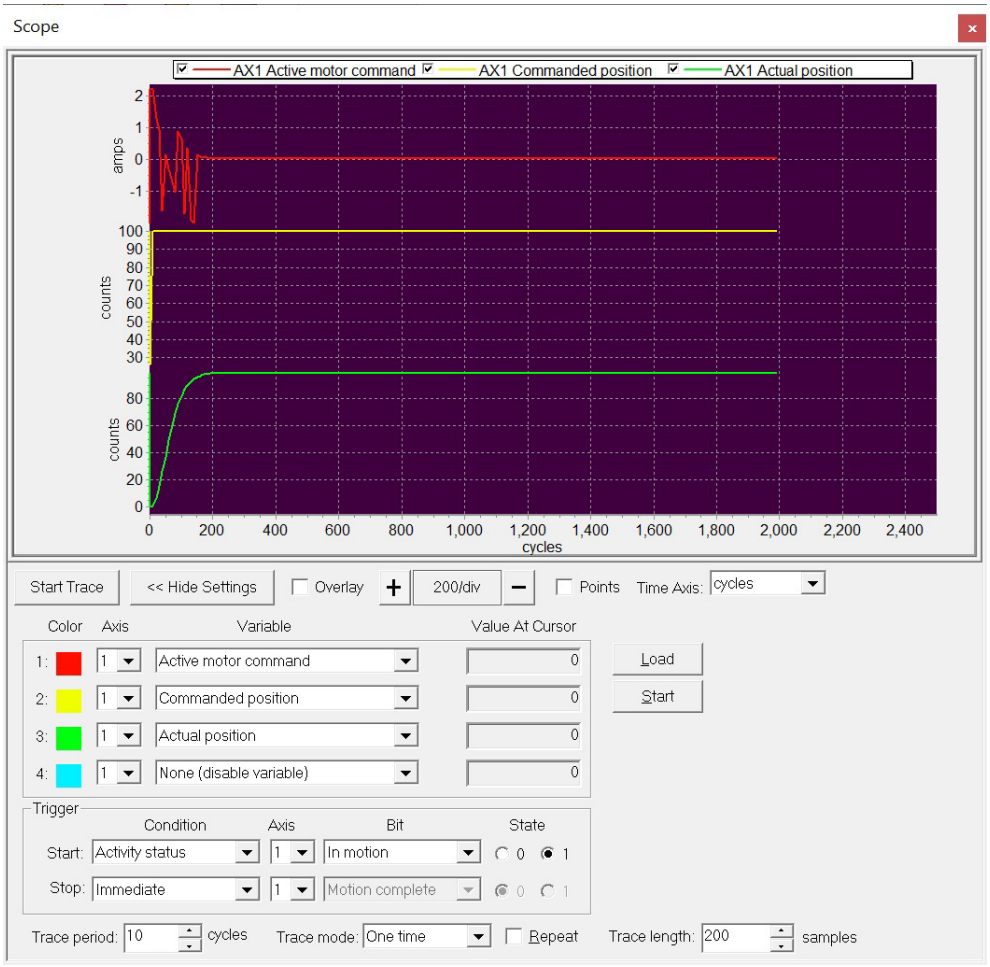


The yellow trace shows that the commanded position instantly changes from a position of 0 to a position of 100 (in your setup this jump will equal whatever you entered into the step response dialog box). The green trace shows the resultant actual motor location.

From this trace, although the actual position does eventually settle to a position value of 100, it overshoots significantly and oscillates several times. Whenever the actual axis position overshoots the commanded position the system is considered underdamped, and to correct for this you should increase the derivative term. The goal of the Kd setting is to determine a value that results in a critically damped response, as shown in the next section.

1.12.4.3 Critically Damped Response

The screen capture below shows a critically damped response.

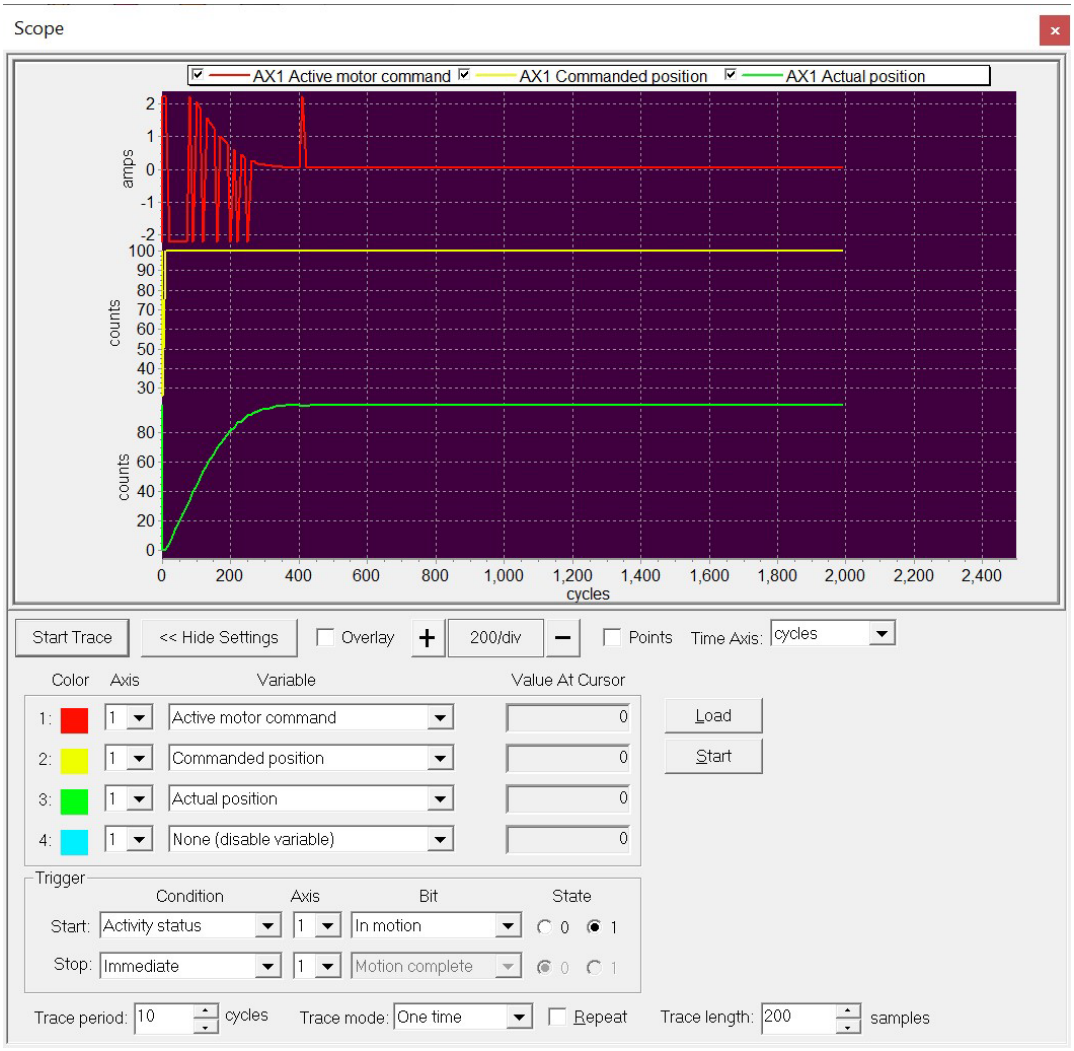


In this trace the actual position does not overshoot that commanded position, and it settles quickly for this particular motor in 10 mSec to the commanded position of 100.

It's worth noting that many Kd settings could result in a response that appears critically damped. The distinction between critically damped and overdamped, in particular, is somewhat subjective when using this manual trial and error process. In general you should try to find the smallest setting of Kd which still provides a critically damped response.

1.12.4.4 Overdamped Response

The screen capture below shows an overdamped response.



This can be seen by comparing with the critically damped response in the previous section. An overdamped system will be stable but unnecessarily slow in its response to changes in the position command.

If the response is overdamped you should reduce Kd trying to find the smallest value of Kd that still gives a critically damped response.

For purposes of illustration in the above three example graphs the Kp setting was 100 and the Kd settings were 2,000, 6,000, and 25,000 for the underdamped, critically damped, and overdamped responses respectively.

1.12.4.5 Getting To Final Gain Settings

Your overall goal is to set the Kp to as high a value as possible while still finding associated Kd values that can create a critically damped response. Higher Kp values will result in more accurate tracking and faster responses to commanded position changes.

As you try higher and higher Kp values, at some value of Kp you will find that there is no value of Kd that can create a critically damped response. At this point you should reduce the Kp setting by 35-50% and determine the associated critically damped Kd setting. Backing down from the 'borderline' Kp value is important for improving stability, accommodating small differences in controller, motor, and attached hardware behavior, and for handling changes that may occur to the system as it operates in the field over time.

Once Kp and Kd have been set, you can enter a value of Ki to improve tracking accuracy. Typically, Ki settings are 1/10 to 1/2 of the Kp value. Smaller motors typically use relatively large ratios, and larger motors with larger loads typically use smaller Ki ratios. Higher Ki settings will increase tracking accuracy during and after the motor move is complete but can also reduce system stability. So you should set Ki to the smallest value that can achieve your goals for final or dynamic tracking while not distorting the critically damped response the Kp and Kd settings created.

The Integration limit can be set to a high value, perhaps 100,000. The Magellan IC PID engine has built in anti-windup logic so it is very unlikely this integration limit will ever be reached.

The final 'core' PID setting, known as derivative time, can often be left at one. Setting it to higher values means the period at which the derivative contribution is calculated is increased. This can be useful to lower the frequency of Kd 'chatter' to below audible, should there be any in the system. Another benefit to increasing the derivative time may be to increase the influence of the Kd term. The impact of the Kd on the PID is a direct multiple of the derivative time. For example if the derivative time is set to 10 and the Kd value is 100 this would be an equivalent impact as a Kd of 1,000 with a derivative time of 1.

1.12.4.6 Frequency Based Tools & Analysis

The above process represents a popular and relatively straightforward approach toward tuning the PID loop. There are more sophisticated approaches however both for determining PID parameters and for verifying the behavior of a given PID setting. PMD provides frequency -based tools known as Bode plots to support such methods.

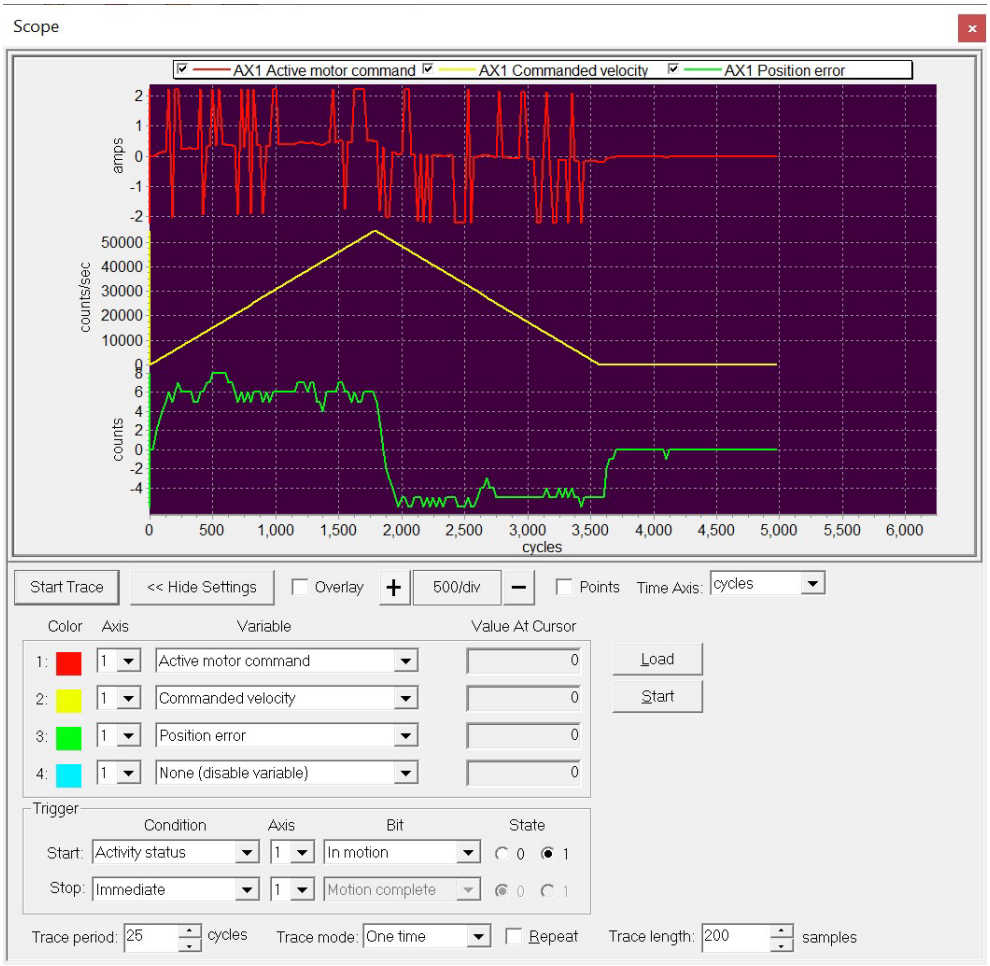
Bode plots can be used in different ways and can help characterize your mechanical system as well as determine important characteristics of the operating control loop such as the bandwidth and phase margin. The Pro-Motion Bode plot tool can be accessed from the View/BodePlots Pro-Motion menu selection.

One output of a Bode-based system characterization may be the identification of natural resonances in the mechanical system. While a properly tuned PID can help reduce the impact of this, the Magellan IC also provide two general purpose biquad filters in the position loop. Biquad filters can be used to create, low pass, band pass, and other filter types to help manage and reduce resonances in the mechanics. For more information on biquads as well as the Magellan Position PID loop refer to the Magellan Motion Control IC User Guide.

1.12.4.7 Feedforward Terms

The graph below shows a complete point to point profile move using the critically damped system settings from above. The yellow line represents the commanded trajectory velocity, and the green line represents the position error (the difference between the commanded and the actual position). Notice that despite the fact that the axis is accelerating and decelerating quite aggressively (the entire move duration is less than 200 mSec) the motion is stable throughout

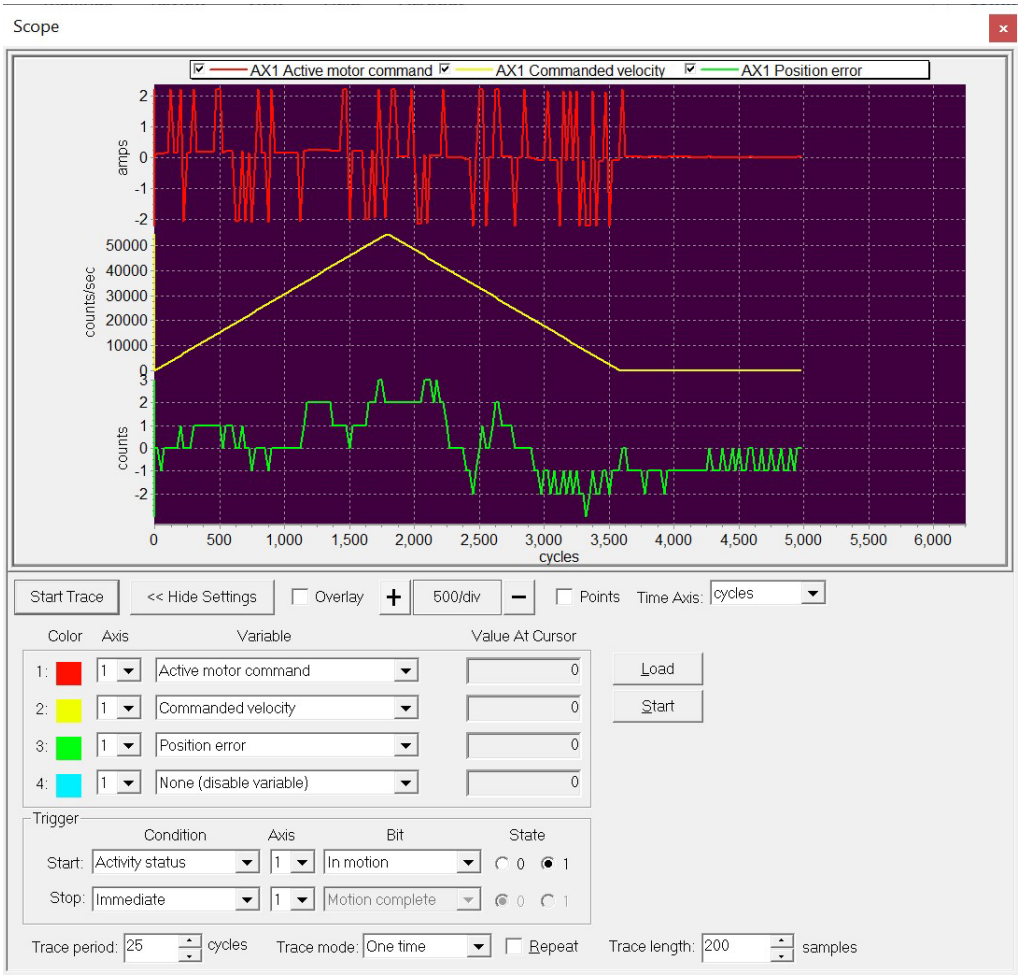
and the tracking accuracy is quite good even during the motion, varying from +8 counts maximum position error to -5 counts.



Nevertheless Acceleration feedforward (Aff for short) can reduce the dynamic tracking error even further. Acceleration feedforward works by adding a torque command proportional to commanded acceleration. PMD's Magellan IC also supports velocity feedforward, which similarly adds to the motor command proportional to the commanded velocity. Velocity feedforward is used less often than acceleration feedforward, typically to compensate for fluid friction such as lubricated bearings or fluid pumping applications.

The reason Aff improves tracking accuracy is that from the position error graph above we can see that the form of the position error echoes the acceleration value. In this trapezoidal profile the acceleration is a constant positive value as the profile increases speed and a constant negative value as the profile speed decreases. Therefore by 'pre-injecting' this feedforward value the servo loop is required to do less work, resulting in better tracking accuracy.

After adding an appropriate Aff gain to the PID loop the results are shown below. Although the tracking is not perfect, the magnitude of the position error is halved during the motion.



One caution with acceleration feedforward is that it is load specific. If you tune the Aff value by increasing and decreasing till the net position error during motion is minimized, you will find that if the load on the motor changes you will need to re-tune the Aff value to achieve a similar result. This means systems that typically carry a variable load may not be good candidates for acceleration feedforward.

1.12.5 Troubleshooting Suggestions

If the first-time verification sequence detailed in Section 1.11, "First-Time System Setup & Verification," was successful problems while using Pro-Motion are not that common. Nevertheless below is a list of issues users may encounter while exercising their motor or developing their motion applications and suggestions for correcting the issue.

The Motor Stops Responding. If a motor previously moving and responding to your commands stops responding the most likely reason is that the operating mode is not set correctly. Although most such 'transitions' from normal operation to an error/protection state result in a dialog box appearing, to check the operating mode select the Operating Mode box in the Axis Control Window and adjust the settings if needed. See Section 1.12.1, "Axis Control Window," for more on the Operating mode dialog box.

Events Aren't Being Reported. As the Magellan Motion Control IC executes its control settings some occurrences representing a change in the control system state may occur. These changes are called events and include items such as motion error, over temperature, breakpoints, and more. To check the Pro-Motion settings for which events will

result in a dialog box popping up click the Event Manager box in the lower right of the Axis Control Window. For detailed information on Magellan events refer to the

An Overvoltage Event Occurred. If during a motion profile an overvoltage event is indicated the most likely reason is that during deceleration of the motor the inertia of the motor and load resulted in net generation of energy. Depending on the design of the power supply you are using this can result in the HV voltage rising and eventually exceeding the overvoltage limit. Remedies are to reduce the profile's rate of deceleration, using the shunt feature of the N-Series ION drive, or adding more capacitance to the HV supply.

An Undervoltage Event Occurred. If during a motion profile an undervoltage event is indicated the most likely reason is that during acceleration the power supply was not able to deliver the needed amount of current, or was not able to deliver an increase in current fast enough. Undervoltage (and overvoltage) events may also occur if the current loop or position loop is unstable. Remedies are to increase the current output limit/rating of the power supply, add more capacitance to the HV supply, or reduce the profile's rate of acceleration or re-tune the servo loops.

A Motion Error Event Occurred. If during a motion profile a motion error event is indicated there are a few potential reasons. Motion error, also called position error, means the difference between the commanded position and the actual motor position has exceeded a user-settable threshold. Potential reasons are that the axis motion was blocked, that the position loop is not well tuned, that the commanded profile velocity exceeds the achievable motor velocity, or that an aggressive profile has temporarily resulted in a motion error during the move. Remedies depend on the cause, and are generally straightforward once the cause is determined.

Communication Errors Are Occurring. Communication errors between Pro-Motion and the controller are rare, but depending on the connection type and motion setup may occur. The 3-pin Programming Port for example is susceptible to noise if high current moves are commanded because it does not use a differential signaling scheme. For whatever communication link is used, you should try to determine if there are specific operating conditions that result in communication errors. If signal noise may be at fault consider switching to a link that uses a differential signaling scheme such as RS-232, CAN, or Ethernet.

1.12.6 Summary

Hopefully this brief guided tour and additional information about Pro-Motion has been useful to help you to better understand Pro-Motion and the N-Series Drive. For more information on the N-Series ION Drive refer to the *ION/CME N-Series Digital Drive User Manual.* For detailed information on the Magellan Motion Control IC, refer to the *Magellan Motion Control IC User Guide.* For information on Magellan command mnemonics and script syntax refer to the *C-Motion Magellan Programming Reference.*

2. Auto-Run Scripts

In This Chapter

2.1 Introduction to Auto-Run Scripts

Auto-run scripts are a feature of Pro-Motion and the N-Series ION Drive which let users create, call up and execute text-formatted script language files. These script language files provide the capability of specifying trajectory profiles, executing conditional testing, performing mathematical operations, and creating formatted output of messages and data.

Auto-run scripts at their core consist of Magellan Motion Control IC commands to specify axis control parameters and motion profiles moves. Added to this are general purpose instructions for language functions such as looping, branching, and logical operations.

Together these functions provide an easy to use system for programming motion sequences such as rapid back and forth profiles (shuttles), life cycle testing, PLC-style digital signal response sequences, and a wide range of other automated motion sequences.

2.2 Auto-Run scripts versus C-Motion Programming

Auto-run scripts are an alternative programming model to C-Motion, PMD's C-language based motion programming system. Broadly speaking auto-run scripts may be more convenient for applications where C-language is not being used for other aspects of the machine control, and when the overall motion sequences are single axis and not complex.

C-Motion may be the preferred programming model when other elements of the machine are already using C-language, and when more complex programming functions are needed such as multi-tasking, multi-axis machine control, or special communications and protocols.

Note that C-Motion and auto-run scripts are not mutually exclusive. Some machines may benefit from a hybrid approach where single-axis drives are programmed with auto-run scripts providing a small library of callable functions, while the

overall machine control software that coordinates execution of these callable functions occurs via C-language programming.

