

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Precision MicroControl MFX-PCI1440* at our website: *Click HERE*

MultiFlex PCI
1000 Series

PCI Motion Controller - User's Manual

Revision 2.5

Precision MicroControl Corporation
2075-N Corte del Nogal
Carlsbad, CA 92009 * USA

Tel: +1-760-930-0101
Fax: +1-760-930-0222

www.pmccorp.com

Information: info@pmccorp.com
Technical Support: support@pmccorp.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

LIMITED WARRANTY

All products manufactured by PRECISION MICROCONTROL CORPORATION are guaranteed to be free from defects in material and workmanship, for a period of **2 years from the date of shipment**. Liability is limited to FOB Factory repair, or replacement, of the product. Other products supplied as part of the system carry the warranty of the manufacturer.

PRECISION MICROCONTROL CORPORATION does not assume any liability for improper use or installation or consequential damage.

© Copyright Precision Micro Control Corporation, 2004-2012. All rights reserved.

Information in this document is subject to change without notice.

Intel is a registered trademark of Intel Corporation.
Microsoft® and Windows® are registered trademarks of Microsoft Corporation.
Acrobat® and Acrobat Reader® are registered trademarks of Adobe Corporation.

Precision MicroControl Corp.
2075-N Corte del Nogal
Carlsbad, CA 92011 • USA

Tel: +1-760-930-0101
Fax: +1-760-930-0222
Web: www.pmccorp.com
Email: info@pmccorp.com
support@pmccorp.com
sales@pmccorp.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table of Contents

Prologue **3**
Introduction **5**
Motion Control Primer **11**
Axis I/O 15
The Command Set - the heart of the motion controller 16
Executing Operations with MCCL 17
Closed loop, open loop, and position verification 20
Why does a servo need to be tuned? 22
Position Feedback - Quadrature Incremental Encoder 24
Servo Amplifiers: Current Mode versus Velocity Mode 25
Stepper Motors - Full Step versus Micro Step 26
Homing - Why, When, and How 27
Software, Programming and Utilities **29**
Controller Interface Types 30
Building Application Programs using Motion Control API 31
MCSpy - application program diagnostic tool 36
PMC Sample Programs 37
Motion Control API On-line Help 38
Motion Integrator 40
PMC Utilities 43
Connecting to the Controller **47**
+/- 10V Analog Servo Command Connections 48
PWM (Pulse Width Modulation) Command Connections 49
Pulse Command Connections 51
Amplifier / Driver Enable Connections - Low Active 52
Driver Disable Connections - Low Active 53
Amplifier / Driver Enable Connections - High Active 54
Amplifier / Driver Fault Connections 55
Differential Incremental Encoder Connections 56
Single Ended Incremental Encoder Connections 57
Over-Travel Limit Connections 58
Home Sensor Connections 60
TTL Digital Input Connections 61
TTL Digital Output Connections 62
A/D Input Connections wiring example 63
Watchdog Relay Connections 64
Motion Control **65**
Servo (analog command) Axis Setup 65
Tuning the Servo 68
Moving Servo Axes with Motor Mover 77
Stepper (pulse command) Axis Setup 78
Moving Stepper Axes with Motor Mover 85
Contour Motion (arcs and lines) 91
Electronic Gearing 100
Jogging 101

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Defining Motion Limits 102
Homing Axes 105
Motion Complete Indicators 117
On the Fly changes 119
Feed Forward (Velocity, Acceleration, Deceleration) 120
Save and Restore Axis Configuration Settings 122
Application Solutions 123
Backlash Compensation 123
Emergency Stop 125
Encoder Rollover 127
Flash Memory Firmware Update 128
Saving and Restoring Axis Configuration Settings 129
Learning/Teaching Points 133
Building MCCL Macro Sequences 135
MCCL Multi-Tasking 137
Position Capture 140
Position Compare 141
Position Verification of an Open Loop Pulse Axis 143
PWM Servo Command 147
Record Motion Data 150
Resetting the Controller 151
Single Stepping MCCL Programs 152
Torque Mode Output Control 154
Turning off Integral gain during a move 156
Defining User Units 159
Watchdog Circuit 163
General Purpose I/O 165
Digital I/O 165
Configuring and Exercising the Digital I/O 167
Using the Digital I/O 168
A/D Inputs 170
Specifications 173
Motion Control Board 173
Analog Command Axis Specifications 174
Pulse Command Axis Specifications 175
Connectors, I/O and Schematics 177
VHDCI Connectors 178
Controller Status LED Indicators 179
Controller Potentiometers 180
Connector Pinout – MultiFlex PCI 1440 181
Connector Pinout – MultiFlex PCI 1040 185
Signal Descriptions 190
Motor Command Signals 190
Encoder Feedback Signals 191
Default Axis Inputs 191
Default Axis Outputs 193
Default Configuration of General Purpose I/O 195
Circuit Schematics 197
Troubleshooting 201
Controller Error Codes 211
Motion Control API Error Codes 212
MCCL Error Codes 213
Glossary 215
Appendix 219
Default Axis Configuration Settings 219
Index 221

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Prologue

This document provides configuration, programming and application information for the ***MultiFlex PCI 1000 Series*** motion controllers. Documentation for this product line includes the following documents:

MultiFlex PCI 1000 Series Quick Start Guide
MultiFlex PCI 1000 Series User's Manual *(this document)*
Motion Control API (Application Programming Interface) Reference Manual
Motion Control Command Language (MCCL) Reference Manual

The latest versions of these documents can be downloaded from the Support section of PMC's web site at: *www.pmccorp.com/support/mfxpci1000.php*.

This user manual applies to all MultiFlex PCI 1000 Series models, which include the following:

Table 1. MultiFlex PCI 1000 Series Models

Model	Total Axes	Analog and/or PWM Axes (Servo)	Step/Dir or CW/CCW Pulse Axes (Stepper/Servo)	Encoder Channels (standard / optional)	Analog Inputs (optional)
MultiFlex PCI 1040	4	-	4	0 / 4	8
MultiFlex PCI 1400	4	4	-	4	8
MultiFlex PCI 1440	8	4	4	4 / 8	8
MultiFlex PCI 1802	8	8*	-	8	8

* PWM onlly

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Page intentionally
left blank

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Chapter 1

Introduction

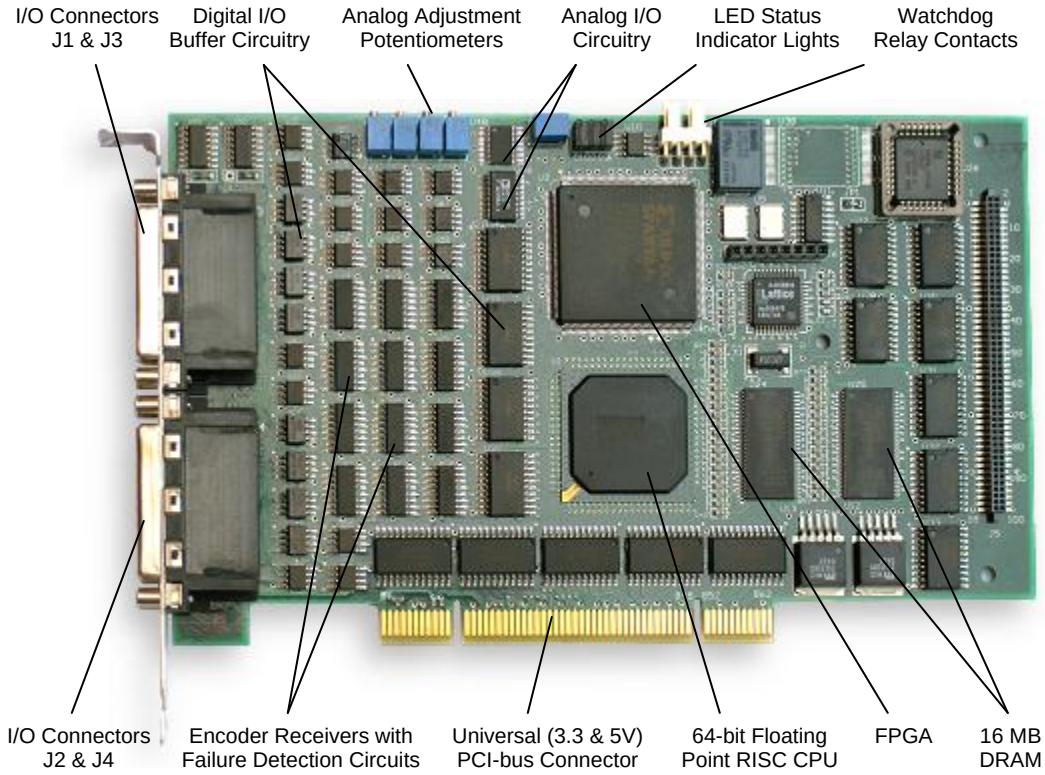


Figure 1. MultiFlex PCI 1000 Series Board Layout

The ***MultiFlex PCI 1000 Series*** are programmable, board-level, PCI-bus motion controllers designed for high-performance multi-axis control of servo and stepper motors and I/O. Features offered by models in this series include:

- Plug & play PCI universal (3.3 & 5V) half-length card
- 64-bit floating-point RISC CPU for high precision and dynamic range
- Customizable FPGA-based I/O architecture
- Up to 8 total axes
- 4 axes analog servo control (models MultiFlex PCI 1400, 1440)
- 8 axes PWM servo control (model MultiFlex PCI 1802)
- Up to 4 axes Step/Dir/CW/CCW pulse control (models MultiFlex PCI 1040, 1440)
- Coordinated motion - interpolation, contouring, spline, master/slave, gearing
- Trapezoidal, S-curve and parabolic velocity profiles

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- User selectable 2, 4 and 8 kHz servo update rate each axis
- 16-bit analog servo command outputs
- 20 mHz encoder inputs for high-speed, high-resolution moves
- 5 MHz step/direction/CW/CCW outputs for high-speed microstepping
- On-the-fly changes in trajectory, direction and PID values
- On-board multi-tasking - frees host PC for other tasks
- Eight general-purpose 14-bit A/D input channels (optional)
- Up to 60 user-assignable digital I/O channels
- Encoder-failure detection circuitry for improved machine safety
- Sub-microsecond position capture & compare I/O for rapid event triggering & synchronization
- Uses widely available, low-noise twisted-pair shielded SCSI cables for all I/O
- Programmable in C/C++/C#/.NET, Delphi, LabVIEW, VB and easy-to-use command language
- Drivers and example programs with source code for Windows and Linux
- Programming API and commands are compatible across all PMC motion controllers
- Graphical setup, tuning, diagnostic and example programs
- Custom features and performance enhancements are available upon request

Processor

MultiFlex PCI 1000 series motion controllers feature an advanced 64-bit floating-point MIPS CPU core coupled with PCI interface logic and internal cache memory to provide a powerful processing engine for high-performance motion control. An embedded multi-tasking real time kernel executes all motion control operations with 64 bit floating point precision. 16 MB of DRAM and 4 MB of non-volatile FLASH memory provide on-board memory space for executing both the intrinsic motion control code as well as user programs. A high-capacity FPGA interfaces directly with I/O such as encoders, analog inputs, control signals and general-purpose I/O and provides a great amount of flexibility for tailoring the controller to specific application and performance requirements.

PC computer minimum requirements

MultiFlex PCI 1000 series controllers can communicate with almost any Windows or Linux based computer equipped with a 32 bit PCI-bus slot. The controller's CPU executes motion functions independently of the host PC, so other than the minimum requirements for the selected operating environment, the controller does not require the use of any additional PC resources.

Programming

Windows and Linux programmers can create flexible and powerful control programs for PMC motion controllers in two ways:

1. Writing a high-level program (C/C++/C#/VB/Pascal/LabVIEW) that uses the functions supplied as part of PMC's ***Motion Control API*** (Application Programming Interface)

and/or

2. Using PMC's embedded multi-tasking ***Motion Control Command Language (MCCL)***

Programmers can use either, or both, programming methods to command and control the motion controller. For example, a multi-threaded C/C++/C# host application program can control and coordinate the execution of motion and I/O, while one or more embedded MCCL routines can run simultaneously as background tasks on the controller board.

PMC's ***WinControl*** terminal emulator utility (a Motion Control API component) provides a low-level command interface used to send MCCL commands and routines to the controller for immediate execution, or to download and save MCCL routines (also called "macros") to the controller for later

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

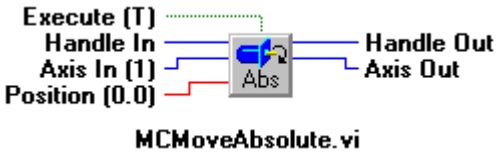
execution. Any MCCL command or routine can also be downloaded and called from a high-level program (C/C++/C#/VB etc) via the appropriate Motion Control API function libraries.

For additional information on Motion Control API and MCCL programming, please refer to the ***Motion Control API Reference Manual*** and the ***Motion Control Command Language (MCCL) Reference Manual*** which are both available for download at: *www.pmccorp.com/support/mfxpci1000.php*.

Motion Control API example

Function prototypes

C/C++ MCMoveAbsolute(HCTRLR hCtlr, WORD axis, double position);
C#/.NET Mcapi.Error Mcapi.MoveAbsolute(Int16 axis, Double position);
Delphi: procedure MCMoveAbsolute(hCtlr: HCTRLR; axis: Word; position: Double); stdcall;
VB: Sub MCMoveAbsolute(ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal position As Double)
LabVIEW:




```
//C/C++ Example, Move Axis 1 to position 85000
    MCMoveAbsolute( hCtlr, 1, 85000.0 );
```

MCCL command example

MA **M**ove **A**bsolute
MCCL command : *a*MA*n* *a* = Axis number *n* = integer or real
applies to: Analog Command Axis, Pulse Command Axis
see also: MR, PM

These MCCL command sequences cause the motion controller to execute a move to absolute position n.

```
;Example:
1MA1000                  ;Axis 1 move to position 1000
2MA-25000                ;Axis 2 move to position -25000
```

The following example illustrates MCCL multi-tasking. This example shows how a user would monitor the state of a digital input while an axis is moving in order to 'automatically' stop the axis as soon as the input is activated:

```
;Example:
AL0,AR10                 ;define user register 10 as input #4 active
AL0,AR100                ;define user register #100 as background task
MD100,IN4,MJ101,NO,1JR-3 ;jump to macro 101 when digital input #4 turns on
MD101,1ST,1WS.05         ; macro 101 defined, stop axis #1.
AL1,AR10,ET@100          ;terminate background task
```

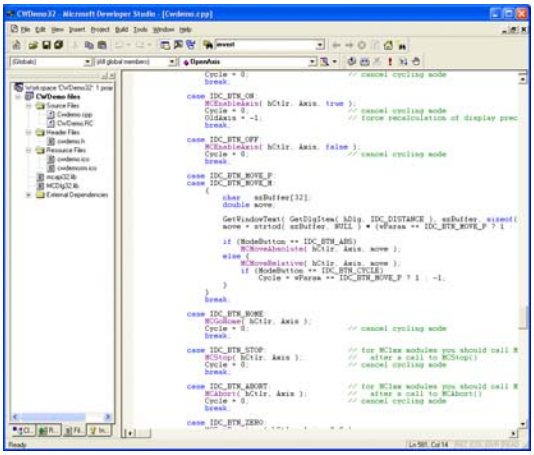


There is not necessarily a one-to-one relationship between each Motion Control API function and eacn MCCL command. Although any MCCL command can be called directly using the Motion Control API's pmccmdex() function, most Motion Control API functions ease the task of programming by encapsulating additional functionality within each function.

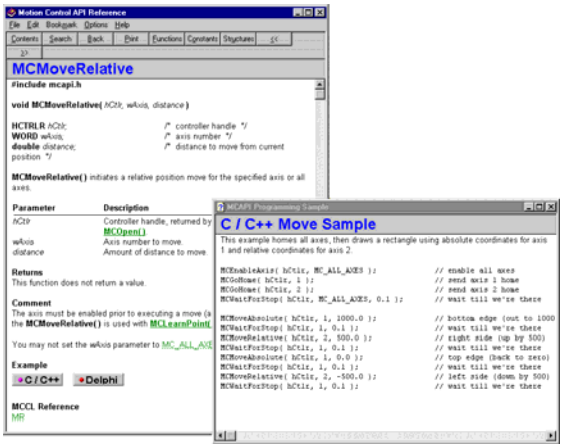
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Programming Tools

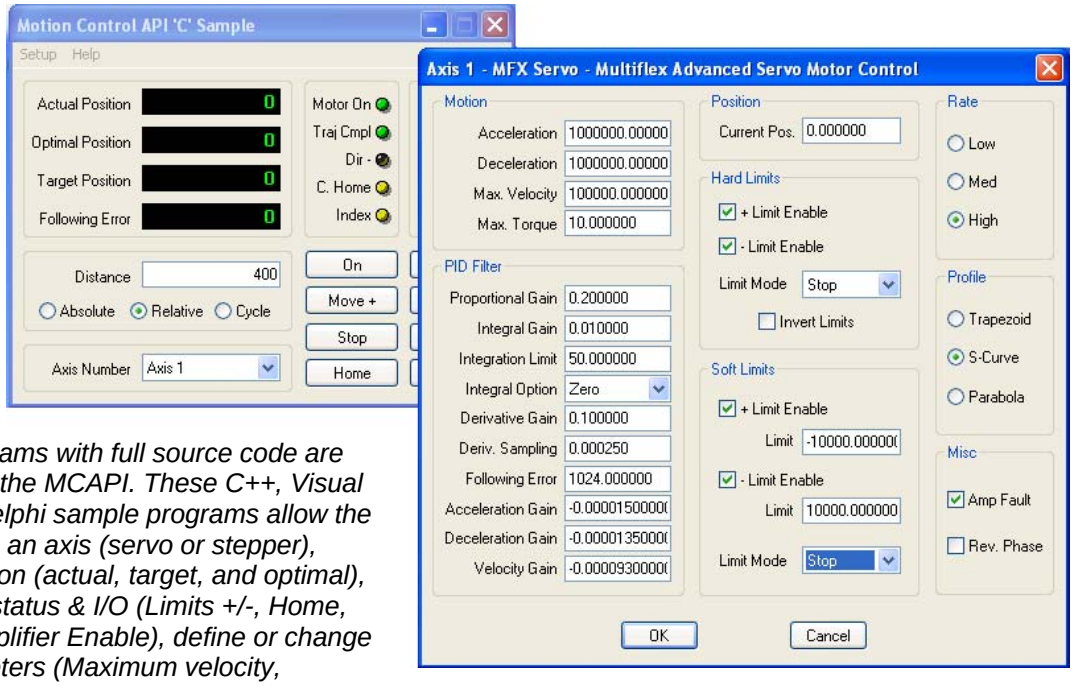
PMC's Motion Control API provides programmers with a comprehensive function library.



Develop application programs with Visual C/C++/.NET



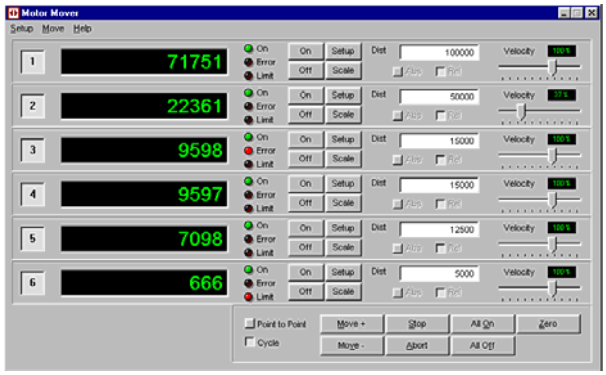
Comprehensive on-line help provides detailed function descriptions and program samples



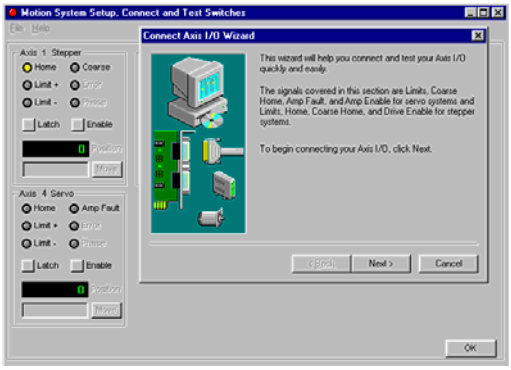
Sample programs with full source code are supplied with the MCAPI. These C++, Visual Basic, and Delphi sample programs allow the user to; move an axis (servo or stepper), monitor position (actual, target, and optimal), monitor axis status & I/O (Limits +/-, Home, Index, an Amplifier Enable), define or change move parameters (Maximum velocity, acceleration/deceleration), Define or change the servo PID parameters

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

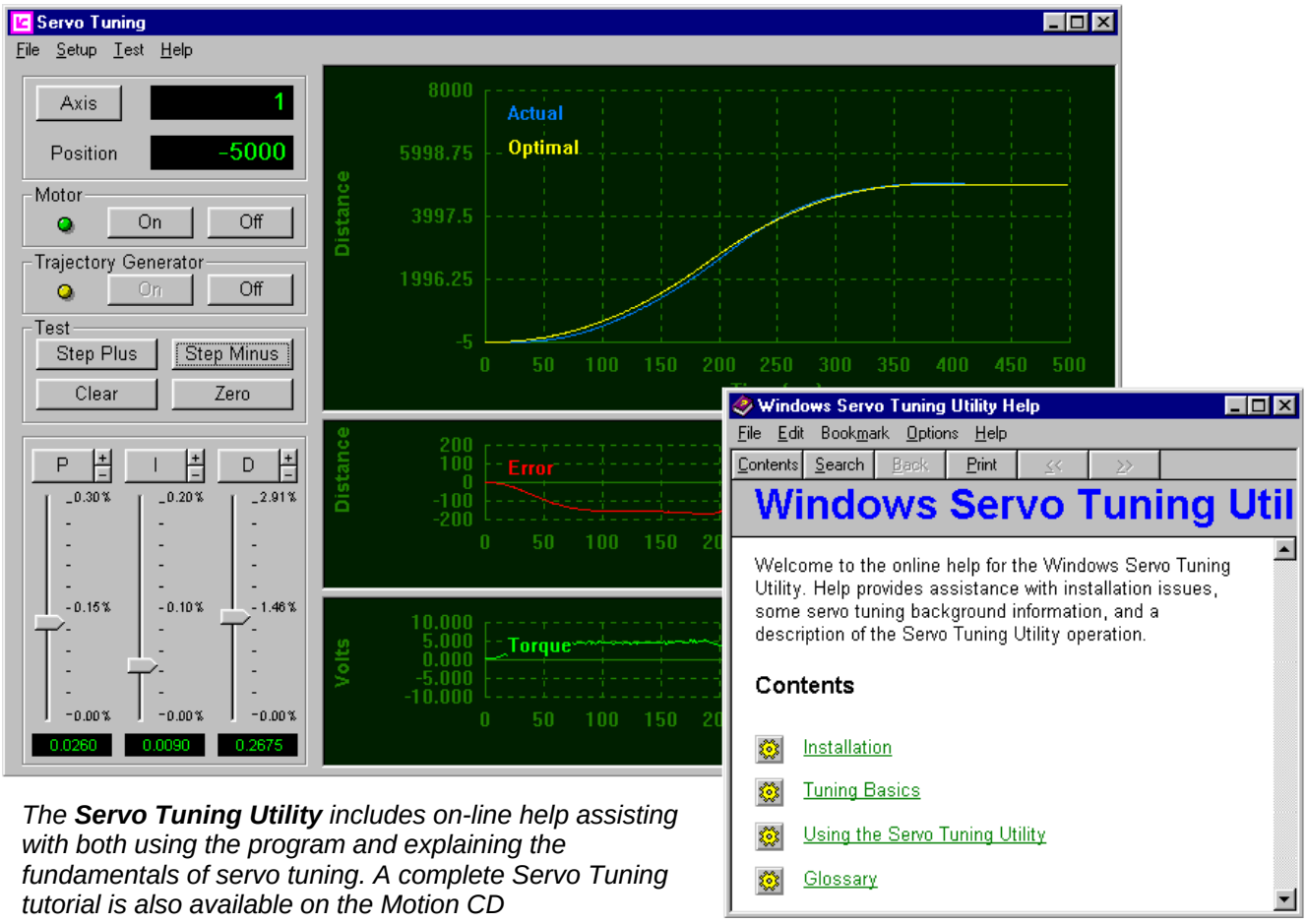
Software Tools & Utilities



*PMC's **Motor Mover** allows users to: move any or all motors, change velocities on the fly, define cycling routines, monitor position and status*



*PMC's **Motion Integrator's** Setup Wizards walk the user through the integration process with external components (motors, encoders, sensors)*



*The **Servo Tuning Utility** includes on-line help assisting with both using the program and explaining the fundamentals of servo tuning. A complete Servo Tuning tutorial is also available on the Motion CD*

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

I/O Configuration Panel

PMC's **I/O Configuration Panel** (accessible from Windows Control Panel) allows users to re-configure the channel numbers and logical functions of the digital I/O. The flexibility provided by this unique feature allows more efficient use of I/O resources and eases the task of connecting the controller to external devices.

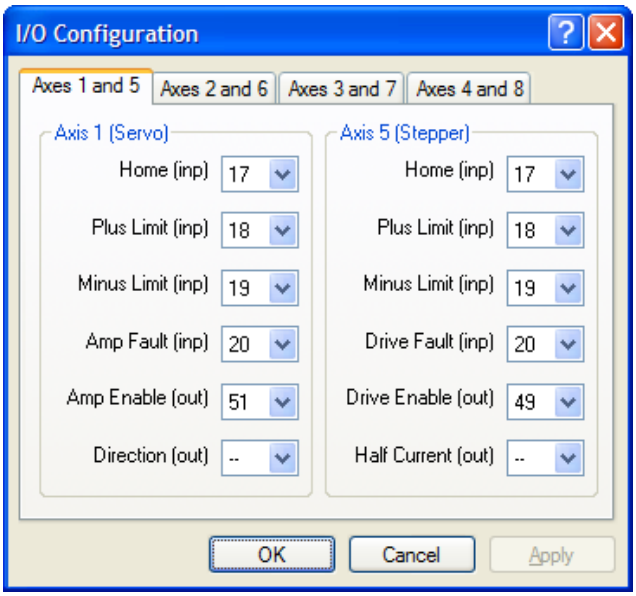


Figure 2. Digital I/O Configuration Panel

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Chapter 2

Motion Control Primer

Motion Control Architecture

A typical PC-based multi-axis motion control system is comprised of :

- A programmable motion controller
- A user interface from which to program, command and monitor the motion controller
- Two or more servo or stepper motors
- An amplifier/driver for each motor
- A position feedback device for servo motors or closed-loop stepper motors
- End of travel sensors (or limit switches) for axes with linear travel
- A mechanical stage and load. In this illustration, a stage is mounted on bearings and a lead screw is coupled to the motor shaft. When the motor shaft rotates, the stage moves along the lead screw.

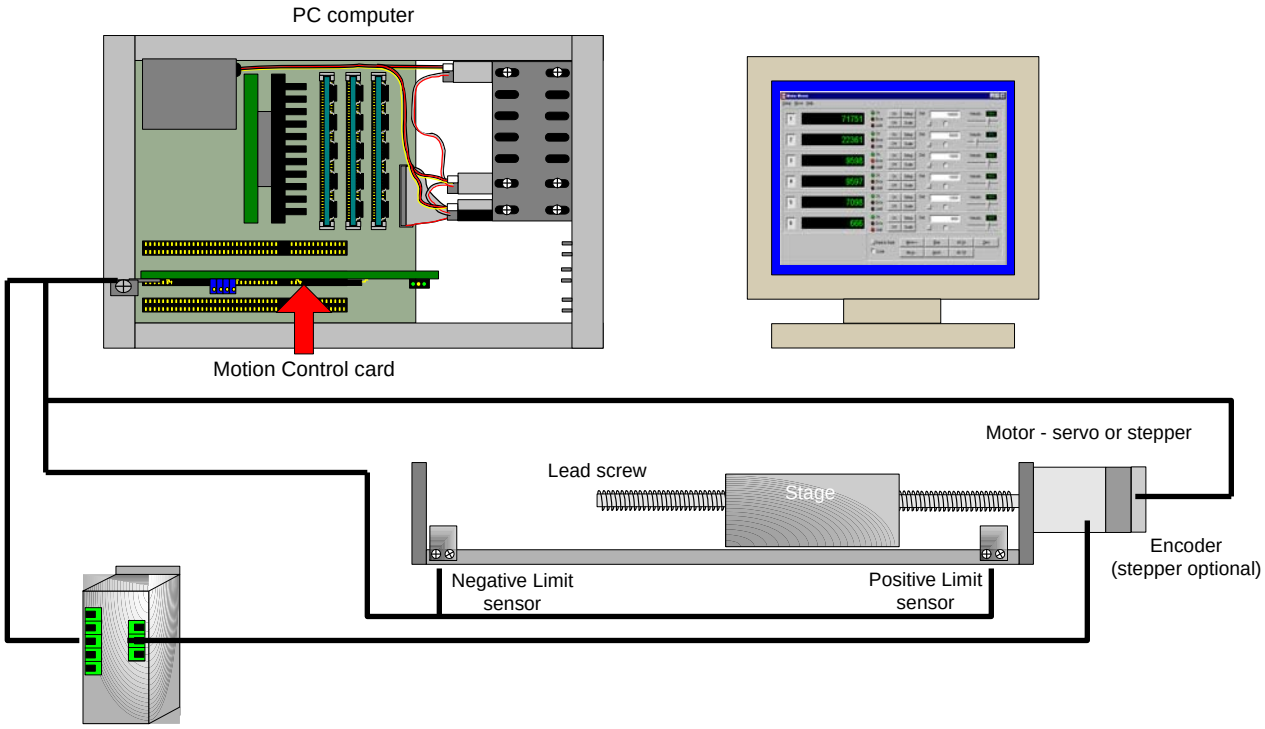
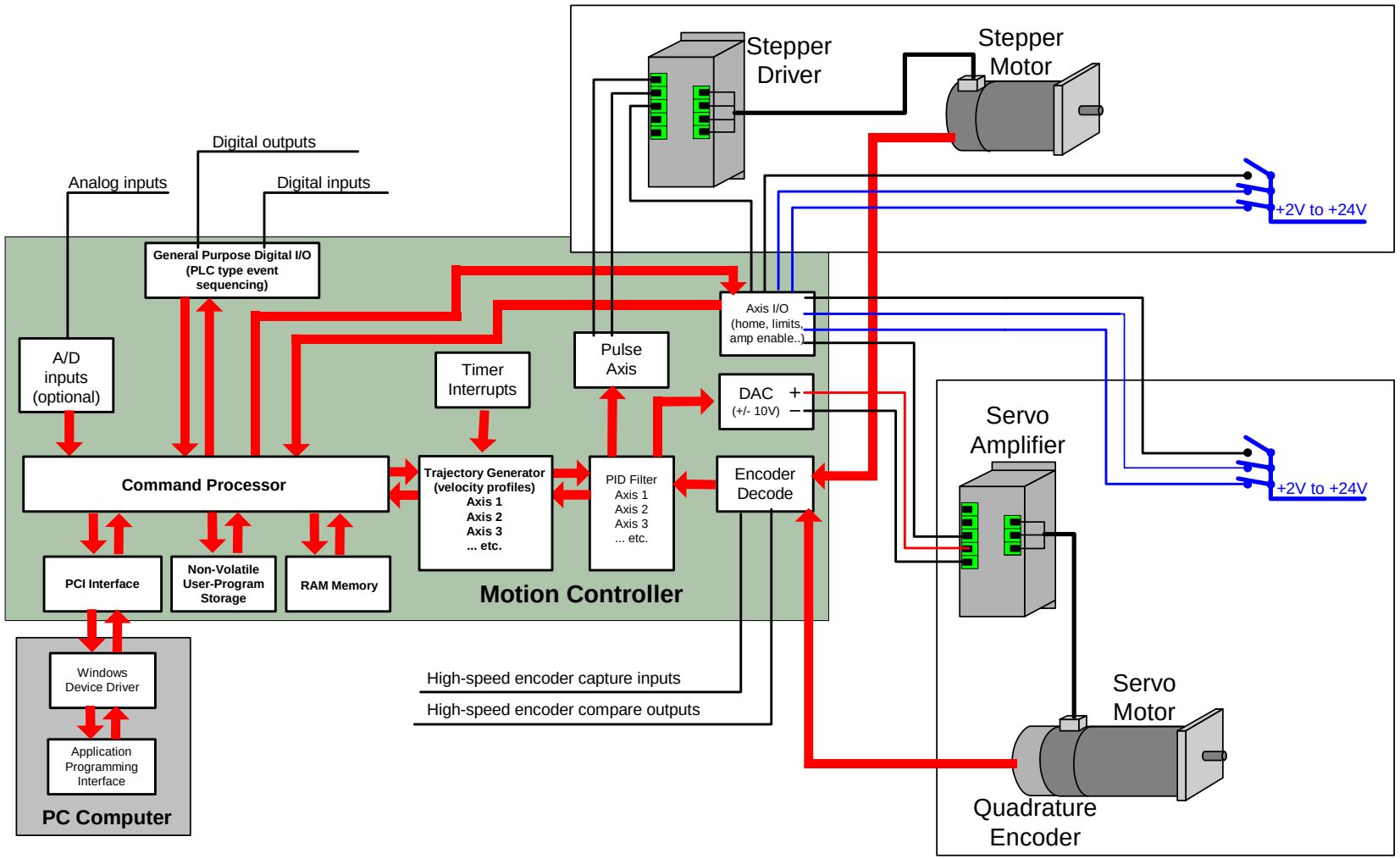


Figure 3: Typical PC-based Motion Control System

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Motion Controller Functional Block Diagram



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Motion Controller Tasks

The MultiFlex PCI 1000 Series motion controllers feature a 64-bit floating-point CPU, FPGA, I/O buffering circuitry, a real time kernel and proprietary motion control firmware which work in combination to control the position, velocity, or torque of as many as twelve axes. The primary operations performed by the motion controller are:

Trajectory generation (Trapezoidal, S curve, and Parabolic)
PID filter (servo loop)
I/O and error handling
Host communication

On a periodic basis (every 250 microseconds to 1 millisecond, depending on the selected Trajectory update rate), the controller's CPU receives an internal interrupt that automatically triggers the execution of the controller's ***trajectory generator***. Based on user commanded motion, the trajectory generator then calculates a new desired positions and velocity values for all axes.

+/- 10 Volt Analog Servo Control

The target positions calculated by the trajectory generator are passed to each respective PID filter for generation of a +/- 10V analog servo command signal for each axis. In addition to the trajectory generator, every millisecond, the controller performs housekeeping and error checking (over travel limits & following error exceeded) tasks.

PID Filter

The position feedback loop or PID (Proportional-Integral-Derivative) filter is executed every 1000, 500 or 250 microseconds (1, 2 or 4 kHz), depending on the servo loop update rate (Low, Medium or High) chosen by the user. Each PID filter execution results in the writing of a value to the DAC (Digital to Analog Converter) which is proportional to:

Position error (the difference between the optimal (desired) position and the current position)
Plus the integral of the error
Plus the derivative of the error

The following discrete-time equation illustrates the control performed by the servo controller:

$$u(n) = Kp*E(n) + Ki \text{ sum } E(n) + Kd[E(n') - E(n' - 1)]$$

where u(n) is the analog command output level at sample time n, E(n) is the position error at sample time n, n' indicates sampling at the derivative sampling rate, and kp, ki, and kd are the discrete-time filter parameters loaded by the users. The first term, the proportional term, provides a restoring force proportional to the position error. The second term, the integration term, provides a restoring force that grows with time. The third term, the derivative term, provides a force proportional to the rate of change of position error. It provides damping in the feedback loop. The sampling interval associated with the derivative term is user-selectable; this capability enables the servo controller to control a wider range of inertial loads.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Position Feedback via Incremental Encoder

The motion controller monitors the position of a servo via an incremental encoder. Both differential (A+, A-, B+, B-, Z+, Z-) and single ended (A, B, Z) incremental encoders are supported. The maximum encoder frequency is 20 MHz (@ 50% duty cycle). The two quadrature signals from the encoder are used to keep track of the position of the motor. Each time a logic transition occurs at one of the quadrature inputs, the controller's position counter is incremented or decremented accordingly. This provides four times the resolution over the number of lines provided by the encoder. The encoder interface is buffered by a differential line receiver that includes error detection circuitry that will indicate an encoder fault for the following conditions:

Open circuit condition
Short circuit condition
Low differential voltage signal
Common mode range violation

Note: For encoder fault detection of a single ended encoder the A-, B-, and Z+/Z- inputs must be terminated to the 1.5V Encoder Reference signal.

Pulse (Step/Dir/CW/CCW) Command for Stepper or Pulsed Servo Systems

The controller supports the following type of pulse-command axes.

Open Loop Stepper
Open Loop Stepper with encoder feedback for position verification
Closed Loop Stepper
Closed Loop Servo (position loop closed by the servo amplifier)

The default format of the pulse command signal pair is Step/Direction but it can be configured by the user for Clockwise/Counter Clockwise operation. The step-rate range for a pulse command axis is from a minimum of 0.1 pulses per second to a maximum of 5 million pulses per second.

Position Feedback via Incremental Encoder

For Pulse Command applications that require position feedback the controller supports both differential (A+, A-, B+, B-, Z+, Z-) and single ended (A, B, Z) incremental encoders. The maximum encoder frequency is 20 MHz (@ 50% duty cycle). The two quadrature signals from the encoder are used to keep track of the position of the motor. Each time a logic transition occurs at one of the quadrature inputs, the controller encoder position counter is incremented or decremented accordingly. This provides four times the resolution over the number of lines provided by the encoder. The encoder interface is buffered by a differential line receiver that includes error detection circuitry for differential encoders that indicates an encoder fault for the following conditions:

Open circuit condition
Short circuit condition
Low differential voltage signal
Common mode range violation

Note: For encoder fault detection of a single ended encoder the A_, B_, and Z+/Z- inputs must be terminated to the 1.5V Encoder Reference signal.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Axis I/O

Digital I/O

Dedicated and uncommitted Digital I/O are available on all MultiFlex motion controllers. The motion controller provides a total of 32 digital inputs and 28 digital outputs. 16 of the digital inputs are bi-directional optically isolated (+3V to +25 V) and 16 are TTL level. For MultiFlex PCI 1000 series model 1440, four of the bi-directional optically isolated inputs (Limit +, Limit -, Encoder Coarse Home, and Amplifier Fault) are shared between an analog command axis and a pulse command axis. 12 of the digital outputs are open-collector drivers and 16 are TTL level. By default each analog command axis includes an open collect Amplifier Enable output that is capable of sinking 100 mA. By using the I/O Configuration Panel described on pages 10 and 165, the user can associate any other digital output with the Amplifier On/Off function.

High-speed I/O

High-speed capture and compare signals are also functions provided via the TTL I/O. Position capture inputs are provided with up to 1 KHz trigger rate and a minimum pulse duration of less than 100 nanoseconds. High-speed TTL position compare outputs are provided with programmable trigger rates of greater than 1 MHz and a maximum latency of less than 100 nanoseconds. Position compare programmable modes of operation include: Strobe (trigger and repeat - up to 1 MHz repeat rate), Static, Toggle, and One-shot. The Strobe mode of operation is especially useful for triggering line-scan cameras for high-resolution inspection applications.

For maximum convenience and ease of wiring, the user can re-assign the default functions of the digital inputs and outputs using the **I/O Configuration Panel** described on page 10.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The Command Set - the heart of the motion controller

The motion controller is much more than an I/O card with DAC outputs and encoder inputs. The primary task of the motion controller is to off load control and monitoring duties from the PC processor. This requires a powerful and efficient low-level ***command set***. Everything that a motion control card can do depends on the command set. The command set of a high-performance motion controller should include the ability to:

- Move one, some, or all motors simultaneously
- Calculate the trajectories and execute synchronized motion (linear interpolation, circular contouring, helical motion)
- Set trajectory parameters (maximum velocity, acceleration, deceleration)
- Set PID filter parameters (proportional gain, derivative gain, derivative sampling period, integral gain, integral limit, allowable following error, and feed forward)
- Report the axis / controller status, current position, target position, and many other parameters
- Provide data or interrupt the host PC based on user defined events
- Home an axis

The controller's command set for is called MCCL (Motion Control Command Language) and it supports more than 200 individual operations. For a complete listing and description of the controller's command set please refer to the separate **Motion Control Command Language (MCCL) Reference Manual** which is available on PMC's Motion CD and online at: *www.pmccorp.com/support/support.php*.

For quick application prototyping and troubleshooting, PMC's WinControl utility allows the user to issue MCCL commands directly to the controller. From the keyboard, MCCL commands can be entered one command at a time and executed as soon as the user hits the 'Return' key. From the File menu, the user can also download an entire MCCL text file to the controller.

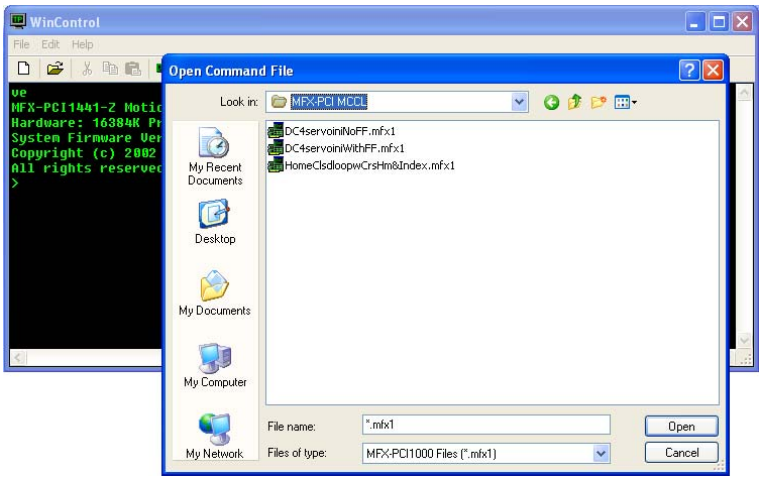
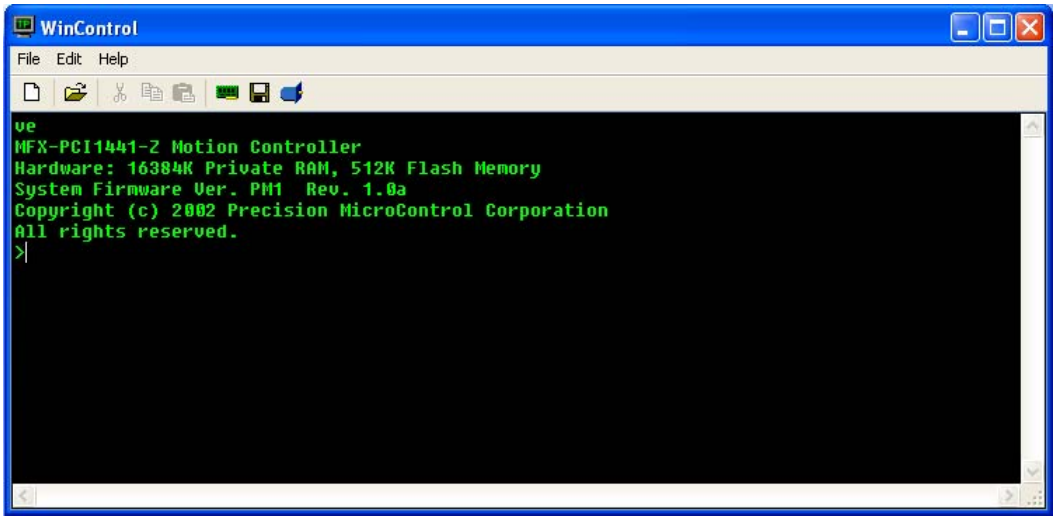


Figure 4. WinControl allows the user to issue MCCL commands to the controller

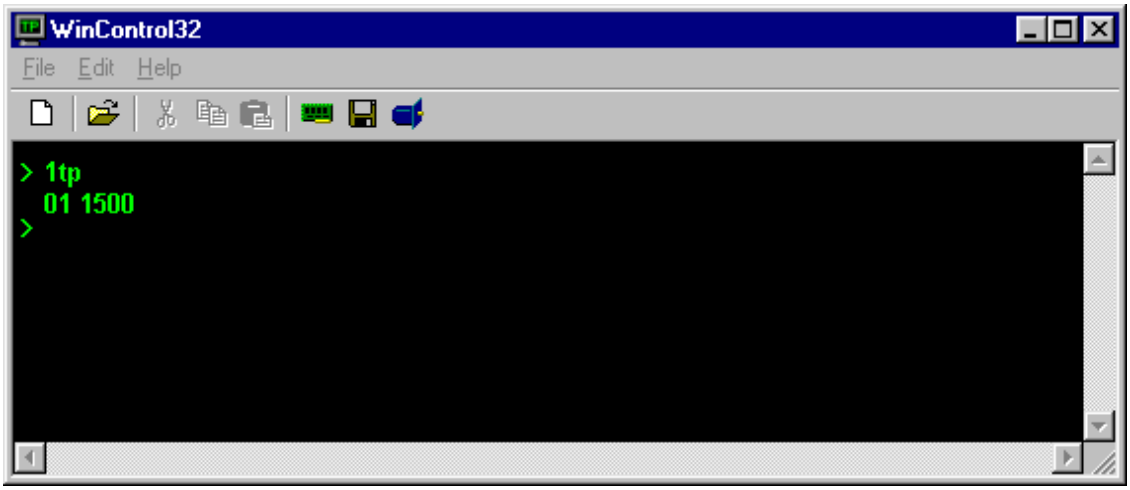
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Executing Operations with MCCL

MCCL commands are two character alphanumeric mnemonics built with two key characters from the description of the operation (eg. "MR" for ***Move Relative***). When an MCCL command (followed by a carriage return) is received by the controller it will be executed immediately. The following graphic shows the result of executing the "VE" command. This command causes the controller to report firmware version and installed memory size.



All axis-related MCCL commands will be preceded by an axis specifier, identifying for which axis the operation is intended. The following graphic shows the result of issuing the Tell Position (*a*TP) command to axis number one.



Note that each character typed at the keyboard should be echoed to the WinControl display. If you enter an illegal character or an illegal series of valid characters, the controller will echo a question mark character, followed by an error code. The MCCL Error Code listing can be found on page 213 of this manual. On receiving this response, you should re-enter the entire command/command string. If you make a mistake in typing, the backspace can be used to correct it, the controller will not begin to execute

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

a command until a carriage return is received.

Once you are satisfied that the communication link is correctly conveying your commands and responses, you are ready to check the motor interface. When the controller is powered up or reset, each axis is automatically set to the "motor off" state. In this state, there should be no drive current to the motors. For servos it is possible for a small offset voltage to be present. This is usually too small to cause any motion, but some systems have so little friction or such high amplifier gain, that a few millivolts can cause them to drift in an objectionable manner. If this is the case, the "null" voltage can be minimized by adjusting the offset adjustment potentiometer.

Before a motor can be successfully commanded to move certain parameters must be set by issuing commands to the controller. These include; PID filter gains (servo only), trajectory parameters (maximum velocity, acceleration, and deceleration), allowable following error (servo only), configuring motion limits (hard and/or soft).

At this point the user should refer to the **Motion Control** chapter sections titled **Theory of Operation – Motion Control**, **Servo Operation** and **Stepper Operation**. There the user will find more specific information for each type of motor, including which parameters must be set before a motor should be turned on and how to check the status of the axis.

Assuming that all of the required motor parameters have been defined, the axis is enabled with the **M**otor o**N** (***a*MN**) command. Parameter '*a*' of the Motor oN command allows the user to turn on a specific axes or all axes. To enable all, enter the Motor oN command with parameter '*a*' = 0. To enable a single axis issue the Motor oN command where '*a*' = the axis number to be enabled.

After turning a particular axis on, it should hold steady at one position without moving. The **T**ell **T**arget (***a*TT**) and **T**ell **P**osition (***a*TP**) commands should report the same number. There are two commands used for basic position mode motion, **M**ove **A**bsolute (***a*MA*n***) and **M**ove **R**elative (***a*MR*n***). To move axis 2 by 1000 encoder counts, enter 2MR1000 and a carriage return. If the axis is in the "Motor oN" state, it should move in the direction defined as positive for that axis. To move back to the previous position, enter 2MR-1000 and a carriage return.

The controller controller supports grouping together several commands. This is not only useful for defining a complex motion that can be repeated by a single keystroke, but is also useful for synchronizing multiple motions. To group commands together, simply place a comma between each command, pressing the return key only after the last command.

A repeat cycle can be set up with the following compound command:

```
2MR1000,WS0.5,MR-1000,WS0.5,RP6   <return>
```

This command string will cause axis 2 to move from position 1000 to position –1000 7 times. The **R**e**P**eat (**RP*n***) command at the end of a command string causes the previous command to be repeated 6 additional times. The **W**ait for **S**top (***a*WS*n***) commands are required so that the first motion will be completed (trajectory complete) before the return motion is started. The number 0.5 following the **WS** command specifies the number of seconds to wait after the axis has ceased motion to allow some time for the mechanical components to come to rest and reduce the stresses on them that could occur if the motion were reversed instantaneously. Notice that the axis number need be specified only once on a given command line.

A more complex cycle could be set up involving multiple axes. In this case, the axis that a command acts on is assumed to be the last one specified in the command string. Whenever a new command string is entered, the axis is assumed to be 0 (all) until one is specified.

Entering the following command:

```
2MR1000,3MR-500,0WS0.3,2MR1000,3MR500,0WS0.3,RP4   <return>
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

will cause axis 2 to move in the positive direction and axis 3 to move in the negative direction. When both axes have stopped moving, the WS command will cause a 0.3 second delay after which the remainder of the command line will be executed.

After going through this complex motion 5 times, it can be repeated another 5 times by simply entering a return character. All command strings are retained by the controller until some character other than a return is entered. This comes in handy for observing the position display during a move.
If you enter:

```
1MR1000   <return>
1TP    <return>
(return)
(return)
(return)
(return)
```

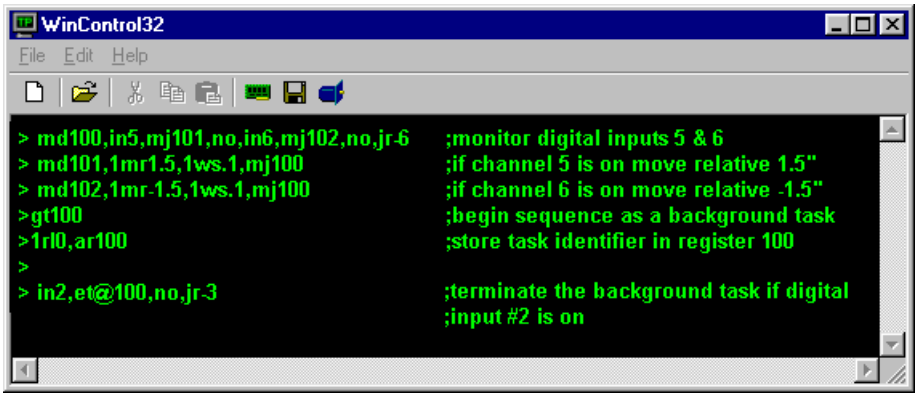
The controller will respond with a succession of numbers indicating the position of the axis at that time. Many terminals have an "auto-repeat" feature that allows you to track the position of the axis by simply holding down the return key.

Another way to monitor the progress of a movement is to use the ***Repeat*** command without a value. If you enter:

