Instruction Manual for Motion Server & ASCII Command Interpreter

March, 1999

Douloi Automation, Inc.
3517 Ryder Street
Santa Clara, CA 95051-0714

Voice (408) 735-6942
Fax (408) 735-6946
EMail Info@douloi.com

Table of Contents

Chapter 1

1) Introduction

Welcome!

Welcome to Motion Server and Douloi Automation's Motion Control software components, tools to simplify and accelerate the creation of motion control applications.

Douloi Automation wants to encourage your project's success. Free technical support is available to answer your questions, assist you through trouble-spots in product use, and to recommend strategies and approaches for solving different aspects of a motion control problem. Sample code, application prototypes, and application notes can be provided to respond to specific questions you may have. We would much rather have you call and get answers than to be frustrated or slowed in your automation project. Please feel free to contact us at:

- voice (408) 735-6942
- fax (408) 735-6946
- EMail info@douloi.com

Additional information is also available at Douloi's web site, www.douloi.com

Objective of Document

This document provides information on the use of the ASCII Command Interpreter for directing Motion Server controllers. The ASCII command interpreter requires the DMS-AUX-P1 accessory to supplement Motion Server with a serial port. ASCII commands can be used from any language system that allows reading and writing to the serial port of the DMS-AUX. For instructions on setting up the controller please consult the setup chapter in the Instruction Manual for Motion Server and Servo Application Workbench.

Motion Server Specifications

Motion System

- 486 DX/2, DX4 or 5x86 Processor
- 4, 8, 12, or 16 axes per system
- Servo or Stepper on per-axis basis
- Multiple independent axis groups
- Trapezoidal and S-Curve profiling
- Custom profiling at application level
- 32 bit position management
- Sample rates from 1 to 4 kHz
- Linear, circular, curve interpolation
- Electronic gearing with phase adjust
- Electronic camming
- Tangent servo
- Master/slave coordination
- High speed registration
- Kinematics
- Motion superposition
- Coordination tailoring

Servo Specifications

- 486 class processor
- On-board real-time operating system supporting 12 seperate activities as well as motion control
- 4 to 16 axis of coordinated motion
- Mixed servo and stepper motor control
- 32 bit position resolution
- 48 general purpose configurable I/O
- 1 Capture signal per axis
- User Disable signal
- 2 amp enable signals per axis, one active high, the other active low
- watchdog safety system

Servo Capabilities

When configured to run a servo motor the hardware provides

- 4 MHz quadrature inputs with 3 bit filters for 4 axis, 1 MHz quadrature rate for 16 axis
- high speed position capture
- high speed position compare
- +/- 10 volt command signal with 12 bit resolution

Stepper Capabilities

When configured to run a stepper motor the hardware provides

- 2 Mhz step rate for 4 axis, 500 kHz step rate for 16 axis
- configurable step pulse polarity

Timer Event

Motion Server provides motion control functions by responding to a timer which occurs generally at 1 kHz although the frequency is programmable. This timer event performs three major functions.

The first function is control law execution. Servo control is accomplished with the familiar zero, pole, integrator filter used in many motion control systems. This PID control law operates at a 1 kHz sample rate providing comfortable closed loop system frequencies of 100 Hz and below. Stepper motor control is accomplished by updating pulse generating electronics at a frequency of of 1 kHz providing continuous velocity control of stepper motors.

The second function of the timer event is motion profiling. Motion Server is able to profile motion for up to 16 physical axes. These axes can be combined in different arrangements to form various coordinated multi-axis groups. Any particular axis group can perform coordinated motion along an arbitrary path. Multiple axis groups can perform motion concurrently and independently. The motion profiler uses a dynamic profiling technique which permits changing profile parameters on the fly including acceleration, deceleration, slew speed, and in some cases destination and motion type. This permits motion mode "splicing" without stopping. For example a positioning move can be changed to a jog at a new speed on the fly.

The third timer event function is multitasking. Multiple user-written motion application programs may be resident in Motion Server. The timer event contains a multitasker which activates and manages the operation of these programs.

Multiple Motion Application Threads

As many as 12 separate motion application "threads" or programs (which are distinct from motion profiles) can be running concurrently and independently at any particular time. These programs are written in Douloi Pascal, a dialect of Object Pascal. Programs can communicate to each other through shared data structures. They can also access the motion control system, communicate to I/O boards in the PC I/O expansion bus, communicate with Windows applications created by the Servo Application Workbench, and to the disk file system if SAW is present.

Microsoft Windows

Microsoft Windows serves as the most common development and target environment for motion control applications using Douloi products. The familiar interface aids both developers and users of the resulting applications reducing the developers learning curve and the operators training time. Motion Server can be used with other operating systems through various communication methods available. Applications programs for downloading into Motion Server are prepared with Servo Application Workbench or the SERVOLIB.DLL under Microsoft Windows. Once downloaded and retained in on-board FLASH memory, these functions can be invoked from other communication methods.

Long-Slot ISA Format

Motion Server occupies a single ISA "long" slot. The end of the Motion Server card furthest from the mounting bracket holds the on-board 486 processor. The heat sink (and possible fan assembly for 5x86 models) protrudes from the board further than the board-to-board spacing preventing the placement of another long-slot card immediately in front of Motion Server, however shorter cards can fit if necessary. Host performance does not effect Motion Server's real-time performance, however some operations are performed by the host on behalf of Motion Server when Motion Server is interacting with SAW and sends "mail" requesting that these operations are done on its behalf. Performance of these "non-real-time" operations is enhanced with a faster host.

Servo Application Workbench

Servo Application Workbench (SAW) is a Windows application which greatly simplifies the creation of multithreading motion application programs and operator control elements to direct them. Applications may contain conventional Windows controls such as buttons and text items as well as more specialized controls such as components available in the on-line software catalog.

Inside Servo Application Workbench is a high level language compiler. The compiler changes the descriptions of the motion applications into native 32 bit 486 object code which executes on Motion Server very quickly. The compiler “knows” about the motion system, the multithreading system, and Windows. This permits the application developer to access different system resources in a consistent way without having to worry about how these resources are being provided.

Servo Application Workbench allows the developer to construct motion applications in a “clip art” fashion by pasting pre-fabricated parts and assemblies into the application. After “screen painting” the application and filling in the program’s behavior Servo Application Workbench compiles the motion application programs and creates the associated Windows application to operate them. This ability to create new real-time behavior and download into Motion Server is constrained to the Windows environment because the language compiler is a Windows DLL. However, new motion controller capabilities (beyond the standard command set) can be created in SAW, downloaded into Motion Server, and remembered in "flash" memory for use under another operating system.

Methods Of Use

Motion Server can be used in a number of ways. Certain capabilities are available only in certain development methods. The following sections describe resources available.

Servo Application Workbench

Servo Application Workbench is the easiest method for development of real-time machine behavior. This behavior can be "downloaded" into the controller and invoked from a control panel also written in SAW, from other Windows programs, or from binary or ASCII commands.

SERVOLIB.DLL Library

Dynamic Link Libaries are a common and simple method of adding features to any Windows language system. SERVOLIB provides procedures and functions to control the Motion Server hardware. As well as motion commands it is possible to use SERVOLIB to compile and download independent Motion Server behavior to run concurrently with the Windows application. Examples of "direct" access and "combined" access are discussed in the chapter illustrating the use of SERVOLIB in the Instruction Manual for Motion Server and Servo Application Workbench. The command reference for SERVOLIB is in the SAW help file.

ASCII Commands

ASCII commands provide a simple method of accessing the basic functions of Motion Server including single axis and multiple axis coordinated motion and I/O. Characters are sent through a FIFO-style hardware and responses collected after noting that the information is available through a status register. Example programs illustrate how to implement the handshake protocol in several languages.

Binary Commands

Binary Commands provide low-level "register" access to Motion Server without requiring that position information and command parameters be converted into an ASCII format. Binary commands can be sent under any operating system and language system that supports reading and writing the PC I/O space. Example drivers are provided for C and Pascal.

The Binary Command Interpreter is itself a Servo Application Workbench application program, stored as an auto-starting factory default application. The interpreter can be altered, extended, and tailored to meet application specific needs through Servo Application Workbench. Consult Douloi for additional details.

Chapter 2

2) ASCII Communication Protocol

Purpose

The following chapter describes details of how to communicate to Motion Server using the RS-232 communication channel. Command details are provided in chapter 3.

Serial Port Configuration

Motion Server communicates through the an RS-232 serial port at 9600 baud, no parity, 1 stop bit. The UART supports a 16 bit incoming and outgoing fifo. There is no hardware or software handshake protocol. Connecting Motion Server to a WINTEL PC requires a "null modem" style cable or adaptor. Note that terminal hardware may require loopback or hardware enabling to satsify host hardware protocol.

Procedure Message Format

In general messages involve 3 letter commands. Some commands are in the form

<procedure> <parameters> <CR>

There might be no parameters, one, or several depending on the function. Parameters are seperated by spaces. The space between the procedure and the first parameter is optional. Adding a space improves readability. The command is terminated with a carriage return (ASCII 13) character.