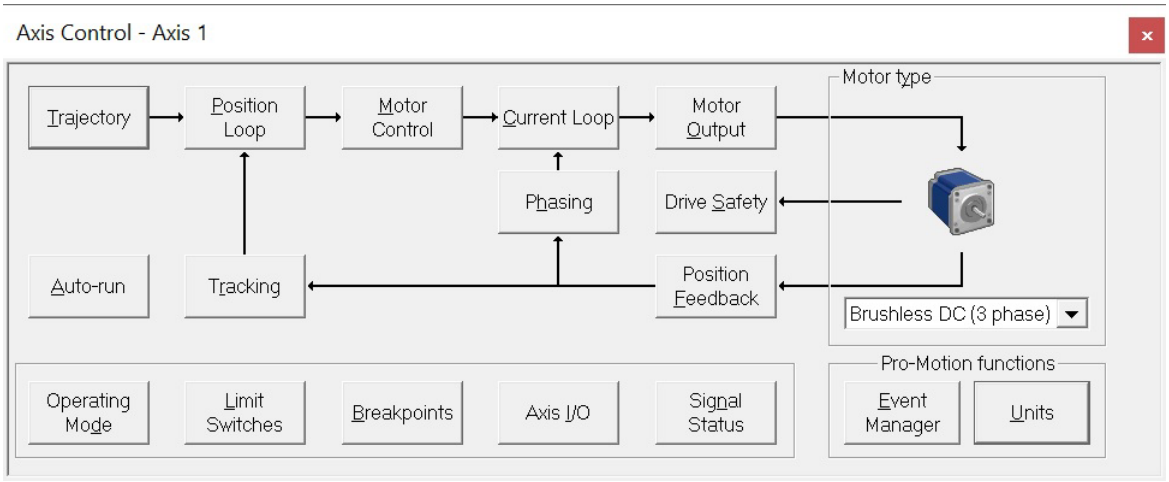
2.3 Using Auto-Run Scripts

Auto-run scripts are created and edited by the user. They are ASCII text files and have a .txt file name extension. Auto-run scripts have a relatively flexible format allowing users to tailor the look and feel to their preferences, however certain syntax conventions must be followed. For more information on this syntax refer to Section 4.2, "Introduction to Auto-run and Setup Scripts."

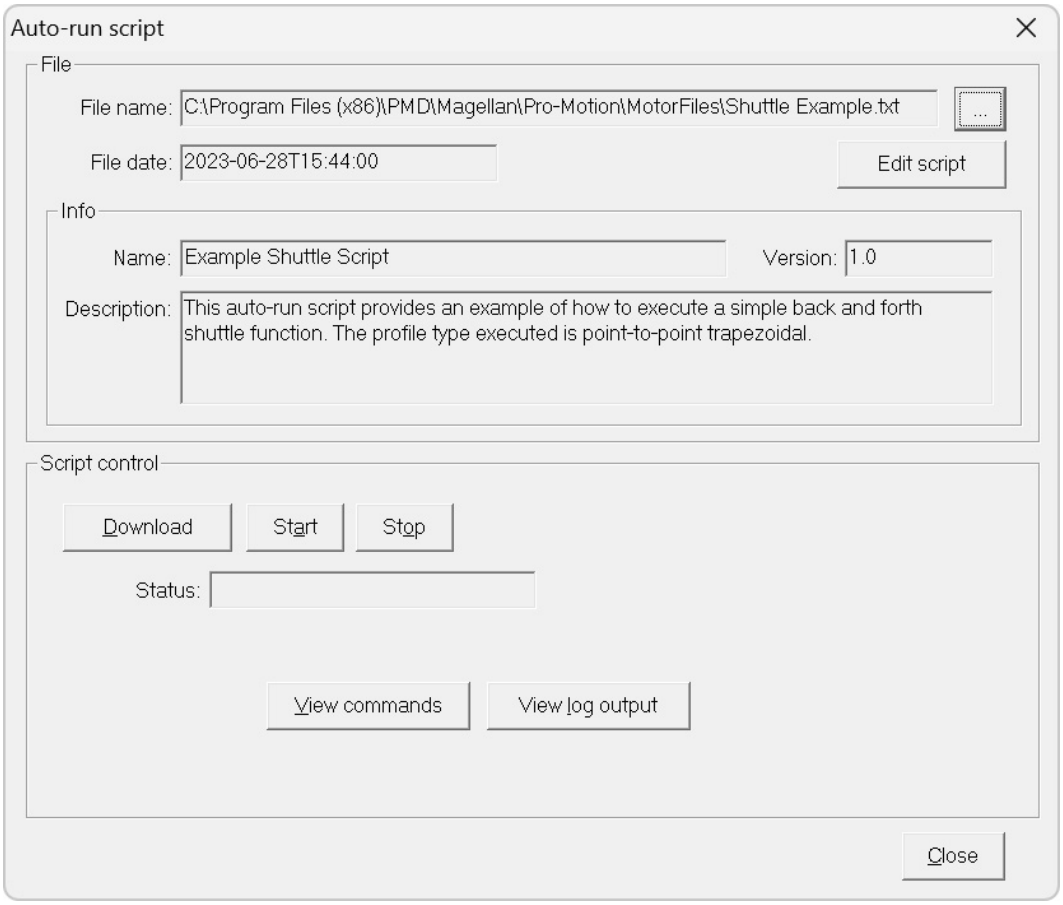
Because auto-run scripts are formatted as ASCII text files they can be created and edited by any word processor that supports text format files. Windows Notepad is a popular choice but many other editors are available. The name of the script files is up to the user to choose, however it must have a .txt extension to be recognized by Pro-Motion as a script file.

Once auto-run scripts are created Pro-Motion supports all functions needed to load, execute, control, and view print output from an auto-run script file. In this section we will detail exactly how a typical auto-run session with Pro-Motion occurs.

To access auto-run scripts from Pro-Motion the "Auto-run" box is selected from the Axis Control Window. The Axis Control window is shown below.



Once the auto-run box is selected a dialog box appears as shown below.



To load a pre-existing auto-run script select the 'File Name' function. This provides a standard Windows File Open function.

Once you have selected the auto-run script the 'Script Info' fields will populate. The content of these fields are specified within the auto-run script file. They are useful to provide at-a-glance information about the contents of the script but are entirely up to the user to specify in the script file, or not. If not specified the fields within the Script Info window will be blank. More later on how these fields are specified within the script file.

Once the script file has been specified, to download the script into the N-Series ION Drive the 'Download Script' button is selected. When download is selected a sequence of actions occur. The first is that the auto-run script is parsed and compiled. If script compiling is successful a resultant executable memory image is created, which is then downloaded into the N-Series ION unit. If compiling is not successful due to a syntax error in the script or some other problem, a dialog box will appear indicating the line that had the error. Note that if desired the "Edit" script button can be selected, which will call up the script file using Microsoft Notepad.

Although the original script file is written in ASCII, the compiled and downloaded memory image is in binary format. This memory image is loaded into a RAM area of the Magellan Motion Control IC where it will await commands to start or stop execution of the script. Note therefore that powering down or resetting the ION unit will result in loss of the downloaded script program. In addition, the RAM-based program content is not affected by execution of the script. Downloaded script programs can be executed as often as desired. Only downloading a new script program to the N-Series ION will result in erasure and replacement of a previously downloaded program.

After successful script compilation and downloading the Auto-run Status Window will indicate "Ready". To run the script the 'Start' button is selected. The Status window will display "Running". If desired, a running script can be

stopped using the "Stop" button. Once the script finishes, whether by stop command or because the script has reached its end, a dialog box will appear indicating this.

A stop command results in a motor disable command being sent to the N-Series ION. This will result in the motor being de-energized, and any actively running profiles free-wheeling to a stop.

To recover from this condition, using the Pro-Motion Axis Control Window select the "Operating Mode" button in the lower left of the window and then select "Enable All" From the Operating Mode dialog box. The motor should now once again be ready for commanded movement, whether from the auto-run script or from another command dialog box such as the Trajectory button within the Axis Control Window.

Note that if the running script is intended to start from a particular commanded position before beginning execution you may want to use the Trajectory button to move the motor to this start position. Use either the Trapezoidal or the S-curve profile mode to most conveniently achieve this.

After a script has completed execution the Start button can be selected again which will result in the script re-starting execution from the beginning of the script. Script execution can not be resumed from the stop location, it can only be re-started from the beginning.

During script execution, if print statements are encountered while running the script a window will be created showing the output of these print statements. Unless explicitly closed by the user, additional executions of the script will generate more print output to the same window. The content of this window can be saved as its own text formatted file for later review in an application such as Notepad, or the content can be copied and imported into a spreadsheet program or other compatible data analysis program. We will discuss printing in detail in a later section.

Finally, if the user exits the auto-run dialog box any actively running auto-run script will be stopped. Similarly upon entering the Auto-run dialog box, in the unlikely event that there is already an actively running script it will be stopped. To start execution of a script it must first be loaded and compiled.

2.3.1 Auto-Run & Trace

Trace is a powerful feature of the Magellan Motion Control IC that allows you to capture, at high speed, up to four simultaneous motion registers and graph the result. You can select which registers get traced. Often these will include the Actual Position, the Position Error, and other motion registers, however there are over 50 different trace variables that you can select to capture and graph.

To access the trace function click the "Scope" Icon from the top Pro-Motion bar of icons. Tracing can occur at the same time that an auto-run script is executing. This may be useful for a number of reasons, especially for optimizing position PID servo control parameters or profile trajectory parameters.

When using trace in this way make sure to leave the auto-run dialog box open while setting up and activating the trace/scope settings. As mentioned above, closing the auto-run script dialog box will result in the script program execution being halted.

2.4 Auto-run Script Key Elements

Shuttle Example

File Edit View

```
' ----------------------------------------------------------------------------------------------------
' Example Script: Motion Shuttle
'
' 6/29/23
'
' Example auto-run script which executes a specified number of back and forth shuttle cycles. Each shuttle
' consists of an outgoing move and a return move. The profile type is trapezoidal point to point. Each
' shuttle move starts without delay after completion of the previous move.
' ----------------------------------------------------------------------------------------------------

:CN Example Shuttle Script

:DESC "This auto-run script provides an example of how to execute a simple back and forth
shuttle function. The profile type executed is point-to-point trapezoidal."

:CVER 1.0

'// Set cycle count, outgoing shuttle position, return shuttle position

        Copy R2 100                     ' Cycle count set to 100
        Copy R3 2500                    ' Outgoing position set to +2,500 counts
        Copy R4 -2500                   ' Return position is set to -2,500 counts

        '// Specify profile type, velocity, accel, and decel

        SetProfileMode 0                ' Specify that this will be a trapezoidal (code 0) profile
        SetVelocity 50335               ' Velocity value corresponds to 15,000 counts/second
        SetAcceleration 85              ' Accel value corresponds to 500,000 counts/sec2
        SetDeceleration 0               ' A 0 value means decel will be same as accel

        '// set up loop. The loop count will be held in R5

        Copy R5 R2                      ' Use R5 as cycle counter, initailize count

LoopStart:
        '// Execute outgoing profile

        ResetEventStatus 0xfffe         ' Clear Event Status register including motion complete bit
        SetPosition R3                  ' Set outgoing destination position
        Update                          ' Start the move

OutgoingComplete:
        GetEventStatus                  ' Load the content of the Event Status Register into the accumulator
        And Acc 1                       ' And the event status word with a single bit mask
                                        ' corresponding to the motion complete bit
        Bz Acc OutgoingComplete         ' Loop back if motion complete bit not set

        '// Execute return profile

        ResetEventStatus 0xfffe         ' Clear Event Status Register including motion complete bit
        SetPosition R4                  ' Set return destination position
        Update                          ' Start the move

ReturnComplete:
        GetEventStatus                  ' Load the content of the Event Status Register into the accumlator
        And Acc 1                       ' And the event status word with a single bit mask
                                        ' corresponding to the motion complete bit
        Bz Acc ReturnComplete           ' Loop back if motion complete bit not set

        '// Print loop count

        Sub R2 R5                       ' Calc # cycles - counter
        Add Acc 1                       ' print count from 1 to N
        Print "Loop #: \n" Acc          ' Print loop # which is in accumulator
        Bdnz R5 LoopStart               ' Decrement loop count, if not zero loop back and continue

        Print "\n"
        Print "Shuttle Completed\n"
```

Ln 1, Col 1 100% Windows (CRLF) UTF-8

Figure 2-1: Example Auto-run Script

Figure 2-1 shows an example auto-run script file.

Here are some key elements within the auto-run script file.

File Content Fields – Several special script file specifiers allow information about the content of the script file to be specified. These are :CN, :DESC, and :CVER. Each of these fields, if specified, can be viewed from Pro-Motion during script loading.

Comments – Comments are content embedded in the script file that are not executed, but rather are intended to be useful to the script programmer or others to better understand the script content. Comments may appear anywhere within a line and result in all subsequent characters up to the end of the line being ignored during script processing. Two characters are recognized as beginning a comment; a single quote ', and a tilde ~.

Commands – Broadly speaking each line of the auto-run script contains a command mnemonic which identifies what command will be executed. Commands may have 0, 1, 2, or 3 arguments. Many commands are Magellan Motion Control IC commands that set, get, and initiate motion control actions such as **SetPosition**, **GetKp**, and **Update**. Some commands are connected with the auto-run scripting language facility such as **Add**, **Bnez** (branch if not equal to zero) and **Print**.

Arguments – Arguments may be of several different types depending on the command they are used with. Argument types include; general purpose registers, motion registers, immediate numerical values, and ASCII strings. General purpose and motion register mnemonics must exactly match the available mnemonics for that argument value. Immediately specified numerical values can be in decimal or in hexadecimal, with hexadecimal denoted by a leading "0x". ASCII strings are contained between beginning and ending double quote characters. More on general purpose registers and motion registers below.

General Purpose Registers – Some of the command arguments in the example script above have special names such as R1, R2 or Acc. These are general purpose registers used in script language functions such as counting, variable storage, or for other purposes. There are 6 user-accessible general purpose registers Acc, R1, R2, R3, R4, and R5. These will be discussed more in later sections.

Motion Registers – Motion Registers refer to motion control registers kept internally by the Magellan Motion Control IC such as the actual encoder position, the motor command output value, the servo position error, etc. Motion registers mnemonics must be an exact match to one of the available listed motion registers to be correctly recognized.

For more information on the syntax of auto-run scripts see Section 4.2, "Introduction to Auto-run and Setup Scripts."

2.4.1 Auto-run Script Units

Various Magellan Motion Control IC parameters and registers relate to different types of physical quantities. For example the **GetActualPosition** command returns the Magellan IC's instantaneous motor axis position. Other commands and registers reference quantities of time, velocity, acceleration, current, voltage, temperature, and more.

In all cases auto-run scripts utilize the N-Series ION's native units and associated scaling when setting, retrieving, and displaying (via Print) these values. The table below provides information on these units and Section 4.6, "Scaling Information for Selected Commands & Registers" along with Section 4.9.1, "Conversion Factors, Medium Power Units" provide examples of numerical scaling as well as equations for converting N-Series ION units to real-world units.

Quantity	N-Series ION Unit	Example Auto-run Script Command	Representation*
Position	Encoder counts or microsteps**	SetPosition	32-bit integer
Time	Sample Times	GetTime	32-bit integer
Velocity	Counts/Sample time	SetVelocity	32-bit integer
Acceleration	Counts/Sample time^2	SetAcceleration	32-bit integer
Jerk	Counts/Sample time^3	SetJerk	32-bit integer

Quantity	N-Series ION Unit	Example Auto-run Script Command	Representation*
Voltage	Volts	GetBusVoltage	16-bit integer
Current	Amps	SetMotorCommand	16-bit integer
Temperature	Degrees	GetTemperature	16-bit integer

**Many of these quantities require conversions to translate integer representation into real-world units. Refer to Section 4.6, "Scaling Information for Selected Commands & Registers" for more information on this.*

***When operated in microstepping mode, position units are in microsteps. When operated in servo (closed loop stepped) mode, position units are in counts.*

2.5 Auto-run Script Basics—Sets, Gets, and Copy

In the following sections we will discuss various aspects of how to use the Auto-run script system. While viewing this information two reference documents will be useful to provide additional context; the *Magellan Motion Control IC User Guide*, and the *C-Motion Magellan Programming Reference*.

We begin our detailed look at the Auto-run scripting system with how the user can specify Magellan Motion Control IC get and set commands and use them together with the **Copy** command to create useful script sequences.

Magellan Gets and Sets form the heart of how control and status fields are set, or read from. Examples of commands that Set and Get control parameters include **SetPosition**, which sets the desired target position for profiles such as trapezoidal point-to-point move, and **SetPhaseOffset**, which sets the commutation angle offset value used during phase initialization. For each of these commands there are corresponding **Get** commands to retrieve these set values. For the above commands these command mnemonics are **GetPosition** and **GetPhaseOffset**.

When run from an auto-run script get commands return their value in a special register known as the Accumulator. This 32-bit register is one of several 'general purpose' registers, the others being R1, R2, R3, R4, and R5 – more on those later. Note that some registers whose values can be requested by a get command have an integer size of 16 rather than 32 bits. These 16-bit integers will be sign-extended to 32 bits when read into the Accumulator.

Auto-run Script Example

Set the destination position to 123456 and read it back.

```
SetPosition 123456      ' Load a decimal 123456 into the destination position register
GetPosition             ' Read this just-set value into the accumulator
```

2.5.1 Sets with General Purpose Registers and the Copy Command

In the previous example the desired set command value to program was explicitly specified in the script. However one of the most powerful features of the Auto-run facility is the ability to set desired register values from the general purpose registers. These are Acc, and R1-R5.

As we illustrate additional features of the auto-run system such as the **Add** and **Sub** command (addition and subtraction) this power will become particularly evident. For the moment, however, we will illustrate the use of another command, **Copy**, using it to store the desired value to program into the R3 register and we will then set a different register with the value contained in R3.

The **Copy** command has the general format: **Copy** *Arg1 Arg2*, with *Arg2* being the 'from' data source, and *Arg1* being the 'to' destination. *Arg1* can be any of the general purpose registers, and *Arg2* can be any of the general purpose registers or an immediate value.

Auto-run Script Example

Store a destination position of 123456 into the R3 general purpose register and then use this register to set the destination position.

```
Copy R3 123456          ' This command copies a value of 123456 into the R3 register
SetPosition R3          ' Load the contents of the R3 register into the destination position
                        ' register
```

Auto-run Script Example

Copy the value of the commanded velocity register to R5.

```
GetCommandedVelocity    ' This command reads the Magellan IC's commanded velocity
                        ' register, which is used during profile generation. The result is
                        ' placed in the Accumulator
Copy R5 Acc             ' This command copies the accumulator value into the R5 register
```

2.5.2 Two-argument Set and Get Commands

In the examples above the set command had a single argument which was the value being specified for the register. Many set commands have two arguments however.

An example of such as command is **SetFOC**. The **SetFOC** command expects the first argument to be a register identifier, and the second argument to be the value to program that register to. The specific numerical codes for each register identifier can be determined from the *C-Motion Magellan Programmers Reference* (as can all available Magellan command mnemonics).

Auto-run Script Example

Set the value of the FOC Direct loop integrator gain parameter to 1234 and read it back.

```
SetFOC 1 1234           ' Load a decimal 1234 into the FOC direct integrator gain register
                        ' 1 is the ID # for this register within the SetFOC command
GetFOC 1                ' Read this just-set value into the accumulator
```

2.6 Mathematical/logical Operations

Auto-run scripts support a small but powerful set of arithmetic and boolean operations. In combination with the general purpose registers (Acc, R1-R5) they can be used to create a wide variety of programming language constructs.

The table below shows the mathematical and logical operations that are supported along with the # of arguments each such command takes:

Command Mnemonics	Number of Arguments	Results Stored In	Description
Add	2	Accumulator	Calculate arg1 + arg2
Sub	2	Accumulator	Calculate arg1 - arg2
And	2	Accumulator	Calculate bitwise and of arg1 and arg2
Abs	1	Accumulator	Calculate absolute value of arg1

Arguments for the commands that take two arguments are either both general purpose registers or one general purpose register and one immediate specified value. For the **Abs** command, which takes one argument, both general purpose registers and immediates can be arguments.

All of these commands perform 32-bit twos-complement signed integer operations. Care should therefore be taken if unsigned values are used in these calculations as unexpected results may occur.

2.6.1 Add Operations

The form of the **Add** command is: **Add** *Arg1 Arg2*. The **Add** operator performs a signed 32-bit addition of two arguments and stores the result into the accumulator. Here are examples of script code using **Add** operations:

Auto-run Script Example

Add the contents of the accumulator to R4 and put result in accumulator.

```
Add Acc R4          ' The content of the accumulator is added to the content of
                    ' R4 the result is stored in the accumulator
```

Auto-run Script Example

Add 10000 to the contents of the destination position register and store back into the destination position register.

```
GetPosition         ' Load the Acc with the content of the destination position register
Add Acc, 10000      ' Add 10,000 to the value in the accumulator
SetPosition Acc     ' Set the destination position register to the value in the accumulator
```

2.6.2 Sub Operations

The form of the **Sub** command is: **Sub** *Arg1 Arg2*, and performing the operation *Arg1 – Arg2*. Therefore unlike for the addition operation the location of the argument in the command expression matters to the end result, which is placed in the accumulator.

Auto-run Script Example

Subtract the contents of the accumulator from R4 and put result in accumulator.

```
Sub R4 Acc          ' The content of the accumulator is subtracted from the content
                    ' of R4 and the result is stored in the accumulator
```

Auto-run Script Example

Subtract 1000 from the contents of R2.

```
Sub R2 1000         ' 1,000 is subtracted from the content of R2 and the result is
                    ' stored in the accumulator. For example if R2 held the value 500,
                    ' after this operation the accumulator would hold -500
```

Auto-run Script Example

Subtract the contents of R2 from 1000.

```
Sub 1000 R2         ' The content of R2 is subtracted from 1,000 and the result is
                    ' stored in the accumulator. For example if R2 held the value 500,
                    ' after this operation the accumulator would hold +500
```

Auto-run Script Example

Subtract the contents of the Position Error register from the contents of the Commanded Position register and store the result into the Destination Position register.

```
GetPositionError          ' Load the Acc with the content of the servo position error register
Copy R1 Acc               ' Save this value in the R1 register
GetCommandedPosition      ' Load the accumulator with the content of the commanded position
Sub Acc R1                ' Subtract the content of R1 from the accumulator
SetPosition Acc           ' Store the value of the resulting computation into the destination
                          ' position register
```

2.6.3 And Operations

The form of the **And** command is: **And** *Arg1 Arg2*, with a bitwise and being performed between each bit of each argument, and the result being stored in the accumulator.

And operations are especially useful for testing the condition of specific bit-oriented status words or specific external signals. In the Magellan system such registers include the Event Status Register, the Activity Status Register, the Drive Exception Register, and the Signal Status Register. By combining with conditional branch statements this allows constructs such as "wait until motion has completed" or "branch to a special code sequence if the AxisIn bit goes low".

Another use of the **And** command is to clip the high 16-bits of a 32-bit word whose contents were sign-extended from a 16-bit word. anding such a 32-bit word with a value 0xffff guarantees that the 32-bit result will have zero in the high 16 bits and the original 16-bit word value in the low 16 bits.