```
1MR10000  <return>
1TP,RP    <return>
```

The position will be displayed continuously. These position reports will continue until stopped by the operator pressing the Escape key.

While the controller is executing commands, it will ignore all alphanumeric keys that are pressed. The user can abort a currently executing command or string by pressing the escape key. If the user wishes only to pause the execution of commands, the user should press the space bar. In order to restart command execution press the space bar again. If after pausing command execution, the user decides to abort execution, this can be done by pressing the escape key.



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Closed loop, open loop, and position verification

As it applies to motion control, there are three recognized control modes:

Closed loop control
Open loop control
Open loop with position verification

Closed loop control

A broadly applied term, relating to any system in which the output is measured and compared to the input. The output is then adjusted to reach the desired condition. In motion control, the term typically describes a system utilizing a position transducer (an incremental encoder) to generate correction signals in relation to desired parameters.

Servo systems are the most prevalent example of closed loop control. In a typical servo system a move operations is initiated by the user issuing a move command to the servo controller. The controller then calculates a velocity profile matching previously defined user trajectory (max. velocity, acceleration, and deceleration) parameters. The controller applies a position/velocity command to the servo amplifier. Based on feedback from an incremental encoder the servo controller calculates the following error (difference between the actual position and the calculated desired position). The following error value is then used by the PID filter to adjust the magnitude of the position / velocity command to the amplifier. For additional information on the trajectory generator please refer to page 13 and 86. For additional information on the PID filter please refer to page 13. For additional information on incremental encoders please refer to page 24.

The significant advantage of a closed loop system is that based on the constant corrections by the PID filter of the command voltage (based on the measured following error), servo systems are inherently more intuitive than open loop systems. The disadvantage to the corrective nature of a servo system is that in order for the PID filter to properly respond to a given error, the servo must be tuned. Tuning a servo is a process in which the PID filter gain values are defined so that the response of the servo system to a given following error meets the requirements of the machine designer. Compared to an open loop system, which does not require PID filter tuning, the requirement of tuning a servo makes the setup of a closed loop system a more complicated and time consuming operation. For additional information on tuning a servo please refer to pages 22 and 68.

Open Loop control

An open loop control system is one in which the control output is not referenced or scaled to an external feedback (typically an incremental encoder). The most widely recognized example of an open loop control system is an axis controlled by a stepper motor. Most stepper motor controlled axes do not included any type of external feedback device, so the axis is said to be operating "open loop". If for some reason the stepper motor did not actually reach its target the stepper controller would be unaware of the fact.

Pulse command servo systems feature the use of digital amplifiers/drives which accept step/direction (or CW/CCW) pulse command signals as inputs, and in which the closed loop control (position or velocity) is executed entirely by the servo amplifier. As with a traditional servo system, a feedback device is required, but in this case it is not necessary to connect it to the motion controller. The controller supports pulse command servos, and if ordered with the stepper axis encoder option, the controller also supports reading and recording the encoder position.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Open loop with position verification

By adding a feedback device (like an encoder) to a traditional open loop stepper you would have what is known as an open loop with position verification system. For some applications where the higher costs, complexity, or torque limitations of closed loop servos may be prohibitive a stepper coupled with an encoder makes for a perfect compromise.

Contrary to many servo applications, where the following error of the axis along the entire path is of great concern to the machine designer, for many stepper applications the user is only concerned about the 'end of move' final position of the axis. By adding an encoder (typically directly coupled to the stepper motor shaft) the user can monitor the 'end of move' final position of the axis and issue 'correction moves' to compensate for any possible lost steps. For additional information on open loop with position verification systems please refer to page 143.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Why does a servo need to be tuned?

A servo is a closed loop system, which the dictionary describes as:

An automatic system in which the output is constantly compared with the input through some form of feedback. The error (or difference) between the two quantities can be used to bring about the desired amount of control.

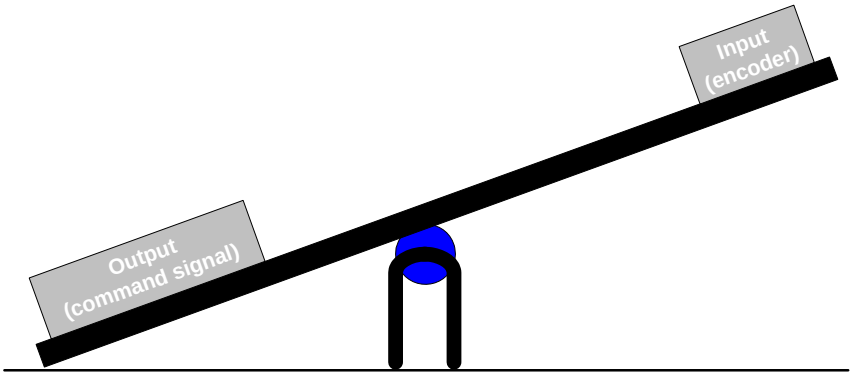
In typical servo systems:

- The output is a +/- 10 volt (torque or velocity) command that is applied as an input to a servo amplifier
- The input described in the dictionary definition comes from an encoder. An encoder is an opto electric device that generates two pulse trains that are phase shifted by 90 degrees
- In order for a servo system to perform properly, the difference (error) between the input and output is multiplied by a set of gain values which results in a new output, bringing about the desired amount of control

Servo tuning is the process in which the gain values are determined. From one servo axis to another the gain values will change depending on differences between the motion controller, motor, encoder, and load. When a user attempts to move an axis without first tuning the servo (determining the gain values) the motion controller will not be able to calculate the **appropriate** output command to apply to the servo amplifier. One of the following undesirable results will probably be observed:

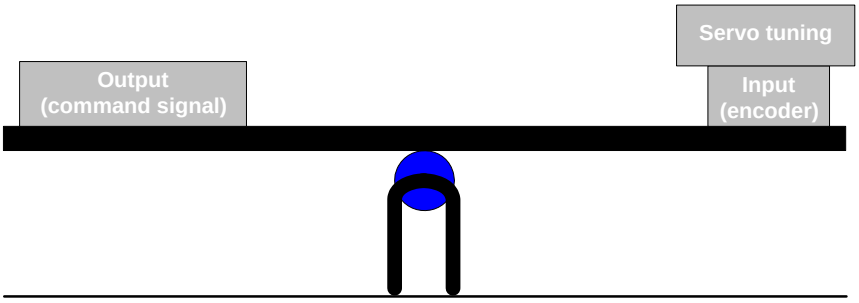
- The axis will not move at all
- The axis moves in the direction of the target but stops well short of the target
- The axis moves in the opposite direction of the commanded target
- The axis towards the target but fails to 'settle', oscillation of one or more encoder counts is detected

Imagine a seesaw, with the +/- 10 volt torque/velocity command on one side and the response of the motor/load (feedback from an encoder) on the other side.



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Until the servo is tuned, the system is effectively out of balance. Only after a servo has been tuned can the controller calculate the appropriate torque/velocity command output for a given user defined motion.



To tune a servo axis use the Servo Tuning program included with PMC's Motion Integrator software. For assistance with servo tuning, refer to the **Motion Control** chapter of this manual or view or the PowerPoint **Servo Tuning** tutorial available at *www.pmccorp.com/support/mfxpci1000.php*.

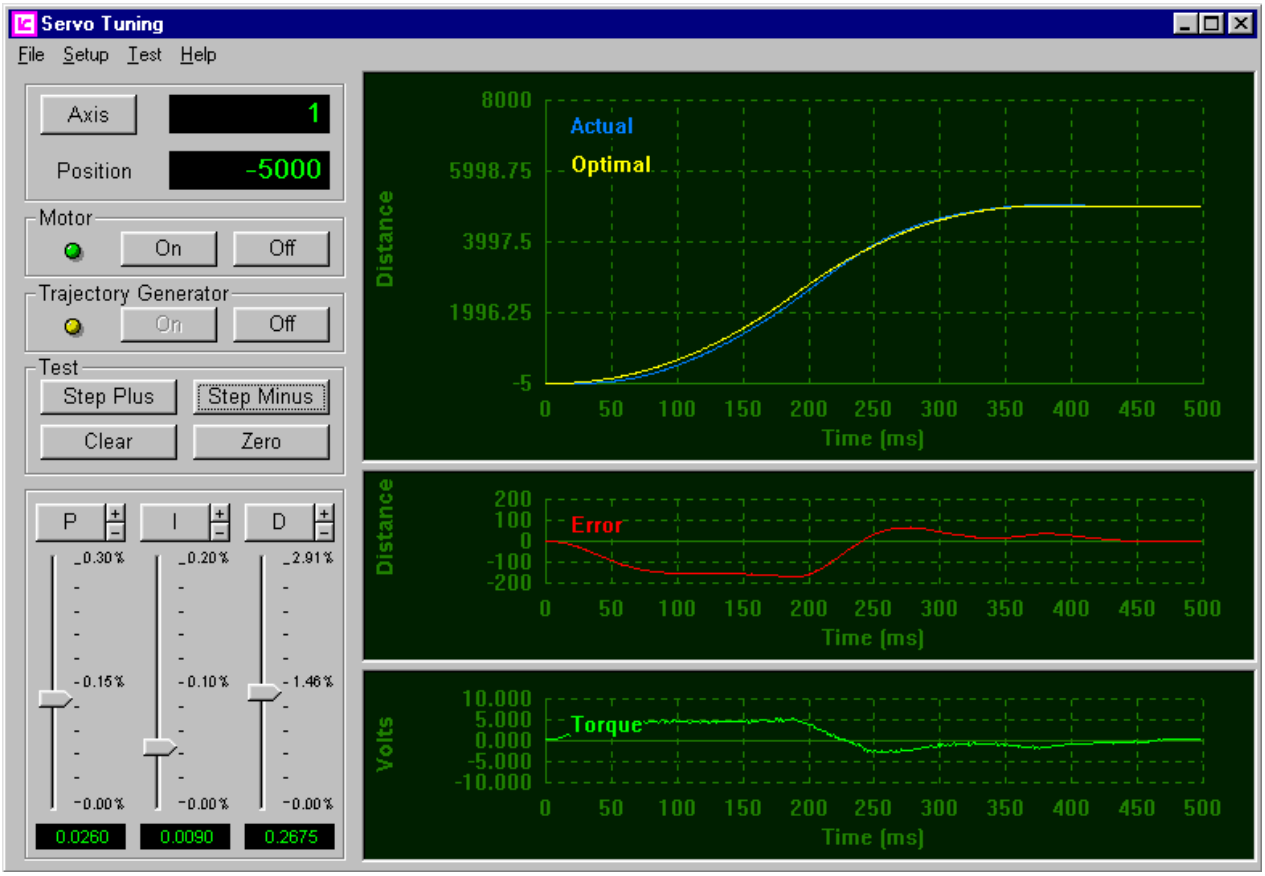


Figure 5. The Servo Tuning program is used to select PID gain values

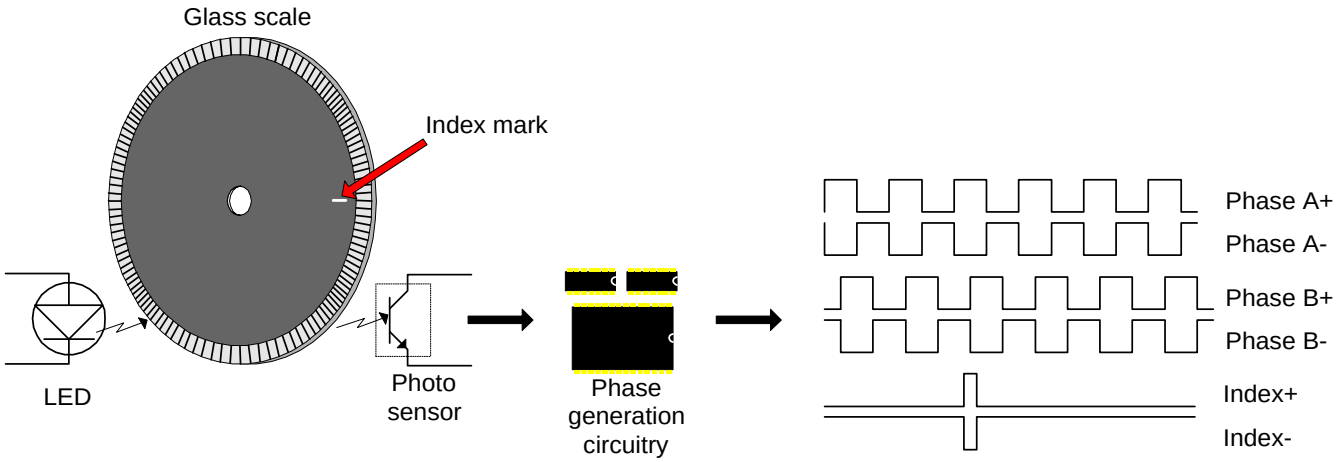
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Position Feedback - Quadrature Incremental Encoder

Quadrature Incremental Encoders are the default standard for providing position / velocity feedback for today's motion control systems. A quadrature incremental encoder is an opto electric feedback device. A light source and photo sensor pickup are used to detect markings on a glass 'scale'. The more markings on the glass scale, the higher the resolution of the encoder. The output of the photo sensor is passed to a Phase Generator circuit, which is used to generate two wave forms (Phase A and Phase B), which have a phase difference of 90 degrees. This phase difference is used by the controller to:

Determine the direction of rotation (positive or negative) of the encoder/motor
Enhance the resolution of the encoder by a factor of 4.

For noise immunity or applications where the encoder is positioned a significant distance from the motion controller the encoder can use a differential driver device to output both the generated wave forms (A+ and B+) and their compliments A- and B-). For Differential or single ended encoders, the controller provides the user with the the option of enabling Encoder Fault detection, which will indicate an error upon open circuit, short circuit, low differential voltage signal, and common mode range violation.



A 500 line quadrature incremental encoder will have 2000 quadrature counts per full rotation. The 90 degree phase difference is also used to determine the direction of motion of the axis/encoder. If phase A comes before phase B, the controller will indicate positive or clockwise direction. If phase B comes before phase A, the controller will indicate negative or counter-clockwise direction.

Some quadrature encoders include an additional 'mark' on the glass scale, which is used to generate an index pulse. This signal, which 'goes active' once per rotation, is used by the motion controller to accurately home (re-define the position of an axis) the axis. Please refer to the **Homing Axes** section of this chapter.

Typically an encoder requires a +5 VDC power supply and ground reference, both of which are available from the controller .

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Servo Amplifiers: Current Mode versus Velocity Mode

For the vast majority of servo applications the user has the option of choosing between using a Current Mode amplifier and a Velocity mode amplifier.

Current Mode amplifier (sometimes called Torque Mode)

The +/- 10V analog command output from the servo controller represents a current command to the motor. The resulting output from the current mode amplifier will be proportional to the analog command voltage output of the servo controller. A current mode amplifier typically requires that the user 'tune' the current loop of the amplifier using either combinations of resistors and capacitors or adjusting potentiometers while following manufacturer provided 'cook book' procedures.

Current mode amplifier advantages:

- Low cost
- Ease of use
- High acceleration / deceleration increases machine throughput

Velocity Mode amplifier

Unlike a torque mode amplifier that closes only the current loop, a velocity mode amplifier closes both the current loop and the velocity loop. A tachometer is used to provide velocity feedback. The +/- 10V analog command output from the servo controller provides a velocity command to the amplifier, and the servo controller uses Feed Forward (other wise known as Velocity Gain) to calibrate the velocity command to the amplifier.

For servo systems that use velocity mode amplifiers the servo controller PID loop closes only the position loop, and its operations are secondary to the velocity loop of the amplifier. Typically the PID gain values of the servo controller will be very low (compared to the gain values of a torque mode amplifier).

Torque mode amplifier advantages:

- High accuracy
- Analog velocity loop results in higher gains (stiffer response) and minimal following error

If there is a downside to using velocity mode amplifiers it would be that they can be more difficult to configure. For one thing, unlike the torque mode amplifier that required minimal setup, a velocity mode amplifier must be well tuned (minimal overshoot and no oscillation) before attempting to tune the position loop of the servo controller. For additional information on working with velocity mode amplifiers please refer to page 71.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Stepper Motors - Full Step versus Micro Step

Stepper motors have long been viewed as a low cost alternative to closed loop servos. The reality is that there are real world application requirements for which stepper motors are better suited, regardless of cost. The primary advantages of a stepper motor are:

High torque to size
High torque at low speed
Holding torque (holds its position while not being commanded to move

Historically one of the primary disadvantages of a stepper motor was the limited number of step per rotation, which limits final positioning resolution of the axis. Typical steps per rotation of today's stepper motors range from 100 (3.6 degrees per step) to 500 (0.72 degrees per step). But with the advent of microstepping driver technology a whole new world of applications have been opened up to the stepper motor. For most stepper applications using a microstepping stepper driver can the user will gain:

Increased positioning resolution
Increased position accuracy
Increased system performance by minimizing resonance
Increased velocity resolution

As it relates to a stepper motor controller like the MultiFlex PCI 1000 Series, full step versus microstepping is not an issue. The microstepping function of a motor occurs entirely in the stepper motor driver - it has nothing to do with the motion controller. When switching to a microstepping stepper driver the only required change is that prior to issuing a move command the user must recalculate the trajectory parameters (max. velocity, acceleration, deceleration, and minimum velocity) and the move distance. For example if a 200 step per rotation (1.8 degrees per step) stepper system with a maximum velocity of 20,000 steps per second is upgraded by using a microstepping driver operating at a ratio of 10:1:

1) The maximum velocity of the axis is increased from 20,000 steps/sec. 200,000 steps/sec.
2) The move the axis one complete rotation the move distance is increased from 200 to 2000

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Homing - Why, When, and How

All data registers on the motion controller are volatile, if power is cycled (turn off and then turned on) or the controller is reset, the position registers will be initialized to zero. In order for the user to position one or more axes to specific locations on the machine the user must first initialize the machine by homing each of the axes.

For most applications, there is some position/angle of the axis (or mechanical apparatus) that is considered 'home'. Typical automated systems use electro-mechanical devices (switches and sensors) to signal the controller when an axis has reached this position. Upon activation of the sensor the controller captures the position of the axis. The controller is not shipped from the factory programmed to perform a specific homing operation. Instead, it has been designed to allow the user to define a custom homing sequence that is specific to the application requirements. For additional information on building homing sequences please refer to page 105.

Homing closed loop systems

The home location of a closed loop system is usually defined by the index mark of an encoder. For systems that use a rotary encoder, where the index mark will be asserted multiple times along the range of travel of the axis, a Coarse Home sensor is used to qualify which of the index mark locations will be the home location. For additional information on homing closed loop axes please refer to page 107.

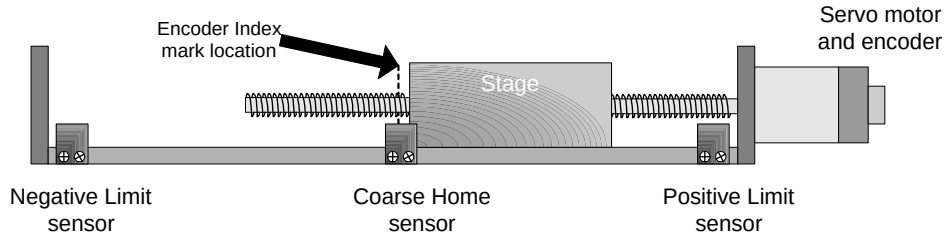


Figure 6. Homing a closed loop system with an encoder index mark and Coarse Home sensor

Homing open loop systems

Open loop steppers are typically homed based on the location of a home sensor. Unlike closed loop systems that use a precision reference index mark, steppers are more prone to homing inaccuracies due the lower repeatability of the single electro mechanical home sensor. To achieve the highest possible repeatability; reduce the velocity of the axis and always approach the home sensor from the same direction. For additional information on homing open loop steppers please refer to page 112 .

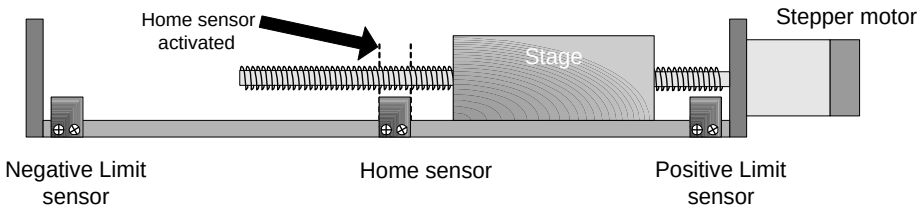


Figure 7. Homing an open loop stepper with a Home sensor

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Chapter 3

Software, Programming and Utilities

MultiFlex motion controllers can integrate seamlessly with host-computer based Windows applications. The **Motion Control Application Programming Interface** (Motion Control API) provides support for all popular high level languages, including C/C++/C#/.NET Delphi, Visual Basic and LabVIEW. Additionally, an embedded Motion Control Command Language (MCCL) allows machine designers to execute motion control routines independent of the host PC.

PMC's Motion Control API is a group of Windows components that, taken together, provide a consistent, high level, Applications Programming Interface (API) for PMC's motion controllers. The difficulties of interfacing to new controllers, as well as resolving controller specific details, are handled by the API, leaving the applications programmer free to concentrate on the application program.

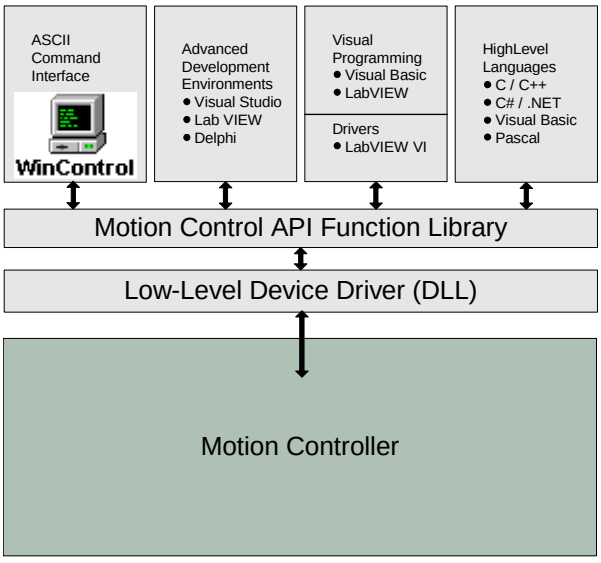


Figure 8. PMC's Motion Control API Architecture

The Motion Control API has been designed with a layered approach. As new versions of of the Motion Control API and new PMC motion controllers become available, API support is provided by simply replacing one or more of these layers. Because the public API (the part the applications programmer sees) lies above these layers, no changes to applications programs will be required to support new versions of the Motion Control API.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The API itself is implemented in three parts. The low level device driver provides communications with the motion controller, in a way that is compatible with the Microsoft Windows operating system. The Motion Control API low level driver passes binary MCCL commands (Motion Control Command Language – the instruction set of the controller) to the controller . By placing the operating system specific portions of the API here it will be possible to replace this component in the future to support new operating systems without breaking application programs, which rely on the upper layers of the API.

Sitting above that, and communicating with the driver is the API Dynamic Link Library (DLL). The DLL layer implements the high level motion functions that make up the API. This layer also handles the differences in operation of the various PMC Motion Controllers, making these differences virtually transparent to users of the API.

At the highest level are environment specific drivers and support files. These components support specific features of that particular environment or development system.

Care has been exercised in the construction of the API to ensure it meets with Windows interface guidelines. Consistency with the Windows guidelines makes the API accessible to any application that can use standard Windows components - even those that were developed after the Motion Control API! Please refer to the Motion Control Application Programming Interface (Motion Control API Reference Manual for additional information on adapting the Motion Control API to other development environments.

Controller Interface Types

The controller supports two onboard interfaces, an ASCII (text) based interface and a binary interface. The binary interface is used for high speed command operation, and the ASCII interface is used for interactive text based operation. The high level sample programs (CWDEMO, PASDEMO, and VBDEMO) use the binary interface, PMC WinControl uses the ASCII interface.

Application programs must indicate which interface they intend to use when they open a handle for a particular controller. A controller may have more than one handle open at a time, but all open handles for a particular controller must specify the same interface (all must be open with the binary interface or all must be open with the ASCII interface). The open mode is specified by setting the second argument of the ***MCOpen( )*** function to either **MC_OPEN_ASCII** or **MC_OPEN_BINARY**.

Note that not all functions are available in the ASCII mode of operation, this mode is intended primarily for use with the ***pmcgetc( )***, ***pmcgets( )***, ***pmcputc( )***, and ***pmcputs( )*** character based functions (these 4 functions are not available in binary mode). This restriction will be eliminated in a future release of the API.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

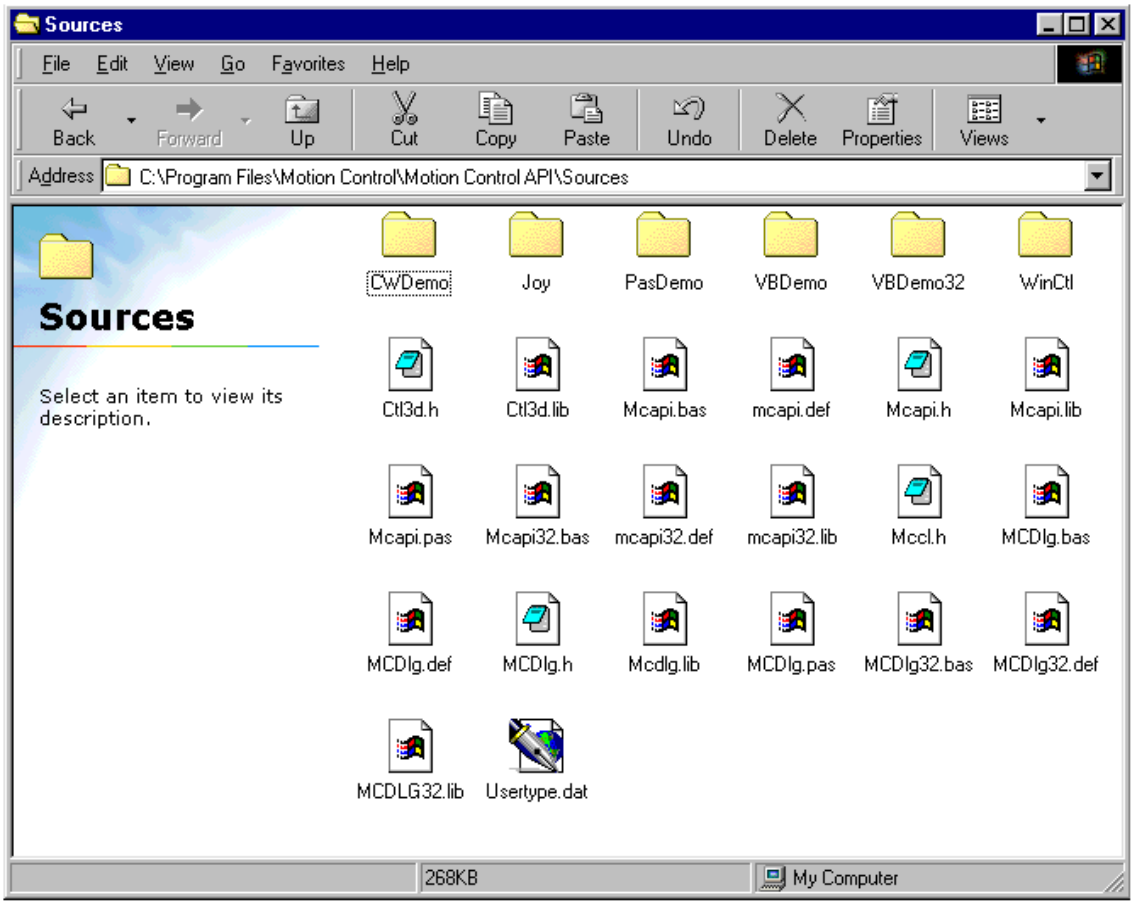
Building Application Programs using Motion Control API

The Motion Control Application Programming Interface is designed to allow a programmer to quickly develop sophisticated application programs using popular development tools. The Motion Control API provides high level function calls for:

- Configuring the controller (servo tuning parameters, velocity and ramping, motion limits, etc.)
- Defining on-board user scaling (units for encoder/step, velocity, dwell time, user and part zero)
- Commanding motion (Point to Point, Constant velocity, Electronic Gearing, Lines and Arcs, Joystick control)
- Reporting controller data (motor status, position, following error, current settings)
- Monitoring Digital and Analog I/O
- Driver functions (open controller handle, close controller handle, set timeout)

A complete description of all Motion Control API functions can be found in the Motion Control API Reference Manual.

Included with the installation of the Motion Control API is the Sources ‘folder’. In this folder are complete program sample source files for C++, Visual Basic, and Delphi.



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

C/C++ Programming

Included with each of the C program samples (CWDemo. Joystick demo, and WinControl) is a read me file (readme.txt) that describes how to build the sample program. The following text was reprinted from the readme.txt file for the CWDemo program sample.

Contents
========
- How to build the sample
- LIB file issues
- Contacting technical support

How to build the sample
=======================
To build the samples you will need to create a new project or make file within your C/C++ development tool. Include the following files in your project:

CWDemo.c
CWDemo.def
CWDemo.rc

For 16-bit development you will also need:

..\mcapi.lib
..\mcdlg.lib
..\ctl3d.lib

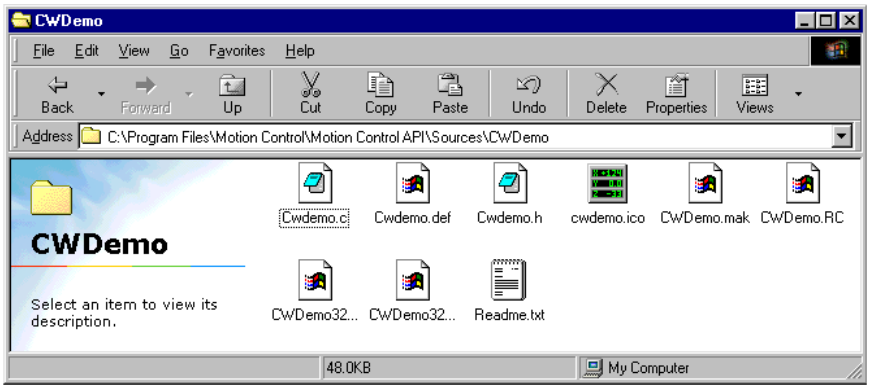
For 32-bit development you will also need:

..\mcapi32.lib
..\mcdlg32.lib

If your compiler does not define the _WIN32 constant for 32-bit projects you will need to define it at the top of the source file (before the header files are included).

LIB File Issues
===============
Library (LIB) files are included with MCAPI for all the DLLs that comprise the user portion of the API (MCAPI.DLL, MCAPI32.DLL, MCDLG.DLL, and MCDLG32.DLL). These LIB files make it easy to resolve references to functions in the DLL using static linking (typical of C/C++). Unfortunately, under WIN32 the format of the LIB files varies from compiler vendor to compiler vendor. If you cannot use the included LIB files with your compiler you will need to add an IMPORTS section to your projects DEF file. We have included skeleton DEF files for all of the DLLs for which we also include a LIB file (MCAPI.DEF, MCAPI32.DEF, MCDLG.DEF, and MCDLG32.DEF).

The 16-bit LIB files were built with Microsoft Visual C/C++ Version 1.52, and the 32-bit LIB files Microsoft Visual Studio Version 5.



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Visual Basic Programming

Included with each of the Visual Basic program samples (VBDemo. VBDemo32) is a read me file (readme.txt) that describes how to build the sample program. The following text was reprinted from the readme.txt file for the VBDemo32 program sample.

Contents
========

- About the sample
- How to build the sample
- Contacting technical support

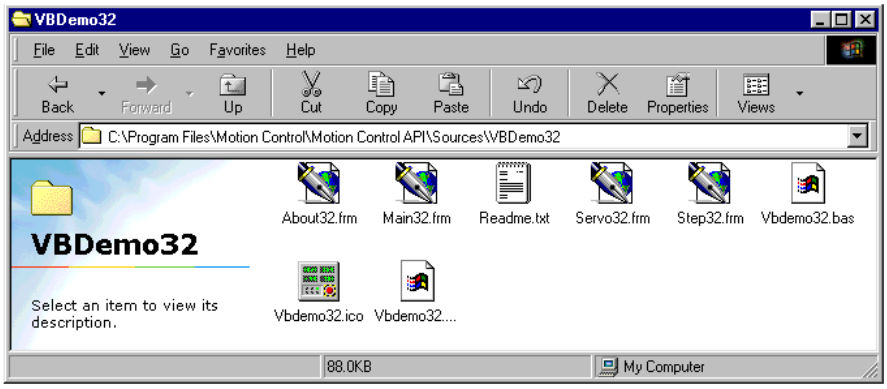
About the sample
================
This sample demonstrates a simple user interface to one axis of a motion controller. The user may program moves and interact with the motion in a number of ways (stop it, abort it, etc.). Sample forms demonstrate how to configure servo or stepper motor axes. A number of the new MCDialog functions (such as a full-featured, ready-to-run
axis configuration dialog) are also demonstrated.

How to build the sample
=======================
To build the samples you will need to create a new project or use the Visual Basic project file (created with Visual Basic v6.0) included with the sample. Include the following files if you create your own project:

About32.frm
Main32.frm
Servo32.frm
Step32.frm
VBDemo.bas

..\mcapi32.bas
..\mcdlg32.bas

Set frmMain as the startup object for the project.



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Delphi Programming

Included with each of the Delphi program sample (PasDemo) is a read me file (readme.txt) that describes how to build the sample program. The following text was reprinted from the readme.txt file for the PasDemo program sample.

Contents
========

- About the sample
- How to build the sample
- Contacting technical support

About the sample
================
This sample demonstrates a simple user interface to one axis of a motion controller. The user may program moves and interact with the motion in a number of ways (stop it, abort it, etc.). Sample forms demonstrate how to configure servo or stepper motor axes. A number of the new MCDialog functions (such as a full-featured, ready-to-run
axis configuration dialog) are also demonstrated.

How to build the sample
=======================
To build the samples you will need to create a new project or use the Delphi project files included with the sample (Pdemo.dpr for 16-bit, Pdemo32.dpr for 32-bit). Include the following files if you create your own project:

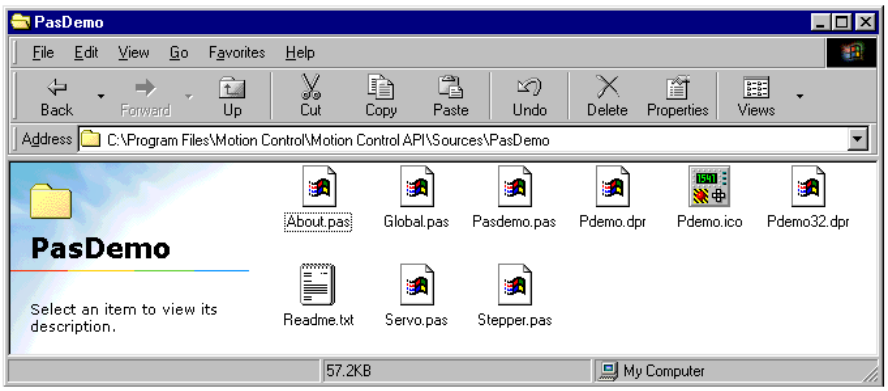
About.pas
Global.pas
PasDemo.pas
Servo.pas
Stepper.pas

For 16-bit projects you will also need:

..\mcapi.pas
..\mcdlg.pas

For 32-bit projects you will also need:

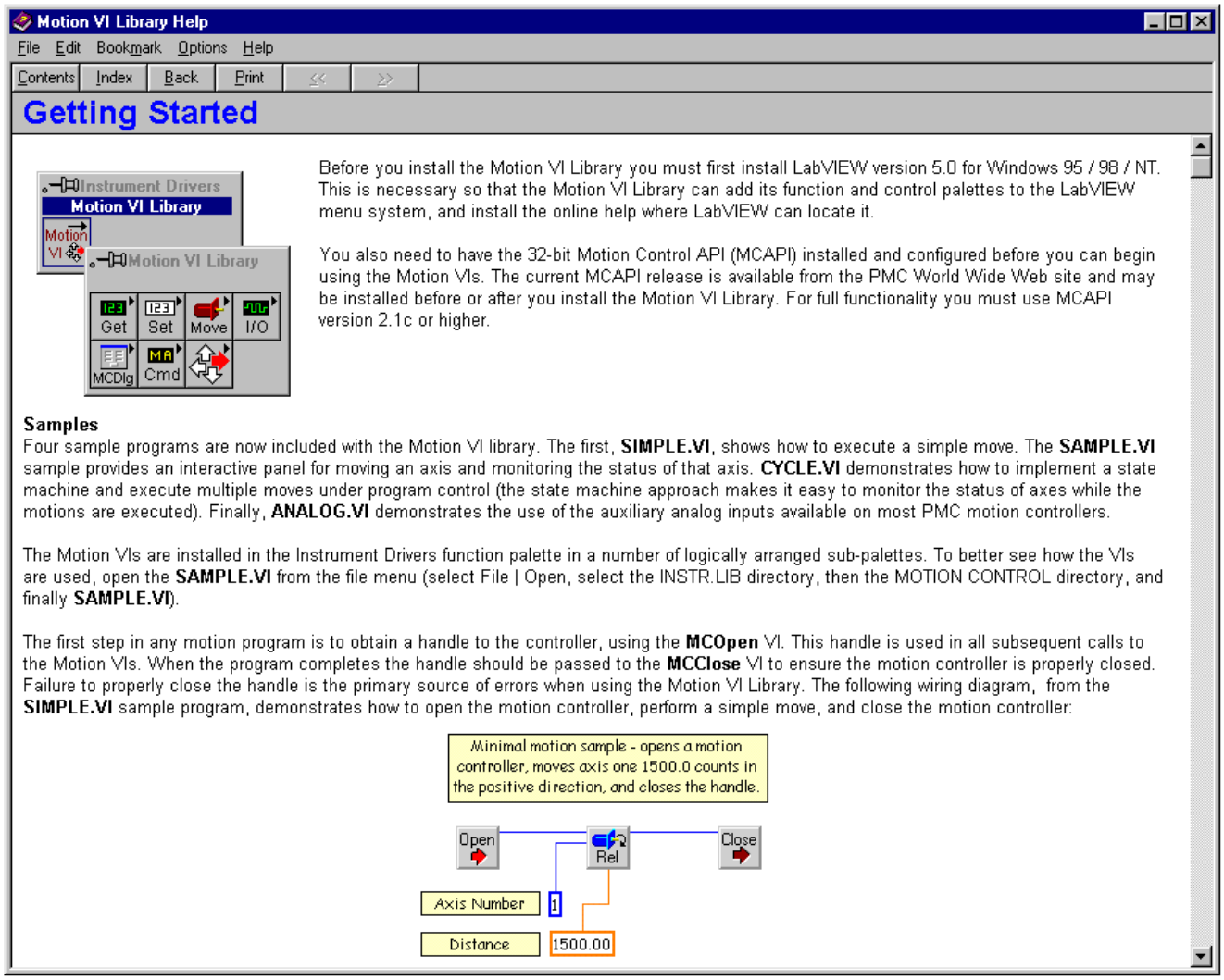
..\mcapi32.pas
..\mcdlg32.pas



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

LabVIEW Programming

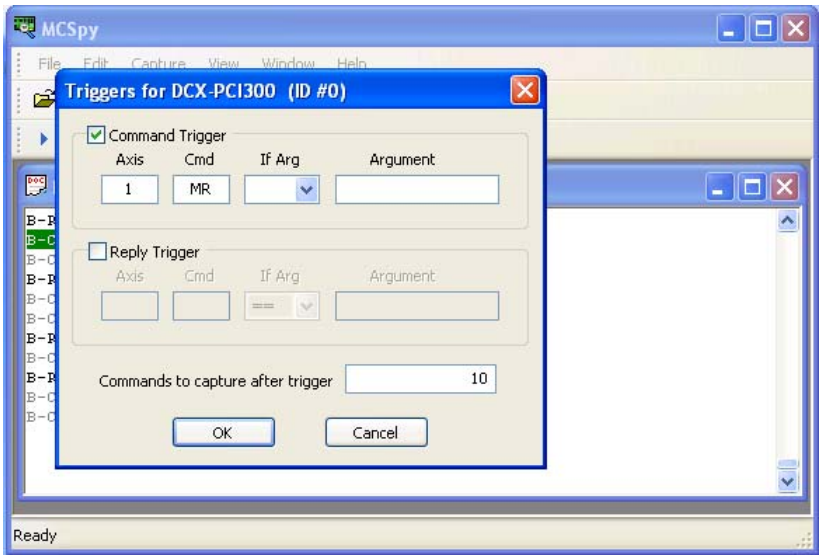
PMC's LabVIEW Virtual Instrument Library includes an On-Line help with a Getting Started guide.



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

MCSpy - application program diagnostic tool

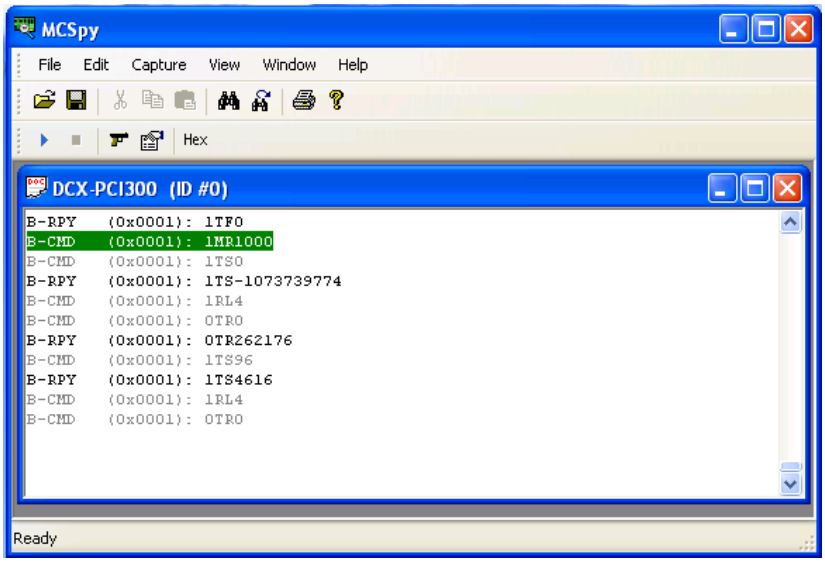
MCSpy is a debugging tool for application programs that use PMC's Motion Control API programming interface. MCSpy captures commands and replies sent between the application program and the motion control card. These commands are displayed in Motion Control Command Language (MCCL), which is the language the Motion Control API uses to communicate with PMC's Motion



The MCSpy Trigger Setup dialog allows the user to terminate the capturing of commands / replies data after the trigger event.

Here the command /reply capture will end 10 commands after a move relative (MR) command has been issued to axis #1.

The Trigger Event (1MR1000) is highlighted in green.

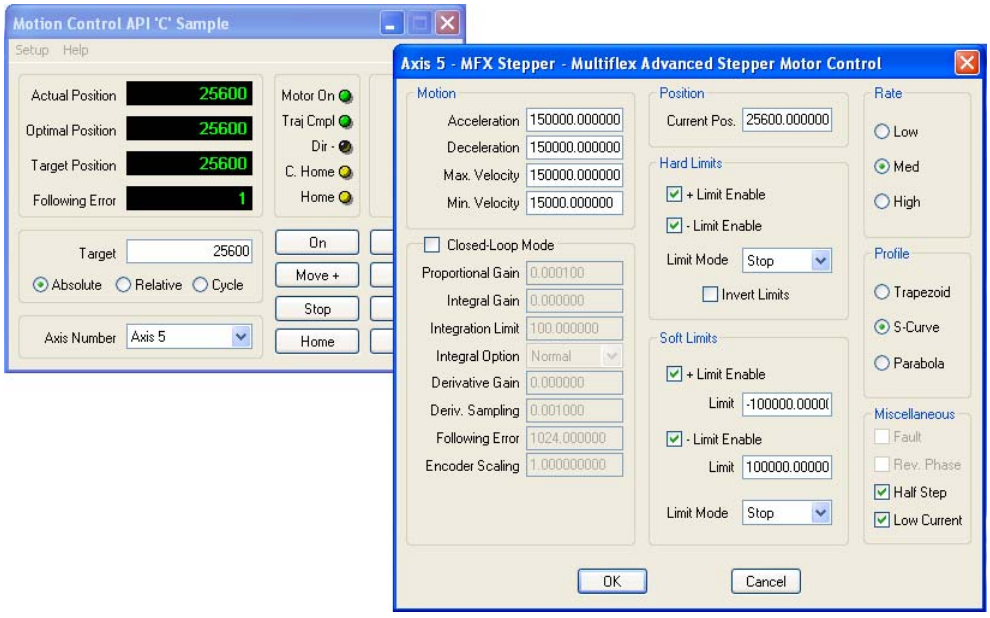


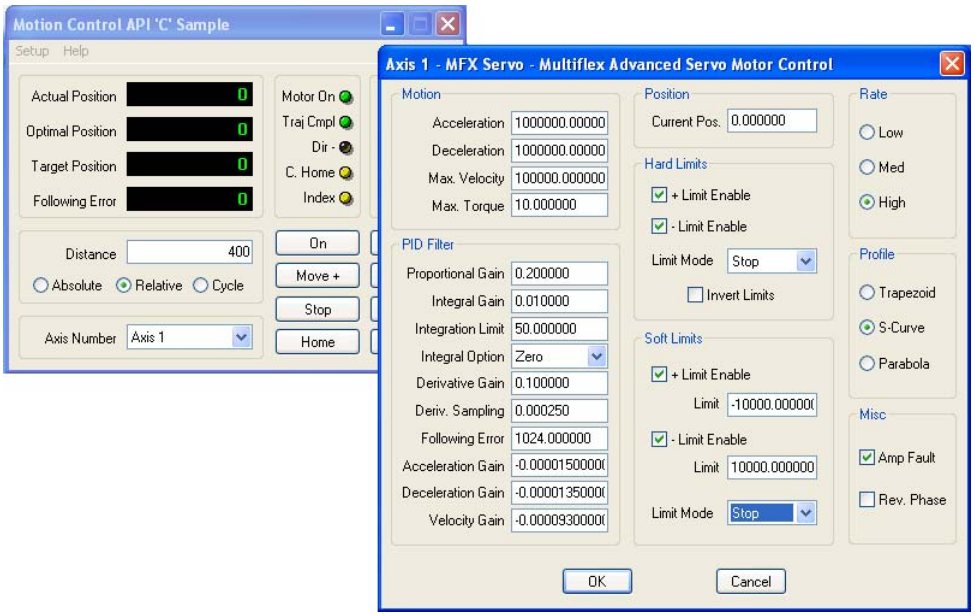
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PMC Sample Programs

Sample programs with full source code are supplied with the Motion Control API. These C++, Visual Basic, and Delphi sample programs allow the user to:

- Move an axis (servo or stepper)
- Monitor the actual, target, and optimal positions of an axis
- Monitor axis I/O (Limits +/-, Home, Index, an Amplifier Enable)
- Define or change move parameters (Maximum velocity, Acceleration/Deceleration)
- Define or change the servo PID parameters



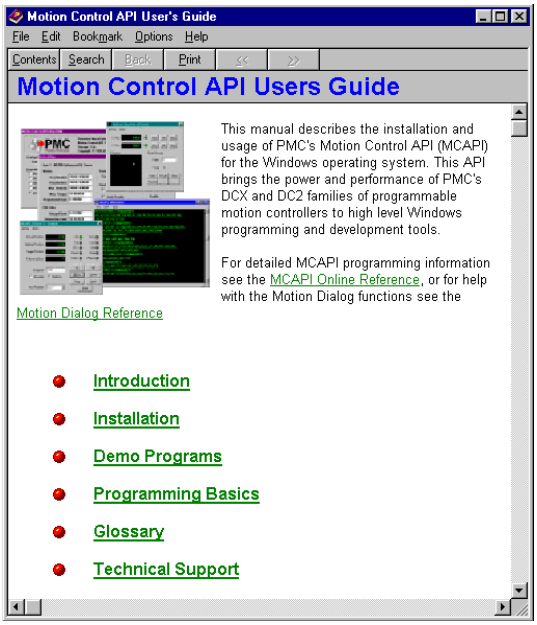


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Motion Control API On-line Help

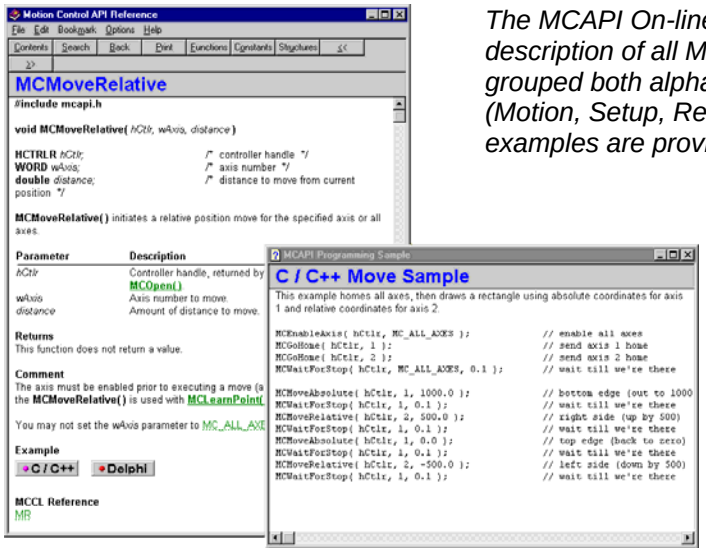
Electronic help files are available for PMC's Motion Control API. Help documents include; installation and basic usage, complete function call reference and examples, high level dialog descriptions, and LabVIEW VI Library reference.





The MCAPI Users Guide On-line Help describes the basics of PMC's MCAPI. This should be the 'first stop' for any questions about the MCAPI.

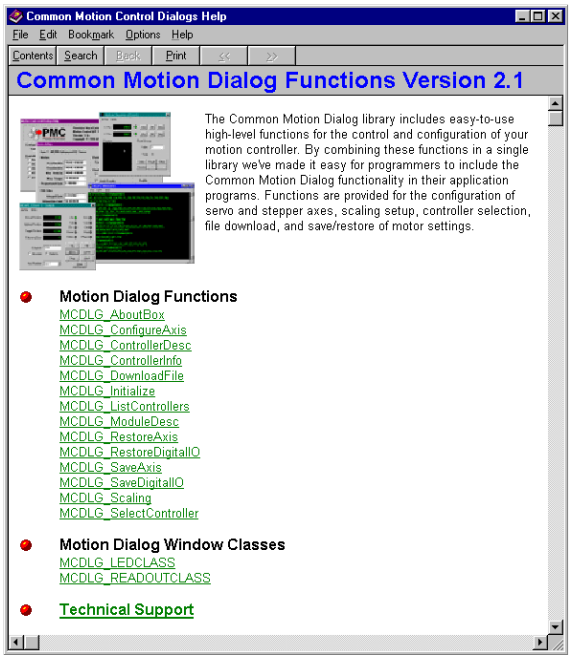




The MCAPI On-line Help provides a complete listing and description of all MCAPI functions. Function calls are grouped both alphabetically and by functional groups (Motion, Setup, Reporting, Gearing, etc...). Source code examples are provided for C++, Visual Basic, and Delphi.

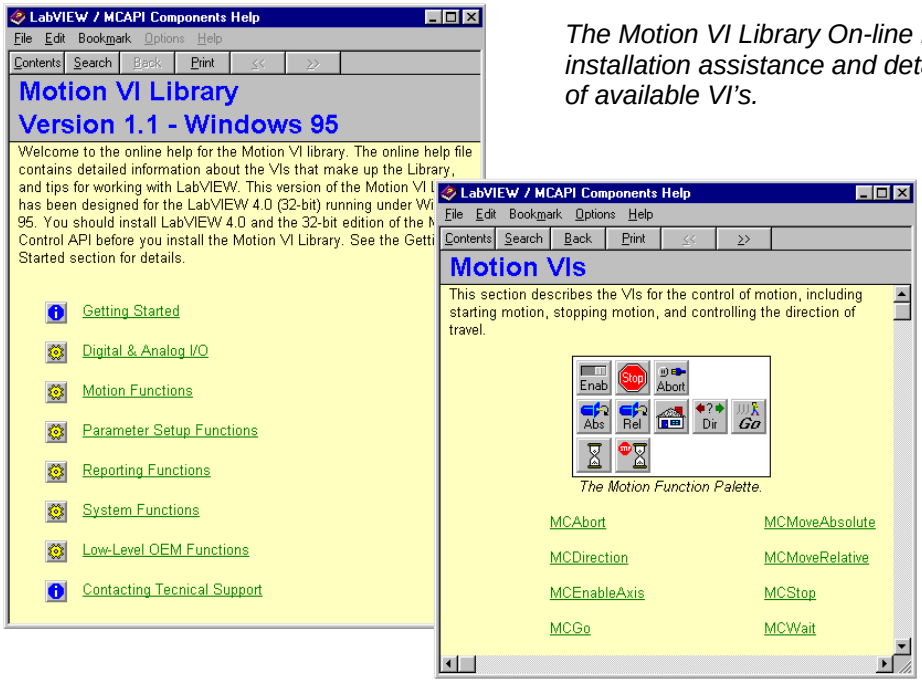
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com





The MCAPI Common Dialog On-line Help describes the high level MCAPI Dialog functions. These operations include: Save and Restore axis configurations (PID and Trajectory), Windows Class Position and Status displays, Scaling, and I/O configuration.





The Motion VI Library On-line Help provides installation assistance and detailed descriptions of available VI's.

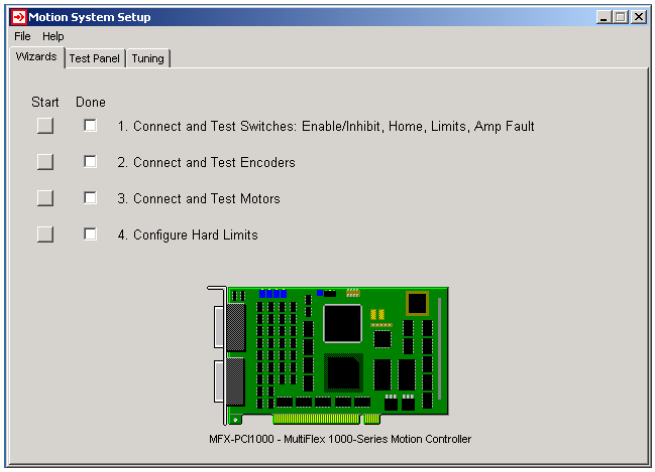
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Motion Integrator

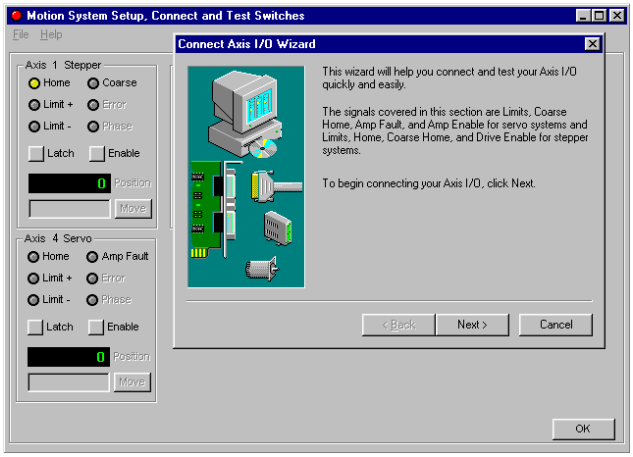
PMC's Motion Integrator program is just like having your own 'Systems Integrator' to assist you with every step of the integration process. Motion Integrator is a suite of powerful Windows tools that are used to:

- Configure the controller
- Verify the operation of the control system
- Connect and test I/O
  - Axis I/O (Home, Limits, Enable)
  - General purpose Digital I/O
  - General purpose Analog I/O
- Tune the servo axes
- Diagnose controller failures
- Execute and plot the results of single and/or multi-axes moves
- Comprehensive on-line help
- Comprehensive on-line help

For first time PMC motion control users, Motion Integrator can be run as a series of Windows Wizards

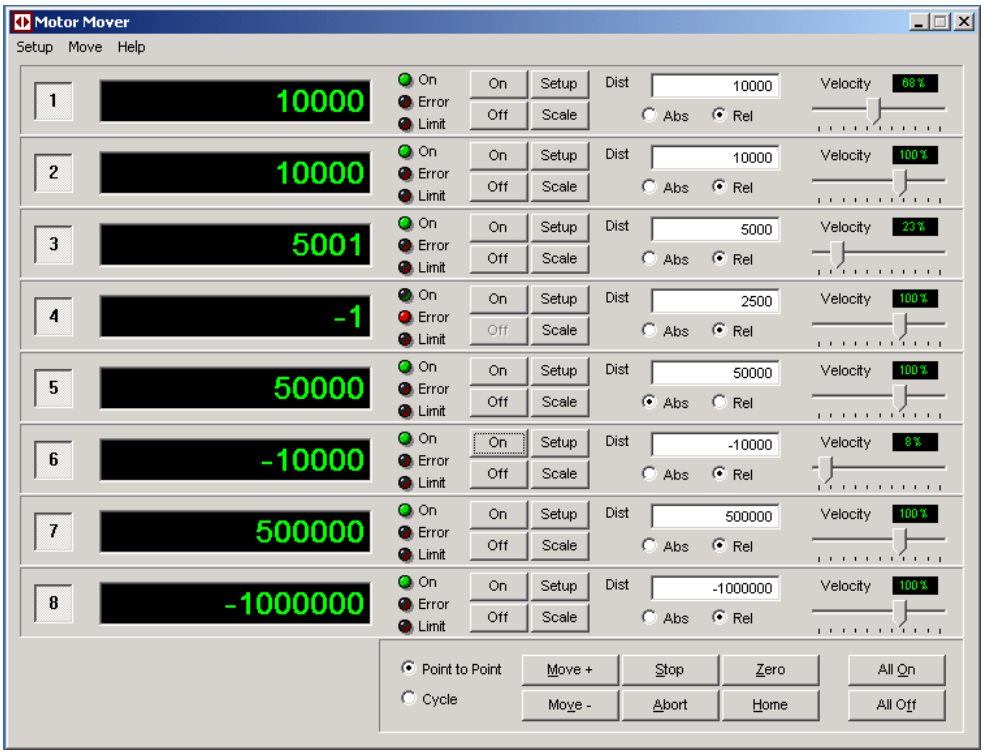