After receiving a command, Motion Server will transmit the following information:

<CRLF> <error number> <return value> <prompt>

The prompt is the "greater than" symbol. The error number indicates if the command was completed successfully. A "0" indicates a success, other numbers represent errors. If the first number is not 0, the remaining information in the response is not valid. After the error number is a space

followed by the return value of the command. All commands have return values. Most return values reflect controller state. Some return values confirm information that was sent when no specific return value is associated with the command. The response terminates with the "greater than" symbol as the default prompt.

Hooking up a terminal, or using a terminal program on the PC permits exercising the board and viewing this transaction behavior. Typing just the "CR" key produces a null command which returns 0 for error and 0 for return value:

```
> {type "enter"}
0 0>
```

The command RWD (Reset Watchdog) which has no parameters can be typed with the following response:

```
>RWD {enter}
0 1>
```

The first 0 indicates no command syntax error was encountered. The "1" return value indicates that the command did indeed reset the watchdog. The value "1" corresponds to "true" and "0" corresponds to false.

An example of a command which requires parameters would be the output command, SetOutputBit (SOB). The first parameter is the bit number and the second the desired output value.

```
> SOB 27 1
0 1>
```

Parameters are separated by spaces or commas.

Object Message Format

Another message form reflects the "object oriented" nature of the on-board software system and has the following form:

<receiver> <verb> <parameters>

The receiver can be either a single motor axis or a coordinated group of axes. Individual axes are presented as the "A" array numbered 1 to 16 (A is for "Axis"). An example of an object message to an individual axis would

be the MTT command, SetMotorType:

```
>A1 MTT SRV
0 0>
```

Here the "receiver" is A1, axis 1, the first "element" in the 16 xis array. MTT is the "verb" which acts on the "receiver". SRV is a named constant. The parameter could have been a number also.

Groups of axes are represented as the "C" array numbered 1 to 10 ("C" is for Coordinated group). Before a group can be directed, it must be described with the INI command. The INI command takes as parameters the axis numbers that constitute the coordinated group. For example, a two axis coordinated group could be associated with group 1 as follows:

```
>C1 INI 1 2
0 2>
```

Ten groups are available. Here the first group is being described as running the first and second axis on the controller board. The answer, "2" corresponds to the dimension of the group which should be the same as the number of parameters provided. Motion Server supports up to 6 axes of coordinated motion.

Many commands apply to both individual axes as well as to groups. For example, turning on the motors for axis 5 and for group 1 would be done with the same command, just different receivers:

```
>A5 MTR ON
0 1>C1 MTR ON
0 1>
```

Performing motion requires configuring the board for the motors being used, setting motion attributes, turning motors on, and directing their motion. Examples at the end of the manual show from start-to-finish examples of what's required to produce motion and interact with on-board IO.

Chapter 3

3) Command Reference

Command Summary

IO Operations

INB	InputBit	Return level of specified input
CIO	ConfigureIOBitAsOutput	Instructs I/O bit to behave as output signal
SOE	SetOutputEnable	Tell output bits to become active
SOB	SetOutputBit	Change state of output bit on hardware

Safety

RWD	ResetWatchdog	Allow tripped safety system to resume servo activity
WHT	WatchdogHasTripped	Returns status of watchdog system
UHD	UserHasDisabled	Indicates if any disable input is asserted

Axis and Coordinated Group Commands

Attributes

INI	Init	Associate axes into coordinated group
DSP	Dispose	Release axis group relationship
RSA	ResetAllocation	Clear all group relationships
MTT	MotorType	Configures motor for servo or stepper operation
ENA	Enable	Allow amplifier to power motor
MTR	Motor	Turns motor operation on and off
LIV	LoopInversion	Include an additional sign inversion in control law
CIV	CoordinateInversion	Reverse which way is regarded as the positive direction
ACL	Accel	Set acceleration rate for trapezoidal moves
DCL	Decel	deceleration rate for trapezoidal moves
SPD	Speed	Set speed of slew phase of trapezoidal moves
GAI	Gain	Set compensation parameter for servo
ZER	Zero	compensation parameter for servo
ITG	Integrator	compensation parameter to eliminate steady state error
ERL	ErrorLimit	Set permissible tracking error before disable occurs
PLT	PositiveLimit	Set boundary for movement in the positive direction
NLT	NegativeLimit	Set boundary for movement in the negative direction
ACP	ActualPosition	Define current position coordinate
CAP	CapturePosition	Return position recorded when latch event occurred
COP	CommandedPosition	Set commanded position for non-trapezoidal moves
DEP	DestinationPosition	Return absolute coordinate of end of move
ERP	ErrorPosition	Return discrepency between current and ideal position
AIC	ArmInputCapture	Prepares axis to latch position based on input signal

AXC	ArmIndexCapture	Prepares axis to latch position based on index signal
TRQ	CommandedTorque	Set output voltage when not servoing
PFV	ProfileVelocity	Return current ideal profile velocity

Motion

MVT	MoveTo	Move to absolute coordinate
MVB	MoveBy	Move to relative coordinate
BMT	BeginMoveTo	Start move to absolute coordinate
BMB	BeginMoveBy	Start move to relative coordinate
MAC	MoveAlongCurve	Perform coordinated multiaxis motion along curve
BMC	BeginMoveAlongCurve	Begin coordinated curved motion
AMT	AppendMoveTo	Add absolute vector segment to curve description
AMB	AppendMoveBy	Add vector segment to curve relative to last segment
ARC	AppendArc	Add circular or helical arc to continuous path curve
CLR	Clear	Erase any established motion curve info
LNK	LinkToBuffer	Assocate curve buffer with axis group
JOG	Jog	Move indefintely at constant speed
STP	Stop	Gently stops any motion that may be in progress
BST	BeginStop	Begins to stop but immediately does next instruction
ABT	Abort	Suddenly aborts any motion that my be in progress
CHT	CaptureHasTripped	Indicate if latch event has occurred
MIF	MoveIsFinished	Return true if move has finished

User Task Control

BUT	BeginUserTask	Spawn independent application behavior in controller
AUT	AbortUserTask	Terminate an independent behavior in controller
SUT	ScheduleUserTask	Spawns independent period application behavior
PUT	SuspendUserTask	Cause task in controller to become inactive
RUT	ResumeUserTask	Cause suspended activity to become active again
UTP	UserTaskPresent	Indicates if particular task is currently present in controller

User Variable Control

USB	UserBoolean	Manipulate user boolean in controller
USL	UserLongint	Manipulate user longint in controller
USS	UserSingle	Manipulate user single in controller

ABT (Abort)

Syntax

```
<axis or group> ABT
```

Description

Abort immediately and abruptly stops motion without a controlled decel. Abort is generally for emergency use. Note that an abort at high speeds will most likely cause a servo tracking error resulting in the servos shutting down. This problem can be solved by increasing the error limit with SetErrorLimit or using the dms_Stop command instead. The return value for abort is the group dimension.

Examples

```
A1 ABT
C4 ABT
```

See Also

BeginStop
Stop

ACL (Accel)

Syntax

```
<axis or group> ACL <value>
```

Description

SetAccel is used to set the acceleration of a profiled move in counts per second squared. If the receiver is a T1Axis the acceleration is for the movement of that motor when operating alone. If the receiver is an axis group , for example a T2Axis or T4Axis, the acceleration applies to the coordinated motion profile of the group. SetDecel may be used to independently set the deceleration of the TNAxis.

Accel sets and returns the current setting of the acceleration that will be used by this axis group during trapezoidal moves. The units are in counts per second squared.

Examples

```
A1 ACL 2000      ; sets acceleration to 2000 counts/sec^2
C2 ACL 3000      ; sets vector acceleration of group
A14 ACL          ; returns current accel without assigning
```

See Also

TNAxis.Decel
TNAxis.Speed
TNAxis.SetAccel
TNAxis.SetDecel
TNAxis.SetSpeed

ACP (ActualPosition)

Syntax

```
<axis> ACP <value>
```

Description

ActualPosition sets and returns the current position coordinate of the T1Axis receiver. This is often used when producing plots of the dynamic response of the motor. Note that this may well be different from the CommandedPosition of the motor, ie the theoretical position of where the motor should be. In some cases it may be more desirable to use the CommandedPosition rather than the Actual position. The value returned is in units of counts.

Examples

```
A1 ACP 0         ; Sets axis 1 actual position to 0
A1 ACP           ; retreives actual position of axis 1
```

See Also

CommandedPosition

AIC (ArmInputCapture)

Syntax

```
<axis> AIC
```

Description

ArmInputCapture is used to setup the system to respond to an input pulse that is anticipated. ArmInputCapture resets the capture latches for the axis associated with the TNAxis machine. When CaptureHasTripped the CapturePosition information is valid and can be used by the motion application. Each axis has a specific input used for high speed capture. The return value contains the axis number armed.

Examples

```
A1 AIC
...
A14 AIC
```

See Also

Capture Inputs
ArmInputCapture
Capture
Capture Inputs
CaptureHasTripped
CapturePosition

AMB (AppendMoveBy)

Syntax

```
<group> AMB <value> <value> ... <value>
```

Description

AppendMoveBy adds an additional descriptive point, expressed in relative coordinates, to the end of a curve description. The number of parameters corresponds to the dimension of the TNAxis.