Auto-run Script Example

And the contents of R2 with the value 0xffff (65,535 decimal).

```
And R2 0xffff             ' The content of R2 is anded with the hex value 0xffff. The result
                          ' is stored in the accumulator.
```

Auto-run Script Example

Read the value of the Event Status word and loop back if the positive limit switch has not been encountered. If it has, continue code execution in line.

```
PosLimLoop:               ' This is a special construct that will be described later when
                          ' branching is discussed. This entry assigns a tag of 'PosLimLoop'
                          ' to this location in the auto-run script
    GetEventStatus        ' Load the content of the Event Status Register into the accumulator
    And Acc 0x20          ' And the event status word with a single bit mask corresponding
                          ' to the positive limit switch event bit.
    Bz Acc PosLimLoop     ' If the content of Acc is zero this means the positive limit switch
                          ' condition is inactive, which means auto-run script code execution
                          ' will loop back to the code location marked 'PosLimLoop' to
                          ' retest the condition
    GetActualPosition     ' This example code entry and subsequent script commands represent
                          ' code that processes a positive limit switch event
```

2.6.4 Abs Operation

The form of the **Abs** command is: **Abs** *Arg1*. This single-argument command places the result in the accumulator.

Auto-run Script Example

Place the magnitude (absolute value) of the current trajectory velocity into the R3 register.

```
GetCommandedVelocity    ' Load the current value of the trajectory generator velocity register
                        ' into the accumulator
Abs Acc                 ' Take the absolute value
Copy R3 Acc             ' Store this value into R3
```

2.7 Branching

Branching is a vital feature of auto-run scripts that allows instruction flow to be conditionally altered so that the next instruction processed may jump to a different part of the auto-run script rather than continue processing in line. Particularly in combination with a conditional branch, this enables a wide variety of programming language functions including looping, special conditions handling, external command processing, and more. We will give examples of some of these important functions later on in this section.

The general form of branch commands is: BranchMnemonic *arg1 arg2 AddressLabel*. BranchMnemonic is one of ten available branch instructions listed in the table below. *Arg1* and *arg2* are used by some but not all branch commands. If used, they consist of general purpose registers. *AddressLabel* must be a mnemonic that matches a separately declared location label in the program script.

Pro-Motion uses *AddressLabel* to calculate relative difference in program address between the branch command and the *AddressLabel*. This difference can be positive or negative, meaning the branch can jump forward or backward relative to the branch instruction location. *AddressLabels* end in a ":" character and can include alpha characters (A-Z), numbers (0-9), and the underscore character anywhere in the label.

The table below shows the available branch commands. These commands are different only in the condition by which a branch occurs and correspondingly in the arguments used to determine whether the branch condition is satisfied.

Command Mnemonic	# Args	Description
B	0	Branch unconditionally
Bz	1	branch if arg1 is zero
Bnz	1	branch if arg1 is non zero
Bneq	2	branch if arg1 and arg2 are not equal to each other
Beq	2	branch if args1 and arg2 are equal to each other
Bgt	2	branch if arg1 greater than arg2
Bgeq	2	branch if arg1 greater than or equal to arg2
Blt	2	branch if arg1 less than arg2
Bleq	2	branch if arg1 less than or qual to arg 2
Bdnz	1	decrement arg1 and branch if resultant value is not zero

Below are script code examples showing how to use branches.

Auto-run Script Example

Change the program flow unconditionally to another section of code.

```
    B CodeSectionB          ' Branch unconditionally to the code section labeled 'CodeSectionB'
    …                       ' other code located here

CodeSectionB:
…                           ' CodeSectionB code located here
```

Auto-run Script Example

Branch to another code section if the value of the accumulator is zero.

```
    Bz Acc CodeToJumpTo     ' If the value of the accumulator is zero branch to the code location
                            ' labeled 'CodeToJumpTo'
    …                       ' Code that executes when accumulator value is not zero located here

CodeToJumpTo:
…                           ' CodeToJumpTo' code located here
```

Auto-run Script Example

Branch to another code section if the value of R3 is greater than 10.

```
    Copy Acc 10             'Load Acc with compare value of 10
    Bgt R3 Acc ProcessingCode   ' If the value of R3 is greater than the value of the accumulator
                            ' branch to the code location labeled 'ProcessingCode'
    …                       ' Code that executes when R3 is less than or equal to the accumulator

ProcessingCode:
…                           'ProcessingCode' code located here
```

Auto-run Script Example

Branch to another code section if the value of R3 is not equal to the value of the Accumulator.

```
    Bneq R3 Acc ProcessingCode  ' If the value of R3 is not equal to the value of the accumulator branch to the
                            ' code labeled 'ProcessingCode'
    …                       ' Code that executes when R3 is equal to the accumulator location

ProcessingCode:
…                           'ProcessingCode' code located here
```

2.7.1 Bdnz Command

The **Bdnz** is a special branch function almost always used in connection with program loops. Program loops are repeating sequences of code, executed a particular number of times. Program loops are a common and very useful programming construct and allow functions such as "Execute this motion profile 100 times".

The form of the **Bdnz** command is the same as other branch instructions which take a single argument: **Bdnz** *Arg1 AddressOffset*.

Auto-run Script Example

Toggle the AxisOut signal high and then low ten times.

```
        SetSignalSense 0x400        ' Establish starting output signal state of AxisOut signal high
        SetAxisOutMask 0 0 0        ' Set up AxisOut signal for manual control
        Copy R3, 10                 ' Initialize R3 with loop count value of 10

BitOutputLoop:                      ' Mark location of start of loop
        SetSignalSense 0x400        ' Set AxisOut signal state to high
        SetSignalSense 0            ' Set AxisOut signal state to low
        Bdnz R3 BitOutputLoop       ' Decrement R3, if not zero loop back and continue
```

2.8 Printing

The Auto-run script system supports the ability to output strings and register values. This may be useful for a number of reasons including helping with script development, outputting specific motion control register values for the purpose of data collection, indicating what function the script is executing at any given time, or any number of other reasons.

Print output is viewed in a separate window that is created automatically by the Pro-Motion Auto-run Control dialog box. When a running script executes a print message it appears in this scrolling window whose sole purpose is to display print messages output from the executing script.

The contents of the print output window can be saved for later analysis, or for incorporation into spreadsheets or other data analysis software. Right-click on the content of the print output window to view options for achieving this. Note that for most such analysis/display software programs the print output must follow a specific format to be correctly imported. For example tabs to delimit one output value from another. Print command examples below show how this can be done.

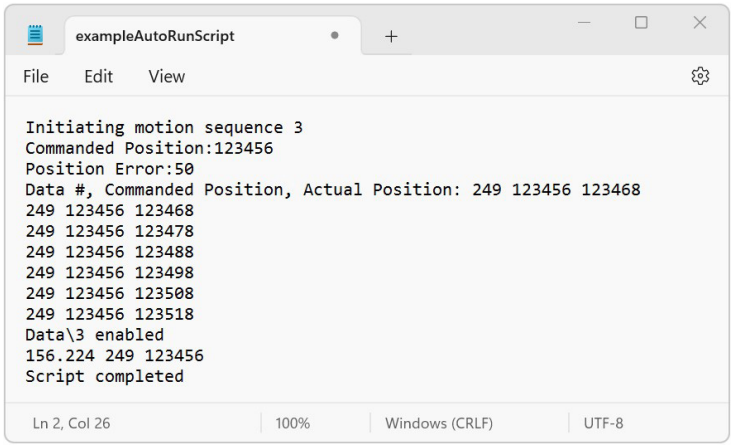


Figure 2-2: Sample Print Output After Saving To Notepad

The auto-run script command used for output is **Print**, and the general format is **Print** *Arg1 Arg2*, with *Arg1* being a string in double quotes, and *arg2* being one of the general purpose registers or a Motion register. Both *Arg1* and *Arg2* are optional, meaning just a string can be output with no numerical register value specified, or just register output with no associated string.

To specify numerical display of a register value mnemonics identifying the register are used. The general purpose register mnemonics are Acc and R1-R5. A list of available internal motion IC registers is shown in Section 2.10, "List of Printable Motion Registers." This list does not include every possible Magellan register, but includes many of the most useful including ActualPosition, PositionError, MotorCommand, and others.

2.8.1 Print Formatting

The **Print** command supports output format control via special character sequences in the ASCII string. These sequences are detailed below:

Newline Control

If the two character sequence "\n" is included at the very end of the ASCII string a new line will be inserted after the print command is processed, meaning the output from any subsequent **Print** commands will appear at the start of the next line. This is true whether a register is specified in the print statement or not. If a register is specified then the newline will be inserted after the register value is output.

Conversely, if there is no "\n" at the end of the ASCII string then the output of a subsequent **Print** will appear immediately after, on the same line, as the previous **Print**.

Finally, if no ASCII string is provided and just a register is specified to be printed the register value name is displayed followed by " = " followed by the numerical value of the register.

Tabs

One or more tab characters can be inserted in the print output via the character sequence "\t". Tabs are useful to align print output fields for improved readability, or sometimes as data delimiters for data import to graphing or analysis programs

Backslash character

A final special character sequence is "\\", which simply allows the backslash to be output as a character in the print output. Without this special sequence the auto-run interpreter would interpret any "\" character as the beginning of a special control sequence rather than a print output of the \ character.

Auto-run Script Example

Print the message "Initiating motion sequence 3" on a separate output line.

```
Print "Initiating motion sequence 3\n"
Print "Next line of output\n"
```

Resulting Output:

```
Initiating motion sequence 3
Next line of output
```

Note that in the above example if the first **Print** command (**Print** "Initiating motion sequence 3\n") had not contained "\n" at the end of the string the output would instead look as follows:

```
Initiating motion sequence 3Next line of output
```

Auto-run Script Example

Print the content of the Commanded Position and on the next line the Position Error, labelling these values "Latest Commanded Position:" and "Resultant Position Error:" respectively. The value of these registers are 123456 and 50 respectively.

```
Print "Latest Commanded Position: \n" CommandedPosition
Print "Resultant Position Error: \n" PositionError
```

Resulting Output:

```
Latest Commanded Position: 123456
Resultant Position Error: 50
```

Note that in the above examples a space is inserted after the colon in the **Print** command string. If these were not included the output would look like:

```
Commanded Position:123456
Position Error:50
```

Auto-run Script Example

Rewrite the above script to output just the register name using an equals sign rather than a colon.

```
Print CommandedPosition        'When no ASCII string and only a register is specified for
                               ' printing output is preformatted to print the register name along
                               ' with the value separated by " = ".
Print PositionError
```

Resulting Output:

```
Commanded Position = 123456
Position Error = 50
```

Auto-run Script Example

Print the value of the R1 register, the Commanded Position, and the Actual Position on the same line with each printed number separated by a space. Label the line as Data #, Commanded Position, Actual Position. The values of these three registers are 249, 123456, and 123468 respectively.

```
Print "Date #, Commanded Position, Actual Position: " R1
Print " " CommandedPosition
Print " \n" ActualPosition
```

Resulting Output:

```
Data #, Commanded Position, Actual Position: 249 123456 123468
```

Note that spaces are needed at the beginning of the second two prints to separate the numerical output values with spaces. Had these not been included the output would be like:

```
Data #, Commanded Position, Actual Position: 249123456123468
```

In the above sequence subsequent **Print** commands would output on the next line owing to the inclusion of the \n at the very end of the last **Print** command string.

Auto-run Script Example

Reprint the above example eliminating the label at the beginning of each line entry.

```
Print "" R1                    ' ASCII string specified with no content
Print " " CommandedPosition
Print " \n" ActualPosition
```

Resulting Output:

```
249 123456 123468
```

Auto-run Script Example

Reprint the above example but replacing space delimiters for the data outputs with tab characters.

```
Print "" R1
Print "\t" CommandedPosition
Print "\t\n" ActualPosition
```

Resulting Output:

```
249 123456 123468
```

Auto-run Script Example

Print the Message "Data\3 enabled".

```
Print "Data\\3 enabled\n"          ' \\ sequence indicates "\" character to be output
```

Resulting Output:

```
Data\3 enabled
```

2.9 Profile Execution

Executing motion profiles is ultimately the primary purpose of the N-Series ION. The available trajectory generation modes offered by the Magellan IC include Trapezoidal point-to-point, S-curve point-to-point, Velocity Contouring, Electronic Gearing, and for some versions of N-Series IONs a User Defined Profile Mode (UDPM) which can execute arbitrary user-defined CAM profiles as well as multi-axis synchronized continuous contouring profiles.

In this section we will introduce some key scripting concepts associated with two of the most popular profile modes; Trapezoidal and S-curve point-to-point. For additional information on related commands, registers, as well as a complete theory of operations refer to the *Magellan Motion Control IC User Guide.*

The basic mechanism by which Magellan trajectory profiles are used is to first specify the desired profile parameters, and to then send an **Update** command which transfers the previously loaded profile parameters to the active profile registers at which time trajectory generation begins and motion starts.

Although script code execution will continue after the **Update** command has been sent, most applications will want to wait for the profile to be complete before continuing script execution. To accomplish this we will use conditional branching to continuously check the motion complete bit of the Magellan's Axis Status Register and only continue when it is set.

Here are some example Auto-run script sequences to set up and execute profile moves

2.9.1 Trapezoidal Profile Script Code

Auto-run Script Example

Set up and run a trapezoidal profile with a destination position of 123456, a maximum velocity of 55555, an acceleration value of 1000000, and a deceleration value of 2000000. Start the move and wait till the move has been completed.

```
        SetProfileMode 0            ' Specify that this will be a trapezoidal (code 0) profile
        SetPosition 123456
        SetVelocity 55555
        SetAcceleration 1000000
        SetDeceleration 2000000
        ResetEventStatus 0xfffe     ' Clear motion complete bit
        Update                      ' Start the move

CheckComplete:                      ' Define loop back location address
        GetEventStatus              ' Load the content of the Event Status Register into the accumulator
        And Acc 1                   ' And the event status word with a single bit mask
                                    ' corresponding to the motion complete bit
```

```
        Bz Acc CheckComplete        ' Loop back if motion complete bit is not set (meaning it has a
                                    ' value of zero)
```

2.9.2 Back And Forth S-Curve Profile Shuttle Script Code

Auto-run Script Example

Set up and run an S-curve profile shuttle (back and forth motion) and run this shuttle 25000 times. The outgoing destination position is 123456, the return destination position is 1000, the velocity for both moves is 55555, the acceleration and deceleration values are both 1000000, and the jerk value is 950.

```
        SetProfileMode 2            ' Specify that this will be an s-curve (code 2) profile
        SetVelocity 55555           ' Set up the s-curve profile parameters
        SetAcceleration 1000000
        SetDeceleration 1000000
        SetJerk 950
        Copy R3 25000               ' Initialize R3 with loop count value of 25000

LoopStart:                          ' Label loop back location for shuttle loop counter
        ResetEventStatus 0xfffe     ' Clear motion complete bit
        SetPosition 123456          ' Set outgoing destination position
        Update                      ' Start the move

OutgoingComplete:
        GetEventStatus              ' Load the content of the Event Status Register into the accumulator
        And Acc 1                   ' And the event status word with a single bit mask corresponding
                                    ' to the motion complete bit
        Bz Acc OutgoingComplete     ' Loop back if motion complete bit is not set (meaning it has a
                                    ' value of zero)
        ResetEventStatus 0xfffe     ' Clear motion complete bit
        SetPosition 1000            ' Set return destination position
        Update                      ' Start the move. Note that we do not need to reload
                                    ' unchanged profile settings such as profile type,
                                    ' velocity, acceleration, etc…
ReturnComplete:
        GetEventStatus              ' Load the content of the Event Status Register into the accumulator
        And Acc 1                   ' And the event status word with a single bit mask corresponding
                                    ' to the motion complete bit
        Bz Acc ReturnComplete       ' Loop back if motion complete bit is not set (meaning it has a
                                    ' value of zero)
        Bdnz R3 LoopStart           ' Decrement R3, if not zero loop back and continue
```

2.9.3 Repeating S-Curve Profile Script Code

Auto-run Script Example

Set up and run a series of S-curve profile moves which, after completing the previous move sets the destination position of the next move to a destination position 10000 counts greater. The velocity, acceleration, deceleration, and jerk value for all moves is 55555, 1000000, and 9999999 respectively. Execute 100 such moves.

```
        SetProfileMode 2            ' Specify that this will be an s-curve (code 2) profile
        SetVelocity 55555           ' Set up the s-curve profile parameters
        SetAcceleration 1000000
```

```
        SetDeceleration 1000000
        SetJerk 9999999
        Copy R4 100                 ' Initialize R4 with loop count value of 100

LoopStart:                          ' Label loop back location for shuttle loop counter
        ResetEventStatus 0xfffe     ' Clear motion complete bit
        GetPosition                 ' Load present destination position into the accumulator
        Add Acc 10000               ' Increase by 10000
        SetPosition Acc             ' Store updated destination position
        Update                      ' Start the move

CheckIfComplete:
        GetEventStatus              ' Load the content of the Event Status Register into the accumulator
        And Acc 1                   ' And the event status word with a single bit mask corresponding
                                    ' to the motion complete bit
        Bz Acc CheckIfComplete      ' Loop back if motion complete bit is not set (meaning it has a
                                    ' value of zero)
        Bdnz R4 LoopStart           ' Decrement R4, if not zero loop back and continue
```

2.10 List of Printable Motion Registers

The following table shows the Magellan motion registers available for printing, along with related information such as units. For additional information regarding these registers refer to the *Magellan Motion Control IC User Guide*.

Mnemonic	Unit	Description
Encoder & Trajectory Registers		
ActualPosition	Counts*	If an encoder is used contains the current axis position measured by the encoder. If
ActualVelocity	Counts/sample time	If an encoder is used contains the current axis velocity measured by the encoder. If
CommandedPosition	Counts	Contains the instantaneous position command of the trajectory generator
PositionError	Counts	For motors operated via position servo control (which includes step motors using closed loop control) contains the difference between the commanded and the actual axis position
CommandedVelocity	Counts/sample time	Contains the instantaneous velocity command of the trajectory generator
CommandedAcceleration	Counts/sample time^2	Contains the instantaneous acceleration command of the trajectory generator
Status Registers		
EventStatusRegister	n/a	Contains the bit-oriented Magellan Event Status Register
ActivityStatusRegister	n/a	Contains the bit-oriented Magellan Activity Status Register
SignalStatusRegister	n/a	Contains the bit-oriented Magellan Signal Status Register
DriveStatus	n/a	Contains the bit-oriented Magellan Drive Status Register
DriveFaultStatus	n/a	Contains the bit-oriented Magellan Drive Fault Status Register
Commutation		
PhaseOffset	Counts	Contains the modulus of the encoder position at the index location
PhaseAngleScaled	Electrical revolutions	

Mnemonic	Unit	Description
Current Control Loop		
ActiveMotorCommand	Amps	Motor Output Command from microstepping generator or for position PID servo loop
DReference	n/a	Field Oriented Control D-loop command value
DFeedback	n/a	Field Oriented Control D-loop measured value
DError	n/a	Field Oriented Control D-loop servo difference
DOutput	n/a	Field Oriented Control D-loop servo output value
QReference	n/a	Field Oriented Control Q-loop command value
QFeedback	n/a	Field Oriented Control Q-loop measured value
QError	n/a	Field Oriented Control Q-loop servo difference
QOutput	n/a	Field Oriented Control Q-loop servo output value
Safety/Amplifier		
BusVoltage	Volts	Measured HV input voltage
Temperature	Degrees	Measured internal amplifier temperature
I2tEnergy	$AmpTime^2$	I2t Energy
BusCurrentReturn	Amps	Measured total return current
Miscellaneous		
MotionICTime	Sample times	Magellan system time

* *References to counts in this table can also be microsteps if microstepping control mode is used.*

2.11 Auto-Run Language Command Reference

The sections below provides a list of auto-run script language commands. For a detailed list of Magellan commands refer to the *C-Motion Magellan Programming Reference*.

2.11.1 Copy, Mathematical, and Logical Commands

Command Format	Arg1	Arg2	Result Stored In	Description
Abs *arg1*	Acc, R1-R5 Immediate		Acc	result - absolute(arg1)
Add *arg1 arg2*	Acc, R1-R5 Immediate*	Acc, R1-R5 Immediate	Acc	Result = arg1 + arg2
And *arg1 arg2*	Acc, R1-R5 Immediate*	Acc, R1-R5 Immediate	Acc	Result = bitwise and of arg1 and arg2
Copy *arg1 arg2*	Acc, R1-R5	Acc, R1-R5 Immediate	arg1	Copy arg2 to arg1
Sub *arg1 arg2*	Acc, R1-R5 Immediate*	Acc, R1-R5 Immediate	Acc	Result = arg1 - arg2

**Either arg1 or arg2 may be immediates, but not both*

For more information on these instructions refer to Section 2.5, "Auto-run Script Basics—Sets, Gets, and Copy" and Section 2.6, "Mathematical/logical Operations."

2.11.2 Branch Commands

Command Format	Arg1	Arg2	Description
B *AddressLabel*			Branch unconditionally
Bz *arg1 AddressLabel*	Acc, R1-R5		Branch if arg1 is zero
Bnz *arg1 AddressLabel*	Acc, R1-R5		Branch if arg1 is not zero
Beq *arg1 arg2 AddressLabel*	Acc, R1-R5	Acc, R1-R5	Branch if arg1 and arg2 are equal to each other
Bneq *arg1 arg2 AddressLabel*	Acc, R1-R5	Acc, R1-R5	Branch if arg1 is arg2 are not equal to each other
Bgt *arg1 arg2 AddressLabel*	Acc, R1-R5	Acc, R1-R5	Branch if arg1 is greater than arg2
Bgeq *arg1 arg2 AddressLabel*	Acc, R1-R5	Acc, R1-R5	Branch if arg1 is greater than or equal to arg2
Blt *arg1 arg2 AddressLabel*	Acc, R1-R5	Acc, R1-R5	Branch if arg1 is less than arg2
Bleq *arg1 arg2 AddressLabel*	Acc, R1-R5	Acc, R1-R5	Branch if arg1 is less than or equal to arg2
Bleq *arg1 arg2 AddressLabel*	Acc, R1-R5	Acc, R1-R5	Branch if arg1 is less than or equal to arg2
Bdnz *arg1 AddressLabel*	Acc, R1-R5		Decrement arg1 and branch if result is not zero

For more information on branch instructions refer to Section 2.7, "Branching."

2.11.3 Print Command

Command Format	Arg1*	Arg2*	Description
Print <*arg1*> <*arg2*>	ASCII string	Acc, R1-R5 Motion Register	arg1 is optional string in quotes. Special control characters are \t, \b, and \\ arg2 is optional register identifier

** Both arg1 and arg2 are optional but at least one argument must be specified*

For more information on the Print command refer to Section 2.8, "Printing." For a list of printable motion registers refer to Section 2.10, "List of Printable Motion Registers."

3. Connecting via Host Interfaces

In This Chapter

- Activating Serial Communications
- Shunt Operation
- C-Motion Engine User Code Development

In the previous chapter's 'getting started' description the PC was communicating to the N-Series ION via its programming port using the 3-pin programming cable. However there will likely be times when you want to connect Pro-Motion to the N-Series ION via its native host interface, either CAN FD, serial, Ethernet, or SPI. Accomplishing this is described in the following sections.