The Motion System Setup program opens with a listing of the recommended integration steps



The Axis I/O wizard allows the user to verify the operation of the Limits, Home, and Amp/Drive Enable

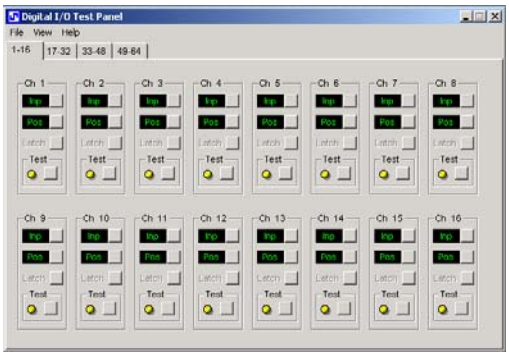
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

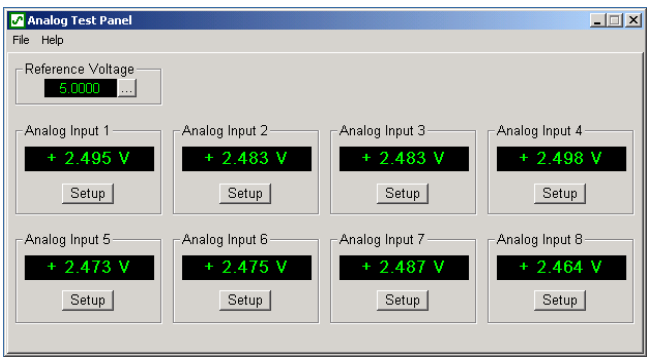


Once the systems has been tested and tuned (servo's only) PMC's Motor Mover allows users to: move any or all motors, change velocities on the fly, define cycling routines, monitor position and status

Digital and Analog I/O Test Panels

The Digital I/O, and Analog Test panels allow the user to verify the operation of the general purpose I/O.





Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

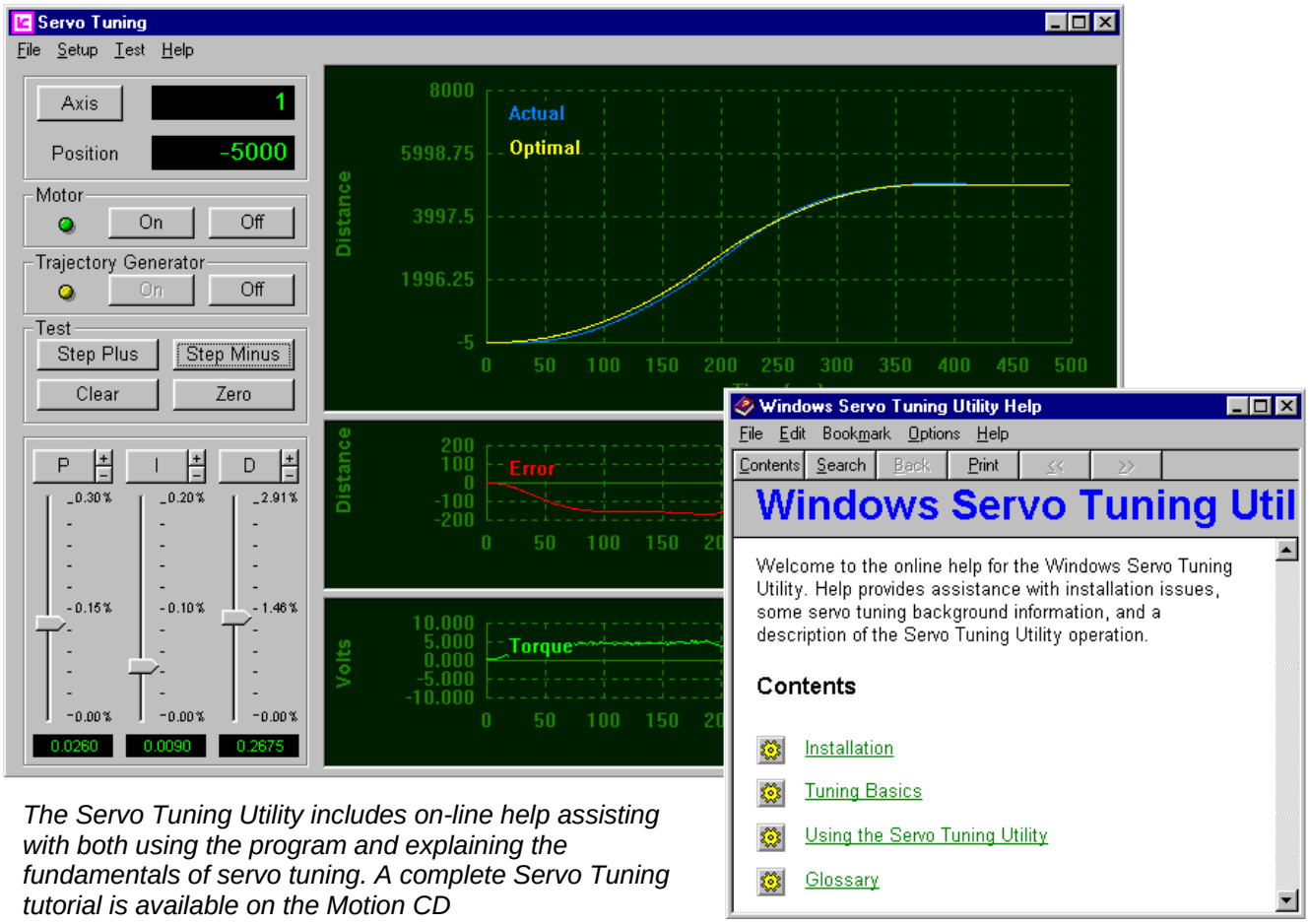
Tuning servo's with Motion Integrator

Motion Integrator provides a powerful and easy to use tool for 'dialing in' the performance of servo systems. From simple current/torque mode amplifiers to sophisticated Digital Drives, Motion Integrator makes tuning a servo is quick and easy.

By disabling the Trajectory generator, the user can execute repeated Gain mode (no ramping - maximum velocity or acceleration/deceleration) step responses to determine the optimal PID filter parameters:

Proportional gain
Derivative gain
Derivative sampling period
Integral gain
Integration Limit

With the Trajectory generator turned on, the user can execute 'real world' moves displaying the calculated position, actual position, following error, and DAC output plots.



The Servo Tuning Utility includes on-line help assisting with both using the program and explaining the fundamentals of servo tuning. A complete Servo Tuning tutorial is available on the Motion CD

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PMC Utilities

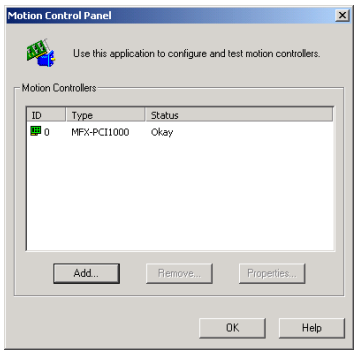
A powerful suite of utilities are included with the Motion Control API. These tools allow the user:

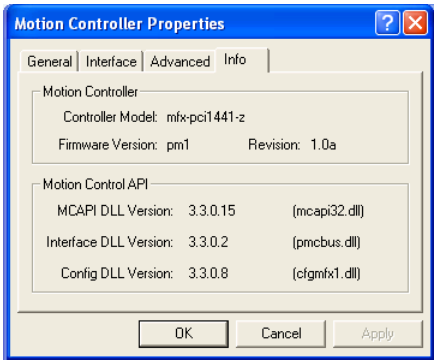
- Query motion control system version information
- Issue native language (MCCL) commands directly to the controller
- Upgrade the firmware of the controller
- Manually position axes with a game port joystick
- Display the Status of an axis

Motion Control Panel

The Motion Control Panel is used to query the motion control system for firmware and Motion Control API version information, and to uninstall a controller. It can be launched either from the Windows **Start** menu or by selecting the Motion Control icon from the Windows Control Panel.



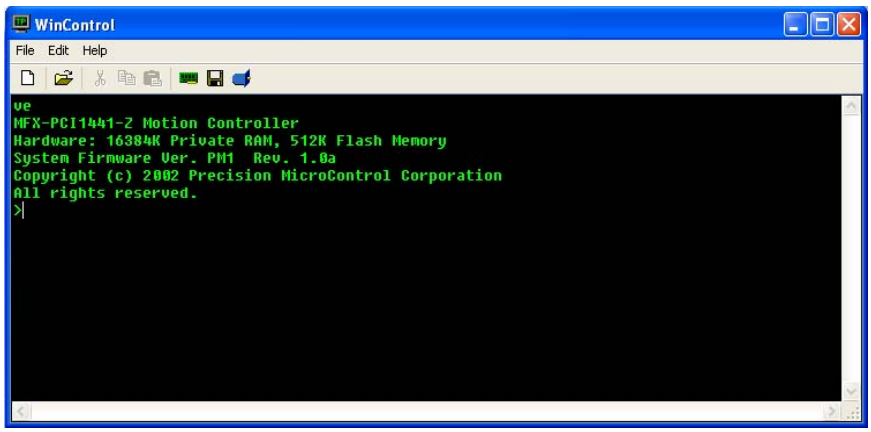




WinControl – MCCL (Motion Control Command Language) command set interface utility

This utility provides the user with a direct communication interface with the controller in its native language (MCCL). This tool is extremely useful not only during initial controller integration but also as a debug tool during application software development. Two methods of executing MCCL commands are supported: A PC keyboard key stroke is passed directly to the controller, and/or download a MCCL command text file via the **File** – **Open** menu options





Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Flash Wizard

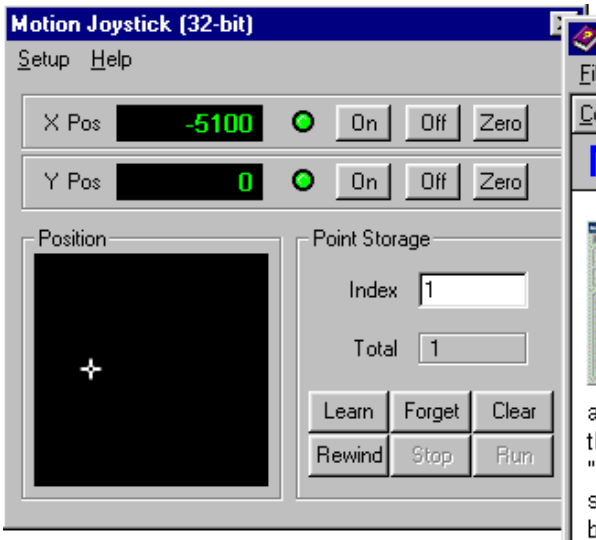
All operational program code (firmware) for the MultiFlex PCI Series controllers is stored in non-volatile memory on-board the controller. PMC's Flash Wizard is a windows application that allows users to easily upgrade controller firmware via software. Users can download the latest firmware revisions from the Support page of PMC's web site at *www.pmccorp.com/support/support.php*.

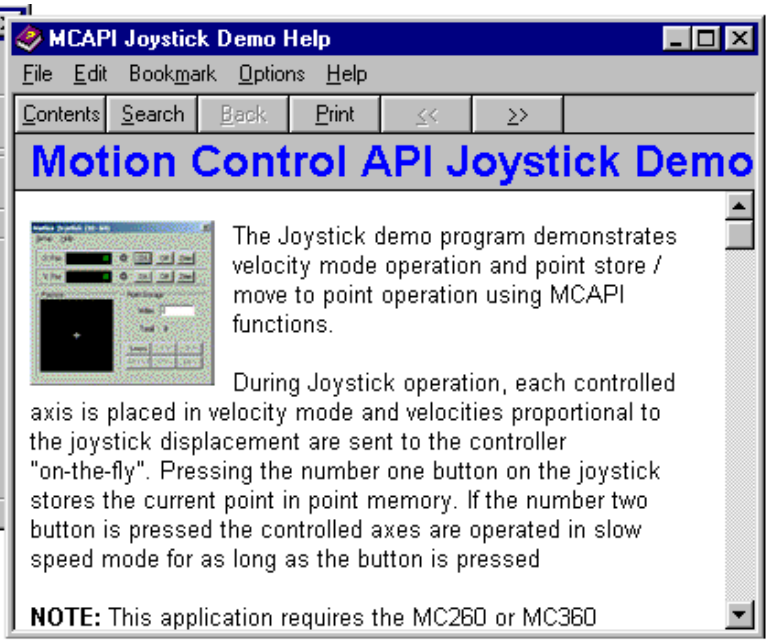




Joystick Applet

Allows the user to manually position two axes using a joystick connected to a USB port on the host PC. Full source code for this applet is provided with the Motion Control API installation.



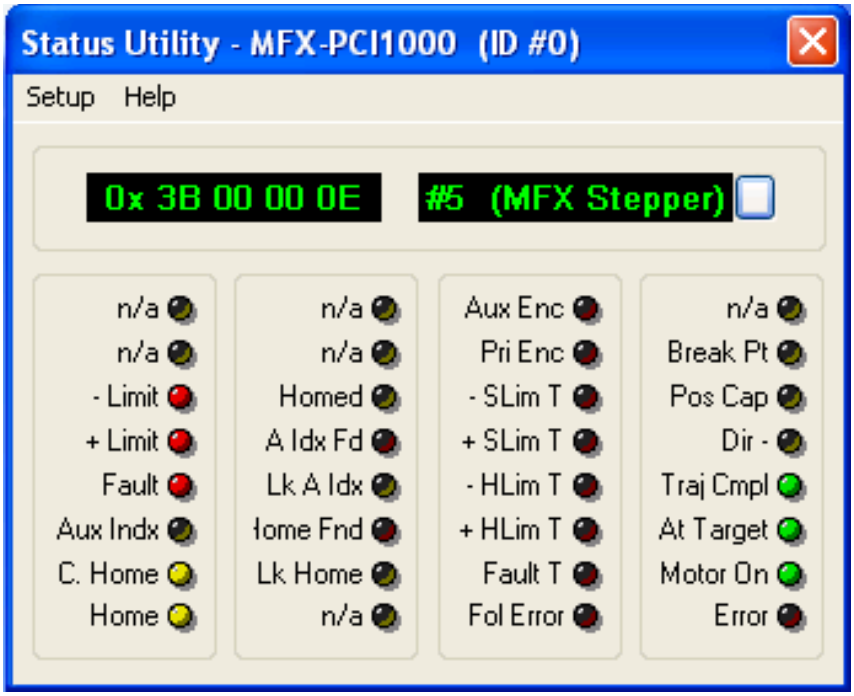


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Status Panel

Allows the user to monitor the status any or all axes (servo or stepper).





Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Chapter

4

Connecting to the Controller

This chapter provides examples of the typical wiring connections and interface circuitry required when using the controller to control the position or velocity of motors and associated I/O events.



For detailed connector and screw-terminal board board signal pin-outs, please see the chapter titled ***Connectors, I/O and Schematics*** beginning on page 177.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

+/- 10V Analog Servo Command Connections

Connectors J1 and J2 each provide two analog command signal pairs for controlling the position of two analog command servos. The 16 bit +/- 10V Analog Command Output signals are available on pins 1 and 2. The Analog Command Return signals are available on pins 35 and 36. The typical interconnections for the +/- 10V analog servo command are shown below.

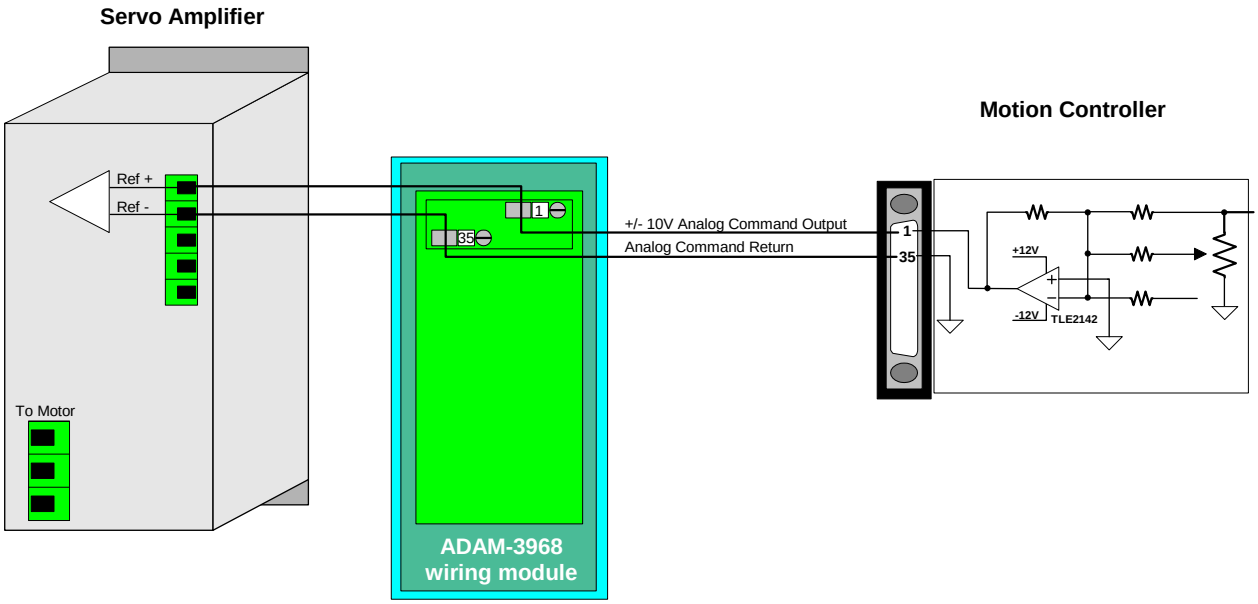


Figure 9. +/- 10V analog command servo wiring example (axis #1)

For Unipolar Analog Command (0.0V to +10.0) Servo Amplifiers an additional connection (to the Direction output) to the amplifier is required to indicate the 'direction of motion'.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PWM (Pulse Width Modulation) Command Connections

Connectors J1 and J2 each provide two PWM command outputs for controlling the position of as many as four PWM command servos. Each output is driven by an Open Collector Driver (75434) and is capable of sinking as much as 100 mA (max. voltage = 30V). The PWM Command outputs are available on pins 6 and 7 of connectors J1 & J2. Any of the Grounds signals can be used as a reference. The typical interconnections for a Bipolar PWM servo are shown below.

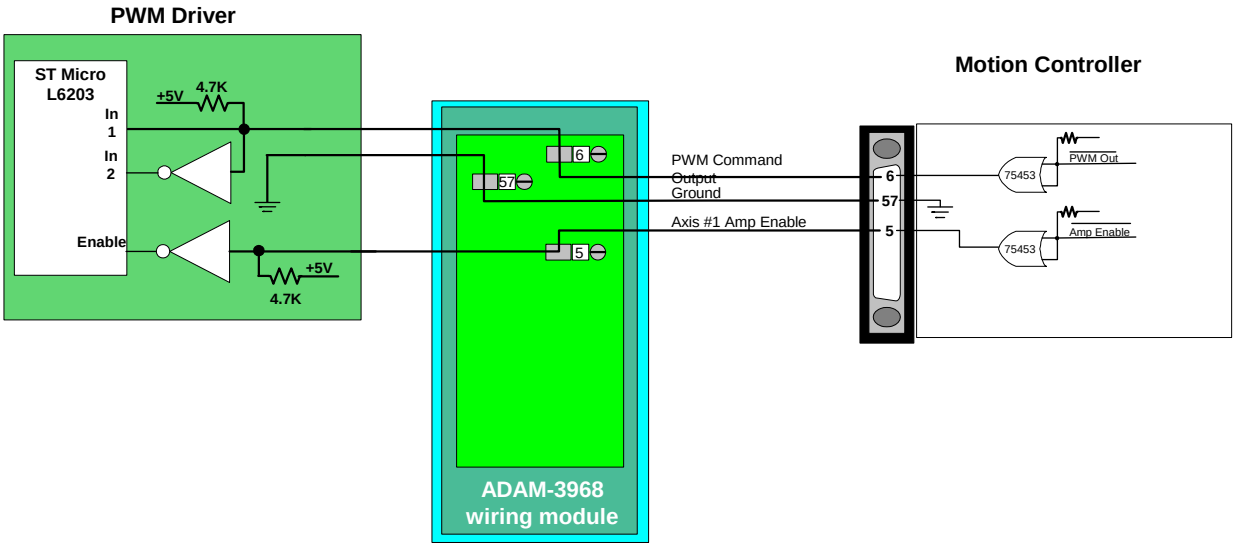


Figure 10. Bipolar PWM command wiring example (axis #1). The ST Micro L6203 is shown for example purposes only - THIS INTERCONNECT DRAWING IS NOT INTENDED TO BE USED FOR CIRCUIT DESIGN.

For information on configuring and operating a PWM servo please refer to the **PWM Command Motion** description in the **Application Solutions** chapter of this manual (page 147 .

Unipolar PWM

A unipolar PWM requires both a PWM Command (Magnitued) signal and a Direction (Sign) signal. Due to I/O limits the controller does not provide a dedicated PWM Direction output, but any of the general purpose Digital Outputs can be configured to this function. For information on configuring the Digital Outputs please refer to page 165. The typical interconnections for a Unipolar PWM servo are shown below.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

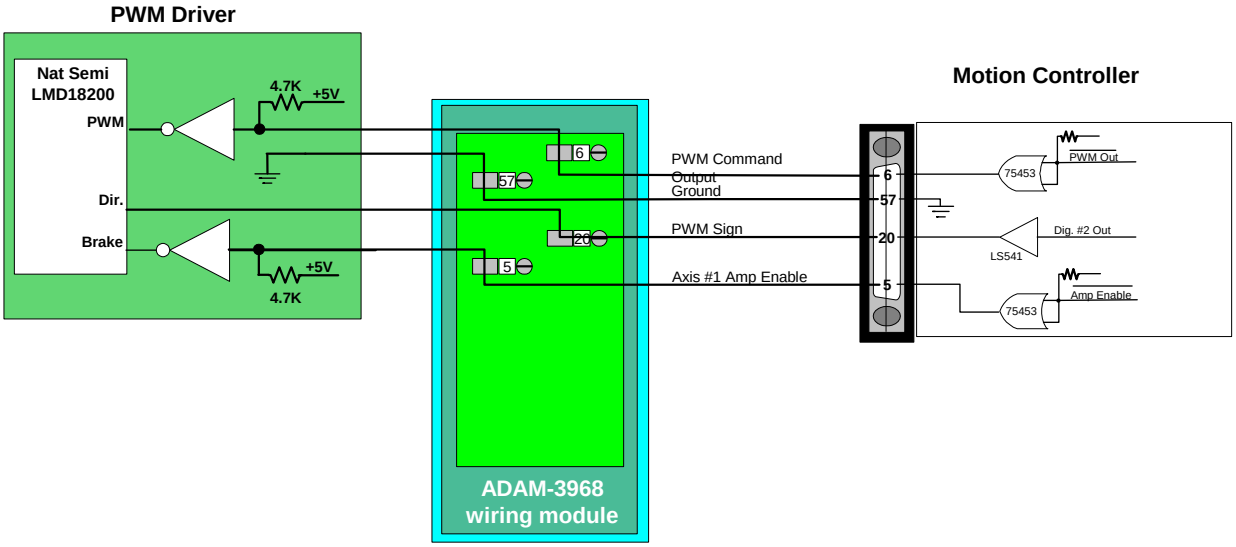


Figure 11. Unipolar PWM command wiring example (axis #1). For this example the PWM Direction output is configured to use Digital Output #2. The National Semi LMD18200T is shown for example purposes only - THIS INTERCONNECT DRAWING IS NOT INTENDED TO BE USED FOR CIRCUIT DESIGN.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Pulse Command Connections

Connectors J3 and J4 each provide two command signal pairs for controlling the position of two stepper motors or two pulse command servo axes. The Step command output is available on pins 2 and 7. A +5 VDC opto isolator supply is available on pins 36 and 41. The Direction command output is available on pins 3 and 8. A +5 VDC opto isolator supply is available on pins 37 and 42.

For Pulse command axes that require Clockwise / Counter Clockwise control signals (instead of Step / Direction) the Motion Control API function ***MCSetModuleOutputMode( )*** will allow the user to reconfigure the pulse command output signals.

The typical interconnections for a pulse command axis are shown below.

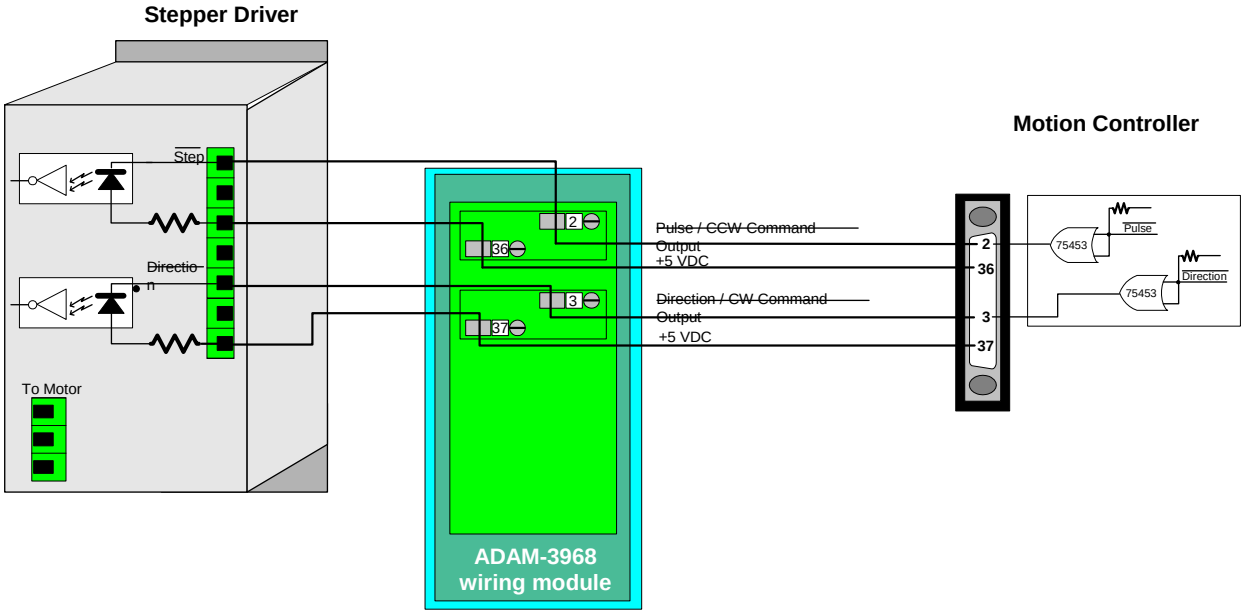


Figure 12. Pulse (Step & Direction) command wiring example (axis #5)

Note:
1) Not all drivers / amplifiers provide an optical isolator current limiting resistor. An external current limiting resistor can be added between the terminal strip contact and the input contact on the driver / amplifier.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Amplifier / Driver Enable Connections - Low Active

Connectors J1 and J2 each provide two Amplifier Enable command signal pairs. The typical interconnections for Low Active Amplifier Enable are shown below.

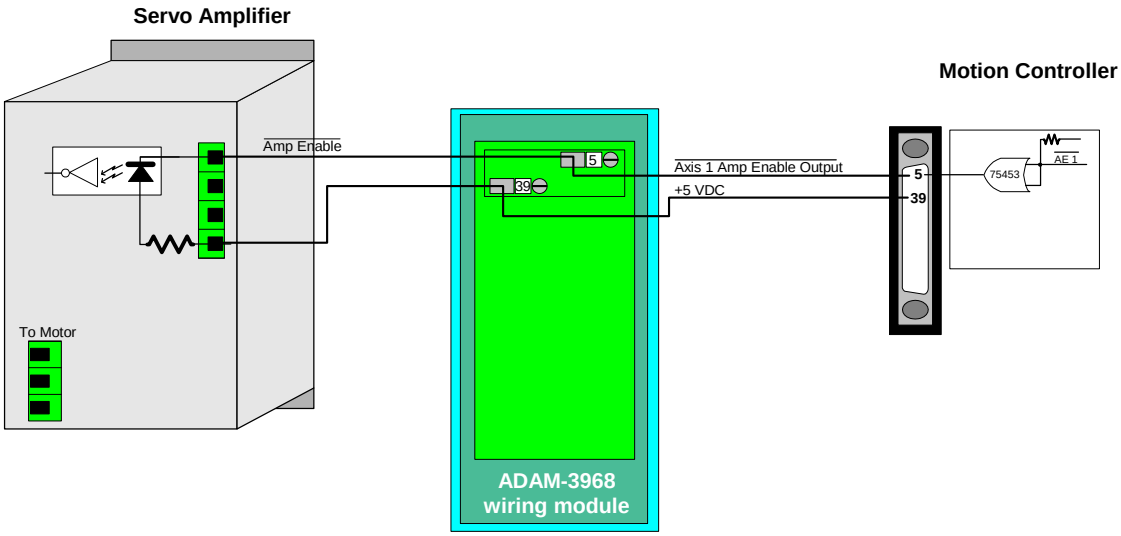


Figure 13. Low Active Amplifier Enable wiring example (axis #1)

Note:
1) Not all drivers / amplifiers provide an optical isolator current limiting resistor. An external current limiting resistor can be added between the terminal strip contact and the input contact on the driver / amplifier.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Driver Disable Connections - Low Active

Connectors J3 and J4 each provide two Driver Disable command signal pairs. The typical interconnections for Low Active Driver Disable are shown below.

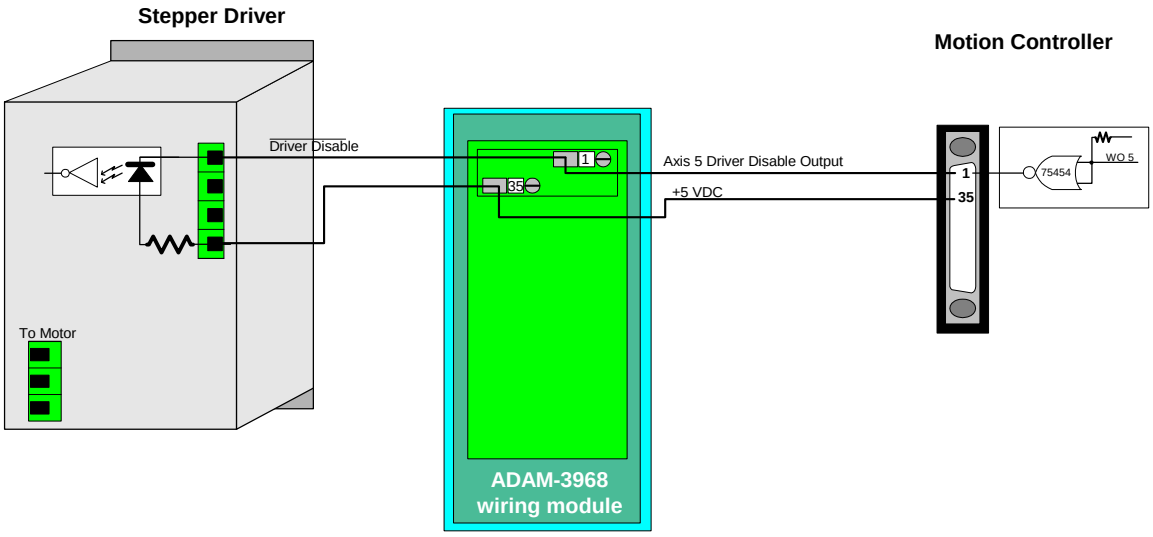


Figure 14. Low Active Amplifier Enable wiring example (axis #5)

Note:
1) Not all drivers / amplifiers provide an optical isolator current limiting resistor. An external current limiting resistor can be added between the terminal strip contact and the input contact on the driver / amplifier.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Amplifier / Driver Enable Connections - High Active

The controller uses open collector drivers (TI SN 75453B) for the Amplifier Enable/Driver outputs. These are current sinking devices which, when turned on, will 'pull' the Amplifier Enable output low (near ground). Until the axis has been enabled by the user an internal resistor forces the Amplifier Enable output to its inactive state (high). This type of circuit provides fail safe operation of 'low' active Amplifier/Driver Enable systems .

For applications that require 'high' active Amplifier/Driver Enable outputs, once **Windows has loaded and an application program has been launched**, the ***MCConfigureDigitalIO( )*** function can be used to change the active level of the Amplifier/Driver Enable outputs.



Warning – High Active Amplifier/Driver Enable is not a fail safe operation.

The typical interconnections for High Active Amplifier Enable are shown below.

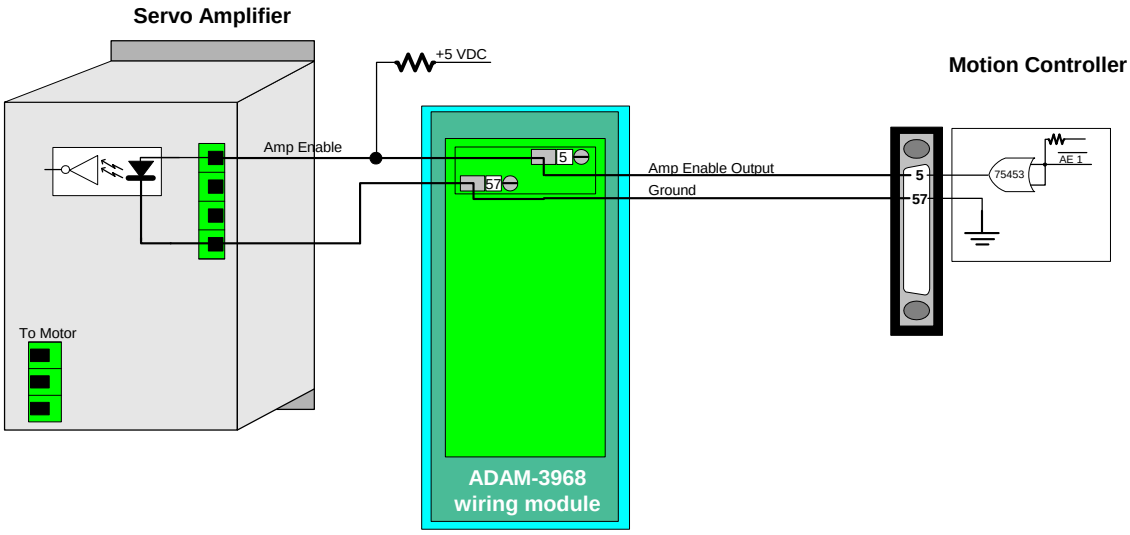


Figure 15. High Active Amplifier Enable output wiring example (axis #1)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Amplifier / Driver Fault Connections

The controller provides four optically isolated inputs for interfacing to an Amplifier / Driver fault sensor. Bi-directional optical isolators are used, so the external device used to indicate an Amplifier/Driver Fault may be either a sinking or sourcing device. By default each Amp Fault circuit is shared between an Analog Command axis and a Pulse Command axis (Amp Fault 1 is shared by axes 1 and 5, Amp Fault is shared by axes 2 and 6, etc...).

The maximum voltage that can be applied to an Amplifier / Driver Fault input is 25V. The minimum voltage that will cause the optical isolator to conduct is 3.0V. The Amplifier / Driver Fault sensor must be capable of sinking/sourcing at least 0.25 mA.

This wiring example shows a 'high' active (sourcing) Amplifier Fault circuit. For a 'low' active Amplifier Fault circuit:

1) The switch connects the Amplifier Fault input to ground
2) Connect a amplifier DC power supply (3.0 VDC to 25 VDC) to Amp Fault supply/return

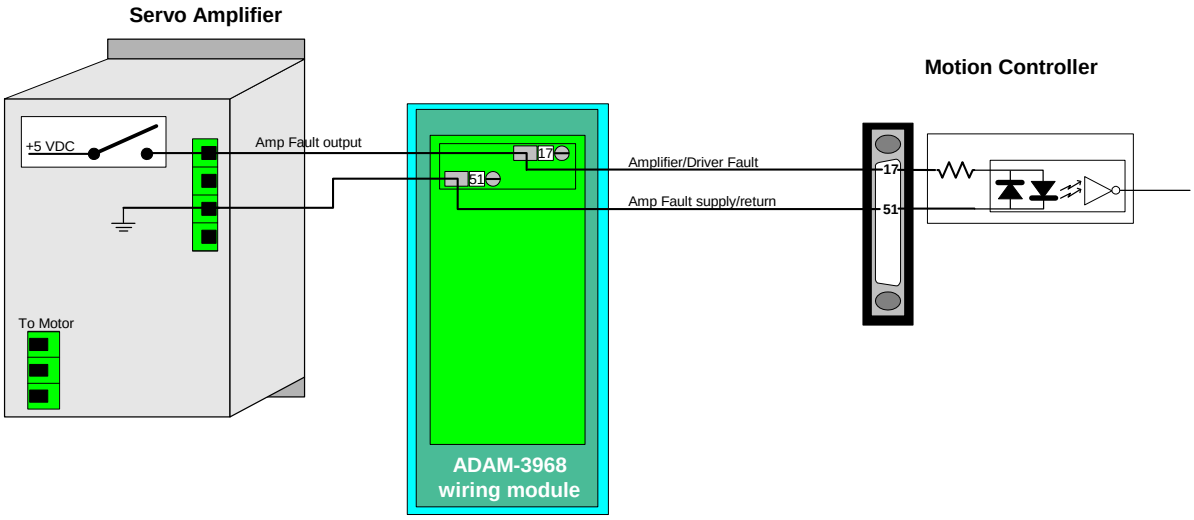


Figure 16. High Active Amplifier Fault input wiring example (axis #1)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Differential Incremental Encoder Connections

Each of the four VHDCI connectors provide two incremental encoders interfaces. These encoder interfaces support either differential encoders (A+, A-, B+, B-, Z+, and Z-) or single ended encoders (A, B, and Z). When differential encoders are used the controller supports hardware encoder error detection.

For additional information on incremental encoder basics please refer to page 24.

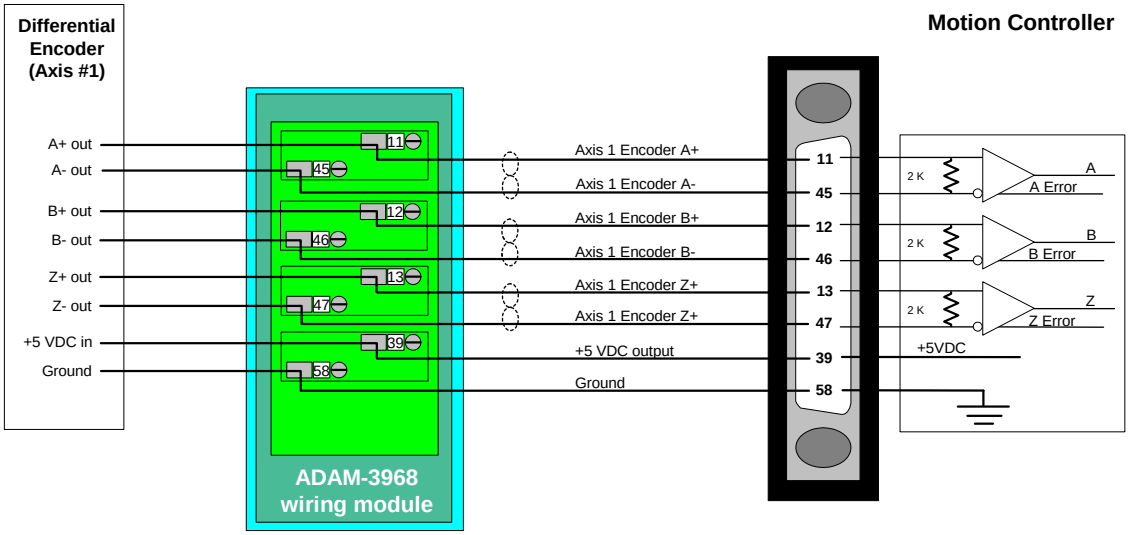


Figure 17. Differential encoder wiring example (axis #1)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Single Ended Incremental Encoder Connections

Each of the four VHDCI connectors provide two incremental encoders interfaces.



As shown in the drawing below the unconnected encoder inputs (A-, B-, and Z-) must be tied to the +1.5 VDC Encoder Reference.

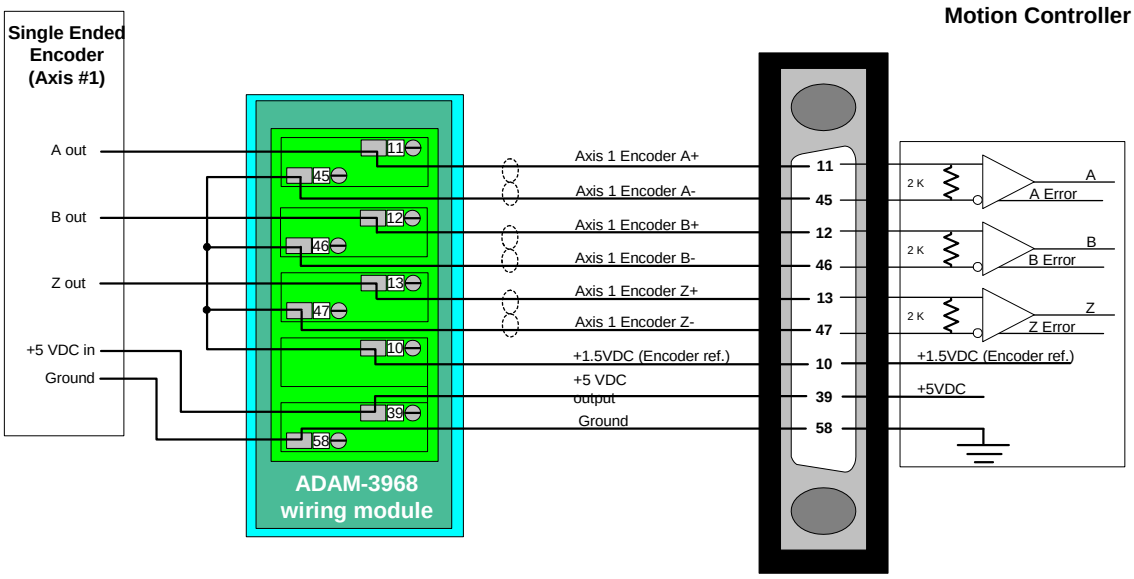


Figure 18. Single ended encoder wiring example (axis #1)



The wiring example above assumes that the encoder outputs (A, B, & Z) are 'high active'. If any of the encoder outputs (most likely the Z output) are 'low active' it should be connected to the '-' input of the MultiFlex and the '+' input should be terminated to the +1.5V reference.



Depending on the current drive capability of the encoder the controller may not support Encoder Fail Detection for single ended encoders. For additional information please contact PMC Tech Support.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Over-Travel Limit Connections

Sourcing Sensor

The controller provides eight optically isolated inputs for monitoring over-travel limit sensors. Bi-directional optical isolators are used, so the over travel sensors may be either sinking or sourcing devices.

By default each over-travel limit input is shared between an Analog Command axis and a Pulse Command axis (a Limit + is shared by axes 1 and 5, a Limit + is shared by axes 2 and 6, etc...). The maximum voltage that can be applied to an over travel limit input is 25V. The minimum voltage that will cause the optical isolator to conduct is 3.0V. The over travel limit sensor must be capable of sinking/sourcing at least 0.25 mA.

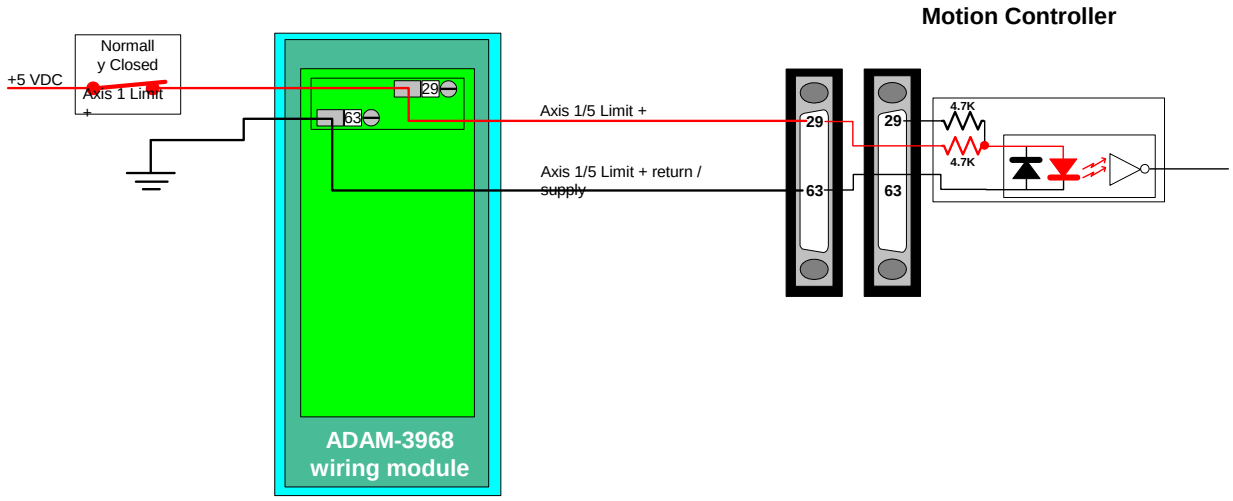


Figure 19. Sourcing over travel limit sensor wiring example

This example uses Normally Closed switches for Fail Safe operation. If the switch is opened or a wire is broken a change of state will be indicated. To configure to controller for Normally Closed switches issue the function ***MCConfigureDigitalIO( )*** with *mode value* = **MC_DIO_LOW**.

In Position and Velocity mode the response to an activated limit input is direction sensitive, the axis will only be stopped if it is moving in the direction of the activated limit switch. In Contour mode, the axis will be stopped regardless of the direction it is moving if a limit is activated. In Torque mode, the controller will ignore the activation of a limit input.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Sinking Sensor

The controller provides eight optically isolated inputs for monitoring over-travel limit sensors. Bi-directional optical isolators are used, so the over travel sensors may be either sinking or sourcing devices.

By default each over-travel limit input is shared between an Analog Command axis and a Pulse Command axis (a Limit + is shared by axes 1 and 5, a Limit + is shared by axes 2 and 6, etc...). The maximum voltage that can be applied to an over travel limit input is 25V. The minimum voltage that will cause the optical isolator to conduct is 3.0V. The over travel limit sensor must be capable of sinking/sourcing at least 0.25 mA.

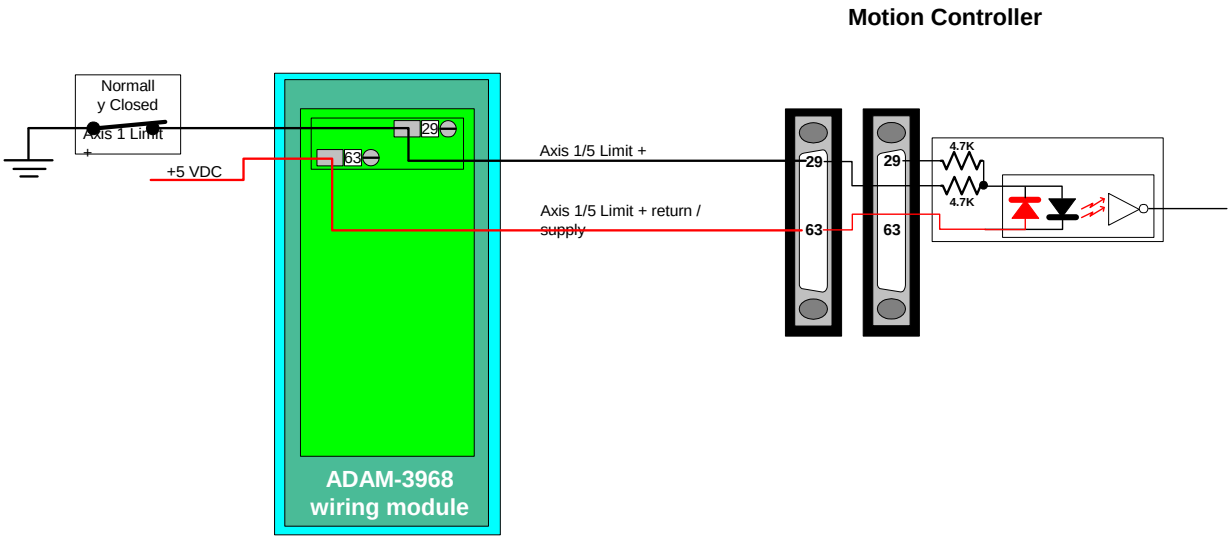


Figure 20. Sinking over travel limit sensor wiring example

This example uses Normally Closed switches for Fail Safe operation. If the switch is opened or a wire is broken a change of state will be indicated. To configure to controller for Normally Closed switches issue the function ***MCConfigureDigitalIO( )*** with *mode value* = **MC_DIO_LOW**.

In Position and Velocity mode the response to an activated limit input is direction sensitive, the axis will only be stopped if it is moving in the direction of the activated limit switch. In Contour mode, the axis will be stopped regardless of the direction it is moving if a limit is activated. In Torque mode, the controller will ignore the activation of a limit input.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Home Sensor Connections

The controller provides four optically isolated inputs for defining the home position of an axis. Bi-directional optical isolators are used, so the sensors may be either sinking or sourcing devices.

By default each Coarse Home / Stepper Home input is shared between an Analog Command axis and a Pulse Command axis (a Coarse Home / Stepper Home is shared by axes 1 and 5, a Coarse Home / Stepper Home is shared by axes 2 and 6, etc...). The maximum voltage that can be applied to an over travel limit input is 25V. The minimum voltage that will cause the optical isolator to conduct is 3.0V. The over travel limit sensor must be capable of sinking/sourcing at least 0.25 mA.

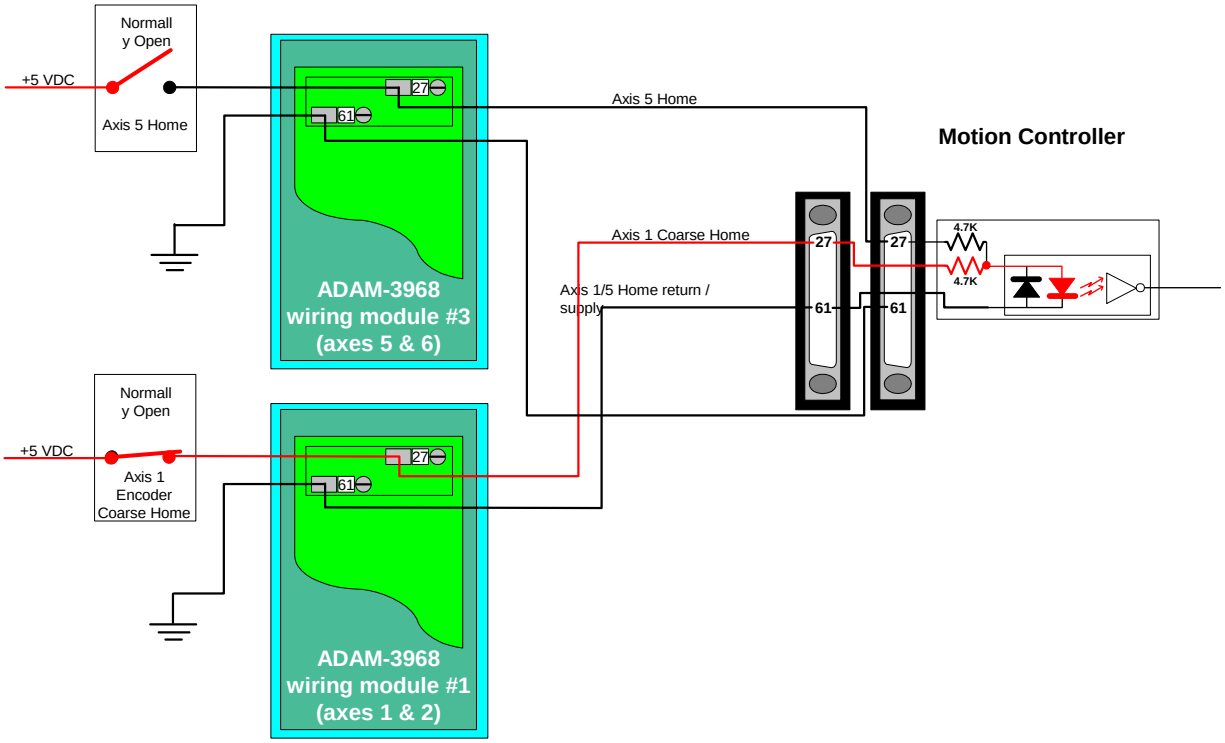


Figure 21. Home sensor wiring example (axes #1 & #5)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TTL Digital Input Connections

Each of the four VHDCI connectors provide four TTL level digital input channels that can be used to monitor external events. An 74LS541 is used as the buffering device. An board 10K ohm pull resistor is provided for each channel.

> Warning – Voltage levels outside valid TTL levels applied to the TTL inputs may damage the controller.

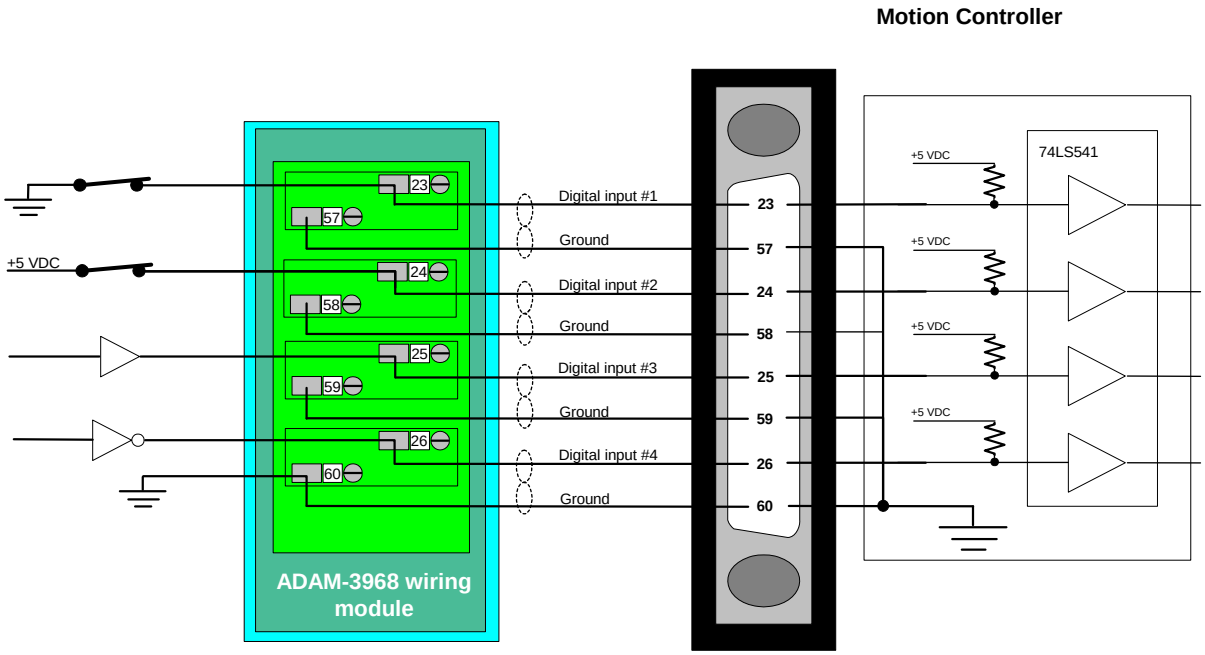


Figure 22. TTL digital inputs wiring example (first 4 channels)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TTL Digital Output Connections

Each of the four VHDCI connectors provide four TTL level digital output channels that can be used to activate external devices. A 74LS541 is used as the buffering device. On power up (until the controller has completed initialization - 30 seconds to 3 minutes) all outputs will go to a TTL 'high' level. When initialization is complete the outputs will change to a TTL low state. The default state for the digital outputs is for 'high true' (sourcing) logic.

When used as sinking (TTL low) outputs each channel can sink a maximum of 24 mA. which is suitable for driving TTL loads, low current optical isolators, and low current solid state switches.
To configure a digital output for sinking (TTL low) use the function ***MCConfigureDigitalIO( )***.
Note: once initialization is complete, a sinking output cannot 'turned off' (set to a TTL high) until an application program has been launched.



Warning – Attempting to drive a load that exceeds the current drive capability of the 74LS541 may damage the controller.

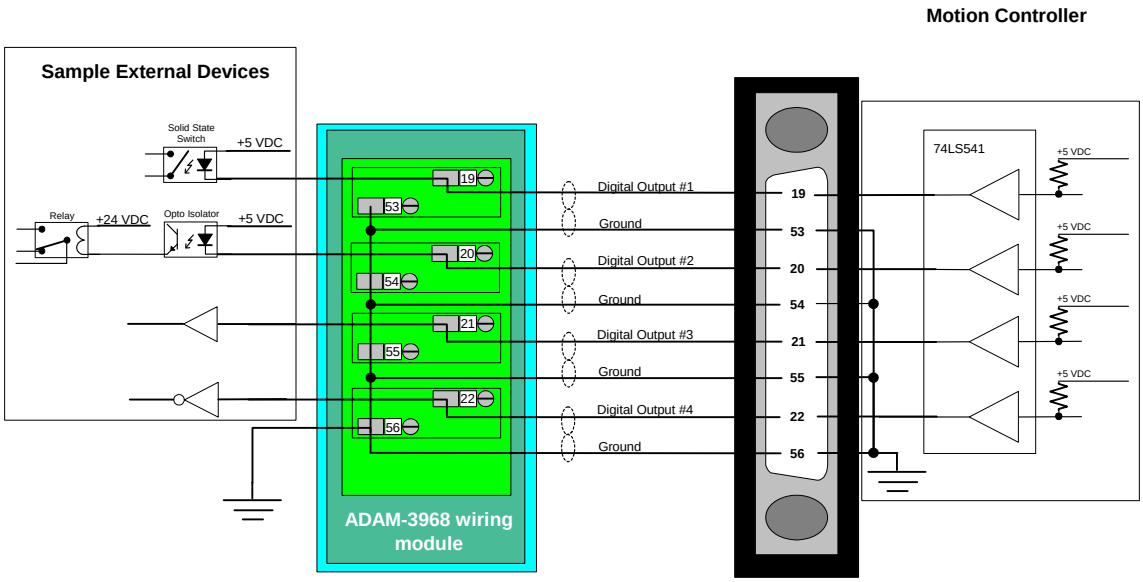


Figure 23. TTL digital outputs wiring example (first 4 channels)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

A/D Input Connections wiring example

If the Analog Input option is present each of the four VHDCI connectors provide two 14 bit A/D channels (total of 8). The A/D option can be ordered with a voltage range of either:

-10V to +10V (standard)
0V to +4V (special order)

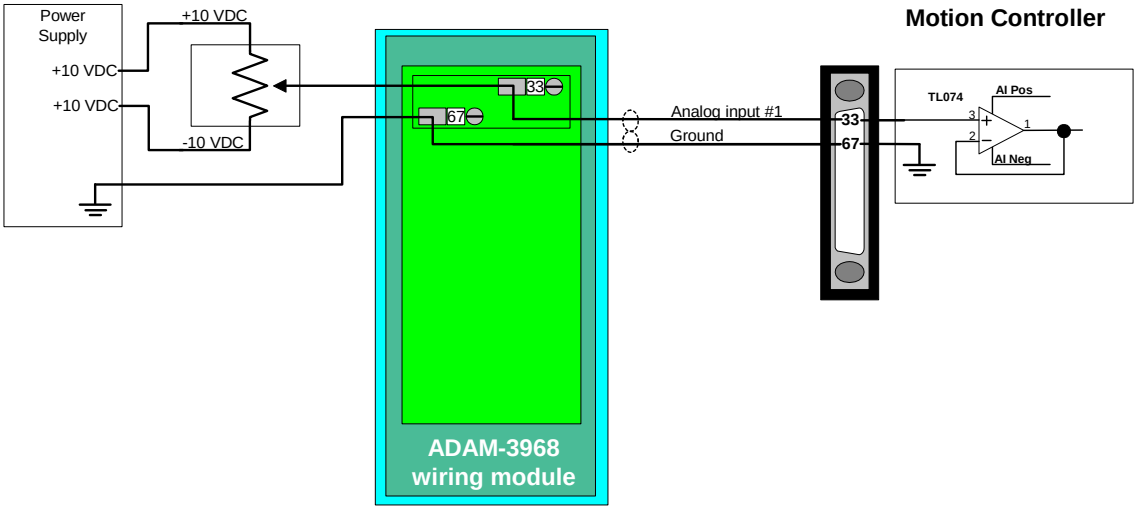


Figure 24. A/D inputs wiring example

Referring to the drawing above:

If configured for +/- 10V A/D input range then AIPos = +12 VDC and AINeg = -12 VDC
If configured for 0 - 4V A/D input range the AIPos = +4 VDC and AINeg = Ground

For more information about how to read the analog input values, please refer to the section titled “A/D Inputs” on page 170.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Watchdog Relay Connections

The controller incorporates a watchdog circuit and relay to protect against improper CPU operation. After a controller reset, PC reset, or PC power cycle, once the controller is initialized (Run LED D3 on) the watchdog circuit is enabled and the normally open watchdog relay is energized (contacts closed).

For some applications it is required that the motion controller watchdog circuit be hard wired into the power distribution system. The diagram below details how the watchdog relay can be used to disable amplifier power in the case of a controller watchdog failure.

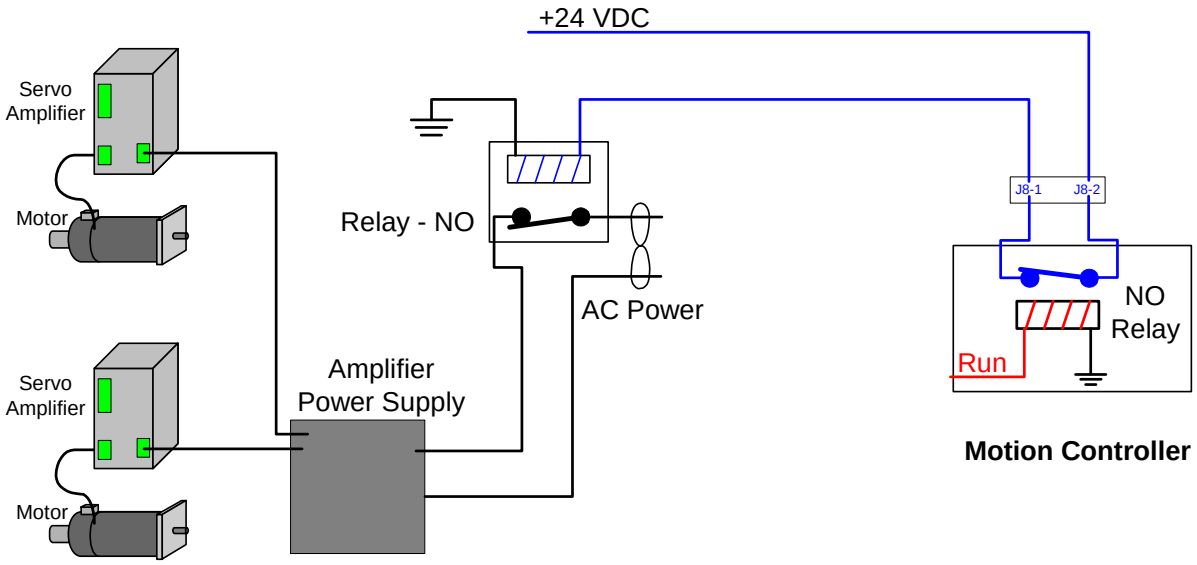


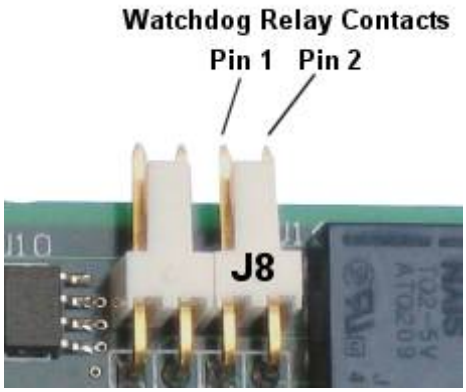
Figure 25. Watchdog relay wiring example

Watchdog relay contact specifications

Max. switching power = 30W
Max. switching current = 1A
Max. switching voltage: DC = 110V, AC=125V

J8 Mating connector:

Pin Housing: Molex P/N 22-01-3027
Crimp pin: Molex P/N 08-50-0114



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Chapter 5

Motion Control

This chapter describes the basic motion control operations that can be performed by the motion controller. The operations described in this chapter are common to both servo and stepper motors, with specific differences detailed in the text.

Servo (analog command) Axis Setup

The basic steps required to implement closed loop servo motion are:

- Verify proper encoder operation
- Setting the allowable following error
- Define trajectory parameters
- Tuning the servo (select PID filter gain parameters)

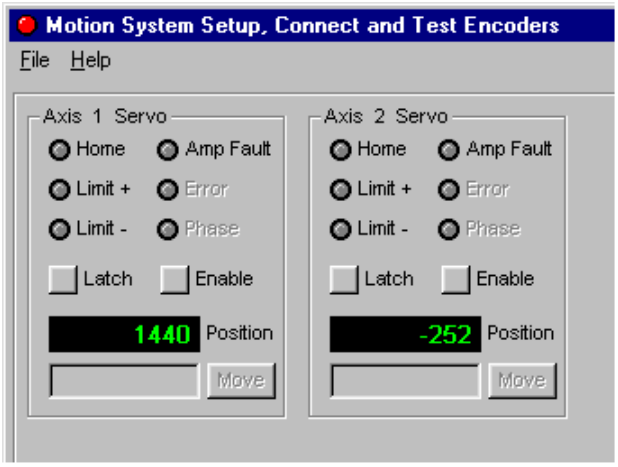
Verify proper encoder operation

The Motion Integrator program provides easy to use tools for testing the operation of an encoder. The user has the option of using the Connect Encoder Wizard or the Motion System Setup Test Panel.



Unlike the **Connect Encoder Wizard**, the **Motion System Setup Test** panel does not allow the user to verify the operation of the encoder Index.





Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

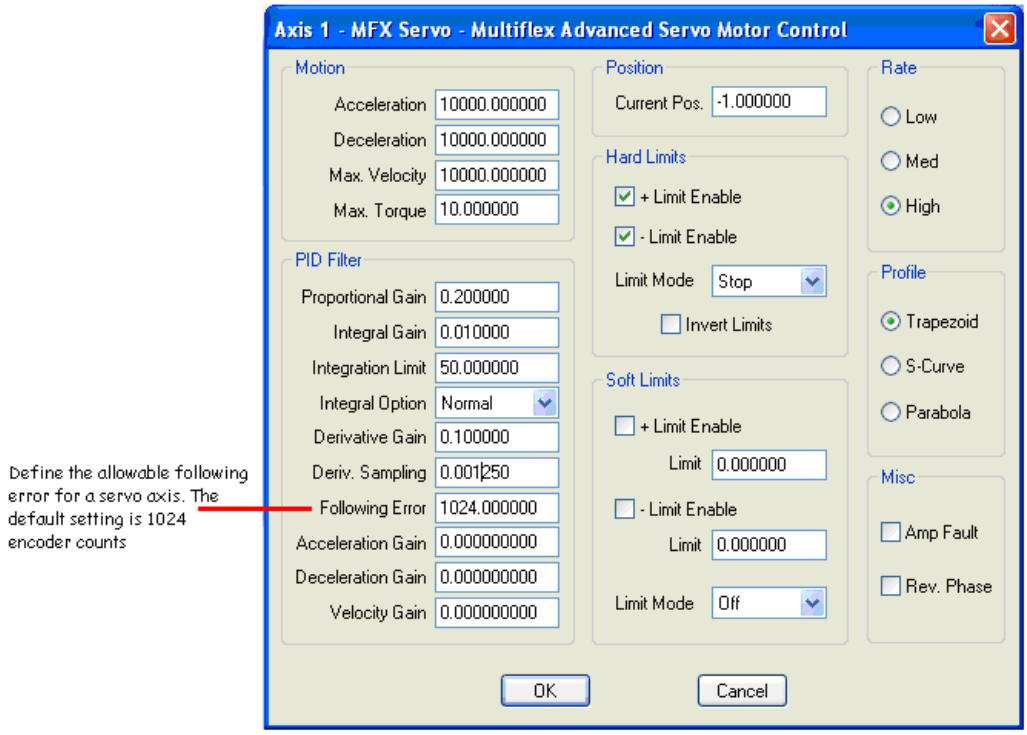
Manually rotate the motor/encoder in either direction, the position reported should increment or decrement accordingly. Refer to the **Troubleshooting Guide** later in this manual if the controller does not report an appropriate change of position.

Setting the Allowable Following Error

Following error is the difference between where an axis **'is'** and where the controller has **'calculated it should be'**. Most all servo systems require 'some' position error to generate motion. When a servo axis is turned on, if a position error exists, the PID algorithm will cause a command voltage to be applied to the servo to correct the error.

While an axis is executing a move, the following error will typically be between 1 and 1000 encoder counts. Very high performance systems can be 'tightly tuned' to maintain a following error within 1 to 10 encoder counts. Systems with low resolution encoders and/or high inertial loads will typically maintain a following error between 150 and 5000 encoder counts during a move.

The controller supports 'hard coded' following error checking. If at anytime the difference between the optimal position and the current position exceeds the user defined 'allowable following error', an error condition will be indicated. The axis will be disabled (Amplifier Enable output turned off, output command signal set to 0.0V) and the axis status word will indicate that an error has occurred. The ***MCEnableAxis( )*** function is used to clear a following error condition. To disable 'hard coded' following error checking set the allowable following error to zero.

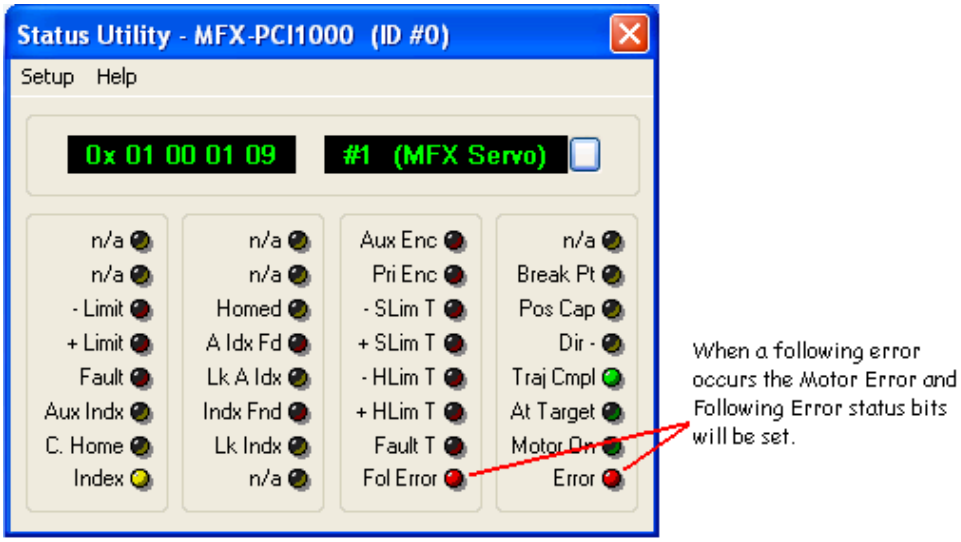


The three conditions that will typically cause a following error are:

1) Improper servo tuning (Proportional gain **too low**)
2) Velocity profile that the system cannot execute (moving too fast)
3) The axis is reversed phased (positive command results in negative motion)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The Status Panel screen shot below shows the typical display when a following error has occurred.



Define trajectory parameters

Prior to issuing any motion commands the user must define the following trajectory parameters:

Maximum Velocity
Acceleration
Deceleration

These parameter values can be defined either by issuing function calls (***MCSetVelocity( )***, ***MCSetAcceleration( )***, ***MCSetDeceleration( )***) or by entering values from the **Servo Setup** dialog (available from **Motor Mover**, **Servo Tuning program**, **CWDemo32.exe**). For additional information on trajectory profiles and velocity profiles please refer to pages 86 and 88.

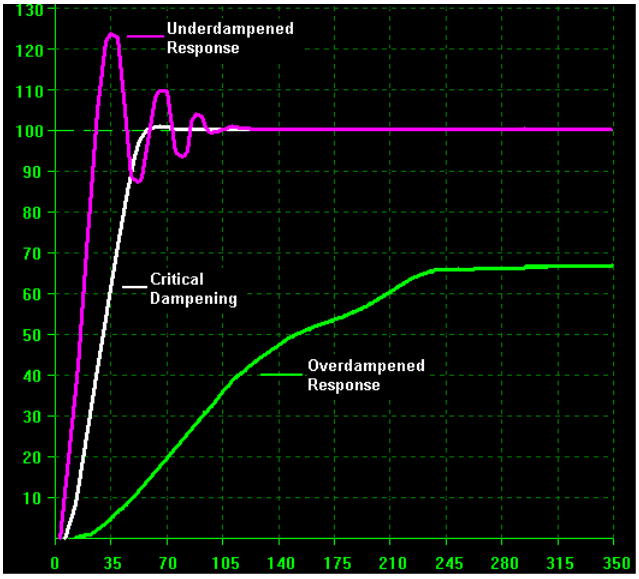
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Tuning the Servo

Servo tuning is the process of setting the digital PID filter gains (proportional, derivative, and integral) to get the best possible performance from an electro mechanical system.

A servo is a closed loop control system. The user commands the motion controller to execute a move of one or more axes. The controller then calculates a velocity profile based on user defined trajectory parameters (maximum velocity, acceleration, and deceleration). Each time the digital PID filter is executed (every 250 usec's.) the difference between the encoder position and the calculated 'desired' is measured and defined as the **Following Error**. Using the following error value, the PID filter adjusts the +/- 10V Analog Command output to reduce the following error.

A servo motor and its load both have inertia, which the servo amplifier must accelerate and decelerate while attempting to follow a change in the input (from the motion controller). The presence of inertia will tend to result in over-correction, with the system oscillating or "ringing" beyond either side of its target (under-damped response). This ringing must be damped, but too much damping will cause the response to be sluggish (over-damped response). Proper balancing will result in an ideal or critically-damped system.



A servo system is tuned by repeatedly executing a ‘step response’ (move of a specific distance), plotting the resulting motion, and adjusting the digital PID filter parameters until an acceptable system response is achieved.

For additional information on tuning a servo please refer to:

PMC Servo Tuning program on-line help
Servo Tuning PowerPoint tutorials (on the Motion CD)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

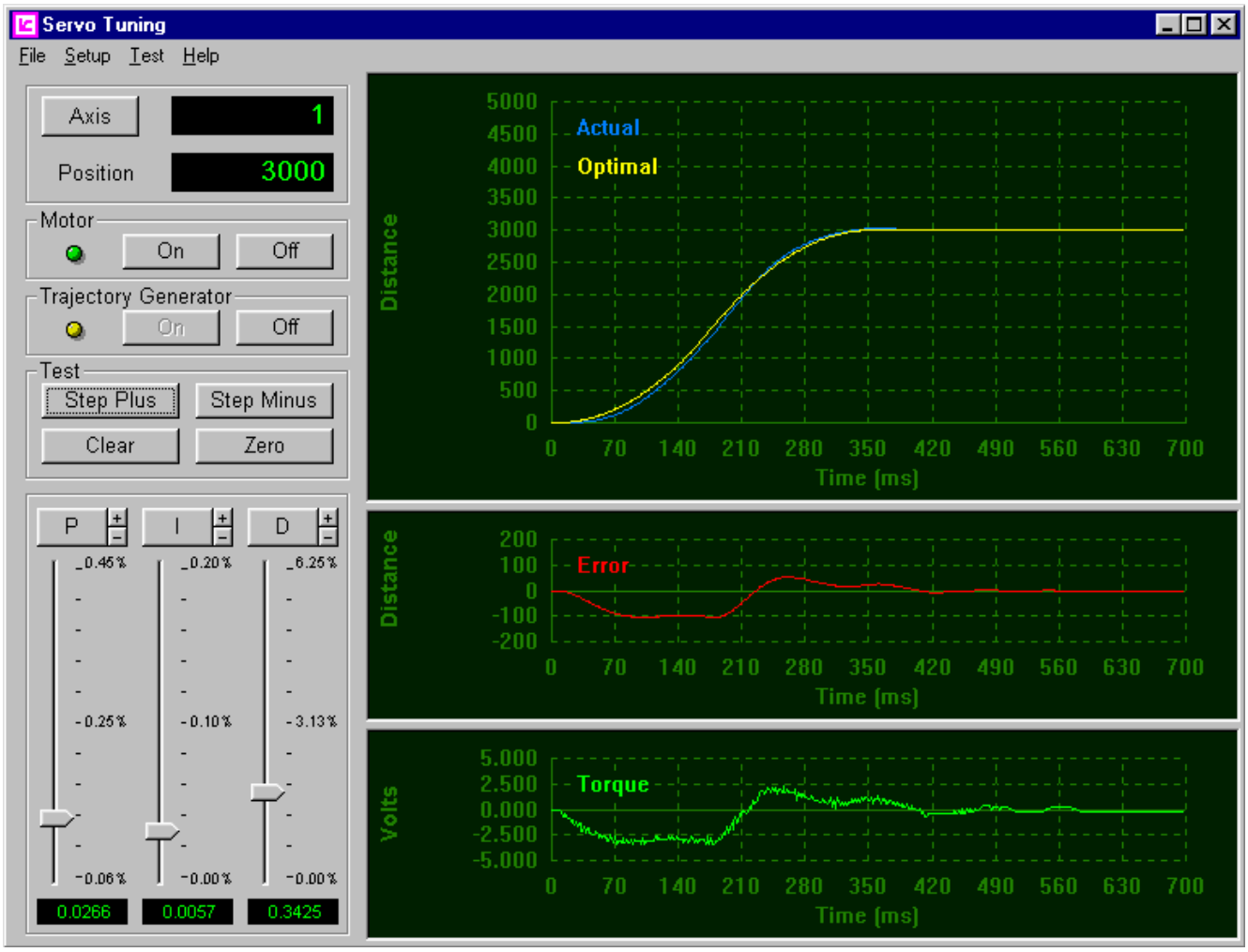
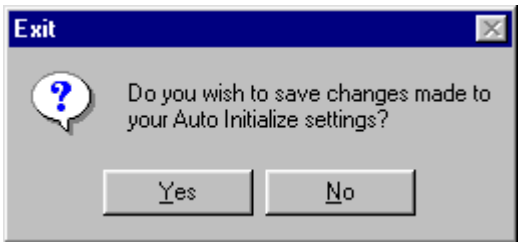


Figure 26. PMC's Servo Tuning Utility is used to: execute moves, plot the response of the servo, and adjust PID gains

Saving the Tuning Parameters

Once an axis has been tuned you should save the PID and trajectory parameters. Select **Save All Axis Settings** from the **File** menu. Selecting this option will load all servo settings into the mcapi.ini file (in the C:\Windows folder). In addition when you elect to close the Servo Tuning program it will prompt the user about saving the settings.



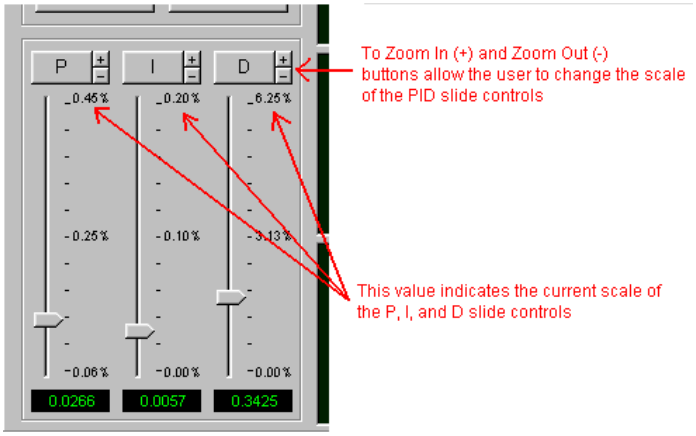
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Electing to save the Auto Initialize settings causes the Servo Tuning utility to call the Motion Control API Common Dialog function **MCDLG_SaveAxis**. All servo parameters (PID, Trajectory, Limits, etc...) will be saved in the dialog

To define these servo parameters from a user's application program, call the Motion Control API Common Dialog function **MCDLG_RestoreAxis**.

Changing the Scale of the Slide Controls

At the top of each slide control is a value showing the current setting as a percentage of the current maximum setting. To change the range of one or more slide controls select the Zoom In (+) or Zoom Out (-) buttons.



Executing cycle operations from the Servo Tuning program.

Beginning with revision 2.4 the servo tuning program allows the user to execute cycle operations. From the Test Setup dialog define the move distance, dwell between positive and negative moves, cycle repeat count, and dwell between cycles.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

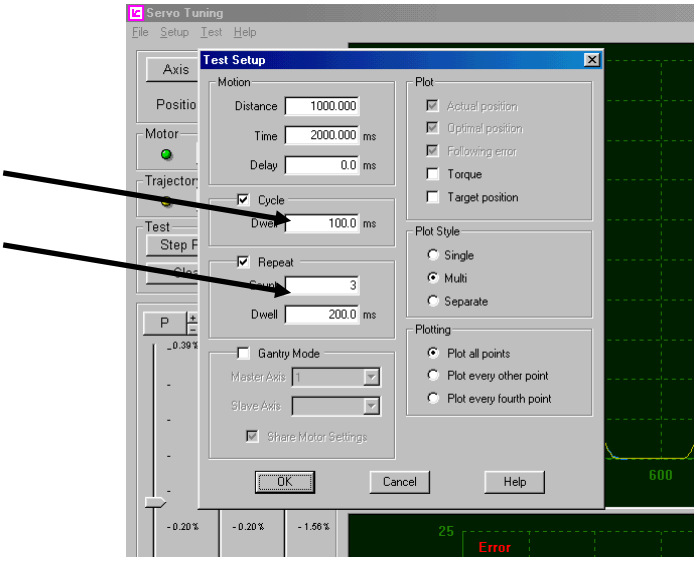


Figure 27. Use the Test Setup dialog to configure the distance, dwell, and repeat count of cycle operations

Tuning Velocity Mode Amplifier Servo Systems

A velocity mode amplifier incorporates an analog tachometer to provide the feedback for the velocity loop, which is closed within the amplifier. The velocity loop is considered the primary or 'inner' loop of the servo system. MultiFlex controllers, which are position controllers, is used to close the secondary or 'outer' position loop of the servo system. For optimum position accuracy and repeatability it is recommended that the encoder for a velocity mode amplifier axis not be directly coupled to the motor. Ideally a linear scale (encoder) should be mounted on the external mechanics, as closely coupled as possible to the load or 'end effector'.



The most important step of tuning a servo that uses a velocity mode amplifier is to carefully follow the amplifier manufacturers setup instructions. Since the amplifier provides the **primary** servo control, if it is not setup correctly there is no possibility of attaining acceptable servo system performance.

There are significant differences when tuning servo systems that close the velocity loop external to the controller's position loop. The digital PID filter of the controller becomes a secondary component in the generation of the output signal that is applied to the velocity mode amplifier. The primary component that the controller will use to generate the servo command signal is the Feed Forward term.



Feed Forward defines a voltage level output from the controller, which in turn commands the velocity mode amplifier to rotate the motor at a specific velocity.

Prior to tuning velocity mode amplifier servo system the velocity feed forward term must be determined. The following example describes how to calculate and set velocity feed forward of a servo axis:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Setting the Velocity Feed Forward

The main component required to set the velocity feed forward of a servo axis is to determine the output level of the tachometer at a specific motor velocity. For this example, a typical tachometer specification would state:

Output Range	0.0 to +10V
Tach Output @ 1K RPM	1.0 volt

The specification describes a tachometer with an output range of 0 – 10V. The tachometer output ratio is 1.0V per 1,000 RPM's. The resolution of the linear scale encoder is 2000 encoder counts per inch, and the maximum velocity of the axis is 50 inches per second. Note: the servo amplifier may require scaling adjustments for the *RPM/Tachometer voltage output* ratio. The velocity feed forward is calculated as follows:

controller output = Velocity (encoder counts/sec) **X** Feed forward term (encoder counts/volt/sec.)

10 volts = 100,000 counts/sec. **X** Feed forward term (encoder counts * volt/sec.)

Feed forward = 10 volts / 100,000 counts per sec.

0.0001 = 10 volts / 100,000 counts per sec.