Examples

```
C1 INI 1 2
C1 LNK
C1 AMB 1000 2000
C1 AMB 2000 2000
C1 MAC

C2 INI 5 6 7 8
C2 LNK
C2 AMB 1000 1000 1000 1000
C2 AMB 1000 1500 1500 2000
C2 MAC
```

See Also

Curved Trajectories
TNAxis.MoveAlongCurve
TNAxis.AppendMoveTo
TNAxis.AppendMoveToVector
TNAxis.AppendMoveByVector

AMT (AppendMoveTo)

Syntax

```
<group> AMT <value> <value> ... <value>
```

Description

AppendMoveTo adds an additional descriptive point, expressed in absolute coordinates, to the end of a curve description. The number of parameters corresponds to the dimension of the TNAxis.

Examples

```
C1 INI 1 2
C1 LNK
C1 AMT 1000 2000
C1 AMT 2000 2000
C1 MAC

C2 INI 5 6 7 8
C2 LNK
C2 AMT 1000 1000 1000 1000
C2 AMT 1000 1500 1500 2000
C2 MAC
```

See Also

Curved Trajectories
TNAxis.MoveAlongCurve
TNAxis.AppendMoveBy
TNAxis.AppendMoveToVector
TNAxis.AppendMoveByVector

ARC (AppendArc)

Syntax

```
<group> ARC <radius> <initial angle> <delta angle> <deltaz>
```

Description

AppendArc adds a circular segment to the continuous path being constructed. The first parameter is the radius of the arc. The InitialAngle indicates, in degrees, the tangent angle of the beginning of the arc. The DeltaAngle indicates how many degrees of rotation should occur. Note that DeltaAngle can indicate more than 360 degrees of rotation. Negative delta angles indicate curves to the right. Positive delta angles indicates curves to the left. Angles are measured with the X pointing in direction 0 and Y pointing in direction 90.

Examples

The following commands illustrate how to produce a continuous path:

```
C1 INI 1 2
C1 LNK
C1 AMB 1000 0
C1 ARC 1000 0 90
C1 AMB 0 1000
C1 BMC
```

SeeAlso

Curved Trajectories
TNAxis.MoveAlongCurve
TNAxis.AppendMoveTo
TNAxis.AppendMoveToVector
TNAxis.AppendMoveByVector

AUT (AbortUserTask)

Syntax

```
AUT <task number>
```

Description

AbortUserTask causes an application task in the controller to begin cease executing as independent thread. Aborting a task which is not currently running is not considered an error. AbortUserTask really means "insure that this indicated task is not active". Not finding the task active indicates it is inactive already.

Examples

```
AUT 1
AUT 10
```

See Also

dms_BeginUserTask
dms_ScheduleUserTask
dms_UserTaskPresent
dms_SuspendUserTask
dms_ResumeUserTask

AXC (ArmIndexCapture)

Syntax

```
<axis> AXC
```

Description

ArmIndexCapture is used to setup the system to respond to an index pulse that is anticipated. ArmIndexCapture resets the capture latches for the axis associated with the TNAxis machine. When CaptureHasTripped the CapturePosition information is valid and can be used by the motion application. The return value contains the axis number assigned.

Examples

```
A1 AXC
A15 AXC
```

See Also

ArmInputCapture
Capture
CaptureHasTripped
CapturePosition

BMB (BeginMoveBy)

Syntax

```
<axis or group> BMB <value> <value> ... <value>
```

Description

BeginMoveBy starts relative coordinated move by the specified position deltas but does not wait for the move to finish. In actual use the "N" in TNAxisBeginMoveBy is replaced by the dimension of the controlled group. For example, a 2 axis call would be T2AxisBeginMoveBy. The method requires as many parameters as the dimension of the receiver axis group, ie a 2 axis group requires 2 parameters, a 4 axis group requires 4 parameters. The motion is performed with a trapezoidal velocity profile based on parameters set with the SetAccel , SetDecel , and SetSpeed methods. These parameters apply to the vector path motion of the coordinated group rather than to any particular axis. BeginMoveBy returns immediatly and does not wait for the motion to finish. For cases where it is important to “blocking” program execution until the end of the move use MoveBy instead of BeginMoveBy. Use MoveHasCompleted to determine when a move started with BeginMoveBy has finished.

Group Numbers required to perform the call is provided by the TNInit functions

Errors

BeginMoveBy will escape if while in motion the resulting destination specified is “behind” the vector path position or if the destination is so close that the axis group cannot accomplish the move at the specified decel rate. In these cases the group will emit a MotionOverrunEscapeCode and come to a stop.

Examples

```
C2 INI 1 2 3
C2 MTR ON
C2 BMB 1000 2000 3000
```

SeeAlso

TNAxis.SetAccel
TNAxis.SetDecel
TNAxis.SetSpeed
TNAxis.MoveIsFinished
TNAxis.MoveBy
TNAxis.MoveTo
TNAxis.BeginMoveTo
MotionOverrunEscapeCode

BMC (BeginMoveAlongCurve)

Syntax

```
<axis group> BMC
```

Description

BeginMoveAlongCurve performs continuous path motion over an arbitrary, multiaxis curve description which was previously setup. This routine is not implemented by a T1Axis single axis. Program execution immediately continues after the motion has started. The return value contains the dimension of the group being controlled.

Examples

```
C1 BMC
C5 BMC
```

SeeAlso

Curved Trajectories
TNAxis.MoveAlongCurve
TNAxis.MoveIsFinished

BMT (BeginMoveTo)

Syntax

```
<axis or group> BMT <value> <value> ...<value>
```

Description

BeginMoveTo starts an absolute coordinated move to the specified absolute position destinations but does not wait for the move to finish. The "N" in TNAxisBeginMoveTo is replaced by the dimension of the group being controlled, for example T2AxisBeginMoveTo for a 2 axis group. The method requires as many parameters as the dimension of the receiver axis group, ie a 2 axis group requires 2 parameters, a 4 axis group requires 4 parameters. The motion is performed with a trapezoidal velocity profile based on parameters set with the SetAccel , SetDecel , and SetSpeed methods. These parameters apply to the vector path motion of the coordinated group rather than to any particular axis for multidimensional axis groups. BeginMoveTo returns immediatly and does not wait for the motion to finish. For cases where it is important to “block” program execution until the end of the move use MoveTo instead of BeginMoveTo. Use MoveHasCompleted to determine when a move started with BeginMoveTo has finished.

Group numbers required for this routine are provided by the TNInit functions.

Errors

BeginMoveTo will escape if while in motion the destination specified is “behind” the vector path position or if the destination is so close that the receiver cannot accomplish the move at the specified decel rate. In these cases an escape will occur with MotionOverrunEscapeCode and the receiver will stop.

Examples

```
C4 INI 1 2
C4 MTR ON
C4 BMT 100 100
```

SeeAlso

TNAxis.SetAccel
TNAxis.SetDecel
TNAxis.SetSpeed
TNAxis.MoveIsFinished
TNAxis.MoveBy
TNAxis.MoveTo
TNAxis.BeginMoveTo
MotionOverrunEscapeCode

BST (BeginStop)

Syntax

```
<axis or group> BST
```

Description

BeginStop directs the axis group to slow down at the specified decel rate and stop motion. A TNAxis group will remain coordinated during the stop. The calling program will not wait until after the stop has finished before continuing but will immediately execute the next statement. The return value contains the group dimension being stopped.

Examples

```
A1 BST
C3 BST
```

See Also

TNAxis.Stop
TNAxis.Abort

BUT (BeginUserTask)

Syntax

```
BUT <task number>
```

Description

BeginUserTask causes an application task in the controller to begin executing as an independent thread. Tasks are saved in controller FLASH memory through the use of the SAVE_APP catalog component. It is necessary to "connect" application tasks to user task numbers through an assignment procedure for access through the binary command interpreter. Consult the chapter on "Using Flash Memory" for more information. The return value contains the task number provided.

Examples

```
BUT 1

...

BUT 10
```

See Also

dms_AbortUserTask
dms_ScheduleUserTask
dms_UserTaskPresent
dms_SuspendUserTask
dms_ResumeUserTask

CAP (CapturePosition)

Syntax

```
<axis> CAP
```

Description

CapturePosition returns the position the axis experienced the capture event, either an index pulse of an input, which was anticipated using the ArmIndexCapture or ArmInputCapture instructions. The CapturePosition is only valid if CaptureHasTripped.

Examples

```
A1 CAP
SAY A5 CAP
```

See Also

ArmIndexCapture
ArmInputCapture
Capture
Capture Inputs
CaptureHas

CAT (CaptureHasTripped)

Syntax

```
<axis> CAT
```

Returns

1 = Capture has tripped, 0 = Still waiting for capture to trip

Description

CaptureHasTripped returns true if the index or input event, configure by ArmIndexCapture or ArmInputCapture, has occurred. If CaptureHasTripped then CapturePosition is valid and contains the position where the event occurred.

Examples

```
A1 CAT
A4 CAT
```

See Also

TNAxis.ArmInputCapture
TNAxis.ArmIndexCapture
Tripped

CIO (ConfigureIOBitAsOutput)

Syntax

```
CIO <bit number> <IsOut>
```

Description

Motion Server I/O can be configured as inputs or outputs in 4-bit nibble sized groups. Groups are indicated in the connector diagram later in this document. After reset I/O defaults to inputs with 4.7k pullups to prevent asserting an active output signal. Nibble-groups can become outputs by using this command with the Bit parameter being the bit number of any bit in the group, and the IsOut parameter being set to "true". An output group can become an input group by indicating IsOut as false.

Examples

```
CIO 1,1    ;configure bit 1 (and 2,3,4) as output
CIO 5,0 ;configure bit 5 (and 6,7,8) as input
```

SeeAlso

dms_SetOutputBit
dms_SetOutputEnable

CIV (SetCoordinateInversion)

Syntax

```
<axis> CIV <on or off>
```

Description

SetCoordinateInversion is used to change the direction a motor regards as positive. Axis direction is influenced by mechanical transmission reversals, encoder phase definition, and wiring conventions. If the motor does not move in the direction regarded as positive this procedure may be used to invert the direction by calling with a parameter value of true. Using the predefined booleans On and Off may improve the readability of the code. A better design option is to change the wiring, most likely of the A and B channels of the encoder so that the axis moves in the correct direction from the default values rather than having to be "setup" by this procedure call. If that wiring is inconvenient this procedure may be the simplest option. Changing the encoder wires also requires changing the motor wires so as to preserve the loop sign. Note that changing the wires of the motor alone will not have the desired effect but will instead cause the servo loop to go unstable.

This command operates in an incremental manner by inverting the coordinate frame about the current actual position rather than 0. The best time to use this command is during initial setup before homing has been performed. This is not intended to be used during motion.

Examples

```
A1 CIV ON
A2 CIV OFF
```

See Also

T1Axis.SetLoopInversion

CLR (Clear)

Syntax

```
<group> CLR
```

Returns

Group dimension

Description

CLR removes any previous curve information and prepares the TNAxis to receive a new curve description with Append commands..

Examples

```
C2 CLR
C5 CLR
```

SeeAlso

Curved Trajectories
TNAxis.MoveAlongCurve
TNAxis.AppendMoveBy
TNAxis.AppendMoveTo
TNAxis.AppendMoveToVector
TNAxis.AppendMoveBy
TNAxis.AppendMoveByVector

COP (CommandedPosition)

Syntax

```
<axis> COP <value>
```

Returns

```
Current commanded position
```

Description

CommandedPosition returns the theoretical position of the motor, i.e. the desired position of the motor. During the course of a profiled motion this number will smoothly change to represent the trajectory of the motor. Actual motor trajectory will differ from this theoretical expectation due to system dynamics and power limits realized in physical, real-world machines. The commanded position only exists when the motor is servoing. If the servo is not active the CommandedPosition is a meaningless number. For multidimensional axis groups the commanded position is the vector path length into the move or curve relative to the beginning of the curve. This can be used to perform events at particular positions along a multidimensional trajectory.

Examples

```
A1 COP 300
SAY A1 COP
SAY C3 COP
```

See Also

T1Axis.ActualPosition
TNAxis.GetActualPositionVector
TNAxis.GetCommandedPositionVector

DCL (Decel)

Syntax

```
<axis or group> DCL <value>
```

Description

Decel returns the current setting of the deceleration that will be used by this axis group during trapezoidal moves. The units are in counts per second squared.

Examples

```
A1 DCL 10000
C2 DCL 20000
A1 DCL
C2 DCL
```

SeeAlso

TNAxis.Accel
TNAxis.Speed
TNAxis.SetAccel
TNAxis.SetDecel
TNAxis.SetSpeed

DEP (DestinationPosition)

Syntax

```
<axis or group> DEP
```

Description

DestinationPosition returns the absolute coordinate of where a move will finish. This can be used to calculate the distance remaining in a move, move often used to overlap motion and reduce cycle time.

Examples

```
SAY A2 DEP
SAY C5 DEP
```

See Also

CommandedPosition
ActualPosition

DSP (Dispose)

Syntax

```
<group> DSP
```

Description

The Motion Server command set is designed to support multiple client programs. Coordinated motion is described by using GroupNumbers provided by TNAxisInit routines. When a client program is done and exiting the client needs to tell Motion Server that it is finished with the resources that were allocated so that another program can use them. This is done with

dms_TNAxisDispose. dms_TNAxisDispose is analagous to releasing memory after use so as to prevent a "memory leak". If dms_TNAxisDispose is not used, an "axis leak" will occur and eventually Motion Server will indicate that there are no more axis groups available for use.

Example

```
C4 DSP
```

ENA (Enable)

Syntax

```
<axis or group> ENA <on or off>
```

Description

Enable returns true (non-0) if all of the axis in the axis group are enabled. This should generally follow the state requested by SetServo, however servo tracking error can cause one or more axis to automatically shutdown. When a servo is turned off, it's enabled is also turned off to shutdown the amplifier. You can re-enable an amplifier without requesting servo activity by using SetEnable

Examples

```
A1 ENA ON
C2 ENA OFF
A1 ENA
C2 ENA
```

See Also

Servo States
SetEnable
SetServo
ServoIsOn

ERL (SetErrorLimit)

Syntax

```
<axis> ELT <value>
```

Description

SetErrorLimit is used to describe how far a physical axis 's actual position can lag behind the commanded position without that lagging being considered an error. Ideally the motor's actual position exactly follows the commanded position however system dynamics and transient response of the motion control law means that in general this idealistic case is not achieved for arbitrary profiles although it can be closely achieved for non-accelerating profiles. Systems which have high accelerations and decelerations are also likely to incur following error during those times if the power system saturates. If the difference between the actual position and commanded positions exceeds the error limit the axis will perform a TNAxis.SetServo(Off) ;

The error limit is always being checked. Set the limit to be a large value if the SetServo(Off) behavior is not desired.

Example

```
A1 ERL 50
A2 ERL 25
```

See Also

TNAxis.SetMotor
TNAxis.MotorIsOn

ERP (ErrorPosition)

Syntax

```
<axis> ERP
```

Description

ErrorPosition returns the difference between where the servo is commanded to be and its actual position. This difference is monitored. If it is found to be greater than the error limit, the servo is turned off.

Examples

```
A1 ERP          ; returns axis 1 error position
```

See Also

CommandedPosition
ActualPosition

GAI (Gain)

Syntax

```
<axis> GAI <value>
```

Description

Motion Server implements PID control. This function returns the current value of the control law gain, one of the primary compensation parameters.

Examples

```
A1 GAI 32       ; sets axis 1 gain to 32
A2 GAI 40     ; sets axis 2 gain to 40
A1 GAI     ; returns axis 1 gain value
```

See Also

Integrator
SetGain
SetIntegrator
SetZero
Zero

INB (InputBit)

Syntax

```
INB <bit number>
```

Description

InputBit returns true if the input level is high and false if the level is low. Values for bit number are 1 through 48. Asking for the input value of a bit configured as an output returns the current level of the output.

Errors

If a bit number is requested beyond the range for the system then a ParameterOutOfRangeEscapeCode occurs.

Examples

```
INB 4
```

See Also

ConfigureIOBit
SetOutputBit

INI (Init)

Syntax

```
<group> INI <axis number> <axis number> ...<axis number>
```

Description

TNAxisInit is used to associate axes into a coordinated group and returns a Group Number to reference the group in the future. In actual use, the "N" in TNAxisInit is replaced by the dimension of the group being constructed, i.e. T2AxisInit for a 2 axis group or T6AxisInit for a 6 axis group. In coordinated motion commands, the group number is the "handle" that refers to this particular axis association. The axis are specified with their axis numbers ranging from 1 to 16. The routine requiers as many axis parameters as dimension of the group being constructed. The order the axes are indicated here becomes the order of parameters used to describe coordinated motion. The first axis listed here receives the first coordinate number in motion commands. Coordinated motion can only be performed on groups that have been initialized.

Example

```
C4 INI 1 2           ; creates 2 axis coordinated group
C5 INI 3 4 5 6 7     ; creates 5 axis coordinated group
```

ITG (Integrator)

Syntax

```
<axis> ITG <value>
```

Description

Motion Server implements PID control. This function returns the current value of the control law integrator, one of the primary compensation parameters.

Examples

```
A1 ITG 10  ; sets integrator for axis 1 to value 10
A2 ITG 5; sets integrator for axis 2 to 5
A1 ITG  ; returns current value of axis 1 integrator
```

See Also

Gain
Zero
Integrator
SetGain
SetZero
SetIntegrator

JOG (Jog)

Syntax

```
<axis> JOG <speed>
```

Description

Jog directs an axis to move at the specified speed indefinitely. if the magnitude of aSpeed is smaller than the magnitude of the current speed the axis will slow down at the decel rate. If the magnitude is greater it will speed up at the accel rate. It is possible to jog in the opposite direction as the current

speed. It is possible to jog at 0 speed. Jog may supersede a move, changing it into a continuous motion.

Although it is possible to use jog to produce movement when searching for home switches or other input events, it is generally a better idea to move a distance which should include the event so that the behavior of the machine if the event is not found is to stop rather than to travel indefinitely.

Jogging is NOT protected by positive and negative limits. Jogging, by it's nature, is a continuous move. To realize jogging velocity to the edge of a machine movement, perform a move to the positive or negative limit at the required jog speed.

Examples

```
A1 MTT STE
A1 MTR ON     ; turn the motor on
A1 JOG 2000   ; accelerate and move at continuous speed
A1 JOG        ; same as A1 SPD, returns current speed
```

See Also

TNAxis.Stop

LIV (LoopInversion)

Syntax

```
<axis> LIV <on or off>
```

Description

SetLoopInversion is used to add an additional sign change in the feedback loop so as to change the total loop sign. This instruction is provided to compensate for encoder wiring or motor wiring which is not providing the correct feedback sense. A better response to the problem of unstable loop sign is to change the wiring of the motor leads (invert loop sign) or encoder A and B channels (invert coordinate frame and sign) rather than use this instruction since forgetting this instruction in a future application causes the motor to be unstable. AxisNumber must be in the range 1 to 16. Group Numbers are not allowed for this routine.

Examples

```
A1 LIV ON
A2 LIV OFF
```

See Also

T1Axis.SetErrorLimit
T1Axis.SetCoordinateInversion

LNK (LinkToBuffer)

Syntax

```
<axis group> LNK
```

Description

Motion Server supports continuous path motion. Curves are described by appending vectors and arcs to a list associated with the axis group that will perform the curve. To indicate to an axis group that space should be made available for this list use the dms_LinkToBuffer command. There are 2 lists in the standard binary command interpreter. Each list can support up to 500 elements and up to a T6Axis group. If both of these buffers have been used the error code will be set to be_OutOfCurveBuffers. If this occurs the most likely explanation is that dms_TNAxisRelease was not used to deallocate the buffers. Refer to this command, or use the dms_ResetAllocation command to provide a "clean slate" on startup.

Example

The following commands illustrate how to produce a continuous path:

```
C1 INI 1 2
C1 LNK
C1 AMB 1000 0
C1 ARC 1000 0 90
C1 AMB 0 1000
C1 BMC
```

SeeAlso

dms_T2AxisAppendArc
dms_T3AxisAppendArc
dms_T2AxisAppendMoveBy
dms_T2AxisAppendMoveTo
dms_ReleaseAllocation
dms_TNAxisDispose

MAC (MoveAlongCurve)

Syntax

```
<group> MAC
```

Description

MoveAlongCurve performs continuous path motion over an arbitrary, multiaxis curve description which was previously setup. This routine is not implemented by a T1Axis single axis. Program execution does not continue past MoveAlongCurve until the curve has been completed.

Examples

```
C2 MAC
```

SeeAlso

Curved Trajectories
TNAxis.BeginMoveAlongCurve
TNAxis.MoveIsFinished

MIF (MoveIsFinished)

Syntax

```
<axis or group> MIF
```

Description

MoveIsFinished indicates if the TNAxis is currently moving or if the move has completed. The DLL function returns 0 to represent false and non-0 to represent true. This would normally be used after starting motion with a procedure that had a name starting with BeginMove.....

Because of the multitasking options with Motion Server and SI-3000 it is sometimes more convenient to seperate functions into two parts, a motion part which uses “synchronous” motion commands that start with Move, and another part which performs the “background” activity, and to have both functions running at the same time.

Example

Imagine you would like to move a fixed distance with the expectation of hitting a switch along the way. The following routine would perform this check:

```
...
A1 BMB 20000
...
```

A1 MIF

```
...
```

SeeAlso

BeginMoveTo
BeginMoveBy
TNAxis.BeginMoveToVector
TNAxis.BeginMoveByVector
TNAxis.BeginMoveAlongCurve

MTR (Motor)

Syntax

```
<axis> MTR <ON or OFF>
```

Description

MTR is used to turn motor activity on and off for all the axis in the TNAxis. Called with a parameter value of true enables the amplifier lines. The motor servos to the current location (if configured for servo). When called with a parameter value of false the amplifier lines are disabled, the motor command is set to 0 volts (if configured for servo) and no further motor activity occurs. Readability of the program is improved by using the predefined boolean constants On and Off . SetMotor is an alias for the outdated SetServo routine, (retained for backward comaptibility) to acknowledge both stepper and servo motor capability.

Example

```
A1 MTR ON       ; sets axis 1 motor on
C3 MTR ON       ; sets all motors in group on
A2 MTR          ; returns 1 if on, 0 if off
```

Escapes

SetMotor(On) will generate a WatchdogFailedToResetEscapeCode if the WatchdogHasTripped .

See Also

Enable

MTT (MotorType)

Syntax

```
<axis> MTT <SRV or STE>
```

Description

SetMotorType is used to configure a particular axis to run a servo motor or a stepper motor. The AxisNumber parameter must be in the range 1 to 16. Group Numbers are not allowed for this parameter. If the configuration is for stepper, it is also possible to indicate whether the step pulse goes high to indicate a step or goes low.This information is conveyed through the bit mask parameter. The following constants are included to aid in specifying the motor configuration:

```
(ServoMotor) or
(StepperMotor + (HighStepPulse or LowStepPulse))
```

When setting an axis for use as a servo motor, just use ServoMotor as the parameter. When specifying a StepperMotor the parameter is StepperMotor with a pulse width contant and a pulse polarity constant added to it.

Example

```
A1 MTT SRV
A2 MTT STE
A1 MTTPFV  (ProfileVelocity)
```

Syntax

```
<axis or group> PFV <value>
```

Description

ProfileVelocity returns the current commanded speed (signed magnitude) that is being used to generate the trapezoidal motion trajectory. During slew, the magnitude of ProfileVelocity is the same as Speed. During accel and decel the profile velocity varies according to the point in the profile.

Examples

```
A1 PFV 1000     ; sets the current profile velocity
A1 PFV          ; returns the current profile velocity
```

See Also

TNAxis.CommandedPosition
T1Axis.ActualPosition

MVB (MoveBy)

Syntax

```
<axis or group> MVB <value> <value> ... <value>
```

Description

TNAxisMoveBy performs a relative coordinated move by the specified position deltas. In actual use, the "N" in TNAxis... is replaced by the dimension of the group being directed, i.e. T3AxisMoveBy for a 3 axis group. The method requires as many parameters as the dimension of the receiver axis group, ie a 2 axis group requires 2 parameters, a 4 axis group requires 4 parameters. The motion is performed with a trapezoidal velocity profile based on parameters set with the SetAccel , SetDecel , and SetSpeed methods. These parameters apply to the vector path motion of the coordinated group rather than to any particular axis. MoveBy does not return until the motion has been accomplished. “Blocking” program execution until the end of the move may be helpful for synchronizing the next event, ie don’t drill the hole until you get to the destination. Some appllications need to continue execution even though the destination has not yet been achieved. For these cases use BeginMoveBy which starts the move and immediately returns to continue with the next instruction.

Group Numbers are provided by TNAxisInit routines.

Errors

MoveBy will escape if while in motion the new destination specified is “behind” the vector path position or if the destination is so close that the axis group cannot accomplish the move at the specified decel rate. In these cases the group will emit a MotionOverrunEscapeCode and come to a stop.

Example

```
C2 INI 1 2 3 4
C2 MVB 100 100 200 200
```

SeeAlso

TNAxis.SetAccel
TNAxis.SetDecel
TNAxis.SetSpeed
TNAxis.MoveTo
MotionOverrunEscapeCode

MVT (MoveTo)

Syntax

```
<axis or group> MVT <value> <value> ... <value>
```

Description

TNAxisMoveTo performs an absolute coordinated move to the specified destination. In actual use, the "N" in TNAxis is replaced by the dimension of the group, i.e. T2AxisMoveTo. The number of parameters provided is the same as the dimension of the axis group, ie a 2 axis group requires 2 parameters, a 4 axis group requires 4 parameters. The motion is performed with a trapezoidal velocity profile based on parameters set with the SetAccel , SetDecel , and SetSpeed methods. These parameters apply to the vector path motion of the coordinated group rather than to any particular axis. TNAxisMoveTo does not return until the motion has been accomplished. “Blocking” program execution until the end of the move may be helpful for synchronizing the next event, ie don’t drill the hole until you get to the destination. Some appllications need to continue execution even though the destination has not yet been achieved. For these cases use TNAxisBeginMoveTo which starts the move and immediately returns to continue with the next instruction.

Errors

MoveTo will escape if while in motion the destination specified is “behind” the vector path position or if the destination is so close that the axis group cannot accomplish the move at the specified decel rate. In these cases the group will emit a MotionOverrunEscapeCode and come to a stop.

Example

```
C5 INI 1 2 3
C5 MVT 1000 2000 3000
```

SeeAlso

TNAxis.SetAccel
TNAxis.SetDecel
TNAxis.SetSpeed
MotionOverrunEscapeCode
TNAxis.MoveBy

NLT (SetNegativeLimit)

Syntax

```
<axis> NLT <value>
```

Description

SetNegativeLimit establishes a negative-direction boundary for movement. If the axis is asked to attempt a move beyond this boundary, a PositionLimitEscapeCode will occur and the axis will stop.

Examples

```
A1 NLT -5000
A1 NLT
```

See Also

PositiveLimit

PLT (PositiveLimit)

Syntax

```
<axis> PLT <value>
```

Description

SetPositiveLimit establishes a positive-direction boundary for movement. If the axis is asked to attempt a move beyond this boundary, a PositionLimitEscapeCode will occur and the axis will stop.

Examples

```
A1 PLT 20000
```

See Also

SetNegativeLimit

PUT (SuspendUserTask)

Syntax

```
PUT <task number>
```

Description

SuspendUserTask causes a task in the controller to to be placed "on hold". Stack information and task residency is kept, however the task does not perform any operations. ResumeUserTask is required to allow the task to run again.

Examples

```
PUT 1
...
AUT 10
```

See Also

dms_BeginUserTask
dms_ScheduleUserTask

RSA (ResetAllocation)

Syntax

RSA

Description

Motion Server is designed to provide motion services to several clients at one time. In the course of providing these services resources are allocated through the dms_T2AxisInit..dms_T6AxisInit commands and through the dms_LinkToBuffer routine. If a client program terminates and does not dispose of these resources with the dms_TNAxisDispose command, eventually the resources will be consumed and Motion Server will report errors. The procedure dms_ResetAllocation is used to provide a "clean slate" for Motion Server resources. In a multiple client situation, dms_ResetAllocation should not be used as it would "pull the resources out from under" another client program which may be active. If you are developing an application that only involves a single client, dms_ResetAllocation can be used in the startup code to insure a full set of resources is available.

Example

RSA

See Also

dms_TNAxisDispose

RSW (ResetWatchdog)

Syntax

RSW

Description

The motion system operates under the supervision of a watchdog system. If for any reason the processor should be delayed in responding to the motion system's timer event the watchdog system will shutdown the power amplifiers to insure that no undesired motion occurs. ResetWatchdog allows servo activity to occur again.

Errors

If ResetWatchdog discovers that the watchdog did not reset a WatchdogFailedToResetEscapeCode will occur.

Example

```
RSW
```

See Also

WatchdogHasTripped

RUT (ResumeUserTask)

Syntax

```
RUT <task number>
```

Description

ResumeUserTask is used to awaken a task on the Motion Server controller that has been suspended with the PUT command.

Examples

```
RUT 1
...
RUT 10
```

See Also

dms_BeginUserTask
dms_ScheduleUserTask

SOB (SetOutputBit)

Syntax

```
SOB <bit number> <ON or OFF>
```

Description

SetOutputBit is used to set output bits to a prescribed level. BitNumber should be in the range of 1 to 48. Value should be a boolean parameter. The predefined constants On and Off can help improve readability of the program. These bits will only take effect if SetOutputEnable(On) has been used since a hardware reset.

Errors

If the bit number is outside of the allowable range for the system configuraiton a ParameterOutOfRangeEscapeCode will occur.

Examples

```
SOB 1 ON
SOB 2 OFF
```

SOE (SetOutputEnable)

Syntax

```
SOE <ON or OFF>
```

Description

After a hardware reset, the general I/O is configured as inputs and the output drives are tristated. Pullups on the signals will assert a “soft” high level as the default signal. Digital outputs on the axis connector will also be tristated after reset. SetOutputEnable activates the outputs (on signals configured to be outputs) so that SetOutputBit works. SetOutputEnable(Off) tristates the outputs in the same manner that a hardware reset would. The DLL call will escape if there are insufficient tasks available to perform the operation.

Examples

```
SOE ON
SOE OFF
```

See Also

SetOutputEnable
`ConfigureIOBit`

SPD (Speed)

Syntax

```
<axis or group> SPD <value>
```

Description

SetSpeed establishes the slew speed to be used during axis movement. aSpeed is in units of counts/second. Values in the range of 80,000 are brisk. Values in the range of 1000 are slow. The speed of a move may be changed on the fly at any point in a move and take immediate effect if the motion is in the slew phase. For single axis machines SetSpeed effects the speed of the axis. For multiaxis groups SetSpeed effects the vector speeed of the group.

Examples

```
A1 SPD 1000
C2 SPD 20000
```

SeeAlso

TNAxis.SetAccel
TNAxis.SetDecel
TNAxis.MoveTo
TNAxis.MoveBy
TNAxis.BeginMoveTo
TNAxis.BeginMoveBy

STP (Stop)

Syntax

```
<axis or group> STP
```

Description

Stop directs the axis group to slow down at the specified decel rate and stop motion. A TNAxis group will remain coordinated during the stop. The calling program will wait until after the stop has finished before continuing.

Examples

```
A1 STP
C2 STP
```

SeeAlso

BeginStop
Abort

SUT (ScheduleUserTask)

Syntax

```
AUT <task number> <invocation period>
```

Description

ScheduleUserTask causes an application task in the controller to begin executing as an independent thread on a period basis. The first parameter is the user task number. The second parameter is how many sample periods should occur between invocations of the task.

Examples

```
SUT 1 20
...
SUT 10 100
```

See Also

dms_BeginUserTask
dms_ScheduleUserTask

TRQ (CommandedTorque)

Syntax

```
<axis> TRQ <value>
```

Returns

Current commanded torque value, +/- 2047 representing +/- 10 volts

Description

CommandedTorque returns the current amount of torque the servo controller is requesting for the receiving axis. This information is returned as an integer and is in the range of MaxTorque to MinTorque. This function can be used to determine if the axis is continually applying torque to a load or undergoing saturation, (ie constantly requesting the maximum or minimum torque).

Examples

```
A1 TRQ 500
SAY A1 TRQ
```

See Also

T1Axis.SetCommandedTorque

UHD (UserHasDisabled)

Syntax

```
UHD
```

Description

If the User Disable input is not held low this function returns 1 indicating that the user is attempting to disable the system, otherwise it returns a 0

Example

```
UHD
```

See Also

ResetWatchdog
WatchdogHasTripped

USB (User Boolean)

Syntax

```
USB <value>
```

Description

UserBoolean writes and reads information from an array of boolean variables. Information may be placed into this array and read from on-board user tasks allowing data to flow between the host and Motion Server as parameters and return results for user task operations.

Examples

```
USB 10 0        ; sets user boolean 10 to false (0)
USB 11 1        ; sets user boolean 11 to true (1)
USB 10          ; retrieve the value of user boolean 10
```

See Also

USB
USB
BUT
AUT

USL (UserLongint)

Syntax

```
USL <value>
```

Description

UserLongint writes and reads information from an array of 32 bit longint variables. Information may be placed into this array and read from on-board user tasks allowing data to flow between the host and Motion Server as parameters and return results for user task operations.

Examples

```
USL 1 1000      ; set user longint 1 to have the value 1000
USL 1           ; retrieve the value of user longint 1
```

See Also

USB
USB
BUT
AUT

USS (UserSingle)

Syntax

```
USS <value>
```

Description

UserSingle writes and reads information from an array of 32 bit floating point variables. Information may be placed into this array and read from on-board user tasks allowing data to flow between the host and Motion Server as parameters and return results for user task operations.

Examples

```
USS 1  56.7     ; set user longint 1 to have the value 56.7
USS 1           ; retrieve the value of user single 1
```

See Also

USB
USB
BUT
AUT

UTP (UserTaskPresent)

Syntax

```
UTP <task number>
```

Description

UserTaskPresent returns a 1 if the indicated task is currently present on the Motion Server controller and 0 if it is not. A task is considered present if it is currently active, suspended, or scheduled to run even at a low frequency.

Examples

```
UTP 1           ; returns 1 if task 1 was present
```

See Also

USB
USB
BUT
AUT

WHT (WatchdogHasTripped)

Syntax

```
WHT
```

Definition

The watchdog safety system will shut down servo activity if the processor fails to respond to the timer event correctly. This function indicates if the watchdog system has shut down activity.

Example

```
WHT
```

See Also

ResetWatchdog

ZER (Zero)

Syntax

```
<axis> ZER <value>
```

Description

Motion Server implements PID servo control. The zero of a control loop is one of the primary parameters used to set the servo's compensation and primarily relates to the damping of the system. This procedure sets the control law zero to be aZeroValue. Values in the range of 200 to 255 are not unusual. Values greater than 255 are not legal.

Examples

```
A1 ZER 232
A2 ZER 240
```

See Also

Gain
Integrator
SetGain
SetIntegrator
Zero

Chapter 4

4) Command Examples

Objective

These examples illustrate the general pattern of use for Motion Server commands. Examples also illustrate particular controller functions. If there is an example you would like to see that is not present, please contact Douloi for sample code.

Single Axis Stepper Motor Movement

The following commands represent what is required to configure and move an individual stepper motor:

```
A1 MTT STE          ; Set axis 1 to be a stepper motor
A1 SPD 1000         ; Set speed to be 1000 steps per second
A1 ACL 10000        ; Set the acceleration to be
                    ; 10000 steps per second squared
A1 DCL 10000        ; Set the deceleration to be 10000
A1 MTR ON           ; Make axis 1 active
A1 MVB 2000         ; Move motor by 2000 relative counts
```

Single Axis Servo Motor Movement

The following commands represent what is required to configure and move an individual servo motor:

```
A2 MTT SRV          ; Set axis 2 to be a servo motor
A2 GAI 32           ; Set compensator gain value to be 32
A2 ZER 240          ; Set compensator zero value to 240
A2 LIV OFF          ; Turn off loop inversion
A2 SPD 1000         ; Set speed to be 1000 steps per second
A2 ACL 10000        ; Set the acceleration to be
                    ; 10000 steps per second squared
A2 DCL 10000        ; Set the deceleration to be 10000
A2 MTR ON           ; Make axis 1 active
A2 MVB 2000         ; Move motor by 2000 relative counts
```

Two Axis Coordinated Group

The following commands associate the previous two motors, a stepper and a servo, into a 2 dimensional coordinated group.

```
C1 INI 1 2          ; Assign group 1 to be composed of motors
                    ; 1 and 2
C1 SPD 1000         ; Set vector speed to be 1000.
                    ; Note that the parameters describing the
                    ; individual axis speeds do not have any
                    ; effect on the group when directed to
                    ; move in coordination.
C1 ACL 10000        ; set vector acceleration
C1 DCL 10000        ; set vector deceleration
C1 MTR ON           ; turn on all the motors in the group
C1 MVB 1000 2000    ; move the axis together so they start at
                    ; the same time and stop at the same time
                    ; even though they are travelling
                    ; different distances.
```

Chapter 5

5) Cables and Connectors

Description

Cabling to Motion Server is performed through flat ribbon cables terminated with IDC connectors.

Axis Group Connectors

There are (4) 60 pin connectors for axis information called "Axis Group" connectors. Each 60 pin ribbon cable supports (4) axis of signals. The 60 pin ribbon cable can be split apart into (4) identical 15 pin axis sub-cables. The signals have been chosen in a very regular pattern so that all of the 15 pin sub-cables are identical in layout.

I/O Connector

There is (1) 50 pin connector containing 48 bits of configurable I/O. Signals are configured as input or output in 4 bit groups.

E-Stop Connector

There is (1) 6 pin header used to configure E-Stop with a jumper or to cable to EStop. The jumper can be used to disable E-Stop, connect I/O signal 1 to be E-Stop, or can serve as a cable connector for an external E-Stop cable assembly.

External Bus Connector

There is (1) 26 pin connector which supports an external 8 bit bus allowing Motion Server to control additional hardware elements.

Axis Signal Descriptions

Encoder A+, A-, B+, B-, I+, I-

Functional Description

Encoder signals provide position feedback from a rotary or linear encoder. In general these signals are provided in a "quadrature" format indicating both position and direction change. Encoders are necessary for servo motors and optional for stepper motors. Differential signals are desirable demonstrating improved noise immunity, however single-ended encoders may also be used. When using a single ended encoders connect the signals to the "+" inputs. The "-" inputs have a "pull-center" resistors connecting the "-" inputs to the differential receivers to a 2 volt reference. This provides a default "-" signal level in the absence of the actual signal. In certain rare cases it may be necessary to change this default reference value. This can be done by removing or switching an resistor network which is socketed on the board. Consult Douloi Automation before attempting any change.

The "I" signal is the index pulse for an optical encoder. This signal can be used for higher speed, more repeatable homing, or for encoder-drift detection.

Electrical Description

Encoder signals go into a 3486-style differential receiver. The receivers are rated for a maximum differential mode voltage of +/- 25 volts and common mode voltage of +/- 15 volts. In most cases the encoder signals are 5 volt signals.

Amp Enable High, Amp Enable Low

Functional Description

The amplifier enable signal is a digital output which allows the motor amplifier to apply power. If the amplifier is not enabled, the amplifier will not produce motor current regardless of the level of the motor command voltage. Different amplifiers have different conventions for what "enable" means. Some apply power if the signal is a high logic level. Some apply power on a low logic level. To accommodate these differences both a high and a low level signal are provided. Review amplifier documentation to learn which level is required. Douloi Automation recommends setting the amplifier (if the option is available) to be inactive until a deliberate amp enable signal is provided by the controller. Providing both a high and low level signal places the decision of amplifier enable sense into the machine wiring harness, not an on-controller jumper which could be misconfigured.

Electrical Description

The amplifier enable signals are driven by a 74LS07 with open collector outputs.

Position Capture

Functional Description

The Position Capture input can be used for high-speed registration applications. The position of the encoder is recorded in hardware in response to a position capture signal. Most often the signal is used as a homing input. Even without an encoder, the level of the signal can be monitored in software with the CaptureBit command.

Electrical Description

The Position Capture signal is the "+" side of a 3486 differential receiver. The "-" side of the receiver goes to a 2 volt reference. Standard TTL level can be used and voltage up to 24 volts maximum can be tolerated. There is no on-board pullup resistor for this input. If the sensor being used is an open-collector style drive, a 4.7k pullup resistor to +5 volts (available on the axis connector) should be used.

Position Compare

Functional Description

Position Compare is an output signal that is set when the encoder hardware detects a specific encoder position. The output can also be used as a general purpose output.

Electrical Description

Position Compare is a TTL level output with a 12 ma sink and approximately no source capability. This signal is the most "exposed" signal on the axis connector set coming directly from a FPGA device on the board with no additional buffering or protection.

Motor Command

Functional Description

The Motor Command signal is a +/- 10 volt signal most often used to specify requested current from a servo motor amplifier. The signal can also represent requested voltage or velocity depending on the amplifier mode selected. In most cases torque mode is most suitable

Electrical Description

The Motor Command signal is +/- 10 volts with 3 ma drive. Many amplifiers have differential receivers. In this case, use the motor command signal on the "+" side of the receiver and ground (from the axis cable set) on the negative side. Providing Motor Command and ground in a twisted pair can improve noise immunity.

Step Pulse, Direction

Functional Description

Step Pulse and Direction signals are used for controlling stepper motors. Standard firmware supports narrow (1 microsecond) step pulses. Alternate firmware for 4-axis controllers is available for supporting wide step pulses (30 microseconds) if the stepper driver is unable to respond to narrow pulses.

Electrical Description

Step Pulse and Direction signals are open collector outputs driven by a 74LS07.

+5 Volts, Ground

Description

+5 Volts and Ground are available for providing encoder power, sensor power, and pull-up references. These signals come directly from the PC's power supply.

Pin Numbering Conventions

There are two different connector styles most often used with the Motion Server Controller. The first is "2-row IDC" style connectors, which are the style commonly used with computer disk-drive cabling etc. In this convention, the pin number corresponds to the wire number, counting sequentially from the end of the wire. This produces a "back and forth" counting pattern on the IDC connector.

The other connector often used is a D subminiature style. This connector has a pin definition which can often be read on the connector itself. Small, inscribed numbers next to the pins indicate that the pin numbering is sequential along the length of the connector, and then resumes at the beginning of the next row. This is quite different from the "back and forth" convention of the 2 row IDC connector. It is most convenient to use D connectors by "splitting apart" the 60 pin IDC cable and then crimping IDC style D connectors. NOTE THAT THE PIN NUMBERING CONVENTION FOR D-CONNECTORS ATTACHED TO THE RIBBON CABLE IS DIFFERENT THAN THE IDC 2-ROW CONVENTION FOR THE CABLE ITSELF. Please refer to the proper table when preparing to wire to the controller.

Axis Group Connector Definitions, 2-Row IDC

The following Table defines the connectors for the axis groups. These connectors are designated "Axis 1-4", "Axis 5-8", "Axis 9-12", and "Axis 13-16" on the printed circuit board silk screen. The signal definitions is a regular pattern both along the connector, and from one connector to the next. For example, Pin 3 is always an Encoder B+ signal with the axis defined by which connector the pin is on. Each pin in any particular connector has 3 other counterparts spaced a multiple of 15 away. For example, pin 18 (pin 3 + 15) is also an Encoder B+ signal as well as pin 33 (pin 3 +30) and pin 48 (pin 3 + 45)

Pin Number	Description	Axis 1-4	Axis 5-8	Axis 9-12	Axis 13-16
1	Encoder A+	Axis 1	Axis 5	Axis 9	Axis 13
2	Encoder A-	Axis 1	Axis 5	Axis 9	Axis 13
3	Encoder B+	Axis 1	Axis 5	Axis 9	Axis 13
4	Encoder B-	Axis 1	Axis 5	Axis 9	Axis 13
5	Encoder I+	Axis 1	Axis 5	Axis 9	Axis 13
6	Encoder I-	Axis 1	Axis 5	Axis 9	Axis 13
7	Amp Enable High	Axis 1	Axis 5	Axis 9	Axis 13
8	Amp Enable Low	Axis 1	Axis 5	Axis 9	Axis 13
9	Position Capture	Axis 1	Axis 5	Axis 9	Axis 13
10	Position Compare	Axis 1	Axis 5	Axis 9	Axis 13
11	Motor Command	Axis 1	Axis 5	Axis 9	Axis 13
12	Step Pulse	Axis 1	Axis 5	Axis 9	Axis 13
13	Direction	Axis 1	Axis 5	Axis 9	Axis 13
14	+5 Volts	Axis 1	Axis 5	Axis 9	Axis 13
15	Ground	Axis 1	Axis 5	Axis 9	Axis 13
16	Encoder A+	Axis 2	Axis 6	Axis 10	Axis 14
17	Encoder A-	Axis 2	Axis 6	Axis 10	Axis 14
18	Encoder B+	Axis 2	Axis 6	Axis 10	Axis 14
19	Encoder B-	Axis 2	Axis 6	Axis 10	Axis 14
20	Encoder I+	Axis 2	Axis 6	Axis 10	Axis 14
21	Encoder I-	Axis 2	Axis 6	Axis 10	Axis 14
22	Amp Enable High	Axis 2	Axis 6	Axis 10	Axis 14
23	Amp Enable Low	Axis 2	Axis 6	Axis 10	Axis 14
24	Position Capture	Axis 2	Axis 6	Axis 10	Axis 14
25	Position Compare	Axis 2	Axis 6	Axis 10	Axis 14
26	Motor Command	Axis 2	Axis 6	Axis 10	Axis 14
27	Step Pulse	Axis 2	Axis 6	Axis 10	Axis 14
28	Direction	Axis 2	Axis 6	Axis 10	Axis 14
29	+5 Volts	Axis 2	Axis 6	Axis 10	Axis 14
30	Ground	Axis 2	Axis 6	Axis 10	Axis 14

Pin Number	Description	Axis 1-4	Axis 5-8	Axis 9-12	Axis 13-16
31	Encoder A+	Axis 3	Axis 7	Axis 11	Axis 15
32	Encoder A-	Axis 3	Axis 7	Axis 1	Axis 15
33	Encoder B+	Axis 3	Axis 7	Axis 11	Axis 15
34	Encoder B-	Axis 3	Axis 7	Axis 11	Axis 15
35	Encoder I+	Axis 3	Axis 7	Axis 11	Axis 15
36	Encoder I-	Axis 3	Axis 7	Axis 11	Axis 15
37	Amp Enable High	Axis 3	Axis 7	Axis 11	Axis 15
38	Amp Enable Low	Axis 3	Axis 7	Axis 11	Axis 15
39	Position Capture	Axis 3	Axis 7	Axis 11	Axis 15
40	Position Compare	Axis 3	Axis 7	Axis 11	Axis 15
41	Motor Command	Axis 3	Axis 7	Axis 11	Axis 15
42	Step Pulse	Axis 3	Axis 7	Axis 11	Axis 15
43	Direction	Axis 3	Axis 7	Axis 11	Axis 15
44	+5 Volts	Axis 3	Axis 7	Axis 11	Axis 15
45	Ground	Axis 3	Axis 7	Axis 11	Axis 15
46	Encoder A+	Axis 4	Axis 8	Axis 12	Axis 16
47	Encoder A-	Axis 4	Axis 8	Axis 12	Axis 16
48	Encoder B+	Axis 4	Axis 8	Axis 12	Axis 16
49	Encoder B-	Axis 4	Axis 8	Axis 12	Axis 16
50	Encoder I+	Axis 4	Axis 8	Axis 12	Axis 16
51	Encoder I-	Axis 4	Axis 8	Axis 12	Axis 16
52	Amp Enable High	Axis 4	Axis 8	Axis 12	Axis 16
53	Amp Enable Low	Axis 4	Axis 8	Axis 12	Axis 16
54	Position Capture	Axis 4	Axis 8	Axis 12	Axis 16
55	Position Compare	Axis 4	Axis 8	Axis 12	Axis 16
56	Motor Command	Axis 4	Axis 8	Axis 12	Axis 16
57	Step Pulse	Axis 4	Axis 8	Axis 12	Axis 16
58	Direction	Axis 4	Axis 8	Axis 12	Axis 16
59	+5 Volts	Axis 4	Axis 8	Axis 12	Axis 16
60	Ground	Axis 4	Axis 8	Axis 12	Axis 16

Axis Group Connector Definitions, D-Style

If the 60 pin axis cable is split into (4) 15 pin groups, it is possible to attach 15 pin IDC style connectors for a simple cable assembly. However the D connector pin numbering convention does not correspond to the wire number sequentially across. When using IDC D connectors please refer to the following table:

D Pin Number	Description
1	Encoder A+
2	Encoder B+
3	Encoder I+
4	Amp Enable High
5	Position Capture
6	Motor Command
7	Direction
8	Ground
9	Encoder A-
10	Encoder B-
11	Encoder I-
12	Amp Enable Low
13	Position Compare
14	Step Pulse
15	+5 Volts

I/O Connector Definition

The 50 pin connector provides TTL level inputs and outputs. Outputs sink 12 ma. The pin number is the I/O number with the exception of 49 (+5) and 50 (ground). Input or output sense is configured in 4 bit groups. The groups are defined by "splitting" the connector into (2) 1x50 strips, and then slicing those strips into (12) groups of (4) bits each. This partitioning was chosen so that the even-pin strip could be configured as inputs allowing a standard OPTO-22 cable to plug into the connector without contention between the cable grounds (located on all the even pins) and signals normally available on those pins.

	Description	Pin	Pin	Description	
Group 1	I/O 1	1	2	I/O 2	Group 2
Group 1	I/O 3	3	4	I/O 4	Group 2
Group 1	I/O 5	5	6	I/O 6	Group 2
Group 1	I/O 7	7	8	I/O 8	Group 2
Group 3	I/O 9	9	10	I/O 10	Group 4
Group 3	I/O 11	11	12	I/O 12	Group 4
Group 3	I/O 13	13	14	I/O 14	Group 4
Group 3	I/O 15	15	16	I/O 16	Group 4
Group 5	I/O 17	17	18	I/O 18	Group 6
Group 5	I/O 19	19	20	I/O 20	Group 6
Group 5	I/O 21	21	22	I/O 22	Group 6
Group 5	I/O 23	23	24	I/O 24	Group 6
Group 7	I/O 25	25	26	I/O 26	Group 8
Group 7	I/O 27	27	28	I/O 28	Group 8
Group 7	I/O 29	29	30	I/O 30	Group 8
Group 7	I/O 31	31	32	I/O 32	Group 8
Group 9	I/O 33	33	34	I/O 34	Group 10
Group 9	I/O 35	35	36	I/O 36	Group 10
Group 9	I/O 37	37	38	I/O 38	Group 10
Group 9	I/O 39	39	40	I/O 40	Group 10
Group 11	I/O 41	41	42	I/O 42	Group 12
Group 11	I/O 43	42	44	I/O 44	Group 12
Group 11	I/O 45	45	46	I/O 46	Group 12
Group 11	I/O 47	47	48	I/O 48	Group 12
	+5 Volts	49	50	Ground	

EStop Connector Definition

The EStop connector has 6 pins defined as follows

pin 1	Not Connected (pin 1 is closest to the mounting bracket, rear of PC)
pin 2	Ground
pin 3	E-Stop input
pin 4	I/O 1 from 50 pin connector
pin 5	12 Volt Input for Unlocking Flash Memory
pin 6	12 Volt Source from PC

Placing a jumper between pins 2 and 3 enables the E-Stop (which must be maintained at ground against its 4.7k pullup). This is not recommended if doing anything besides bench testing free spinning motors.

Placing the jumper between pins 3 and 4 redirects the EStop to be from the general I/O connector where an OPTO-22 module rack may be hooked in, or some other IO interconnect that has been chosen for general purpose I/O

A third option is to put a 6 x 1 plug into this header with a cable for pins 2 and 3. A normally closed switch would serve as an E-Stop switch. If the switch disconnected, or the cable was missing, the controller will not enable power to the amplifiers.

The on-board Flash memory chip is used to store application programs. If the board contains a 28F class Flash Memory chip, a 12 volt level must be supplied to the chip to "unlock" the chip and permit alteration of its contents. This level can be provided by turning on switch number 4 on the board itself. In some applications, accessing switch 4 may be inconvenient. In this case, an external switch can be provided that connects pin 5 and pin 6 allowing the memory device to be programmed. Alternately, consult with Douloi Automation regarding a newer 29F class memory chip which does not require the 12 volt level.

External Bus Connector

The remaining 26 pin connector provides a simplified 8-bit bus that can be used to connect to additional hardware. Note that Douloi provides a PC/104 "bridge" accessory that is driven by this connector. The PC/104 format allows the use of many third part cards

Power signals from this connector should only be for signal-level power. If you need any significant current, use a disk-drive connector. Additional details about the use of this bus are available from Douloi Automation on request.

Pin	Description
1	Data 0
2	Data 1
3	Data 2
4	Data 3
5	Data 4
6	Data 5
7	Data 6
8	Data 7
9	Addr 0
10	Addr 1
11	Addr 2
12	Addr 3
13	Addr 4
14	Addr 5
15	Addr 6
16	Select
17	Write/Read
18	Comm_Capture_1
19	Comm_Capture_2
20	Comm_Capture_3
21	Comm_Capture_4
22	Reset
23	+12 Volts
24	-12 Volts
25	+5 Volts
26	Ground