3.1 Activating Serial Communications

N-Series IONs of serial host interface type support RS232, RS422, and RS485 communications. For detailed information on N-Series ION's serial interfaces refer to the *ION/CME N-Series Digital Drive User Manual.*

In the following sections we will detail how to connect from a PC to the N-Series ION's serial connections using two different connection schemes; RS232 and RS485.

3.1.1 RS232 Hardware Setup

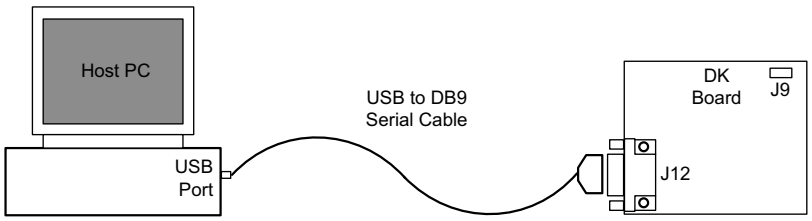


Figure 3-1: Hardware Setup for Communications via RS232

For your PC to communicate via RS232 you will use the USB to DB9 serial converter cable (PMD P/N Cable-USB-DB9) which is included with serial host N-Series ION DKs. As shown in Figure 3-1 the DK interconnect board directly accepts this cable at its J12 connector. Note that you should plug this cable into a USB port on your PC different than the one already being used for the USB to 3-pin programming port cable that you used for setup and first time verification.

In addition, the N-Series ION should be set to RS232 rather than RS485 communication. With a factory default N-Series ION unit connected as shown in the figure above RS232 is automatically selected. If desired however, this setting can be set and stored in the N-Series ION's NVRAM. See Section 3.1.6, "Setting Up the ION for RS485 Communications" for more information.

Although in this RS232 setup description we will only use Serial1, serial host N-Series IONs provide two RS232 interfaces, Serial1 and Serial2. PMD has available in its accessories library a cable that allows access to both of the serial

ports. To order this cable contact your local PMD representative. For convenience the two cables mentioned above are listed in the table below:

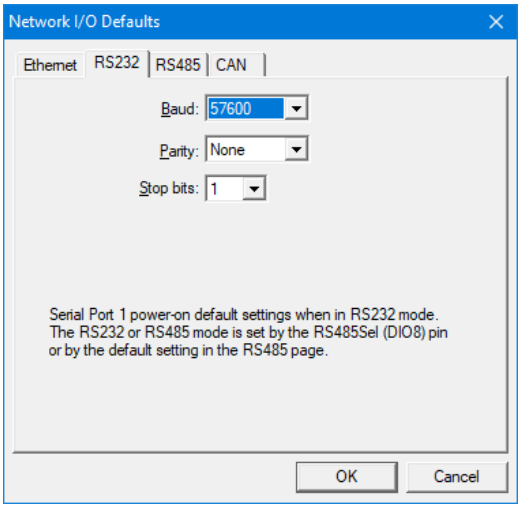
Vendor	P/N	Description	Included with Serial DK
PMD	Cable-USB-DB9	Male DB9 to USB serial converted cable.	Yes
PMD	Cable-4355-01.R	Male DB9 dual serial to two single-channel female DB9 cable.	No

For detailed wiring information on both of these cables refer to Section 4.8, "Selected Cable & Accessory Specifications."

3.1.2 Setting Up the ION for RS232 Communications

Here is the Pro-Motion sequence used to set up the N-Series ION to connect via its Serial1 RS232 host interface.

1. With serial communications via the 3-pin programming cable functioning properly, click the Device toolbar button. The Device window appears.
2. Click Network I/O. The Network I/O Defaults dialog box appears.
3. Click the Serial tab. This window appears with data entry fields for the default serial port setting such as the baud rate. This is shown below with default values visible.



4. Enter the desired baud in the corresponding data fields.
5. Click OK to store as the power on default.
6. Click the Reset toolbar button. After the ION is reset, it uses the default parameters that you specified.

The N-Series ION has now been programmed to be ready for RS232 communications.

3.1.3 Setting Up Pro-Motion for RS232 Communications

To setup Pro-Motion to communicate by RS232:

1. Click the Connect toolbar button.
2. Select RS232 and then click OK.
3. Enter the same serial port settings as were programmed into the ION previously.
4. When complete, click OK.

If RS232 communication is successful, an additional set of graphical icons representing your N-Series ION and axis will be loaded into the Project window. If communication is not successful, after about 2 seconds, a Communications Timeout Error dialog box appears. If this happens, recheck your connections, and retry to establish RS232 communications.

A useful optimization of the serial connection may be to reduce the latency between message packet sends. If this was already done for your PC while communicating using the 3-pin programming port then this optimization will automatically apply for the RS232 USB to 9-pin cable connection. If not, the next paragraph describes how this can be done:

First, open the Windows Device Manager by typing "Device Manager" in the Windows search box. Next, find Ports (COM & LPT) from the list and click on it, then right-click the USB Serial Port (COMn) of the serial port you are using and then select Properties. Click on the Port Settings tab and then select the Advanced button. Change the field for Latency Timer (msec) to a value of 1. Click OK on both windows and close the Device Manager. The driver is now optimized for use with the N-Series ION drive.

When RS232 communications are functioning properly, the final step is to disable serial communications to the ION's Serial3 programming port connection

3.1.4 Disconnecting Serial3 Programming Port Communications

The last step is to disconnect the programming port serial connection. To accomplish this:

1. Select the serial link version of the ION in the Project window.
2. Click the Disconnect toolbar button. A dialog box appears asking if you are sure you want to disconnect.
3. Click OK. You will notice that the serial ION icon and axes graphical icons in the Project box disappear, leaving only the Serial1 RS232 link icons for the ION and axis.

RS232 communication setup is now complete. You are ready to execute all Pro-Motion functions via RS232.

3.1.5 RS485 Hardware Setup

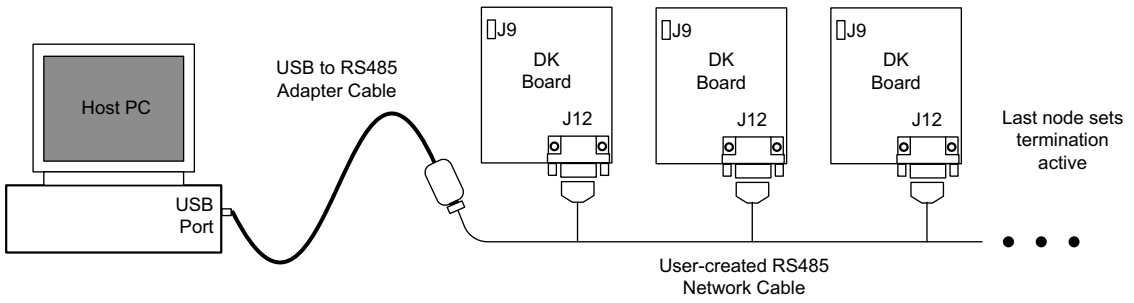


Figure 3-2: Hardware Setup for Communications via RS485

In addition to RS232 communications at Serial1 and Serial2, N-Series IONs support RS422 and RS485 communications via the Serial1 interface. Communicating by RS485 is typically less of a 'plug and play' process than communicating by RS232 however, and requires the user to create their own cables.

RS422 is essentially a subset of RS485 which uses a similar differential electrical interface but has only a single node connection. We will therefore focus on RS485 and show how to wire a network of RS485-connected N-Series IONs.

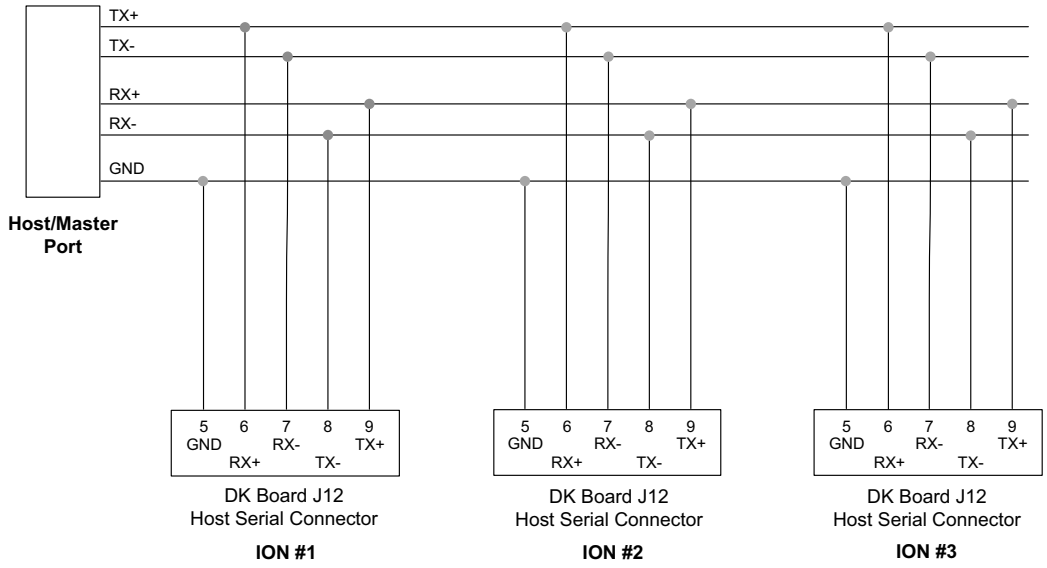


Figure 3-3: RS485 Signal Connection Diagram

RS485 is a master slave system. When connecting to a PC running Pro-Motion the PC functions as the master, and each N-Series ION in the network functions as a slave. Figure 3-3 shows the basic wiring scheme that is used for a full duplex (four-wire) connection. Four wire is recommended over two-wire because it has higher reliability. Additional details such as what wire type to use, wire shielding, twisting differential wire pairs, length of wire etc. will not be detailed here but are important so a reference text on RS485 networks should be consulted.

For reliable communications the last unit on both sides of the network cable should provide termination. For most systems 120 ohms of termination resistance is recommended. N-Series IONs provide user programmable internal termination of 120 ohm.

For additional information on serial communication functions with N-Series IONs refer to the *ION/CME N-Series Digital Drive User Manual.*

To prepare the PC to function as the RS485 master an RS422/485 USB adapter or RS485 card should be purchased and installed. Follow the product directions to install the software drivers. If properly installed and visible to Windows,

Pro-Motion should recognize the RS485 port and be able to communicate with it automatically. For convenience the following table lists a vendor and P/N for such an adaptor product:

Vendor	P/N	Description
Advantech	BB-USOPTL4	Isolated high retention USB to RS422/485 converter (USB cable included)

3.1.6 Setting Up the ION for RS485 Communications

Each ION unit to be installed in the RS485 network needs to be programmed with network settings. If the IONs are mounted on the DK interconnect board, these settings can be programmed using Pro-Motion communicating to the ION via the 3-pin programming port.

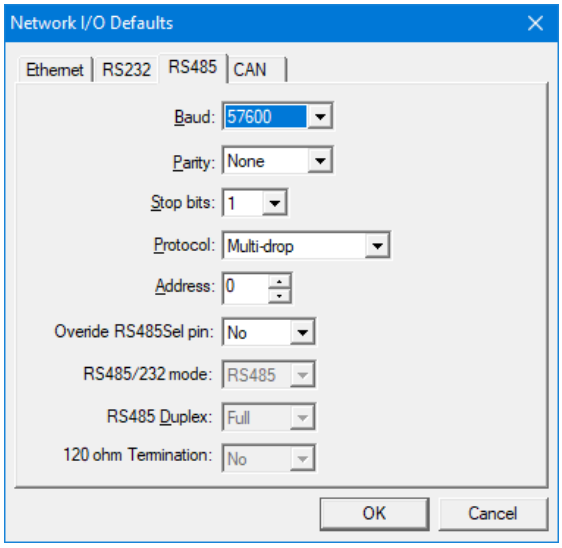
Each successive ION will be programmed, powered down, and then the 3-pin cable is detached and attached to the next unit to be programmed. After the final unit is programmed the 3-pin programming cable should be detached.

The table below provides a summary of the available serial network settings as well as recommended values for creating the RS485 network:

Parameter	Default Setting	Setting Options	Comment
Override RS485Sel pin	no	no yes	If this field is set to "no" the value of pin 9 of the serial N-Series ION's Signal Connector (which can be accessed on the DK board via Pin 9 of J7, the Indexer Connector), will be read to select RS232 or RS485 (high = RS232, low = RS485). If this field is set to "yes" the pin value will be ignored and the RS232/ RS485 setting will be read from the setting of the next field.
RS232 or RS485	RS232	RS232 RS485	See above. This setting is only used if the 'Override RS485Sel pin' is programmed to "yes".
Node ID	0	0-31	Each ION unit should be programmed with a different node address.
Duplex (full/half)	full duplex	half duplex full duplex	If using a four wire connection scheme full duplex should be selected. Two wire schemes should select half duplex.
Termination active	no	application specific	The last ION in the network should have termination active. All other units should have termination inactive.

Once each N-Series ION unit has been programmed and wired into the network the system is ready to operate.

3.1.7 Setting Up Pro-Motion for RS485 Communications



With a RS422/RS485 USB Adapter or other RS485 network card installed in the PC and drivers loaded, the instructions provided in Section 3.1.3, "Setting Up Pro-Motion for RS232 Communications" can be followed to configure the PC to talk to the RS485 network IONs, except that the RS485 tab rather than the RS232 tab should be selected. The programmed baud rate should match the baud rate entered into the N-Series IONs, the protocol should be programmed to multi-drop, and the correct address/node ID should be entered.

With the full network powered on, If RS485 communication is successful, an additional set of graphical icons representing each of the N-Series IONs on the network will be loaded into the Project window. If communication is not successful, after about 2 seconds a Communications Timeout Error dialog box appears. If this happens, recheck your connections, and retry to establish communications with the ION units.

If RS485 communications are functioning properly, the last step is to disconnect the Serial3 programming port. Instructions to do this can be found at Section 3.1.4, "Disconnecting Serial3 Programming Port Communications."

RS485 communication setup is now complete and you are ready to execute all Pro-Motion and N-Series ION functions via RS485 network.

3.2 Shunt Operation

N-Series IONs provide the ability to drive a shunt resistor via their Motor D output. Shunt control is a technique for managing the DC Bus voltage, particularly in situations where the motor axis is decelerating and acting as a generator, thereby driving the DC bus voltage higher.

Shunt control is an option for use with DC Brush and Brushless DC motors. It is not used with step motors. Whether or not it should be used depends on a number of factors including the design of the DC power supply, the motor inertia, and the worst case expected motor deceleration rate. But to some extent if during application development a trace of the DC Bus supply shows an undesired rise in HV voltage, shunt can be consider as a remedy.

As shown in Figure 3-4 the N-Series ION provides a shunt PWM output at the Motor D output pin on the unit's Power Connector, which is at the DK PCB at Connector J2, Pin 4. A connection to HV is made at the DK Connector J1, Pin 2.

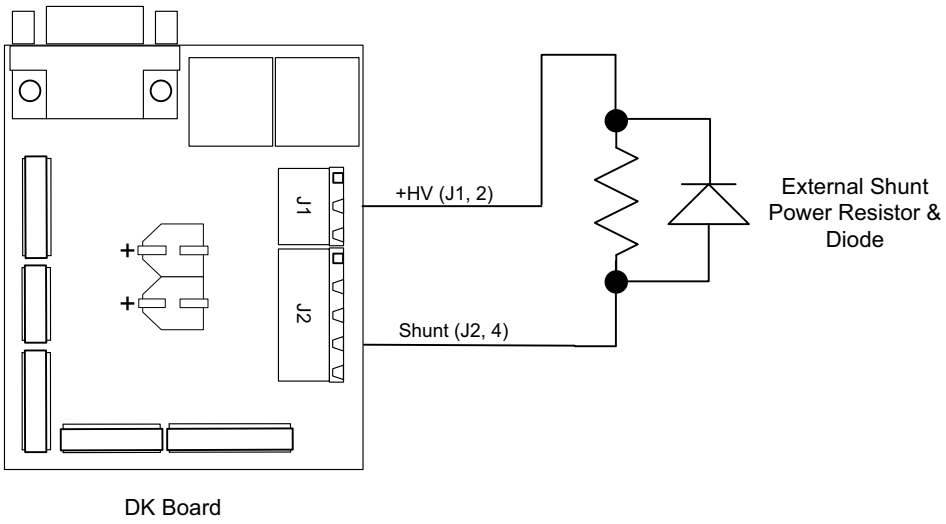


Figure 3-4: Shunt Resistor Wiring Diagram

The shunt functions by continually comparing the DC bus voltage to a user programmable threshold. If the DC bus voltage exceeds the comparison threshold the Motor D pin signal outputs a PWM waveform at a user programmable duty cycle, variously connecting the motor D/shunt pin to GND in its active state, and HV in its inactive state. The PWM frequency used with shunt is equal to the motor drive PWM frequency. Once active, shunt PWM output will stop when the DC bus drops to 2.5% below the threshold comparison value.

The shunt resistor connected should have a resistance such that the current flow through the shunt switch, diode and resistor do not exceed the ratings of those components at the expected DC bus voltage. The diode, which is connected in parallel to the resistor, should have a voltage and current rating at least equal to the anticipated current flow.

For more information on the Shunt function and how to set the voltage threshold and PWM level settings refer to the *Magellan Motion Control IC User Guide*.

3.3 C-Motion Engine User Code Development

The next few sections will provide information and suggestions to get you started if your application calls for user-written code to be executed on the N-Series ION's C-Motion Engine (CME).

Creating, compiling, downloading, and verifying a user-written C-Motion application on a N-Series ION is accomplished with PMD's C-Motion Engine development system, described in the *C-Motion Engine Development Tools Manual*. The outcome of such a development sequence is a downloadable .bin file that contains the user application code and that can be executed by the C-Motion Engine.

3.3.1 Connections Overview

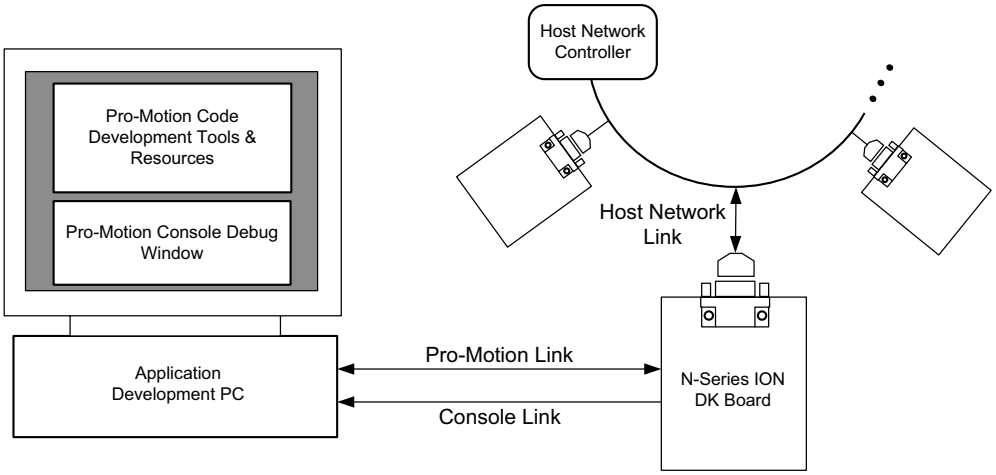


Figure 3-5: Typical Connection Links During C-Motion Engine Code Development

Figure 3-5 shows a typical connection architecture for C-Motion Engine user code development with a N-Series ION. There are three separate communication links:

Pro-Motion Link

The Pro-Motion link connects the PC that holds Pro-Motion, the user source code, and the source code editing and compiling tools to the N-Series ION that will receive the user code. This link allows the user code, once compiled and converted into a .bin file format, to be downloaded and eventually executed on the ION's C-Motion Engine. In addition this link can be used to monitor the status of the motion system even while the CME-based user application code is executing. This may be useful, for example, to track the location and status of the controlled axis.

The Pro-Motion link uses PRP (PMD Resource Access Protocol) to communicate with the N-Series ION. For more information on PRP refer to the *C-Motion/PRP II Programming Reference*.

The Pro-Motion link is generally always present during CME code development and execution.

Host Network Link

Many N-Series ION units in the actual application will be linked to a host/supervisory network to receive commands and report results and status. An example of a host network link is a factory floor network.

The host network link often uses an application-specific custom protocol or end-industry specific standard protocol to command the N-Series ION. Examples of commands that might be sent on such a host network are "Arm Extend" or "Move indexer to slot #15". The protocol and interpretation of such commands are specific to the end application and so the user code downloaded into the N-Series ION, among other things, will have the job of interpreting and responding to such commands. It is also possible that the host network utilizes PRP for communication, particularly if all of the devices on the network are PMD based products such as ION/CME or Prodigy/CME boards.

Not all systems executing user code on the CME will have a host network. For example some applications are fully standalone and do not have a host network.

For examples of different network topologies and applications supported by N-series IONs refer to the "Typical Applications" section of the *ION/CME N-Series Digital Drive User Manual*.

Console Link

The final link in a typical CME code development setup is the console channel. This function is discussed in more detail in the *ION/CME N-Series Digital Drive User Manual* but provides a convenient pathway for sending printf type

statements from the user code running on the N-Series ION to a separate monitor or to the Pro-Motion debug console window.

The console link does not have a protocol as such and transmits in ASCII whatever messages are sent using printf commands executed in the CME user code.

While a console port connection may be useful, particularly in the earlier stages of user code development, not all systems will need or use a console channel link.

3.3.2 Development Connections by ION Unit Type

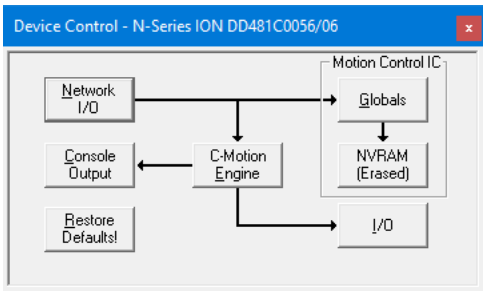
The table below provides information on typical connections, for each N-Series ION host interface type, for the above three link types. As noted earlier not every application will require all three connection links.

ION Host Interface Type	Pro-Motion Link	Host Network Link	Console Link
CAN/SPI	CAN - PRP node ID*	CAN - Host node ID	CAN - PRP node ID*
CAN/SPI	Serial3	SPI	Serial3
Serial	Serial3	RS232 - Serial1	Serial3**
Serial	Serial3	RS422 - Serial1	Serial3
Serial	Serial3	RS485 - Serial1	Serial3
Ethernet	TCP port 40100	TCP port 0-65,535 except 40100	UDP

** Pro-Motion and console traffic are carried via PRP on the PRP node ID. The Host node ID is user selected but must be different than the PRP node ID to avoid a conflict. If it is preferred that no additional CAN traffic is introduced to the Host CAN Network, Serial3 may instead be used for Pro-Motion and Console traffic.*

*** Serial3 carries Pro-Motion communications along with console traffic (using PRP's virtual console connection peripheral type). An alternative is to use the RS232 Serial2 connection for raw console traffic.*

3.3.3 Setting Up the Development Connections



Assuming that a functional Pro-Motion connection exists to the N-Series ION via Serial3, the programming port, the first step is to set up the host network connection if this has not been done already. Refer to Section 3.1, "Activating Serial Communications" for detailed instructions on specifying the host interface connection.