```
1VG0.0001                                   ;set velocity gain (velocity feed
                                               ;forward ) with MCCL command

// set velocity gain (velocity feed forward) using Motion Control API
function
//
    MCGetFilterConfig( hCtrlr, iAxis, &Filter );
    Filter.VelocityGain     = ( hCtlr, 1, 0.0001 );
    MCSetFilterConfig( hCtrlr, iAxis, &Filter );
```

Tuning the Servo

After setting the velocity feed forward (velocity gain) as shown above, open the Servo Tuning Utility. Configure the utility as follows:

1) From the Setup menu, select Servo Setup and define the trajectory parameters (velocity, acceleration, and deceleration) to match the application requirements.

2) From the Test Setup menu define a typical application move distance and duration. For this example, the move distance is 5000 encoder counts. The move duration is set to 600 milliseconds.

3) Set the Proportional (P), Integral (I), and Derivative (D) slide controls to 0%.

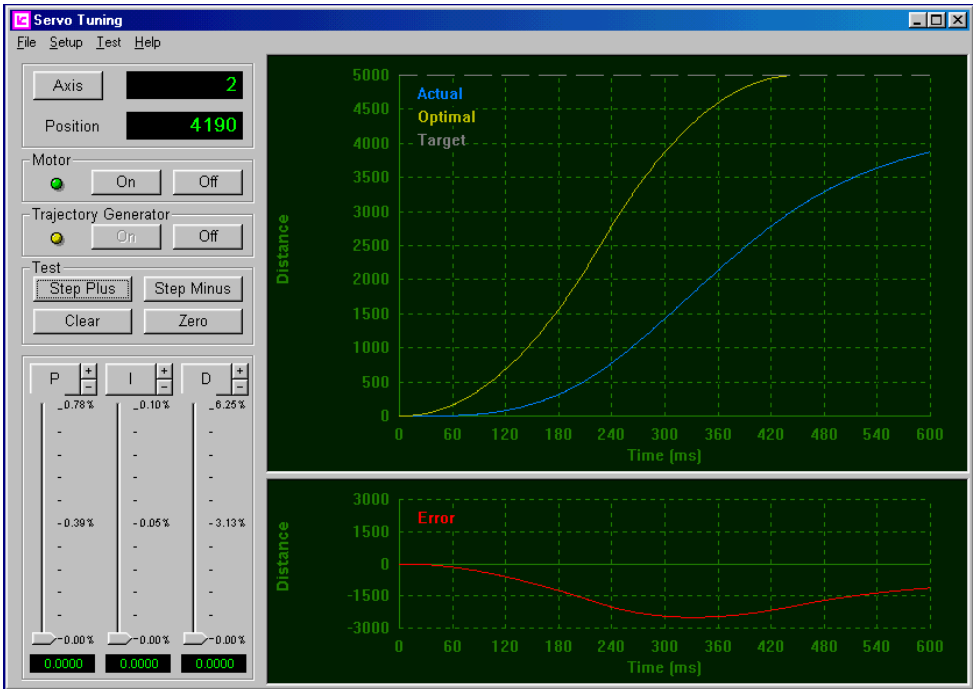
4) Turn on the Trajectory generator

5) Turn the motor on

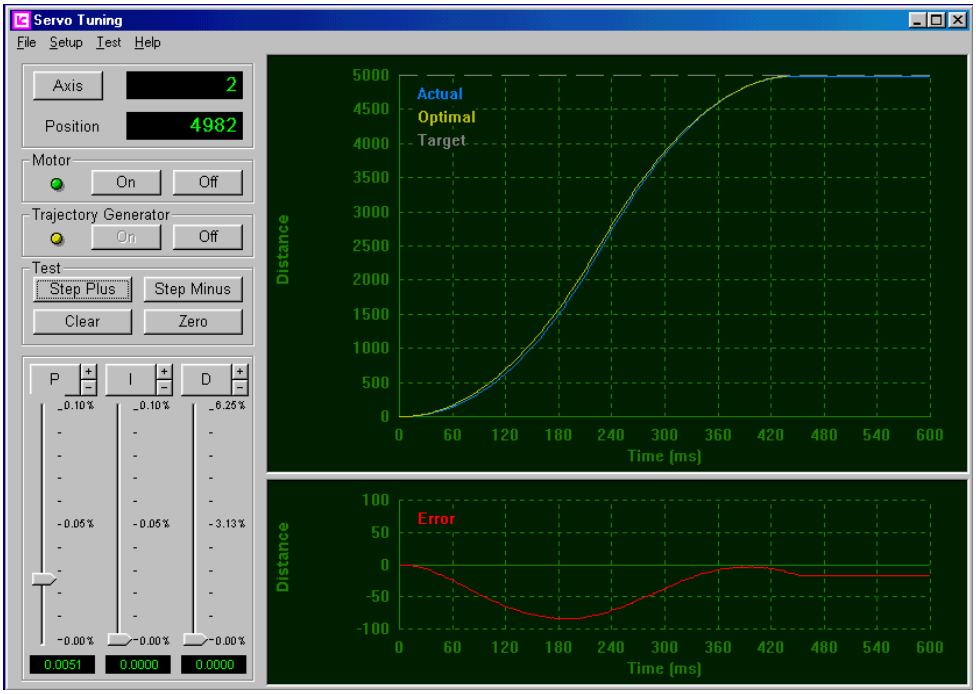
6) Press the Step Plus pushbutton

A response similar to the following graphic should be observed:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



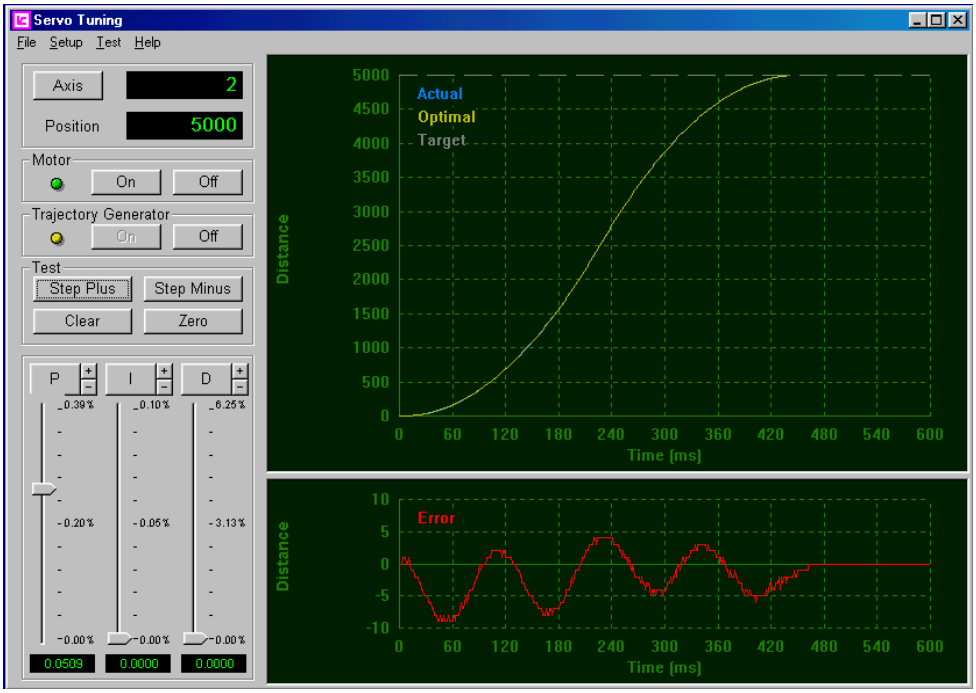
Increase the ‘P’ term 1-2 % at a time (it usually requires very little P for velocity mode amplifier systems) and repeat the move until the following error value is no longer reduced by increases in the P gain.



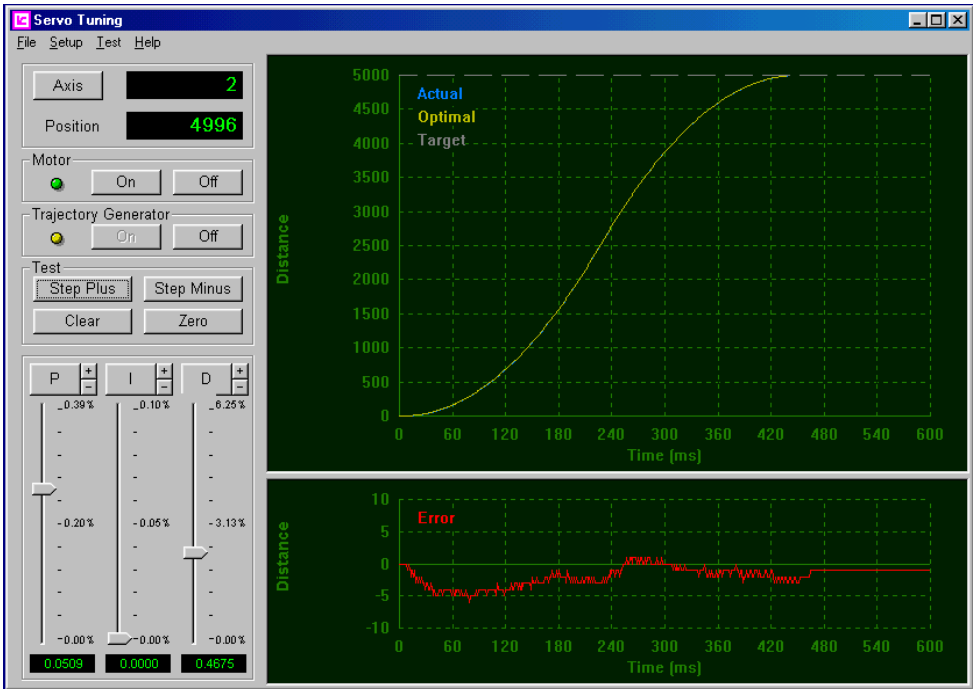
The current P gain is your baseline value. Note this value in case you need get back to this step.

Increase the ‘P’ term and repeat the move until the following error begins to oscillate around a following error of 0.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

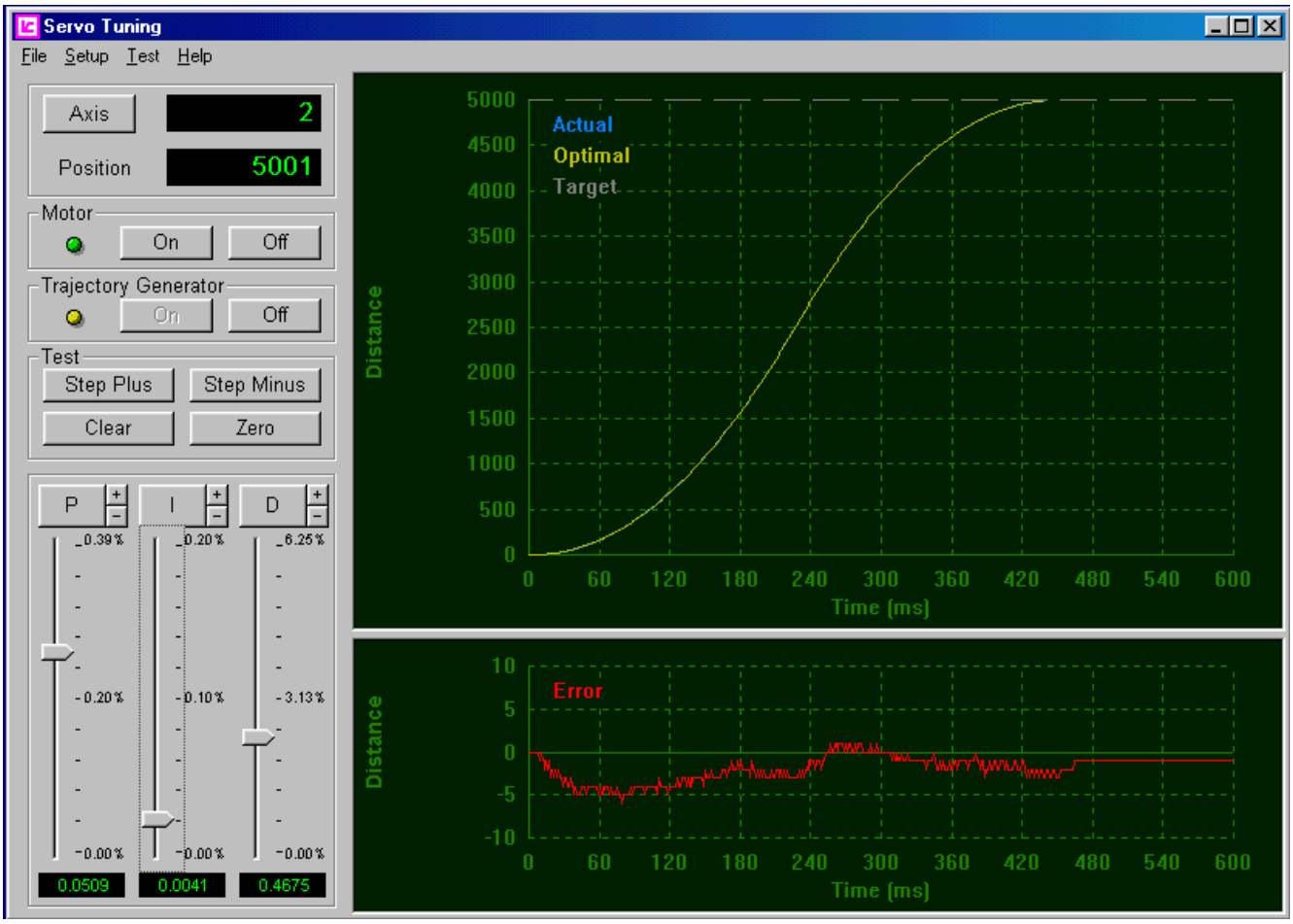


Reduce the 'P' term by 15% to 20%. Begin adding derivative gain until the oscillations have been dampened.



If the axis is not within a acceptable target range (typically +/- 1 encoder count) I gain is used to reduce the static error at the end of the move. Without issuing another move, very slowly begin to increase the I gain setting until the axis 'slews' to within +/- 1 encoder count. Now execute another move, if the axis oscillates at the end of the move the I gain is too high, reduce the I gain and repeat the move.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

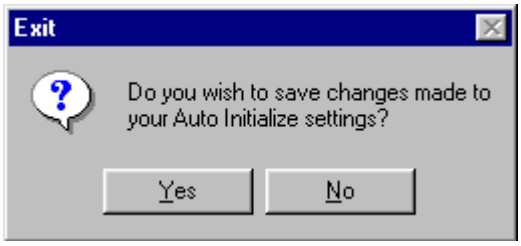


The motion controller provides the user with an option for the selecting the mode of Integral gain operation. For Velocity mode amplifiers it is recommended that the Integral gain option be set to **Zero**. In this mode of operation Integral gain is only calculated after the calculated trajectory has been completed. For additional information on Integral gain options please refer to page 156.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Saving the Tuning Parameters

When servo tuning is complete, closing the tuning utility will prompt this message about saving the Auto Initialize setting, selecting **Yes** will store all settings for all installed axes. Selecting **No** will cause all settings to be discarded.



Acceleration and Deceleration Feed Forward

For most applications velocity feed forward is sufficient for accurately positioning the axis. However for applications that require a very high rate of change, acceleration and deceleration gain must be used to reduce the following error at the beginning and end of a move.

Acceleration and deceleration feed forward values are calculated using a similar algorithm as used for velocity gain. The one difference is the velocity is expressed as encoder counts per second, while acceleration and deceleration are expressed as encoder counts per second per second.

Controller output = Accel./Decel. (encoder counts/sec/sec.) **X** Feed forward term (encoder counts * volt/sec./sec.)

Acceleration and deceleration feed forward values should be set prior to using the Servo Tuning Utility to set the proportional and integral gain. For additional information please contact PMC Tech support.

Systems with Electrical or Mechanical Deadband

Some servo systems may demonstrate significant dead band due to friction, sticktion, or insufficient amplifier drive power. This will typically be indicated when the output command to the servo is relatively high but the axis does not move.

Systems of this type can be very difficult to ‘tune’. To overcome the limitations of the system and get the axis moving, the proportional gain would need to be set very high. This will tend to make the system become unstable, causing the axis to ‘oscillate’ at the end of a move. The ***Output Deadband*** (***aODn***) command is used to compensate for the electrical and or mechanical dead band in a system by modifying the calculated output signal, allowing the module to simulate a ‘frictionless’ system. The deadband value will be added to a positive output and subtracted from a negative output.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Moving Servo Axes with Motor Mover

Once the servo is tuned, the axis is ready to perform velocity profile moves. PMC's Motor Mover program allows the user to execute absolute, relative, and cycle move sequences, monitor position and status of the axis. By selecting the Setup button the user can; set velocity parameters (maximum velocity, acceleration, and deceleration), set velocity profile (Trapezoidal, S curve, or Parabolic), and enable over travel limits.

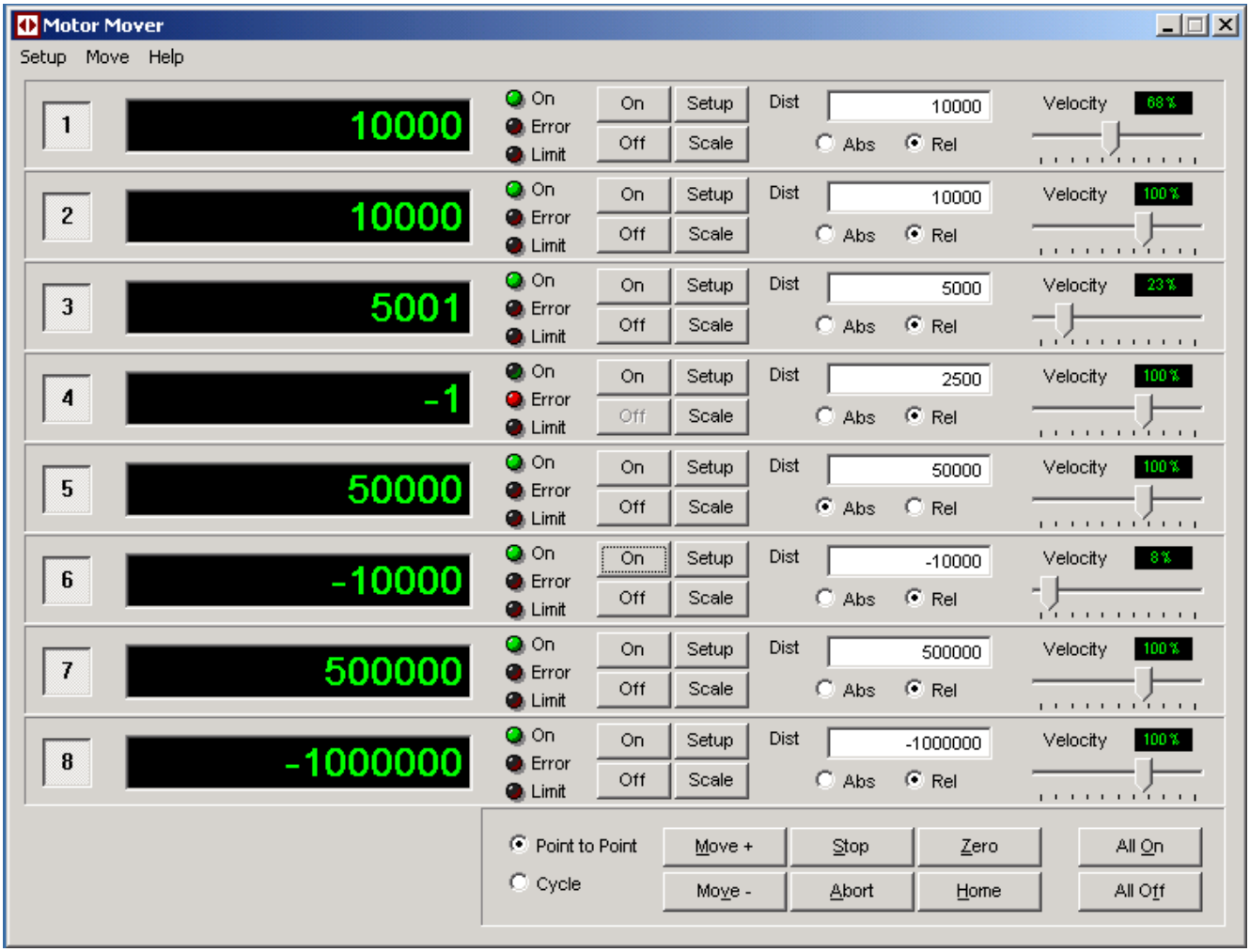


Figure 28. PMC's Motor Mover can be used to move as many as 8 axes simultaneously

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Stepper (pulse command) Axis Setup

The controller supports three modes for controlling stepper motors or pulse command servo's:

Open loop pulse command
Open loop pulse command with encoder feedback
Closed loop pulse command

Open Loop Pulse Command Motion

Controlling the position and velocity of a stepper motor with no component of the command output adjusted based on encoder feedback from the electro mechanical system is called open loop control. The steps required to implement open loop pulse command motion are:

- Select the Velocity Profile type
- Define Trajectory parameters (max. velocity, acceleration, deceleration, min. velocity)
- Select the Pulse Rate range
- Configure axis inputs (over travel limits and driver fault)

To configure a Pulse Command axis for motion use the Stepper Axis Setup Dialog (select Setup from Motor Mover).

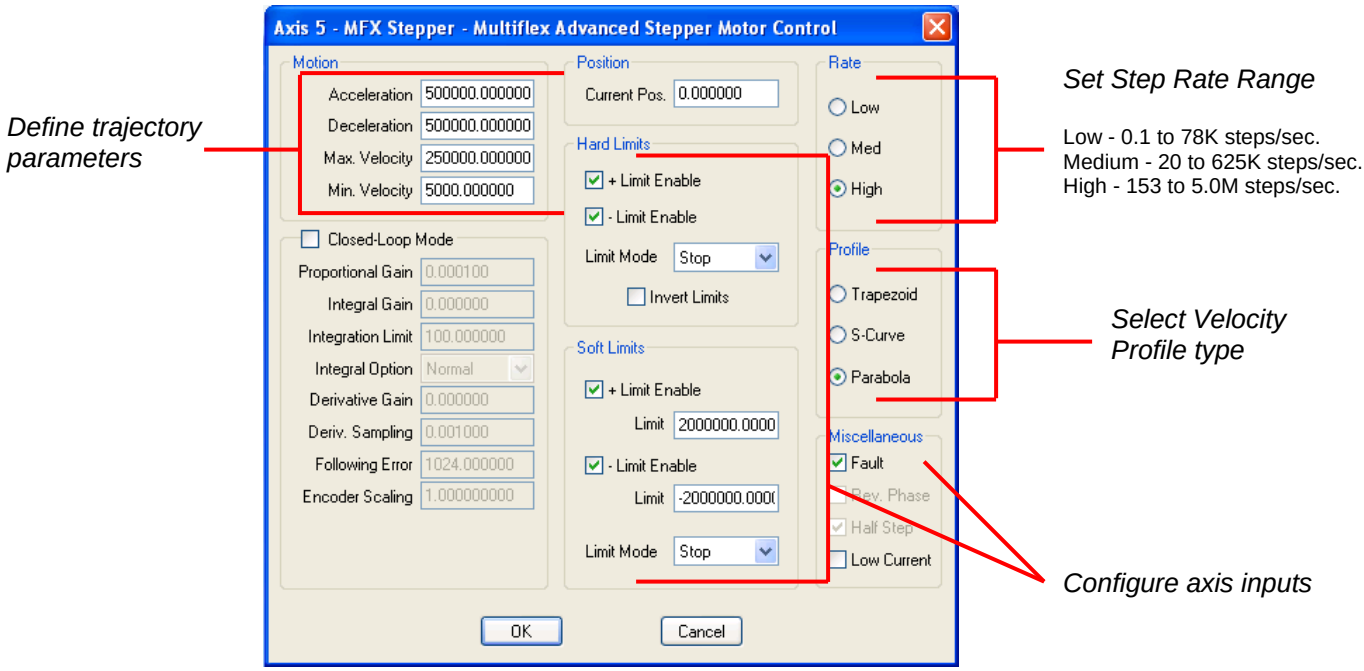


Figure 29. Stepper axis Setup Dialog

The Minimum Velocity of a stepper axis must be set to a non zero value. The default value is 1,000 steps per second. The recommended setting of the

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

minimum velocity is from 1% to 10% of the maximum velocity.

When the user commands a move the controller counts each pulse that is issued to the stepper motor driver. When the position of an axis is queried (by issuing the function ***MCGetPosition ( )*** or MCCL **T**ell **P**osition (***a*TP**) command), the number of pulses issued to the stepper driver is reported. Since there is no position (or velocity) feedback there is no need to 'tune' the axis. However, the axis module must be configured (Trajectory parameters, Velocity Profile, Limits etc...).



Stepper drivers typically use the Direction output from the stepper controller signals to determine the observed direction of motion. If the observed direction of motion is not correct (moving positive causes counter clockwise instead of clockwise rotation) set axis scaling to -1.0.

Open Loop Pulse Command Motion with Position Verification Encoder

For some open loop stepper applications it is required that an encoder be incorporated to allow the user to compensate for 'lost steps'. For these type of applications follow the steps described in the previous section for configuring **Open Loop Pulse Command Motion**. After the axis has been configured and basic motion has been verified, issue the Motion Control API function ***MCGetAuxEncPosEX( )*** to read the position of the auxiliary encoder. For auxiliary encoder connection information please refer to the **Connectors, I/O and Schematics** chapter. For additional information please refer to the **Application Solutions** chapter.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Closed Loop Steppers

The advancements in stepper motor/micro stepping driver technology have allowed many machine builders to maintain ‘servo like’ performance while reducing costs by moving to closed loop stepper systems. While closed loop steppers will still be susceptible to ‘stalling’, they are not plagued with the familiar open loop stepper system problem of loosing steps due to friction (mechanical binding) or system resonance.

For high accuracy stepper applications, the controller supports closed loop control of stepper motors using quadrature incremental encoders for position feedback. The stepper axis will be controlled as if it is a closed loop servo, the quantity and frequency of step pulses applied to the stepper driver is based on the trajectory parameters of the move and the position error of the axis. Prior to attempting to operate a stepper motor in closed loop mode the basic system components (motor, driver, wiring, and controller) should be verified by moving in open loop mode. For information on operating an open loop stepper please refer to the **Stepper Basics** and **Moving Motors with Motor Mover** sections in this chapter. If the stepper motor does not operate as expected please refer the **Troubleshooting** chapter.



While executing closed loop stepper motion, when the target position equals the current encoder position, the step pulse generator (PID filter) will be turned off within 1 micro second.

Unlike a closed loop servo, if the final position of the stepper encoder is beyond the target position of the move **the motor will not be commanded to move back to the target**.

Closed Loop Stepper Setup
There are four steps required to configure a stepper to operate in closed loop :

1) Connect and verify operation of the encoder
2) Define the Encoder / Steps ratio
3) Set the trajectory parameters
4) Tune the axis

Connect and verify the encoder
Connect the stepper motor's encoder to the controller per the connector pinouts described in chapters 5 and 10 of this manual.

To verify the operation of the encoder open the Motor Mover program (Start\Programs\Motion Control\Motion Integrator\Motor Mover). From the Setup dialog select Closed Loop Mode and OK.:

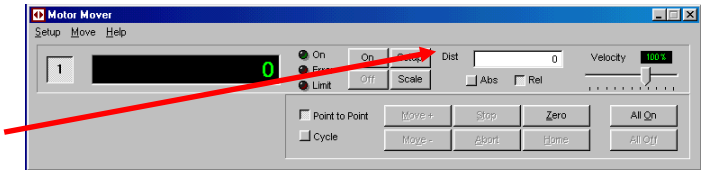


Figure 30. Select Setup to open the dialog

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

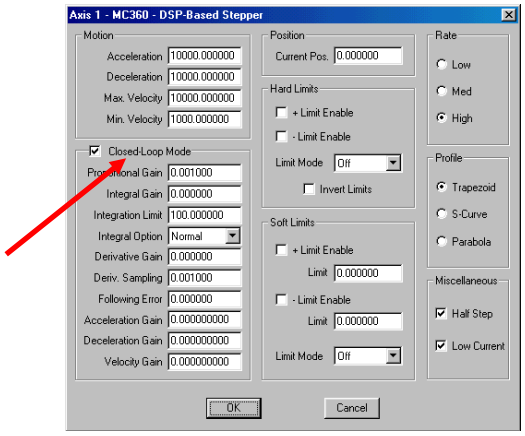


Figure 31. From the Stepper Setup dialog select the Closed Loop Mode check box

After closed loop mode has been enabled the **Motor Mover** position readouts will display the position of the encoder (versus displaying the 'pulse count'). Rotate the motor / encoder shaft back and forth and verify that the position display changes accordingly.

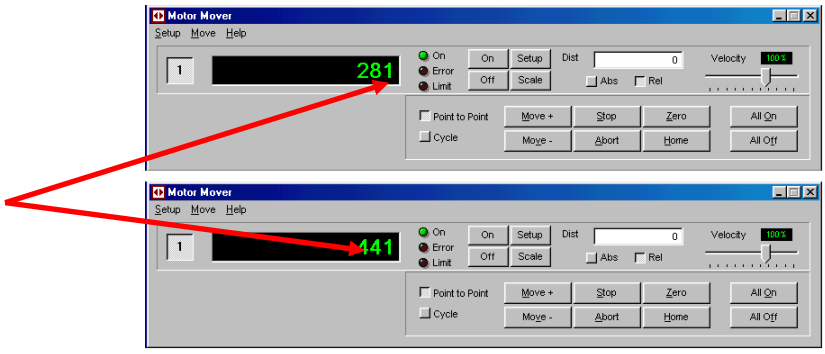


Figure 32. In closed loop mode the Motor Mover position readout displays encoder position

After switching a stepper axis into or out of closed loop mode the axis must be disabled and then enabled. From a PMC program (Motor Mover, Cwdemo, Servo Tuning, etc...) select the Motor Off and then the Motor On buttons). From a user application issue the ***MCEnableAxis ( )*** function with ***state*** = **False** and then ***state*** = **True**.

Define the motor steps per rotation / encoder counts per rotation ratio

When a stepper axis is operating in closed loop mode, move commands are issued in units of encoder counts. The **EncoderScaling** member of the ***MCFILTEREX*** data structure is used to configure the controller for converting encoder units to step pulses. The value is calculated by dividing motor steps per rotation by encoder counts per rotation. For example, if there 2000 encoder counts per rotation (500 line encoder) and the stepper motor has 51,200 steps per rotation, the Encoder Scaling value would be

EncoderScaling = motor steps per rotation / encoder counts per rotation.
EncoderScaling = 51,200 / 2000
EncoderScaling = 25.6

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The Encoder Scale can also be defined from the **Stepper Setup** dialog of the **Servo Tuning** or **Motor Mover** programs.

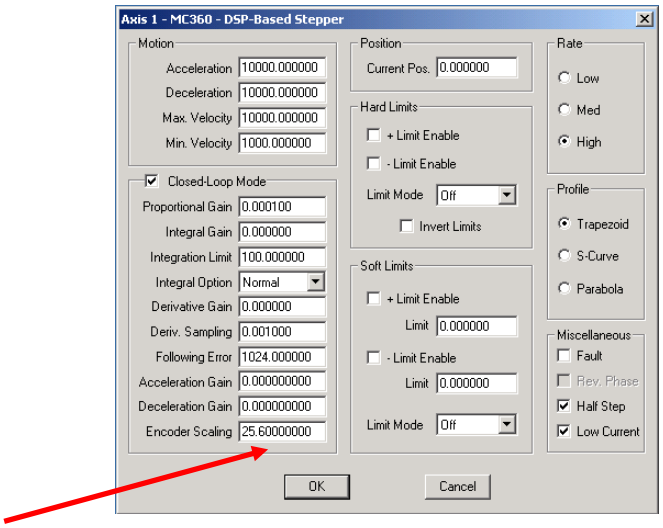


Figure 33. Enter the closed loop steps / encoder scale

Set the trajectory parameters

As with an open loop stepper, the trajectory parameters (maximum velocity, acceleration, deceleration, and minimum velocity) must be set prior to commanding motion. These values can be set using the ***MCMOTION*** data structure or can be entered from the **Stepper Setup** dialog of **Servo Tuning** or **Motor Mover**.

Closed loop stepper trajectory parameters (and move distances) are specified in **encoder units**, not motor step units.

Tune the axis

When a stepper axis is configured for closed loop operation the default proportional gain is set to 0.0001, which should be sufficient to move the axis **near** the specified target. Further adjustments of the proportional and integral gain allow the controller to:

Minimize the following error while moving
Eliminate slow speed slewing of the axis near the end of the move
Settle within 1 encoder count of the target

Use the PMC Servo Tuning program (\Start\Programs\Motion Control\Motion Integrator\Servo Tuning) to tune the closed loop stepper.

Step 1 - Enter a typical move distance (in encoder counts) and move duration (in milliseconds) using the **Test Setup** dialog (Setup\Test Setup).

Step 2 - Verify that the Trajectory Generator is on (yellow LED)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Step 3 - Set the Proportional gain Slide Control Scale 0.20% (**Press P+ zoom button**)

Step 4 - Verify that the Proportional gain is set to 0.0001, Integral and Derivative gain = 0. Generally Derivative gain and Integral gain are not required to tune a closed loop stepper.

Step 5 - From the **Servo Setup** dialog verify that **Closed Loop Mode** is enabled and that the **Encoder Scaling** has been set

Step 6 - Toggle the **Motor Off** and **Motor On** buttons to initialize the closed loop position registers

Step 7 - Start the move with the **Move +** or **Move -** buttons

Step 8 - Observe the plot of following error during the move

Step 9 - Increase the proportional gain and repeat the move until the point of diminishing returns is reached (the following error no longer decreases). Further increases of the proportional gain will tend to cause the motor to emit a grinding noise or stall during a commanded move.

Step 10 - If the axis moves slowly near the end of the move and/or stops a few counts short of the target the Minimum Velocity is probably set too low.

Step 11 - Save the closed loop stepper settings by selecting **Save All Axes Settings** from the Servo Tuning **File** menu. This operation will copy all settings into the mcapi.ini file so that any windows application program can load axis settings upon opening.



For additional information on using the Servo Tuning program please refer to:

The **Tuning the Servo** section of the **Motion Control** chapter
The Servo Tuning program on-line help



To disable closed loop stepper operation, issue the ***MCSetModuleInputMode*** function with *Mode* = **MC_IM_OPEN_LOOP** or deselect the closed loop check box in the Servo Tuning Servo Setup dialog..

Reverse Phasing of a closed loop stepper

If the closed loop stepper is reverse phased, issuing a move command will cause the motor to 'take off' in the wrong direction at full torque / speed. Once the position error exceeds the value entered for the allowable following error (default = 1024) a motor error will occur and the axis will stop. To change the phasing either:

Issuing the ***MCSetServoOutputPhase ( )*** function with *Phase* = **MC_PHASE_REVERSE**
Selecting the **Reverse Phase** option in the Servo Tuning Servo Setup dialog
Swap the encoder phase A and B connections to the controller.

Closed loop stepper example

Axis 5 is a 51,200 micro steps per rotation stepper motor. A 2,000 count (500 line) incremental encoder is coupled to the stepper motor shaft. The required maximum step rate for this application is 896,000 steps

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

per second (1050 RPM), which requires the axis to be configured for High Speed step range. After verifying the operation of the closed loop stepper from within the Servo Tuning program, save the configuration with the File menu **Save All Axis Settings** option. From a users application program to load the closed loop configuration call the ***MCDLG_RestoreAxis*** function from the PMC Common Motion Dialog Library. To load the closed loop axis configuration from a PMC application program (Servo Tuning or Motor Mover) select **Auto Initialize** from the **File** menu.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Moving Stepper Axes with Motor Mover

After configuring the stepper setup parameters it is ready to execute motion. The Motor Mover program allows the user to execute absolute, relative, and cycle move sequences, monitor position and status of the axis. By selecting the **Setup** button the user can; set velocity parameters (maximum velocity, acceleration, and deceleration), set velocity profile (Trapezoidal, S curve, or Parabolic), and enable motion limits.

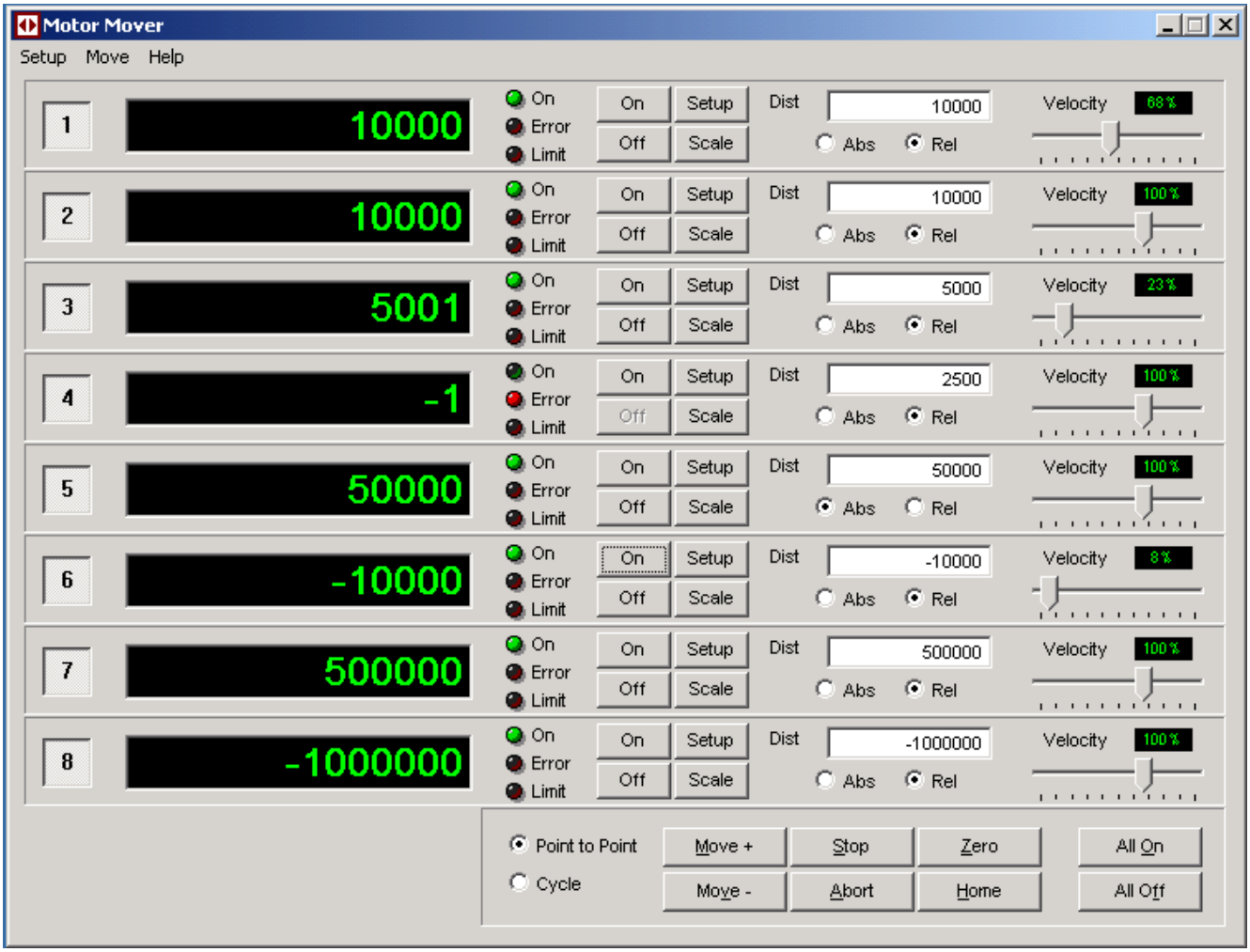


Figure 34. PMC's Motor Mover can be used to command motion for as many as 8 axes simultaneously

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

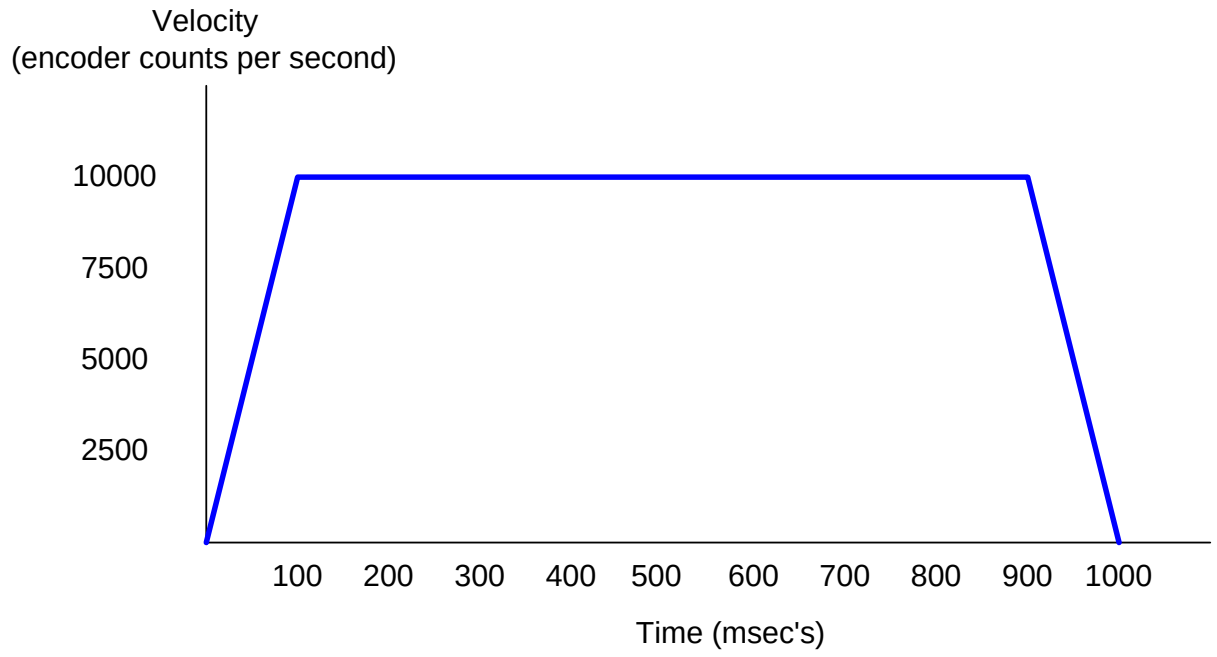
Defining the Characteristics of a Move

Prior to executing any move, the user should define the parameters of the move. The components that make up a move are:

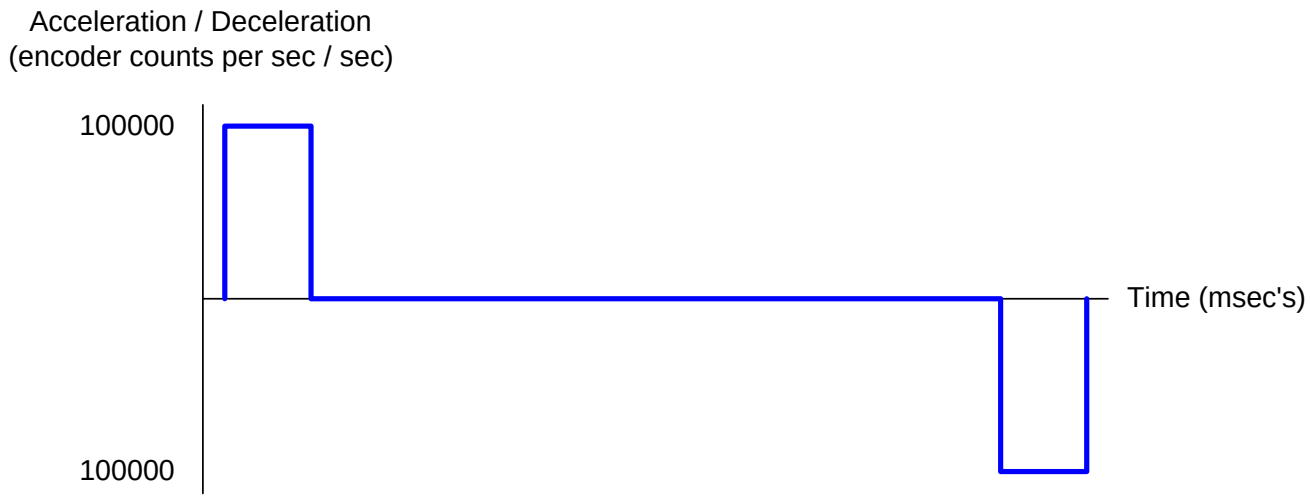
```
// Set axis 1 maximum velocity
// Set axis 1 acceleration
// Set axis 1 deceleration
// Set profile as Trapezoidal
// Set Position mode
// Set target (10000), begin move


MCSetVelocity( hCtlr, 1, 10000.0 );
MCSetAcceleration( hCtlr, 1, 100000.0 );
MCSetDeceleration( hCtlr, 1, 100000.0 );
MCSetProfile( hCtlr, 1, MC_PROF_TRAPEZOID );
MCSetOperatingMode( hCtlr, 1, 0, MC_MODE_POSITION );
MCMoveRelative( hCtlr, 1, 100000.0 );
```

The parameters defined in the program example above specify a move to position 100,000. During the move the velocity will not exceed 10,000 encoder counts per second. A trapezoidal velocity profile will be calculated by the motion controller The rate of change (acceleration and deceleration) will be 100,000 encoder counts per second/per second, there by reaching the maximum velocity (10,000 counts per second) in 100 msec's. The resulting velocity and acceleration profiles follow:



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



The default units for expressing Trajectory Parameters are:

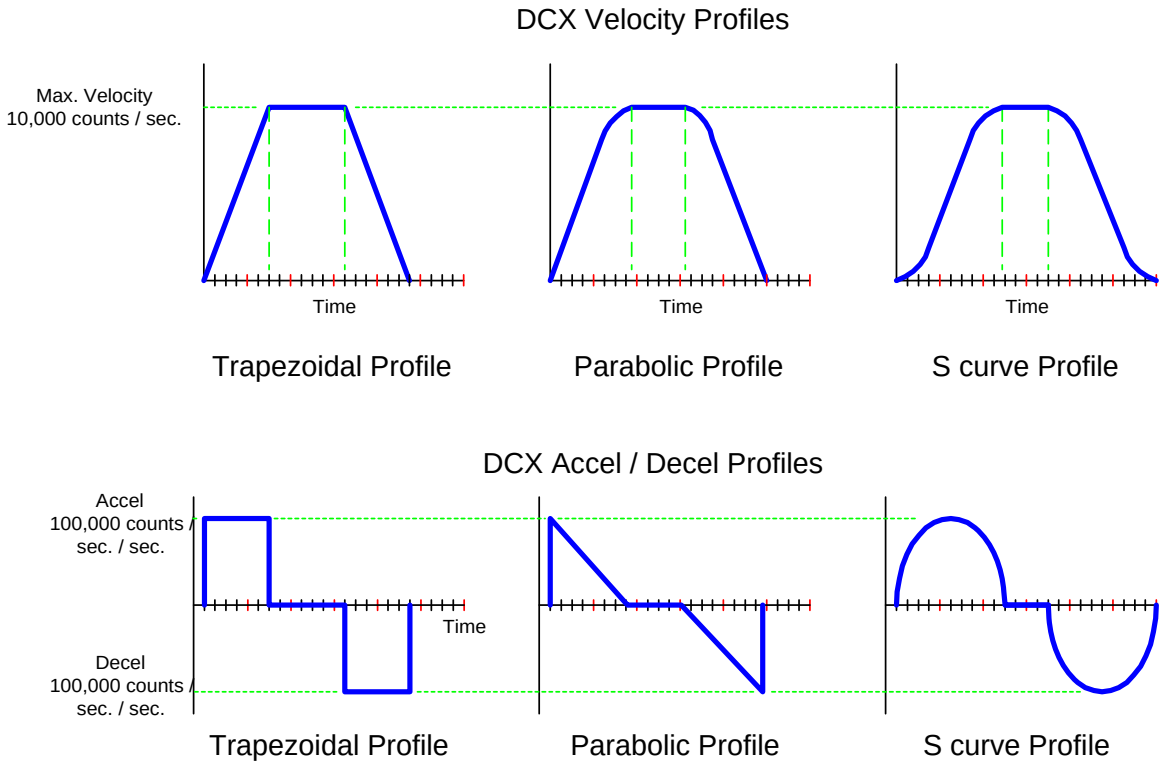
Velocity - encoder counts / second
Acceleration - encoder counts / second / second
Deceleration - encoder counts / second / second

If user unit parameters **Scale** and / or **Rate** are set to values other than 1 then the Velocity, Acceleration, and Deceleration units will change as well.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Velocity Profiles

The user can select one of three different velocity profiles that the controller will then use to calculate the trajectory of a move.



Trapezoidal Profile – (servo & steppers) MCSetProfile(hCtlr, 1, MC_PROF_TRAPEZOID);
Shortest time to target when using the same trajectory parameters
Profile most likely to result 'jerk' and/or oscillation
Supports 'on the fly' target changes

Parabolic Profile – (stepper only) MCSetProfile(hCtlr, 1, MC_PROF_PARABOLIC);
Slow 'roll off' minimizes lost steps at high velocity
Initial linear rate of change eliminates 'cogging'
On the fly changes will cause the axis to first decelerate to a stop

S curve Profile – (servo only) MCSetProfile(hCtlr, 1, MC_PROF_SCURVE);
'True sine' rate of change effectively eliminates 'jerk' and/or oscillation
Longest time to target when using the same trajectory parameters
On the fly changes will cause the axis to first decelerate to a stop

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Point to Point Motion

To perform point to point motion of a servo or stepper motor::

```
// Enable the axis
// Enable Position mode
// Define the velocity profile (trapezoidal, S curve, or parabolic)
// define maximum velocity
// define acceleration
// define deceleration
// execute the move

MCEnableAxis( hCtlr, 1, TRUE );
MCSetOperatingMode( hCtlr, 1, 0, MC_MODE_POSITION );
MCSetProfile( hCtlr, 1, MC_PROF_TRAPEZOIDAL );
MCSetVelocity( hCtlr, 1, 10000.0 );
MCSetAcceleration( hCtlr, 1, 25000.0 );
MCSetDeceleration( hCtlr, 1, 50000.0 );
MCMoveRelative( hCtlr, 1, 122.5 );
```

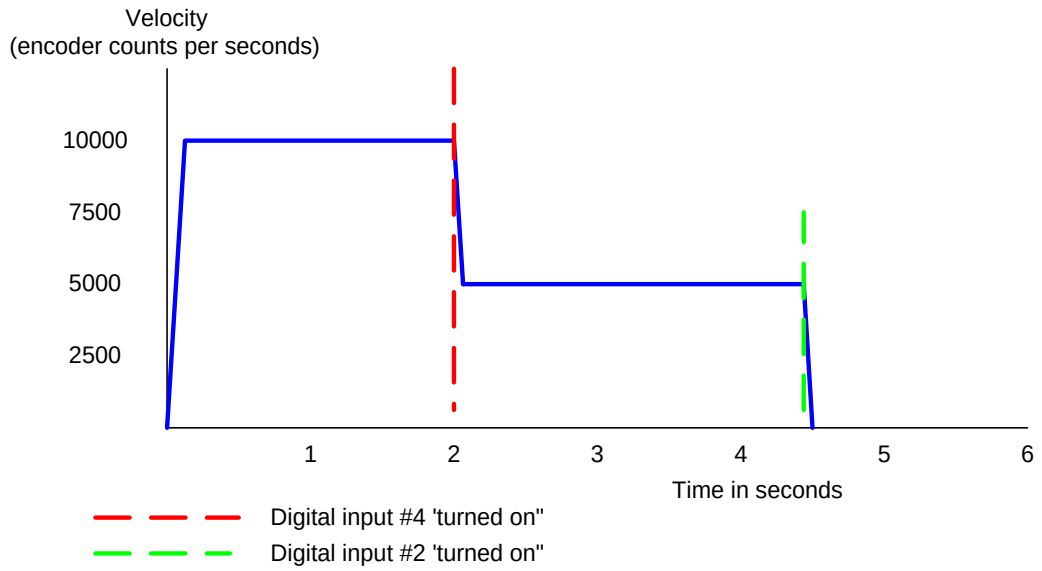

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Constant Velocity Motion

To move a servo or stepper at a continuous velocity until commanded to stop:

```
// Enable the axis
// Enable Velocity mode
// Define the velocity profile (trapezoidal, S curve, or parabolic)
// define maximum velocity
// define acceleration
// define deceleration
// define the direction (positive or negative) of the move
// begin motion of axis 1
// wait for digital I/O #4 to be true
// reduce velocity
// wait for digital I/O #2 to be true
// stop the motion of axis 1


MCEnableAxis( hCtlr, 1, TRUE );
MCSetOperatingMode( hCtlr, 1, 0, MC_MODE_VELOCITY );
MCSetProfile( hCtlr, 1, MC_PROF_TRAPEZOIDAL );
MCSetVelocity( hCtlr, 1, 10000.0 );
MCSetAcceleration( hCtlr, 1, 100000.0 );
MCSetDeceleration( hCtlr, 1, 100000.0 );
MCSetDirection( hCtlr, 1, POSITIVE );
MCGo( hCtlr, 3 );
MCWait For DigitalIO( hCtlr, 4, TRUE );
MCSetVelocity( hCtlr, 1, 5000.0 );
MCWait For DigitalIO( hCtlr, 2, TRUE );
MCStop( hCtlr, 1 );
```



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Contour Motion (arcs and lines)

The controller supports Linear Interpolated motion with any combination of two to eight axes and Circular Contouring on as many as four groups of two axes. Executing a multi axis contour move requires:

Turn the axes on
Define the axes in the contour group and the controlling axis
Define the trajectory (Vector Velocity, Vector Acceleration and Vector Deceleration)
Define the type of contour move (Linear, Circular, user defined) and the move targets
Loading the Contour Buffer for Continuous Path Contouring

Define the contour group

The ***MCSetOperatingMode( )*** command is used to define the axes in a contour group. Issue this command to each of the axes in the contour group. The parameter *wMaster* should be set to the lowest axis number that will be moving on the contour. This axis will then be defined as the 'controlling' axis for the contour group. The following example configures axis 1, 2, and 3 for contour motion with axis #1 defined as the controlling axis.

```
MCSetOperatingMode( hCtlr, 1, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 2, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 3, 1, MC_MODE_CONTOUR );
```

Define the trajectory parameters

The ***MCGetContourConfig( )***, ***MCSetContourConfig( )***, and **MCContour** data structure are used to define the trajectory parameters of a contour motion. The default units of the vector velocity are encoder counts or steps per second. The default units of vector acceleration and vector deceleration are encoder counts or steps per second per second. The default units of velocity override is a percentage of the setting for vector velocity.

```
//  Motion settings (GetDlgItemDouble() is a helper function defined
//  elsewhere)
//
case IDOK:
    MCGetContourConfig( hCtrlr, iAxis, &Contour );
    Contour.Vector.Accel     = GetDlgItemDouble( hDlg, IDC_TXT_ACCEL );
    Contour.VectorDecel      = GetDlgItemDouble( hDlg, IDC_TXT_DECEL );
    Contour.VectorVelocity     = GetDlgItemDouble( hDlg, IDC_TXT_VELOCITY );
    Contour.VelocityOverride     = GetDlgItemDouble( hDlg, IDC_TXT_MAX_TORQUE);
    MCSetContourConfig( hCtrlr, iAxis, &Contour );
```

Define the type of contour move

The *nMode* parameter of the ***MCBlockBegin( )*** function is used to define the type of contour move to be executed. The following types of contour motion are supported:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 2. Contour Mode Parameters

nMode parameter	Contour move type	Description
MC_BLOCK_CONTR_USER	User defined, 1 to 8 axes	Specifies that this block is a user defined contour path motion. *lNum* should be set to the controlling axis number.
MC_BLOCK_CONTR_LIN	Linear interpolated move, 1 to 8 axes	Specifies that this block is a linear contour path motion. *lNum* should be set to the controlling axis number.
MC_BLOCK_CONTR_CW	Clockwise arc, 2 axes	Specifies that this block is a clockwise arc contour path motion. *lNum* should be set to the controlling axis number.
MC_BLOCK_CONTR_CCW	Counter Clockwise arc, 2 axes	Specifies that this block is a counter-clockwise arc contour path motion. *lNum* should be set to the controlling axis number.

Examples of a linear move and a clockwise arc follow:

```
//  Linear move
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_LIN, 1 );
   MCMoveAbsolute( hCtlr, 1, 10000.0 );
   MCMoveAbsolute( hCtlr, 2, 20000.0 );
   MCMoveRelative( hCtlr, 3, -5000.0 );
MCBlockEnd( hCtlr, NULL );


//  Clockwise arc move
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_CW, 1 );
   MCArcCenter( hCtlr, 1, MC_CENTER_ABS, 20000.0 );
   MCArcCenter( hCtlr, 2, MC_CENTER_ABS, 0.0 );
   MCMoveAbsolute( hCtlr, 1, 40000.0 );
   MCMoveAbsolute( hCtlr, 2, 0.0 );
MCBlockEnd( hCtlr, NULL )
```

Loading the Contour Buffer for Continuous Path Contouring

The controller uses a wrap around buffer known as the Contour Buffer to support Continuous Path Contouring. When a single contour move is executed, the axes will begin moving towards the targets at the user specified vector velocity and then will decelerate (at the specified vector velocity) and stop at the target. If multiple contour move commands are issued, the contour buffer allows moves to smoothly transition from one to the other. The vector motion will not decelerate and stop until the contour buffer is empty or an error condition (max following error exceeded, limit sensor ‘trip’, etc...) occurs.

The wrap around Contour Buffer can be queued with as many as 256 linear motions or 128 arc motions (an arc move takes up twice as much buffer space). The ***MCGetContouringCount( )*** command will report how many contour moves have been executed since the axes were last configured for contour motion with ***MCSetOperatingMode( )***. The contouring count is stored as a 32 bit value, which means that 2,147,483,647 contour moves can be executed before the contour count will ‘roll over’.

To delay starting contour motion until the contour buffer has been fully loaded use the ***MCEnableSynch()*** command. This command should be issued to the controlling axis **before** issuing any contour moves. Moves issued after the ***MCEnableSynch( )*** command will be queued into the contour buffer. To begin executing the moves in the buffer, issue the ***MCGoEx( )*** command to the controlling axis . To return to

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

normal operation (immediate execution of contour move commands), issue ***MCEnableSynch( )*** to the controlling axis with the state = FALSE.

Multi Axis Linear Interpolated moves

An example of three linear interpolated moves is shown below. Once the first compound move command is issued, motion of the three axes will start immediately (at the specified vector velocity). The other two compound commands are queued into the contouring buffer. As long as additional contour moves reside in the contour buffer continuous path contour motion continue. In this example, smooth vector motion will continue (without stopping) until all three linear moves have been completed (the contour buffer has been emptied). At this time the axes will simultaneously decelerate and stop.

```
MCSetOperatingMode( hCtlr, 1, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 2, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 3, 1, MC_MODE_CONTOUR );


//  Motion settings (GetDlgItemDouble() is a helper function defined
//  elsewhere)
//
case IDOK:
   MCGetContourConfig( hCtrlr, iAxis, &Contour );
   Contour.Vector.Accel     = GetDlgItemDouble( hDlg, IDC_TXT_ACCEL );
   Contour.VectorDecel     = GetDlgItemDouble( hDlg, IDC_TXT_DECEL );
   Contour.VectorVelocity     = GetDlgItemDouble( hDlg, IDC_TXT_VELOCITY
);
   Contour.VelocityOverride     = GetDlgItemDouble( hDlg,
IDC_TXT_MAX_TORQUE);
   MCSetContourConfig( hCtrlr, iAxis, &Contour );


//  Linear move #1
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_LIN, 1 );
   MCMoveAbsolute( hCtlr, 1, 85000.0 );
   MCMoveRelative( hCtlr, 2, 12000.0 );
   MCMoveAbsolute( hCtlr, 3, -33000.0 );
MCBlockEnd( hCtlr, NULL );

//  Linear move #2
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_LIN, 1 );
   MCMoveAbsolute( hCtlr, 1, 0.0 );
   MCMoveAbsolute( hCtlr, 2, 0.0 );
   MCMoveAbsolute( hCtlr, 3, 0.0 );
MCBlockEnd( hCtlr, NULL );

//  Linear move #3
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_LIN, 1 );
   MCMoveAbsolute( hCtlr, 1, 5000.0 );
   MCMoveRelative( hCtlr, 2, 23000.0 );
   MCMoveAbsolute( hCtlr, 3, -16000.0 );
MCBlockEnd( hCtlr, NULL );
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Arc Motion

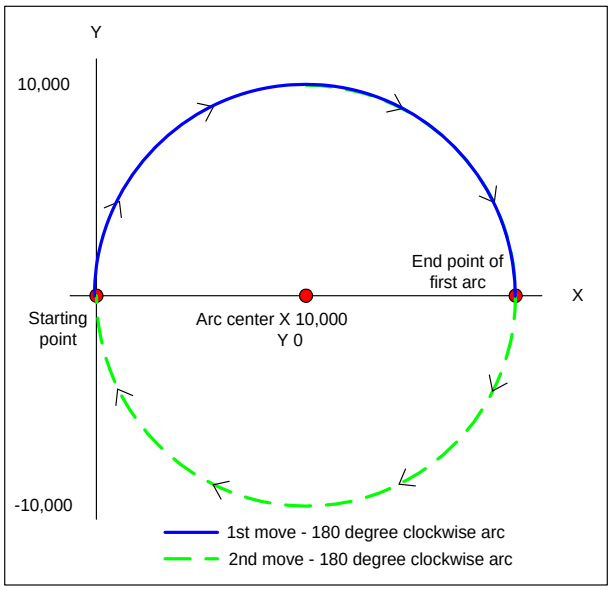
The controller supports specifying an arc motion in two axes in any of three different ways:

Specify center and end point
Specify radius and end point (not supported by the Motion Control API)
Specify center and ending angle (not supported by Motion Control API)

When the first arc motion is issued, motion of the two axes will start immediately (at the specified vector velocity). Additional contour motions will be queued into the contouring buffer. As long as additional contour moves reside in the contour buffer continuous path contour motion will occur. In this example, smooth vector motion will continue (without stopping) until all both arc motions have been completed (the contour buffer has been emptied). At this time the axes will simultaneously decelerate and stop.

Arc motions by specifying the center point and end point

The ***MCArcCnter( )*** command is used to specify the center position of the arc. This command also defines which two axes will perform the arc motion. The ***MCMoveAbsolute( )*** or ***MCMoveRelative( )*** commands are used to specify the end point of the arc. A spiral motion will be performed if the distance from the starting point to center point is different than the distance from the center point to end point. An example of two arc motions is shown below:




```
MCSetOperatingMode( hCtlr, 1, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 2, 1, MC_MODE_CONTOUR );


//  Motion settings (GetDlgItemDouble() is a helper function defined
//  elsewhere)
//
case IDOK:
   MCGetContourConfig( hCtrlr, iAxis, &Contour );
   Contour.Vector.Accel     = GetDlgItemDouble( hDlg, IDC_TXT_ACCEL );
   Contour.VectorDecel     = GetDlgItemDouble( hDlg, IDC_TXT_DECEL );
   Contour.VectorVelocity     = GetDlgItemDouble( hDlg, IDC_TXT_VELOCITY );
   Contour.VelocityOverride     = GetDlgItemDouble( hDlg, IDC_TXT_MAX_TORQUE);
   MCSetContourConfig( hCtrlr, iAxis, &Contour );
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
//  Clockwise arc move #1
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_CW, 1 );
   MCArcCenter( hCtlr, 1, MC_CENTER_ABS, 10000.0 );
   MCArcCenter( hCtlr, 2, MC_CENTER_ABS, 0.0 );
   MCMoveAbsolute( hCtlr, 1, 20000.0 );
   MCMoveAbsolute( hCtlr, 2, 0.0 );
MCBlockEnd( hCtlr, NULL );



//  Clockwise arc move #2
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_CCW, 1 );
   MCArcCenter( hCtlr, 1, MC_CENTER_REL, -10000.0 );
   MCArcCenter( hCtlr, 2, MC_CENTER_REL, 0.0 );
   MCMoveRelative( hCtlr, 1, -20000.0 );
   MCMoveRelative( hCtlr, 2, 0.0 );
MCBlockEnd( hCtlr, NULL );
```

Arc motions by specifying the radius and end point

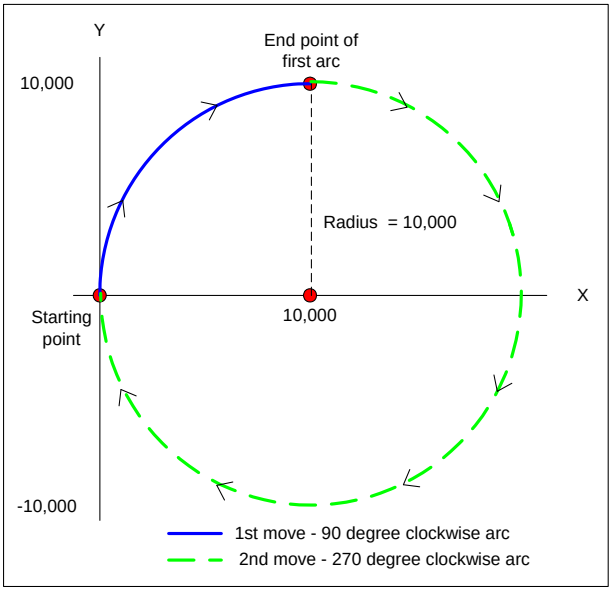
The ***MCArcRadius( )*** function is used to execute an arc move by specifying the radius and end point of an arc. The *Axis* parameter should equal the controlling axis for the contour move. The parameter *Radius* should equal the radius of the arc. If the arc is greater than 180 degrees, the parameter *Radius* must be expressed as a negative number. The ***MCMoveAbsolute( )*** or ***MCMoveRelative( )*** commands are used to specify the end point of the arc. An example of two arc motions is shown below:

```
MCSetOperatingMode( hCtlr, 1, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 2, 1, MC_MODE_CONTOUR );


//  90 degree Clockwise arc move #1
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_CW, 1 );
   MCArcRadius( hCtlr, 1, 10000.0 );
   MCMoveRelative( hCtlr, 1, 10000.0 );
   MCMoveRelative( hCtlr, 2, 10000.0 );
MCBlockEnd( hCtlr, NULL );


//  270 degree Clockwise arc move #2
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_CW, 1 );
   MCArcRadius( hCtlr, 1, -10000.0 );
   MCMoveRelative( hCtlr, 1, -10000.0 );
   MCMoveRelative( hCtlr, 2, -10000.0 );
MCBlockEnd( hCtlr, NULL );
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Arc motions by specifying the center point and ending angle

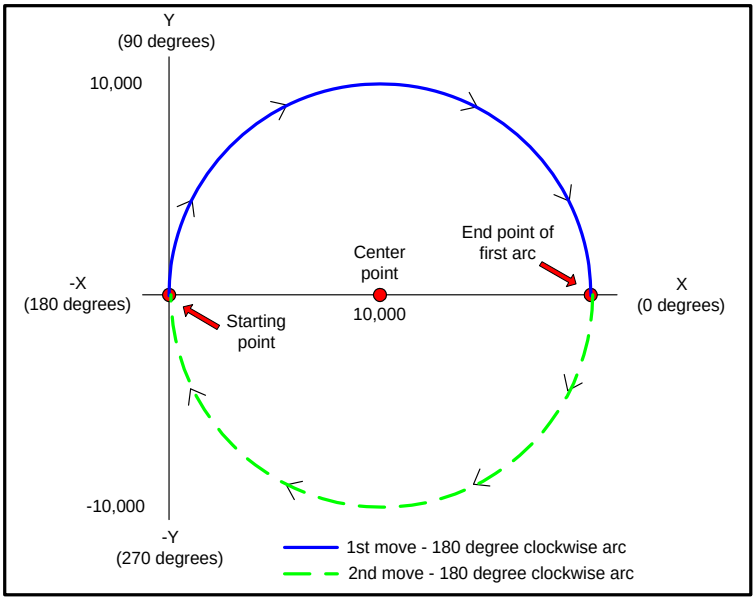
The ***MCArcEndingAngle( )*** function is used to execute an arc move by specifying the ending angle and center point of an arc. The *Axis* parameter should equal the controlling axis for the contour move. The parameter *Angle* should equal the ending angle (absolute or relative) of the arc. When using this method to specify an arc, the ***MCMoveAbsolute( )*** and ***MCMoveRelative( )*** functions are not used.
The ***MCArcCenter( )*** function defines the radius of the arc. An example of two arc motions is shown below:

```
MCSetOperatingMode( hCtlr, 1, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 2, 1, MC_MODE_CONTOUR );


//  Clockwise arc move #1
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_CW, 1 );
   MCArcCenter( hCtlr, 1, MC_CENTER_ABS, 10000.0 );
   MCArcCenter( hCtlr, 2, MC_CENTER_ABS, 0.0 );
   MCArcEndAngle( hCtlr, 1, MC_ABSOLUTE, 0.0 );
MCBlockEnd( hCtlr, NULL );


//  Clockwise arc move #2
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_CW, 1 );
   MCArcCenter( hCtlr, 1, MC_CENTER_REL, -10000.0 );
   MCArcCenter( hCtlr, 2, MC_CENTER_REL, 0.0 );
   MCArcEndAngle( hCtlr, 1, MC_RELATIVE, 1800.0 );
MCBlockEnd( hCtlr, NULL );
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Changing the velocity 'on the fly'

'On the fly' velocity changes during contour mode motion are accomplished by using the **VelocityOverride** member of the **MCContour** data structure. Issue the command (to the controlling axis) to scale the vector velocity of a linear or arc motion. The rate of change is defined by the current settings for vector acceleration and vector deceleration.

Changing the velocity of a contour group using Velocity Override is not supported for S-curve and/or Parabolic velocity profiles.

Cubic Spline Interpolation of linear moves

To have the controller perform 'curve fitting' (cubic spline interpolation) on a series of linear moves, issue the ***MCEnableSynch( )*** command to the controlling axis before issuing any contour move commands. Next issue linear contour path commands to points on the curve. After loading the desired number of moves into the contour buffer, issue a ***MCGOEx( )*** command with the value *Param* set to 1. Motion will continue from the first to the last point in the contour buffer. To return to normal operation, issue the ***MCEnableSynch( )*** command with parameter *pState* = FALSE.

Note that when performing cubic spline interpolation, only **128 motions** can be queued up in the contouring buffer.

User Defined Contour path

When executing contour motion the controller assumes that the axes are arranged in an orthogonal geometry. The controller will calculate the distance and period of a move as follows:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Beginning position: X=0 Y=0 Z=0
Target position: X=10,000 Y=10,000 Z=1000

Calculated Contour Distance = $\sqrt{(X^2 + Y^2 + Z^2)}$
$= \sqrt{(10{,}000^2 + 10{,}000^2 + 1{,}000^2)}$
$= \sqrt{(100{,}000{,}000 + 100{,}000{,}000 + 1{,}000{,}000)}$
$= \sqrt{201{,}000{,}000}$
$= 14177.44$

The period, or elapsed time of the move, is a simple matter of applying the current settings for Vector Acceleration + Vector Velocity + Vector Deceleration to the Calculated Contour Distance.

For applications where orthogonal geometry is not applicable, the controller allows the user to define a custom contour distance. This is accomplished by:

1) The command sequence must be preceded by the **C**ontour **P**ath (***a*CP*n***) command (***a*** = the controlling axis) with parameter ***n*** = 0.
2) **C**ontour **D**istance (***a*CD*n***) must be the last command in the compound command sequence, with parameter ***n*** = the Calculated Contour Distance of the move

The controller will use the current settings for vector velocity, vector acceleration, and vector deceleration to calculate the period of the motion. When a User Defined Contour Path is selected (***MCBlockBegin*** with parameter *nMode* set to **MC_BLOCK_CONTR_USER**), the ***MCContourDistance( )*** function is used to enter the non-orthogonal contour distance.

```
MCSetOperatingMode( hCtlr, 1, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 2, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 3, 1, MC_MODE_CONTOUR );


//  User defined move #1
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_USER, 1 );
   MCMoveAbsolute( hCtlr, 1, 1000.0 );
   MCMoveAbsolute( hCtlr, 2, 1000.0 );
   MCMoveAbsolute( hCtlr, 3, 1000.0 );
   MCContourDistance( hCtlr, 1, 10000.0 );
MCBlockEnd( hCtlr, NULL );


//  User defined move #2 - the Distance parameter is 10,000 + 10,000 =
20,000
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_USER, 1 );
   MCMoveAbsolute( hCtlr, 1, 0.0 );
   MCMoveAbsolute( hCtlr, 2, 0.0 );
   MCMoveAbsolute( hCtlr, 3, 0.0 );
   MCContourDistance( hCtlr, 1, 20000.0 );
MCBlockEnd( hCtlr, NULL );
```



For the ***MCContourDistance( )*** function, the parameter *Distance* is an absolute value, relative to the positions of the included axes when the ***MCSetOperatingMode( )*** function was last issued. Re-issuing the ***MCSetOperatingMode( )*** function will reset the current contour distance to zero.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Special case: setting the Maximum Velocity of an Axis

When executing simple point to point or velocity mode motions the maximum velocity of each axis is set individually. When executing multi axis contour moves, the maximum velocity is typically expressed as the velocity of the 'end effector' of the contour group. In a cutting application the 'end effector' would be the tool doing the cutting (torch, laser, knife, etc…). Setting the maximum velocity of an axis in the contoured group is not typically supported.

By combining a user define contour path (***MCBlockBegin*** with parameter *nMode* set to **MC_BLOCK_CONTR_USER**) with the ***MCContourDistance( )*** command with parameter *Distance* = 0, the user can execute multi axis contour moves and limit the maximum velocity of an individual axis. In this mode of operation the ***MCVectorVelocity( )*** command is **not** used to set the velocity of the contour group. The axis with the **longest move time** (calculated by distance, velocity, acceleration, and deceleration) will define the total time for the contour move. For moves of sufficient distance where the axis has enough time to fully accelerate, this one axis will move at its preset maximum velocity. All axes will move at or below their specified maximum velocities. All axes will start and stop at the same time. In the following example, axes 1 and 2 are commanded to move the same distance but the maximum velocity for axis two is 1/3 that of axis one. Since both axes are moving the same distance, they will both travel at a maximum velocity of 100 counts per second.

```
MCSetVelocity( hCtlr, 1, 300.0 );
MCSetAcceleration( hCtlr, 1, 1000.0 );
MCSetDeceleration( hCtlr, 1, 1000.0 );

MCSetVelocity( hCtlr, 2, 100.0 );
MCSetAcceleration( hCtlr, 2, 1000.0 );
MCSetDeceleration( hCtlr, 2, 1000.0 );


MCSetOperatingMode( hCtlr, 1, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 2, 1, MC_MODE_CONTOUR );

MCContourdistance( hCtlr, 1, 0.0 );


MCBlockBegin( hCtlr, MC_BLOCK_CONTR_USER, 1 );
   MCMoveRelative( hCtlr, 1, 1000.0 );
   MCMoveRelative( hCtlr, 2, 1000.0 );
MCBlockEnd( hCtlr, NULL );
```

If the commanded move distance of axis one was 2000 counts it would move at a higher velocity than axis two, but it would not reach its maximum velocity as set by the **MCSetVelocity( )** command.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Electronic Gearing

MultiFlex motion controllers support slaving any axis or axes to a master. Moving the master axis will cause the slave to move based on the specified slave ratio. The optimal position of the slave axis is calculated by multiplying the optimal position of the master by the gearing ratio of the slave. The slave's optimal position is maintained proportional to the master's position. This can be used in applications where multiple motors drive the same load. Gearing supports both servos and stepper axes, with the master axis operating in position, velocity, or contouring mode. If a following error or limit error occurs on any of the geared axes (master or slaves) all axes in the geared group will stop.

The Motion Control API function ***MCEnableGearing( )*** configures and initiates gearing. The slave ratio can be set to any integer or real value. If the slave ratio is a positive value, a move in the positive direction of the master will cause a move in the positive direction of the slave. If the slave ratio is a negative value, a move in the positive direction of the master will cause a move in the negative direction of the slave. The following program example configures axes 2, 3, and 4 as slaves of axis 1.

```
// Enable gearing of axis 2, 3, and 4
// Move axis 1 (master), slaves (axes 2, 3, and 4) will move at define
ratio
MCEnableGearing( hCtlr, 2, 1, 0.5, TRUE );
MCEnableGearing( hCtlr, 3, 1, 12.87, TRUE );
MCEnableGearing( hCtlr, 4, 1, -125, TRUE );
MCMoveRelative (hCtlr, 1, 215.0 );


// disable gearing
MCEnableGearing( hCtlr, 2, 1, 0.5, FALSE );
MCEnableGearing( hCtlr, 3, 1, 12.87, FALSE );
MCEnableGearing( hCtlr, 4, 1, -125, FALSE );
```



Note – if the slave axes are servo's or closed loop steppers, the **PID parameters** for each axis **must be defined prior** to beginning master/slave operation.



Note – Changing the slave ratio 'on the fly' may cause the mechanical system to 'jerk' or to 'error out' (following error).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Jogging

In some applications it may be necessary to have a means of manually positioning the motors. Since the controller is able to control the motion of servos and steppers with precision at both low and high speeds, all that is required to support manual positioning is: .

- A PC with a game port
- A PC joystick
- PC based software that positions the axes in Velocity mode

Jogging without writing software

One of the tools provided with the Motion Control API is the Joystick Demo. This tool allows the user to configure and then jog one or two axes.

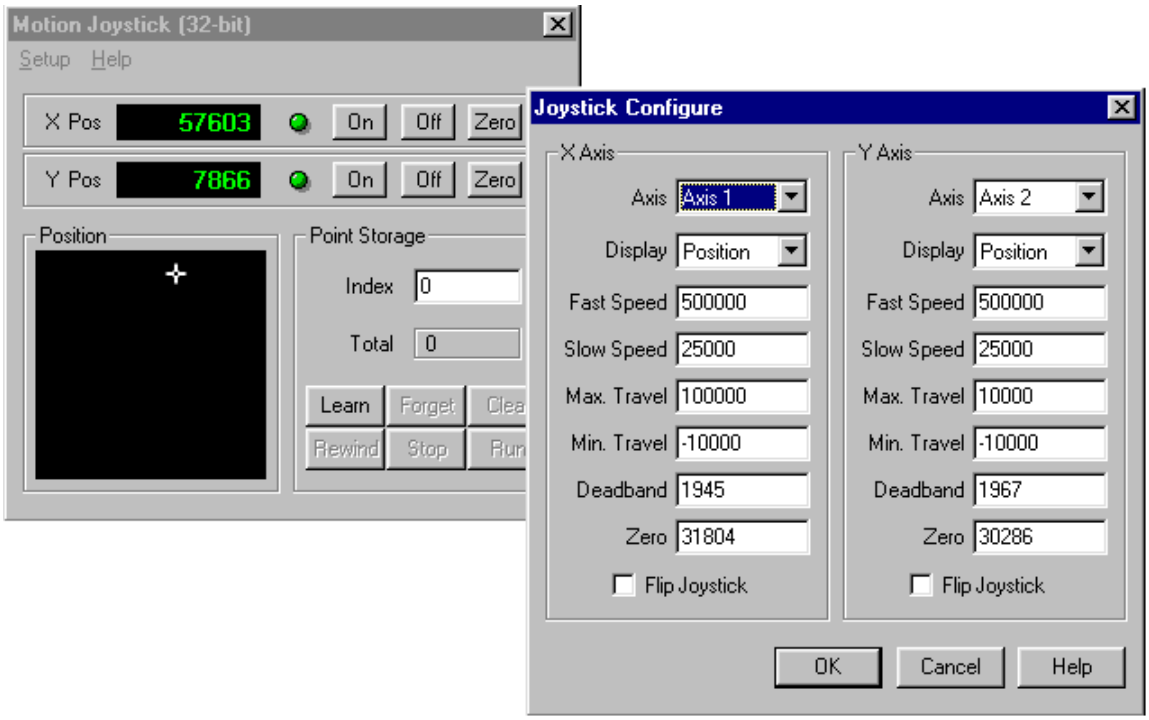


Figure 35. Joystick Demo program

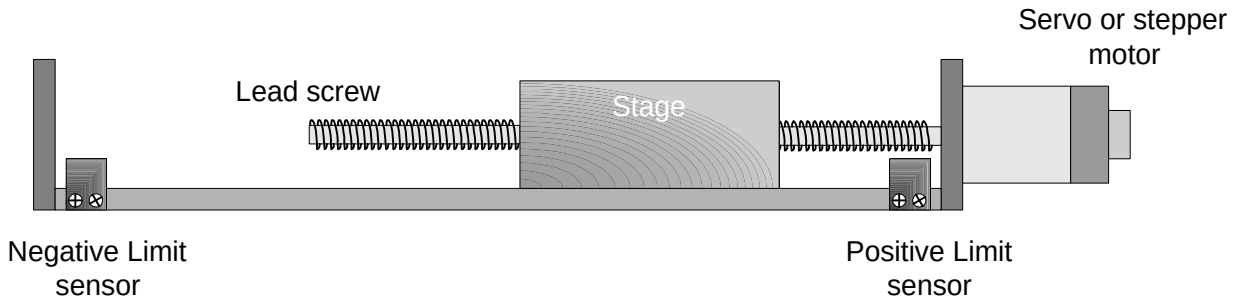
Using the Joystick Demo in your application program

After the Motion Control API has been installed the source files for the Joystick Demo are available in the Motion Control folder \Program Files\Motion Control\Motion Control API\Sources\Joy.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

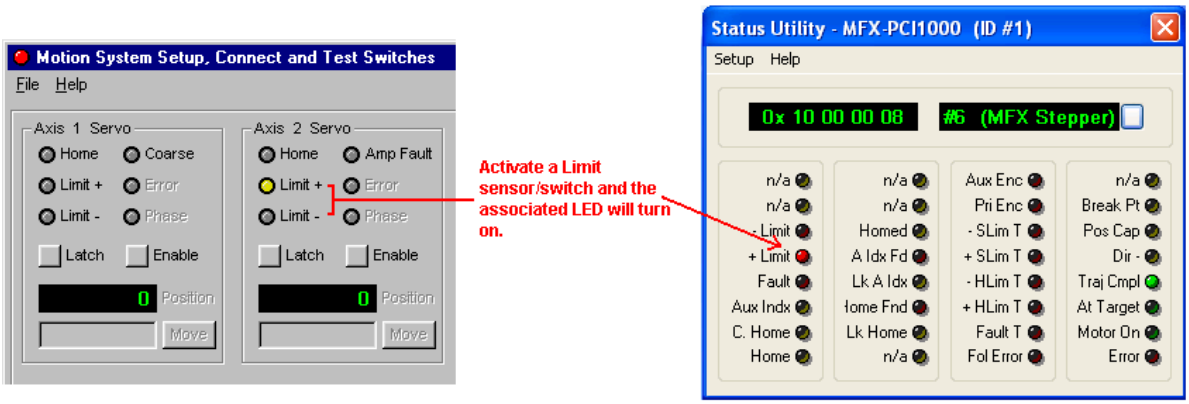
Defining Motion Limits

The controller implements two types of motion limits error checking. End of travel or 'Hard' limit switch/sensor inputs and 'soft' user programmable position limits.



Hard Limits

The Limit + and Limit - inputs of the controller use bi-directional optical isolators for interfacing to the external limit sensors. For example wiring diagrams refer to pages 58 and 59. For axis I/O circuit schematics refer to page 199. Use the Motion Integrator Motion System Setup Test Panel or the Status Panel Utility to test the limit sensors, wiring, and controller operation.



By default all optically isolated inputs indicate that an input is **on** when the opto device is conducting. For a limit sensor circuit that operates like a 'normally open' switch, when the switch is closed (opto isolator is conducting) the associated Limit +/- status bit will be set to a

For fail safe limit operation a 'normally closed' circuit can be used by issuing the ***MCConfigureDigitalIO( )*** with *Mode* **value = MC_DIO_LOW**. This will invert the reported limit state so that the Limit +/- status bit will be set to a 1 if either the switch is opened or one of the limit circuit wires is broken.

The controller supports two levels of limit switch handling:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

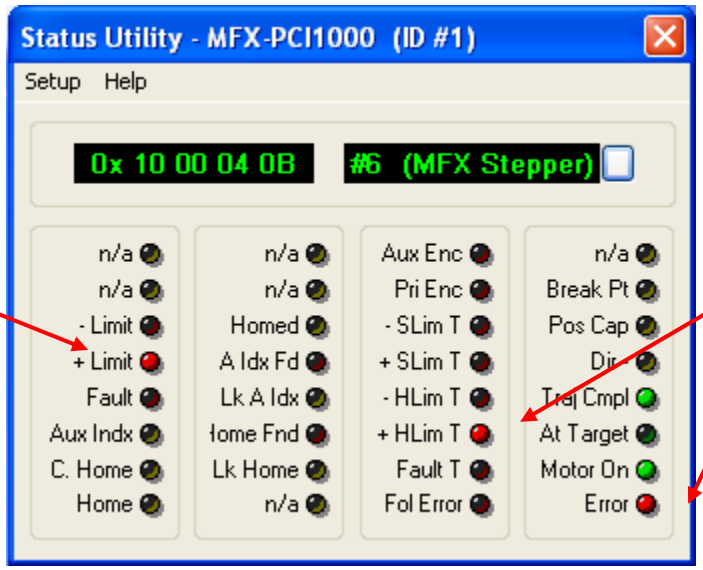
Auto axis disable
Simple monitoring

The Motion Control API function ***MCSetLimits( )*** allows the user to enable the Auto Axis Disable capability of the controller. This feature implements a hard coded operation that will stop motion of an axis when a limit switch is active. This background operation requires no additional controller processor time, and once enabled, requires no intervention from the user's application program. However it is recommended that the user periodically check for a limit tripped error condition using the ***MCGetStatus( )***, ***MCDecodeStatus( )*** functions. The ***MCSetLimit( )*** function provides the following limit flags:

Flag	Description
MC_LIMIT_PLUS	Enables the Positive/High hard limit
MC_LIMIT_MINUS	Enables the Negative/Low hard limit
MC_LIMIT_BOTH	Enables the Positive and Negative hard limits
MC_LIMIT_OFF	Turn off the axis when the hard limit input 'goes' active
MC_LIMIT_ABRUPT	Stop the axis abruptly when the hard limit input goes active
MC_LIMIT_SMOOTH	Decelerate and stop the axis when the hard limit input goes active
MC_LIMIT_INVERT	Invert the active level of the hard limit input. Typically used for normally closed limit sensors. Do not use if ***MCConfigureDigitalIO( )*** with *Mode* **value = MC_DIO_LOW** is being used to invert the reported state of a limit input .

When a limit event occurs, motion of that axis will stop and the error flags (MC_STAT_ERROR and MC_STAT_PLIM_TRIP or MC_STAT_MLIM_TRIP) will remain set until the motor is turned back on by ***MCEnable( )***. The axis must then be moved out of the limit region with a move command (**MCMoveAbsolute( )**, ***MCMoveRelative( )***). The Status Panel screen shot below shows the typical display when a hard limit sensor is tripped during a move.

The Limit + bit indicates the current state of the Limit + input.



The +HLim T and Error bits indicate that a over travel limit error has occurred.

```
// Set the both hard limits of axis 1 to stop smoothly when tripped,
ignore
// soft limits:
//
MCSetLimits( hCtlr, 1, MC_LIMIT_BOTH | MC_LIMIT_SMOOTH, 0, 0.0, 0.0 );
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
// Set the positive hard limit of axis 2 to stop by turning the motor
off.
// Because axis 2 uses normally closed limit switches we must also invert
the
// polarity of the limit switch. Soft limits are ignored.
MCSetLimits( hCtlr, 2, MC_LIMIT_PLUS | MC_LIMIT_OFF | MC_LIMIT_INVERT, 0,
0.0, 0.0 );
```



In Position and Velocity mode the response to an activated limit input is direction sensitive, the axis will only be stopped if it is moving in the direction of the activated limit switch. In Contour mode, the response to an activated limit input is not direction sensitive, the axis will be stopped regardless of the direction it is moving if either limit switch is activated. In Torque mode, the controller will ignore the activation of a limit input, the axis will continue to move.

If the user does not want to use the Auto Axis Disable feature, the current state of the limit inputs can be determined by polling the controller using the ***MCGetStatus( )***, ***MCDecodeStatus( )*** functions. The flag for testing the state of the Limit + input is **MC_STAT_INP_PLIM**. The flag for testing the state of the Limit - input is **MC_STAT_INP_MLIM**.