For the Ethernet interface typically all connections are made through Ethernet. The host network and Pro-Motion connections use TCP, and they must specify IP addresses different from each other to avoid conflicts. The console connection is also carried over Ethernet but uses the UDP protocol.

To set the console connection type use the Console Output Button located in the Device Control window.

3.3.4 Typical Code Development Session

With the CME code development connections in place, you are ready to start building your application using the tools provided in the C-Motion engine development system. These C-Motion engine development tools are used to create/edit, compile, and download user application code into the N-Series ION's CME (C-Motion Engine).

The development system can download the file image for the current code project being worked on, or a specific named file can be downloaded. Downloaded files images end with a ".bin" extension. Only one code image file may be downloaded into the C-Motion Engine at a time. Downloading a new image automatically erases the previous code image.

There are times when it may be useful to read specific characteristics of a code file that has been downloaded into the C-Motion Engine. For example a host controller in a production environment may want to confirm that the host application code version actually loaded on the C-Motion Engine matches the expected production code version. To confirm this information Pro-Motion displays, in the C-Motion Engine Window, the file name of the downloaded user application code, the checksum of the downloaded file, the date & time of file creation, and the version number of the C-Motion Engine itself is displayed.

To retrieve this information Pro-Motion sends PRP commands to the N-Series ION requesting this data. For complete information on the format and function of these, and other C-Motion calls refer to the *C-Motion/PRP II Programming Reference*.

3.3.4.1 Code Compilation, Downloading & Execution

The C-Motion Engine Development Tools Manual provides detailed step by step information on how to the edit, compile, and link source application code. The C-Motion Engine development environment uses the 'Programmer's Notepad' text editor and the GNU Compiler Collection (GCC) compiler and linker.

Once a .bin file has been generated it can be transferred and stored in the N-Series ION's CME using Pro-Motion. After the user code has been loaded into the N-Series ION unit it is ready to be executed. If the auto-execution flag has been set then the simplest way to begin executing the code is to re-power the N-Series ION. After initialization the user code will execute automatically.

If code execution is set to manual, with the N-Series ION powered on the code can be started using Pro-Motion via the C-Motion Engine Window. Once the code executes you should begin to see any printf statements that you have embedded in the code displayed in the console channel. During initial code development this is a convenient method to confirm that the code loaded correctly and is executing.

3.3.4.2 Sample "Hello World" Code Development Sequence

To exercise and become familiar with the C-Motion Development tools you may find it helpful to compile, link, and download a sample "hello world" application included in the SDK, using the steps detailed in the *C-Motion Engine Development Tools Manual*.

When executed, this simple "hello world" code sends a "hello world" message to the console. As a next step you can edit the source code file, for example changing the output message slightly, and then re-compile, download, and execute this modified .bin code image.

If you have trouble during any of the steps above contact your PMD representative for assistance.

Congratulations! You are now ready to begin developing your C-Motion Engine-based application code. For a general introduction to the N-Series ION's C-Motion Engine refer to *ION/CME N-Series Digital Drive User Manual*. For detailed information on C-Motion commands refer to the *C-Motion/PRP II Programming Reference*. For more information on the tools provided for code development refer to the *C-Motion Engine Development Tools Manual*.

4. Reference

In This Chapter

4.1 N-Series ION Unit Part Numbers and Configurations

There are 36 different ION/CME N-Series Digital Drives in all, consisting of the combinations of four motor types (step motor, Brushless DC, DC Brush, Multi-Motor), three host interfaces (Serial, CAN/SPI, Ethernet), and three power levels (low, medium, high). Note that multi-motor units allow the motor type, either Brushless DC, DC Brush, or step motor, to be user programmed.

The following table shows the available N-Series ION part numbers:

P/N	Host Interface	Power Level	Voltage	Motor Type
Step Motor				
DD441S0056/02	Serial	Low (75W)	12-56V	Step Motor
DD441S0056/06	Serial	Medium (350W)	12-56V	Step Motor
DD441S0056/18	Serial	High (1,000W)	12-56V	Step Motor
DD441C0056/02	CAN/SPI	Low (75W)	12-56V	Step Motor
DD441C0056/06	CAN/SPI	Medium (350W)	12-56V	Step Motor
DD441C0056/18	CAN/SPI	High (1,000W)	12-56V	Step Motor
DD441D0056/02	Ethernet	Low (75W)	12-56V	Step Motor
DD441D0056/06	Ethernet	Medium (350W)	12-56V	Step Motor
DD441D0056/18	Ethernet	High (1,000W)	12-56V	Step Motor

P/N	Host Interface	Power Level	Voltage	Motor Type
Brushless DC				
DD431S0056/02	Serial	Low (75W)	12-56V	Brushless DC
DD431S0056/06	Serial	Medium (350W)	12-56V	Brushless DC
DD431S0056/18	Serial	High (1,000W)	12-56V	Brushless DC
DD431C0056/02	CAN/SPI	Low (75W)	12-56V	Brushless DC
DD431C0056/06	CAN/SPI	Medium (350W)	12-56V	Brushless DC
DD431C0056/18	CAN/SPI	High (1,000W)	12-56V	Brushless DC
DD431D0056/02	Ethernet	Low (75W)	12-56V	Brushless DC
DD431D0056/06	Ethernet	Medium (350W)	12-56V	Brushless DC
DD431D0056/18	Ethernet	High (1,000W)	12-56V	Brushless DC
DC Brush				
DD411S0056/02	Serial	Low (75W)	12-56V	DC Brush
DD411S0056/06	Serial	Medium (350W)	12-56V	DC Brush
DD411S0056/18	Serial	High (1,000W)	12-56V	DC Brush
DD411C0056/02	CAN/SPI	Low (75W)	12-56V	DC Brush
DD411C0056/06	CAN/SPI	Medium (350W)	12-56V	DC Brush
DD411C0056/18	CAN/SPI	High (1,000W)	12-56V	DC Brush
DD411D0056/02	Ethernet	Low (75W)	12-56V	DC Brush
DD411D0056/06	Ethernet	Medium (350W)	12-56V	DC Brush
DD411D0056/18	Ethernet	High (1,000W)	12-56V	DC Brush
Multi Motor				
DD481S0056/02	Serial	Low (75W)	12-56V	Multi Motor
DD481S0056/06	Serial	Medium (350W)	12-56V	Multi Motor
DD481S0056/18	Serial	High (1,000W)	12-56V	Multi Motor
DD481C0056/02	CAN/SPI	Low (75W)	12-56V	Multi Motor
DD481C0056/06	CAN/SPI	Medium (350W)	12-56V	Multi Motor
DD481C0056/18	CAN/SPI	High (1,000W)	12-56V	Multi Motor
DD481D0056/02	Ethernet	Low (75W)	12-56V	Multi Motor
DD481D0056/06	Ethernet	Medium (350W)	12-56V	Multi Motor
DD481D0056/18	Ethernet	High (1,000W)	12-56V	Multi Motor

4.2 Introduction to Auto-run and Setup Scripts

Text editable scripts are a powerful feature of Pro-Motion and the N-Series ION Digital Drive. Two different script types are supported; Auto-run scripts and Setup scripts. Although they are used for different purposes they both utilize PMD Scripting Format, which is an ASCII text-based file format and set of associated syntax rules for specifying script file content.

4.2.1 Setup Scripts

Setup scripts are used to save and restore settings used by Pro-Motion and by the Magellan motion IC in the N-Series ION. Many setup scripts will be initially created by Pro-Motion's Axis Wizard. The parameters specified during the Axis Wizard process are then used to generate a setup script. This script file, once saved, can later be loaded at the beginning of a working session with Pro-Motion to set the motor control system to its initial state. Approximately 30 different parameters are saved in the Setup script by the Axis Wizard including servo gain settings, safety limit settings, high/low signal settings, and more. See Section 4.3, "Setup Scripts Content Reference," for a complete list of saved setting parameters.

Setup scripts can also be created and edited directly by the user using a .txt compatible program such as Microsoft Notepad. The Magellan commands that can be specified in PMD script are detailed in the *C-Motion Magellan Programming Reference*. There are approximately 100 different Magellan commands which together provide a system for specifying control settings such as trajectory parameters, servo settings, and other commands useful for controlling and monitoring the function of a motor.

4.2.1.1 Saving and Restoring Editted Setup Scripts

When starting a new Pro-Motion session, to load the content of a setup script and have its content become the active control parameters, the File/Open Project menu operation is used. Upon executing this menu function a dialog box will appear which lets you browse for and select a particular setup script.

Similarly, at the end of a Pro-Motion session, to save any changes made via Pro-Motion the File/Save Project menu function is used. At this time the user may save the script content with the same name (overwriting the previous script) or save the content with a new file name.

Note however that when Pro-Motion saves the control settings to a setup script any comments that may have been previously inserted into the script by a user via a text editor will be lost. This is because when creating the setup script as part of the File/Save Project operation Pro-Motion uses a fixed list of control variables, and reads only these variables from the N-Series ION. It does not read or save comments. Therefore if maintaining user-created script comments is desired then the File/Save Project function should not be used, and any desired control setting changes should be made directly by the user to the script file via a text editor.

When creating and maintaining their own setup scripts users are free to create scripts which contains fewer, or more, control setting commands than the standard script saved with the File/Save Project menu operation. Each time a File/Open Project request is made the specified script is executed. Pro-Motion does not check whether the list of saved control parameters is the same as the parameters it saves with the File/Save Project operation. If desired, multiple File/Open Project operations can therefore be performed if the user has split the control settings into multiple setup script files.

4.2.2 Magellan Command Mnemonics

At the heart of both setup scripts and auto-run scripts are command lines which request the execution of Magellan motion IC commands. To determine the exact mnemonic format that is needed for a particular Magellan command refer to the *C-Motion Magellan Programming Reference* and use the Instruction Reference chapter, which gives detailed content and function information for all available Magellan commands listed in alphabetical order.

The diagram below shows an example entry in the programming reference for a common Magellan command, **SetPosition**, which specifies the desired destination position for trapezoidal and s-curve point-to-point moves. In addition to the script syntax required, which is listed at the very top of the entry as 'Syntax,' information about the command arguments, the compiled packet format, a functional description of the command, and additional format descriptions for C-Motion or .NET programming access is contained in each command documentation entry.

SetPosition / GetPosition — buffered — 10h / 4Ah

2

Syntax

SetPosition *axis position*
GetPosition *axis*

Motor Types

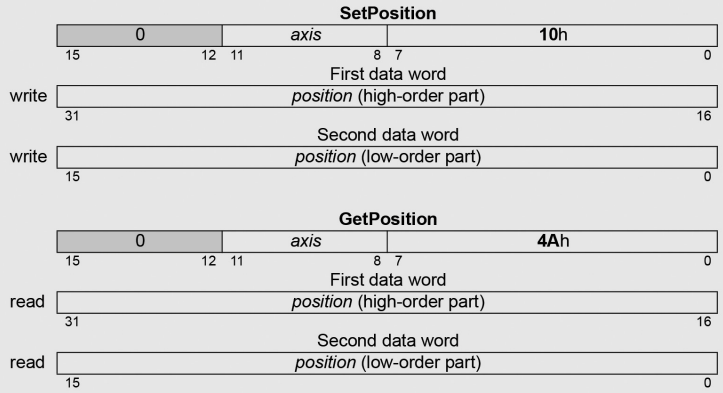
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	RangeScalingUnits		
position	signed 32 bits	-2^{31} *to* $2^{31}-1$	unity	counts microsteps

Packet Structure



Description

SetPosition specifies the trajectory destination of the specified ***axis***. It is used in the Trapezoidal and S-curve profile modes.

GetPosition reads the contents of the buffered position register.

Restrictions

SetPosition is a buffered command. The value set using this command will not take effect until the next **Update** or **MultiUpdate** command, with the Trajectory Update bit set in the update mask.

C-Motion API

```
PMDresult PMDSetPosition(PMDAxisInterface axis_intf,
                         PMDint32 position);
PMDresult PMDGetPosition(PMDAxisInterface axis_intf,
                         PMDint32* position)
```

VB-Motion API

```
Dim position as Long
MagellanAxis.Position = position
position = MagellanAxis.Position
```

see

Set/GetAcceleration (p. 83), **Set/GetDeceleration** (p. 122), **Set/GetJerk** (p. 148), **Set/GetVelocity** (p. 213), **MultiUpdate** (p. 65), **Update** (p. 215)

172 C-Motion Magellan Programming Reference

As can be seen from the image above the Syntax entry for **SetPosition** is:

SetPosition axis position

The command mnemonic is 'SetPosition' which takes arguments *axis* and *position*. The axis argument normally specifies which axis will recevie the command but in single axis controllers such as the N-Series IONs specifying the axis is not necessary. Therefore the correct ASCII script entry to specify a particular desired destination position (we will use 123456 as an example) is:

SetPosition 123456

4.2.2.1 Multi-argument Commands

SetPosition is a relatively simple command that takes a single argument, the desired destination position. Some Magellan commands take two or even three arguments however. An example of such a command is **SetDriveFaultParameter**, which allows various safety limits such as overvoltage and undervoltage thresholds to be set. Here is the first page of the *C-Motion Magellan Programming Reference* entry for this command:

SetDriveFaultParameter 62h
GetDriveFaultParameter 60h

2

Syntax

SetDriveFaultParameter *axis parameter value*
GetDriveFaultParameter *axis parameter*

Motor Types

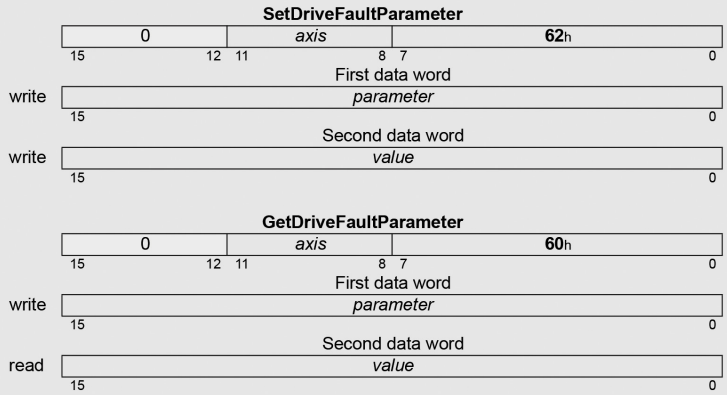
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding	Units
axis	*Axis1*	0	
	Axis2	1	
	Axis3	2	
	Axis4	3	
parameter	*Overvoltage Limit*	0	V
	Undervoltage Limit	1	V
	Event Recovery Mode	2	N/A
	Watchdog Limit	3	s
	Temperature Limit	4	°C
	Temperature Hysteresis	5	°C
	— (Reserved)	6	
	— (Reserved)	7	
	Shunt voltage limit	8	V
	Shunt duty	9	%
	Bus current supply limit	10	A
	Bus current return limit	11	A

	Type	Range	Scaling
value	unsigned 16 bits	see below	see below

Packet Structure



Description

SetDriveFaultParameter sets various drive operation limits. The particular limit set depends on the parameter argument. When an operation limit is exceeded, motor output will be disabled and either a Drive Exception or Overtemperature event will be raised, and a bit set in the Drive Fault Status register to indicate the fault.

This command allows multiple parameter settings to be specified, with the 'parameter' argument index used to specify which parameter to set, and the value used to specify the actual setting. So for example to set the undervoltage limit (encoded as a parameter value of 1) to 20, the following script entry would be used:

SetDriveFaultParameter 1 20

Note again that the axis argument is not specified.

Some two or three argument commands have a more complex encoding, requiring multiple parameters to be loaded into the argument word at certain bit field locations. Here again though, the information for each command provided in the *C-Motion Magellan Programming Reference* details exactly how the argument value should be constructed.

In cases where separate values must be loaded into specific bit field positions it may be easier to specify the argument value as a hexadecimal number. This is accomplished by prefacing the specified argument value with '0x'. More on numerical argument syntax in Section 4.2.4, "PMD Scripting Format Syntax."

4.2.2.2 Get Commands

Both of the command description entries shown above contained both a Set and a Get command. For example the entry for **SetPosition** described both the **SetPosition** and the **GetPosition** command, and the entry for **SetDriveFaultParameter** also included the **GetDriveFaultParameter**.

In cases where there are paired Set and Get commands accessing the same parameter settings the entry describes both for the sake of efficiency. Using a Get command in a script is similarly straightforward, although doing so is generally only done in auto-run scripts where the returned value is used in a subsequent calculation or operation.

As can be seen from the Syntax entries the **GetPosition** command has no associated arguments, and the **GetDriveFaultParameter** command has only a parameter argument. As for Set commands there is no need to include the axis argument in the script command line.

The ASCII script entry to request the current destination position kept in the Magellan IC is:

GetPosition

and the ASCII script entry to request the current undervoltage threshold setting kept in the Magellan IC is:

GetDriveFaultParameter 1

4.2.3 Auto-Run Scripts

Like setup scripts, auto-run scripts, at the core, consist of Magellan commands to execute motion profiles, control signals, or change control settings. There are two main differences between setup scripts and auto-run scripts. The first is that setup scripts are processed, one command at a time, by Pro-Motion with any resulting Magellan commands immediately being sent to the connected N-Series ION. The Magellan IC receiving these commands processes them as it would any other packet command sent to it by Pro-Motion.

By contrast auto-run scripts are compiled, converted into a block of packet content, transferred into the Magellan IC's RAM, and then when 'started', executed from the Magellan IC. So rather than receiving a series of packets via a host connections command packets are executed from the Magellan's internal RAM, having being previously transferred as a memory block into RAM by Pro-Motion.

The second difference is that setup scripts do not use auto-run programming constructs such as mathematical operations (add, sub, and) branches (b, bz, etc...), or print output. This again stems from the fact that setup scripts are not run in the Magellan as a program, but instead are processed by Pro-Motion one command at a time and sent via the host interface to the Magellan IC.

4.2.4 PMD Scripting Format Syntax

Below is listed complete syntax information for PMD Scripting Format.

File type – Script files are comprised of ASCII text files

Comments – Comments are content embedded in the script file that are not executed, but rather are used by the script developer to clarify its content. Comments may appear anywhere within a line and result in all subsequent characters to the end of the line being ignored during script processing. Two characters are recognized as beginning a comment; a single quote ', and a tilde ~.

Case sensitivity – Command mnemonics, argument mnemonics, and address labels are case insensitive. For example the command **SetPosition** will be compiled exactly the same as **setposition** or **SETPOSITION**, and an argument mnemonic of **ActualPosition** will be compiled the same as **actualposition** or **ACTUALPOSITION**.

Leading spaces and tabs – With the exception of commands which being with a ":" (more on these below) leading spaces and tabs are ignored.

Blank lines – Similarly, blank lines are ignored. Inserting blank lines for the purpose of visual presentation will have no affect on the compilation of the script.

Commands and Newlines – Only one command and its associated arguments (if any are needed) are recognized per line. Recognized characters marking the end of the line are \R (ASCII 13) and \N (ASCII 10).

Arguments – Arguments may be of several different types depending on the command they are used with. Argument types include; general purpose registers, motion registers, immediate numerical values, and ASCII strings. General purpose and motion register mnemonics must exactly match the available mnemonics for that argument value.

Argument delimiters – Arguments must be separated from adjacent commands or arguments by either a space or a tab. Commas are not an allowed delimiter for arguments.

Numerical arguments – Numerical arguments may be specified in decimal or hexadecimal, with hexadecimal numbers being denoted with a leading '0x' character sequence with no intervening space, for example 0x1234. For both decimal and hex numbers leading zeros are ignored. For example 0012 is interpreted as decimal 12 and 0x01B3 as hexadecimal 1B3. Commas are not allowed in number arguments. Therefore '123456' is OK but '123,456' is not. Finally, decimal numbers (but not hexadecimal) may have a leading "-" before the digits without an intervening space to indicate a negative number, for example -12.

ASCII strings – ASCII strings are a special type of argument used only in printing. They begin and end with a double quote character with everything in between the quotes being interpreted as the content of the string. See Section 2.8.1, "Print Formatting," for more information on interpretation of special character sequences within ASCII strings by the Print command.

AddressLabels – Address labels are distinguished from commands or arguments in that they end with a ":" character. With the exception of comments, which may appear on the same line after an address label, address labels must occur by themselves on a line. AddressLabels can include alpha characters (A-Z), numbers (0-9), and the underscore character anywhere in the label. All other characters are not allowed. In addition they can not include delimiter characters such as spaces or tabs. So a label of StartLoop_1 is OK but Start Loop_1 is not.

PSF entries (:CN, :CVER, :DESC) – The following three entries describe special character sequences defined in PSF (PMD Structured data Storage Format) that are used with auto-run and setup scripts. Use of these entries is optional. They are not required to be present in auto-run or setup scripts. PSF entries must begin at the first character in the newline. Leading spaces or tabs are not allowed. Unlike most other script constructs PSF entries are case sensitive. So :CN will be properly recognized but :cN will not. Although by convention PSF entries are listed at the top of the script, this is not a requirement. They can be located anywhere in the script and still be recognized correctly.

:CN – The :CN PSF entry registers an ASCII name string that is readable by Pro-Motion when the script is parsed. The specified string does not use double quotes to begin and end the string. The name string follows the ":CN" entry

after a space delimiter and consists of all characters till the newline excepting trailing spaces. Although Pro-Motion displays the content of this PSF field as the script "Name", in fact any text string can be placed here by the script developer.

:CVER – The :CVER PSF entry is intended to register a version number for the script. This may be useful to track different versions of the script as it is being developed. However the actual format of the :CVER field is the same as for :CN, namely an ASCII string without beginning and ending double quote characters that is read up to the end of the line.

:DESC – The :DESC PSF entry is intended to register a narrative description of the content of the script. As for :CN and :CVER this entry can be read by Pro-Motion and displayed. Unlike these PSF entries however :DESC uses double quotes to demarcate the beginning and end of the DESC content. In addition, newlines do not represent the end of the field content, only a second double quote character. This means description text can be specified over multiple lines of ASCII text.

4.3 Setup Scripts Content Reference

When saving the application configuration at the end of an Axis Wizard session, or when specifying the File/Save Project function, a specific set of control settings are saved into the specified setup script file.

The tables below provide the list of parameters saved by Pro-Motion in setup scripts. There are four tables, the first is saved parameters that are processed by Pro-Motion only. The second parameters are saved for all motor types. The third and fourth are parameters that are saved for motors operated in microstepping control mode and closed loop stepper control mode respectively.

4.3.1 Pro-Motion Processed Setup Script Parameters

The table below shows the saved setup script parameters that are processed by Pro-Motion but that are not sent to the N-Series ION motion controller.

Parameter	Script Command*	Comments
Time units	#SetUnits TIME *value*	*value* can have one of the following values: 0 – Magellan IC cycles 1 – milliseconds 2 – seconds
Position units	#SetUnits UNITS *value*	*value* can have one of the following values: 0 - encoder counts 1 – millimeters 2 – inches 4 – meters 5 – micrometers 9 – degrees 11 – revolutions

Parameter	Script Command*	Comments
Position scale	#SetUnits SCALE *EncoderCount UnitCount*	This command specifies the ratio of encoder counts to the selected unit. For example if the unit type is revolutions and an encoder provides 4,096 counts per revolution the command: #SetUnits SCALE 4096 1 would be specified. *EncoderCount* and *UnitCount* are encoded as 32-bit integers. By specifying values of *UnitCount* other than 1 non integer ratios can be specified. For example a particular motor system has 100 encoder counts per 1.5 millimeters of motion the command #SetUnits SCALE 200 3 could be used, as could any ratiometrically equivalent pair of integer values.
Number of pole pairs	#MotorSpec POLEPAIRS *value*	*value* is the number of commutation electrical cycles per mechanical revolution. For Brushless DC motor this is almost always equal to the number of pole pairs, however for step motors this may not be the case. value must be specified as an integer.
I2t continuous current maximum	#MotorSpec I2TCURRENTLIMIT *value*	*value* is the maximum allowed I2t continuous current setting in amps. The specified value may be an integer or floating point number.
I2t energy maximum	#MotorSpec I2TENERGYLIMIT *value*	*value* is the maximum allowed I2t energy setting in amp^2secs. The specified value may be an integer or floating point number.
Voltage ramp rate maximum	#MotorSpec VOLTAGERAMPLIMIT *value*	*value* is the maximum allowed voltage output ramp rate in volts/second. The specified value may be an integer or floating point number.
Voltage output maximum	#MotorSpec VOLTAGEOUTPUTLIMIT *value*	*value* is the maximum allowed voltage output in volts. The specified value may be an integer or floating point number.
Motor command mode	#ControlMode *value*	*value* is the motor command mode. A value of 1 selects Voltage mode, a value of 2 selects Torque Control mode, and a value of 4 selects Position and Velocity Control mode.

4.3.2 Setup Script Parameters

The table below shows the saved setup script parameters when three-phase Brushless motors are used. All of the script commands listed below are detailed in the *C-Motion Magellan Programming Reference.*

Parameter	Script Command	Comments
Sample time	SetSampleTime *value*	
Motor type	SetMotorType *value*	*value* for microstepping control mode is 3 *value* for closed loop stepper is 1
Position error limit	SetPositionErrorLimit *value*	
Axis settled time	SetSettleTime *value*	
Axis settled window	SetSettleWindow *value*	
Tracking window	SetTrackingWindow *value*	
Encoder source	SetEncoderSource *value*	Setting for primary* axis (axis #1)
Motion complete mode	SetMotionCompleteMode *value*	
Signal sense register	SetSignalSense *value*	Setting for primary* axis (axis #1)
Position capture source	SetCaptureSource *value*	
Position loop Kp gain	SetPositionLoop 0 *value*	
Position loop Kd gain	SetPositionLoop 3 *value*	
Position loop Kd time	SetPositionLoop 4 *value*	

Parameter	Script Command	Comments
Position loop Ki gain	SetPositionLoop 1 *value*	
Position loop Ilimit	SetPositionLoop 2 *value*	
Position loop Kout	SetPositionLoop 5 *value*	
Position loop Kvff	SetPositionLoop 6 *value*	
Position loop Kaff	SetPositionLoop 7 *value*	
Motor bias	SetMotorBias *value*	
Motor limit	SetMotorLimit *value*	
Dual loop encoder source	SetAuxiliaryEncoderSource *value*	
Biquad1 Enable	SetPositionLoop 8 *value*	
Biquad1 B0	SetPositionLoop 9 *value*	
Biquad1 B1	SetPositionLoop 10 *value*	
Biquad1 B2	SetPositionLoop 11 *value*	
Biquad1 A1	SetPositionLoop 12 *value*	
Biquad1 A2	SetPositionLoop 13 *value*	
Biquad1 K scaling	SetPositionLoop 14 *value*	
Biquad2 Enable	SetPositionLoop 15 *value*	
Biquad2 B0	SetPositionLoop 16 *value*	
Biquad2 B1	SetPositionLoop 17 *value*	
Current control mode	SetCurrentControlMode *value*	
FOC D Kp gain	SetFOC 0 *value*	
FOC D Ki gain	SetFOC 1 *value*	
FOC D Ilimit	SetFOC 2 *value*	
FOC Q Kp gain	SetFOC 256 *value*	
FOC Q Ki gain	SetFOC 257 *value*	
FOC Q Ilimit	SetFOC 258 *value*	
AxisOut mask	SetAxisOutMask *value1 value2 value3*	
FaultOut mask	SetFaultOutMask *value*	
Overvoltage limit	SetDriveFaultParameter 0 *value*	
Undervoltage limit	SetDriveFaultParameter 1 *value*	
Temperature limit	SetDriveFaultParameter 4 *value*	
Temperature hysteresis	SetDriveFaultParameter 5 *value*	
Bus current return limit	SetDriveFaultParameter 11 *value*	
I^2t continuous current limit	SetCurrentFoldback 0 *value*	
I^2t energy limit	SetCurrentFoldback 1 *value*	
Positive limit event	SetEventAction 1 *value*	
Negative limit event	SetEventAction 2 *value*	
Motion error event	SetEventAction 3 *value*	
Current foldback event	SetEventAction 4 *value*	
Brake action	SetEventAction 13 *value*	
PWM frequency	SetDrivePWM 3 *value*	
PWM duty cycle limit	SetDrivePWM 0 *value*	
Encoder source	SetEncoderSource *value*	Setting for auxiliary* axis (axis #1)
Signal sense register	SetSignalSense *value*	Setting for auxiliary* axis (axis #1)

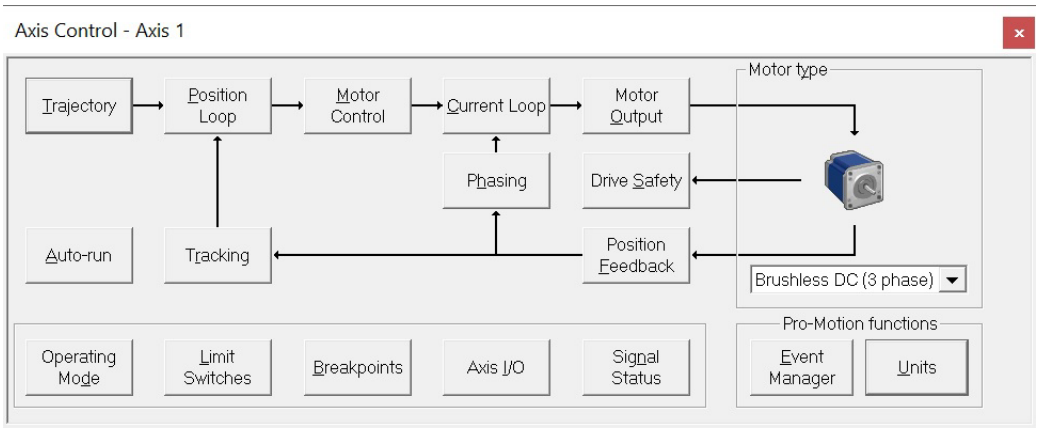
** To specify the axis # the script command #Axis 1 or #Axis 2 is used. Script commands located after these commands will apply to the specified axis.*

4.4 Motor Control Mode Operation

Below are the three user-selectable motor control modes provided by the Portescap Motor Application Development Kit:.

1. **Velocity Mode** outputs a programmed voltage command to the motor. Neither position, velocity, or the torque are explicitly controlled.
2. **Torque Control Mode** provides torque control. The position and velocity are not explicitly controlled.
3. **Position and Velocity Control Mode** provides trajectory profile generation and a position PID servo control loop. Applications which require precise control of the motor's position or velocity should choose this motor control mode.

The Magellan Motion Control IC provides control for these three different modes by selectively enabling or disabling certain control modules. For reference the diagram below shows the main 'full function' control flow of the Magellan Motion Control IC.



The mainline control sequence of the Magellan IC, when all control modules are operating, begins with the Trajectory generator which outputs a desired position to the Position PID servo loop module. The position PID servo loop in turn outputs a desired current command to the Current loop module. The current (torque) loop in turn outputs commanded voltages to each coil of the motor. The commanded voltages are synthesized using a PWM (Pulse Width Modulated) technique to drive the switching amplifier bridge.

The table below shows which of the Magellan control modules are operated for each of the three user-available motor control modes:

Motor Control Mode	Trajectory/ Ramp Generator	Position Servo Loop	Current Servo Loop	PWM Amplifier Motor Output
Voltage	Yes	No	No	Yes
Torque Control	Yes	No	Yes	Yes
Position and Velocity Control	Yes	Yes	Yes	Yes

The Trajectory Generator is operated in two different ways depending on the motor control mode selected. Its normal operating function, to generate various types of position and velocity trajectories, is used when the Position and Velocity Control Mode is active. However Magellan supports an alternate use for the trajectory generator module when the position loop is not enabled. When that is the case the same profile generating features of this module are active but their output is not interpreted by the downstream modules as a commanded position. The output values are instead interpreted as a commanded torque or voltage depending on what downstream modules are active.

In this alternate mode the trapezoidal profile is used with a very high acceleration parameter and with the velocity parameter specifying the ramp (slew) rate. The 'position' output is limited to 16 bits of resolution which as mentioned above is interpreted as the torque command or the voltage command.

The above details are provided to illustrate how the features provided by Pro-Motion connect with the underlying commands and control structures provided by the Magellan IC. Nevertheless most users will not need to be aware of these details, and will simply use Pro-Motion, which handles all these details, to control the motor as desired.

4.5 Portescap Motor Control Settings Reference

The tables below provides the pre-determined Magellan Motion Control IC motor control settings used with each different Portescap motor part number and configuration.

4.5.1 45ECF22 8B 25 Motor Settings, Hall-Based Operation

Parameter Setting	Position and Torque Control Mode	Torque Control Mode	Voltage Mode
PWM Frequency (kHz)	20	20	20
# Pole Pairs	8	8	8
I2t Continuous Current (Arms)	2.2	3.0	3.0
I2t Energy Limit (S*Arms^2)	25.0	25.0	25.0
Maximum current (Arms)	4.4	3.0	N/A
Voltage Limit (Volts)	20.0	20.0	20.0
Voltage ramp rate limit (V/Sec)	N/A	N/A	20.0
Position Feedback Source	Halls	Halls	Halls
Control Method	3rd Leg Floating	3rd Leg Floating	3rd Leg Floating
Current Kp	16	16	N/A
Current Ki	9	9	N/A
Current Ilimit	16,000	16,000	N/A
FOC Decoupling	N/A	N/A	N/A
Sample time (mSec)	1.024	1.024	1.024
Position Kp	50	N/A	N/A
Position Ki	40	N/A	N/A
Position Ilimit	10,000	N/A	N/A
Position Kd	2000	N/A	N/A
Position Derivative Time	6	N/A	N/A
Kout (%)	50	N/A	N/A
Signal Sense	0x380	0x380	0x380

4.5.2 30ECT64 10B 7 01 Motor Settings, Hall-Based Operation

Parameter Setting	Position and Torque Control Mode	Torque Control Mode	Voltage Mode
PWM Frequency (kHz)	80	80	80

Parameter Setting	Position and Torque Control Mode	Torque Control Mode	Voltage Mode
# Pole Pairs	2	2	2
I2t Continuous Current (Arms)	7.8	7.8	7.8
I2t Energy Limit (S*Arms^2)	25.0	25.0	25.0
Maximum current (Arms)	12.0	7.8	N/A
Voltage Limit (Volts)	19.0	19.0	19.0
Voltage ramp rate limit (V/Sec)	N/A	N/A	20.0
Position Feedback Source	Halls	Halls	Halls
Current Control Method	3rd Leg Floating	3rd Leg Floating	3rd Leg Floating
Current Kp	45	45	N/A
Current Ki	45	45	N/A
Current Ilimit	16,000	16,000	N/A
FOC Decoupling	N/A	N/A	N/A
Sample time (mSec)	1.024	1.024	1.024
Position Kp	100	N/A	N/A
Position Ki	20	N/A	N/A
Position Ilimit	100,000	N/A	N/A
Position Kd	1,000	N/A	N/A
Position Derivative Time	4	N/A	N/A
Kout (%)	100	N/A	N/A
Signal Sense	0x380	0x380	0x380

4.5.3 40ECP44 8 7 Motor Settings, Hall-Based Operation

Parameter Setting	Position and Torque Control Mode	Torque Control Mode	Voltage Mode
PWM Frequency (kHz)	120	120	120
# Pole Pairs	2	2	2
I2t Continuous Current (Arms)	7.8	7.8	7.8
I2t Energy Limit ($S*Arms^2$)	25	25	25
Maximum current (Arms)	12.0	7.8	N/A
Voltage Command Limit	14.0	14.0	14.0
Voltage ramp rate limit (V/Sec)	N/A	N/A	20.0
Position Feedback Source	Halls	Halls	Halls
Current Control Method	3rd Leg Floating	3rd Leg Floating	3rd Leg Floating
Current Kp	20	20	N/A
Current Ki	30	30	N/A
Current Ilimit	16,000	16,000	N/A
FOC Decoupling	N/A	N/A	N/A
Sample time (mSec)	1.024	1.024	1.024
Position Kp	600	N/A	N/A
Position Ki	10	N/A	N/A
Position Ilimit	2,000	N/A	N/A
Position Kd	6,000	N/A	N/A
Position Derivative Time	6	N/A	N/A
Kout (%)	50	N/A	N/A
Signal Sense	0x380	0x380	0x380

4.5.4 B0912N1010 Motor Settings, Hall-Based Operation

Parameter Setting	Position and Torque Control Mode	Torque Control Mode	Voltage Mode
PWM Frequency (kHz)	120	120	120
# Pole Pairs	2	2	2
I2t Continuous Current (Arms)	7.8	7.8	7.8
I2t Energy Limit (S*Arms^2)	25	25	25
Maximum current (Arms)	12.0	7.8	N/A
Voltage Command Limit	9.0	9.0	9.0
Voltage ramp rate limit (V/Sec)	N/A	N/A	20.0
Position Feedback Source	Halls	Halls	Halls
Current Control Method	3rd Leg Floating	3rd Leg Floating	3rd Leg Floating
Current Kp	40	40	N/A
Current Ki	35	35	N/A
Current Ilimit	16,000	16,000	N/A
FOC Decoupling	N/A	N/A	N/A
Sample time (mSec)	1.024	1.024	1.024
Position Kp	20	N/A	N/A
Position Ki	5	N/A	N/A
Position Ilimit	100,000	N/A	N/A
Position Kd	15,000	N/A	N/A
Position Derivative Time	8	N/A	N/A
Kout (%)	15	N/A	N/A
Signal Sense	0x380	0x380	0x380

4.5.5 22ECT48 10B 23 01 Motor Settings, Hall-Based Operation

Parameter Setting	Position and Torque Control Mode	Torque Control Mode	Voltage Mode
PWM Frequency (kHz)	40	40	40
# Pole Pairs	2	2	2
I2t Continuous Current (Arms)	2.2	2.2	2.2
I2t Energy Limit (S*Arms^2)	25	25	25
Maximum current (Arms)	4.4	2.2	N/A
Voltage Command Limit	20.0	20.0	20.0
Voltage ramp rate limit (V/Sec)	N/A	N/A	20.0
Position Feedback Source	Halls	Halls	Halls
Current Control Method	3rd Leg Floating	3rd Leg Floating	3rd Leg Floating
Current Kp	120	120	N/A
Current Ki	100	100	N/A
Current Ilimit	16,000	16,000	N/A
FOC Decoupling	N/A	N/A	N/A
Sample time (mSec)	1.024	1.024	1.024
Position Kp	200	N/A	N/A
Position Ki	20	N/A	N/A

Parameter Setting	Position and Torque Control Mode	Torque Control Mode	Voltage Mode
Position Ilimit	5,000	N/A	N/A
Position Kd	2,000	N/A	N/A
Position Derivative Time	4	N/A	N/A
Kout (%)	25	N/A	N/A
Signal Sense	0x380	0x380	0x380

4.5.6 22ECT48 10B 23 01 Motor Settings, Encoder-Based Operation

Parameter Setting	Position and Torque Control Mode	Torque Control Mode	Voltage Mode
PWM Frequency (kHz)	40	40	40
# Pole Pairs	2	2	2
I2t Continuous Current (Arms)	2.2	2.2	2.2
I2t Energy Limit (S*Arms^2)	25	25	25
Maximum current (Arms)	4.4	2.2	N/A
Voltage Command Limit	20.0	20.0	20.0
Voltage ramp rate limit (V/Sec)	N/A	N/A	20.0
Position Feedback Source	Encoder	Encoder	Encoder
Current Control Method	FOC	FOC	3rd Leg Floating
Current Kp	120	120	N/A
Current Ki	100	100	N/A
Current Ilimit	16,000	16,000	N/A
FOC Decoupling	0	0	N/A
Sample time (mSec)	.0512	.0512	.0512
Position Kp	60	N/A	N/A
Position Ki	40	N/A	N/A
Position Ilimit	100,000,000	N/A	N/A
Position Kd	1,000	N/A	N/A
Position Derivative Time	12	N/A	N/A
Kout (%)	10	N/A	N/A
Signal Sense	0x380	0x380	0x380

4.6 Scaling Information for Selected Commands & Registers

4.6.1 Position

For servo motors native Magellan position units are encoder counts. The position is kept as a signed 32-bit number.

Several auto-run script commands use native Magellan position units including **Set/GetPosition**, **Set/GetActualPosition**, **AdjustActualPosition**, **GetCommandedPosition**, **GetCapturePos**, **GetPositionError**, **Set/GetPositionErrorLimit**. In addition these motion registers use native position units: **CommandedPosition**, **ActualPosition**, **PositionError**.

To convert position from native to real world units and vice versa we use the following equations:

$Position_{RW} = Position_{N} * Ratio_{Pos}(RW/N)$

and

$Position_{N} = Position_{RW} / Ratio_{Pos}(RW/N)$

where:

$Position_{RW}$ is the position in real world units
$Position_{N}$ is the position in counts
$Ratio_{Pos}(RW/N)$ is the ratio of real world position to counts.

Example

A rotary encoder has a resolution of 2,048 counts per rotation. For a measured position of 1,000 counts what is the position in degrees?

First we calculate the conversion ratio. Here, this will be 360.0 degrees/rotation / 2,048 counts/rotation = 0.1758 deg/count

This gives:

$Position_{RW} = Position_{N} * Ratio_{Pos}(RW/N)$
PositionRW = 1,000 counts * 0.1758 deg/count
PositionRW = 175.8 deg

Example

A rotary encoder has a resolution of 2,048 counts per rotation. It is attached to a belt which converts one full motor rotation into 1.2 cm of linear motion. For a measured position of 25,000 counts what is the position in cms?

$Ratio_{Pos}(R/W) = 1.2$ cm/rotation / 2,048 counts/rotation

$Ratio_{Pos}(R/W) = .0005859$ cm/count

$Position_{RW} = Position_{N} * Ratio_{Pos}(RW/N)$

PositionRW = 25,000 counts * .0005859 cm/count

PositionRW = 14.648 cm

Example

In the above example what is the native Magellan position when the position in cms is 10.0 cm.

$Position_{N} = Position_{RW} / Ratio_{Pos}(RW/N)$

$Position_{N} = 10.0$ cm / .0005859 cm/count

$Position_{N} = 17{,}068$ counts

4.6.2 Velocity

Native velocity units for the Magellan are counts/cycle, however this quantity is scaled by a factor of 65,536 meaning a velocity 1.0 counts/cycle will have a numerical representation of 65,536. Native time units for the Magellan are cycles, which are 51.2 μSec long, meaning that there are 19,531 cycles per second. The velocity is kept as a 32 bit number.

Several auto-run script commands use native Magellan velocity units including **Set/GetVelocity**, and **GetCommandedVelocity**. In addition these motion registers use native velocity units: **CommandedVelocity**, **ActualVelocity**

To convert velocity from native units & scaling to real world position units per second and vice versa we use the following equations:

$Velocity_{RW} = Velocity_{N} * ratio_{Vel}RW/N) / 65,536$

and

$Velocity_{N} = 65,536 * Velocity_{RW} / ratio_{Vel}(RW/N)$

where:

$Velocity_{RW}$ is the velocity in real world velocity units/second
$Velocity_{N}$ is the velocity in counts/cycle
$Ratio_{Vel}(RW/N)$ is the ratio of real world velocity units/sec to counts/sample time

Example

A rotary encoder has a resolution of 1,000 counts per rotation. What is the native Magellan velocity value for a rotation speed of 25,000 deg/sec?

First we calculate the conversion ratio. Here, this will be 360.0 deg/sec / (1,000 counts / sec / 19,531 cycle/sec)

$Ratio_{Vel}(RW/N) = 7,031.3$ deg/sec / counts/cycle

This gives:

$Velocity_{N} = 65,536 * Velocity_{RW} / Ratio_{Vel}(RW/N)$

$Velocity_{N} = 65,536 * 25,000$ deg/sec / 7,031.3 deg/sec / counts/cycle

$Velocity_{N} = 65,536 * 3.5555$ counts/cycle

$Velocity_{N} = 233,015$

Example

A linear encoder has a resolution of 100,000 counts per cm. What is the native Magellan velocity value for a velocity of 25.0 cm/sec?

$Ratio_{Vel}(RW/N) = 1.0$ cm/sec / (100,000 counts / sec / 19,531 cycles/sec)

$Ratio_{Vel}(RW/N) = 0.1953$ cm/sec / counts/cycle

$Velocity_{N} = 65,536 * Velocity_{RW} / Ratio_{Vel}(RW/N)$

$Velocity_{N} = 65,536 * 25.0$ cm/sec / 0.19531 cm/sec / counts/cycle

$Velocity_{N} = 65,536 * 128.0$ counts/cycle

$Velocity_{N} = 8,388,608$

Example

In the above system the Magellan reports a velocity of 2,000,000. What velocity does this correspond to in cm/sec?

$Velocity_{RW} = Velocity_N * Ratio_{Vel}(RW/N) / 65,536$

$Velocity_{RW} = 2,000,000 * 0.1953$ cm/sec / counts/cycle/ 65,536

$Velocity_{RW} = 5.9604$ cm/sec

4.6.3 Acceleration

Native acceleration units for the Magellan are counts/cycle^2, however this quantity is scaled by a factor of 65,536 meaning an acceleration of 1.0 counts/cycle^2 will have a numerical representation of 65,536. Native time units for the Magellan are cycles, which are 51.2 μSec long, meaning that there are 19,531 cycles per second. The acceleration is kept as a 32 bit number.

Several auto-run script commands use native Magellan acceleration units including **Set/GetAcceleration**, and **Set/GetDeceleration**. In addition these motion registers use native acceleration units: **CommandedAcceleration**

To convert acceleration from native units & scaling to real world position units per second^2 and vice versa we use the following equations:

$Acceleration_{RW} = Acceleration_N * Ratio_{Acc}(RW/N) / 65,536$

and

$Acceleration_N = 65,536 * Acceleration_{RW} / Ratio_{Acc}(RW/N)$

where:

$Acceleration_{RW}$ is the acceleration in real world position units/second^2

$Acceleration_N$ is the acceleration in scaled counts/cycle^2

$Ratio_{Acc}(RW/N)$ is the ratio of real world position units/sec^2 to counts/cycle^2

Example

A rotary encoder has a resolution of 1,000 counts per rotation. What is the native Magellan acceleration value for a rotational acceleration of 500,000 deg/sec^2?