Soft Limits

Soft motion limits allow the user to define an area of travel that will cause an error condition. When enabled, if an axis is commanded to move to a position that is outside the range of motion defined by the ***MCSetLimit( )*** function, an error condition is indicated and the axis will stop. The ***MCSetLimit( )*** function provides the following limit flags:

Table 3. Motion Control API Limit Mode Flags

Flag	Description
MC_LIMIT_PLUS	Enables the High/Positive soft limit
MC_LIMIT_MINUS	Enables the Low/Negative soft limit
MC_LIMIT_BOTH	Enables the High and Low soft limits
MC_LIMIT_OFF	Turn off the axis when the hard limit input ‘goes’ active
MC_LIMIT_ABRUPT	Stop the axis abruptly when the hard limit input goes active
MC_LIMIT_SMOOTH	Decelerate and stop the axis when the hard limit input goes active

When a soft limit error event occurs, the error flags (**MC_STAT_ERROR** and **MC_STAT_PSOFT_TRIP** or **MC_STAT_MSOFT_TRIP**) will remain set until the motor is turned back on by ***MCEnable( )***. The axis must then be moved back into the allowable motion region with a move command (**MCMoveAbsolute( )**, ***MCMoveRelative( )***).

```
// Assume axis 3 is a linear motion with 500 units of travel. Set the
both
// hard limits of this axis to stop abruptly. Set up soft limits that
will
// stop the motor smoothly 10 units from the end of travel (i.e. at 10
// and 490).

MCSetLimits( hCtlr, 3, MC_LIMIT_BOTH | MC_LIMIT_ABRUPT, MC_LIMIT_BOTH |
MC_LIMIT_SMOOTH, 10.0, 490.0 );
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Homing Axes

When power is applied or the controller is reset, the current position of all servo and stepper axes are initialized to zero. If they are subsequently moved, the controller will report their positions relative to the position where they were last initialized. At any time the user can call the ***MCSetPosition( )*** function to re-define the position of an axis.

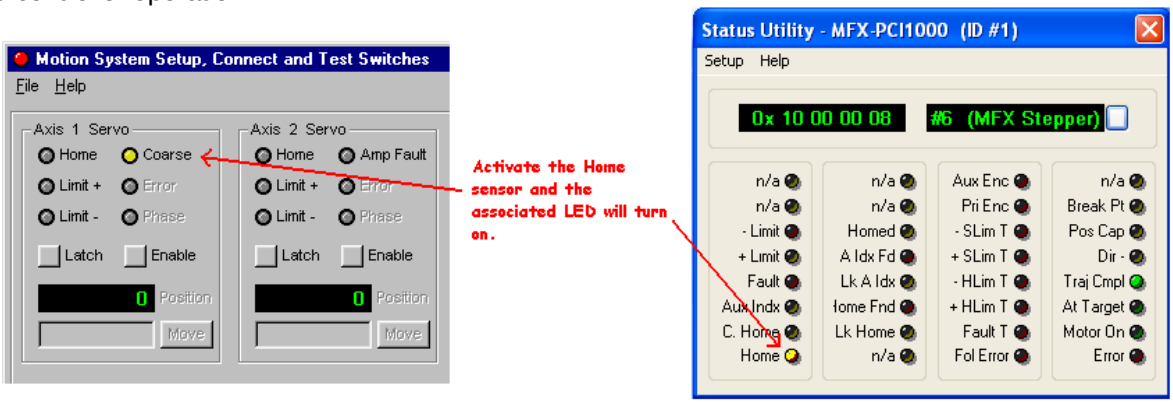
In most applications, there is some position/angle of the axis (or mechanical apparatus) that is considered 'home'. Typical automated systems utilize electro-mechanical devices (switches and sensors) to signal the controller when an axis has reached this position. The controller will then define the current position of the axis to a value specified by the user. This procedure is called a homing sequence. The controller is not shipped from the factory programmed to perform a specific homing operation. Instead, it has been designed to allow the user to define a custom homing sequence that is specific to the system requirements. The controller provides the user with two different options for homing axes:

1) **High level function calls using the Motion Control API** - Easy to program homing sequences usingMotion Control API function calls.

2) **MCCL homing macro's stored in on-board memory** - When executed as background tasks, MCCL homing macro's allow the user to home multiple axes simultaneously. For additional information on macro's and background tasks please refer to the **Motion Control Command Language (MCCL) Reference manual**.

Connecting a Home Sensor

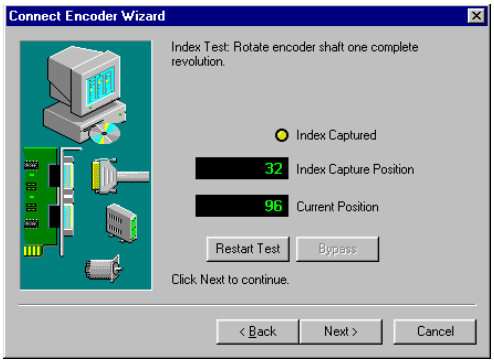
The Home inputs (Coarse Home - servo's & closed loop steppers, Home – open loop stepper) of the controller use bi-directional optical isolators for interfacing to the external home sensor. For example wiring diagrams refer to page 60. For axis I/O circuit schematics refer to page 199. Use the Motion Integrator Motion System Setup Test Panel or the Status Panel Utility to test the home sensors, wiring, and controller operation.



Verifying the operation of the Index Mark of an Encoder

Most closed loop system applications will use the Index mark of the encoder to define the 'home' position of a servo. Use Motion Integrator's Connect Encoder Wizard to verify the proper operation of the encoder index.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Programming Homing Routines

The controller provides sophisticated programming support for homing closed loop servos, Closed Loop Steppers, and Open Loop Steppers. The following two tables summarize which commands are provided for homing operations.

Table 4. Motion Control API Homing Functions

Axis Type	Functions	Input	Notes
Closed Loop Servo	MCIndexArm MCWaitForIndex MCIsIndexFound	Encoder Index	
Closed Loop Servo	MCFindIndex	Encoder Index	Use only from within background task
Closed Loop Stepper	MCIndexArm MCWaitForIndex MCIsIndexFound	Aux. Encoder Index	
Closed Loop Stepper	MCFindIndex	Aux. Encoder Index	Use only from within background task
Open Loop Stepper	MCEdgeArm MCWaitForEdge MCIsEdgeFound	Home	
Open Loop Stepper	MCFindEdge	Home	Use only from within background task

Table 5. MCCL Homing Commands

Axis Type	Command	Input	Notes
Closed Loop Servo	IA & WI	Encoder Index	
Closed Loop Servo	FI	Encoder Index	Use only from within background task
Closed Loop Stepper	IA & WI	Aux. Encoder Index	
Closed Loop Stepper	FI	Aux. Encoder Index	Use only from within background task
Open Loop Stepper	EL & WE	Home	
Open Loop Stepper	FE	Home	Use only from within background task

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Homing a Rotary Stage (closed loop servo or closed loop stepper) with the Encoder Index

Many servo motor encoders generate an index pulse once per rotation. For a multi turn rotary stage, where one rotation of the encoder equals one rotation of the stage, an index mark alone is sufficient for homing the axis. When an axis need only be homed within 360 degrees no additional qualifying sensors (coarse home) are required.



The following C example uses the ***MCIndexArm( )***, ***MCIsIndexFound( )***, and ***MCWaitForIndex( )*** functions for homing a closed loop system. For complete C code homing samples that can be cut and pasted into an application program please refer to the Motion Control API on-line help (mcapi.hlp).

```
// Arm index and wait for index to be found
//
MCIndexArm( hCtlr, 1, 0.0 );
if (!MCIsIndexFound( hCtlr, 1, 10.0 )) {
    // Index not found within time limit (10 seconds),
    // error handling code goes here
}
//
// Process index and stop motor
//
MCWaitForIndex( hCtlr, 1 );       // controller 'processes' index data
MCStop( hCtlr, 1 );               // stop
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
Sleep( 100 );     // let motor settle 100 msec (WIN32 API function)
```



The following MCCL example uses the **I**ndex **A**rm (***a*IA*n***) and the Wait for **I**ndex (***a*WI**) commands to home a closed loop system.

```
;MCCL rotary axis homing sequence index mark
MD10,1SV10000,1VM,1DI0,1GO,1IA0,LU"STATUS",1RL@0,IC18,JR-3,NO,1WI,MJ11
                                        ;move, arm and capture index
MD11,1ST,1WS.01,1PM,1MN,1MA0,1WS.01     ;stop, initialize axis, move to
index
                                        ;mark
```

Homing a Closed Loop Axis with Coarse Home and Encoder Index Inputs

A typical axis will incur multiple rotations of the motor/encoder over the full range of travel. This type of system will typically utilize a coarse home sensor to qualify which of the index pulses is to be used to home the axis. The Limit Switches (end of travel) provide a dual purpose:

1) Protect against damage of the mechanical components.
2) Provide a reference point during the initial move of the homing sequence

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

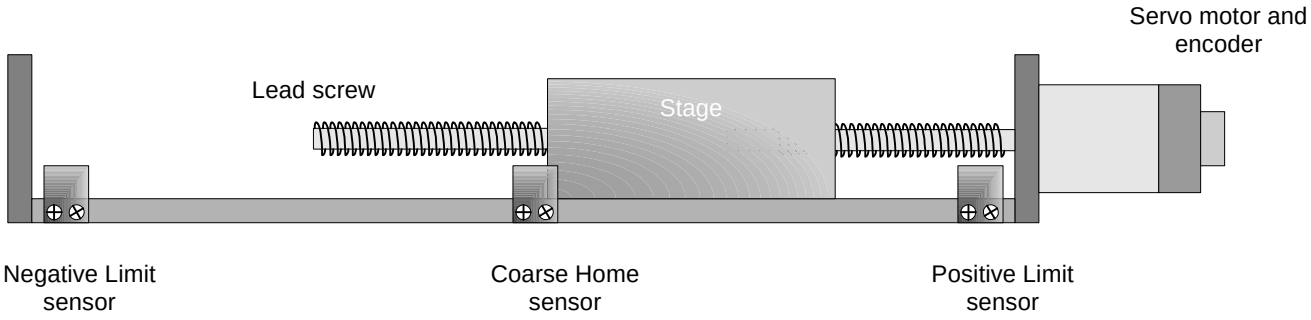


Figure 36. Typical Linear Stage

When power is applied or the controller is reset, the position of the stage is unknown. To home the axis a velocity mode move in the positive direction is commanded, checking the status of both the Coarse Home sensor and the Limit + sensor. Once the axis is within the Coarse Home sensor the ***MCIndexArm( )***, ***MCIsIndexFound( )***, and ***MCWaitForIndex( )*** functions are used to reference the reported position of the axis to the index mark. The ***MCEnableAxis( )*** function completes the homing operation by reinitialize all position registers. The following flow chart describes a typical homing procedure. If the positive limit sensor is activated the stage will change direction prior to homing the axis.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Homing a Closed Loop System -
Encoder Index, Coarse Home Sensor, and Over Travel Limits

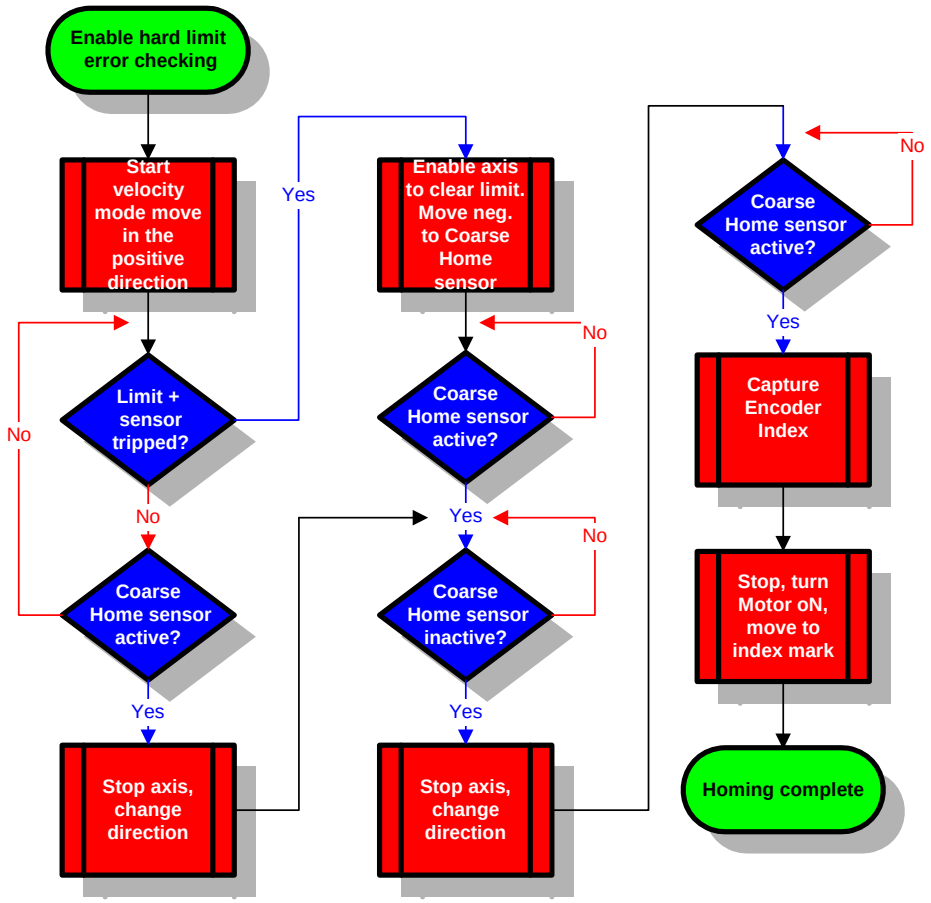


Figure 37. Typical homing routine for a servo

The following C example uses the ***MCIndexArm( )***, ***MCIsIndexFound( )***, and ***MCWaitForIndex( )*** functions for homing a closed loop system. For complete C code homing samples that can be cut and pasted into an application program please refer to the Motion Control API on-line help (mcapi.hlp).

```
// Motion Control API linear stage homing sequence using the index mark
//
MCIndexArm( hCtlr, 1, 1000.0 );
if (!MCIsIndexFound( hCtlr, 1, 10.0 )) {
    // Index not found within time limit (10 seconds),
    // error handling code goes here
}
// Process index and stop motor
MCWaitForIndex( hCtlr, 1 );       // controller 'processes' index data
MCStop( hCtlr, 1 );               // stop
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
Sleep( 100 );     // let motor settle 100 msec (WIN32 API function)


// Move back to location of index mark
//
MCSetOperatingMode( hCtlr, 1, 0, MC_MODE_POSITION );
MCEnableAxis( hCtlr, 1, TRUE );
MCMoveAbsolute( hCtlr, 1, 0.0 );
MCIsStopped( hCtlr, 1, 2.0 );
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
Sleep( 100 );
```



The following MCCL example uses the **I**ndex **A**rm (***a*IA*n***) and the Wait for **I**ndex (***a*WI**) commands to home a closed loop system.

```
;MCCL linear stage homing sequence using the index mark
MD10,1IA1000,MC20,1WI,1ST,1WS.01,MJ11      ;capture index (position = 1000) then
                                        stop
MD11,1PM,1MN,1MA1000,1WS.1                 ;initialize axis, move to index


;homing sub routines
MD20,LU"STATUS",1RL@0,IS18,BK,NO,JR-5          ;test for Index Found
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Homing a Closed Loop Axis with a Limit sensor

An axis can be homed even if no index mark or coarse home sensor is available. This method of homing utilizes one of the limit (end of travel) sensors to also serve as a home reference.



This method **is not recommended** for applications that require **high repeatability and accuracy**. To achieve the highest possible accuracy when using this method, significantly reduce the velocity of the axis while polling for the active state of the limit input.

The following Motion Control API and MCCL sequences will home an axis at the position where the positive limit sensor 'goes active':



The following C example uses the ***MCSetPosition( )*** function to redefine the encoder position a closed loop system. For complete C code homing samples that can be cut and pasted into an application program please refer to the Motion Control API on-line help (mcapi.hlp).

```
// Motion Control API homing sequence (using positive limit sensor)
// the axis must have already been moved into (and tripped) the positive
limit
// sensor


// Once the positive limit switch is active, move negative until switch is inactive
//
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
MCEnableAxis( hCtlr, 1, TRUE );
MCDirection( hCtlr, 1, MC_DIR_NEGATIVE );
MCSetVelocity( hCtlr, 1, 1000.0 );
MCGoEx( hCtlr, 1, 0.0 ) );
dwStatus = MCGetStatus( hCtlr, 1);
if (!MCDecodeStatus( hCtlr, dwStatus, MC_STAT_INP_PLIM)) {
   dwStatus = MCGetStatus( hCtlr, 1)
}

// Stop the axis and define the leading edge of the limit switch as position 0
//
MCAbort( hCtlr, 1 );
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
MCSetPosition( hCtlr, 1, 0.0 );
MCSetOperatingMode( hCtlr, 1, 0, MC_MODE_POSITION );
MCEnableAxis( hCtlr, 1, TRUE );
MCMoveAbsolute( hCtlr, 1, -100.0 );
```

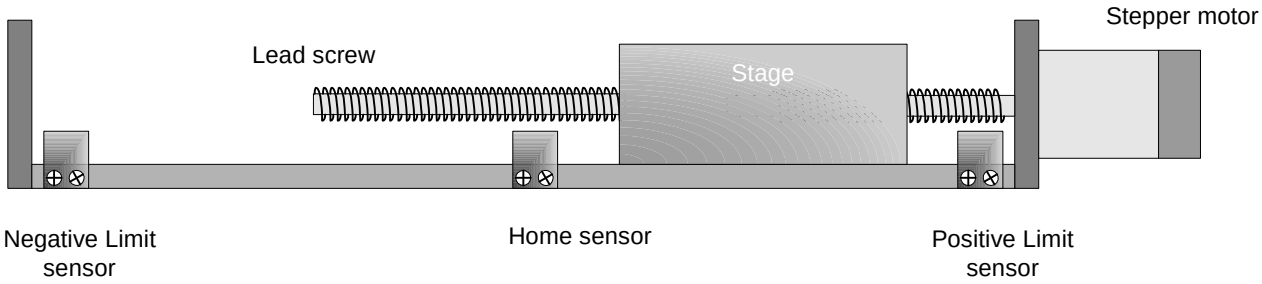

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The following MCCL example uses the **D**efine **H**ome (***aDHn***) command to redefine the encoder position of a closed loop system.

```
; MCCL linear stage homing sequence using the positive limit sensor
MD1,1LM2,1LN3,MJ10                          ;call homing macro
MD10,1VM,1DI0,1GO,LU"STATUS",1RL@0,IS10,MJ11,NO,JR-5
                                            ;move and poll the Limit + sensor
MD11,1WS0.01,1MN,1DI1,1SV1000,1GO,LU"STATUS",1RL@0,IC28,MJ12,NO,JR-5
                                            ;move negative until limit + inactive
MD12,1AB,1WS.1,1DH0,1PM,1MN,1MA-100         ;stop immediately when limit + not active,
                                            ;define position as 0. Move to position –
100.
```

Homing open loop steppers

Open loop steppers are typically homed based on the position of a home sensor. Unlike servos that use a precision reference index mark, steppers are more prone to homing inaccuracies due the lower repeatability of the single electro mechanical home sensor. To achieve the highest possible repeatability; reduce the velocity of the axis and always approach the home sensor from the same direction. Here is a typical linear axis controlled by an open loop stepper motor. A home sensor defines the home position of the axis. End of travel or Limit Switches are used to protect against damage of the mechanical components.



When power is applied or the controller is reset, the position of the stage is unknown. The following command sequence will move the stage in the positive direction. If the positive limit sensor is activated before the Home sensor the stage will change direction, until home sensor is located. When the Home sensor is activated the ***MCEdgeArm ( )*** and ***MCIsEdgeFound ( )*** functions are used to capture the position of the Home sensor active edge.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

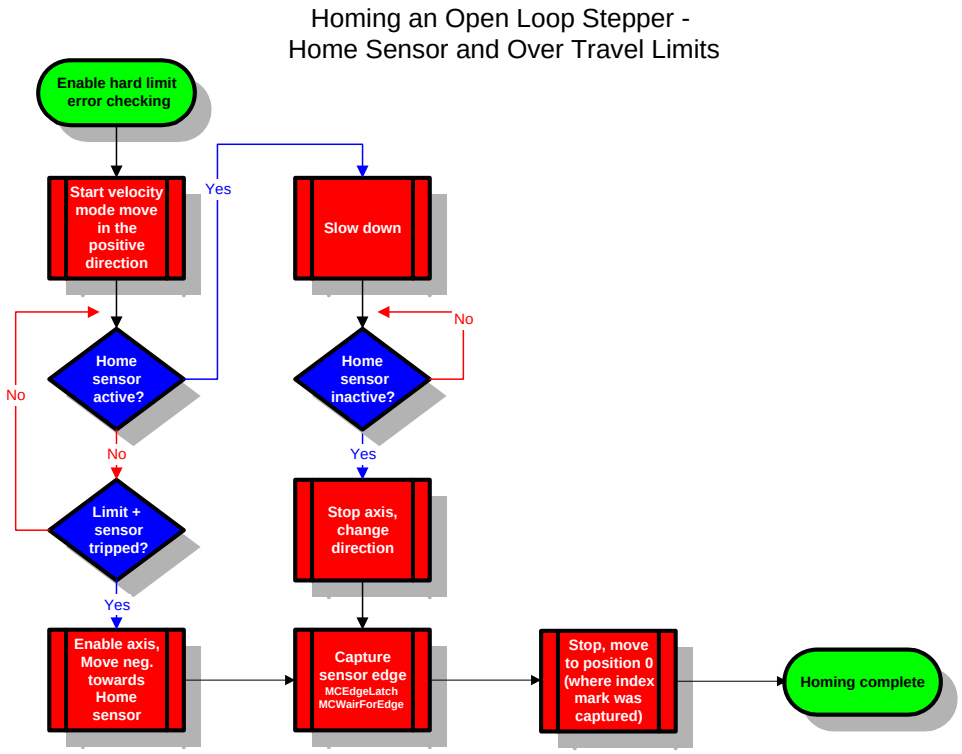


Figure 38. Typical homing routine for a stepper

The following C example uses the ***MCEdgeArm( )***, ***MCIsEdgeFound( )***, and ***MCWaitForEdge( )*** functions for homing a closed loop system. For complete C code homing samples that can be cut and pasted into an application program please refer to the Motion Control API on-line help (mcapi.hlp).

```
// Motion Control API open loop stepper linear stage homing sequence using
the home sensor
//
MCEdgeArm( hCtlr, 1, 1000.0 );
if (!MCIsEdgeFound( hCtlr, 1, 10.0 )) {
    // Edge not found within time limit (10 seconds),
    // error handling code goes here
}
// Process edge and stop motor
MCWaitForEdge( hCtlr, 1 );       // controller 'processes' edge data
MCStop( hCtlr, 1 );               // stop
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
Sleep( 100 );     // let motor settle 100 msec (WIN32 API function)


// Move back to location of home sensor edge
//
MCSetOperatingMode( hCtlr, 1, 0, MC_MODE_POSITION );
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
MCEnableAxis( hCtlr, 1, TRUE );
MCMoveAbsolute( hCtlr, 1, 0.0 );
MCIsStopped( hCtlr, 1, 2.0 );
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
Sleep( 100 );
// Enable / disable axis to set MC_STAT_INP_INDEX to monitor the current
// state (not capture & latch) of Home sensor
MCEnableAxis( hCtlr, 1, FALSE );
MCWait( hCtlr, 0.01 );
MCEnableAxis( hCtlr, 1, TRUE );
```



Prior to issuing ***MCEdgeArm ( )*** the status flag **MC_STAT_INP_INDEX** will indicate the current state of the Home Sensor (1 = active, 0 = inactive). After issuing ***MCEdgeArm ( )*** **MC_STAT_INP_INDEX** will be latched when the Home sensor edge has been captured. To clear latching of **MC_STAT_INP_INDEX** issue:

MCEnableAxis(hCtlr, 1, FALSE);
MCEnableAxis(hCtlr, 1, TRUE);



The following MCCL example uses the **E**dge **A**rm (***aEAn***) and the Wait for **E**dge (***a*WE**) commands to home a closed loop system.

```
; MCCL Stepper linear stage homing sequence using Home & positive limit
;sensors
MD5,5LM2,5LN3,MJ10                          ;enable limits, call homing macro
MD10,5VM,5DI0,5SV10000,5GO,LU"STATUS",5RL@0,IS24,MJ11,NO,IS10,MJ13,NO,JR-8
                                            ;test for sensors (home and
+limit)
MD11,LU"STATUS",5RL@0,IC24,MJ12,NO,JR-5     ;continue moving until home
sensor off
MD12,5ST,5WS.1,5DI1,5SV5000,5GO,MJ14        ;move back to the home sensor
MD13,5WS0.01,5MN,5DI1,5SV5000,5GO,MJ14      ;move out of limit sensor range
                                            ;back toward the home sensor
MD14,5EL0,MC15,5WE,5ST,5WS.1,5MF,5MN,5PM,5MA-100
                                             ;capture the active edge of the
                                             ;home sensor. Stop axis and
                                             ;define a position 0, ;move to
                                             ;position -100
MD15,LU"STATUS",5RL@0,IS18,BK,NO,JR-5       ;loop status for Edge found bit
set
```



Prior to issuing **E**dge **L**atch (***a*ELn**) the status bit 24 Index / Home will indicate the current state of the Home Sensor (1 = active, 0 = inactive). After issuing **E**dge **L**atch (***a*ELn**) status bit 24 will be latched when the Home sensor edge has been captured. To clear latching issue:

1MF,1MN

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Homing a Open Loop Stepper with a Limit sensor

An axis can be homed even if no home sensor is available. This method of homing utilizes one of the limit (end of travel) sensors to also serve as a home reference. The following Motion Control API and MCCL sequences will home an axis at the position where the positive limit sensor 'goes active':



The following C example uses the ***MCSetPosition( )*** function to redefine the encoder position a closed loop system. For complete C code homing samples that can be cut and pasted into an application program please refer to the Motion Control API on-line help (mcapi.hlp).

```
// Motion Control API homing sequence (using positive limit sensor)
// the axis must have already been moved into (and tripped) the positive
limit
// sensor


// Once the positive limit switch is active, move negative until switch is inactive
//
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
MCEnableAxis( hCtlr, 1, TRUE );
MCDirection( hCtlr, 1, MC_DIR_NEGATIVE );
MCSetVelocity( hCtlr, 1, 1000.0 );
MCGoEx( hCtlr, 1, 0.0 ) );
dwStatus = MCGetStatus( hCtlr, 1);
if (!MCDecodeStatus( hCtlr, dwStatus, MC_STAT_INP_PLIM)) {
   dwStatus = MCGetStatus( hCtlr, 1)
}

// Stop the axis and define the leading edge of the limit switch as position 0
//
MCAbort( hCtlr, 1 );
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
MCSetPosition( hCtlr, 1, 0.0 );
MCSetOperatingMode( hCtlr, 1, 0, MC_MODE_POSITION );
MCEnableAxis( hCtlr, 1, TRUE );
MCMoveAbsolute( hCtlr, 1, -100.0 );
```



The following MCCL example uses the **D**efine **H**ome (***aDHn***) command to redefine the encoder position of a closed loop system.

```
; MCCL linear stage homing sequence using the positive limit sensor
MD5,5LM2,5LN3,MJ10                      ;call homing macro
MD10,5VM,5DI0,5GO,LU"STATUS",5RL@0,IS10,MJ11,NO,JR-5
                                        ;move and poll the Limit + sensor
MD11,5WS0.01,5MN,5DI1,5SV1000,5GO,LU"STATUS",5RL@0,IC28,MJ12,NO,JR-5
                                        ;move negative until limit + inactive
MD12,5AB,5WS.1,5DH0,5PM,5MN,5MA-100     ;stop immediately when limit + not active,
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
                                        ;define position as 0. Move to position –
100.
```

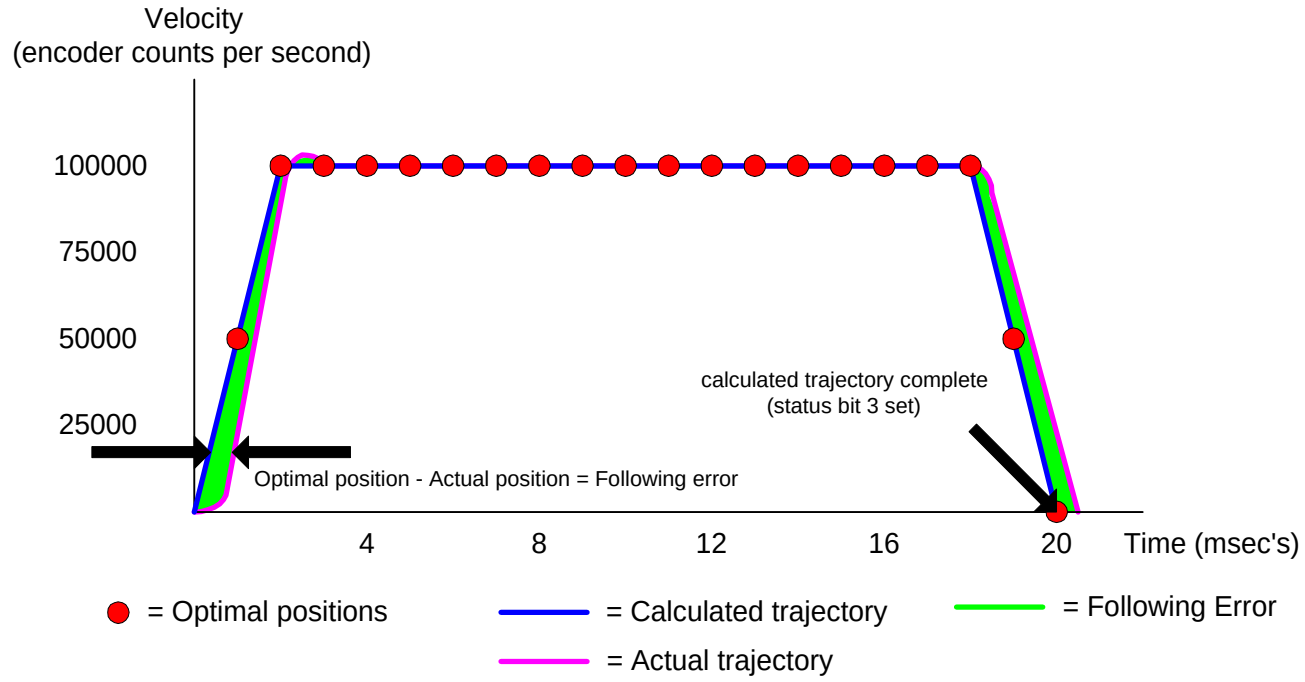

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Motion Complete Indicators

When the controller receives a move command, the Trajectory Generator calculates a velocity profile. This profile is based on:

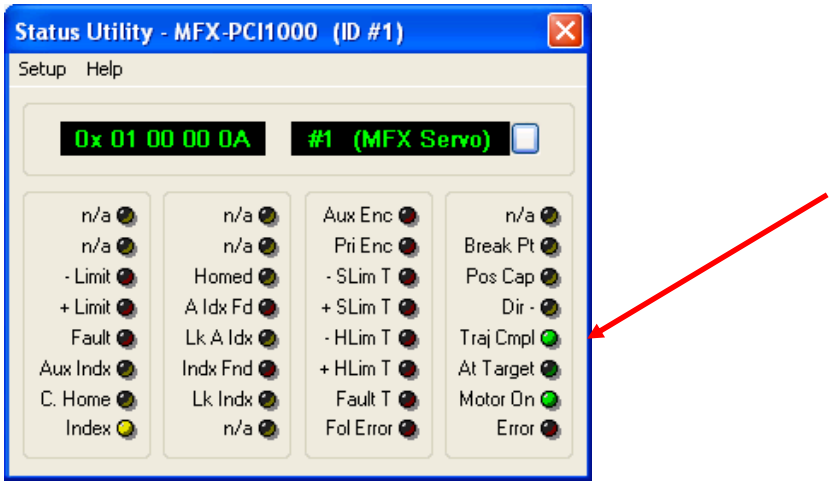
The target position (absolute or relative)
The user defined trajectory parameters (velocity, acceleration, and deceleration)
The user selected velocity profile type (trapezoidal, s-curve, parabolic)

The velocity profile, as calculated by the trajectory generator, is made up by a series of calculated 'Optimal Positions' that are evenly spaced along the motion path in increments of 1 msec's. For an analog command servo axis these 1 msec optimal positions are passed to the PID filter, which then performs a linear interpolation, calculating intermediate target points every 250 usec's.



For a **closed loop servo**, when the **calculated optimal position** of an axis is equal to the **move target**, the calculated 'digital trajectory' of the move has been completed and the **MC_STAT_TRAJ** status flag (MCCL status trajectory complete bit 3) will be set (as shown in the Status Panel graphic below). For a **closed loop stepper** axis when the **encoder position is equal to the move target**, the trajectory of the move has been completed and the **MC_STAT_TRAJ** status flag will be set. For an **open loop stepper** axis when the **step count (pulses issued) is equal to the move target**, the trajectory of the move has been completed and the **MC_STAT_TRAJ** status flag will be set.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



The **MC_STAT_TRAJ** status flag is the conditional component of the ***MCIsStopped( )*** and ***MCWaitForStop( )*** functions. As shown by the trajectory graph above, the typical lag or following error during a servo move can cause the **MC_STAT_TRAJ** flag to be set **before the axis has reached its target**. Issuing **MCIsStopped( )** with a timeout value specified or **MCWaitForStop( )** with a *Dwell* time specified allows the user to delay execution move has been completed (following error = 0). In the example below, the **MCIsStopped( )** function (with a 2 second timeout) is used to poll the axis for MC_STAT_TRAJ = true. The Windows SLEEP function is used to allow the axis to stop and settle for 100 milliseconds. command includes a Dwell of 5 msec's, allowing the axis to stop and settle.

```
MCMoveRelative( hCtlr, 2, 500.0 );      // move 500 counts
MCIsStopped( hCtlr, 1, 2.0 );
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
Sleep( 100 );
```

Another method of indicating the end of a move of a servo is to use ***MCIsAtTarget( )*** or ***MCWaitForTarget( )*** functions. To satisfy the conditions of ***MCIsAtTarget( )*** and ***MCWaitForTarget( )*** , the axis must be within the **Deadband** range (encoder counts +/- or stepper pulses +/-) for the time period specified by **DeadbandDelay**, both of which are defined within the **MCMotion** data structure.
The **MC_STAT_AT_TARGET** flag will be set when the conditions for both Deadband and Deadbanddelay have been met.

```
MCMoveRelative( hCtlr, 1, 1250.0 );     // move 1250 counts
MCWaitForTarget( hCtlr, 1, 0.005 );     // wait till MC_STAT_TRAJ set
plus                                            // msec's

MCIsAtTarget( hCtlr, 1, 2.0 );
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to reach the target within time limit (2 seconds),
    // error handling code goes here
}
Sleep( 100 );
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

On the Fly changes

During a Trapezoidal profile point to point or constant velocity move of one or more axes, the controller supports 'on the fly' changes of:

- Target
- Maximum Velocity
- Acceleration
- Deceleration
- PID parameters

Changes made to any or all of these motion settings while an axis is moving will take affect within 1 msec.



Note – Changing the PID parameters (Proportional gain, Derivative gain, Integral gain) 'on the fly' may cause the axis to jump, oscillate, or 'error out'.



S-curve or Parabolic velocity profiles:

1) Changing the target position on the fly will cause the axis to **decelerate to a stop before** proceeding to the new target
2) On the fly changes of trajectory parameters (max. velocity, accel, decel) will not be implemented until the current move has been completed



If an "on the fly" target position change requires the axis to change direction the axis will first decelerate to a stop. The axis will then move in the opposite direction to the new target. This will occur if:

1) The new target position is in the opposite direction of the current move
2) A **'near target**' is defined. A near target is a condition where the current deceleration rate will not allow the axis to stop at the new target position. In this case the axis will decelerate to a stop at the user define rate, which will result in an overshoot. The axis will then move in the opposite direction to the new target.

If an on the fly change requires the axis to change direction, the command interpreter will stall, not accepting any additional commands, until the change of direction has occurred (deceleration complete).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Feed Forward (Velocity, Acceleration, Deceleration)

Feed forward is a method in which the controller increases the command output to a servo in order to reduce the following error of an axis. Traditionally feed forward is associated with servo systems that use velocity mode amplifiers, but simple torque mode amplifiers used for high velocity and high rate of change applications can also benefit from the use of feed forward.

The basic concept of feed forward is to match the servo command voltage output of the controller to a specific velocity of axis. This essentially adds a user defined offset to the digital PID filter, resulting in more accurate motion by reducing the following error. For example:

The maximum velocity of an axis is 500,000 encoder counts per second. With a typical load applied, the user determines that a servo command voltage of 8.25V will cause the motor to rotate at 500,000 encoder counts per second. The feed forward algorithm used by the controller to generate the servo command output is:

controller output = Velocity (encoder counts/sec) **X** Feed forward term (encoder counts/volt/sec.)

with a velocity of 500,000 counts per second at a command input of 8.25V the algorithm will be:

8.25 volts = 500,000 counts/sec. **X** Feed forward term (encoder counts * volt/sec.)

Feed forward = 8.25V / 500,000 counts per sec.

0.0000165 = 10 volts / 100,000 counts per sec.



Because the controller's PID filter uses negative feedback, feed forward values are expressed as negative values.

```
// set velocity gain (velocity feed forward) using Motion Control API
function
//
   MCGetFilterConfig( hCtrlr, iAxis, &Filter );
   Filter.VelocityGain     = ( hCtlr, 1, -0.0000165 );
   MCSetFilterConfig( hCtrlr, iAxis, &Filter );


;set velocity gain (velocity feed forward) using MCCL VG command

1VG-0.0000165                          ;set velocity gain (velocity feed
                                          ;forward ) with MCCL command
```



An axis that has been tuned without feed forward will need to be **re-tuned** when the feed forward has been changed to a non zero value.

See the description of Tuning a Velocity Mode amplifier in the **Tuning the Servo** section of the **Motion Control** chapter

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

When feed forward is incorporated into the digital PID filter it becomes the primary component in generating the servo command output voltage. Typically the setting of the other terms of the filter will be:

Proportional gain – reduced by 25% to 50%
Integral gain – reduced by 5% to 25%
Derivative gain – set to zero, if the axis is too responsive reduce the gain of the amplifier

Acceleration and Deceleration Feed Forward

For most applications, velocity feed forward is sufficient for accurately positioning the axis. However for applications that require a very high rate of change, acceleration and deceleration gain must be used to reduce the following error at the beginning and end of a move.

Acceleration and deceleration feed forward values are calculated using a similar algorithm as used for velocity gain. The one difference is the velocity is expressed as encoder counts per second, while acceleration and deceleration are expressed as encoder counts per second per second.

controller output = Accel./Decel. (encoder counts/sec/sec.) * Feed forward term (encoder counts * volt/sec./sec.)



Acceleration and deceleration feed forward values should be set prior to using the Servo Tuning Utility to set the proportional and integral gain.



Acceleration feed forward and deceleration feed forward are not supported during Contour Mode (multi-axes lines and/or arcs).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Save and Restore Axis Configuration Settings

The Motion Control API Motion Dialog library includes ***MCDLG_SaveAxis( )*** and ***MCDLG_RestoreAxis( )***. These high level dialogs allow the programmer to easily maintain and update the settings for servo and stepper axes.

MCDLG_SaveAxis( ) encodes the motion controller type into a signature that is saved with the axis settings. ***MCDLG_RestoreAxis( )*** checks for a valid signature before restoring the axis settings. If you make changes to your hardware configuration (i.e. change controller type) ***MCDLG_RestoreAxis( )*** will refuse to restore those settings.

You may specify the constant **MC_ALL_AXES** for the *wAxis* parameter in order to save the parameters for all axes installed on a motion controller with a single call to this function.

If a NULL pointer or a pointer to a zero length string is passed as the *PrivateIniFile* argument the default file (mcapi.ini) will be used. Most applications should use the default file so that configuration data may be easily shared among applications. Acceptance of a pointer to a zero length string was included to support programming languages that have difficulty with NULL pointers (e.g. Visual Basic).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Chapter 6

Application Solutions

Backlash Compensation

In applications where the mechanical system isn't directly connected to the motor, it may be required that the motor move an extra amount to compensate for system backlash. When backlash compensation is enabled, the controller will offset the target position of a move by the user defined backlash distance. This feature is only available for Analog Command Servo axes.

The function ***MCEnableBacklash( )*** is used to initiate backlash compensation. The *Backlash* parameter of this function sets the amount of compensation and should be equal to one half of the amount the axis must move to take up the backlash when it changes direction. The units for this command parameter are encoder counts, or the units established by the ***MCSetScale( )*** command for this axis.

When this feature is enabled, the controller will add or subtract the backlash distance from the motor's commanded position during all subsequent moves. If the motor moves in a positive direction, the distance will be added; if the motor moves in a negative direction, it will be subtracted. When the motor finishes a move, it will remain in the compensated position until the next move.

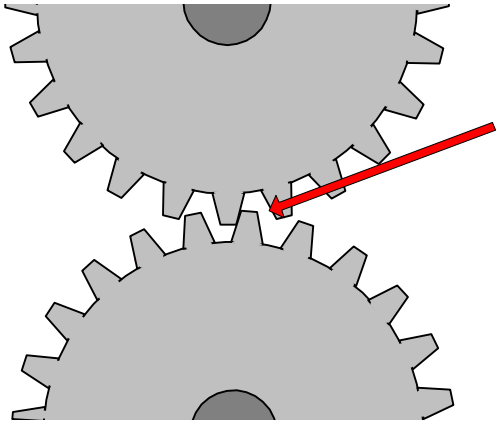
Prior to enabling backlash compensation, the motor should be positioned halfway between the two positions where it makes contact with the mechanical gearing. This will allow the controller to take up the backlash when the first move in either direction is made, without "bumping" the mechanical position.

While backlash compensation is enabled, the response to the ***MCGetPosition( )***, ***MCTellTarget( )*** and ***MCTellOptimal( )*** commands will be adjusted to reflect the ideal positions as if no mechanical backlash was present.

For the example below assume that the system has 200 encoder counts of backlash. This example moves the system to the middle of the backlash range and enables compensation. Note that the compensation value (in encoder counts) used with ***MCEnableBacklash()*** is half of the total amount of backlash.

```
MCMoveRelative( hCtlr, 1, -100.0 );        // move to middle of backlash
MCWaitForStop( hCtlr, 1 );                 // let motion finish
MCEnableBacklash( hCtlr, 1, 100.0, TRUE ); // enable backlash
compensation
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Emergency Stop

Many applications that use motion control systems must accommodate regulatory requirements for immediate shut down due to emergency situations. Typically these requirements do not allow an emergency shut down to be controlled by a programmable computing device. The drawing below depicts an application where an emergency stop must be a completely 'hard wired' event.

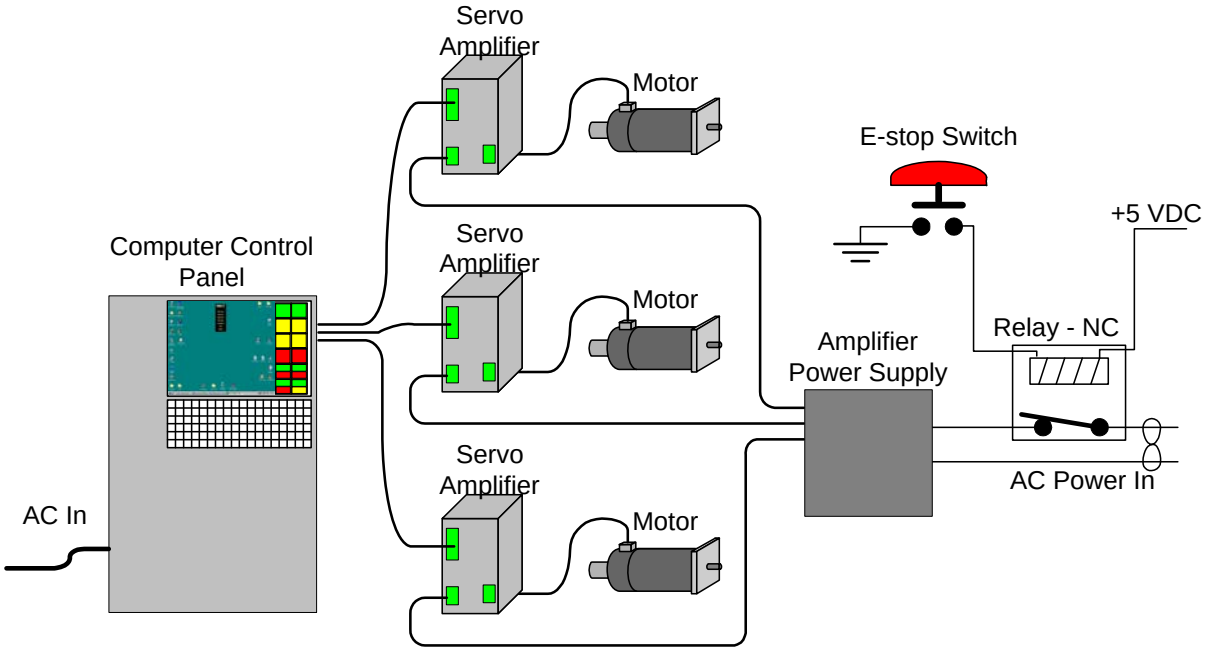


Figure 39. Typical 'hard wired' E-stop

This 'hard wired' E-stop circuit uses a relay to disconnect power from the servo amplifiers. The motors and amplifiers would certainly be disabled, but the motion controller and the application program will have no indication that an error condition exists.

E-stop switch connected to Amplifier Fault servo module input

The Amplifier / Driver Fault inputs can be used to disable motion with no user software action required. The E-stop switch is wired to the Amplifier/Drive Fault input of **each axis being used**. Auto shut down of motion upon activation of the E-stop switch is enabled by the **MCMotion** structure member **EnableAmpFault**. When the E-stop switch is activated:

1) The axis is disabled (PID loop terminated, Amplifier Enable / Driver Disable output turned off)
2) The status flag **MC_STAT_AMP_FAULT** will be set for each axis
3) The status flag **MC_STAT_ERROR** will be set for each axis

When the E-stop condition has been cleared, motion can be resumed after issuing the ***MCEnableAxis*** function with the parameter *wAxis* set to **MC_ALL_AXES**.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

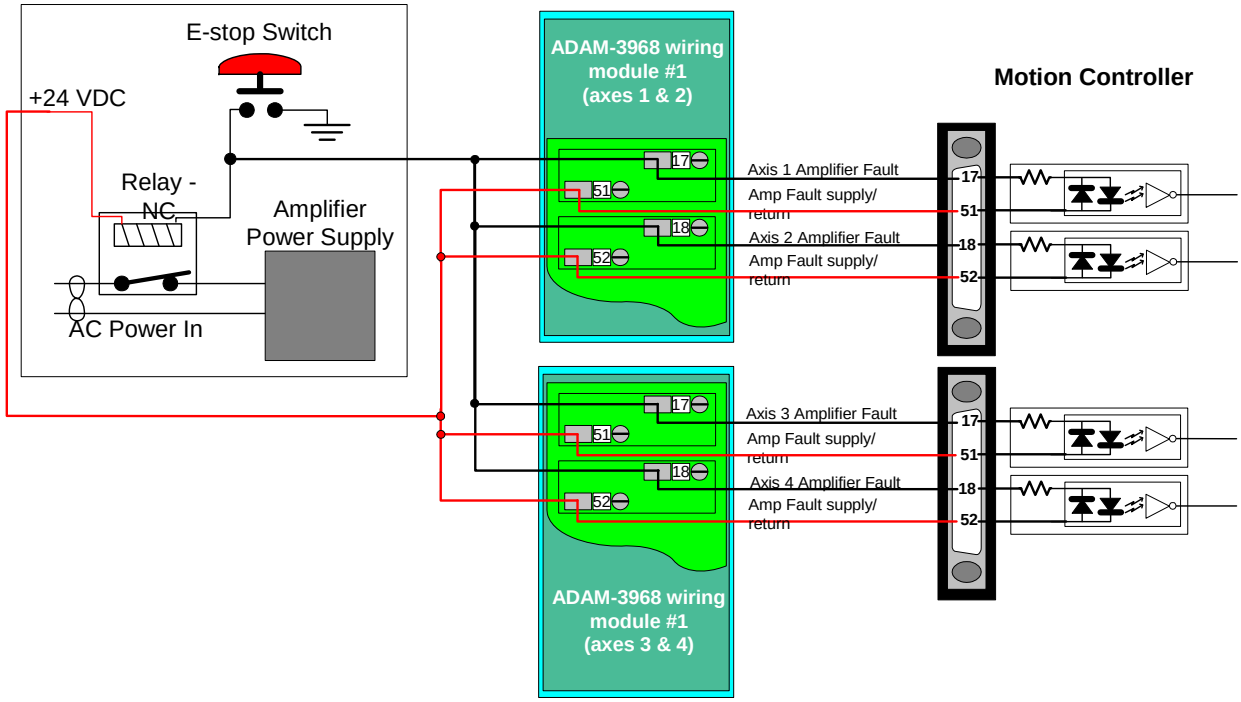


Figure 40. E-stop switch wired to the Amplifier / Driver Fault inputs

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Encoder Rollover

The controller provides 32 bit position resolution, resulting in a position range of –2,147,483,647 to 2,147,483,647. For an application where the axis is moving at maximum velocity (20 million encoder counts per second), the encoder would rollover in approximately 1.6 minutes. When the encoder rolls over, the reported position of the axis will change from a positive to a negative value. For example, if the axis is at position 2,147,483,647 the next positive encoder count will cause the controller to report the position as –2,147,483,647.

If a user scaling other than 1:1 has been defined the controller will report the position in user units. The reported position at which the value will rollover is based on the user scaling. If user scaling is set to 10,000 encoder counts to one position unit, the reported position will rollover at position 214,748.3647. The next positive encoder count will cause the controller to report the position as –214,748.3647.

Encoder rollover during Position Mode moves
The controller does not support executing Position Mode moves when the encoder rolls over. No matter what the commanded position, the axis will stop at the rollover position (2,147,483,647 or –214,748.3647).

Encoder rollover during Velocity Mode moves
No disruption or unexpected motion will occur if a rollover occurs during a Velocity mode (***MCSetOperatingMode***, **MC_MODE_VELOCITY**) move.



Prior to executing a velocity mode move in which the encoder position may rollover the axis **must** be homed (MCFindIndex or MCSetPosition) to position 0. Defining a offset to the home position will cause the axis to pause at the rollover point.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Flash Memory Firmware Update

PMC's **Flash Wizard** is a windows utility that allows the user to easily update the controller's firmware code. Firmware updates for most PMC motion controllers are available for download from from the Support section of PMC's web site *www.pmccorp.com/support/support.php*. Note: Please contact PMC if firmware is not available for your model.



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Saving and Restoring Axis Configuration Settings

Users of PMC motion controllers can:

- Save desired motion controller and axis configuration settings to a text file
- Download the saved settings from a text file to any installed controller
- Copy the saved settings from one PC to another for use by multiple controllers
- View the saved settings by opening a saved text file

Whenever the motion controller is reset or powered up, all motion settings (velocity, acceleration, deceleration, limits, PID values, etc) and all global controller settings (user scaling and I/O configuration) revert to factory default values (factory default values are listed on page 165). Therefore, after power-up, you should always initialize all controller and axis settings to their desired values. You can accomplish this in any one of three ways:

1. Use the Setup Menu selections "**save all axis settings**" and "**initialize (restore) axis settings**" in any of PMC's application programs.

2. Use the Motion Control Dialog ***MCDLG_SaveAxis()*** and ***MCDLG_RestoreAxis()*** functions.

3. Save and initialize each parameter individually from a high-level program or an MCCL command text file.



On power-up or reset, all motion controller settings revert to factory default values. Therefore, after power-up, users should always initialize all controller settings to their desired values.

Saving and restoring configuration settings using PMC application programs

For initial testing and system configuration, users will probably find it most convenient to save and restore (initialize) axis settings via the menu selections in PMC's application programs.

The first time that the Motion Control API recognizes that one or more PMC motion controllers are installed, a (**mcapi.ini**) text file is created in the C:\Windows\ folder. Initially mcapi.ini contains only information about the controller type and interface settings. When users select "Save All Axis Settings" in the application program's setup menu, the settings currently being used by the motion controller are saved to the mcapi.ini file.

The available menu choices are:

a. "**Save All Axis Settings**" - Copies the settings used by the controller to mcapi.ini
b. "**Initialize All Axes**" - Copies the axis settings saved in mcapi.ini to the controller
c. "**Always (or Auto) Initialize All Axes**" - Every time that the program starts, the axis settings will automatically be copied from mcapi.ini to the controller

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

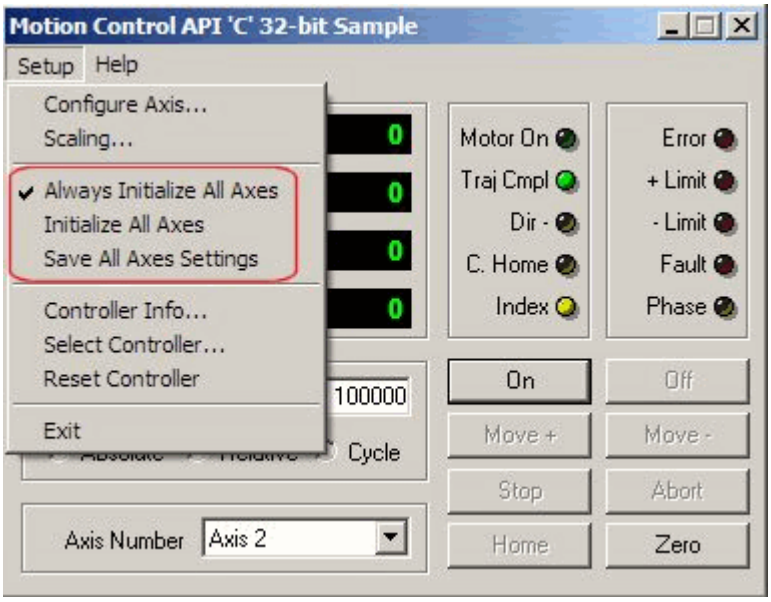


Figure 41. Saving and Restoring Settings From a PMC Sample Program

Example: PMC's Motion Integrator provides first-time users with a step by step process that helps them install, configure and troubleshoot their motion controller. Like the provided sample programs, Motion Integrator also allows users to conveniently save all settings to C:\Windows\mcapi.ini. Once the desired settings are saved, users can direct other PMC application programs to load and use the saved setting by selecting **Auto Initialize** from each program's **File** menu.

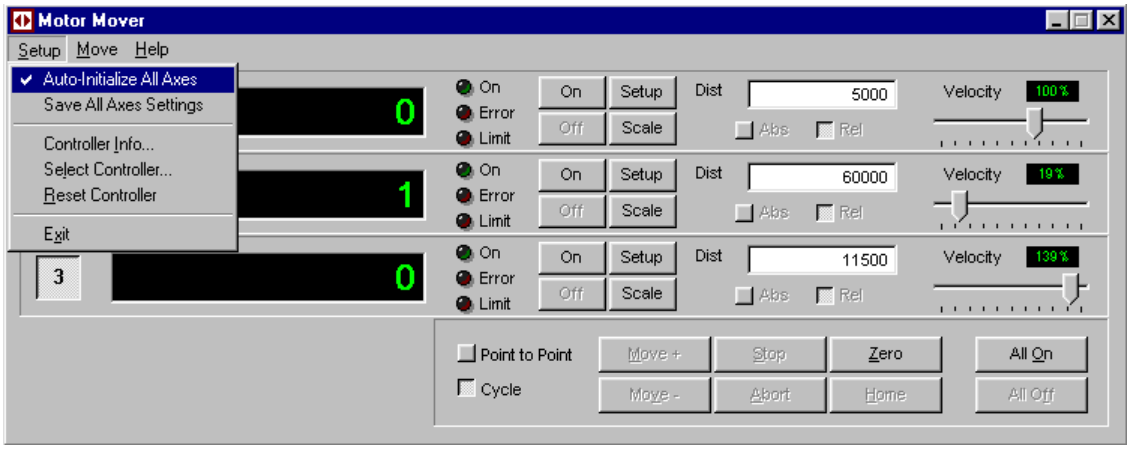


Figure 42. Use Previously Saved Settings by Checking the "Auto Initialize" Menu Selection



Selecting **Save All Axes** from the File menu of a PMC application program will **overwrite all previously stored settings**.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Saving and restoring configuration settings using the MCDLG functions

Programmers can programmatically save axis settings to the mcapi.ini file (or any other text file) and restore (initialize) the settings from the saved file back to to the motion controller using the MCDLG (Motion Control Dialog) functions ***MCDLG_SaveAxis()*** and ***MCDLG_RestoreAxis()***. These functions are convenient because they save all controller and axis settings at once. These are the same functions invoked by the Setup menu selections of PMC's sample programs. These functions are documented in more detail in the **MCDLG Reference** online help, accessible from the Motion Control API program group, as shown in the following figure.

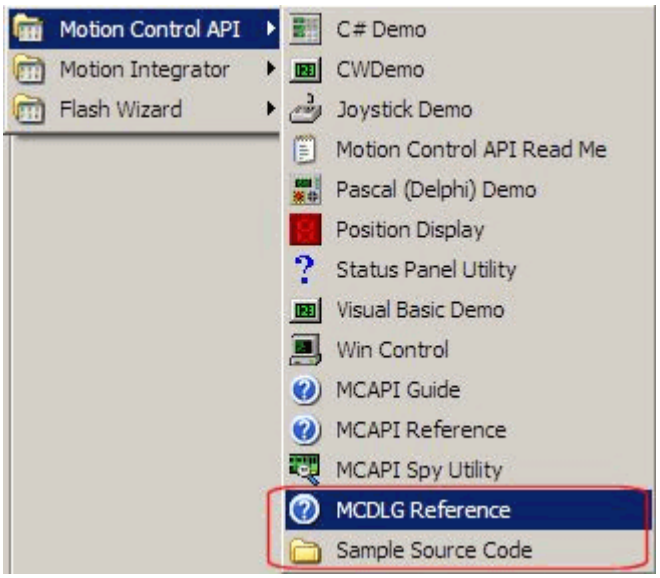


Figure 43. MC Dialog Box Reference and Sample Source Code Program Groups

The default name for the initialization file used to save and restore controller and axis settings is **mcapi.ini**. However users can programmatically save the axis settings to any filename, with a different filename extension if they wish.

As a helpful guide for programmers, PMC's Motion Control API also includes a **Sample Source Code** folder which contains the source code for all of PMC's sample programs. For example, the following source code performs the saving and restoring functions for the "Save All Axis Settings" and "Initialize All Axes" menu selections in the sample application programs.

```
case IDM_AUTO_INIT:
    SkipTimer = true;
    if (GetMenuState(GetMenu(hDlg), IDM_AUTO_INIT, MF_BYCOMMAND) & MF_CHECKED)
        CheckMenuItem(GetMenu(hDlg), IDM_AUTO_INIT, MF_UNCHECKED);
    else
    {
        CheckMenuItem(GetMenu(hDlg), IDM_AUTO_INIT, MF_CHECKED);
        MCDLG_RestoreAxis(hCtlr, MC_ALL_AXES, MCDLG_PROMPT |
MCDLG_CHECKACTIVE, NULL);
    }
    Cycle = SkipTimer = false;
    break;
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
case IDM_INIT:
    SkipTimer = true;
    MCDLG_RestoreAxis(hCtlr, MC_ALL_AXES, MCDLG_PROMPT | MCDLG_CHECKACTIVE,
NULL);
    Cycle = SkipTimer = false;
    break;

case IDM_SAVE_SETTINGS:
    SkipTimer = true;
    MCDLG_SaveAxis(hCtlr, MC_ALL_AXES, 0, NULL);
    SkipTimer = false;
    break;
```

Saving and restoring configuration settings via individual function or MCCL calls

In addition to the MCDLG functions, programmers can selectively save and restore one, some, or all settings individually according to their needs. Users can do this either by Motion Control API function calls or by sending one or more MCCL commands to the motion controller. For example, after boot-up, users can send the motion controller a text file containing the MCCL commands required to initialize the desired configuration settings. See the Motion Control API Reference Manual and Motion Control Command Language Reference Manual for more specific information about programming the controller.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Learning/Teaching Points

As many as 256 points can be stored for ***each axis*** in the controller's point memory by using the **MCLearnPoint( )** function. A stored point can be either the actual position of an axis (**MC_LRN_POSITION**) or the target position of an axis (**MC_LRN_TARGET**).

The value **MC_LRN_POINT** would typically be used in conjunction with jogging. The operator would jog the axes along the desired path, issuing the ***MCLearnPoint( )*** command at regular intervals. The ***MCMovePoint( )*** command would then be used to 'play back' the path traversed by the operator.

For applications where the target point data was previously recorded and stored in the PC, the value **MC_LRN_TARGET** would be used to load the target points into the controller. For some applications, using MCLearnPoint( ) to load a series of moves may be 'easier' than issuing a series of contour mode linear moves, even though the results would be the same.

Once all points have been stored, the axes are commanded to move to the stored positions with ***MCMoveToPosition( )***. The parameter *wIndex* indicates to which stored point the axis should move.

```
// Move axis 1 and store position in consecutive point storage locations.

WORD wIndex;
MCEnableAxis( hCtlr, 1, TRUE );         // motor on
MCGoHome( hCtlr, 1 );                   // start from absolute zero
MCWaitForStop( hCtlr, 1, 0.100 );

for (wIndex = 0; wIndex < 5; wIndex++) {
   MCMoveRelative( hCtlr, 1, 1234.0 );  // move
   MCWaitForStop( hCtlr, 1, 0.100 );    // are we there yet?
   MCLearnPoint( hCtlr, 1, wIndex, MC_LRN_POSITION );
}


// Store several positions for axis 4 without actually moving the axis. Note //
that axis is disabled with MCEnableAxis( ) prior to storing positions

WORD wIndex;
MCEnableAxis( hCtlr, 4, FALSE );        // motor off
for (wIndex = 0; wIndex < 5; wIndex++) {
   MCMoveRelative( hCtlr, 4, 2468.0 );  // nothing actually moves
   MCLearnTarget( hCtlr, 4, wIndex, MC_LRN_TARGET );
}


// This example moves to the stored positions, dwelling for 0.2 seconds at
// each point.

WORD wIndex;
MCEnableAxis( hCtlr, 4 );               // enable axis
for (wIndex = 0; wIndex < 5; wIndex++) {
   MCMoveToPoint( hCtlr, 4, wIndex );   // move to next point
   MCWaitForStopped( hCtlr, 4, 0.2 );
}
```

To cause the controller to perform linear interpolated moves between the taught points, place each of the axes in contour mode. Use the lowest axis number as the contour mode command parameters, this is the controlling axis. Set the vector velocity and accelerations of the controlling axis. Issue a single ***MCMoveToPoint( )*** command to the controlling axis with the point numbers as the command parameter. Note that when point memory is used with motors in contour mode, point 0 should not be used. This

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

example executes linearly interpolated moves through three stored points of axes 1, 2, and 3.

```
MCSetOperatingMode( hCtlr, 1, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 2, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 3, 1, MC_MODE_CONTOUR );

//  Linear interpolated move sequence through stored points

for (wIndex = 1; wIndex < 4; wIndex++) {
   MCBlockBegin( hCtlr, MC_BLOCK_CONTR_LIN, 1 );
      MCMoveToPoint( hCtlr, 1, wIndex );
      MCMoveToPoint( hCtlr, 1, wIndex );
      MCMoveToPoint( hCtlr, 1, wIndex );
   MCBlockEnd( hCtlr, NULL );
}
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Building MCCL Macro Sequences

A powerful feature is the ability to define MCCL (Motion Control Command Language) command sequences as macros.



For additional information on macro's and MCCL (Motion Control Command Language) commands please refer to the **MCCL Reference Manual**.

A macro is a user define sequence of operations that is executed by issuing a single command. For example:

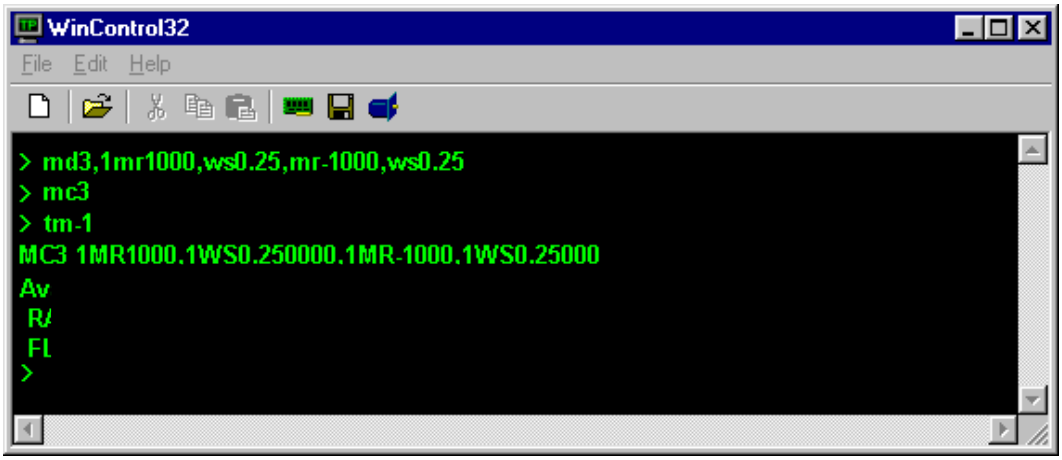
```
1MR1000,WS0.25,MR-1000,WS0.25
```

will cause the motor attached to axis 1 to move 1000 counts in the positive direction, wait one quarter second after it has reached the destination, then move back to the original position followed by a similar delay. If this sequence were to represent a frequently desired motion for the system, it could be defined as a macro command. This is done by inserting a Macro Define (**MD*n***) command as the first command in the command string. For example:

```
MD3,1MR1000,WS0.25,MR-1000,WS0.25
```

will define macro #3. Whenever it is desired to perform this motion sequence, issue the command Macro Call (MC3).

To command the controller to display the contents of a macro, issue the **T**ell **M**acro (**TM*n***) command with parameter 'n' = the number of the macro to be displayed. To display the contents of all stored macro's issue the Tell macro command with parameter 'n' = -1.





Once a macro operation has begun, the host will not be able to communicate with the controller until the **macro has completed execution**. For information on communicating with the controller while executing macro's please refer to the section titled **MCCL Multi-Tasking**.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The controller can store up to 1024 user defined macros. Each macro can include as many as 255 bytes, resulting in a total macro capacity of 255K bytes. Depending on the type of command and type of parameter, a command can range from 2 bytes (a command with no parameter) to 10 bytes (a command with a 64 bit floating point parameter).



If the amount of available macro memory exceeds 255K bytes the controller will respond with error code - 18

All memory on the controller is volatile, which means that the data in memory will be cleared when the controller is reset or power to the board is turned off. The **R**eset **M**acro (**RM*n***) command is used to erase macros.

To terminate the execution of any macro that was started from WinControl press the escape key.
To start a macro that runs indefinitely without 'locking up' communication with the host, start the macro's with the ***generate a Background task*** (GT) command instead of the ***Call macro command*** (MC). This will allow the operation execute as a background task. Please refer to the next section **Multi-Tasking**.



The controller supports single-stepping of any MCCL macro command executing as the foreground task. For additional information please refer to **Single Stepping MCCL Programs** later in this chapter.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

MCCL Multi-Tasking

The controller's command interpreter is designed to accept commands from the user and execute them immediately. With the addition of sequencing commands, the user is able to create sophisticated command sequences that run continuously, performing repetitive monitoring and control tasks. The drawback of running a continuous command sequence is that the command interpreter is not able to accept other commands from the user.



Once a macro operation has begun, the host will not be able to communicate with the controller until the **macro has completed execution**.

The controller supports Multi-tasking, which allows the controller to execute continuous monitoring or control sequences as background tasks while the foreground task communicates with the 'host'.

With the exception of ***reporting commands*** (Tell Position, Tell Status, etc...), which are not compatible with Multi-Tasking, any MCCL commands, can be executed in a background task. Prior to executing a command sequence/macro as a background task, the ***user should always test the macro by first executing it as a foreground task***. When the user is satisfied with the operation of the macro, it can be run as a background task by issuing the **G**enerate **T**ask (**GT*n***) command, specifying the macro number as the command parameter. After the execution of the Generate Task command, the accumulator (register 0) will contain an identifier for the background task. Within a few milliseconds, the controller will begin running the macro as a background task in parallel with the foreground command interpreter. The controller will then be free to accept new commands from the user.

```
;Multitasking example – while axis #1 is moving, monitor the state of
digital
;input #4. When the input goes active, stop axis #1 and terminate the
;background task


AL0,AR10                          ;define user register 10 as input #4
active
                                  ;flag register
AL0,AR100                         ;define user register #100 as
background task
                                  ;ID register


MD100,IN4,MJ101,NO,1JR-3          ;jump to macro 101 when digital input
#4
                                  ;turns on
MD101,1ST,1WS.05,AL1,AR10,ET@100  ;stop axis #1. Terminate background
task

GT100,AR@100,1VM,1DI0,1GO         ;spawn macro #10 as background task.
Store
                                  ;task ID into register #100. Start axis
#1
                                  ;moving in velocity mode,
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Note: Immediately after 'spawning' the background task (with the GT*n* command), the value in the accumulator (task identifier) should be stored in a user register. This value will be required to terminate execution of the background task.

Another way to create a background task is to place the Generate Task command as the first command in a command line, using a parameter of 0. This instructs the command interpreter to take all the commands that follow the Generate Task command and cause them to run as a background task. The commands will run identically to commands placed in a macro and generated as a task.

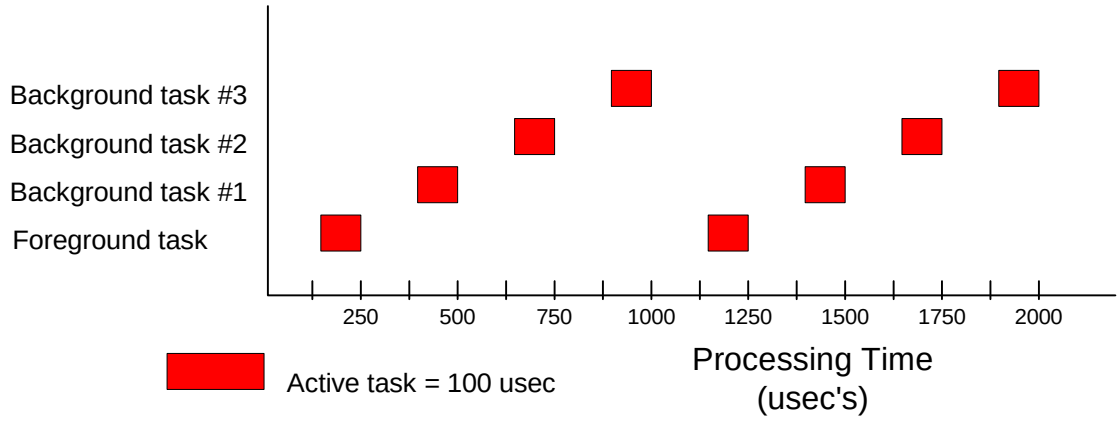
```
;Multitasking example – while axis #1 is moving, monitor the state of the
;motor error status bit (bit 7). If error occurs set bit #1 of user
;register 200

GT0,AR@100,LU"STATUS",1RL@0,IC0,JR-3,NO,AL1,AR200,ET@100
                                    ;loop on axis #1 status bit 0, if set;
set
                                    ;bit #1 of register 200, terminate task
using
                                    ;Task ID (in register #100)
```

Within the background task, the commands can move motors, wait for events, or perform operations on the registers, totally independent of any commands issued in the foreground. However, the user must be careful that they do not conflict with each other. For example, if a background task issues a move command to cause a motor to move to absolute position +1000, and the user issues a command at the same time to move the motor to -1000, it is unpredictable whether the motor will go to plus or minus 1000.

In order to prevent conflicts over the registers, the background task has its own set of registers 0 through 9 (register 0 is the accumulator). These are private to the background task and are referred to as its 'local' registers. The balance of the registers, 10 through 255, are shared by the background task and foreground command interpreter, they are referred to as 'global' registers. If the user wishes to pass information to or from the background task, this can be done by placing values in the global register. Note that when a task is created, an identifier for the task is stored in register 0 of both the parent and child tasks.

When one or more background tasks are active the Task Handler will begin issuing local interrupts every 250 microseconds. Each time the task handler interrupt is asserted, the controller will switch from executing one task to the next every 250 micro seconds. For example if three background tasks are active, plus the foreground task (always active), each of the four tasks will receive approximately 100 micro seconds of processor time every 1 millisecond.



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

While a background task executes a **Wait** command, that task no longer receives any processor time. For tasks that perform monitoring functions in an endless loop, the command throughput of the controller can be improved by executing a **Wait** command at the end of the loop until the task needs to run again.

A common way for a background task to be terminated, is when the command sequence of the task finishes execution. This will occur at the end of the macro or if a **B**rea**K** (**BK**) command is executed. When a task is terminated, the resources it required are made available to run other background tasks.

```
;Multitasking example – this background task will terminate itself if the
;motor error status bit for axis #1 is set. This sequence is similar to
the ;previous example except that the task is self terminating, so
register #100
is not required.

GT0,LU"STATUS",1RL@0,IC0,JR-3,NO,AL1,AR200,BK
                                ;loop on axis #1 status bit 0, if set;
set
                                ;bit #1 of register 200, task self
terminates
                                ;(no commands left to execute)
```

Alternatively, the **E**scape **T**ask (**Te*n***) command can be used to force a background task to terminate. When a task is generated by the **GT** command, a value known as the **Task ID** is placed into the accumulator. This value should immediately be copied into a user register. The parameter to this command must be the value that was placed in accumulator (register 0) of the parent task, when the Generate Task command was issued.

```
;Multitasking example – Terminating a background task with the Escape
Task command.

GT100,AR@150                    ;call macro #100 as a background task,
copy
                                ; task ID into user register 150

ET@150                          ;to terminate background task issue
escape
                                ; task command with parameter n = Task
ID
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Position Capture

The controller features versatile high-speed position capture circuits that allow users to precisely synchronize motion with external events. The controller supports capturing the position of a closed loop servo or stepper encoder or step count (of an open loop stepper) on the **rising edge** of a TTL Position Capture input. As many as 1024 captured positions can be stored in the recording memory for each axis. The maximum frequency of position captures is 1 KHz. The maximum latency between the rising edge of the position capture input and the loading of the captured position is 100 nano seconds.



The active level of a position capture input is fixed as a TTL high. Unlike the general purpose digital I/O channels, a position capture input cannot be configured for 'low true' operation.

The Motion Control API function ***MCEnableCapture ( )*** is used to initiate position capture. When this feature is enabled the current position will be recorded on the rising edge of the capture input. If parameter *count* equals 1 the module will capture only one position. If parameter *count* equals 2 the module will capture two positions, and so on. When the number of positions captured = *count* , the **MC_STAT_POS_CAPT** flag (status bit 5) will be set. To report the number of positions captured issue the ***MCGetCount ( )*** function with the *type* = **MC_COUNT_CAPTURE**. To disable position capture issue ***MCEnableCapture ( )*** with parameter *count* equal to 0. Captured positions may be retrieved using the ***MCGetCapturedData( )*** function.

```
Long int count;
double    data{10};

MCEnableAxis( hCtlr, 1, 1 );
MCMoveRelative( hCtlr, 1, 10000.0 );


// Capture 10 positions
//
MCEnableCapture( hCtlr, 1, 10.0 );


//  Retrieve the 10 captured positions into local array
//

do {
   MCGetCount(( hCtlr, 1, MC_COUNT CAPTURE, &count);
}  while (count <10);

MCGetCaptureData( hCtlr, 1, MC_CAPTURE_ACTUAL, 0, 10, &data );
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Position Compare

The controller features versatile high-speed position compare circuits that allow users to precisely synchronize external events with encoder position. The position compare circuits assert a TTL logic output when one or more pre-defined encoder positions are reached. The controller provides two high speed TTL outputs to signal that an encoder position has been reached (a “compare” event has occurred). The assertion of this output is based on the position of the encoder of a closed loop servo or stepper axis or the step count register of an open loop stepper axis. As many as 1024 compare positions can be stored in the recording memory.

Compare predefined positions

To configure an axis for position compare, first use the Motion Control API function ***McConfigureCompare ( )*** to define the number of compare positions (as many as 1024) and the compare output mode. Then issue the Motion Control API function ***MCEnableCompare ( )*** with the *flag* = **MC_COMPARE_ENABLE**. This will terminate any current compare operation and initializes the compare index to 0. After starting a move, when the actual position is equal to the compare position the compare output will be turned on (TTL high by default) and the next compare position will be loaded into the compare register. When all position compare events have been completed the **MC_STAT_BREAKPOINT** flag of the axis status will be set.

Compare at incremental distances

For compare events at fixed distances of travel use the function ***MCEnableCompare ( )*** and:

1) Store the beginning point (first compare position) in the first location of *values*
2) Set the *num* parameter to 1
3) Set the *inc* parameter to the distance (counts or steps) between compare events

Compare frequency and output latency

The compare position update frequency is based on the trajectory generator, which is executes at a rate determined by the trajectory generator rate setting; (low (the default) = 1 mSec (1 kHz), medium = 500 mSec (2 kHz), high = 250 mSec (4 kHz). Therefore the distance between compare positions cannot be such that the time from one compare event to the next is less than the position update frequency. The maximum latency between the an axis reaching the compare position and the activation of the compare output is 100 nano seconds.

Compare output signal configuration

When the compare output is activated as the result of a compare or breakpoint occurrence, the compare output signal will react according to the which mode has been selected with the *mode* parameter of the ***MCConfigureCompare ( )*** function.

Mode	Description
MC_COMPARE_DISABLE	Disables the compare output
MC_COMPARE_INVERT	Inverts the active level of the compare output

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

MC_COMPARE_ONESHOT	Configures the compare output for one shot operation (one shot period is defined by the *period* parameter of ***McConfigureCompare*** ( ) function. The one shot pulse period range is from 1millisecond to 1.0 second. For one shot periods less than 50 milliseconds the timer resolution is 1 millisecond. For one shot periods greater than 50 milliseconds the timer resolution is 50 milliseconds.
MC_COMPARE_STATIC	Configures the compare output to turn on when a compare event occurs. The output will stay on until a new compare event is called
MC_COMPARE_TOGGLE	Configures the compare output to toggle between the active and inactive state each time a compare event occurs

For all of the output modes, the compare output will be activated within 100 nano seconds of the encoder/step count reaching the compare position. When the compare output mode is set to Disabled, the output will be at its' in-active level (TTL low). The controller sets the output mode to Disabled on power up or reset.

To report the number of compare events that have occurred issue the ***MCGetCount ( )*** function with the *type* = **MC_COUNT_COMPARE**. To disable position compare issue ***MCEnableCompare ( )*** with parameter *flag* value = **MC_COMPARE_DISABLE**.

```
//
//  Use positions spaced 5 units apart, beginning at 10.0 as compare
//  positions. Toggle the output pin on valid compares. Wait for 20
//  compares to complete.
//
data[0] = 10.0;     // starting point
MCConfigureCompare( hCtlr, 1, data, 1, 5.0, MC_COMPARE_TOGGLE, 0.0 );

MCEnableCompare( hCtlr, 1, MC_ENABLE_COMPARE );   // enable compare
MCMoveRelative( hCtlr, 1, 100.0 );

do {      // wait for 5 points
    MCGetCount( hCtlr, 1, MC_COUNT_COMPARE, &count );
} while (count < 20);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Position Verification of an Open Loop Pulse Axis

Historically stepper motors have been used as a reduced cost alternative to servos for applications that do not require the accuracy, repeatability, and acceleration of a closed loop servo. One of the necessary 'evils' of the tradeoff of selecting a stepper motor over a servo is the tendency of steppers to 'lose steps' due to motor / load resonance. By adding an encoder (typically directly coupled to the stepper motor shaft) the user can monitor the final position of the stepper and issue 'correction moves' to compensate for the lost steps.

One of the available controller options is the support of encoder feedback for Pulse Command axes. A Pulse Command axis can be configured to use an encoder in one of two different ways:

Closed Loop Mode
Open Loop with Position Verification

For a detailed description of Closed Loop Operation of a Pulse Command axis please refer to page 80. In order to differentiate between an encoder used for Closed Loop operation of a Analog or Pulse Command axis and an encoder used for position verification of a Pulse Command axis, the encoder of an Open Loop Pulse Command axis will be referred to as an Auxiliary Encoder. The advantages of an open loop stepper with auxiliary encoder versus a closed loop axis are:

The pulse train of an open loop stepper 'at velocity' is much more stable
Easier to configure - open loop systems require no tuning
Lower cost

Position verification example
Typically an encoder is added to an open loop stepper to allow the user to retrieve the encoder position **at the end of a move**. The reported position of the auxiliary encoder is used to determine whether or not the axis is properly positioned.

```
// After a move compare the target and auxiliary encoder position.
// If short of the target, execute a move = the difference of the target &
// encoder position

MCMoveAbsolute( hCtlr, 1, 122.5 );
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
if (MCGetTargetEx( hCtlr, 2, &Target ) == MCERR_NOERROR )
if (MCGetAuxEncPosEx( hCtlr, 1, &Position ) == MCERR_NOERROR)
   if (Position < 122.0)
      (Target – Position = AuxEncDiff)
      MCMoveRelative( hCtlr, 1, AuxEncDiff );
      if (!MCIsStopped( hCtlr, 1, 2.0 )) {
         // Motor failed to stop within time limit (2 seconds),
         // error handling code goes here
      }
if (MCGetAuxEncPosEx( hCtlr, 1, &Position ) == MCERR_NOERROR)
   if (Position < 122.0)
            . . .  // print error message
```

Homing the auxiliary encoder of an open loop stepper

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The encoder of an open loop stepper may be homed in one of two ways:

Home the encoder using the Auxiliary Encoder Index input
Re-define the position of the encoder when the axis is homed



If no encoder index mark output is available, the position of the auxiliary encoder can be redefined at anytime using the Motion Control API function **MCSetAuxEncPos( )**.

If the encoder includes an index mark output it is recommended that this signal be used to home **both the reported position of the axis and the auxiliary encoder**. The repeatability of a system homed using the index mark will be significantly better than that of a system that uses a mechanical switch/electromechanical sensor. The following programming example will reference both the reported position of an open loop stepper and the auxiliary encoder at the location of the Index mark:



The following C example uses the ***MCFindAuxEncIdx( )*** and ***MCSetPosition ( )*** functions to redefine the encoder position register and the step count register of an open loop stepper with an auxiliary encoder. For complete C code homing samples that can be cut and pasted into an application program please refer to the Motion Control API on-line help (mcapi.hlp).

```
MCFindAuxEncIdx( hCtlr, 5, 0.0 );
dwStatus = MCGetStatus( hCtlr, 5 );
while (! MCDecodeStatus( hCtlr, dwStatus, MC_STAT_AUX_IDX_FND ))
       dwStatus = MCGetStatus( hCtlr, 5 );
IdxPosition = MCGetPosition( hCtlr, 5 );
MCStop( hCtlr, 5 );
if (!MCIsStopped( hCtlr, 5, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
Position = MCGetPosition( hCtlr, 5 );
HomePosn = (Position - IdxPosition) * -1
MCSetOperatingMode( hCtlr, 5, 0, MC_MODE_POSITION );
MCEnableAxis( hCtlr, 5, TRUE );
MCMoveRelative( hCtlr, 5, HomePosn );
if (!MCIsStopped( hCtlr, 5, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
MCSetPosition( hCtlr, 5, 0.0 );
```



After issuing the ***MCFindAuxEncIdx( )*** function the reported position of the encoder of an open loop stepper will be redefined to equal parameter *position* once the index has been captured.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
;MCCL example - define positions at auxiliary encoder index mark
MD1,5MN,5VM,5DI0,5GO,WA.1,MJ10          ;start velocity mode move
MD10,5AF0,WA0.1,LU"STATUS",5RL@0,IS20,MJ11,NO,JR-5
                                        ;Enable aux. encoder index mark
                                        ;capture, loop until Aux. Index
                                        ;Found = True
MD11,LU"POSITION",5RD@0,AR100,5ST,5WS.1,LU"POSITION",5RD@0,AR101,5PM,5MN,
MJ12                                       ;load accumulator with step
count
                                        ;position at location of index
mark.
                                        ;Stop the move. Load current
position,
                                        ;enable position mode
AL@101,AS@100,AM-1,5MR@0,5WS.1,5DH0     ;calculate step count distance to
                                        ;index, move to index, define
step
                                        ;count to 0
```



For MCCL homing samples that can be downloaded to the controller and executed please refer to PMC's Motion CD.

Verifying the Operation of the encoder of an open loop stepper

To verify the operation of the encoder of an open loop stepper use either **WinControl** or **Motor Mover**. From **WinControl**, issuing the **A**uxiliary encoder **T**ell position (***a*AT**) command will cause the current position of the open loop steppers encoder to be reported.

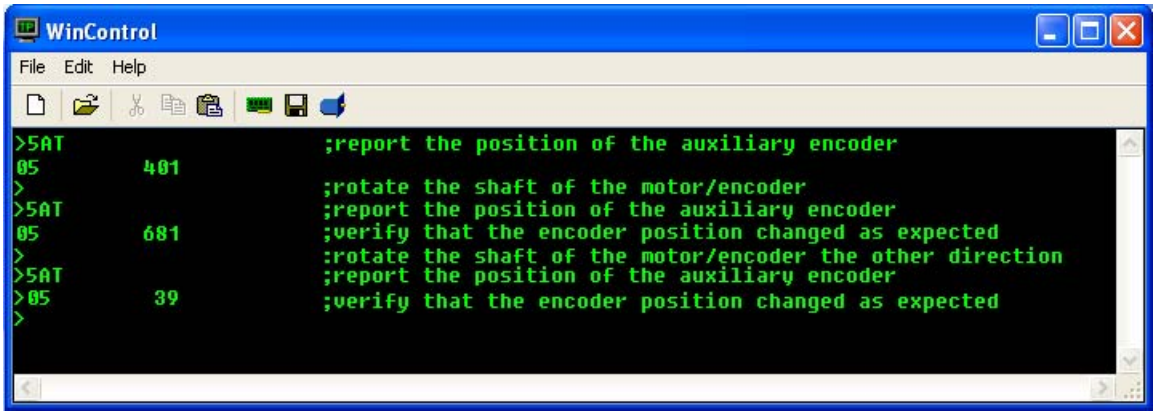


Figure 44. Verify auxiliary encoder operation using WinControl

To use **Motor Mover** you must configure the stepper axis for closed loop mode. This is because **Motor Mover** uses the ***MCGetPositionEX( )*** function for the position readouts. For an open loop stepper the ***MCGetPositionEX( )*** function returns the Step Count Position Register value (not the auxiliary encoder count). To enable closed loop stepper mode, from the **Motor Mover Setup** menu select:

Closed Loop Mode checkbox
OK

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Toggle the Off & On buttons

The Motor Mover position display will report the position of the encoder.

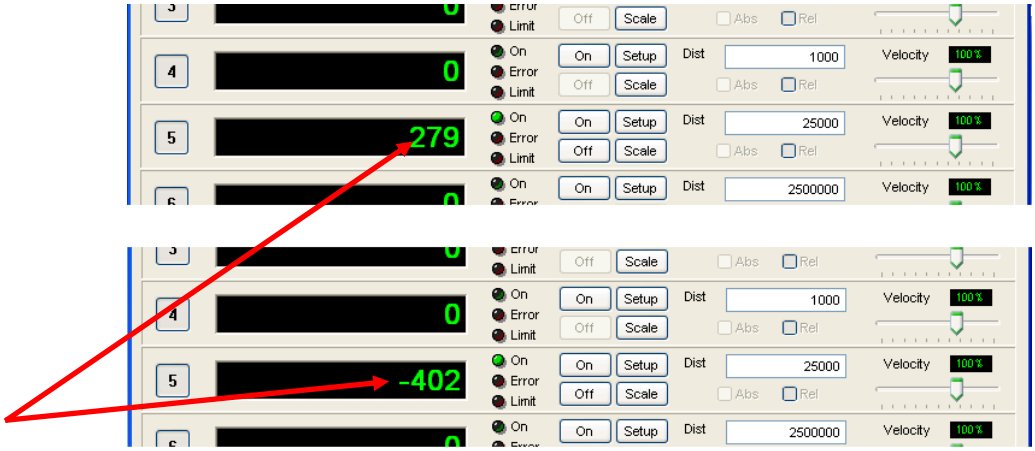


Figure 45. Rotate the motor / encoder shaft back and forth. Verify that the position is changing accordingly

If Motor Mover was used to verify proper auxiliary encoder operation don't forget to disable Closed Loop Stepper mode.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

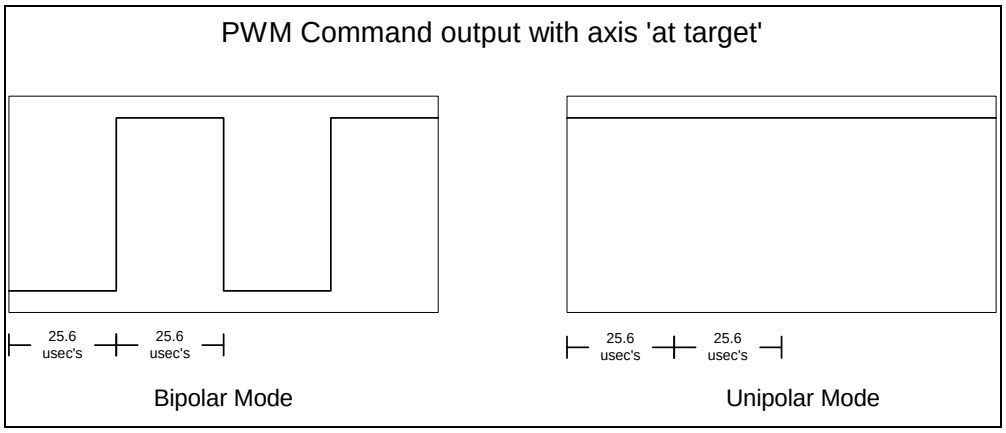
PWM Servo Command

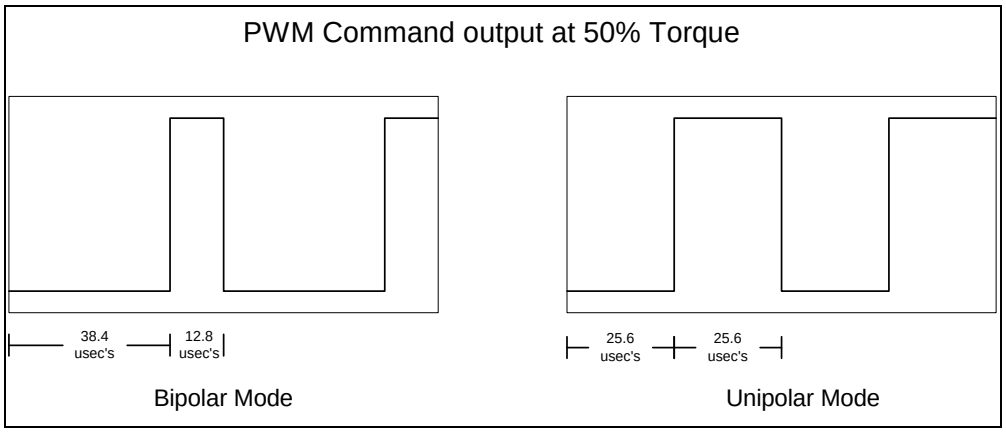
For cost sensitive DC servo motor applications the combination of the MultiFlex motion controller and an inexpensive external analog H-Bridge Driver can provide a very cost-effective solution. The controller provides the PWM command which the H-Bridge Driver then proportionally converts to motor current to drive the servo motor.



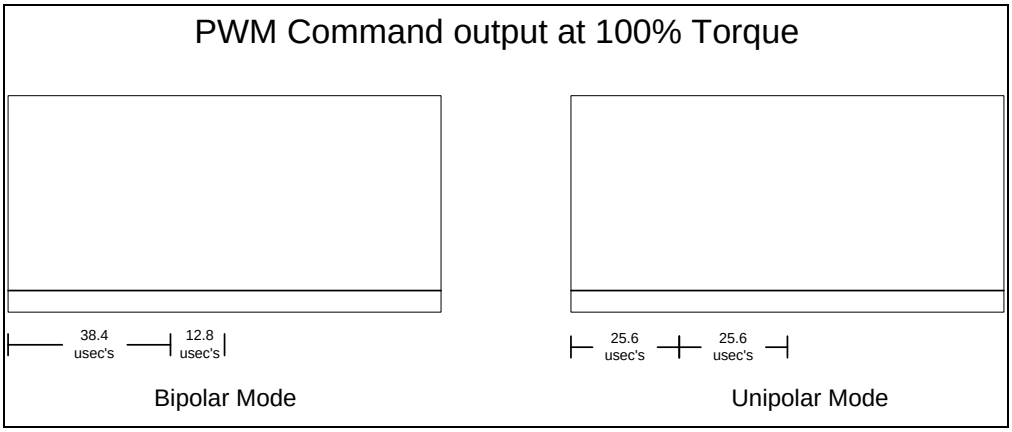
PWM Command motion requires both Motion Control API 3.5.0 or higher AND firmware 2.8a or higher

For each PWM servo axis the controller provides a PWM command output that can be configured for Bipolar or Unipolar mode (PWM frequency = 19.53 KHz). For applications that require a different frequency please contact the factory. In Biolar mode, when the motor is at its target position the PWM output will be at 50% duty cycle. In Unipolar mode, when the motor is at its target position the PWM output will be off (0% duty cycle). For many applications the Unipolar mode may be preferrable because it is more heat efficient (no current across the motor when it is at its target and enabled).





Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



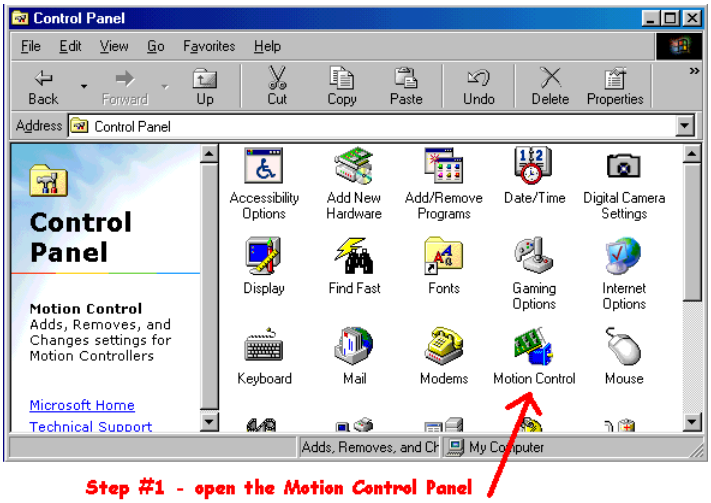
Configuring PWM operation

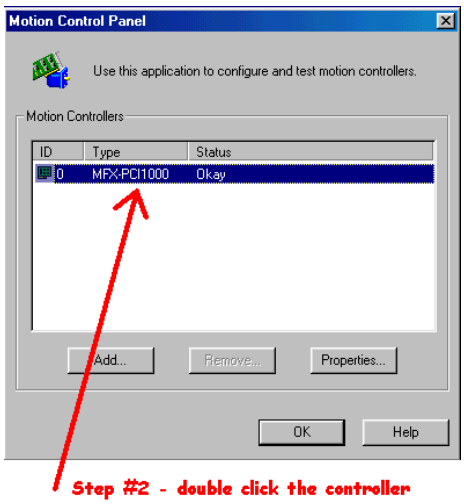
To configure / enable a servo axis for PWM Command set the PWM mode use the Motion Control API function (rev. 3.5.0 or higher) ***MCSetModuleOutputMode( )*** with the *mode* parameter set to **MC_OM_UNI_PWM** for Unipolar mode, or to **MC_OM_BI_PWM** for Bipolar mode.

```
MCSetModuleOutputMode( hCtlr, 1, MC_OM_UNI_PWM );
MCSetModuleOutputMode( hCtlr, 2, MC_OM_BI_PWM );
```

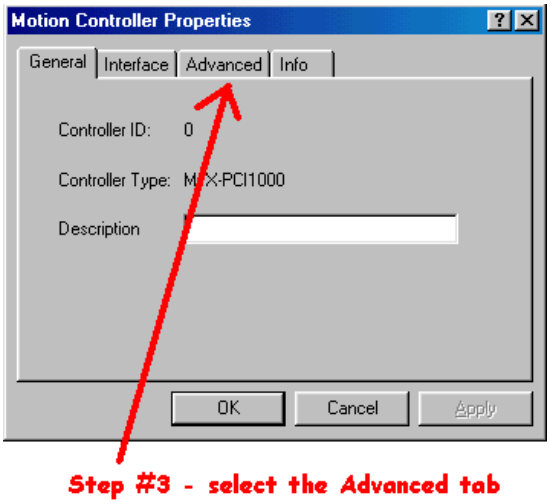
For Unipolar operation you will need to configure one of the TTL general purpose digital output channels to be used for the PWM Direction (Sign). For PWM Command wiring examples please refer to page 49.

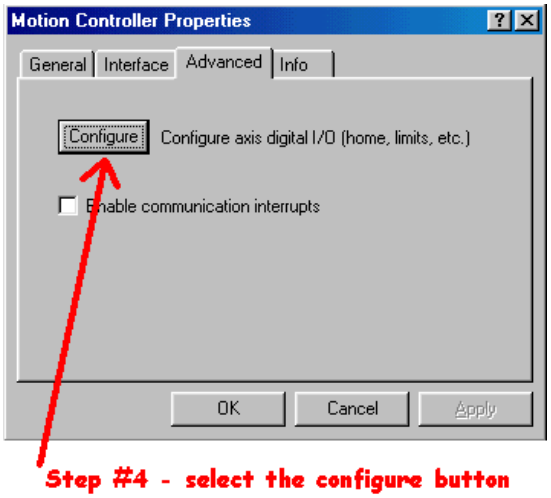
The following screen captures detail how to associate Digital output #2 (channel 34) with the PWM Direction function by using PMC's Motion Control Panel.

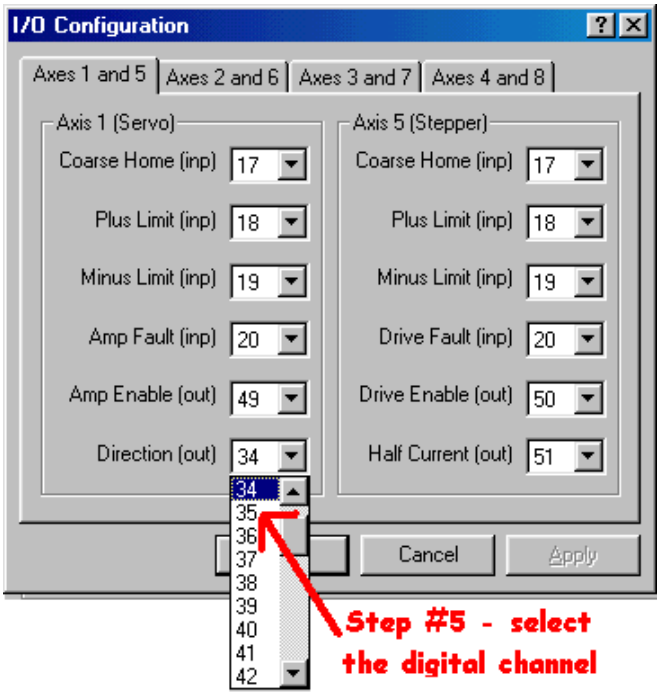




Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com







The PWM Command output (pins 6 and/or 7) is provided via an Open Collector driver. If the PWM input connection to the H-Bridge is a TTL input you will need to add a pullup resistor to +5 volts.

Once the PWM Mode has been selected the servo can be tuned and exercised using any the standard PMC tools (Servo Tuning utility and Motor Mover).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Record Motion Data

The controller supports capturing and retrieving motion data for closed loop axes and open loop steppers. As many as 1024 'data sets' (actual, optimal, following error, DAC output) can be captured for each axis. Captured position data is typically used to analyze servo motor performance and PID loop tuning parameters. PMC's Servo Tuning utility uses this function to analyze servo performance. The Motion Control API function ***MCCaptureData( )*** is used to acquire motion data for a servo axis. This function supports capturing:

- Actual Position versus time
- Optimal Position versus time
- Following error versus time
- DAC output versus time (Analog Command axes only)
- Auxiliary encoder position (for tuning an open loop Pulse Command servo axis)

The time base (4 KHz, 2 KHz, 1 KHz) for captured data is set by **Rate** member of the **MCMotion** data structure. The function ***MCGetCapturedData( )*** is used to retrieve the captured data. This example captures 1000 data points from axis 3, then reads the captured data into an array for further processing.

```
double Data[1000];

MCBlockBegin( hCtlr, MC_BLOCK_COMPOUND, 0 );
MCCaptureData( hCtlr, 3, 1000, 0.001, 0.0 );
MCMoveRelative( hCtlr, 3, 1000.0 );
MCWaitForStop( hCtlr, 3, 0.0 );
MCBlockEnd( hCtrlr, NULL );

//  Retrieve captured actual position data into local array
//
if (MCGetCaptureData( hCtlr, 3, MC_DATA_ACTUAL, 0, 1000, &Data ) {
    . . .        // process data
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Resetting the Controller

The controller supports software controlled reset. To reset the controller CPU and all axes issue the Motion Control API function ***MCReset( )***. For additional information please refer to the **Motion Control API Reference Manual**.

Most PMC application programs (Motor Mover, Servo Tuning, WinControl) allow the user to reset the controller by selecting ***Reset Controller*** from the WinControl File menu.

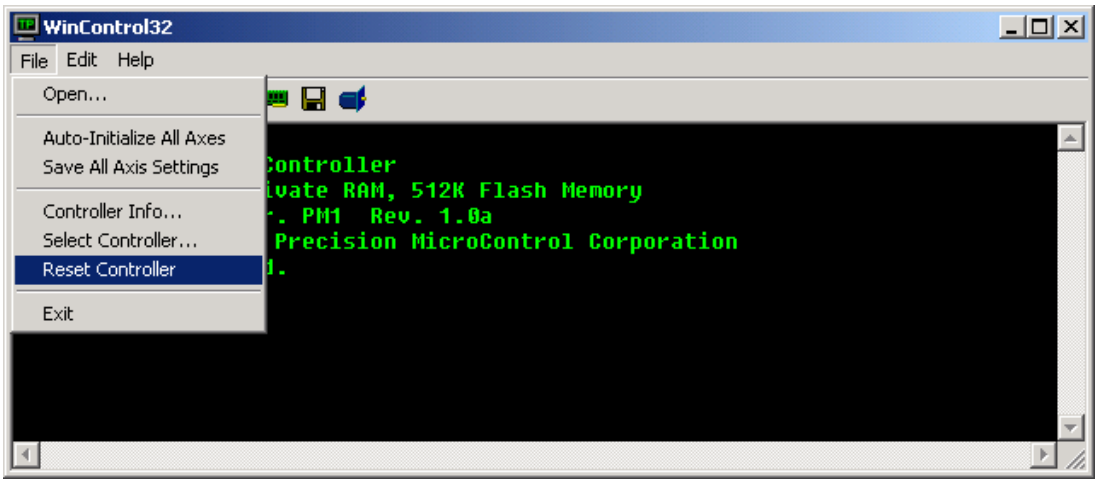


Figure 46. Resetting the Controller

Resetting the controller from a user application program (with ***MCReset( )***) or from one of a PMC's software programs (by selecting ***Reset Controller*** from: Motor Mover, WinControl, Servo Tuning, etc...) will cause the controller to revert to default settings (PID, velocity, accel/decel, limits, etc...). For information restoring the user defined settings please refer to the **Initializing and Restoring Controller Configuration** section in this chapter.



In the event of a 'hang up' of the application program and/or controller, the application program may fail to resume operation after issuing the ***MCReset( )*** function. The user will have to terminate and then re-open the application program.



The contacts of a normally open relay are available on pins 1 and 2 of connector J8. Following a reset (***MCReset( )***) or after a PC re-boot / power cycle the relay will not be energized until the controller has been fully initialized.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Single Stepping MCCL Programs

While the controller is executing any Motion Control Command Language (MCCL) macro program, the user can enable single step mode by entering <ctrl> <B>. Each time this keyboard sequence is entered, the next MCCL command in the program sequence will be executed. The following macro program will be used for this example of single stepping:

```
MD10,WA1,1MR1000,1WS.1,1TP,1MR-1000,1WS.1,1TP,RP
```

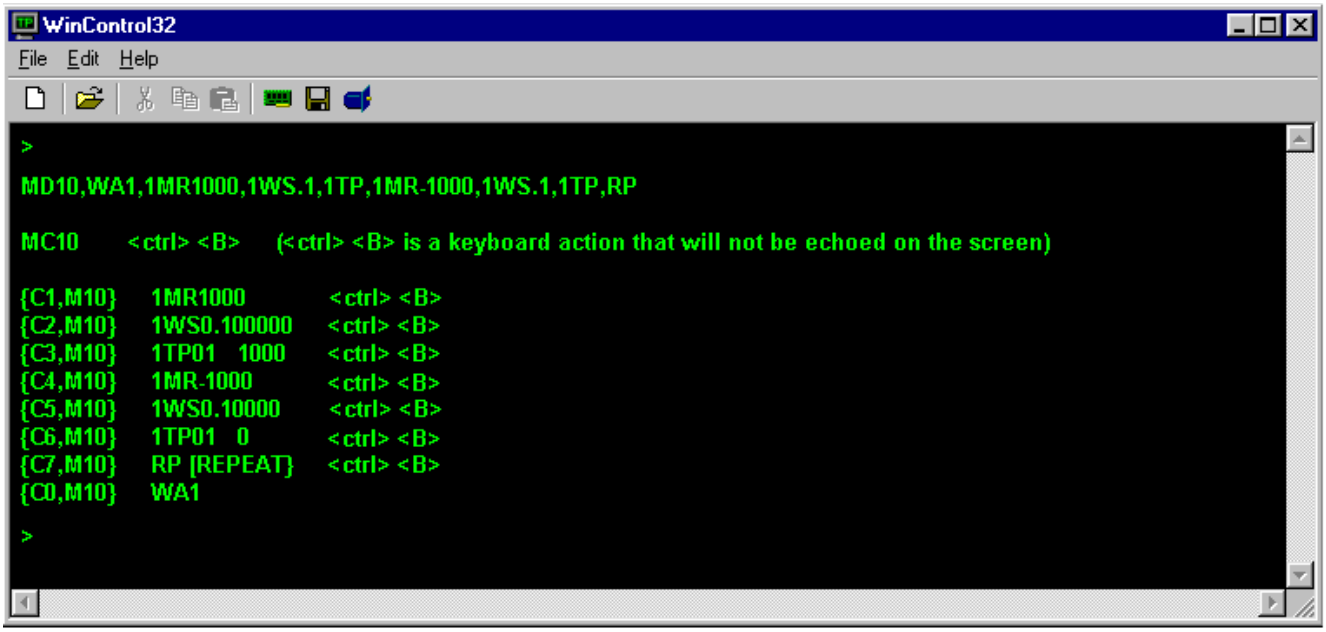
This sample program will: wait for 1 second, move 1000 encoder counts, report the position 100 msec's after the calculated trajectory is complete, move -1000 encoder counts, report the position 100 msec's after the calculated trajectory is complete, repeat the command sequence.

This command sequence can be entered directly into the controller's memory by typing the command sequence in the terminal interface program WinCtl32.exe or by downloading a text file via WinControl's file menu.

To begin single step execution of the above example macro enter MC10 (call macro #10) then <ctrl> <B> the following will be displayed:

```
{C1,MC10} 1MR1000 <
```

The display format of single step mode is: {Command #,Macro #} Next command to be executed



To end single stepping and return to immediate MCCL command execution press <Enter>. To abort the MCCL program enter <Escape>. Single step mode is not supported for a MCCL sequence that is executing as a background task.

Single stepping can also be enabled from within a MCCL program by using the break command immediately followed by a "string" parameter. When the break command is executed the controller will display the characters in the string (inside the quotation marks) and then delay additional command execution until the space bar (execute next command and then delay) or the enter key (terminate single stepping and resume program execution) are selected. In the following example axis one will move 1000

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

counts, report the position, move –1000 counts, report the position, halt command execution until the space bar is entered, repeat one time.

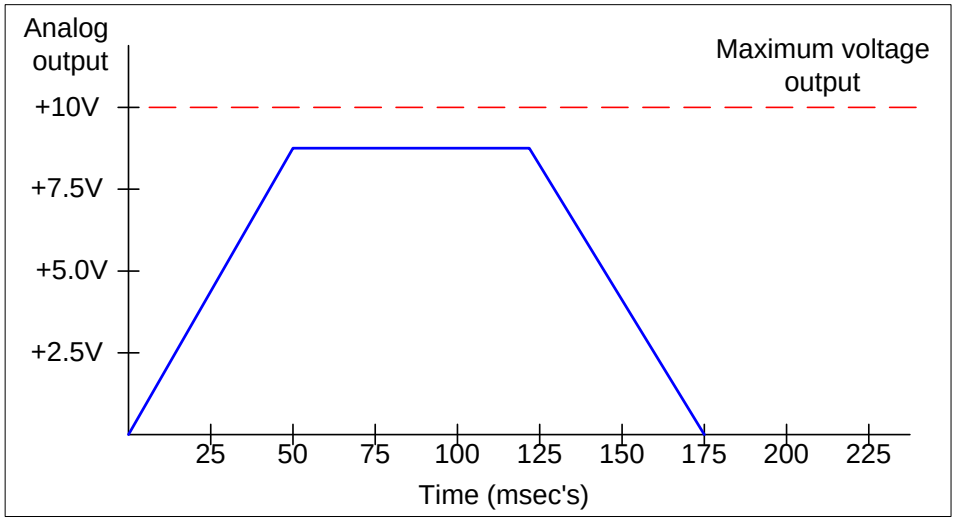
```
MC10 1MR1000,1WS0.100000,1TP,1MR-1000,1WS0.100000,1TP,BK"wait",RP1


>mc10
01 997
01 0
BREAK AT COMMAND 6, MACRO 10
wait
 {C7,M10} RP10 [REPEAT] <
   <space bar>
01 997
01 0
BREAK AT COMMAND 6, MACRO 10
wait
 {C7,M10} RP10 [REPEAT] <
>
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

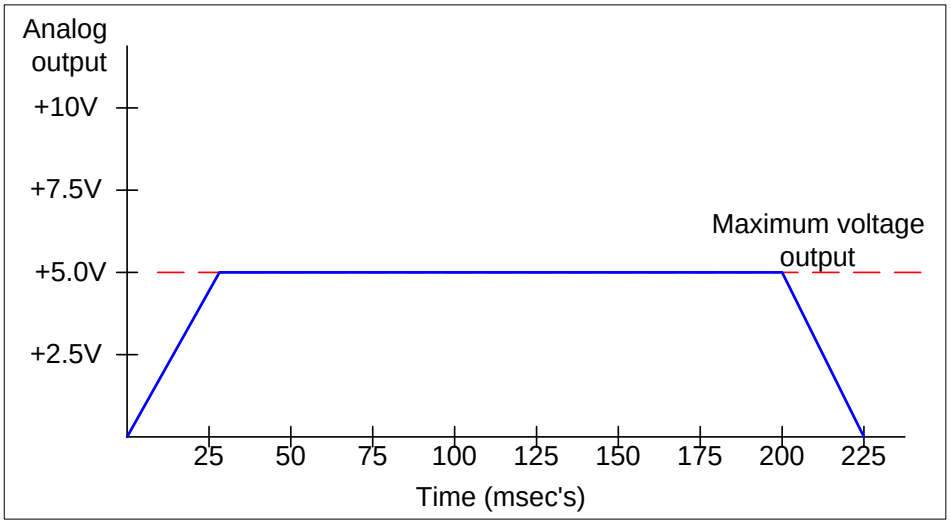
Torque Mode Output Control

The +/- 10V Analog Command outputs channels provide two methods of directly and completely controlling the Torque/Velocity of a axis. When executing closed loop servo motion in Position or Velocity mode, the MCSetTorque( ) command allows the user to limit the output signal or duty cycle to a specific level. The following graph depicts a simple position mode move of 1000 encoder counts with the default torque setting of 10 volts (no limit).



The graphic below depicts the same 1000 encoder count move, but the maximum voltage output has been limited to 5.0 volts.

```
MCSetTorque( hCtlr, 1, 5.0 );
MCMoveRelative( hCtlr, 1, 1000.0 );
```



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Analog Command output channels as simple D/A output with encoder reader

Selecting Torque mode (Mode = MC_MODE_TORQUE) using the MCSetOperatingMode( ) function allows the user to directly write values to the servo control DAC. This mode does not support closed loop servo control, but the user can read the position of the encoder at any time.

```
MCSetOperatingMode( hCtlr, 1, 0, MC_MODE_VELOCITY );
MCSetTorque( hCtlr, 1, 2.5 );          ;axis 1 output to 2.5V (MC300)
MCSetTorque( hCtlr, 1, 7.5 );          ;set duty cycle to 75% (MC320)
```



When operating in Torque Mode the Following Error and Limit error checking is disabled. If either or both of these error conditions exist the controller will not command the axis to stop.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Turning off Integral gain during a move

Servo controllers primarily use Proportional gain to determine the current/velocity command signal that the controller applies to the servo amplifier during a move. For motion control applications, integral gain is used primarily to reduce the static position error **at the end of a move**. For additional information about servo tuning and integral gain please refer to :

- the Servo Tuning description in the **Motion Control** chapter of this manual
- the Servo tuning tutorials on PMC's Motion CD and available for download from the Support section of PMC's web site.

For some applications, integral gain has a tendency to cause bounce or oscillation of the command signal during a move. This tendency can be is especially problematic in:
High gain servo systems

- Systems with high and / or irregular friction
- Systems with unbalanced loads
- Systems with unbalanced and / or high offset amplifiers

The following graphic shows the typical response of a high gain servo system when integral gain is enabled through out the move. Even though the following error never exceeds 10 encoder counts during the 100,000 count move, a significant oscillation (+/- 10 counts) occurred.

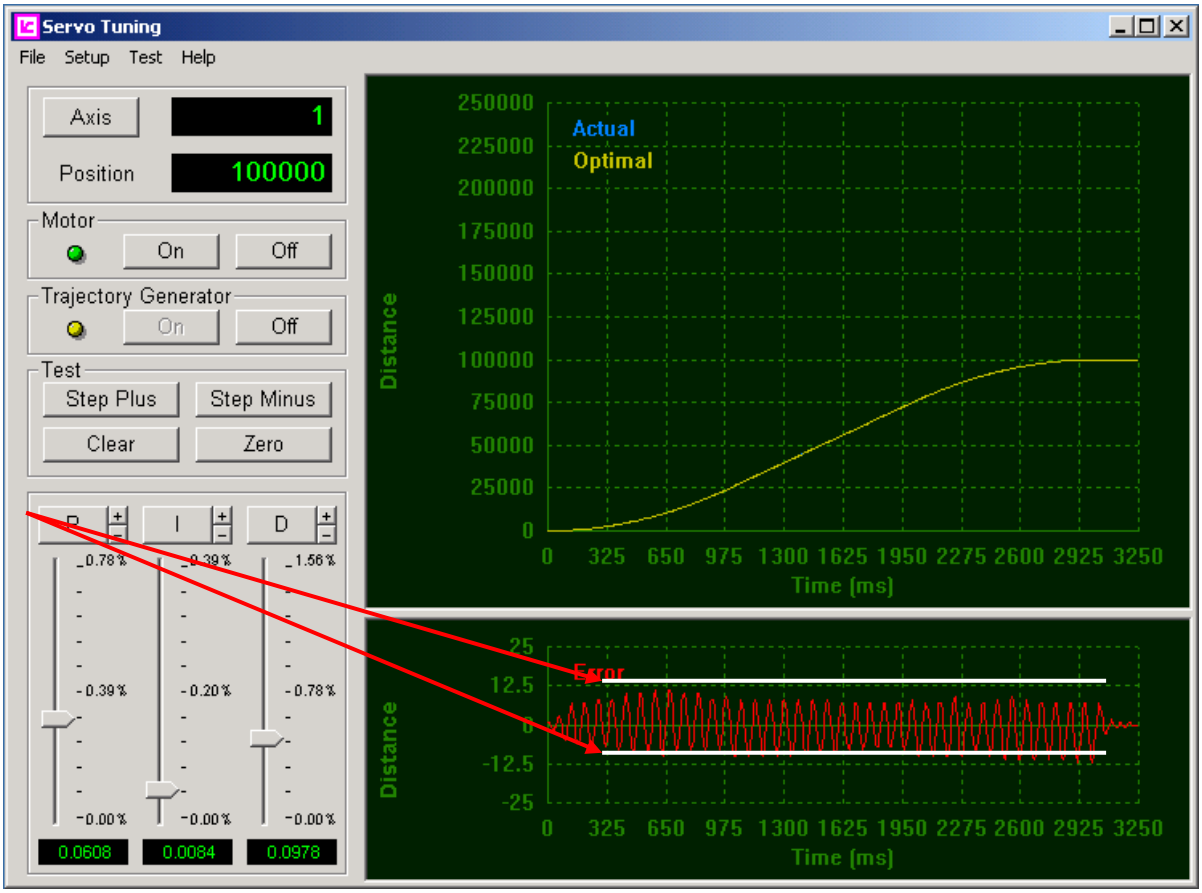


Figure 47. Typical servo response when integral gain is enabled throughout the move

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

By disabling the integral gain term until after the trajectory is complete (desired position = target position) the same move is accomplished with a following error of +/- 3 counts versus +/- 10 counts.

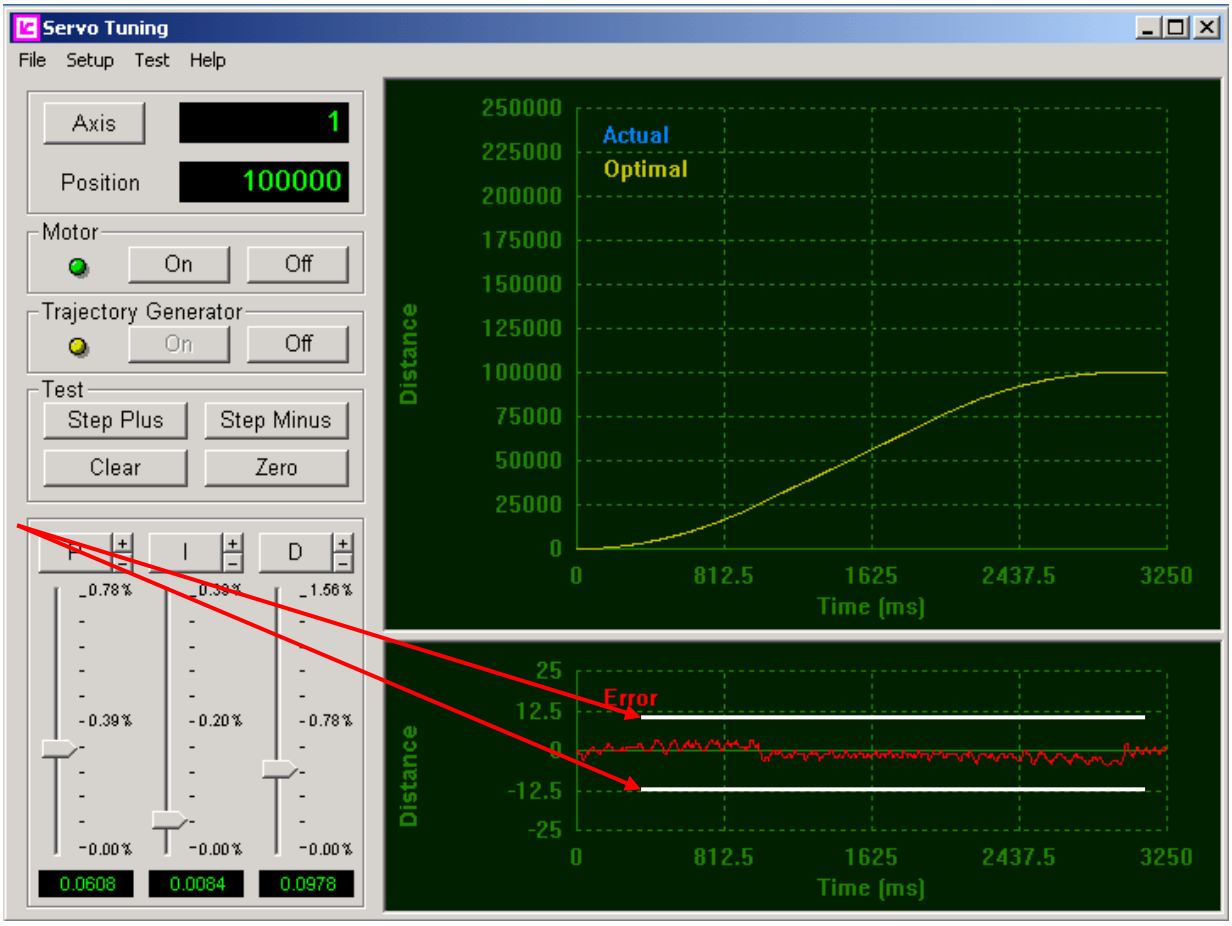


Figure 48. Typical servo response when integral gain is disabled until the calculated is complete

The IntegralOption member of the MCFilterEx structure allows the user to select from three different mode of integral gain operation for servo or closed loop stepper.

IntegralOptiont value	**Notes – (all other servo parameters remaining unchanged)**
MC_INT_NORMAL - integral term always on (default)	Smallest following error during move. As the integral term is increased the command output / following error will tend to bounce
MC_INT_FREEZE - Freezes accumulation of integration term during movement. Integration will continued once the calculated trajectory (trajectory complete, status bit 3 = 1) has been completed.	Ideal for applications with unbalanced loads (robotic arm with vertical axis, hoist)
MC_INT_ZERO - Zero and freeze accumulation of the integration term when motion begins. When the calculated trajectory (trajectory complete, status bit 3 = 1) has completed, enable the integration term	Most stable command signal / servo performance during the move. Largest following error during the move. Not acceptable for applications with unbalanced load.

From PMC application programs like Servo Tuning and Motor Mover the integral gain mode can be

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

selected from the Servo Setup Dialog.

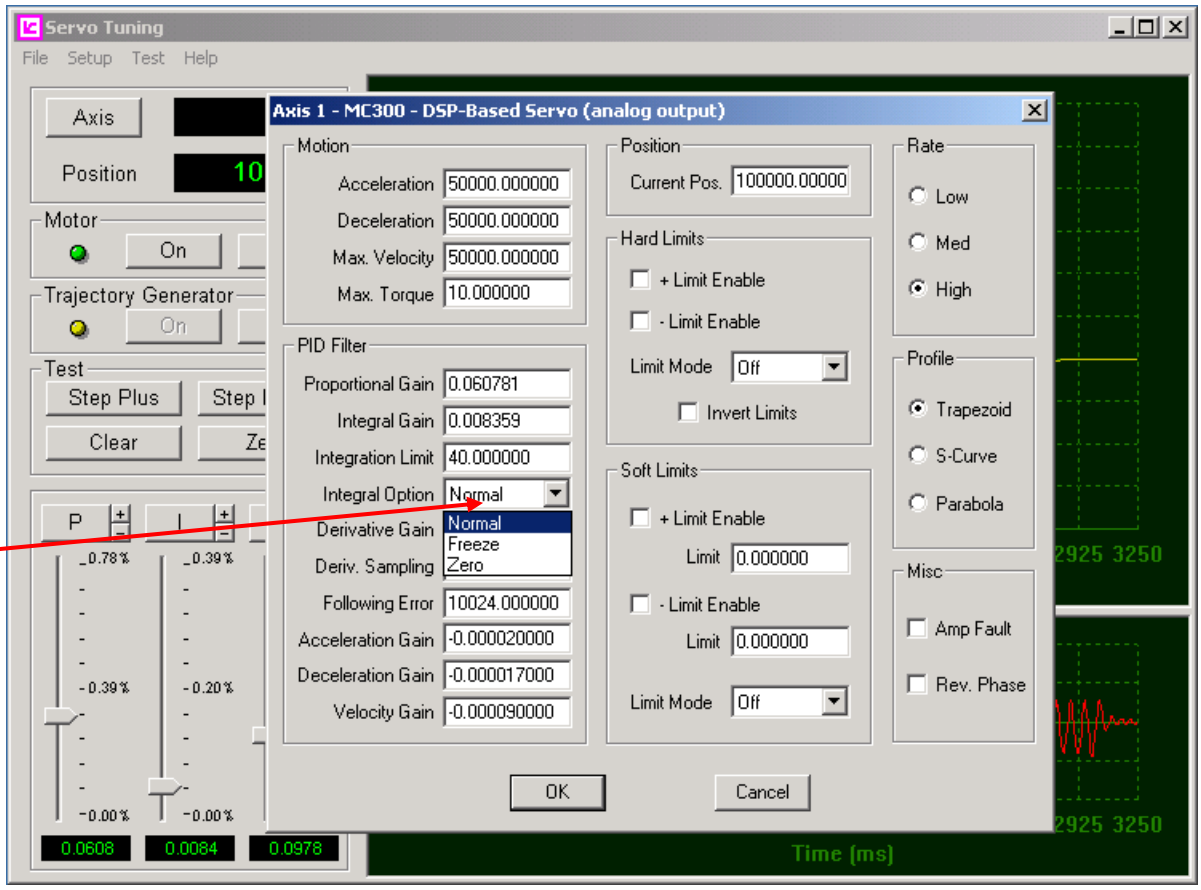


Figure 49. Using Servo Tuning's Servo Setup Dialog to set the integral gain mode of operation

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Defining User Units

When power is applied or the controller is reset, it defaults to encoder counts or stepper pulses as its units for motion command parameters. If the user issues a move command to a servo with a target of 1000, the controller will move the servo 1000 encoder counts. If the user issues the same command to a stepper motor, it will issue 1000 step pulses.

In many applications there is a more convenient unit of measure than the encoder counts of the servo or steps of the stepper motor. If there is a fixed ratio between the encoder counts or steps and the desired 'user units', the controller can be programmed with this ratio and it will perform conversions implicitly during command execution.

Defining user units is accomplished with the function ***MCSetScale( )***, which uses the **MCSCALE** data structure. This function provides a way of setting all scaling parameters with a single function call using an initialized **MCSCALE** structure. To change scaling, call ***MCGetScale( )***, update the **MCSCALE** structure, and write the changes back using ***MCSetScale( )***.



Before changing any/all of the axis related scaling values (Scale, Rate, Offset, or Zero) **the axis must first be disabled (turned off)**. To complete a scaling change enable (turn on) the axis.

MCScale Data Structure

```
typedef struct {

double    Constant;       // Define output constant
   double    Offset;            // Define the work area zero
   double    Rate;              // Define move (vel., accel, decel)
time units
   double    Scale;          // Define encoder scaling
   double    Zero;              // Define part zero
   double    Time;              // Define time scale

} MCMOTION;
```

Setting Move (Encoder/Step) Units

The value of the **Scale** member is the number of encoder counts or steps per user unit. For example, if the servo encoder on axis 1 has 1000 quadrature counts per rotation, and the mechanics move 1 inch per rotation of the servo, then to setup the controller for user units of inches:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
MCSCALE Scaling;

MCEnableAxis( hCtlr, 3, False);
MCGetScale( hCtlr, 3, &Scaling );
Scaling.Scale = 1000.0;                 // 1000 encoder counts/inch
MCSetScale( hCtlr, 3, &Scaling );
MCEnableAxis( hCtlr, 3, True);
```

Prior to issuing the **Scale** member, the parameters to all motion commands for a particular axis are rounded to the nearest integer. After setting a new encoder scale and calling **MCEnableAxis( )** to initialize the axis, motion targets are multiplied by the ratio prior to rounding to determine the correct encoder position. Calling the ***MCGetPosition( )*** will load the scaled encoder position.



Note – setting a user scale other than 1:1 will require a change of trajectory settings (Velocity, Acceleration, Deceleration, and Velocity Gain) but not PID settings.

Trajectory Time Base

The value of the **Rate** member sets the time unit for velocity, acceleration and deceleration values, to a time unit selected by the user. If velocities are to be in units of inches per minute, the user time unit is a minute. The value of the **Rate** member is the number of seconds per 'user time unit'. If the velocity, acceleration and deceleration are to be specified in units of inches per minute and inches per minute per minute for axis 1, then the **Rate** value should be set to 60 seconds/1 minute = 60 (1UR60). The function **MCEnableAxis( ) must** be issued before the user rate will take effect.

```
MCSCALE Scaling;

MCEnableAxis( hCtlr, 3, False);
MCGetScale( hCtlr, 3, &Scaling );
Scaling.Rate = 60.0;                    // set rate to inches per minute
MCSetScale( hCtlr, 3, &Scaling );
MCEnableAxis( hCtlr, 3, True);
```

Time Unit	User Rate Conversion
second	1 (default)
minute	60
hour	3600

Figure 50. Typical Trajectory Rate Values

Defining the Time Base for Wait commands

For the ***MCWait( )***, ***WaitForStop( )*** and ***WaitForTarget( )*** functions, the default units are seconds. By setting the member **Time**, these three commands can be issued with parameters in units of the user's preference. The parameter to member is the number of 1 second periods in the user's unit of time. If the user prefers time parameters in units of minutes, **Time** = 60 should be issued.

```
MCSCALE Scaling;

MCEnableAxis( hCtlr, 3, False);
MCGetScale( hCtlr, &Scaling );
Scaling.Time = 60.0;                       // set Wait time unit to minutes
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
MCSetScale( hCtlr, &Scaling );
MCEnableAxis( hCtlr, 3, True);
```

Defining a System/Machine zero

The member **Offset** allows the user to define a 'work area' zero position of the axis. The **Offset** value should be the distance from the servo or stepper motor home position, to the machine zero position. This offset distance must use the same units as currently defined by set User Scaling command. **Offset** does not change the index or home position of the servo or stepper motor, it only establishes an arbitrary zero position for the axis.

```
MCSCALE Scaling;

MCEnableAxis( hCtlr, 3, False);
MCGetScale( hCtlr, 3, &Scaling );
Scaling.Offset = 12.25;                 // define offset to 12.25 inches
MCSetScale( hCtlr, 3, &Scaling );
MCEnableAxis( hCtlr, 3, True);
```

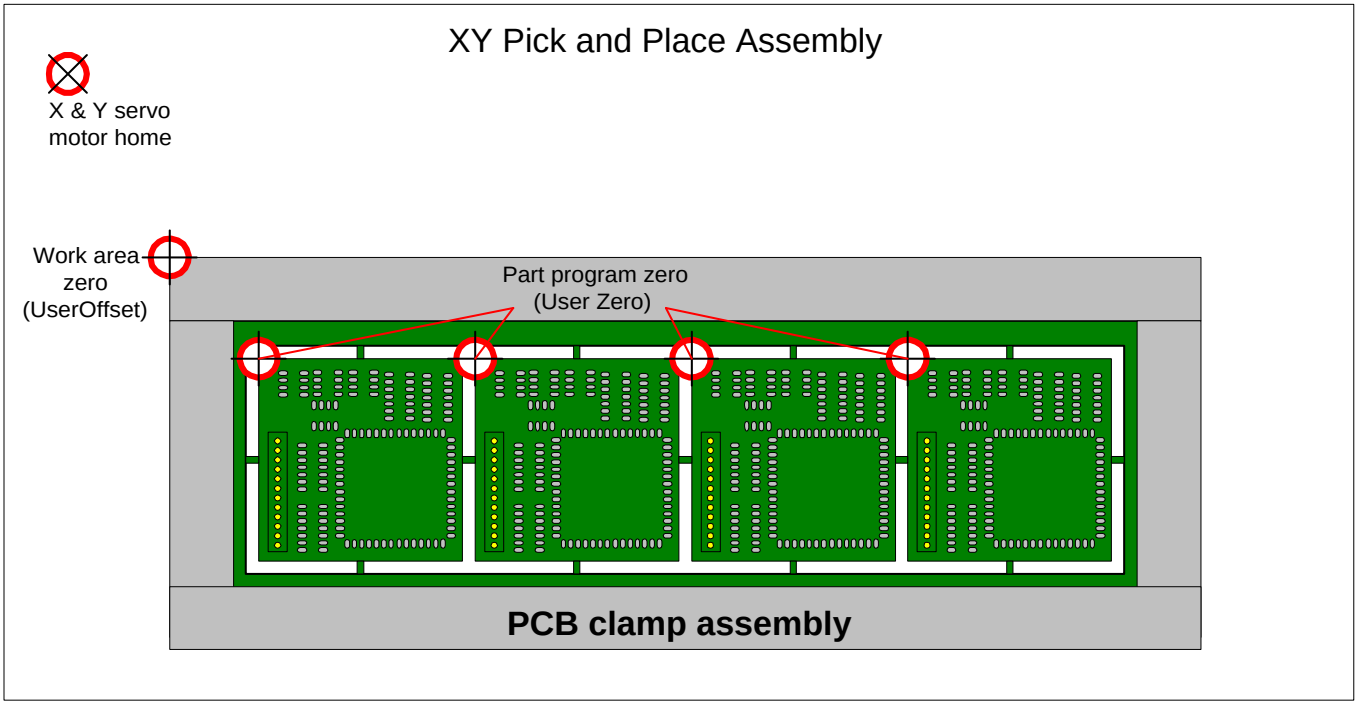
Defining a Part Zero

The member Zero would typically be used in conjunction with Offset to define a 'part zero' position. A PCB (Printed Circuit Board) pick and place operation is a good example of how this function would be used. After a new PCB is loaded and clamped into place the X and Y axes would be homed. The Offset member is used to define the 'work area' zero of the PCB. The Zero member is used to define the 'part program' or 'local' zero position. This way a single 'part placement program' can be developed for the PCB type, and a 'step and repeat' operation can be used to assemble multiple part assemblies.

```
MCSCALE Scaling;

MCEnableAxis( hCtlr, 3, False);
MCGetScale( hCtlr, 3, &Scaling );
Scaling.Offset = 12.25;                 // define offset to 12.25 inches
Scaling.Zero = 1.25;                    // define 'part zero' to 1.25
inches
MCSetScale( hCtlr, 3, &Scaling );
MCEnableAxis( hCtlr, 3, True);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Defining the output constant for velocity gain

The member **Constant** allows the user to define the units to be used for setting the Velocity Gain parameters. Please refer to the description of **Using Velocity Gain** in the **Application Solutions** chapter of this user manual.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Watchdog Circuit

The controller incorporates a watchdog circuit to protect against improper CPU operation. After a controller reset, PC reset, or PC power cycle, once the controller is initialized the watchdog circuit is enabled.

If the controller's processor fails to properly execute firmware code for a period of 200 msec's, the watchdog circuit will 'time out' and the on-board reset will be latched by the 'watchdog reset relay'. This in turn will hold the controller in a constant state of reset. All motor outputs (+/- 10V & Step/Direction) will be disabled. When the watchdog circuit has tripped, the green **Run LED** will be disabled. To clear the watchdog error either cycle power to the computer *(recommended), or r*eset the computer

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Chapter 7

General Purpose I/O

Digital I/O

The controller board provides:

- 16 TTL inputs (digital I/O channels 1 - 16)
- 16 optically isolated inputs (digital I/O channels 17 - 32)
- 16 TTL outputs (digital I/O channels 33 - 48)
- 12 open collector outputs (digital I/O channels 49 - 64)

I/O Configuration Panel

By default the optically isolated inputs and open collector outputs are associated with 'hard coded' motion control functions (Limits, Homing, Amp/Drive fault, Amp/Drive Enable). For maximum application flexibility the controllerallows the user to reassign most of the default digital I/O assignments. The Windows I/O Configuration dialog is used to change the default digital I/O configuration.

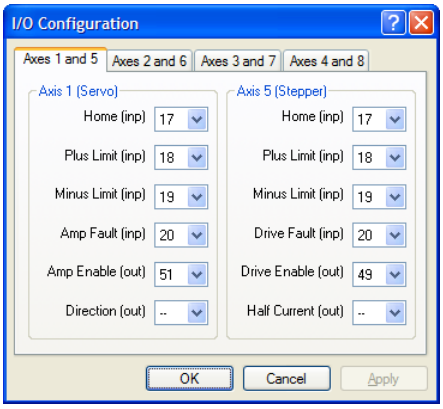


Figure 51. Digital I/O configuration panel

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The configuration dialog is launched from the Motion Control Panel (\Properties\Advanced\Configure)



The Stepper Home function **cannot be reassigned** to a different digital input channel. An open loop stepper axis can only be 'homed' by applying an active level on the controller's SCSI connector pin, #27. Additionally, Position Capture input (rising edge) and Position Compare output (high active) functions cannot be reassigned to different digital I/O channels.

Table 6. Default Function Assigned to Digital I/O Channels

Ch. #	Description		Ch. #	Description
1	TTL input 1 (Capture axes 1 & 2)		33	TTL output 1 (Compare axes 1 - 4)
2	TTL input 2		34	TTL output 2
3	TTL input 3		35	TTL output 3
4	TTL input 4		36	TTL output 4
5	TTL input 5 (Capture axes 3 & 4)		37	TTL output 5
6	TTL input 6		38	TTL output 6
7	TTL input 7		39	TTL output 7
8	TTL input 8		40	TTL output 8
9	TTL input 9 (Capture axes 5 & 6)		41	TTL output 9 (Compare axes 5 - 8)
10	TTL input 10		42	TTL output 10
11	TTL input 11		43	TTL output 11
12	TTL input 12		44	TTL output 12
13	TTL input 13 (Capture axes 7 & 8)		45	TTL output 13
14	TTL input 14		46	TTL output 14
15	TTL input 15		47	TTL output 15
16	TTL input 16		48	TTL output 16
17	Opto isolated (3V - 25V) Axis 1 Coarse Home Axis 5 Home		49	Open collector (100mA) Axis 1 Amp Enable
18	Opto isolated (3V - 25V) Axis 1/5 Limit +		50	Open collector (100mA) Axis 5 All Driver Disable
19	Opto isolated (3V - 25V) Axis 1/5 Limit -		51	Open collector (100mA) Axis 5 Half Current
20	Opto isolated (3V - 25V) Axis 1/5 Amp Fault			
21	Opto isolated (3V - 25V) Axis 2 Coarse Home Axis 6 Home		53	Open collector (100mA) Axis 2 Amp Enable
22	Opto isolated (3V - 25V) Axis 2/6 Limit +		54	Open collector (100mA) Axis 6 Driver Disable
23	Opto isolated (3V - 25V) Axis 2/6 Limit -		55	Open collector (100mA) Axis 6 Half Current
24	Opto isolated (3V - 25V) Axis 2/6 Amp Fault			
25	Opto isolated (3V - 25V) Axis 3 Coarse Home Axis 7 Home		57	Open collector (100mA) Axis 3 Amp Enable
26	Opto isolated (3V - 25V) Axis 3/7 Limit +		58	Open collector (100mA) Axis 7 Driver Disable
27	Opto isolated (3V - 25V) Axis 3/7 Limit -		59	Open collector (100mA) Axis 7 Half Current
28	Opto isolated (3V - 25V) Axis 3/7 Amp Fault			
29	Opto isolated (3V - 25V) Axis 4 Coarse Home Axis 8 Home		61	Open collector (100mA) Axis 4 Amp Enable
30	Opto isolated (3V - 25V) Axis 4/8 Limit +		62	Open collector (100mA) Axis 8 Driver Disable
31	Opto isolated (3V - 25V) Axis 4/8 Limit -		63	Open collector (100mA) Axis 8 Half Current
32	Opto isolated (3V - 25V) Axis 4/8 Amp Fault			

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

All Digital I/O signals can be accessed via the connectors on the available interconnection boards.

Upon completion of General Purpose Digital I/O re-configuration, selecting OK will cause the updated configuration to be written into the Windows registry.

Configuring and Exercising the Digital I/O

The configuration of the digital I/O is accomplished using either PMC's Motion Integrator software or the Motion Control API function ***MCConfigureDigitalIO( )***. The screen shot that follows shows the Motion Integrator Digital I/O test panel. This tool can be used to configure and exercise each digital I/O channel. Comprehensive on-line help is available from the Help menu.

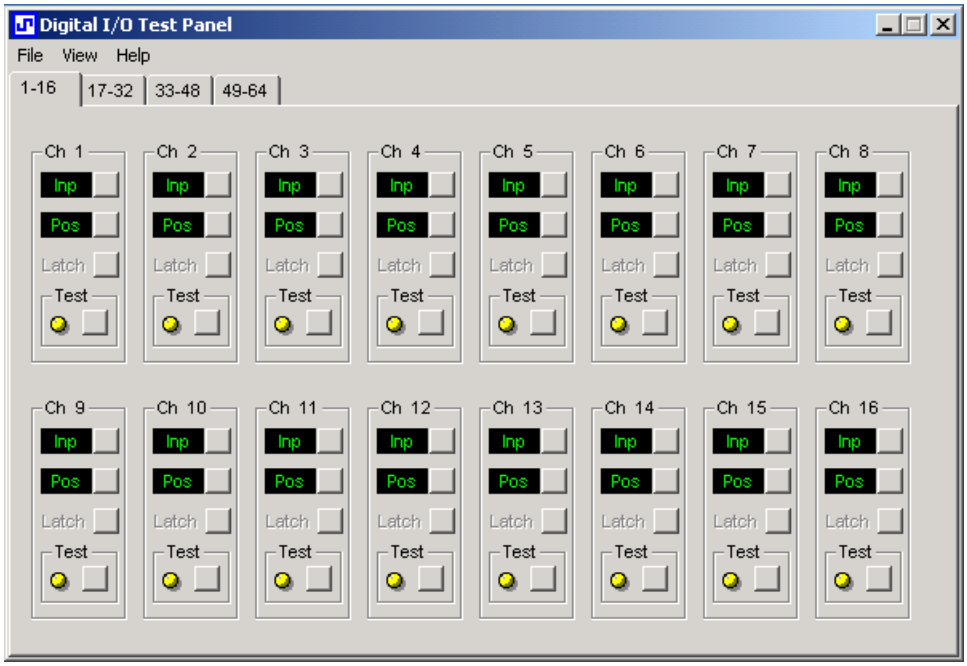


Figure 52. Digital I/O Test Panel

The Digital I/O Test Panel groups the 64 digital I/O channels into 4 banks of 16:

- TTL inputs (channels 1 - 16) = **Standard I/O** tab
- Optically isolated inputs (channels 17 - 32) = **Module 1** tab
- TTL outputs (channels 33 - 48) = **Module 2** tab
- Open collector driver outputs (channels 49 - 64) = **Module 3** tab

Each channel is individually programmable as either:

- High true/Positive logic (MC_DIO_HIGH)
- Low true/Negative logic (MC_DIO_LOW)

For each digital I/O channel, the Test LED indicates the current state of the channel.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Using the Digital I/O

After configuring the Digital I/O channels with the ***MCConfigureDigitalIO( )*** function, three Motion Control API functions are available for activating and monitoring the digital I/O:

MCEnableDigitalIO( )	set digital output channel state
MCGetDigitalIO( )	get digital input channel state
MCWaitForDigitalIO( )	wait for digital input channel to reach specific state

Enable Digital IO

Turns the specified digital I/O on or off, depending upon the value of *bState*.