First we calculate the acceleration conversion ratio. Here, this will be 360.0 deg/sec / (1,000 counts / sec / (19,531^2) cycles/sec)

$Ratio_{Acc}(RW/N) = 137,325,586.0$ deg/sec^2 / counts/cycle^2

This gives:

$Acceleration_N = 65,536 * Acceleration_{RW} / Ratio_{Acc}(RW/N)$

$Acceleration_N = 65,536 * 500,000$ deg/sec^2 / 137,325,586.0 deg/sec^2 / counts/cycle^2

$Acceleration_N = 65,536 * .003641$ counts/cycle^2

$Acceleration_N = 239$

Example

A linear encoder has a resolution of 100,000 counts per cm. What is the native Magellan acceleration value for an acceleration of 25.0 cm/sec^2?

$Ratio_{Acc}(RW/N)$ = 1.0 cm/sec / (100,000 counts / sec / ($19,531^2$) cycles/sec)

$Ratio_{Acc}(RW/N)$ = 3,814.6 cm/sec^2 / counts/$cycle^2$

$Acceleration_N$ = 65,536 * $Acceleration_{RW}$ / $Ratio_{Acc}(RW/N)$

$Acceleration_N$ = 65,536 * 25.0 cm/sec^2 / 3,814.6 cm/sec^2 / counts/$cycle^2$

$Acceleration_N$ = 65,536 * .006554 counts/$cycle^2$

$Acceleration_N$ = 430

Example

In the above system the Magellan reports an acceleration of 10,000. What Acceleration does this correspond to in cm/sec^2?

$Acceleration_{RW}$ = $Acceleration_N$ * $Ratio_{Acc}(RW/N)$ / 65,536

$Acceleration_{RW}$ = 10,000 * 3,814.6 cm/sec^2 / counts/$cycle^2$ / 65,536

$Acceleration_{RW}$ = 582.1 cm/sec^2

4.7 Developer Kit Hardware Contents

The table below lists the hardware components and accessories provided with the Portescap N-Series ION DK, part number DK431S0056/06 F36:

Name	PMD P/N	Description
N-Series ION Stack		
N-Series ION unit	DD431S0056/06F36	Medium power three-phase Brushless DC motor serial interface N-Series ION unit
Serial DK PCB	PCB-1047P-41	Populated four-layer interconnect PCB for serial host interface N-Series ION DKs
Base plate	NION-DKH-01	Black anodized aluminum N-Series ION DK base plate
Mounting screws	N/A	Four M2.5 x 0.45 mm thread, 8 mm long, button head screws
Thermal pad	NION-DKH-02	10 mil (.25 mm) thick thermal pad cut to dimensions 33.5 mm by 33.5 mm
Other Cables & Components		
3-pin programming cable	Cable-USB-3P	USB to 3-pin programming cable for communicating with N-Series ION
DB9 serial cable	Cable-USB-DB9	Male DB9 to USB serial converter cable

4.7.1 Additional DK Accessories

The table below lists additional components that may be useful when working with N-Series ION DKs.

Name	Vendor & P/N	Description
Dual serial channel cable	PMD, P/N: Cable-4355-01.R	Male DB9 dual serial to two single-channel female DB9 cable. See Section 3.1, "Activating Serial Communications," for more information on the use of this cable.

4.8 Selected Cable & Accessory Specifications

PMD Part #: Cable-4355-01.R

Description: Male DB9 dual serial to two single-channel female DB9 cable

Length: 5 ft (1.5 m)

Notes: Bifurcated shielded cable with male DB9 (P1 in wiring table) splitting to female DB9 carrying Serial1 channel (Srl1 in table) and female DB9 carrying Serial2 channel (Srl2 in table).

P1 DB9 Pin#	P1 DB9 Signal Name	Srl1 DB9 Pin#	Srl2 DB9 Pin#
1	N.C.*	N.C.	N.C.
2	Srl1Xmt	2	N.C.
3	Srl1Rcv	3	N.C.
4	N.C.	N.C.	N.C.
5	GND	5	5
6	N.C.	N.C.	N.C.
7	Srl2Rcv	N.C.	3
8	Srl2Xmt	N.C.	2
9	N.C.	N.C.	N.C.
Shield	Shield	Shield	Shield

**N.C. = No Connection*

4.9 Conversion Factors, Defaults and Limits

To correctly control various N-Series ION features it may be helpful to know certain drive-specific scale factors. The following tables summarize these values.

4.9.1 Conversion Factors, Medium Power Units

The following table provides electrical conversion factors for medium power level N-Series ION units (P/Ns: DD4X1X0056/06).

These factors convert various integer Magellan IC command arguments (referred to as having units of counts) to physical quantities such as amperage, volts, etc. For more information on the Magellan motion control IC refer to the

Magellan Motion Control IC User Guide. For more information on C-Motion commands refer to the *C-Motion/PRP II Programming Reference*.

Unit	Example C-Motion Commands	Scaling	Example Usage
Amps	GetCurrentLoopValue	.733 mA/count*	A command request to read the ActualCurrent parameter returns a value 12,345. This corresponds to a current of 12,345 counts * 0.733 mA/count = 9.049
Volts	SetDriveFaultParameter GetBusVoltage	10.0 mV/count	To set an overvoltage threshold of 50V, the command value should be 50 V *1,000 mV/V / 10.0 mV/count = 5,000
Temperature	SetDriveFaultParameter GetTemperature	.0039 °C/count	To set an overtemperature threshold of 65 °C, the command value should be 65 °C / .0039 °C/count = 16,667
I^2t continuous current	SetCurrent	.733 mA/count	To set an I^2t continuous current of 4.0A, the command value should be 4.0A * 1,000 mA/A / 0.733 mA/count = 5,457
I^2t energy	SetCurrent	.0590 A^2Sec/count	To set an I^2t energy of 20.0 A^2Sec, the command value should be 20.0 A^2Sec / 0.0590 A^2Sec/count = 339

4.9.2 Defaults & Limits, Medium Power Units

The following table provides default values, medium limits and high limits for various specifiable drive-related parameters for low power level N-Series ION units (P/Ns: DD4X1X0056/06).

Setting	Default Setting	Low Limit	High Limit
Overtemperature limit	75.0 °C	0 °C	75.0 °C
Overtemperature hysteresis	5.0 °C	0 °C	25.0 °C
Overvoltage limit	60.0 V	10.0 V	60.0 V
Undervoltage limit	10.0 V	10.0 V	56.0 V
I^2t continuous current limit, Brushless DC motor	5.5 A	0.0 A	5.5 A
I^2t continuous current limit, DC Brush motor	7.1 A	0.0 A	7.1 A
I^2t continuous current limit, step motor	5.0 A	0.0 A	5.0 A
I^2t energy limit, Brushless DC Motor	25.1 A^2sec	0.0 A^2sec	25.1 A^2sec
I^2t energy limit, DC Brush Motor	28.1 A^2sec	0.0 A^2sec	28.1 A^2sec
I^2t energy limit, step motor	28.2 A^2sec	0.0 A^2sec	28.2 A^2sec

4.10 Component Developer Kit Assembly

4.10.1 Overview

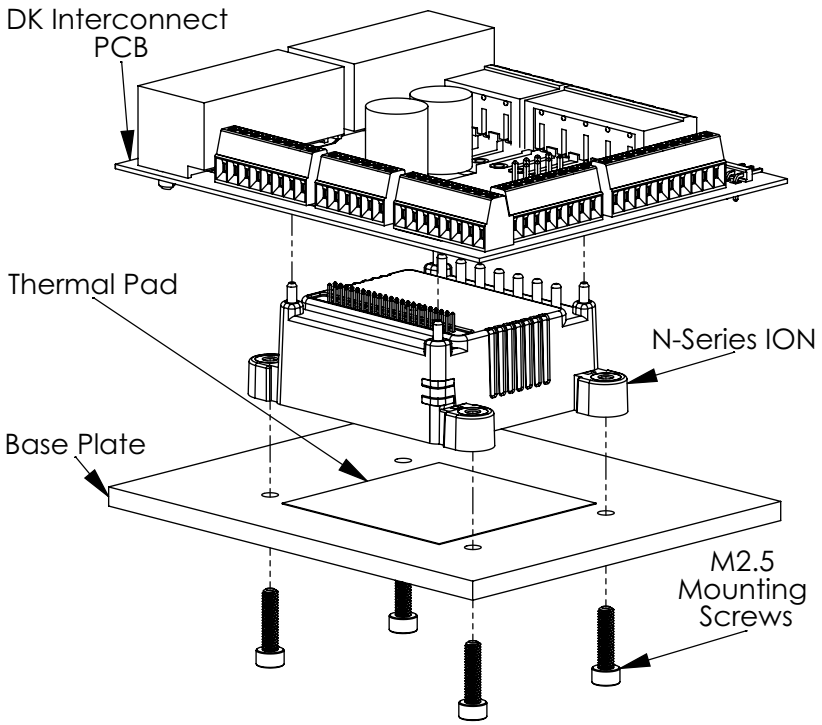


Figure 4-1: N-Series ION DK Showing Component Stack

When using non pre-assembled N-Series ION DKs assembly of the N-Series ION with the DK components is required. Along with the N-Series ION unit, which is not included with non pre-assembled DKs and must be ordered separately, the component stack elements are listed in the table below from top to bottom.

DK Stack Component Name	Description
DK Interconnect PCB	One of three possible PCBs that is soldered to the N-Series ION and provides convenient connectors to access the ION functions. The ION unit interface type (serial, CAN/SPI, or Ethernet) must be matched with the associated DK interconnect PCB type. For more information on this and exact P/Ns refer to Section 4.7, "Developer Kit Hardware Contents."
N-Series ION unit	One of 36 different N-Series ION units. As noted above the ION unit type must match the PCB interconnect PCB unit type
Thermal pad	33.5 mm x 33.5 mm x .25 mm thermal pad that insures correct thermal flow between the ION unit and the included base plate or application mounting surface
Mounting screws	Four M2.5 mounting screws
Base plate	Black anodized aluminum mounting plate which functions both as a heat sink and as a platform support for bench-top exercising of the DK. The bottom of the plate has four rubber feet for stability.

The next few sections will provide detailed instructions for assembling a complete DK stack from the components listed above. For additional information on mechanical mounting of the N-Series ION unit refer to Section 4.12, "Mechanical Mounting." For additional information on DK P/Ns and contents refer to Section 4.7, "Developer Kit Hardware Contents."

The N-Series ION contains electronic components and is therefore subject to damage from ESD (Electro Static Discharge). Therefore, during the assembly procedures described below and at all other times while handling the ION unit the operator should apply measures to reduce the potential for ESD damage which include use of grounding straps, anti-static mats, environmental controls, and other measures.

4.10.2 Assembly Sequence

Attaching the thermal pad to the N-Series ION

The first step of the assembly procedure is to insure that there is good thermal contact between the N-Series ION and the base plate heat sink. For this purpose, a thermal pad will be attached to the ION's metallic heat sink interface.

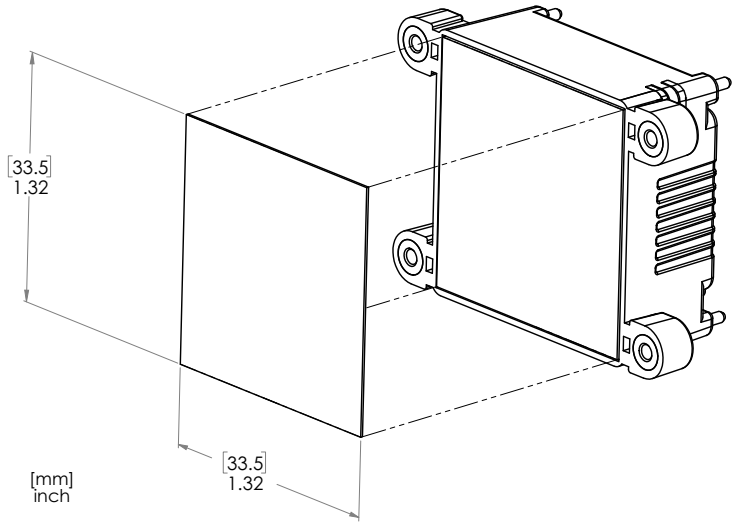


Figure 4-2: Mounting Thermal Pad to N-Series ION

To accomplish this, locate the thermal pad in the developer kit box. Next, carefully remove the thin plastic protective sheets on either side of the thermal pad and mount onto the ION unit, carefully aligning the pad with the ION unit's metallic backing and applying finger pressure to adhere the pads to the metal.

After adherence to the ION's metallic backing the thermal pad should appear flat and without bubbles or bulges. Once pressed in place the pad should stay in place but if required the pad can be removed and remounted.

Mounting the N-Series ION to the base plate

The next step of the assembly process is to mount the N-Series ION with thermal pad attached to the rectangular base plate. Four M2.5 screws and the Allen key are used for this purpose as shown in Figure 4-1. Make sure to orient the base plate so that the rubber feet are away from the ION unit.

When tightening the screws use only modest torquing force, and alternate tightening in an X pattern so that the applied force is increased slowly and equally amongst the four screws. Should you hear or see a snap of the ION unit tab while tightening a screw this means the ION unit's protective fuse tab has engaged because too much torque was applied. If this occurs the N-Series ION unit can not be used. It must be de-installed and discarded, and a new unit installed.

A fuse break event serves as a signal to the operator to revise their procedure so that the tab mounting torque limits are not exceeded during future attachment procedures. For detailed mounting torque specifications refer to Section 4.12.2.1, "Installation & Torque Limits."

Soldering the N-Series ION to the DK PCB

The final step of the DK stack assembly process is to solder the DK PCB to the N-Series ION. To accomplish this place the assembled base plate and N-Series ION onto a level surface with the base plate resting on its rubber pads.

With the PCB oriented so that the connector-side faces up, locate the PCB in the correct mounting position taking note of the location of the hole patterns in the PCB which mate with the N-Series ION. When ready gently push the PCB down onto the ION unit until the bottom of the PCB sits flush with the ION unit's PCB interface surface.

Altogether you will solder three different groups of pins; the four alignment posts located at the corners of the ION unit, the 44 signal connector pins, and the 7 power connector pins. Carefully solder each of these pins into the PCB taking care to insure proper solder flow so that good electrical and mechanical contact is made.

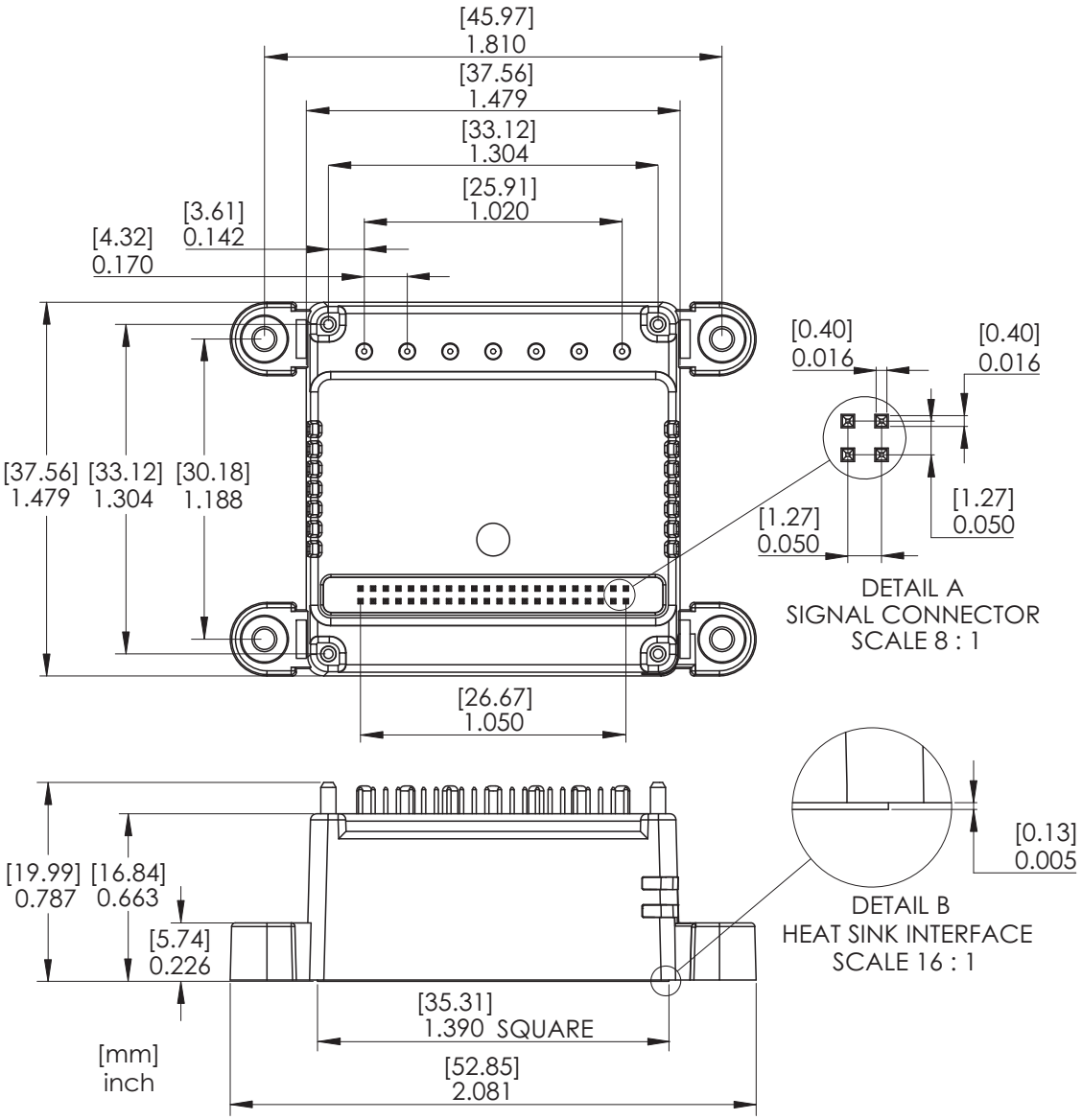
Congratulations! Once soldering has been completed assembly of your N-Series ION developer kit is complete and the DK is ready for operation.

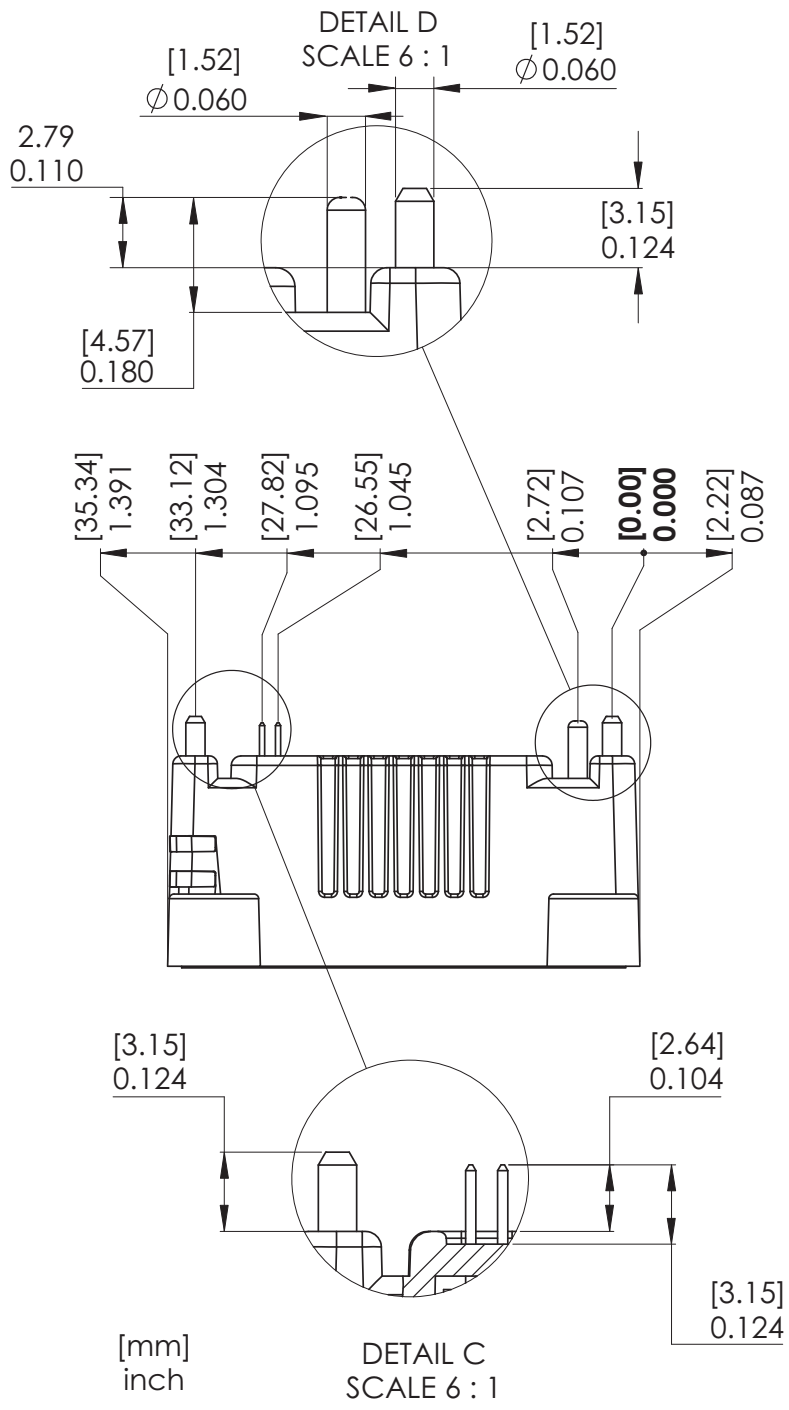
Take special care to insure proper solder flow occurs for the seven power connector pins. These larger pins may require more time to heat or a higher soldering iron temperature setting. To aid confirmation of a proper solder joint the area around the ION unit's power pins is recessed, allowing the technician visual access of the power pins from the underside of the PCB.

4.11 Physical Dimensions

4.11.1 N-Series ION Unit

Figure 4-3: ION/CME N-Series Digital Drive Physical Dimensions




DETAIL D
SCALE 6 : 1
[1.52]
∅0.060
[1.52]
∅0.060
2.79
0.110
[3.15]
0.124
[4.57]
0.180
[35.34]
1.391
[33.12]
1.304
[27.82]
1.095
[26.55]
1.045
[2.72]
0.107
[0.00]
0.000
[2.22]
0.087
[3.15]
0.124
[2.64]
0.104
[3.15]
0.124
[mm]
inch
DETAIL C
SCALE 6 : 1

4.11.2 Developer Kit

Figure 4-4: DK Base Plate Dimensions

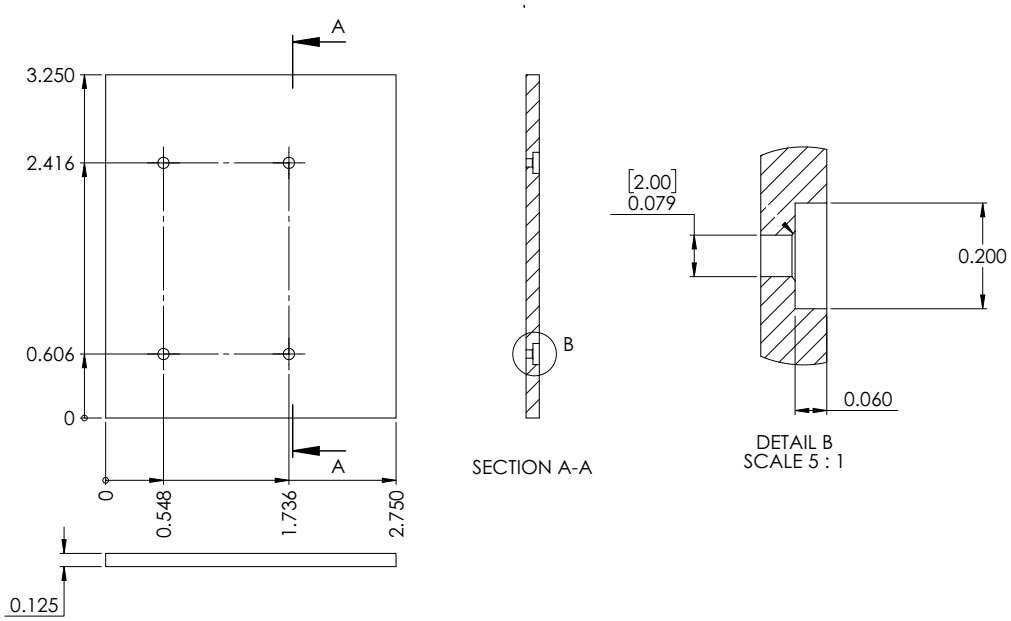
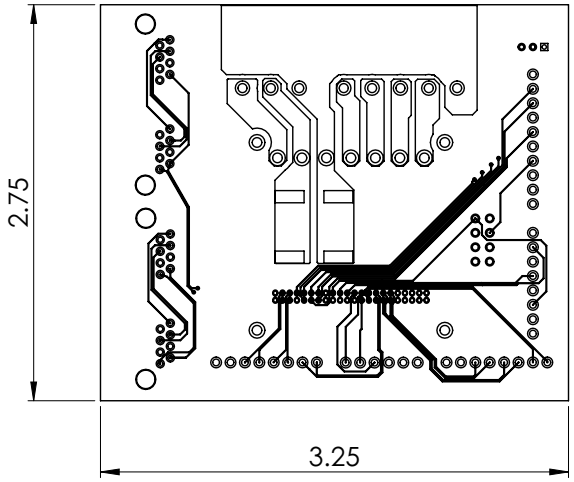


Figure 4-5: DK Interconnect Board Dimensions



4.12 Mechanical Mounting

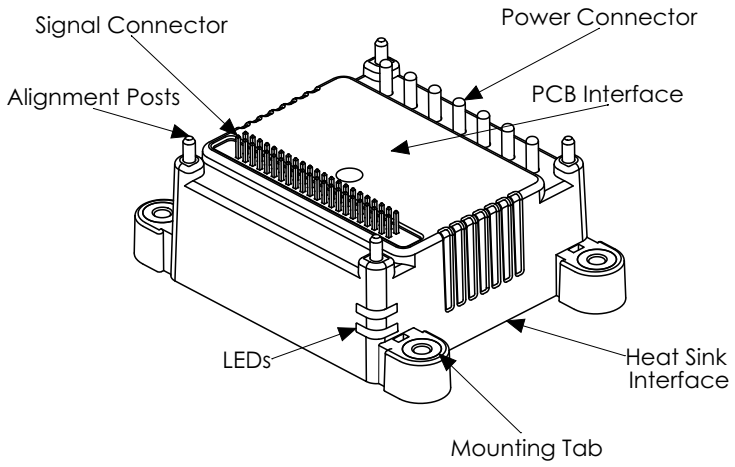


Figure 4-6: Mechanical Elements of N-Series ION Drive

The N-Series ION Drives are designed to be used in a wide variety of operating conditions. As shown in Figure 4-6 they have a robust mechanical design that allows rigid mechanical attachment at the two primary interface points; the interface between the N-Series ION and the user PCB, and the interface between the N-Series ION and the heat sink or cold plate also called a supporting plate.

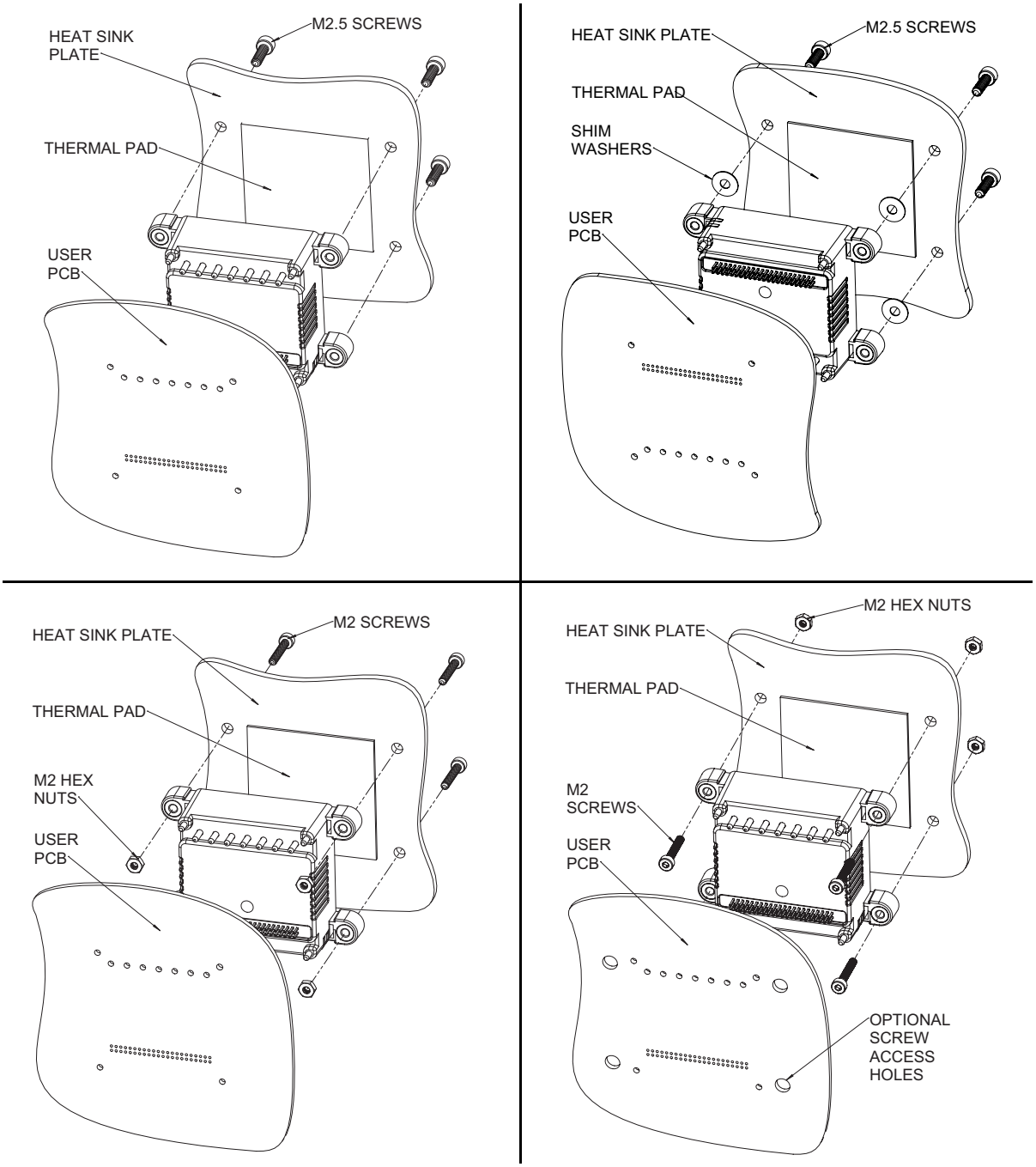
The following sections describe how mechanical attachment at both these interfaces occurs. There is an alternate mechanical configuration of the N-Series ION which does not have mounting tabs and which may be useful for some applications. See Section 4.12.4, "Tabless N-Series IONs," for more information on this mechanical configuration.

4.12.1 Attachment to the User PCB

Attachment at the PCB is accomplished by soldering the PCB to the N-Series ION's four mechanical alignment posts. While these posts do not serve an electrical function, they provide a rigid attachment of the PCB to the N-Series ION, and thereby reduce strain that may occur in the solder connections between the user PCB and the Power and Signal Connectors of the N-Series ION. Use of the four PCB alignment posts are recommended for all applications.

4.12.2 Attachment to the Heat Sink

Figure 4-7: N-Series ION Mounting Options



Attachment to the heat sink or supporting plate is accomplished via screws at the N-Series ION's four mounting tabs. As shown in Figure 4-7 there are four typical approaches to mechanically attaching to the heat sink.

The first is mounting from the heat sink side. In this approach M2.5 screws are installed directly into the ION's tab which is threaded for a M2.5 screw. A variation of this option may be used when greater mounting rigidity is desired, typically because the application environment has higher g forces. This mounting method locates shim washers under the tabs to limit tab deflection and to increase the amount of torque that can be applied to tighten down the tabs.

When using this method the thickness of the shim washer should be determined by the type and thickness of thermal transfer material used to create a vibration resistant mount, while maintaining optimal thermal transfer. Note that in this approach, the N-Series ION's mounting tab mechanism, which normally acts to limit the maximum force that can be applied via the tabs, is effectively bypassed. Therefore it is up to the user to ensure that the force applied to the ION unit's heat sink interface, the limits for which are provided in the table below, are not exceeded. For more information on PMD's patented mounting tab design see Section 4.12.2.2, "N-Series ION Mounting Tab Design."

To achieve the shim function using washers, the recommended dimensions are .280" (7.0 mm) outer diameter and .105" (2.5 mm) inner diameter. Representative suppliers for washers such as this include Bokers, Superior Washer, and Phoenix Specialty.

When using shim washers or other approaches that raise the mounting interface under the N-Series ION tabs, the ION unit's force-limiting mounting tab design is effectively disabled. It is therefore up to the user to ensure that the mounting force limit on the ION unit's heat sink interface plate is not exceeded.

Two other mounting options use a nut and a screw. Most commonly M2 screws are used which go through the tab's M2.5 thread without engaging the thread, with the nut used to capture the M2 screw. The two possible orientations of the nut and screw represent these two different mounting options. Note that if the preferred mounting orientation is to have the screw heads facing the user PCB, it may be desirable to have holes in the PCB to give access to the ION screw hardware. However if the ION is mechanically mounted first, and then soldered onto the user PCB, these access holes may not be necessary.

4.12.2.1 Installation & Torque Limits

Before mating the N-Series ION to the heat sink, in a typical installation a thermal transfer material will first be installed to enhance thermal contact. Refer to Section 4.12.2.3, "Thermal Transfer Materials," for more information on selection of these materials.

Once this material is in place the tabs can be tightened under careful torque limit control. The table below shows the recommended maximum torque for both M2 and M2.5 screws along with equivalent force values if an alternate attachment method such as a clamp is used, or if shim washers are used as described above.

Type	Minimum	Recommended	Maximum
M2 screw torque	0.082 N-m (0.722 in-lb)	0.094 N-m (0.836 in-lb)	0.113 N-m (1.00 in-lb)
M2.5 screw torque	0.113 N-m (1.00 in-lb)	0.130 N-m (1.15 in-lb)	0.147 N-m (1.30 in-lb)
Force	25 kgs (55 lbs)	29 kgs (68 lbs)	33 kgs (71 lbs)

Exceeding the N-Series ION's mechanical attachment specifications indicated above may cause a failure of the drive unit at the time of installation, or at a later time in the field.

4.12.2.2 N-Series ION Mounting Tab Design

N-Series IONs utilize a patented mounting tab design to reduce the probability that overtightening of the tab attachment hardware will cause damage to the N-Series ION unit.

This is achieved by the mounting tab deliberately breaking (snapping) under conditions of over tightening. Such a protective 'fuse' break event (snap) of the tab should be audible to the operator during tightening. While an N-Series ION unit that has undergone a fuse break event during tightening is no longer usable, it serves as a signal to the

operator to check and revise their torque control procedure so that the tab mounting torque limits are not exceeded during future attachment procedures.

An ION/CME N-Series Digital Drive which has undergone a mounting tab fuse break event during installation must be de-installed and discarded. It is not possible to repair or to otherwise use an N-Series ION unit which has undergone a fuse break.

The N-Series ION's mounting tab fuse break functions when there is an air gap between the bottom of the tab and the surface that it is mounted to. If there is no air gap, or if the air gap is too small then the fuse/snap function may not occur even though the recommended mounting torque limit may have been exceeded. In all cases it is the responsibility of the user to determine that the recommended mounting torque limits or force indicated in Section 4.12.2.1, "Installation & Torque Limits," are not exceeded.

4.12.2.3 Thermal Transfer Materials

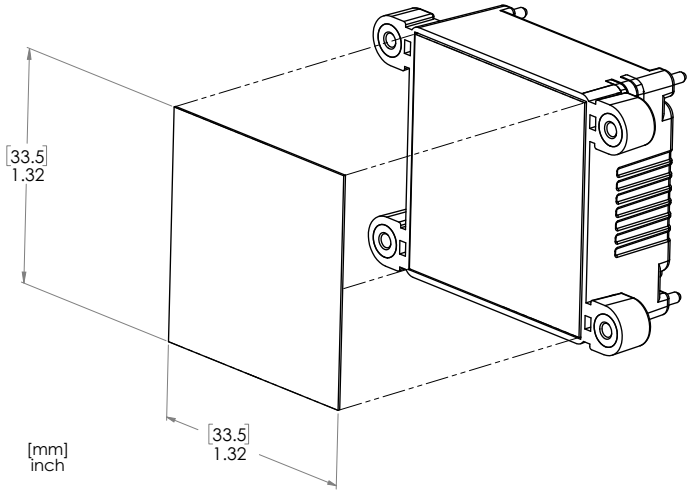


Figure 4-8: Recommended N-Series ION Thermal Transfer Material Dimensions

Thermal transfer materials in the form of thermal tape, pads, paste, or epoxy may be used to improve thermal transfer between the N-Series ION's metal plate and an attached heat sink or supporting plate. These materials improve thermal conductivity by filling in air gaps that form when two metallic surfaces are mated.

Figure 4-8 shows a typical application of a thermal transfer material between the drive unit and a heat-removing metal surface. The following guidelines may be helpful in selecting and sizing the thermal transfer material best-suited to your application.

The capacity of thermal transfer materials to transfer heat (known as the bulk conductivity) is much lower than that of metals such as aluminum or copper. Therefore, in general, the thinner the transfer material the better. Thickness of the material is only precisely controllable for thermal pads and thermal tapes, with thermal pads providing the thinnest available interfaces beginning at 5 mils (.127 mm) or even less. For use with N-Series IONs thermal transfer materials that are thicker than 25 mils (0.7 mm) are not recommended regardless of the material used.

When using thermal paste or thermal epoxy glue the thickness should be carefully controlled via a silk screen or other wet film application process. The N-Series unit itself should not be used to squeeze non-uniformly applied paste or epoxy flat during installation. Doing so may result in damage to the drive unit.

Whether using tape, pads, paste, or epoxy, as shown in Figure 4-8, the thermal transfer material that is used as the interface should not extend to the area under the N-Series ION's tabs because this may reduce the amount of compression that occurs in the thermal transfer area. The following table provides dimensions for the applied thermal transfer material for N-Series IONs:

Parameter	Maximum Pad Dimensions
Value	33.5 mm x 33.5 mm (1.32" x 1.32")

4.12.3 Recommended Mounting Procedures

There are a number of additional precautions and procedures that should be followed to maintain the electrical and mechanical integrity of the N-Series ION unit during installation.

Soldering N-Series ION units in place. Applications that involve N-Series ION units mated to a supporting plate should take special care to insure that the solder joints are not stressed by the supporting plate once installed. The recommended method to achieve this is to mechanically mate the unit to the supporting plate before soldering into the PCB. If, for whatever reason, this is not possible, then special care should be taken to insure that the N-Series ION is aligned with the supporting plate after soldering and before mechanical attachment so that upon mechanical attachment no stress is placed on the ION unit, the solder contacts, or the PCB.

Mounting surface flat and clean. Thermal performance as well as safe operation of the N-Series ION requires that the surface that it is mounted to be flat and clean, free of dust, grease, or foreign objects. The recommended maximum deviation of the mating surface flatness is 3 mils (.076 mm).

Progressive tightening. N-Series ION units that are mated to a heat sink or supporting plate should be attached by progressively tightening each of the unit's four tabs. This means that one screw may be tightened, followed by the others, than back to the first etc. until the desired torque at each screw has been achieved. Following this procedure is particularly important when installing N-Series ION units over paste or epoxy, where the subsurface layer will undergo compression and movement before settling to a final installed position.

It is the responsibility of the user to ensure that all N-Series ION units have been installed within the prescribed mechanical stress limits and following the above described procedures. Failure to observe any of the above recommended procedures and limits may result in incorrect operation or failure of the unit during operation.

4.12.4 Tabless N-Series IONs

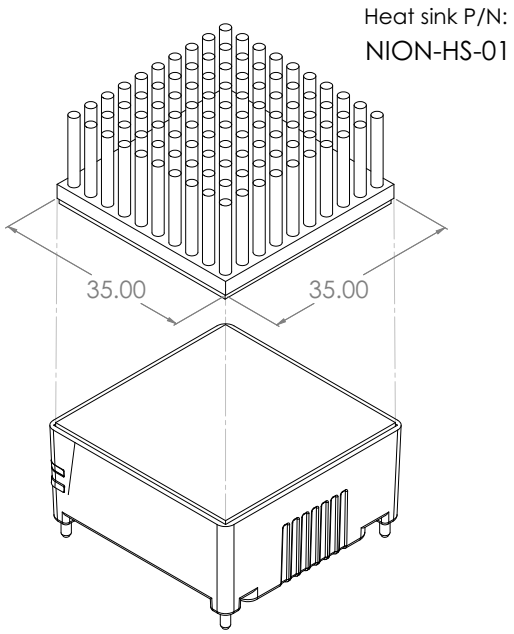


Figure 4-9: Tabless N-Series ION and Heat Sink

In addition to the standard mechanical configuration of the N-Series ION, which uses mounting tabs for mechanical attachment to the heat sink, an alternate configuration of the N-Series ION is available which does not have mounting tabs. This configuration along with a typical heat sink arrangement is shown above.

The primary benefit of this alternate mechanical configuration is that it allows N-Series ION units to be packaged more tightly on the PCB. The figure below gives an example of this, showing a compact three-axis N-Series ION-based control board.

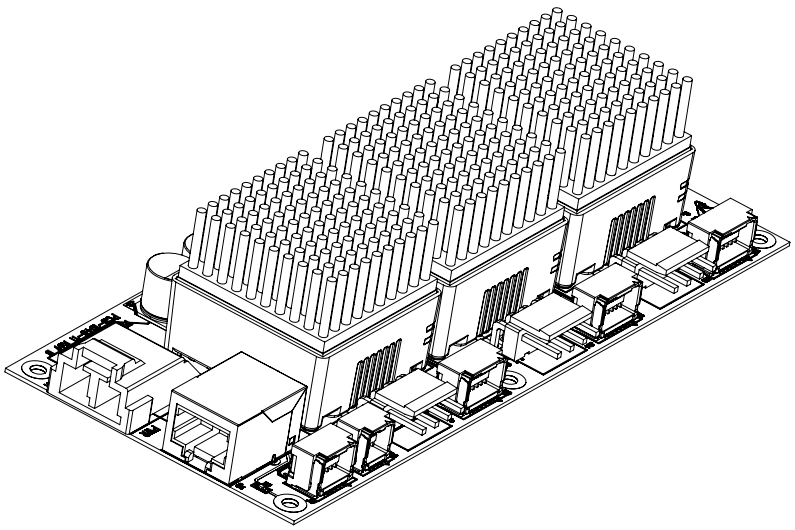


Figure 4-10: Example Three-Axis PCB Using Tabless N-Series IONs

4.12.4.1 Air-Cooled Heat Sink

Most often when the tabless N-Series ION is used and when heat sinking is needed air-cooled heat sinks are used. These heat sinks are typically attached to the N-Series ION unit via an adhesive thermal pad or via thermal epoxy.

For more information on how to order the tabless N-Series ION and available heat sinks contact your local PMD representative.

Due to the risk of excessive mechanical stress being placed on the user PCB or the N-Series ION the tabless N-Series ION should not be used in systems that experience significant vibration or acceleration. This is especially true if an air-cooled heat sink is attached to the N-Series ION unit. For these operating environments the regular N-Series ION, which provides the ability to implement much more robust methods of mechanical mounting, should instead be used. Regardless of the N-Series ION type or mounting method it is the responsibility of the user to ensure that the resultant controller can be operated safely under the mechanical conditions the controller is expected to experience.

4.13 LEDs

N-Series IONs have two LEDs for indicating the status of the unit, one green and one red. The green LED is called the power LED and is solid on when the unit is powered and the ***Enable*** signal is active (low). If the unit is powered but the ***Enable*** signal is inactive (high) this LED will blink. If the unit is not powered this LED is off.

The red LED is called the status LED and is normally off indicating no errors. There are three additional states of this LED consisting of fast blink (three times per second), slow blink (once per second), and solid on. The table below summarizes the above:

Green (Power) LED	Red (Status) LED	Condition
off	off	Unit not powered
solid on	off	Unit Enabled (signal level low)
blinking slow	off	Unit Disabled (Enable signal level high)
solid on	blinking fast	Unit experienced an overvoltage or undervoltage condition
solid on	blinking slow	Unit experienced an overtemperature or I^2t overcurrent condition
solid on	solid on	Unit experienced a hard fault condition

For more information on overvoltage, undervoltage, overtemperature, I^2t overcurrent, and overcurrent conditions including suggested recovery procedures refer to the *ION/CME N-Series Digital Drive User Manual*. Note that the LED states for these conditions show the latched condition. That is, the LED status will not return to normal until the Magellan IC receives a **ResetEventStatus** command with the condition no longer present.

If operating in its default LED display mode any activation of the red LED indicates a potentially serious fault in the drive or in the system being controlled. It is the responsibility of the user to determine the correct and safe recovery procedure for such a condition.

This page intentionally left blank.

5. DK Interconnect Board Schematics

In This Chapter

5.1 DK Interconnect Board Schematics

The schematic on the next page is the design for the ION/CME N-Series Digital Drive Developer Kit's printed circuit board serial host interface.

As the N-series ION is designed to limit the requirement for external components, the interconnect boards simply provide connectors to all pins on the N-series ION, similar to a breakout board. Aside from host communication pins, which use common connectors, all signals are accessible through screw terminals.

5.1.1 Serial Host Interface Type

Figure 5-1: Serial Host Interface DK Interconnect Board Schematic