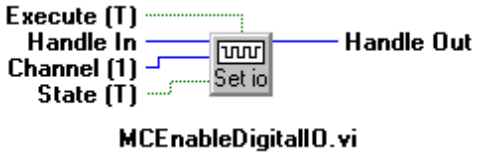
TRUE Turns the channel on
FALSE Turns the channel off

Note that depending upon how a channel has been configured "on" (and conversely "off") may represent either a high or a low voltage level.

see also: Configure Digital IO

C++ Function: void MCEnableDigitalIO(HCTRLR hCtlr, WORD wChannel, short int bState);
Delphi Function: procedure MCEnableDigitalIO(hCtlr: HCTRLR; wChannel: Word; bState: SmallInt);
VB Function: Sub MCEnableDigitalIO (ByVal hCtrlr As Integer, ByVal channel As Integer, ByVal state As Integer)
MCCL command: CF, CN

LabVIEW VI:



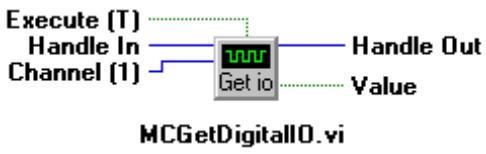
Get Digital IO

Returns the current state of the specified digital I/O channel. This function will read the current state of both input and output digital I/O channels. Note that this function simply reports if the channel is "on" or "off"; depending upon how a channel has been configured "on" (and conversely "off") may represent either a high or a low voltage level.

see also:

C++ Function: short int MCGetDigitalIO(HCTRLR hCtlr, WORD wChannel);
Delphi Function: function MCGetDigitalIO(hCtlr: HCTRLR; wChannel: Word): SmallInt;
VB Function: Function MCGetDigitalIO (ByVal hCtrlr As Integer, ByVal channel As Integer) As Integer
MCCL command : TC

LabVIEW VI:



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

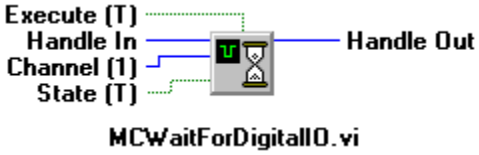
Wait for Digital IO

Waits for the specified digital I/O channel to go on or off, depending upon the value of bState.

see also: Wait for digital channel on

C++ Function: void MCWaitForDigitalIO(HCTRLR hCtlr, WORD wChannel, short int bState);
Delphi Function: procedure MCWaitForDigitalIO(hCtlr: HCTRLR; wChannel: Word; bState: SmallInt);
VB Function: Sub MCWaitForDigitalIO (ByVal hCtrlr As Integer, ByVal channel As Integer, ByVal state As Integer)
MCCL command: WF, WN

LabVIEW VI:



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

A/D Inputs

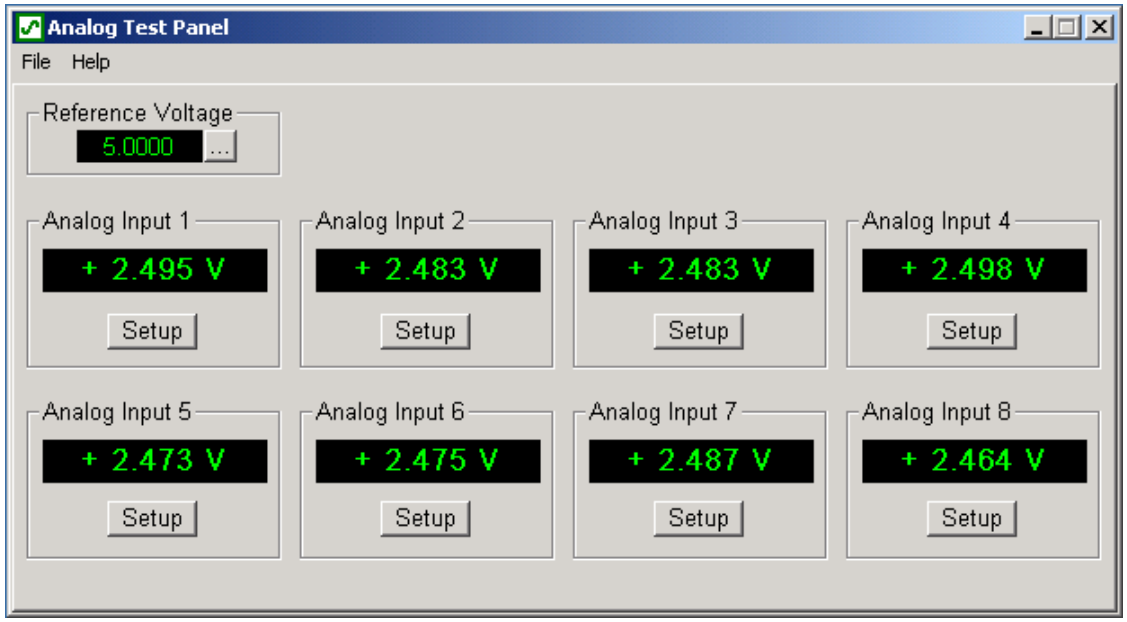
One of the available controller options is the capability to read 8 A/D input channels (with 14 bit resolution). This A/D input option can be obtained in two different voltage ranges: -10V to +10V (default) or 0V to +4V (by special order).

Because the controller is implemented in digital electronics, all analog input signals levels must be converted into a digital value. Even though the current A/D device provides 14 bits of resolution, to support possible future increases in resolution up to 16 bits, the value returned by the ***MCGetAnalogEx( )*** function is a 'left justified' 16 bit value (0 - 65532). A returned value of 0 translates to the lowest analog voltage in the input range. A digital value of 65532 translates to the highest analog voltage in the input range. These inputs are very high impedance with leakage currents less than 10 nano amps.

Using the A/D inputs

You can read the analog input values using either the Motion Control API function ***MCGetAnalogEx( )***, or by issuing the MCCL command **T**ell **A**nalog (**TA*x, x=channel number***) from PMC's WinControl program. The value returned for each input channel will be a number between 0 and 65536, corresponding to the entire input voltage range. For example, if the input voltage range is -10V to +10V; then -10.0V=0, 0.0V=32768 and +10.0V=65536. If the input voltage range is 0.0 to +4.0V; then 0.0V=0, 2.0V=32768 and 4.0V=65536.

The screen capture that follows shows the Motion Integrator Analog I/O test panel, which can also be used to report the measured voltage level. Comprehensive on-line help is available from the Help menu.



The Motion Control API function for reading an A/D input channel is as follows:

MCGetAnalogEx( ) get digital input channel digitized level

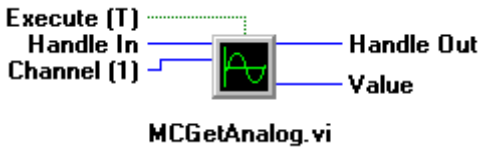
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Get Analog

Reads the digitized level of the specified input *wChannel*. For each of A/D input channel, this function will return a value between 0 and 65532.

C++ Function: WORD MCGetAnalogEx( )(HCTRLR hCtlr, WORD wChannel);
Delphi Function: function MCGetAnalogEx( )(hCtlr: HCTRLR; wChannel: Word): Word;
VB Function: Function MCGetAnalogEx( ) (ByVal hCtrlr As Integer, ByVal channel As Integer) As Integer
MCCL command: TA

LabVIEW VI:



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Chapter 8

Specifications

Motion Control Board

Function	1 - 8 Axis, servo, stepper and I/O controller
Installation	PCI-bus universal (3.3 & 5V) short-card
Main Processor	Toshiba 64-bit MIPS RISC CPU
Processor Clock Rate	200 MHz
Memory	Flash Memory: 2 KB Synchronous DRAM: 16 MB
Processor Fault Detection	Watchdog Circuit with Reset Relay (normally open) Max. switching power = 30W Max. switching current = 1A max. switching voltage: DC = 110V, AC=125V
Status LED's	Power, Reset, Run
Standard Communication Interface	32-bit, 33 MHz PCI Bus 4 KB dual ported memory in Memory Address Space 'Plug & Play' dynamic addressing
Connectors	4 SCSI (VHDCI) connectors (2 axes each)
Required PCI-bus Supply Voltages	+3.3V (0.6A), +5V (0.45A), +12V (0.07A), and -12V (0.07A)
PCI-bus Signaling Level compatibility	3.3V and 5V (connector is keyed for Universal PCI bus support)
Form Factor	Half-length, full-height PCI-bus card (6.875" x 4.2")
Operating Temperature range	0^0 C to 55^0 C, non-condensing

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Analog Command Axis Specifications

Function	Closed Loop Servo Motor Control
Operating Modes	Position, Velocity, Contouring, Torque, and Gain
Filter Algorithm	PID with Velocity, Accel / Decel Feed-Forward (PID-VAFF)
Servo Filter Update Rate	4, 2, or 1 KHz - software selectable
Trajectory Generator	Trapezoidal, Parabolic or S-Curve Independent Acceleration and Deceleration
Command output	Analog Signal (+/- 10 vdc @ 10 ma, 16 bit)
Position Feedback	Incremental Encoders with Index and Hardware Error Detection
Position and Velocity Resolution	64 bit floating point
Encoder	
Encoder and Index Inputs	Differential or Single-ended, -25 to +25 vdc max.
Encoder Count Rate	Up to 20,000,000 Quadrature Counts/sec. per axis
Encoder Supply Voltage	+5 vdc or +12 vdc
Minimum Phase differential	200mV
Hardware Error Checking	Yes (for differential encoders)
Axis Inputs (Optically isolated)	Limit+, Limit-, Home, Amplifier/Drive Fault
Device conducting	minimum voltage = 3V maximum voltage = 25V
Minimum current required	0.25 mA
Axis Outputs (Open Collector)	Amplifier/Drive Disable, Amplifier/Drive Enable
Maximum voltage	30V
Maximum current sink	100 ma
Position Capture (Latch) input	TTL (0 - +5V), 1 per servo axis pair
Active level	Rising edge (TTL high, > 2.4 VDC)
Minimum pulse duration	100 nanosecond
Maximum trigger frequency	1 KHz
Position Compare (trigger) output	TTL (0 - +5V), 1 per four servo axes
Active level	Programmable, default = TTL high
Minimum pulse duration	<1 nanosecond
Maximum repeat frequency	>1 MHz (programmable trigger modes)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Pulse Command Axis Specifications

Function	Open Stepper, Open Loop Stepper with Position Verification, Closed Loop Stepper, or Pulse Command Servo
Operating Modes	Position, Velocity, Contouring, Torque, and Gain
Trajectory Generator	Trapezoidal, Parabolic or S-Curve Independent Acceleration and Deceleration
Position Feedback	Incremental Encoder with Index (for closed loop stepper operation or position verification of an open loop stepper)
Position and Velocity Resolution	64 bit floating point
Step Outputs	Step/Direction or CW/CCW (software selectable), 50% duty cycle open collector drivers (max. 30V, 100ma current sink)
Step Rates (Software Selectable)	High Speed - 153 Steps/Sec. - 5.0M Steps/Sec. Medium Speed - 20 Steps/Sec. - 625K Steps/Sec. Low Speed - .1 Steps/Sec. – 78K Steps/Sec.
Position Feedback	Incremental Encoder with Index
Position and Velocity Resolution	32 bit
Encoder	
Encoder and Index Inputs	Differential or single ended, -25 to +25 vdc max.
Encoder Count Rate	To 20,000,000 Quadrature Counts/Sec.
Encoder Supply Voltage	+5 vdc or +12 vdc
Minimum Phase differential	200mV
Hardware Error Checking	Yes (differential encoder only)
Axis Inputs (Optically isolated)	Limit+, Limit -, Home, Drive Fault (shared with analog cmd. axis)
Device conducting	minimum voltage = 3V maximum voltage = 25V
Minimum current required	0.25 mA
Axis Outputs (Open Collector)	Driver Disable, Driver Enable, Full/Half Current
Maximum voltage	30V
Maximum current sink	100ma
Position Capture (Latch) input	TTL (0 - +5V)
Active level	Rising edge (TTL high, > 2.4 VDC)
Minimum pulse duration	100 nanosecond
Maximum re-trigger frequency	1 KHz
Position Compare output	TTL (0 - +5V), 1 per four servo axes
Active level	Programmable, default = TTL high
Minimum pulse duration	< nanosecond
Maximum re-trigger frequency	>1 MHz, (programmable trigger modes)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

General Purpose I/O Specifications

Digital Inputs	16 channels, TTL, Buffered (74LS541)
Active level	Programmable
TTL high level input min. voltage	2.0V
TTL high level input max. voltage	5.0V
TTL low level input min. voltage	0.0V
TTL low level input max. voltage	0.6V
Input termination (pull up/down)	None
Digital Outputs	16 channels, TTL, Buffered (74LS541)
Active level	Programmable
TTL low level current sink max.	24 mA
TTL high level curr. source max.	15 mA
TTL high level out. min. voltage	2.4V
TTL high level out. max. voltage	5.0V
TTL low level output min. voltage	0.0V
TTL low level out. max. voltage	0.5V
Analog Inputs	8 channels, 14 bit per channel
Input voltage range	-10.0V to +10.0V (standard), 0.0V to +4.0V (special order)
Nominal read rates	
From a Windows program	~200 usec* (results will vary depending PC configuration)
From on-board MCCL routine	~42 usec *

* results will vary based on the state of controller at time of A/D conversion.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Chapter

9

Connectors, I/O and Schematics

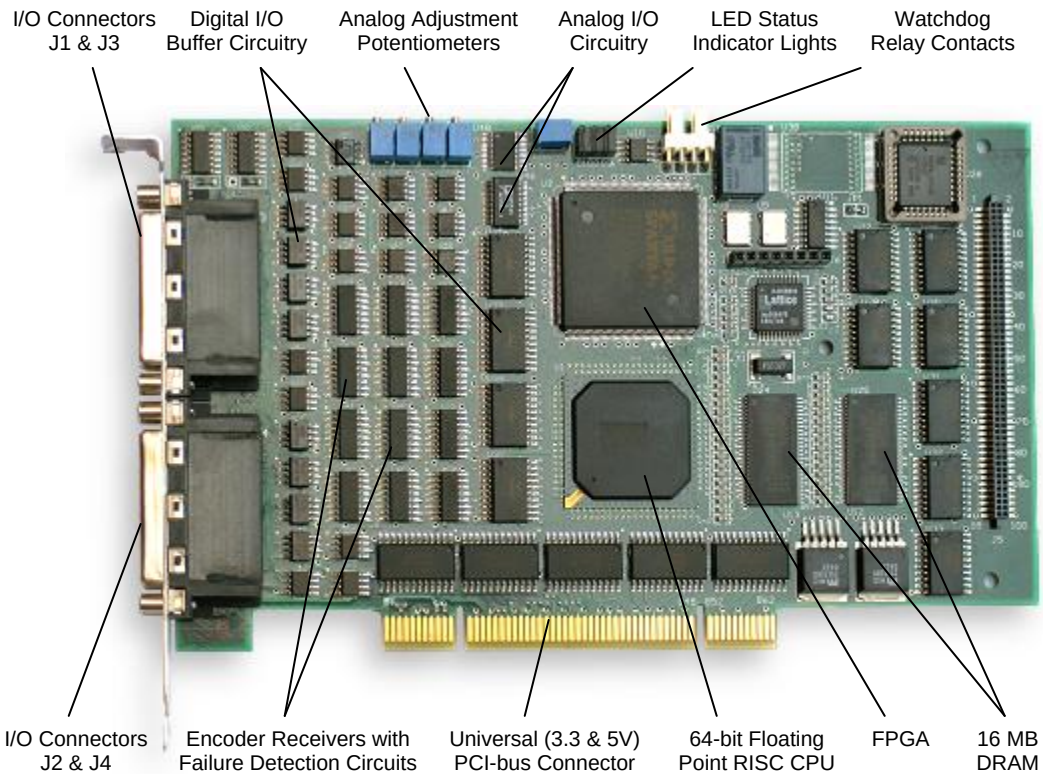


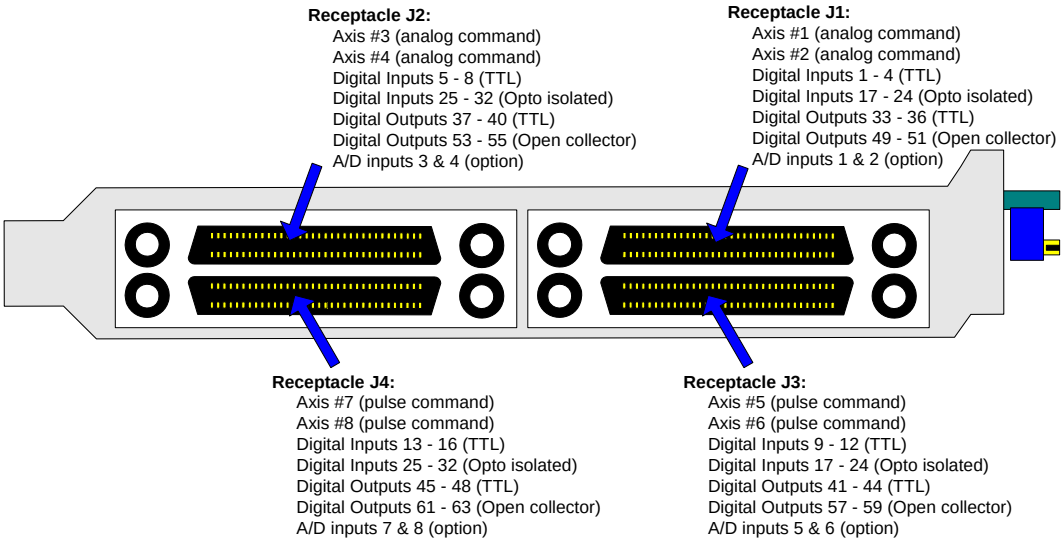
Figure 53. MultiFlex PCI 1000 Series Board Layout

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

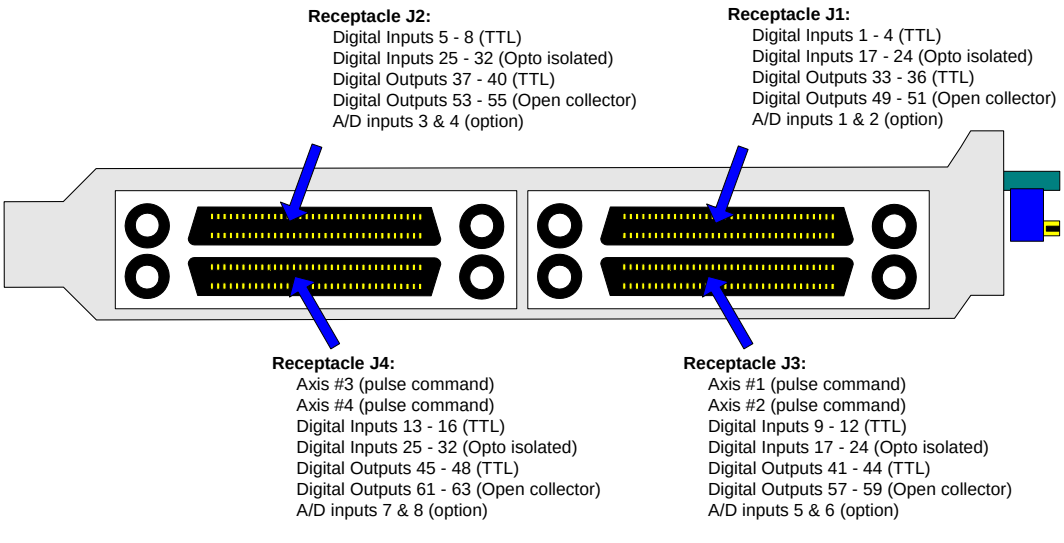
VHDCI Connectors

The controller uses four 68 contact SCSI VHDCI (Very High Density Cable Interconnect) receptacles for all signal connections. These high density shielded connectors (AMP P/N 787962-2) mate with widely available industry-standard cables that are often used in RAID (Redundant Array of Independent Disks) network servers.

MultiFlex PCI 1440 Connector Locations and Signal Assignments

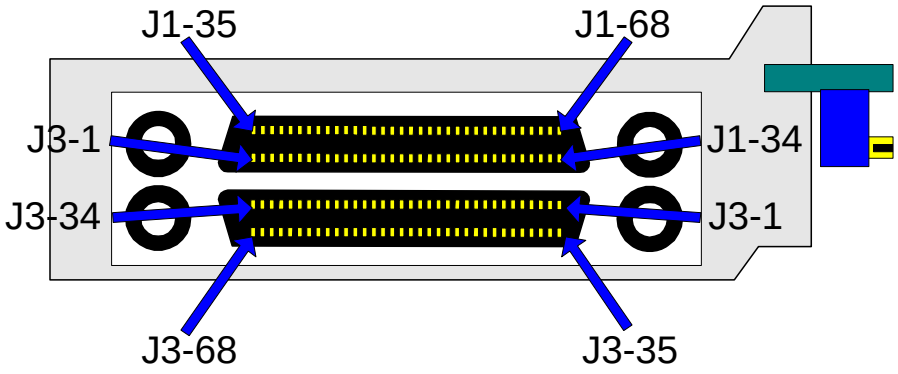


MultiFlex PCI 1040 Connector Locations and Signal Assignments



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

MFX-PCI VHDCI Connector pin numbering
(only J1 and J3 shown)



Prior to inserting the VHDCI cable 'plug' connector into the controller-mounted VHDCI receptacle **always visually inspect the contacts of both the plug and receptacle**. Contacts that are visibly out of alignment may damage the contacts of the mating connector or cause a wiring short circuit which could damage the controller (voiding the warranty).

If any contacts are visibly out of alignment, contact PMC Tech Support at support@pmccorp.com.

VHDCI SCSI mating connectors and cables

J1 - J4 Mating connector:
Cable Plug (offset): Amp P/N 787801-1
Backshell (offset): Molex P/N 788362-1

VHDCI SCSI cables supplied by PMC:
PMC P/N: **CBL-VH68-6**: VHDCI male to HD68 (SCSI-II) male, 6' (~2M) (compatible with ADAM 3968 Wiring Terminal Board).

Controller Status LED Indicators

LED #	Color	Description
D1	Green	+3.3 VDC PC logic supply OK
D2	Yellow	Controller reset active (watchdog relay de-energized)
D3	Green	Run (motion control code executing, watchdog relay energized)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Controller Potentiometers

POT1 - Axis #1 offset adjustment potentiometer
Adjust the analog command output offset of axis #1 (measured between J1 pins 1 and 35).Maximum adjustment range is approximately 1.0 volt.

POT2 - Axis #2 offset adjustment potentiometer
Adjust the analog command output offset of axis #2 (measured between J2 pins 1 and 35).Maximum adjustment range is approximately 1.0 volt.

POT3 - Axis #3 offset adjustment potentiometer
Adjust the analog command output offset of axis #3 (measured between J3 pins 1 and 35).Maximum adjustment range is approximately 1.0 volt.

POT4 - Axis #4 offset adjustment potentiometer
Adjust the analog command output offset of axis #4 (measured between J4 pins 1 and 35).Maximum adjustment range is approximately 1.0 volt.

POT5 - Reserved for factory use only (Analog input reference adjustment)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Connector Pinout – MultiFlex PCI 1440

Connector J1 (Analog Command Axes 1 & 2, Dig. inputs 1-4, Dig. outputs 1-4, A/D inputs 1 & 2)

VHDCI Pin #	Digital I/O Channel	Circuit Type	Description (default configuration)	ADAM-3968 Pin #
J1 - 1			Axis 1 Servo Command (+/- 10V)	1
J1 – 35			Axis 1 return / Analog Ground	35
J1 – 2			Axis 2 Servo Command (+/- 10V)	2
J1 – 36			Axis 2 return / Analog Ground	36
J1 – 3				3
J1 – 37				37
J1 – 4	53	Output - open collector driver	Axis 2 Amp. Enable output	4
J1 – 38			+5 VDC	38
J1 – 5	49	Output - open collector driver	Axis 1 Amp. Enable output	5
J1 – 39			+5 VDC	39
J1 – 6	51	Output - open collector driver	#1 PWM out / #1 Direction / #5 Full Current out	6
J1 – 40			+5 VDC	40
J1 – 7	55	Output - open collector driver	#2 PWM out / #2 Direction / #6 Full Current out	7
J1 – 41			+5 VDC	41
J1 – 8				8
J1 – 42				42
J1 – 9			+12 VDC	9
J1 – 43			+12 VDC	43
J1 – 10			Axis 1/2 Encoder Ref. (1.5V)	10
J1 – 44			Axis 1/2 Encoder Ref. (1.5V)	44
J1 – 11			Axis 1 Encoder Phase A+	11
J1 – 45			Axis 1 Encoder Phase A-	45
J1 – 12			Axis 1 Encoder Phase B+	12
J1 – 46			Axis 1 Encoder Phase B-	46
J1 – 13			Axis 1 Encoder Phase Z+	13
J1 – 47			Axis 1 Encoder Phase Z -	47
J1 – 14			Axis 2 Encoder Phase A+	14
J1 – 48			Axis 2 Encoder Phase A-	48
J1 – 15			Axis 2 Encoder Phase B+	15
J1 – 49			Axis 2 Encoder Phase B-	49
J1 – 16			Axis 2 Encoder Phase Z+	16
J1 – 50			Axis 2 Encoder Phase Z -	50
J1 – 17	20	Input - opto isolated (bi-directional)	Axis 1 Amp. Fault input (shared by Axis 5 Drive	17
J1 – 51			Axis 1/5 Amp Fault supply / return	51
J1 – 18	24	Input - opto isolated (bi-directional)	Axis 2 Amp. Fault input (shared by Axis 6 Drive	18
J1 – 52			Axis 2/6 Amp Fault supply / return	52
J1 – 19	33	Output - TTL	Digital Out #1 / Axis 1 - 4 Compare	19
J1 – 53			+5 VDC	53
J1 – 20	34	Output - TTL	Digital Output #2	20
J1 – 54			+5 VDC	54
J1 – 21	35	Output - TTL	Digital Output #3	21
J1 – 55			+5 VDC	55
J1 – 22	36	Output - TTL	Digital Output #4	22
J1 – 56			+5 VDC	56
J1 – 23	1	Input - TTL	Dig. In. #1 / Axis 1 & 2 Position Capture (Latch)	23
J1 – 57			Ground	57
J1 – 24	2	Input - TTL	Digital Input #2	24
J1 – 58			Ground	58
J1 – 25	3	Input - TTL	Digital Input #3	25
J1 – 59			Ground	59
J1 – 26	4	Input - TTL	Digital Input #4	26
J1 – 60			Ground	60
J1 – 27	17	Input - opto isolated (bi-directional)	Axis 1 Coarse Home (shared by Axis 5 Home)	27
J1 – 61			Axis 1/5 Coarse Home / Home return / supply	61
J1 – 28	21	Input - opto isolated (bi-directional)	Axis 2 Coarse Home (shared by Axis 6 Home)	28
J1 – 62			Axis 2/6 Coarse Home / Home return / supply	62
J1 – 29	18	Input - opto isolated (bi-directional)	Axis 1Limit + (shared by Axis 5 Limit +)	29
J1 – 63			Axis 1/5 Limit + return / supply	63
J1 – 30	22	Input - opto isolated (bi-directional)	Axis 2 Limit + (shared by Axis 6 Limit +)	30
J1 – 64			Axis 2/6 Limit + return / supply	64
J1 – 31	19	Input - opto isolated (bi-directional)	Axis 1 Limit - (shared by Axis 5 Limit -)	31
J1 – 65			Axis 1/5 Limit - return / supply	65
J1 – 32	23	Input - opto isolated (bi-directional)	Axis 2 Limit - (shared by Axis 6 Limit -)	32
J1 – 66			Axis 2/6 Limit - return / supply	66
J1 – 33			Analog Input #1 (option)	33
J1 – 67			Analog In #1 return / An. Ground	67
J1 – 34			Analog Input #2 (option)	34
J1 – 68			Analog In #2 return / An. Ground	68

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Connector Pinout – MultiFlex PCI 1440 (continued)

Connector J2 (Analog Command Axes 3 & 4, Dig. inputs 5-8, Dig. outputs 5-8, A/D inputs 3 & 4)

VHDCI Pin #	Digital I/O Channel	Circuit Type	Description (default configuration)	Adam-3968 Pin #
J2 - 1			Axis 3 Servo Command (+/- 10V)	1
J2 – 35			Axis 3 return / Analog Ground	35
J2 – 2			Axis 4 Servo Command (+/- 10V)	2
J2 – 36			Axis 4 return / Analog Ground	36
J2 – 3				3
J2 – 37				37
J2 – 4	61	Output - open collector driver	Axis 4 Amp. Enable output	4
J2 – 38			+5 VDC	38
J2 – 5	57	Output - open collector driver	Axis 3 Amp. Enable output	5
J2 – 39			+5 VDC	39
J2 – 6	59	Output - open collector driver	#3 PWM out / #3 Direction / #7 Full Current out	6
J2 – 40			+5 VDC	40
J2 – 7	63	Output - open collector driver	#4 PWM out / #4 Direction / #8 Full Current out	7
J2 – 41			+5 VDC	41
J2 – 8				8
J2 – 42				42
J2 – 9			+12 VDC	9
J2 – 43			+12 VDC	43
J2 – 10			Axis 3/4 Encoder Ref. (1.5V)	10
J2 – 44			Axis 3/4 Encoder Ref. (1.5V)	44
J2 – 11			Axis 3 Encoder Phase A+	11
J2 – 45			Axis 3 Encoder Phase A-	45
J2 – 12			Axis 3 Encoder Phase B+	12
J2 – 46			Axis 3 Encoder Phase B-	46
J2 – 13			Axis 3 Encoder Phase Z+	13
J2 – 47			Axis 3 Encoder Phase Z -	47
J2 – 14			Axis 4 Encoder Phase A+	14
J2 – 48			Axis 4 Encoder Phase A-	48
J2 – 15			Axis 4 Encoder Phase B+	15
J2 – 49			Axis 4 Encoder Phase B-	49
J2 – 16			Axis 4 Encoder Phase Z+	16
J2 – 50			Axis 4 Encoder Phase Z -	50
J2 – 17	28	Input - opto isolated (bi-directional)	Axis 3 Amp. Fault input (shared by Axis 7 Drive	17
J2 – 51			Axis 4/8 Amp Fault supply / return	51
J2 – 18	32	Input - opto isolated (bi-directional)	Axis 4 Amp. Fault input (shared by Axis 8 Drive	18
J2 – 52			Axis 4/8 Amp Fault supply / return	52
J2 – 19	37	Output - TTL	Digital Output #5	19
J2 – 53			+5 VDC	53
J2 – 20	38	Output - TTL	Digital Output #6	20
J2 – 54			+5 VDC	54
J2 – 21	39	Output - TTL	Digital Output #7	21
J2 – 55			+5 VDC	55
J2 – 22	40	Output - TTL	Digital Output #8	22
J2 – 56			+5 VDC	56
J2 – 23	5	Input - TTL	Dig. In. #5 / Axis 3 & 4 Position Capture (Latch)	23
J2 – 57			Ground	57
J2 – 24	6	Input - TTL	Digital Input #6	24
J2 – 58			Ground	58
J2 – 25	7	Input - TTL	Digital Input #7	25
J2 – 59			Ground	59
J2 – 26	8	Input - TTL	Digital Input #8	26
J2 – 60			Ground	60
J2 – 27	25	Input - opto isolated (bi-directional)	Axis 3 Coarse Home (shared by Axis 7 Home)	27
J2 – 61			Axis 3/7 Coarse Home / Home return / supply	61
J2 – 28	29	Input - opto isolated (bi-directional)	Axis 4 Coarse Home / Axis 8 Stepper Home	28
J2 – 62			Axis 4 Coarse Home (shared by Axis 8 Home)	62
J2 – 29	26	Input - opto isolated (bi-directional)	Axis 3 Limit + (shared by Axis 7 Limit +)	29
J2 – 63			Axis 3/7 Limit + return / supply	63
J2 – 30	30	Input - opto isolated (bi-directional)	Axis 4 Limit + (shared by Axis 8 Limit +)	30
J2 – 64			Axis 4/8 Limit + return / supply	64
J2 – 31	27	Input - opto isolated (bi-directional)	Axis 3 Limit - (shared by Axis 7 Limit -)	31
J2 – 65			Axis 3/7 Limit - return / supply	65
J2 – 32	31	Input - opto isolated (bi-directional)	Axis 4 Limit - (shared by Axis 8 Limit -)	32
J2 – 66			Axis 4/8 Limit - return / supply	66
J2 – 33			Analog Input #3 (option)	33
J2 – 67			Analog In #3 return / An. Ground	67
J2 – 34			Analog Input #4 (option)	34
J2 – 68			Analog In #4 return / An. Ground	68

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Connector Pinout – MultiFlex PCI 1440 (continued)

Connector J3 (Pulse Command Axes 5 & 6, Dig. inputs 9-12, Dig. outputs 9-12, A/D inputs 5 & 6)

VHDCI Pin #	Digital I/O Channel	Circuit Type	Description (default configuration)	Adam-3968 Pin #
J3 - 1	50	Output - open collector driver	Axis 5 All Windings Off output	1
J3 – 35			+5 VDC	35
J3 – 2			Axis 5 Step / CCW Pulse	2
J3 – 36			+5 VDC	36
J3 – 3			Axis 5 Direction / CW Pulse	3
J3 – 37			+5 VDC	37
J3 – 4	51	Output - open collector driver	Axis 5 Full Current / Axis 1 Unipolar Direction output	4
J3 – 38			+5 VDC	38
J3 – 5	55	Output - open collector driver	Axis 6 Full Current / Axis 2 Unipolar Direction output	5
J3 – 39			+5 VDC	39
J3 – 6	54	Output - open collector driver	Axis 6 All Windings Off output	6
J3 – 40			+5 VDC	40
J3 – 7			Axis 6 Step / CCW Pulse	7
J3 – 41			+5 VDC	41
J3 – 8			Axis 6 Direction / CW Pulse	8
J3 – 42			+5 VDC	42
J3 – 9			+12 VDC	9
J3 – 43			+12 VDC	43
J3 – 10			Axis 5/6 Encoder Ref. (1.5V)	10
J3 – 44			Axis 5/6 Encoder Ref. (1.5V)	44
J3 – 11			Axis 5 Encoder Phase A+	11
J3 – 45			Axis 5 Encoder Phase A-	45
J3 – 12			Axis 5 Encoder Phase B+	12
J3 – 46			Axis 5 Encoder Phase B-	46
J3 – 13			Axis 5 Encoder Phase Z+	13
J3 – 47			Axis 5 Encoder Phase Z -	47
J3 – 14			Axis 6 Encoder Phase A+	14
J3 – 48			Axis 6 Encoder Phase A-	48
J3 – 15			Axis 6 Encoder Phase B+	15
J3 – 49			Axis 6 Encoder Phase B-	49
J3 – 16			Axis 6 Encoder Phase Z+	16
J3 – 50			Axis 6 Encoder Phase Z -	50
J3 – 17	20	Input - opto isolated (bi-directional)	Axis 5 Drive Fault input (shared by Axis 1 Amp	17
J3 – 51			Axis 1/5 Amp Fault supply / return	51
J3 – 18	24	Input - opto isolated (bi-directional)	Axis 6 Drive Fault input (shared by Axis 2 Amp	18
J3 – 52			Axis 2/6 Amp Fault supply / return	52
J3 – 19	41	Output - TTL	Digital Out #9 / Axis 5 - 8 Position Compare	19
J3 – 53			+5 VDC	53
J3 – 20	42	Output - TTL	Digital Output #10	20
J3 – 54			+5 VDC	54
J3 – 21	43	Output - TTL	Digital Output #11	21
J3 – 55			+5 VDC	55
J3 – 22	44	Output - TTL	Digital Output #12	22
J3 – 56			+5 VDC	56
J3 – 23	9	Input - TTL	Dig. In. #9 / Axis 5 & 6 Position Capture (Latch)	23
J3 – 57			Ground	57
J3 – 24	10	Input - TTL	Digital Input #10	24
J3 – 58			Ground	58
J3 – 25	11	Input - TTL	Digital Input #11	25
J3 – 59			Ground	59
J3 – 26	12	Input - TTL	Digital Input #12	26
J3 – 60			Ground	60
J3 – 27	17	Input - opto isolated (bi-directional)	Axis 5 Home (shared by Axis 1 Coarse Home)	27
J3 – 61			Axis 1/5 Coarse Home / Home return / supply	61
J3 – 28	21	Input - opto isolated (bi-directional)	Axis 6 Home (shared by Axis 6 Coarse Home)	28
J3 – 62			Axis 2/6 Coarse Home / Home return / supply	62
J3 – 29	18	Input - opto isolated (bi-directional)	Axis 5 Limit + (shared by Axis 1 Limit +)	29
J3 – 63			Axis 1/5 Limit + return / supply	63
J3 – 30	22	Input - opto isolated (bi-directional)	Axis 6 Limit + (shared by Axis 2 Limit +)	30
J3 – 64			Axis 2/6 Limit + return / supply	64
J3 – 31	19	Input - opto isolated (bi-directional)	Axis 5 Limit - (shared by Axis 1 Limit -)	31
J3 – 65			Axis 1/5 Limit - return / supply	65
J3 – 32	23	Input - opto isolated (bi-directional)	Axis 6 Limit - (shared by Axis 2 Limit -)	32
J3 – 66			Axis 2/6 Limit - return / supply	66
J3 – 33			Analog Input #5 (option)	33
J3 – 67			Analog In #5 return / An. Ground	67
J3 – 34			Analog Input #6 (option)	34
J3 – 68			Analog In #6 return / An. Ground	68

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Connector Pinout – MultiFlex PCI 1440 (continued)

Connector J4 (Pulse Command Axes 7 & 8, Dig. inputs 13-16, Dig. outputs 13-16, A/D inputs 7 & 8)

VHDCI Pin #	Digital I/O Channel	Circuit Type	Description (default configuration)	Adam-3968 Pin #
J4 - 1	58	Output – open-collector driver	Axis 7 All Windings Off output	1
J4 – 35			+5 VDC	35
J4 – 2			Axis 7 Step / CCW Pulse	2
J4 – 36			+5 VDC	36
J4 – 3			Axis 7 Direction / CW Pulse	3
J4 – 37			+5 VDC	37
J4 – 4	59	Output - open collector driver	Axis 7 Full Current / Axis 3 Unipolar Direction output	4
J4 – 38			+5 VDC	38
J4 – 5	63	Output - open collector driver	Axis 8 Full Current / Axis 4 Unipolar Direction output	5
J4 – 39			+5 VDC	39
J4 – 6	62	Output - open collector driver	Axis 8 All Windings Off output	6
J4 – 40			+5 VDC	40
J4 – 7			Axis 8 Step / CCW Pulse	7
J4 – 41			+5 VDC	41
J4 – 8			Axis 8 Direction / CW Pulse	8
J4 – 42			+5 VDC	42
J4 – 9			+12 VDC	9
J4 – 43			+12 VDC	43
J4 – 10			Axis 7/8 Encoder Ref. (1.5V)	10
J4 – 44			Axis 7/8 Encoder Ref. (1.5V)	44
J4 – 11			Axis 7 Encoder Phase A+	11
J4 – 45			Axis 7 Encoder Phase A-	45
J4 – 12			Axis 7 Encoder Phase B+	12
J4 – 46			Axis 7 Encoder Phase B-	46
J4 – 13			Axis 7 Encoder Phase Z+	13
J4 – 47			Axis 7 Encoder Phase Z -	47
J4 – 14			Axis 8 Encoder Phase A+	14
J4 – 48			Axis 8 Encoder Phase A-	48
J4 – 15			Axis 8 Encoder Phase B+	15
J4 – 49			Axis 8 Encoder Phase B-	49
J4 – 16			Axis 8 Encoder Phase Z+	16
J4 – 50			Axis 8 Encoder Phase Z -	50
J4 – 17	28	Input - opto isolated (bi-directional)	Axis 7 Drive Fault input (shared by Axis 3 Amp	17
J4 – 51			Axis 3/7 Amp Fault supply / return	51
J4 – 18	32	Input - opto isolated (bi-directional)	Axis 8 Amp. Fault input (shared by Axis 8 Amp	18
J4 – 52			Axis 4/8 Amp Fault supply / return	52
J4 – 19	45	Output - TTL	Digital Output #13	19
J4 – 53			+5 VDC	53
J4 – 20	46	Output - TTL	Digital Output #14	20
J4 – 54			+5 VDC	54
J4 – 21	47	Output - TTL	Digital Output #15	21
J4 – 55			+5 VDC	55
J4 – 22	48	Output - TTL	Digital Output #16	22
J4 – 56			+5 VDC	56
J4 – 23	13	Input - TTL	Dig. In. #13 / Axis 7 & 8 Position Capture (Latch)	23
J4 – 57			Ground	57
J4 – 24	14	Input - TTL	Digital Input #14	24
J4 – 58			Ground	58
J4 – 25	15	Input - TTL	Digital Input #15	25
J4 – 59			Ground	59
J4 – 26	16	Input - TTL	Digital Input #16	26
J4 – 60			Ground	60
J4 – 27	25	Input - opto isolated (bi-directional)	Axis 7 Home (shared by Axis 3 Coarse Home)	27
J4 – 61			Axis 3/7 Coarse Home / Home return / supply	61
J4 – 28	29	Input - opto isolated (bi-directional)	Axis 8 Home (shared by Axis 4 Coarse Home)	28
J4 – 62			Axis 4/8 Coarse Home / Home return / supply	62
J4 – 29	26	Input - opto isolated (bi-directional)	Axis 7 Limit + (shared by Axis 3 Limit +)	29
J4 – 63			Axis 3/7 Limit + return / supply	63
J4 – 30	30	Input - opto isolated (bi-directional)	Axis 8 Limit + (shared by Axis 4 Limit +)	30
J4 – 64			Axis 4/8 Limit + return / supply	64
J4 – 31	27	Input - opto isolated (bi-directional)	Axis 7 Limit - (shared by Axis 7 Limit -)	31
J4 – 65			Axis 3/7 Limit - return / supply	65
J4 – 32	31	Input - opto isolated (bi-directional)	Axis 8 Limit - (shared by Axis 8 Limit -)	32
J4 – 66			Axis 4/8 Limit - return / supply	66
J4 – 33			Analog Input #7 (option)	33
J4 – 67			Analog In #7 return / An. Ground	67
J4 – 34			Analog Input #8 (option)	34
J4 – 68			Analog In #8 return / An. Ground	68

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Connector Pinout – MultiFlex PCI 1040

Connector J1 (Digital inputs 1-4, Digital outputs 1-4, A/D inputs 1 & 2)

VHDCI Pin #	Digital I/O Channel	Circuit Type	Description (default configuration)	Adam-3968 Pin #
J1 - 1				1
J1 – 35				35
J1 – 2				2
J1 – 36				36
J1 – 3				3
J1 – 37				37
J1 – 4	53	Output - open collector driver		4
J1 – 38			+5 VDC	38
J1 – 5	49	Output - open collector driver		5
J1 – 39			+5 VDC	39
J1 – 6	51	Output - open collector driver	Axis 1 Full Current output	6
J1 – 40			+5 VDC	40
J1 – 7	55	Output - open collector driver	Axis 2 Full Current output	7
J1 – 41			+5 VDC	41
J1 – 8				8
J1 – 42				42
J1 – 9			+12 VDC	9
J1 – 43			+12 VDC	43
J1 – 10				10
J1 – 44				44
J1 – 11				11
J1 – 45				45
J1 – 12				12
J1 – 46				46
J1 – 13				13
J1 – 47				47
J1 – 14				14
J1 – 48				48
J1 – 15				15
J1 – 49				49
J1 – 16				16
J1 – 50				50
J1 – 17				17
J1 – 51				51
J1 – 18				18
J1 – 52				52
J1 – 19	33	Output - TTL	Digital Out #1	19
J1 – 53			+5 VDC	53
J1 – 20	34	Output - TTL	Digital Output #2	20
J1 – 54			+5 VDC	54
J1 – 21	35	Output - TTL	Digital Output #3	21
J1 – 55			+5 VDC	55
J1 – 22	36	Output - TTL	Digital Output #4	22
J1 – 56			+5 VDC	56
J1 – 23	1	Input - TTL	Digital Input #1	23
J1 – 57			Ground	57
J1 – 24	2	Input - TTL	Digital Input #2	24
J1 – 58			Ground	58
J1 – 25	3	Input - TTL	Digital Input #3	25
J1 – 59			Ground	59
J1 – 26	4	Input - TTL	Digital Input #4	26
J1 – 60			Ground	60
J1 – 27				27
J1 – 61				61
J1 – 28				28
J1 – 62				62
J1 – 29				29
J1 – 63				63
J1 – 30				30
J1 – 64				64
J1 – 31				31
J1 – 65				65
J1 – 32				32
J1 – 66				66
J1 – 33			Analog Input #1 (option)	33
J1 – 67			Analog In #1 return / An. Ground	67
J1 – 34			Analog Input #2 (option)	34
J1 – 68			Analog In #2 return / An. Ground	68

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Connector Pinout – MultiFlex PCI 1040 (continued)

Connector J2 (Digital inputs 5-8, Digital outputs 5-8, A/D inputs 3 & 4)

VHDCI Pin #	Digital I/O Channel	Circuit Type	Description (default configuration)	Adam-3968 Pin #
J2 - 1				1
J2 – 35				35
J2 – 2				2
J2 – 36				36
J2 – 3				3
J2 – 37				37
J2 – 4	61	Output - open collector driver		4
J2 – 38			+5 VDC	38
J2 – 5	57	Output - open collector driver		5
J2 – 39			+5 VDC	39
J2 – 6	59	Output - open collector driver	Axis 3 Full Current output	6
J2 – 40			+5 VDC	40
J2 – 7	63	Output - open collector driver	Axis 4 Full Current output	7
J2 – 41			+5 VDC	41
J2 – 8				8
J2 – 42				42
J2 – 9			+12 VDC	9
J2 – 43			+12 VDC	43
J2 – 10				10
J2 – 44				44
J2 – 11				11
J2 – 45				45
J2 – 12				12
J2 – 46				46
J2 – 13				13
J2 – 47				47
J2 – 14				14
J2 – 48				48
J2 – 15				15
J2 – 49				49
J2 – 16				16
J2 – 50				50
J2 – 17				17
J2 – 51				51
J2 – 18				18
J2 – 52				52
J2 – 19	37	Output - TTL	Digital Output #5	19
J2 – 53			+5 VDC	53
J2 – 20	38	Output - TTL	Digital Output #6	20
J2 – 54			+5 VDC	54
J2 – 21	39	Output - TTL	Digital Output #7	21
J2 – 55			+5 VDC	55
J2 – 22	40	Output - TTL	Digital Output #8	22
J2 – 56			+5 VDC	56
J2 – 23	5	Input - TTL	Digital Input #5	23
J2 – 57			Ground	57
J2 – 24	6	Input - TTL	Digital Input #6	24
J2 – 58			Ground	58
J2 – 25	7	Input - TTL	Digital Input #7	25
J2 – 59			Ground	59
J2 – 26	8	Input - TTL	Digital Input #8	26
J2 – 60			Ground	60
J2 – 27				27
J2 – 61				61
J2 – 28				28
J2 – 62				62
J2 – 29				29
J2 – 63				63
J2 – 30				30
J2 – 64				64
J2 – 31				31
J2 – 65				65
J2 – 32				32
J2 – 66				66
J2 – 33			Analog Input #3 (option)	33
J2 – 67			Analog In #3 return / An. Ground	67
J2 – 34			Analog Input #4 (option)	34
J2 – 68			Analog In #4 return / An. Ground	68

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Connector Pinout – MultiFlex PCI 1040 (continued)

Connector J3 (Pulse Command Axes 1 & 2, Dig. I/O 9-12, An. in 5 & 6)

VHDCI Pin #	Digital I/O Channel	Circuit Type	Description (default configuration)	Adam-3968 Pin #
J3 - 1	50	Output - open collector driver	Axis 1 All Windings Off output	1
J3 – 35			+5 VDC	35
J3 – 2			Axis 1 Step / CCW Pulse	2
J3 – 36			+5 VDC	36
J3 – 3			Axis 1 Direction / CW Pulse	3
J3 – 37			+5 VDC	37
J3 – 4	51	Output - open collector driver	Axis 1 Full Current	4
J3 – 38			+5 VDC	38
J3 – 5	55	Output - open collector driver	Axis 2 Full Current	5
J3 – 39			+5 VDC	39
J3 – 6	54	Output - open collector driver	Axis 2 All Windings Off output	6
J3 – 40			+5 VDC	40
J3 – 7			Axis 2 Step / CCW Pulse	7
J3 – 41			+5 VDC	41
J3 – 8			Axis 2 Direction / CW Pulse	8
J3 – 42			+5 VDC	42
J3 – 9			+12 VDC	9
J3 – 43			+12 VDC	43
J3 – 10			Axis 1/2 Encoder Ref. (1.5V)	10
J3 – 44			Axis 1/2 Encoder Ref. (1.5V)	44
J3 – 11			Axis 1 Encoder Phase A+	11
J3 – 45			Axis 1 Encoder Phase A-	45
J3 – 12			Axis 1 Encoder Phase B+	12
J3 – 46			Axis 1 Encoder Phase B-	46
J3 – 13			Axis 1 Encoder Phase Z+	13
J3 – 47			Axis 1 Encoder Phase Z -	47
J3 – 14			Axis 2 Encoder Phase A+	14
J3 – 48			Axis 2 Encoder Phase A-	48
J3 – 15			Axis 2 Encoder Phase B+	15
J3 – 49			Axis 2 Encoder Phase B-	49
J3 – 16			Axis 2 Encoder Phase Z+	16
J3 – 50			Axis 2 Encoder Phase Z -	50
J3 – 17	20	Input - opto isolated (bi-directional)	Axis 1 Drive Fault input	17
J3 – 51			Axis 1 Drive Fault supply / return	51
J3 – 18	24	Input - opto isolated (bi-directional)	Axis 2 Drive Fault input	18
J3 – 52			Axis 2 Drive Fault supply / return	52
J3 – 19	41	Output - TTL	Digital Out #9 / Axis 1 - 4 Position Compare	19
J3 – 53			+5 VDC	53
J3 – 20	42	Output - TTL	Digital Output #10	20
J3 – 54			+5 VDC	54
J3 – 21	43	Output - TTL	Digital Output #11	21
J3 – 55			+5 VDC	55
J3 – 22	44	Output - TTL	Digital Output #12	22
J3 – 56			+5 VDC	56
J3 – 23	9	Input - TTL	Dig. In. #9 / Axis 1 & 2 Position Capture (Latch)	23
J3 – 57			Ground	57
J3 – 24	10	Input - TTL	Digital Input #10	24
J3 – 58			Ground	58
J3 – 25	11	Input - TTL	Digital Input #11	25
J3 – 59			Ground	59
J3 – 26	12	Input - TTL	Digital Input #12	26
J3 – 60			Ground	60
J3 – 27	17	Input - opto isolated (bi-directional)	Axis 1 Home	27
J3 – 61			Axis 1 Home return / supply	61
J3 – 28	21	Input - opto isolated (bi-directional)	Axis 2 Home	28
J3 – 62			Axis 2 Home return / supply	62
J3 – 29	18	Input - opto isolated (bi-directional)	Axis 1 Limit +	29
J3 – 63			Axis 1 Limit + return / supply	63
J3 – 30	22	Input - opto isolated (bi-directional)	Axis 2 Limit +	30
J3 – 64			Axis 2 Limit + return / supply	64
J3 – 31	19	Input - opto isolated (bi-directional)	Axis 1 Limit -	31
J3 – 65			Axis 1 Limit - return / supply	65
J3 – 32	23	Input - opto isolated (bi-directional)	Axis 2 Limit -	32
J3 – 66			Axis 2 Limit - return / supply	66
J3 – 33			Analog Input #5 (option)	33
J3 – 67			Analog In #5 return / An. Ground	67
J3 – 34			Analog Input #6 (option)	34
J3 – 68			Analog In #6 return / An. Ground	68

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Connector Pinout – MultiFlex PCI 1040 (continued)

Connector J4 (Pulse Command Axes 3 & 4, Dig. I/O 13-16, An. in 7 & 8)

VHDCI Pin #	Digital I/O Channel	Circuit Type	Description (default configuration)	Adam-3968 Pin #
J4 - 1	58	Output - open collector driver	Axis 3 All Windings Off output	1
J4 – 35			+5 VDC	35
J4 – 2			Axis 3 Step / CCW Pulse	2
J4 – 36			+5 VDC	36
J4 – 3			Axis 3 Direction / CW Pulse	3
J4 – 37			+5 VDC	37
J4 – 4	59	Output - open collector driver	Axis 3 Full Current	4
J4 – 38			+5 VDC	38
J4 – 5	63	Output - open collector driver	Axis 4 Full Current	5
J4 – 39			+5 VDC	39
J4 – 6	62	Output - open collector driver	Axis 4 All Windings Off output	6
J4 – 40			+5 VDC	40
J4 – 7			Axis 4 Step / CCW Pulse	7
J4 – 41			+5 VDC	41
J4 – 8			Axis 4 Direction / CW Pulse	8
J4 – 42			+5 VDC	42
J4 – 9			+12 VDC	9
J4 – 43			+12 VDC	43
J4 – 10			Axis 3/4 Encoder Ref. (1.5V)	10
J4 – 44			Axis 3/4 Encoder Ref. (1.5V)	44
J4 – 11			Axis 3 Encoder Phase A+	11
J4 – 45			Axis 3 Encoder Phase A-	45
J4 – 12			Axis 3 Encoder Phase B+	12
J4 – 46			Axis 3 Encoder Phase B-	46
J4 – 13			Axis 3 Encoder Phase Z+	13
J4 – 47			Axis 3 Encoder Phase Z -	47
J4 – 14			Axis 4 Encoder Phase A+	14
J4 – 48			Axis 4 Encoder Phase A-	48
J4 – 15			Axis 4 Encoder Phase B+	15
J4 – 49			Axis 4 Encoder Phase B-	49
J4 – 16			Axis 4 Encoder Phase Z+	16
J4 – 50			Axis 4 Encoder Phase Z -	50
J4 – 17	28	Input - opto isolated (bi-directional)	Axis 3 Drive Fault input	17
J4 – 51			Axis 3 Drive Fault supply / return	51
J4 – 18	32	Input - opto isolated (bi-directional)	Axis 4 Drive Fault input	18
J4 – 52			Axis 4 Drive Fault supply / return	52
J4 – 19	45	Output - TTL	Digital Output #13	19
J4 – 53			+5 VDC	53
J4 – 20	46	Output - TTL	Digital Output #14	20
J4 – 54			+5 VDC	54
J4 – 21	47	Output - TTL	Digital Output #15	21
J4 – 55			+5 VDC	55
J4 – 22	48	Output - TTL	Digital Output #16	22
J4 – 56			+5 VDC	56
J4 – 23	13	Input - TTL	Dig. In. #13 / Axis 3 & 4 Position Capture (Latch)	23
J4 – 57			Ground	57
J4 – 24	14	Input - TTL	Digital Input #14	24
J4 – 58			Ground	58
J4 – 25	15	Input - TTL	Digital Input #15	25
J4 – 59			Ground	59
J4 – 26	16	Input - TTL	Digital Input #16	26
J4 – 60			Ground	60
J4 – 27	25	Input - opto isolated (bi-directional)	Axis 3 Home	27
J4 – 61			Axis 3 Home return / supply	61
J4 – 28	29	Input - opto isolated (bi-directional)	Axis 4 Home	28
J4 – 62			Axis 4 Home return / supply	62
J4 – 29	26	Input - opto isolated (bi-directional)	Axis 3 Limit +	29
J4 – 63			Axis 3 Limit + return / supply	63
J4 – 30	30	Input - opto isolated (bi-directional)	Axis 4 Limit +	30
J4 – 64			Axis 4 Limit + return / supply	64
J4 – 31	27	Input - opto isolated (bi-directional)	Axis 3 Limit -	31
J4 – 65			Axis 3 Limit - return / supply	65
J4 – 32	31	Input - opto isolated (bi-directional)	Axis 4 Limit -	32
J4 – 66			Axis 4 Limit - return / supply	66
J4 – 33			Analog Input #7 (option)	33
J4 – 67			Analog In #7 return / An. Ground	67
J4 – 34			Analog Input #8 (option)	34
J4 – 68			Analog In #8 return / An. Ground	68

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

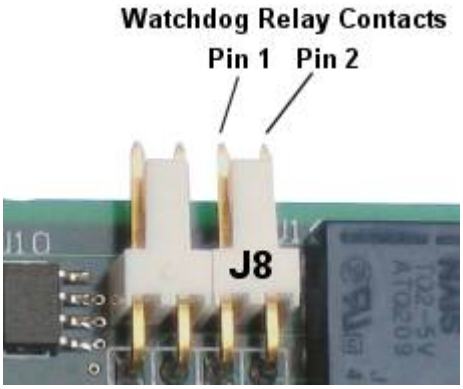
Other Connectors

Connector J5 - Reserved for factory use

Connector J7 - Reserved for factory use

Connector J8 - Watchdog relay contacts

The watchdog relay will be energized anytime the Run LED (D3) is on. When the relay is energized, the normally open contact (J8 pin 1) will be connected to the relay 'common' (J8 pin 2). These signals can be brought out to the 'outside world' allowing external components to monitor the 'basic status' of the motion controller.



J8 Mating connector:

Pin Housing: Molex P/N 22-01-3027
Crimp pin: Molex P/N 08-50-0114

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Signal Descriptions

Motor Command Signals

+/- 10 Analog Command Outputs

signal type: +/- 10V analog, 16 bit
notes: Connects to servo amplifier motor command input (Ref+)
explanation: This output signal is used to control the servo amplifier's output. When connected to the command input of a velocity mode amplifier, the voltage level on this signal should cause the amplifier to drive the servo at a proportional velocity. For current mode amplifiers, the voltage level should cause a proportional current to be supplied to the servo. The range of the signal is -10 to +10 volts (with 16 bit resolution), with 0 volts being the null output level. Positive voltages indicate a desired velocity or current in one direction. Negative voltages indicate velocity or current in the opposite direction. The maximum drive current of this signal is +/-10 milliamps.

By using the function ***MCSetModuleOutputMode( )***, the output can be changed to Unipolar, where the analog signal range is 0 to +10 volts, and a separate signal (Unipolar Direction) is used to indicate the desired direction of velocity or current.

Pulse and Direction Command Outputs

signal type: Open collector, current sink, 100ma max. current sink, 30V max.
notes: External pull-up may be required
explanation: In the control of a stepper motor or Pulse command servo, the two primary control signals are Pulse and Direction (or CW Pulse and CCW Pulse). These signals are connected to the external driver that supplies current to the motor windings.

Both of these signals are driven by high current open collector drivers and are suitable for direct connection to optically isolated inputs commonly found on stepper motor drivers. Because of the characteristics of open collector drivers, no measurable voltages will be present on these output signals unless a pull-up path to a supply voltage is provided.

Pulse: The motor driver should advance the motor by one increment for each pulse. The motor may advance a full step or a micro step. This is determined by the mode of the stepper motor driver. The Pulse signal is normally high, and is pulled low at the beginning of a step. It stays low for one half the step period (50% duty cycle), and then goes back high. When it is time for the next step, the signal will be pulled low again.

Direction: This signal indicates the direction the motor will move. When the stepper is incrementing the current position (moving positive) this signal will remain high (pulled up). When the stepper is decrementing the current position (moving negative) this signal will be pulled low. For a servo motor configured for uniploar mode, this output is used to indicated the commanded direction of the servo.

The function ***MCSetModuleOutputMode( )*** is used to change the operation of these signals to CW and CCW. In this mode, pulses will be generated on the CW output when the current position is increasing, and on the CCW output when the current position is decreasing.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Encoder Feedback Signals

Encoder Inputs (Phase A+, Phase A-, Phase B+, Phase B-, Z+, Z-)

signal type: TTL or Differential driver output
Minimum signal differential (Phase + to Phase -) = 200mV
Maximum range = (-25V to +25V)

notes: For single ended encoders connect the **Encoder Reference Output (1.5V)** to all unused encoder (A-, B-, Z-) inputs

explanation: These input signals should be connected to an incremental quadrature encoder for supplying position feedback information for the Analog Command axes (1 - 4). The plus (+) and minus (-) signs refer to the two sides of differential inputs. If no index is being used connect Z+ and Z- to the **Encoder Reference Output**.

Encoder - Reference Output

signal type: 1.5 VDC (output from resistor voltage divider)

notes:

explanation: This output is made available so that any unconnected encoder inputs can be properly terminated. Most typically this output would be used to terminate the phase '-' inputs of a single ended encoders.

Default Axis Inputs



The default configuration for the controller is for an Analog Command axis and a Pulse Command axis to share opto isolated inputs and open collector drivers. The user can change any of the default I/O function assignments by using the **I/O Configuration Panel**, described on pages 10 and 165.

Amp. / Drive Fault Input

signal type: Bi-directional optical isolator, 0.25 mA min., 3.0V – 25.0V range

notes:

explanation: This input is designed to be connected to the Fault or Error output signal of a servo amplifier or stepper driver. The state of this signal will appear as a status bit in the axis status word. By default this input is shared by an Analog command axis and a Pulse command axis.

The ***EnableAmpFault*** member of the ***MCMotion*** structure will enable the axis to be disabled if the Amp / Drive Fault input is activated. No further motion will occur until the fault signal is deactivated and the axis has been enabled. The input device is a bi-directional optical isolator. The allowable voltage range for this signal is **3.0 VDC to 25.0 VDC**. For I/O systems operating outside of this range consult the factory.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Coarse Home / Stepper Home Input

signal type: Bi-directional optical isolator, 0.25 mA min., 3.0V – 25.0V range
notes: The Home operation of a Pulse command axis cannot be re-assigned to use a different controllerinput
explanation: This input is used to determine the proper zero position of an axis. By default this input is shared by an Analog command axis and a Pulse command axis.

Servo systems: If a rotary encoders with index outputs is used, an index pulse will be asserted once per rotation of the encoder. While this signal occurs at a very repeatable angular position on the encoder, it may occur many times within the motion range of the servo. In these cases, a Coarse Home switch is required to qualify which index pulse is to be the true zero position of the servo.
The Coarse Home switch should be installed (and the encoder adjusted) so that while the switch is active, the index pulse that is to be used to define the 'home position' will be asserted.

The input device for this signal is a bi-directional optical isolator. The allowable voltage range for this signal is **3.0 VDC to 25.0 VDC**. For I/O systems operating outside of this range please contact the factory. For additional information on homing a servo axis please refer to the section titled **Homing Axes** in the **Motion Control chapter**. Typical wiring examples for the **Coarse Home / Stepper Home Input** can be found in **Chapter 5**.

Stepper systems: This input is used to set the zero position of an open loop stepper axis. It is typically connected to a sensor/switch that is activated at a fixed position in the motor's range of motion. The input device is a bi-directional optical isolator. The allowable voltage range for this signal is **3.0 VDC to 25.0 VDC**.

The allowable voltage range for this signal is **3.0 VDC to 25.0 VDC**. For I/O systems operating outside of this range please contact the factory. For additional information on homing a stepper axis please refer to the section titled **Homing Axes** in the **Motion Control chapter**. Typical wiring examples for the **Coarse Home / Stepper Home Input** can be found in **Chapter 5**.

Limit Positive Input

signal type: Bi-directional optical isolator, 0.25 mA min., 3.0V – 25.0V range
notes:
explanation: The limit switch inputs are used to cause the controller to stop the motion of a servo or stepper axis when it reaches the end of travel. By default this input is shared by both an Analog command axis and a Pulse command axis. In Position and Velocity mode the response to an activated limit input is direction sensitive, the axis will only be stopped if it is moving in the direction of the activated limit switch. In Contour mode, the response to an activated limit input is not direction sensitive, the axis will be stopped regardless of the direction it is moving if either limit switch is activated. In Torque mode, the controller will ignore the activation of a limit input, the axis will continue to move. For I/O systems operating outside the range of 3V to 25V contact the factory.

There are three modes of stopping (decelerate to a stop, stop immediately, turn off the axis) that can be configured by the function ***MCSetLimits( )***. The limit switch inputs can be enabled and disabled by ***MCSetLimits( )***. See the description of **Motion Limits** in the **Motion Control** chapter.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Limit Negative Input

signal type: Bi-directional optical isolator, 0.25 mA min., 3.0V – 25.0V range
notes:
explanation: The limit switch inputs are used to cause the controller to stop the motion of a servo or stepper axis when it reaches the end of travel. By default this input is shared by both an Analog command axis and a Pulse command axis. In Position and Velocity mode the response to an activated limit input is direction sensitive, the axis will only be stopped if it is moving in the direction of the activated limit switch. In Contour mode, the response to an activated limit input is not direction sensitive, the axis will be stopped regardless of the direction it is moving if either limit switch is activated. In Torque mode, the controller will ignore the activation of a limit input, the axis will continue to move. For I/O systems operating outside the range of 3V to 25V contact the factory.

There are three modes of stopping (decelerate to a stop, stop immediately, turn off the axis) that can be configured by the function ***MCSetLimits( )***. The limit switch inputs can be enabled and disabled by ***MCSetLimits( )***. See the description of **Motion Limits** in the **Motion Control** chapter.

Position Capture (Latch) Input

signal type: TTL (buffered by 74LS541)
Active level = Rising edge, (TTL high, > 2.4 volts)
Minimum pulse duration = 100 nano second
Maximum re-trigger frequency = 1 KHz
TTL high level input min. voltage = 2.0V
TTL high level input max. voltage = 5.0V
TTL low level input min. voltage = 0.0V
TTL low level input max. voltage = 0.6V
notes: Dual purpose signal, can also be used as a general purpose TTL digital input
explanation: Used to initiate the capture of position data. See the description of **Position Capture** in the **Application Solutions chapter**.

Default Axis Outputs



The default configuration for the controller is for an Analog Command axis and a Pulse Command axis to share opto isolated inputs and open collector drivers. The user can change any of the default I/O function assignments by using the **I/O Configuration Panel**, described on pages 10 and 165.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Drive Disable

signal type: Open collector, current sink, 100ma max. current sink, 30V max.
notes: External pull-up may be required
explanation: This output signal should be connected to the **disable input** of the stepper driver or servo amplifier. When the axis is disabled (or the controller is reset) the open collector driver will be turned on, sinking current through the interface device of the stepper driver / servo amplifier. When the axis is turned on this signal will immediately go to its' inactive high level. Anytime there is an error on the respective axis, including ***exceeding the following error, a limit switch input activated or the Amplifier / Driver Fault input activated***, the Driver Disable signal will be activated.

This signal is driven by a high current open collector driver and is suitable for direct connection to optically isolated inputs commonly found on a amplifier / driver. Because of the characteristics of open collector drivers, no voltages will be present on these output signals unless signals unless a pull-up path to a supply voltage is provided.

Amplifier / Driver Enable

signal type: Open collector, current sink, 100ma max. current sink, 30V max.
notes: External pull-up may be required
explanation: This output signal should be connected to the **enable input** of the servo amplifier or stepper driver. When the axis is enabled the open collector driver will be turned on, sinking current through the interface device of the servo amplifier / stepper driver. When the axis is turned off (or the controller is reset) this signal will immediately go to its' inactive high level. Anytime there is an error on the respective axis, including ***exceeding the following error, a limit switch input activated or the Amplifier / Driver Fault input activated***, the Amplifier Enable signal will be deactivated.

This signal is driven by a high current open collector driver and is suitable for direct connection to optically isolated inputs commonly found on a amplifier / driver. Because of the characteristics of open collector drivers, no voltages will be present on these output signals unless a pull-up path to a supply voltage is provided.

Position Compare Output

signal type: TTL (buffered by 74LS541)
Active level = programmable, default = TTL high
Minimum pulse duration (one Shot mode) = 1 msec. (+/- 0.5 msec.)
Servo Maximum re-trigger frequency = 4 KHz
Stepper Maximum re-trigger frequency = 1 KHz
TTL low level current sink max. = 24 mA
TTL high level current source max. = 15 mA
TTL high level output min. voltage = 2.4V
TTL high level output max. voltage = 5.0V
TTL low level output min. voltage = 0.0V
TTL low level output max. voltage = 0.5V
notes: Dual purpose, also can be used for general purpose TTL digital output
explanation: Used to indicate when a position compare event has occurred. See the description of **Position Compare** in the **Application Solutions chapter**.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Full/Half Current & Unipolar Direction Output

signal type: Open collector, current sink, 100ma max. current sink, 30V max.
notes: External pull-up may be required
explanation: (Full/Half current) This signal is used if the stepper driver has a digital input for current control. The default condition of this signal is to be inactive (pulled high). Setting the **MC_CURRENT_FULL** parameter of the **MCMotion** structure will cause the signal to be pulled low.

This signal is driven by a high current open collector driver and is suitable for direct connection to optically isolated inputs commonly found on a amplifier / driver. Because of the characteristics of open collector drivers, no voltages will be present on these output signals unless a pull-up path to a supply voltage is provided.

explanation (Unipolar Direction): For servo drives requiring a Unipolar output. The velocity or current command input consists of a magnitude signal and a separate direction signal . The magnitude signal is provided by the modules Analog Command Signal, while this signal provides a digital direction command.

Default Configuration of General Purpose I/O



The default configuration for the controller is for a Analog Command axis and a Pulse Command axis to share opto isolated inputs and open collector drivers. The user can change any of the default I/O function assignments by using the **I/O Configuration Panel**, described on pages 10 and 165.

TTL Digital Inputs

signal type: TTL (buffered by 74LS541)
Active level = programmable
TTL high level input min. voltage = 2.0V
TTL high level input max. voltage = 5.0V
TTL low level input min. voltage = 0.0V
TTL low level input max. voltage = 0.6V
notes: Dual purpose, channels 1, 5, 9, and 13 can also be used for capturing the position of axes 1/2, 3/4, 5/6, and 7/8
explanation: Two 74LS541 octal buffers are used to provide 16 TTL level digital inputs that allow the user to monitor external events. For additional information please refer to the **General Purpose I/O chapter**.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TTL Digital Outputs

signal type: TTL (buffered by 74LS541)
Active level = programmable, default = TTL high
TTL low level current sink max. = 24 mA
TTL high level current source max. = 15 mA
TTL high level output min. voltage = 2.4V
TTL high level output max. voltage = 5.0V
TTL low level output min. voltage = 0.0V
TTL low level output max. voltage = 0.5V

notes: Dual purpose, channels 1 and 9 can also be used for indicating when a position compare event has occurred on axes 1 - 8

explanation: Two 74LS541 octal buffers are used to provide 16 TTL level digital outputs that allow the user to control external devices. For additional information please refer to the **General Purpose I/O chapter**.

Analog Inputs (optional)

***signal type*:** Analog input range: -10.0V to +10.0V (default) or 0.0V to +4.0V (special order)
A/D resolution = 14 bit
Nominal read rates:
From a Windows program = ~200 usec's
From on-board MCCL routine = ~42 usec's

***notes*:** Reported value = 0 to 65536 corresponding to the available input range

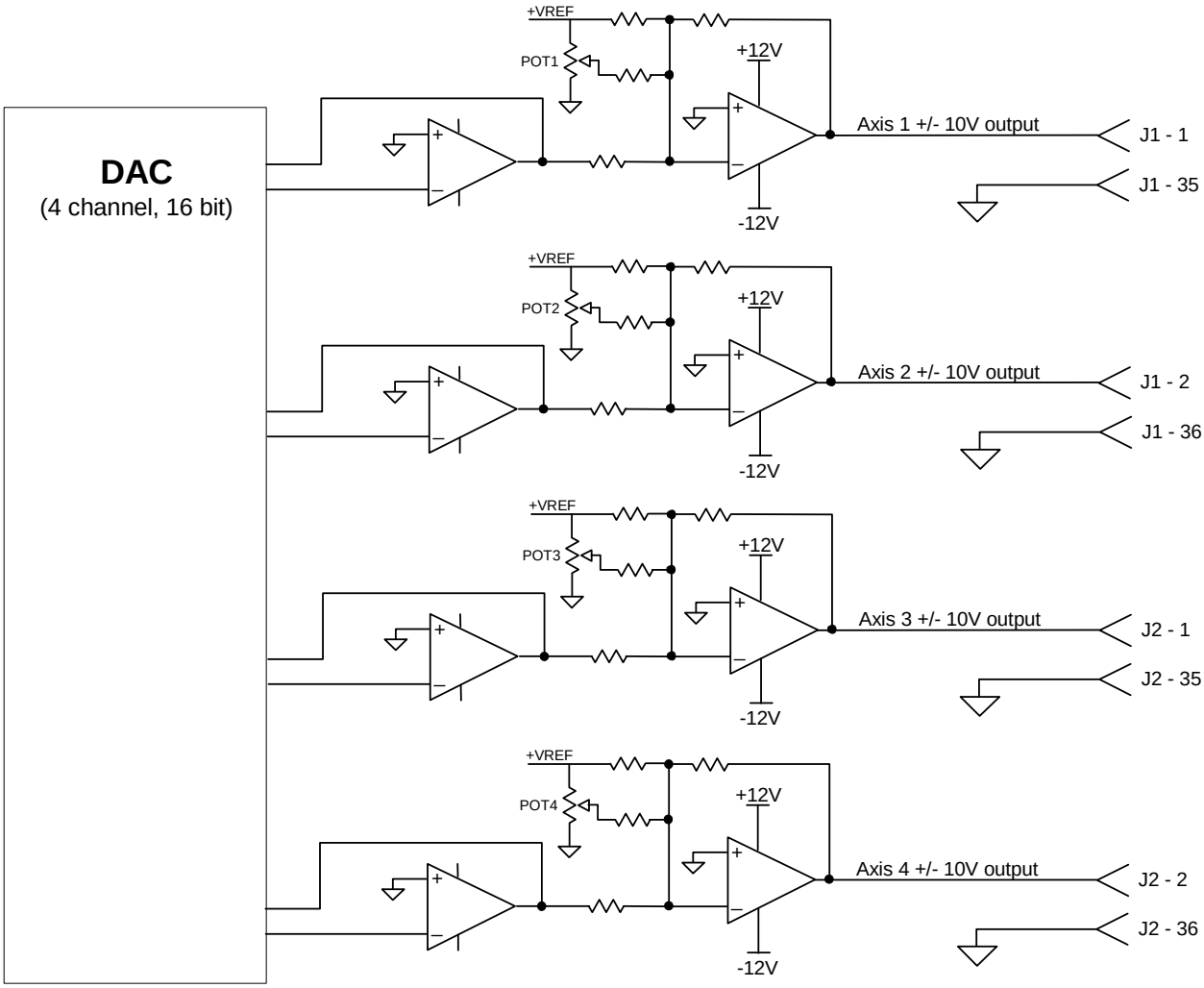
***explanation*:** Eight (8) 14 bit analog inputs that allow the user to monitor external events. For additional information please refer to the General Purpose I/O chapter.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Circuit Schematics

+/- 10V servo command circuit schematic

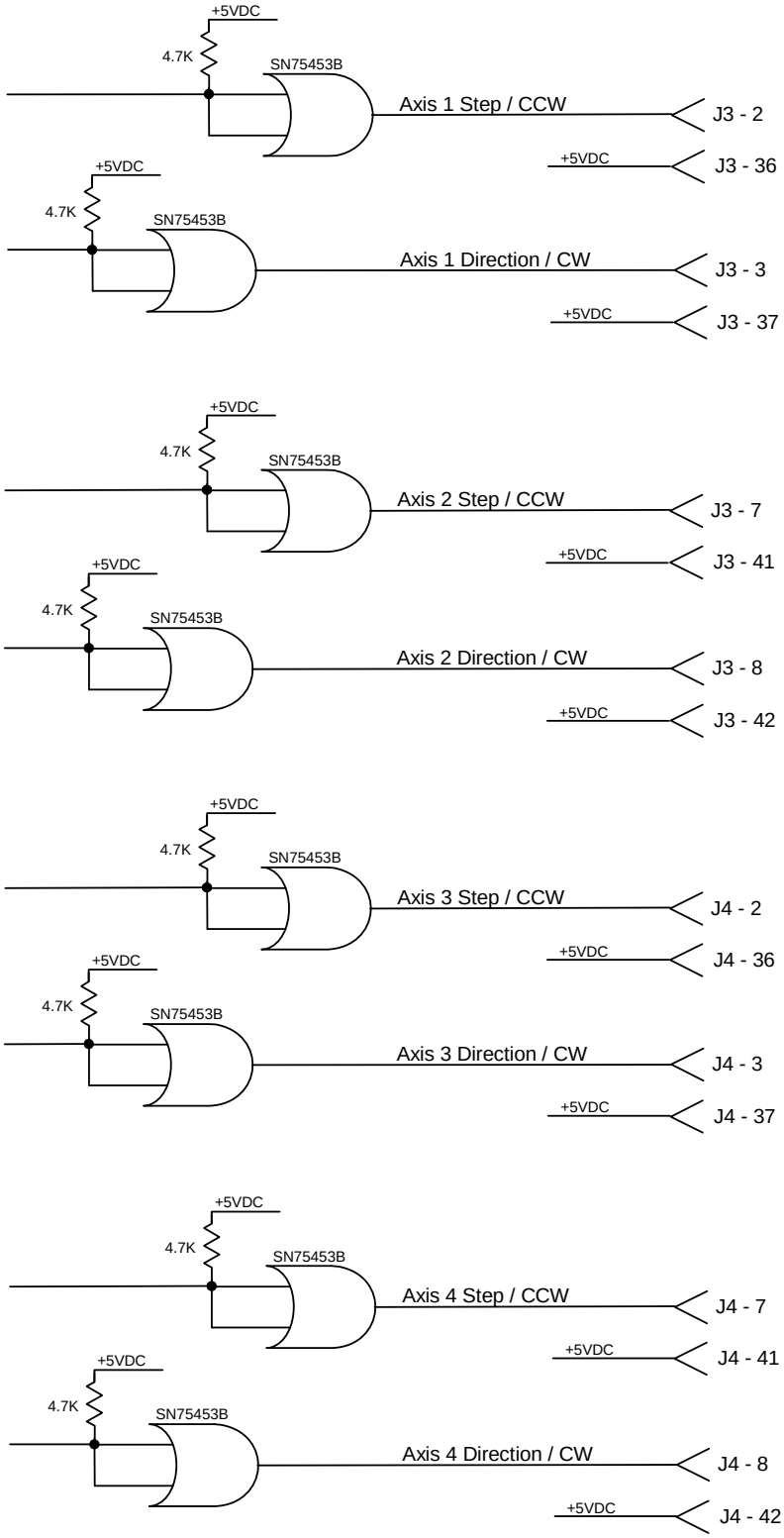
(connector pin-outs reference connectors on the controller)



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Pulse Command Circuit Schematic

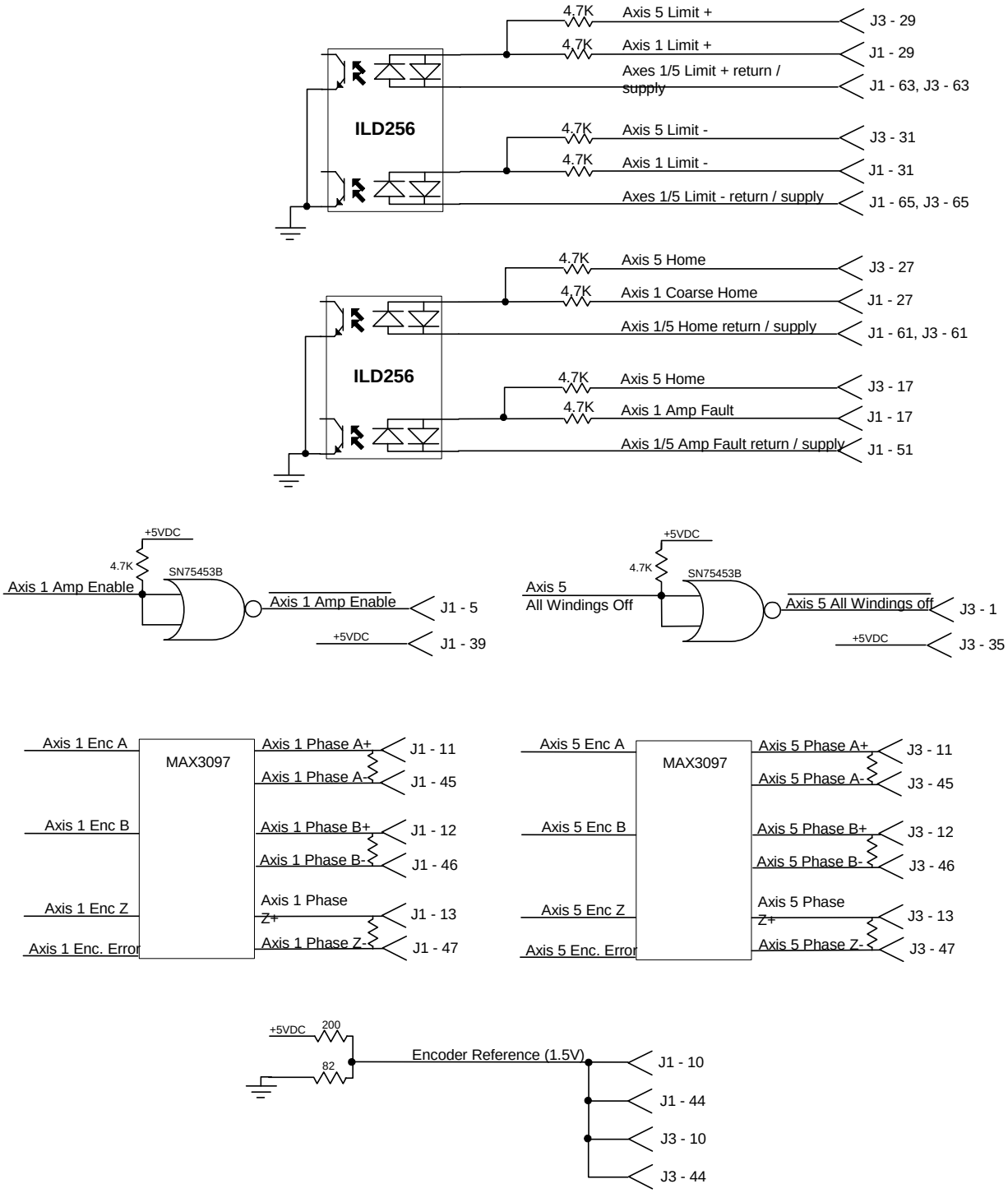
(connector pin-outs reference connectors on the controller)



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Axis I/O circuit schematic

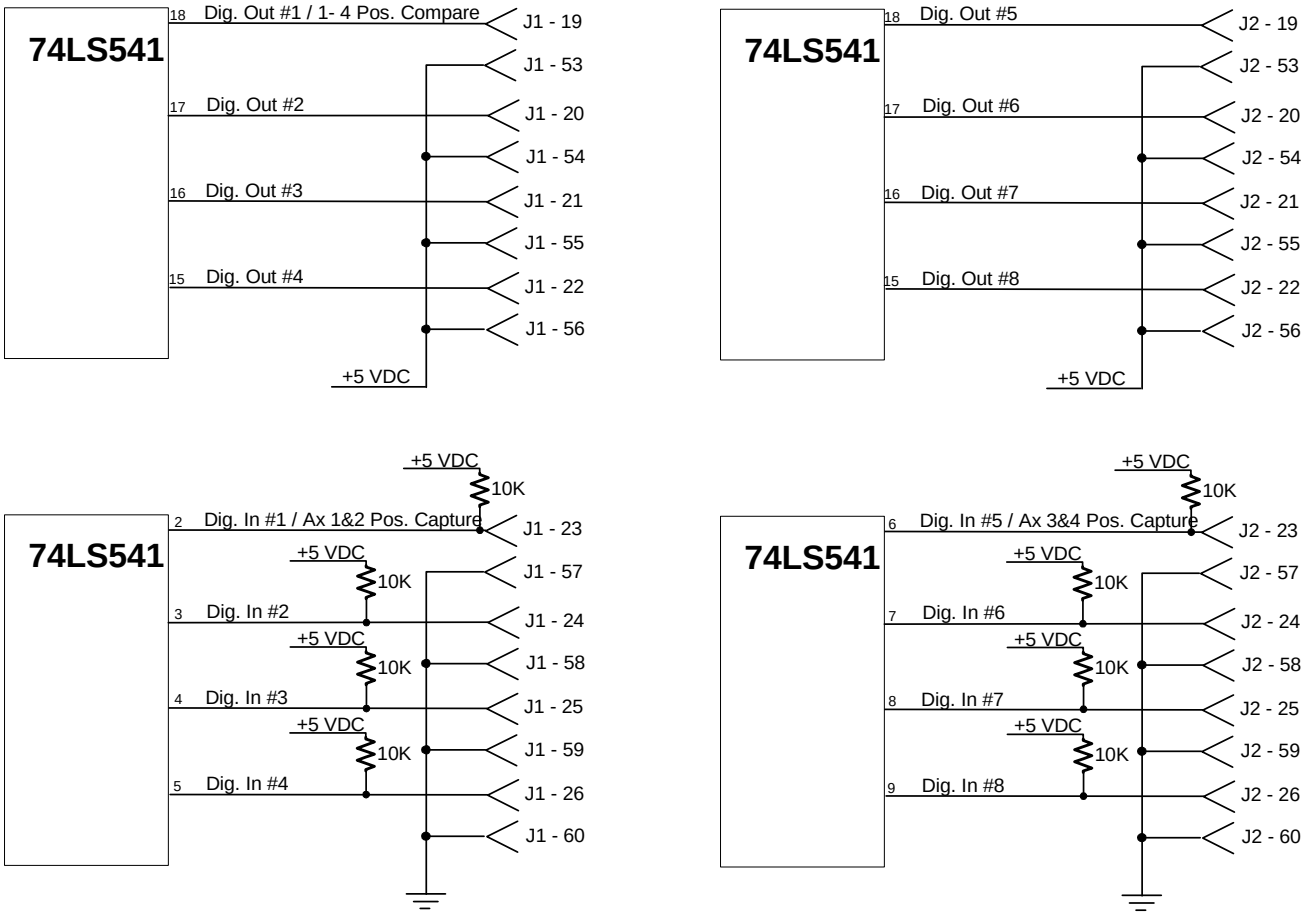
(connector pin-outs reference connectors on the controller)



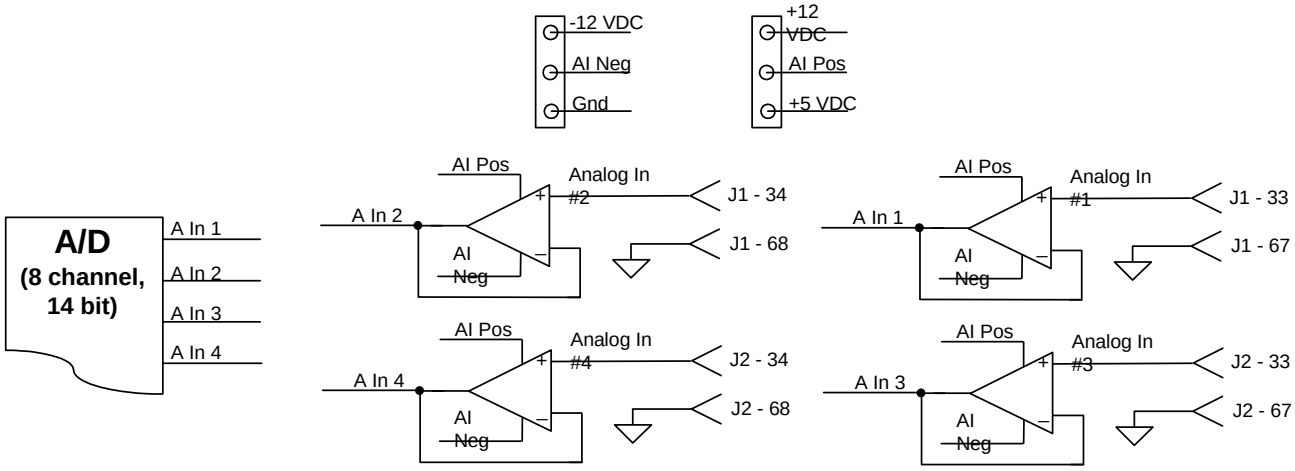
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

General-Purpose I/O Circuit Schematic

(connector pin-outs reference connectors on the controller)



Optional A/D inputs



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Chapter 10

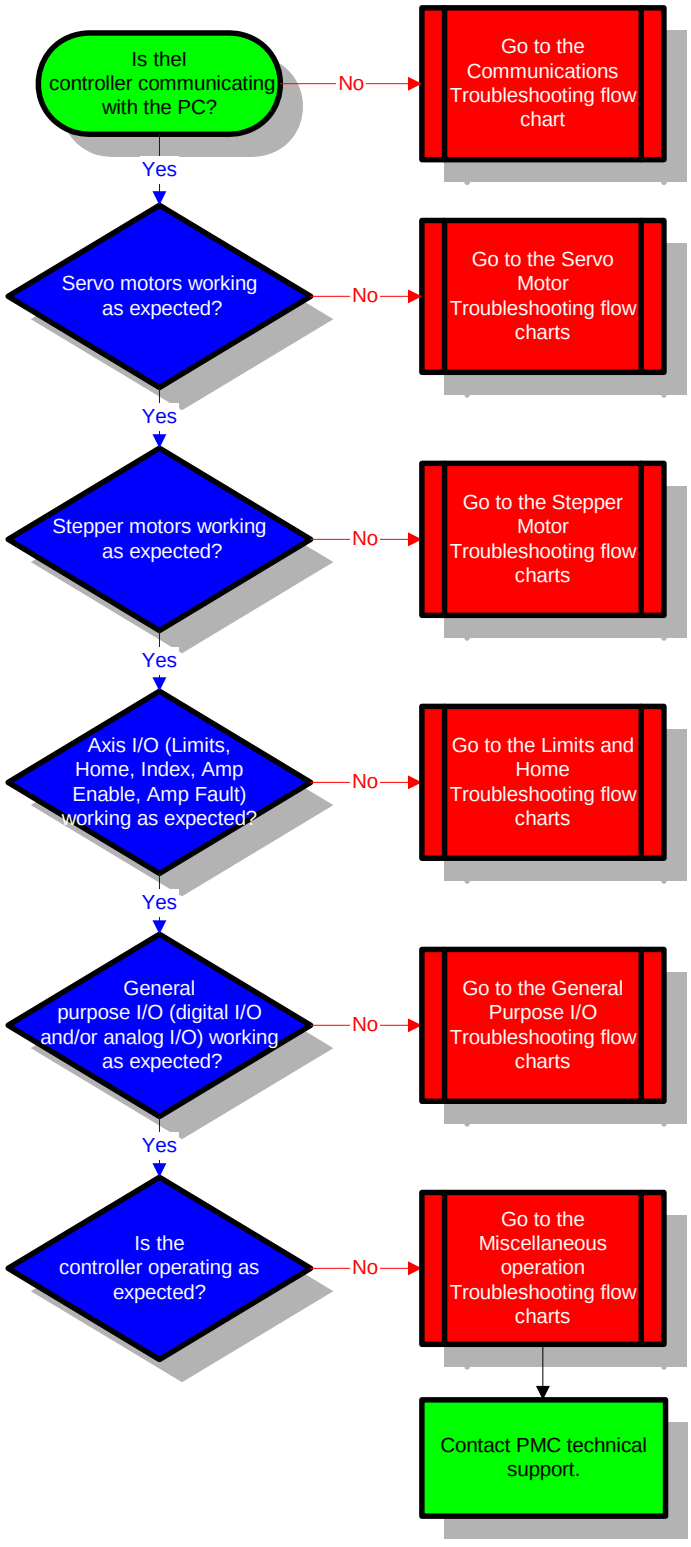
Troubleshooting

On the following pages you will find troubleshooting flow charts to assist the with diagnosis of motion control system failures.

The steps described in these flow charts will direct you to programs installed with PMC's Motion Control API (Motion Integrator, Motor Mover, CWDemo, Servo Tuning, WinControl, etc.). These programs can be used to help diagnose and resolve system problems.

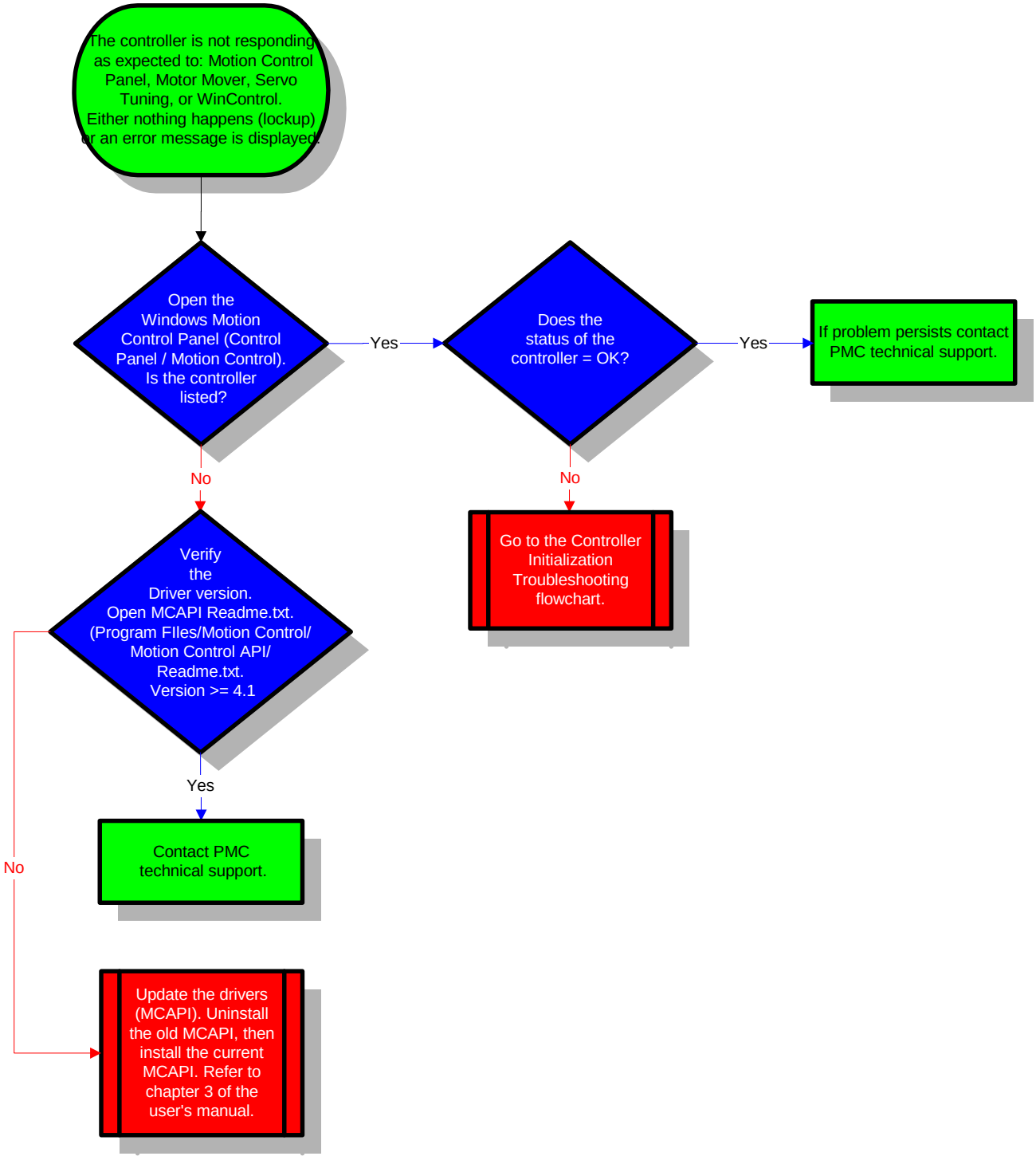
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

System Troubleshooting



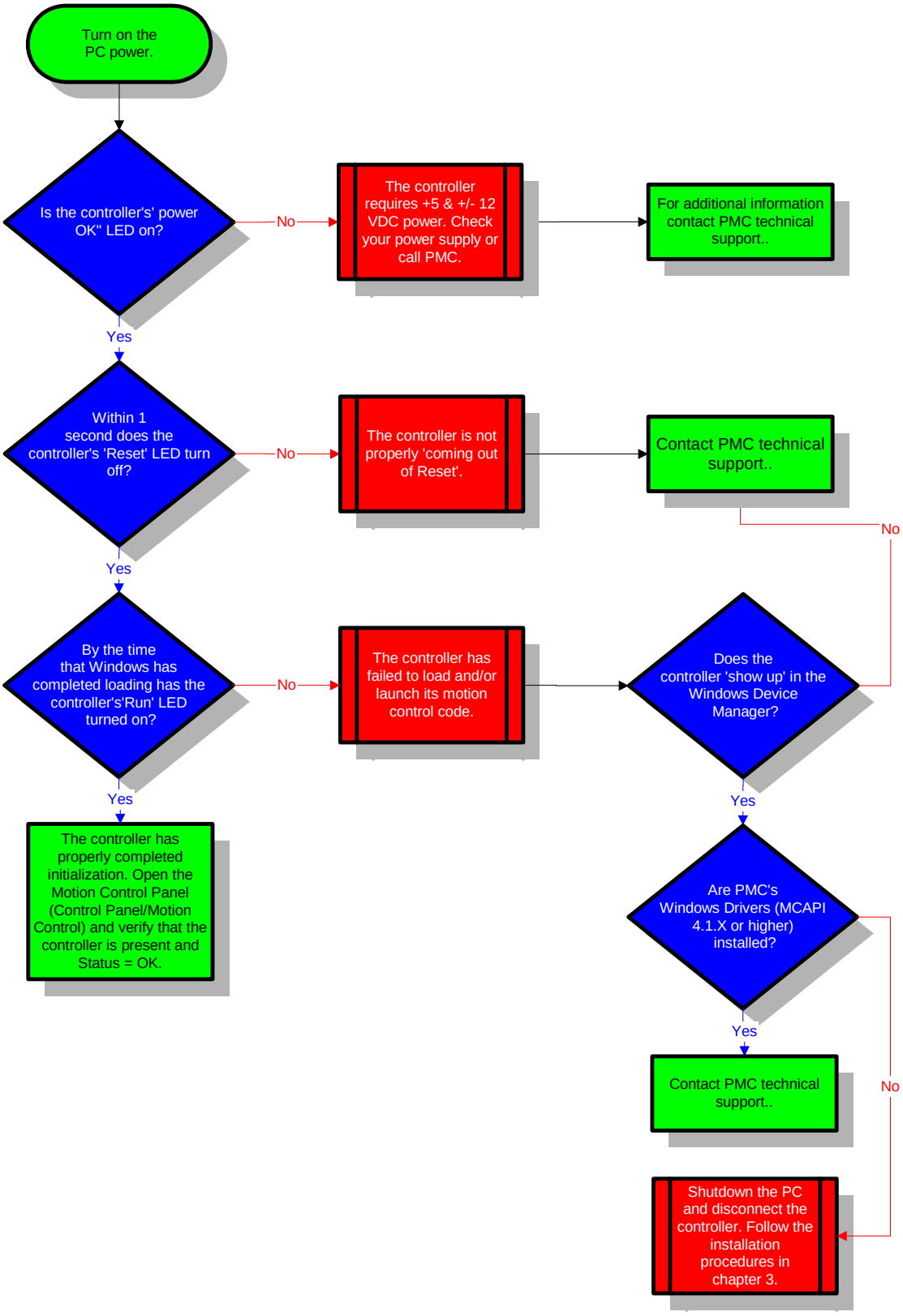
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Communications Troubleshooting



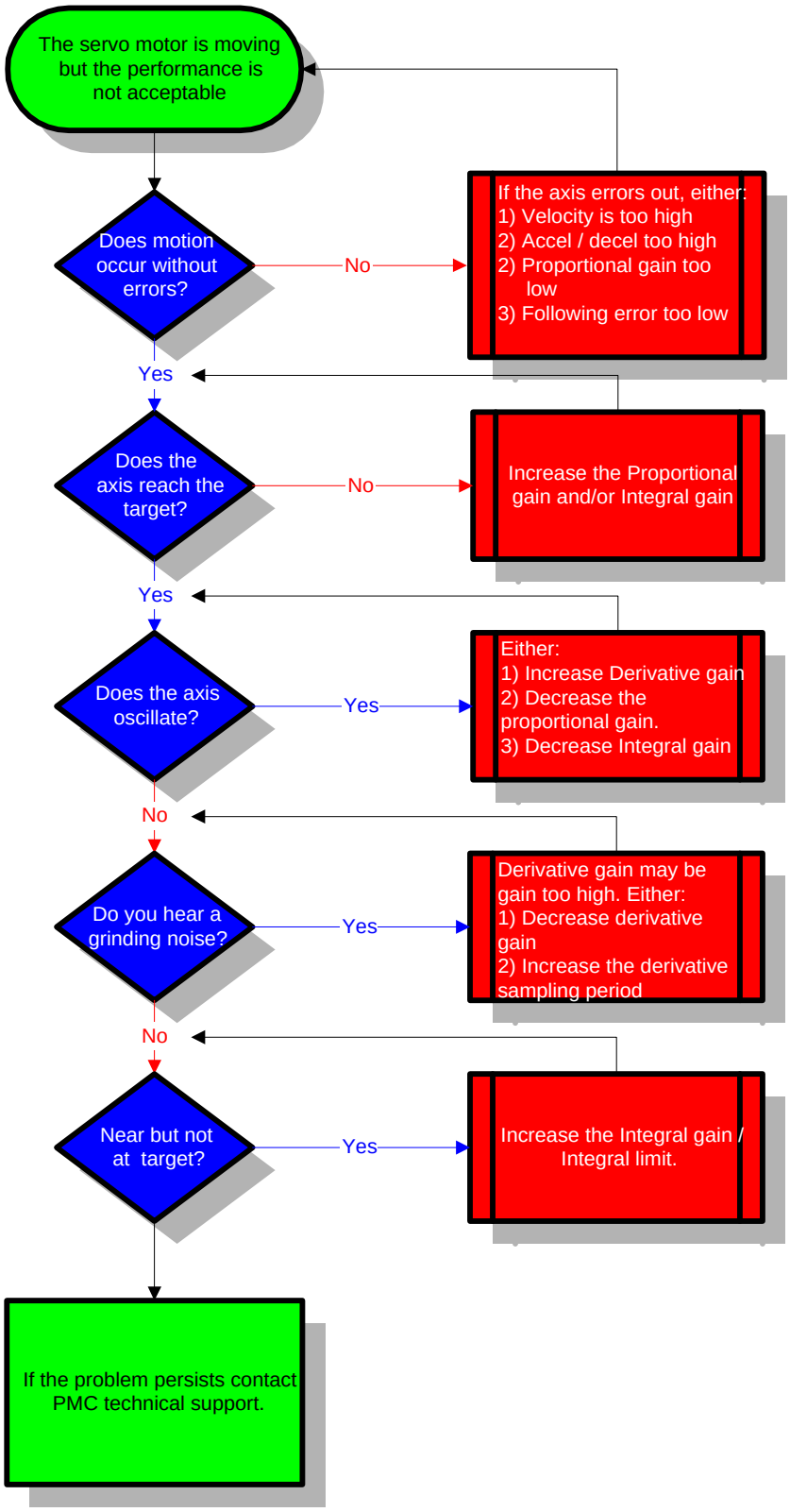
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Controller Initialization Troubleshooting



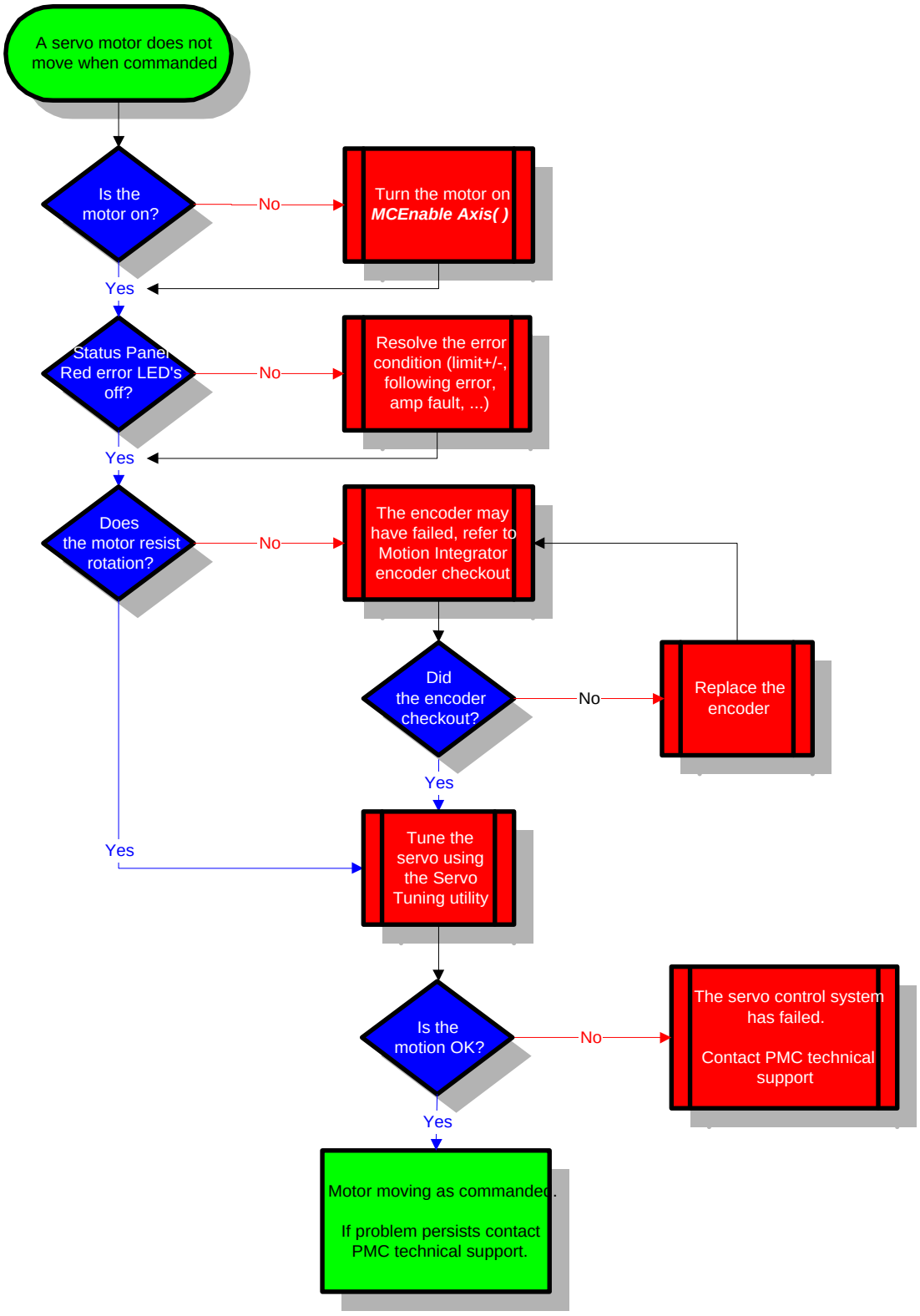
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Troubleshooting - Tuning a Servo Motor



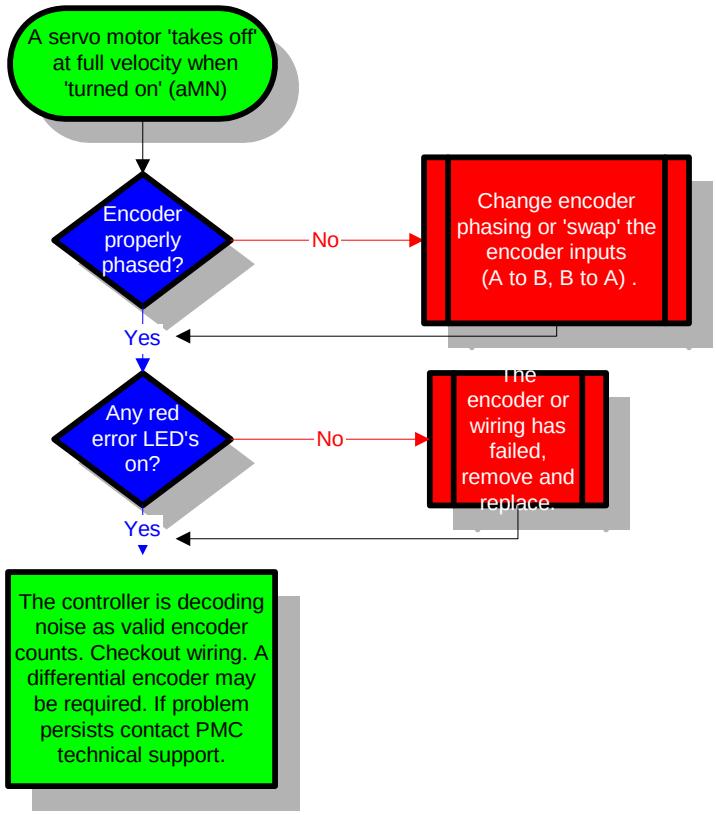
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Troubleshooting - Servo Motion chart #1



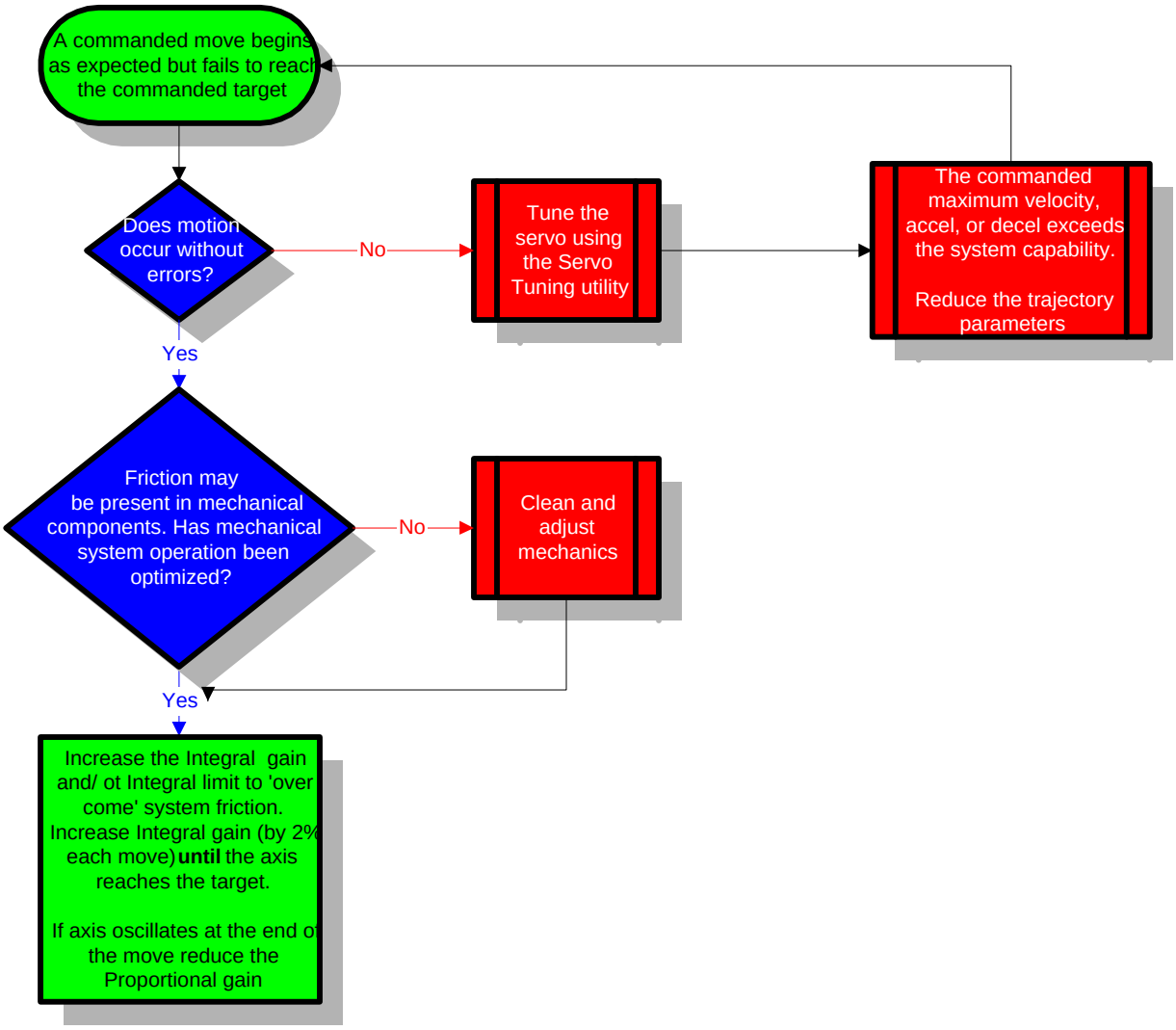
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Troubleshooting - Servo Motion chart #2



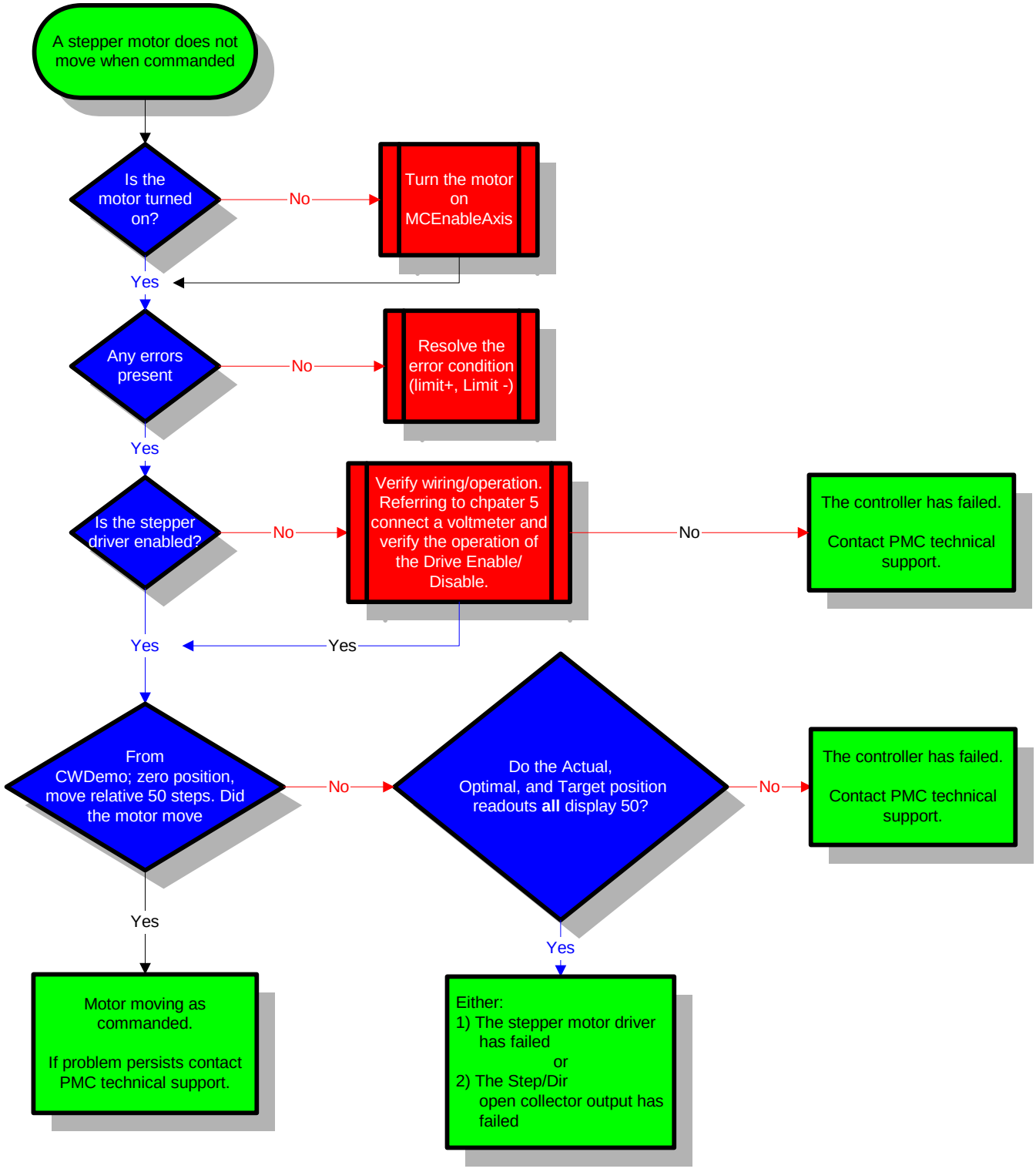
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Troubleshooting - Servo Motion chart #3



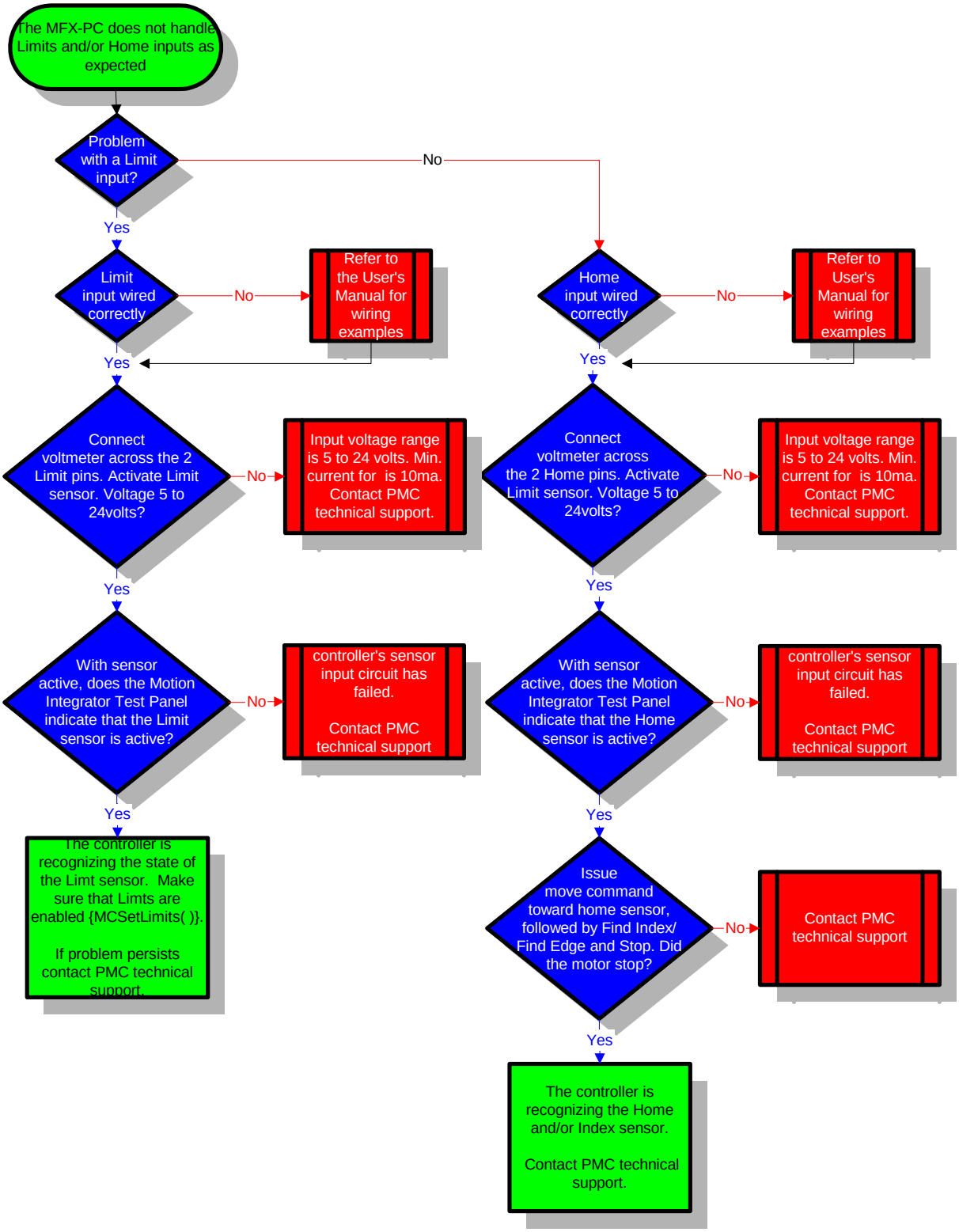
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Troubleshooting - Stepper Motion chart #1



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Troubleshooting - Limits and Home



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Chapter 11

Controller Error Codes

Both the Motion Control API functions and the Motion Control Command Language (MCCL) provide error code and interface status information to the user.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Motion Control API Error Codes

Motion Control API defined error messages are listed numerically in the table below. Where possible corrective action is included in the description column. Please note that many Motion Control API function descriptions also include information regarding errors that are specific to that function.

Error	Constant	Description
0	MCERR_NOERROR	No error has occurred
1	MCERR_NO_CONTROLLER	No controller assigned at this ID. Use MCSETUP to configure a controller.
2	MCERR_OUT_OF_HANDLES	Motion Control API driver out of handles. The driver is limited to 32 open handles. Applications that do not call MCClose( ) when they exit may leave handles unavailable, forcing a reboot.
3	MCERR_OPEN_EXCLUSIVE	Cannot open - another application has the controller opened for exclusive use
4	MCERR_MODE_UNAVAIL	Controller already open in different mode. Some controller types can only be open in one mode (ASCII or binary) at a time
5	MCERR_UNSUPPORTED_MODE	Controller doesn't support this mode for MCOpen( ) - i.e. ASCII or binary
6	MCERR_INIT_DRIVER	Couldn't initialize the device driver
7	MCERR_NOT_PRESENT	Controller hardware not present
8	MCERR_ALLOC_MEM	Memory allocation error. This is an internal memory allocation problem with the DLL, contact Technical Support for assistance
9	MCERR_WINDOWSERROR	A windows function returned an error - use GetLastError () under WIN32 for details
10		reserved
11	MCERR_NOTSUPPORTED	Controller doesn't support this feature
12	MCERR_OBSOLETE	Function is obsolete
13	MCERR_AXIS_TYPE	Axis type doesn't support this feature
14	MCERR_CONTROLLER	Invalid controller handle
15	MCERR_WINDOW	Invalid window handle
16	MCERR_AXIS_NUMBER	Axis number out of range
17	MCERR_ALL_AXES	Cannot use MC_ALL_AXES for this function
18	MCERR_RANGE	Parameter was out of range
19	MCERR_CONSTANT	Constant value inappropriate
20	MCERR_UNKNOWN_REPLY	Unexpected or unknown reply
21	MCERR_NO_REPLY	Controller failed to reply
22	MCERR_REPLY_SIZE	Reply size incorrect
23	MCERR_REPLY_AXIS	Wrong axis for reply
24	MCERR_REPLY_COMMAND	Reply is for different command
25	MCERR_TIMEOUT	Controller failed to respond
26	MCERR_BLOCK_MODE	Block mode error. Caused by calling MCBlockEnd( ) without first calling MCBlockBegin( ) to begin the block
27	MCERR_COMM_PORT	Communications port (RS232) driver reported an error
28	MCERR_CANCEL	User canceled action (such as when an MCDLG dialog box is dismissed with the CANCEL button
29	MCERR_NOT_INITIALIZED	Feature was not correctly initialized before being enabled or used

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

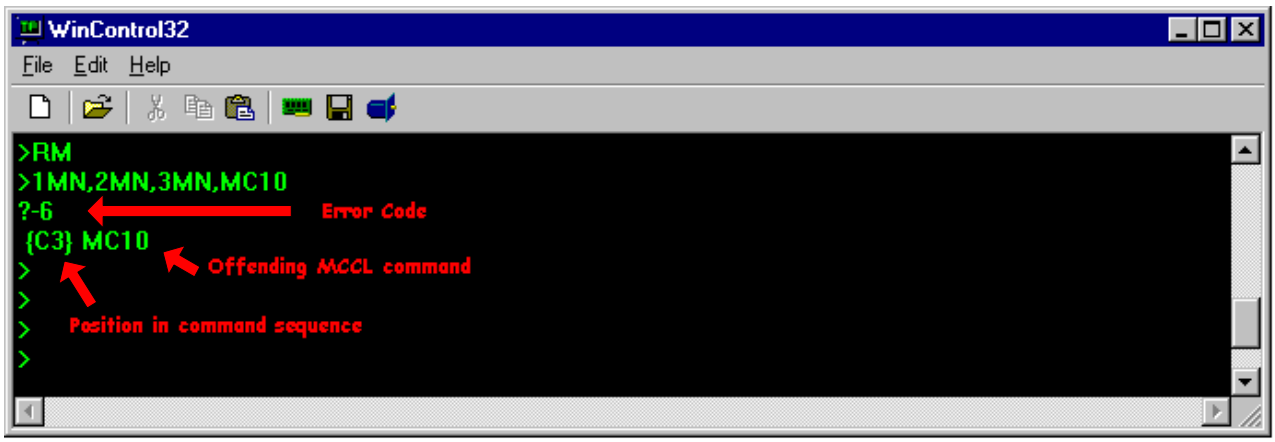
MCCL Error Codes

When executing MCCL (Motion Control Command Language) command sequences the command interpreter will report the following error code when appropriate:

Table 7. MCCL Error Codes

Description	Error code
No error	0
Unrecognized command	1
Bad command format	2
I/O error	3
Command string to long	4
Command Parameter Error	-1
Command Code Invalid	-2
Negative Repeat Count	-3
Macro Define Command Not First	-4
Macro Number Out of Range	-5
Macro Doesn't Exist	-6
Command Canceled by User	-7
Contour Path Command Not First	-8
Contour Path Command Parameter Invalid	-9
Contour Path Command Doesn't Specify an AXIS	-10
Axis error (over travel error, max. following error exceeded	-13
No axis specified	-14
Axis not assigned	-15
Axis already assigned	-16
Axis duplicate assigned	-17
Insufficient memory	-18
Unrecognized variable name	-19
Invalid background task ID	-20
Command not supported	-21

Many error code reports will not only include the error code but also the offending command. In the following example the Reset Macro command was issued. This command clears all macro's from memory. The next command sequence turns on 3 motors and then calls macro 10. The command MC10 is a valid command but with no macros in memory error code –6 is displayed.



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Glossary

Accuracy - A measure of the difference between the expected position and actual position of a motion system.

Actuator - Device that creates mechanical motion by converting energy to mechanical energy.

Axis Phasing - An axis is properly phased when a commanded move in the positive direction causes the encoder decode circuitry of the controller to increment the reported position of the axis.

Back EMF - The voltage generated when a permanent magnet motor is rotated. This voltage is proportional to motor speed and is present regardless of whether the motor windings are energized or de-energized.

Closed Loop - A broadly applied term, relating to any system in which the output is measured and compared to the input. The output is then adjusted to reach the desired condition. In motion control, the term typically describes a system utilizing a velocity and/or position transducer to generate correction signals in relation to desired parameters.

Command Set – Defines the operations that can be executed by the motion controller

Commutation - The action of applying currents or voltages to the proper motor phases in order to produce optimum motor torque.

Critical Damping - A system is critically damped when the response to a step change in desired velocity or position is achieved in the minimum possible time with little or no overshoot.

DAC - The digital-to-analog converter (DAC) is the electrical interface between the motion controller and the motor amplifier. It converts the digital voltage value computed by the motion controller into an analog voltage. The more DAC bits, the finer the analog voltage resolution. DACs are available in three common sizes: 8, 12, and 16 bit. The bit count partitions the total peak-to-peak output voltage swing into 256, 4096, or 65536 DAC steps, respectively.

Dead Band - A range of input signals for which there is no system response.

Driver - Electronics that convert step and direction inputs to high power currents and voltages to drive a step motor. The step motor driver is analogous to the servo motor amplifier.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Dual Loop Servo – A servo system that combines a velocity mode amplifier/tachometer with a position loop controller/encoder. It is recommended that the encoder not be directly coupled to the motor. The linear scale encoder should be mounted on the external mechanics, as closely coupled as possible to the 'end effector'

Duty Cycle - For a repetitive cycle, the ratio of on time to total time:

Efficiency - The ratio of power output to power input.

Encoder - A type of feedback device that converts mechanical motion into electrical signals to indicate actuator position or velocity.

End Effector – The point of focus of a motion system. The tools with which a motion system will work. Example: The leading edge of the knife is the *end effector* of a three axis (XYZ) system designed to cut patterns from vinyl.

Feed Forward - Defines a specific voltage level output from a motion controller, which in turn commands a velocity mode amplifier to rotate the motor at a specific velocity.

Following Error - The difference between the calculated desired trajectory position and the actual position.

Friction - A resistance to motion caused by contacting surfaces. Friction can be constant with varying speed (Coulomb friction) or proportional to speed (viscous friction).

Holding Torque - Sometimes called static torque, holding torque specifies the maximum external torque that can be applied to a stopped, energized motor without causing the rotor to rotate continuously.

Inertia - The measure of an object's resistance to a change in its current velocity. Inertia is a function of the object's mass and shape.

Kd - K is a generally accepted variable used to represent gain, an arbitrary multiplier, or a constant. The lower case 'd' designates derivative gain.

Ki - K is a generally accepted variable used to represent gain, an arbitrary multiplier, or a constant. The lower case 'i' designates integral gain.

Kp - K is a generally accepted variable used to represent gain, an arbitrary multiplier, or a constant. The lower case 'p' designates proportional gain.

Limits - Motion system sensors (hard limits) or user programmable range (soft limits) that alert the motion controller that the physical end of travel is being approached and that motion should stop.

MCAPI - The Motion Control Application Programming Interface - this is the programming interface used by Windows programmers to control PMC's family of motion control cards.

MCCL - Motion Control Command Language - this is the command language used to program PMC's family of motion control cards.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Micro-Stepping - Stepper drive systems have a fixed number of electromechanical detents or steps. Micro stepping is an electronic technique to break each detent or step into smaller parts. This results in higher positional resolution and smoother operation.

Open Loop – A control system in which the control output is not referenced or scaled to an external feedback.

Position Error - see following error.

Position Move - Unlike a velocity move, a position move includes a predefined stopping position. The trajectory generator will determine when to begin deceleration in order to ensure the actual stopping point is at the desired target position.

PWM - Pulse Width Modulation is a method of controlling the average current in a motor's phase windings by varying the duty cycle of transistor switches.

Repeatability - The degree to which the positioning accuracy for a given move performed repetitively can be duplicated.

Resonance - A condition resulting from energizing a motor at a frequency at or close to the motor's natural frequency.

Resolution - The smallest positioning increment that can be achieved.

Resolver - A type of feedback device that converts mechanical position into an electrical signal. A resolver is a variable transformer that divides the impressed AC signal into sine and cosine output signals. The amplitude of these signals represents the absolute position of the resolver shaft.

Servo - An automatic system in which the output is constantly compared with the input through some form of feedback. The error (or difference) between the two quantities can be used to bring about the desired amount of control.

Servo tuning – the process in which the appropriate gain values for the PID filter are determined

Slew - That portion of a move made at constant, non-zero velocity.

Step Response - An instantaneous command to a new position. Typically used for tuning a closed loop system, ramping (velocity, acceleration, and deceleration) is not applied nor calculated for the move.

Tachometer - A device attached to a moving shaft that generates a voltage signal directly proportional to rotational speed.

Torque -

Velocity Mode Amplifier – An amplifier that requires a tachometer to provide the feedback used to close the velocity loop within the amplifier.

Velocity Move - A move where no final stopping position is given to the motion controller. When a start command is issued the motor will rotate indefinitely until it is commanded to stop.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Appendix

Default Axis Configuration Settings

Table 8. Default Axis Configuration Settings

Description	Setting
Programmed Velocity	10,000
Programmed Acceleration	10,000
Programmed Deceleration	10,000
Minimum Velocity	1,000
Current Velocity	0
Velocity Gain	0
Acceleration Gain	0
Deceleration Gain	0
Velocity Override	1
Torque Limit	10
Proportional Gain	.2
Derivative Gain	.1
Integral Gain	.01
Integration Limit	50
Maximum Following Error	1024
Motion Limits	disabled
Low Limit of Movement	0
High Limit of Movement	0
Servo Loop Rate	HS (High)
Stepper Pulse Range	HS (High)
Position Count	0
Optimal Count	0
Index Count	0
Auxiliary Status	0
Position	0
Target	0
Optimal Position	0
Breakpoint Position	0
Position Dead band	0

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

User Scale	1
User Zero	0
User Offset	0
User Rate Conversion	1
User Output Constant	1
Sampling Frequency	4 KHz
Slave Ratio	1

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Index

A

A/D inputs
description 170
wiring 63
Acceleration
default units 86
disable 42
setting 8, 37, 67, 77, 86
Active level
limit switches 103
Amplifier
Torque mode 25
Velocity mode 25
Amplifier fault
wiring 55
Analog command offset potentiometer 180
Analog I/O
configuring 170
testing 170
Analog I/O test panels 41
Analog input
reporting 171
Analog inputs
connector pin out 181, 185
description 170
signal description 196
specifications 176
Anplifier Enable
wiring 52, 54
Application program samples ... 8, 32, 33, 34, 35, 37
Application programming
C++ 32
Delphi 34
LabVIEW 35
Visual Basic 33
Arc motion 91
Contour buffer 92
enable 94
on the fly changes 97
Vector acceleration 91
Vector deceleration 91
Vector velocity 91
At target
commanding 118
description 117
Auto Initialize
loading user defined settings 130
Auxiliary encoder
stepper 143
testing 145
Axis I/O
connector pin out 181, 185
Axis settings
saving user defined settings 122, 129

B

Backlash compensation
description 123
enable 123

C

C++ programming 32
Capture data
actual position 150
DAC output 150
following error 150
optimal position 150
Capture position 140
Closed loop mode
described 20
Closed loop stepper control
described 80

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

homing 107
wiring 80
Coarse home sensor
wiring 60
Coarse Home switch/sensor
voltage range 192
Command format
MCCL 17
Compare output
described 141
mode,
toggle 141
mode, one-shot 141
mode, period 141
Connector
mating connector, J1 - J4 179
mating connector, J8 ..... 64, 189
Connector pinout
MultiFlex PCI 1440 185
MultiFlex PCI 1440 181
Contour buffer
description 92
tell contour count 92
Cubic spline interpolation 97

D

DAC output
plotting 42
Debug application programs 36
Deceleration
default units 86
disable 42
setting 8, 37, 67, 77, 86
Default settings 219
Delphi programming 34
Derivative gain
description 13
Digital I/O
configuraing as an overtravel limit 14
configuring 167
connector pin out 181, 185
description 165
reconfigure 148, 166
specifications 176
testing 167
turn off 168
turn on 168
Digital I/O test panels 41
Digital inputs
signal description 195
wiring 61
Digital outputs
signal description 196
wiring 62
Direction
configure for Unipolar PWM 148
setting 90
Documentation
MultiFlex PCI 1000 Series controllers 3
Driver disable
wiring 53
Driver fault
wiring 55

E

Encoder
auxiliary 143
checkout 65
checkout, stepper 80
description 14
descritpion 24
fault 14, 24, 57
reverse phased 83
rollover 127
wiring, differential 56
wiring, single ended 57
Encoder Index
checkout 105
description 24
Error codes
MCAPI 212
MCCL 213
Error LED's 179
E-stop
enable 125
examples 125
hard wired 125
Example
homing routine 114

F

Fail safe operation
watchdog circuit 163
Fault
encoder 14, 24, 57
Feed forward 71, 120, 162
acceleration 76, 121
calculating 72, 120
deceleration 76, 121
described 71
setting 72, 120
Firmware
update 44
Version 43
Following error
default setting 66
description 66
disable 66
plotting 42
Friction 76
Frictionless servo
using output deadband 76

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

G

Gearing
- enable 100
- setting ratio 100
- terminate 100

H

Home sensor
- checkout 105
- wiring 60, 105

Home switch/sensor
- voltage range 192

Homing an axis
- closed loop stepper 107
- encoder index 109
- home sensor 113
- limit sensor 111, 115
- servo 27, 105, 107
- stepper 27
- stepper, open loop 27, 112

Host interrupt support
- limit switches 103

I

I/O Configuration Panel 14

Integral gain
- description 13
- disable while moving 156

Interrupt PCI host
- limit switches 103

J

Jogging
- description 101

Joystick controlled motion 101

L

LabVIEW programming 35

Learning points 133

LED's
- error 179

Limit switch/sensor
- voltage range 192, 193

Limiting the servo command output 154

Limits
- active level 103
- checkout 102
- disable 102
- enable 102
- hard (switch / sensor) 102
- homing an axis 111, 115
- inverting active level 102, 103
- normally closed switch 102, 103
- programmable 102
- TTL vs. opto isolated 14
- wiring 58, 59, 102

Linear interpolation 91
- Contour buffer 92
- enable 93, 133
- on the fly changes 97
- specifying 91
- Vector acceleration 91
- Vector deceleration 91
- Vector velocity 91

M

Macro command
- as background task 137
- defining 135
- described 135
- memory size 136
- reporting 135
- resetting (deleting) 136
- single stepping a program 152
- volatile 136

Manual positioning 101

Master / Slave
- description 100
- enable 100
- slave ratio 100
- termination 100

Mating connector
- J1 - J4 179
- J8 ..... 64, 189

MCAPI
- Version 43

MCCL command
- move absolute 7

MCCL commands
- single stepping a program 152

MCCL mnemonic
- MA 7

MCSpy
- debug application programs 36

Minimum PC requirements 6

Motion complete
- at target 117
- description 117
- trajectory complete 117

Motion control
- backlash compensation 123
- Constant velocity move 90
- Contour move 91
- Learning / Teaching points 133
- Master / Slave 100
- Point to point 89
- required settings 86
- theory of operation 13, 14

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Torque mode 154
Motion Control
defined 11
Motion Integrator
description 40
digital I/O 167
encoder checkout 65
encoder index checkout.....105
home sensor checkout 105
limit sensor checkout 102
troubleshooting 201
Motor control output
limiting.....154
Move
absolute 7
Moving motors
Motor Mover program 77, 85
required settings 18
Servo motor 65
stepper motor 78, 79
MultiFlex command (MCCL)
description 17
pausing a command / sequence 19
repeating.....18
single stepping.....152
terminating a command / sequence 19
MultiFlex motion command language (MCCL)
format.....17
MultiFlex PCI
documentation 3
resetting.....151
MultiFlex PCI 1440
connector pinout 181, 185
Multiple moves sequences
servo tuning 70
Multi-tasking
commands not supported.....137
CPU utilization 138
described 137
example 7, 137, 138, 139
global data registers 138
passing data between.....138
private data registers 138
termination.....139
testing 137

N

Normally closed limit switch 102, 103

O

On the fly changes
arc and linear motion 97
Constant velocity motion 119
Point to point.....119
Trapezoidal velocity profile 119
Open loop mode
described 20
Operating systems 6
Opto isolated inputs
wiring 58, 59, 60

P

Parabolic velocity profile
description 88
Pausing
MCCL command / sequence.....19
PC requirements
minimums 6
Phasing
output/encoder 66
PID digital filter.....*See* Tuning the servo
algorithm.....13
'D' term 13
description 13
'I' term 13
'P' term 13
restoring settings.....122, 129
Pin out
Analog inputs.....181, 185
Axis I/O.....181, 185
Digital I/O.....181, 185
Watchdog relay 189
Point to point motion
execution 89
Position
Recording 150
Position capture
description 140
Position Capture
signal description 193
Position compare
description 141
fixed increment distances.....141
user defined positions 141
Position Compare
signal description 194
Position mode
enable.....89
Position verification
open loop stepper 21, 79
stepper 143
Potentiometers:.....180
Program samples.....8, 32, 33, 34, 35, 37
Proportional gain
description 13
Pulse command servo
described.....20, 175
PWM command
description 147
wiring 49
PWM direction
configure.....148

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

R

Recording position data 150
Registry
  updating I/O configuring 167
Relay
    reset 151
Repeating
  command or sequence 18
Report
  axis 'at target' 118
  captured data 150
  current position of axis 17, 18, 79
  status of axis 103, 104
  trajectory complete 117
Reset
  relay 151
  the controller 151
Restore
  controller settings 122, 129
Restoring user defined axis settings 122
Reverse phased
  encoder 83
Rollover
  encoder 127

S

Saving user defined axis settings 122, 129
Scaling
  defining user units 159
S-curve velocity profile
  description 88
Servo - Pulse control
  described 20
  specifications 175
  wiring 51
Servo command output
  limiting 154
Servo control
  description 13
  specifications 174
Servo motor control
  homing 27, 105, 107
  tuning the servo 68
Single stepping a program 152
Software
  Demo programs 37, 38
  Flash Wizard 44
  Game port joystick 44
  Motion Integrator 40, 102, 167, 201
  Motor Mover 77, 85
  On-line help 38
  Servo Tuning utility 68
  source code 37, 38
  Status Panel 45, 67, 103, 117
  WinControl 43, 152, 213
Specifications
  analog command axis 174
  analog inputs 176
  digital I/O 176
  pulse command axis 175
  stepper control 175
Status LED's 179
Status Panel utility 67, 103, 117
Stepper motor
  reverse phased 83
Stepper motor control
  changing the direction of motor 79
  closed loop 80, 107
  encoder position verification 21, 79
  homing 27, 112
  open loop 78
  specifications 175

T

Teaching points 133
Terminating
  MCCL command / sequence 19
Testing
  digital I/O 167
Torque mode amplifier 25
Trajectory complete
  description 117
Trajectory generator
  description 13, 14
Trapezoidal velocity profile
  description 88
Troubleshooting
  axis reverse phased 22
  communications 203
  encoder checkout 65
  encoder checkout, stepper 80
  general 202
  initialization 204
  no motion by a servo 22
  oscillation by a servo 22
  servo motion 206, 208
  servo tuning 205
  status LED's 179
Troubleshooting application programs
  MCSpy 36
TTL inputs
  wiring 61
TTL outputs
  wiring 62
Tuning the servo
  description 22, 68
  multiple move sequences 70
  range of slide controls 70
  saving settings 69, 76
  Servo tuning utility 68
  Velocity mode amplifier 71

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

U

Update
  firmware 44
User units
  controller time base 160
  description 159
  machine zero 161
  output constant 162
  part zero 161
  setting 159
  trajectory time 160
  user scale 159, 160

V

Vector acceleration 91
Vector deceleration 91
Vector velocity 91
Velocity
  default units 86
  disable 42
  restoring settings 122, 129
  set too high 66
  setting 8, 37, 77
Velocity gain 162
Velocity mode
  enable 90
Velocity mode amplifier 25
  description 71, 120
  tuning 71
Velocity mode move
  execution 90
  setting the direction 90
  starting 90
Velocity profiles
  Contour mode motion 91
  Parabolic 88
  S-curve 88
  Trapezoidal 88
Velocity, maximum
  setting 67, 86
Version
  firmware 43
  MCAPI 43
Visual Basic programming 33

W

Wait
  for 'at target' 118
  for trajectory complete 117
Watchdog circuit
  description 163
Watchdog relay
  contacts pin out 189
  wiring 64
Windows
  registry, updating I/O configuring 167
Wiring
  +/- 10V command output 48
  A/D inputs 63
  Anplifier Enable output 52, 54
  Anplifier Fault input 55
  closed loop stepper 80
  Driver Disable output 53, 54
  Driver Enable output 54
  Driver Fault input 55
  encoder, differential 56
  encoder, single ended 57
  E-stop 125
  home sensor 105
  home sensor inputs 60
  Limit +/- inputs 58, 59
  limit sensor 102
  Opto isolated inputs 58, 59, 60
  Pulse command output 51
  PWM command output 49
  servo, analog command 48, 51
  servo, pulse command 51
  servo, PWM command 49
  stepper, pulse command 51
  TTL digital inputs 61
  TTL digital outputs 62
  watchdog relay 64

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Precision MicroControl Corporation
2075-N Corte del Nogal
Carlsbad, CA 92009-1415 USA

Tel: (760) 930-0101
Fax: (760) 930-0222

www.pmccorp.com

Information: info@pmccorp.com
Technical Support: support@pmccorp.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP