

Motion

Application Specific Function Block Manual

Version 15.0.1

NOTE

Progress is an on-going commitment at G & L Motion Control Inc. We continually strive to offer the most advanced products in the industry; therefore, information in this document is subject to change without notice. The illustrations and specifications are not binding in detail. G & L Motion Control Inc. shall not be liable for any technical or editorial omissions occurring in this document, nor for any consequential or incidental damages resulting from the use of this document.

DO NOT ATTEMPT to use any G & L Motion Control Inc. product until the use of such product is completely understood. It is the responsibility of the user to make certain proper operation practices are understood. G & L Motion Control Inc. products should be used only by qualified personnel and for the express purpose for which said products were designed.

Should information not covered in this document be required, contact the Customer Service Department, G & L Motion Control Inc., 672 South Military Road, P.O. Box 1960, Fond du Lac, WI 54936-1960. Telephone us at (800) 558-4808 (within the United States only) or (920) 921-7100 or by e-mail at glmotion.support@danahermotion.com.

DISCLAIMER: All programs in this release (application demos, application specific function blocks (ASFB's), etc.), are provided "AS IS, WHERE IS", WITHOUT ANY WARRANTIES, EXPRESS OR IMPLIED. There may be technical or editorial omissions in the programs and their specifications. These programs are provided solely for user application development and user assumes all responsibility for their use. Programs and their content are subject to change without notice.

Release 2705

Catalog No. (Order No.) M.1300.7573

Version Part No. M.1302.8312

© 1993-2005 G & L Motion Control Inc.

IBM is a registered trademark of International Business Machines Corporation.

Windows 95, 98, NT, Microsoft, and MS-DOS are registered trademarks of Microsoft Corporation.

Pentium and PentiumPro are trademarks of Intel Corporation.

ARCNET is a registered trademark of Datapoint.

PiC900, PiCPro, MMC, PiCServoPro, PiCTune, PiCProfile, LDO Merge, PiCMicroTerm and PiC Programming Pendant are trademarks of G & L Motion Control Inc.

Table of Contents: Motion ASFB Manual

CHAPTER 1-Application Specific Function Block Guidelines	1-1
Installation.....	1-1
Revisions	1-1
Network 1	1-2
Network 2	1-2
Network 3	1-3
Network 4	1-3
Using ASFBs.....	1-3
CHAPTER 2-Motion ASFBs	2-1
ADDCKSUM.....	2-6
BYTE2HEX.....	2-6
CHKCKSUM.....	2-7
DWOR2HEX.....	2-7
HEX2BYTE.....	2-8
HEX2DWOR.....	2-9
HEX2WORD.....	2-9
M_C2M_1.....	2-10
M_CHK1.....	2-37
M_CHK101.....	2-38
M_CHK109.....	2-39
M_CHK49.....	2-40
M_CHK57.....	2-41
M_CHK65.....	2-42
M_CHK73.....	2-43
M_CHK9.....	2-44
M_CLOS1.....	2-45
M_CLOS9.....	2-47
M_CLS101.....	2-49
M_CLS109.....	2-51
M_CRSFIN.....	2-53
M_DATCAP.....	2-55
M_DATCPT.....	2-59
M_DNJOGC.....	2-63
M_DNPOSC.....	2-65
M_DNSTAT.....	2-67
M_DSMCOM.....	2-69
RS232 Connections.....	2-73
RS422/RS485 Connections.....	2-73
M_DW2BOO.....	2-74
M_ERROR.....	2-76
M_FHOME.....	2-77
M_INCPTR.....	2-79

M_JOG.....	2-81
M_LHOME.....	2-82
M_LINCIR.....	2-85
M_PRTCAM.....	2-89
M_PRTREL.....	2-91
M_PRTSLP.....	2-93
M_RATREL.....	2-95
M_RATSLP.....	2-96
M_RDTUNE.....	2-98
M_RGSTAT.....	2-99
M_RSET49.....	2-101
M_RSET57.....	2-102
M_RSET65.....	2-103
M_RSET73.....	2-104
M_SACC.....	2-105
M_SCRVLC.....	2-107
M_SRCMON.....	2-113
M_SRCPRC.....	2-115
M_SRCRDL.....	2-117
M_SRCWT.....	2-119
M_SRCWTL.....	2-121
ERR Output.....	2-122
SERR Output.....	2-125
BSER Output.....	2-126
M_STATUS.....	2-127
M_WTTUNE.....	2-129
M_XL2CM.....	2-131
S_CLOS1.....	2-135
S_CLOS9.....	2-137
S_CLS101.....	2-140
S_CLS109.....	2-143
S_ERRORC.....	2-146
S_FHOME.....	2-149
S_IO_C.....	2-152
S_LHOME.....	2-154
WORD2HEX.....	2-157
SD_STAT.....	2-158
SD_IO.....	2-166

APPENDIX A-M_DSMCOM Commands..... A-1

Exception Responses.....	A-1
Host Command Set.....	A-2
Common Product Line Commands.....	A-2
General Commands.....	A-5
Position Loop Commands.....	A-7
Velocity Loop Commands.....	A-10

Torque Current Conditioning Commands	A-12
Motor Commands	A-14
Motor Commands (Continued).....	A-16
Motor Commands (Continued).....	A-18
Digital I/O Commands	A-19
Analog I/O Commands	A-21
Analog I/O Commands (Continued).....	A-25
Serial Port Commands	A-26
Operating Mode Commands	A-27
Operating Mode Commands (Continued).....	A-30
Alternative Operating Mode Commands	A-31
Alternative Operating Mode Commands (Continued).....	A-34
Runtime Command and Control Commands	A-36
Runtime Status Commands	A-38
Runtime Status Commands (Continued)	A-41
Runtime Data Commands	A-43
Runtime Data Commands (Continued).....	A-46
Runtime Data Collection Commands	A-48
Runtime Data Collection Commands (Continued).....	A-51
APPENDIX B-Press Transfer ASFBs	B-1
Introduction.....	B-1
M_PR2MV	B-6
M_PR1MV	B-17
M_PRFERR.....	B-18
M_PROFL	B-20
M_PRFDWL.....	B-22
M_SETVAJ	B-23
M_SC_ACC.....	B-25
M_CNST_V.....	B-26
M_SC_DEC	B-27
APPENDIX C-Contents of the Applications CD	C-1
APPENDIX D-Standalone MMC Examples.....	D-1
Descriptions of the Standalone MMC Examples	D-1
Basic Application Examples D-1	
Operator Interface Application Examples D-2	
SERCOS Drive Interface Application Example D-3	
Press Transfer Application Example D-3	
Stepper Application Example D-4	
Centurion DeviceNet Positioning MicroDSM Drive Application Example D-4	
Starting a New MMC Application from a Standalone MMC Example	D-4
APPENDIX E-Servo Setup Assistant	E-1
Index	Index-1

CHAPTER 1 **Application Specific Function Block Guidelines**

Installation

The following guidelines are recommended ways of working with Application Specific Function Blocks (i.e. ASFBs) from G & L Motion Control Inc.

The Applications CD includes the ASFB package as follows:

- .LIB file(s) containing the ASFB(s)
- source .LDO(s) from which the ASFB(s) was made
- example LDO(s) with the ASFB(s) incorporated into the ladder which you can then use to begin programming from or merge with an existing application ladder

When you install the Applications CD, the ASFB paths default to:

C:\Program Files\G&L Motion Control\Applications vxx.x.r\ASFB

and

C:\Program Files\G&L Motion Control\Applications vxx.x.r\Examples

where vxx.x is the PiCPro version number that these ASFBs and examples were built under. The .r is the revision number of the Application software itself.

The .LIB files and source .LDO files are put in the ASFB subdirectory. The example .LDO files are put in the Examples subdirectory.

Revisions

The first four networks of each ASFB source ladder provide the following information:

Network 1

The first network just informs you that the ASFB is provided to assist your application development.

Network 2

The second network is used to keep a revision history of the ASFB. Revisions can be made by G&L Motion Control personnel or by you.

The network identifies the ASFB, lists the requirements for using this ASFB, the name of the library the ASFB is stored in, and the revision history.

The revision history includes the date, ASFB version (see below), the version of PiCPro used while making the ASFB, and comments about what the revision involved.

When an ASFB is revised, the number of the first input (EN__ or RQ__) to the function block is changed in the software declarations table. The range of numbers available for G&L Motion Control personnel is 00 to 49. The range of numbers available for you is 50 to 99. See chart below.

Revision	G&L Motion Control revisions	User revisions
1st	EN00	EN50
2nd	EN01	EN51
.	.	.
.	.	.
.	.	.
50th	EN49	EN99

Network 3

The third network describes what you should do if you want to make a revision to the ASFB.

Network 4

The fourth network describes the ASFB and defines all the inputs and outputs to the function block.

Using ASFBs

When you are ready to use the ASFB in your application, there are several approaches you can take as shown below.

- Create a new application LDO starting with the example LDO for the ASFB package. The advantage is that the software declarations table for the ASFB has been entered for you.
- If you already have an application LDO, copy and paste the example LDO into yours. The software declaration tables for both LDOs will also merge.

NOTES

CHAPTER 2 **Motion ASFBs**

The motion support function blocks are contained in the libraries as shown. They are used to aid in the application of servo and digitizing axes. Included with these library files are other example LDO files as listed. The motion support function blocks are described in alphabetical order.

The SERCOS motion function blocks are also shown. They are used to aid in the application of SERCOS servo and digitizing axes. Their names start with S_. They are written to replace the corresponding M_ motion function block, when the axes use SERCOS control rather than analog control.

Library	Function Block	Description
M_C2M_1		
	M_C2M_1	Translates a M and G code format ASCII file into servo motion.
M_COMMON		
	BYTE2HEX	Places the data type byte into hexadecimal notation.
	DWOR2HEX	Places the data type double word into hexadecimal notation.
	HEX2BYTE	Places the hexadecimal notation into a byte.
	HEX2DWOR	Places the hexadecimal notation into a double word.
	HEX2WORD	Places the hexadecimal notation into a word.
	M_DW2BOO	Places the data type double word into 32 booleans
	WORD2HEX	Places the data type from word into hexadecimal notation.
M_DATA		
	M_DATCAP	Captures axis information on an interrupt basis.
	M_DATCPT	Captures axis information on an interrupt basis to printable text file
	M_ERROR	Returns E-stop, C-stop, and programming errors for a servo axis or E-stop errors for a digitizing axis.
	M_INCPTR	Increment buffer pointers for M_DATCPT (not used in your LDO).
	M_PRTCAM	Creates a text file for the CAM input of RATIOCAM.
	M_PRTREL	Creates a text file for the REAL input of RATIO_RL.
	M_PRTSLP	Creates a text file for the SLOPE input of RATIOSLP.
	M_RATREL	Calculates ending ratio and slope for use in ratio real profile.
	M_RATSLP	Calculates ending ratio and slope for use in ratio slope profile.
	M_RDTUNE	Reads tuning parameters for a closed loop axis.
	M_RGSTAT	Returns registration information for a closed loop or digitizing axis.

M_STATUS	Returns status information (for example, position and following error) for a closed loop, time, or digitizing axis.
M_WTTUNE	Changes tuning parameters on a closed loop axis

M_DEVNET

M_DNJOGC	Jogs a Centurion DeviceNet drive axis
M_DNPOSC	Moves a Centurion DeviceNet drive to a position (either absolute or incremental)
M_DNSTAT	Obtains the DeviceNet module status

M_DRVCOM

ADDCKSUM	Support routine for M_DSMCOM. (Not used in your LDO.)
CHKCKSUM	Support routine for M_DSMCOM. (Not used in your LDO.)
M_DSCOM	Allows interfacing between the PiC and one or more Centurion DS100/200 servo drives.

M_INIT

M_CHK1	Checks to see which servo axes (1 to 8) have been initialized.
M_CHK101	Checks to see which servo axes 101 to 108 (17 to 24) have been initialized.
M_CHK109	Checks to see which servo axes 109 to 116 (25 to 32) have been initialized.
M_CHK49	Checks to see which digitizing axes (49 to 56) have been initialized.
M_CHK57	Checks to see which digitizing axes (57 to 64) have been initialized.
M_CHK65	Checks to see which digitizing axes (65 to 72) have been initialized.
M_CHK73	Checks to see which digitizing axes (73 to 80) have been initialized.
M_CHK9	Checks to see which servo axes (9 to 16) have been initialized.
M_CLOS1	Closes the loop on servo axes 1 to 8.
M_CLOS9	Closes the loop on servo axes 9 to 16.
M_CLS101	Closes the loop on servo axes 101 to 108 (17 to 24).
M_CLS109	Closes the loop on servo axes 109 to 116 (25 to 32).
M_RSET49	Resets E-stop errors on digitizing axes 49 to 56.
M_RSET57	Resets E-stop errors on digitizing axes 57 to 64.
M_RSET65	Resets E-stop errors on digitizing axes 65 to 72.
M_RSET73	Resets E-stop errors on digitizing axes 73 to 80.

M_MOVE

M_JOG	Jogs a closed loop axis.
M_LINCIR	Performs linear, circular, and simultaneous endpoint arrival moves on closed loop axes.
M_SACC	Calculates the ACC and JERK values to be used with the ACC_JERK function.
M_SCRVLC	Provides the interface from the application .LDO to the RATIO_RL function in order to perform linear coordinated, circular, or third axis departure (simultaneous endpoint arrival) moves with S-curve acceleration and deceleration.

M_PROFL

M_CNST_V	Constant velocity segment.
M_PRF1MV	One slave move for master.
M_PRF2MV	Two slave moves for master
M_PRFDWL	Slave dwell in profile.
M_PRFERR	Check for profile errors.
M_PROFL	Make profile for 1 move
M_SC_ACC	Acceleration segment.
M_SC_DEC	Deceleration segment
M_SETVAJ	Set velocity, acceleration, and jerk values.

M_REF

M_CRSFIN	Implements coarse, medium and fine resolvers.
M_FHOME	Performs a home cycle on a closed loop axis using the fast input as the reference switch.
M_LHOME	Performs a home cycle on a closed loop axis using a discrete input as the reference switch.

M_SERCOS

M_SRCMON	Monitors up to five SERCOS IDNs.
M_SRCPRC	Executes a SERCOS procedure command function.
M_SRCRDL	Reads a list of SERCOS IDNs.
M_SRCWT	Writes and reads up to five SERCOS IDNs.
M_SRCWTL	Writes a list of SERCOS IDNs.

M_XL2CM

M_XL2CM	Converts an ASCII CSV file in the correct format to a data structure that can be called directly by RATIOCAM, RATIOSLP or RATIO_RL.
---------	---

S_ASFB

S_CLOS1	Closes the loop on SERCOS servo axes 1 to 8 (to replace M_CLOS1)
S_CLOS9	Closes the loop on SERCOS servo axes 9 to 16 (to replace M_CLOS9)
S_ERRORC	Returns e-stop, c-stop, and programming errors for a SERCOS servo axis or e-stop errors for a SERCOS digitizing axis; SERCOS ring and slave errors are also returned (to replace M_ERROR for SERCOS axis).
S_FHOME	Performs a home cycle on a SERCOS servo axis using the fast input as the reference switch (to replace M_FHOME for SERCOS axis)
S_IO_C	Allows control of the discrete I/O for a SERCOS servo axis with a Centurion drive.
S_LHOME	Performs a home cycle on a SERCOS servo axis using a discrete input as the reference switch (to replace M_LHOME for SERCOS axis)

Example LDOs

The following example LDOs are included:

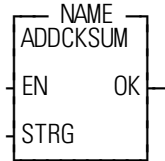
- M_CAMREL An example .LDO that uses the M_RATREL function block to convert a RATIOCAM profile to a RATIO_RL profile.
- M_CAMSLP An example .LDO that uses the M_RATSLP function block to convert a RATIOCAM profile to a RATIO-SLP profile. The M_PRTSLP function block is then used to print the RATIOSLP profile.
- M_CAPTUR An example .LDO that shows how to use the M_DATCAP function block.
- M_COORD An example .LDO that uses the M_LINCIR function block to perform linear and circular coordinated moves on a pair of axes.
- M_DSM_EX An example .LDO that uses the M_DSMCOM function block to communicate with Centurion drives through a serial communications board in rack 0, slot 10, channel 2.

- M_EXAMPL An example .LDO that shows how to use the M_CHK1, M_CHK49, M_CLOS1, M_CRSFIN, M_ERROR, M_FHOME, M_JOG, M_LHOME, M_RGSTAT, M_RSET49, and M_STATUS function blocks.
- M_PRF_EX An example .LDO that shows how to use the M_PRF2MV function block to configure a slave profile for a RATIO_RL move.
- M_TUNE An example .LDO that shows how to use the M_RDTUNE and M_WTTUNE function blocks.
- MMC_DND An example .LDO that controls a Centurion DeviceNet drive axis. The axis is homed, jogged or moved to a position (either an absolute position or a relative distance).

ADDCKSUM

Add checksum to string

USER/M_DRVCOM



Inputs: EN (BOOL) - enables execution
STRG (STRING) - input string
Outputs: OK (BOOL) - execution complete

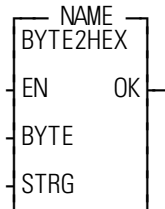
ADDCKSUM(EN := <<BOOL>>, OK => <<BOOL>>);

This function block appends the one-byte checksum to the end of an input string. This is a support routine and is not used in your LDO.

BYTE2HEX

Converts a byte to a hex value

USER/M_COMMON



Inputs: EN (BOOL) - enables execution
BYTE (BYTE) - value to convert
STRG (STRING) - converted value
Outputs: OK (BOOL) - execution complete

<<INSTANCE NAME>>:BYTE2HEX(EN := <<BOOL>>, BYTE := <<BYTE>>, STRG := <<STRING>>, OK => <<BOOL>>);

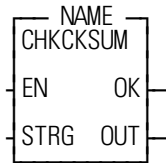
This function block places the hexadecimal notation of the value at BYTE into the string at STRG.

Example: If 27 is entered at the BYTE input, 1B will be reported at STRG.

CHKCKSUM

Check checksum in string

USER/M_DRVCOM



Inputs: EN (BOOL) - enables execution
STRG (STRING) - input string to check

Outputs: OK (BOOL) - execution complete
OUT (BOOL) - checksum OK output

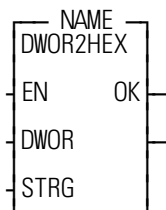
```
CHKCKSUM(EN := <<BOOL>>, STRG := <<STRING>>, OK => <<BOOL>>,
          OUT => <<BOOL>>);
```

This function block checks the checksum in an input string. This is a support routine and is not used in your LDO.

DWOR2HEX

Converts a double word to a hex value

USER/M_COMMON



Inputs: EN (BOOL) - enables execution
DWOR (DWOR) - value to convert
STRG (STRING) - converted value

Outputs: OK (BOOL) - execution complete

```
<<INSTANCE NAME>>:DWOR2HEX(EN := <<BOOL>>, DWOR :=
                             <<DWOR>>, STRG := <<STRING>>, OK => <<BOOL>>);
```

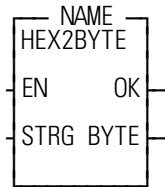
This function block places the hexadecimal notation of the value at DWOR into the string at STRG.

Example: If 845,621 is entered at the DWOR input, CE735 will be reported at STRG.

HEX2BYTE

Converts a hex value to a byte

USER/M_COMMON



Inputs: EN (BOOL) - enables execution
STRG (STRING) - hexadecimal value to convert

Outputs: OK (BOOL) - execution complete
BYTE (BYTE) - converted value

```
<<INSTANCE NAME>>:HEX2BYTE(EN := <<BOOL>>, STRG :=  
  <<STRING>>, OK => <<BOOL>>, BYTE => <<BYTE>>);
```

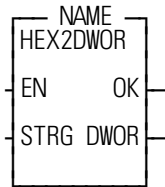
This function block places the hexadecimal notation of the string at STRG into the output at BYTE.

Example: If 1B is entered at the STRG input, 27 will be reported at the BYTE output.

HEX2DWOR

Converts a hex value to a double word

USER/M_COMMON



Inputs: EN (BOOL) - enables execution
STRG (STRING) - hexadecimal value to convert

Outputs: OK (BOOL) - execution complete
DWOR (DWORD) - converted value

```
<<INSTANCE NAME>>:HEX2DWOR(EN := <<BOOL>>, STRG :=  
  <<STRING>>, OK => <<BOOL>>, DWOR => <<DWORD>);
```

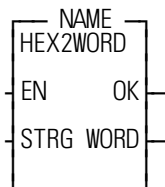
This function block places the hexadecimal notation at STRG into the output at DWOR.

Example: If CE735 is entered at the STRG input, 845,621 will be reported at the DWOR output.

HEX2WORD

Converts a hex value to a word

USER/M_COMMON



Inputs: EN (BOOL) - enables execution
STRG (STRING) - hexadecimal value to convert

Outputs: OK (BOOL) - execution complete
WORD (WORD) - converted value

```
<<INSTANCE NAME>>:HEX2WORD(EN := <<BOOL>>, STRG :=  
  <<STRING>>, OK => <<BOOL>>, WORD => <<WORD>);
```

This function block places the hexadecimal notation at STRG into the output at WORD.

Example: If 26,854 is entered at the STRG input, 68E6 will be reported at the WORD output.

M_C2M_1

Translates M and G ASCII file to servo motion

USER/M_C2M_1

NAME		
M_C2M_1		
EN01	DONE	Inputs: E: EN01 (BOOL) - enables execution
FNAM	FAIL	FNAM (STRING[32]) - filename string (name of program to be executed).
STRT	ACTV	STRT (BOOL) - pulsed to start program execution
CONT	MDID	CONT (BOOL) - energized to repeat program execution continuously
SNGL	FERR	SNGL (BOOL) - energized to enter single step mode
EXEC	PERR	EXEC (BOOL) - pulsed to execute next instruction in single step mode
MDIM	MERR	MDIM (BOOL) - energize to select Manual Data Input mode
MDIS	LINE	MDIS (STRING[130]) - manual data input string
ABRT	LACT	ABRT (BOOL) - pulse to abort program execution
SETP		SETP (STRUCT) - defines the operation of this application of M_C2M_1
I		I (STRUCT) - user Input structure
O		O (STRUCT) - user Output structure
VLIN		VLIN (STRING[64]) - string that shows the current program line being executed.
DATA		DATA- (STRUCT) - modal data structure
OVRD		OVRD (USINT) - Path feedrate override

Outputs: DONE (BOOL) - initialization completed without error or was aborted by an ABRT request
FAIL (BOOL) - indicates that an error occurred while trying to execute the program
ACTV (BOOL) - active indicates that program execution is in progress
MDID(BOOL) -Manual Data Input mode done
FERR (INT) - indicates file read error
PERR (INT) - indicates program error
MERR (INT) - indicates motion error
LINE (DINT) – line number
LACT (BOOL) – line active

```

<<INSTANCE NAME>>:M_C2M_1(EN01 := <<BOOL>>, FNAM :=
  <<STRING[32]>>, STRT := <<BOOL>>, CONT := <<BOOL>>, SNGL :=
  <<BOOL>>, EXEC := <<BOOL>>, MDIM := <<BOOL>>, MDIS :=
  <<STRING[130]>>, ABRT := <<BOOL>>, SETP := <<STRUCT>>,
  I := <<STRUCT>>, O := <<STRUCT>>, VLIN := <<STRING[64]>>, DATA
  := <<STRUCT>>, OVRD := <<USINT>>, DONE => <<BOOL>>, FAIL =>
  <<BOOL>>, ACTV => <<BOOL>>, MDID => <<BOOL>>, FERR =>
  <<INT>>, PERR => <<INT>>, MERR => <<INT>>, LINE => <<DINT>>,
  LACT => <<BOOL>>);

```

M_C2M_1, the Cad2Motion ASFB translates an M and G code format ASCII file into servo motion. Many applications require description of their motion path using CAD software. Third party packages (such as Gcode2000) will convert the CAD package DXFoutput to M and G code text files. M_C2M_1 will translate the M and G code file to servomotion. Applications include glue laying and textile cutting, to name a few. This ASFB is not intended for application to metal cutting machine tools such as lathes and mills and as such does not support features required by CNC applications such as cutter radius compensation, tool offsets and constant surface speed.

M_C2M_1 G Code Instruction Summary

Code	Type	Use	Description
%		Start of Program	All characters before the % are ignored
/		Start/End Comment	After first / is encountered all code is ignored until another / encountered or the end of the program
N		Line Number	For reference only, ignored
X		X Command	X axis endpoint (G90) or incremental distance (G91)
Y		Y Command	Y axis endpoint (G90) or incremental distance (G91)
Z		Z Command	Z axis endpoint (G90) or incremental distance (G91)
W		W Command	W axis endpoint (G90) or incremental distance (G91)
A		A Command	A axis endpoint (G90) or incremental distance (G91)
B		B Command	B axis endpoint (G90) or incremental distance (G91)
I		X Center Point	Incremental distance from starting position to X Center point
J		Y Center Point	Incremental distance from starting position to Y Center point
K		Z Center Point	Incremental distance from starting position to Z Center point
F	Modal	Rate	Path velocity for G01, G02, G03, time for G04. Reset to 0 by abort or M30 end of program when not continuous
L	Modal	Rate	Value for use by application program to specify use logic sequence to execute
S	Modal	Rate	Rate value for use by application program, reset to 0 by abort or end of program when not continuous

G00		Rapid	Move all axes at default Rapid rate to position/increment, G00 is default
G01	Modal Group	Linear	Move all axes linearly at programmed rate to position/increment
G02		Circular Clockwise	Move all axes linearly at programmed rate to position/increment
G03		Circular Counter-clockwise	Move all axes linearly at programmed rate to position/increment
G04		Dwell	Wait length of time specified by F
G09		Decel-to-zero	Wait until all axes are in position with no move queued
G17	Modal Group	XY Plane	XY Plane Select, G17 is default, Z, W, A, B departure
G18		XZ Plane	XZ Plane Select, Y, W, A, B departure
G19		YZ Plane	YZ Plane Select, X, W, A, B departure
G90	Modal Group	Absolute	Select Absolute positioning mode, G90 is default
G91		Incremental	Select Incremental positioning mode
G92		Assign Dimension	Assign dimension specified to X, Y, Z, W, and B data to the respective axis current position

M_C2M_1 M Code Instruction Summary

Code	Type	Use	Description
M03	Modal	Start Cut	Set M03 indicator for use by user application
M05		Stop Cut	Reset M03 indicator for use by user application
M30		End Program	Stop executing program, reset all modal data flags to default
M10x	Modal	Turn On Output	Turn on user output 1 to 9, all outputs cleared by abort or end of program when not continuous
M20x		Turn Off Output	Turn off user output 1 to 9
M50x		Wait for Input On	Wait until specified input (x = 1 to 9) is on M10x and M20x instructions programmed in the same line as motion (G00..G03) will cause the specified output to turn on/off when the queued move becomes active
M60x		Wait for Input Off	Wait until specified input (x = 1 to 9) is off M50x and M60x instructions programmed in the same line as motion (G00..G03) will cause the queuing of the specified move to be delayed until the wait for on/off states are satisfied
M70x		Wait for Input On at end of instruction	At end of instruction, wait until specified input (x = 1 to 9) is on. After all other operations programmed in the instruction have been executed begin waiting until the specified input is On.

M_C2M_1 Instruction Descriptions

Code	Type	Use	Description
%		Start of Program	All characters before the % are ignored. All information in all program lines is ignored until a “%”, Start of Program, is encountered. This allows a detailed description of the program at its beginning. All subsequent “%” characters are ignored and have no effect.

Optionally, the name of the application can be specified in SETP.Application_Name. When this string is not null, the comment field immediately after the % must be the same string, e.g. %.Saw_Type_1/. If SETP.Application_Name is a null then this check is not performed. If it is specified and the name does not match, PERR is set to 7016 and FAIL is set.

Code	Type	Use	Description
/		Start/End Comment	After first / is encountered all code is ignored until another / encountered or the end of the program

All information after the first “/” slash character is encountered is ignored until a second “/” character or the end of the program file is found. The second “/” may be programmed in the same program line or later in the program. A single line can contain multiple sections which surrounded by slashes and ignored. In the line “N1000 /M101/ M201 G90 / M501/” the M101 and M501 commands would both be ignored.

Code	Type	Use	Description
N		Line Number	For reference only, ignored

The N data word is used by the programmer as a reference point to determine which line is being processed. It is not used in any way and is ignored.

Code	Type	Use	Description
X		X Command	X axis endpoint (G90) or incremental distance (G91)
Y		Y Command	Y axis endpoint (G90) or incremental distance (G91)
Z		Z Command	Z axis endpoint (G90) or incremental distance (G91)
W		W Command	W axis endpoint (G90) or incremental distance (G91)
A		A Command	A axis endpoint (G90) or incremental distance (G91)
B		B Command	B axis endpoint (G90) or incremental distance (G91)

Including X,Y, Z, W, A, or B endpoints/distances indicates that the specified axes should be moved according to the modal move mode (G00 to G03) and modal plane select (G17 to G19). Examples of valid X,Y,Z data format include: “X1”, “X0”, “X-1.23”, “X.001”.

For more detail see the Instruction Execution section below.

Code	Type	Use	Description
I		X Center point	Incremental distance from starting position to X Center point
J		Y Center point	Incremental distance from starting position to Y Center point
K		Z Center point	Incremental distance from starting position to Z Center point

When circular clockwise (G02) or circular counterclockwise (G03) modal move type is active the circle centerpoints must be specified by I and J when the modal XY plane (G17) is active, or J and K when the modal XZ plane (G18) is active, or I and K when the modal XZ plane (G19) is selected. Circle centerpoints must always be programmed as incremental distances from the circle starting position to the circle centerpoint, independent of whether the G90 absolute or G91 incremental positioning mode is active.

Code	Type	Use	Description
F	Modal	Rate	Path velocity for G01, G02, G03, time for G04. Reset to 0 by abort or M30 end of program when not continuous

The F data word specifies the path feedrate for G01 linear and G02/G03 circular moves. It is modal and does not need to be specified again until a new value is required. It will be reset to zero if a fault or abort occurs or if an M30 end of program occurs and continuous program repeat mode is not selected.

The F data word specifies the delay in seconds when specified with a G04 dwell instruction. To select a one-half second delay “G04 F0.5” would be programmed.

Code	Type	Use	Description
S	Modal	Rate	Rate value for use by application program, reset to 0 by abort or end of program

The value specified by S is presented to the user application by the output SDAT of the M_C2M_1 ASFB. Typically it is used for rate control. It is modal and its value will not change until another occurrence of S. It will be cleared to 0 if an error occurs, after M30 end of program when not in continuous mode or when program execution is aborted.

Code	Type	Use	Description
G00	Modal Group	Rapid	Move all axes at default Rapid rate to position/increment, G00 is default
G01		Linear	Move all axes linearly at programmed rate to position/increment
G02		Circular Clockwise	Move all axes linearly at programmed rate to position/increment
G03		Circular Counterclockwise	Move all axes linearly at programmed rate to position/increment

G00, G01, G02 and G03 are the modal path positioning group. G00 is the default mode. Once selected the position type does not need to be specified again until you wish to change it. It will be reset to G00 if an error occurs, after M30 end of program when not in continuous mode or when program execution is aborted.

Code	Type	Use	Description
G04		Dwell	Wait length of time specified by F

The F data word specifies the delay in seconds when specified with a G04 dwell instruction. To select a one-half second delay “G04 F0.5” would be programmed.

The time delay begins immediately unless a Wait for Input (M50x/M60x) is programmed in the same line, in which case the delay begins after the Wait condition is satisfied.

Code	Type	Use	Description
G09		Decel-to-zero	Wait until all axes are in position with no move queued

Stalls execution until all axes are within the in position bandwidth specified by servo setup and no moves are queued. If programmed in a line with motion the G09 begins after the motion has been queued.

Code	Type	Use	Description
G17	Modal Group	XY Plane	XY Plane Select, G17 is default, Z, W, A, B departure
G18		XZ Plane	XZ Plane Select, Y, W, A, B departure
G19		YZ Plane	YZ Plane Select, X, W, A, B departure

G17, G18 and G19 are the modal plane select group. G17 is the default mode. Once selected the plane does not need to be selected again until you wish to change it. It will be reset to G17 if an error occurs, after M30 end of program when not in continuous mode or when program execution is aborted. The plane specified is the plane which axes can be moved using G02 circular clockwise and G03 circular counterclockwise moves. If endpoints are specified for the departure axes and G02 or G03 is performed, the departure axis will arrive at their programmed endpoints at the same time as the circular interpolation axes.

Code	Type	Use	Description
G90	Modal Group	Absolute	Select Absolute positioning mode, G90 is default
G91		Incremental	Select Incremental positioning mode

G90 and G91 select absolute or incremental position mode. G90 is the default mode. Once selected, the positioning mode does not need to be specified again until you wish to change it. It will be reset to G90 if an error occurs, after M30 end of program when not in continuous mode or when program execution is aborted.

Code	Type	Use	Description
G92		Assign Dimension	Assign dimension specified to X, Y, Z, W, A and B data to the respective axis current position

G92 will wait for all axes to decel-to-zero and be in-position and then will assign the value programmed in the X, Y, Z, W, A and B data words to be the current position for each respective axes.

Code	Type	Use	Description
M03		Start Cut	Set M03 indicator for use by user application
M04	Modal	Spindle CCW	Set Data.M04, reset Data.M03 for use by user application
M05		Stop Cut	Reset M03 indicator for use by user application

M03, M04 and M05 are the modal start/stop indicators. M05 is the default mode. Once selected the start/stop mode does not need to be specified again until you wish to change it. It will be reset to M05 if an error occurs, after M30 end of program when not in continuous mode or when program execution is aborted. Programming an M03/M04 in a line without motion will cause the M03 output of the M_C2M_1 ASFB to energize immediately. Programming an M03/M04 in a line with motion will cause the M03/M04 output of the M_C2M_1 ASFB to energize when the move becomes active. Programming an M05 in a line without motion will cause the M03/M04 indicator to de-energize immediately. Programming an M05 in a line with motion will cause the M03/M04 indicator to de-energize when the move becomes active.

Code	Type	Use	Description
M30		End Program	Stop executing program, reset all modal data flags to default

M30 indicates the end of the program. When a line contains M30 all instructions in the line will execute, the equivalent of a G09 decel to zero will execute and then the file will be closed. If the CONT, continuous mode input to M_C2M_1 is energized the program file will be opened and executed again. If CONT is not energized then all modal data will be reset, all User Outputs will be de-energized (O.O1 to O.O9), the program file will be closed and the DONE output of M_C2M_1 will be energized indicating the program has finished executing. All lines following a line with M30 are ignored.

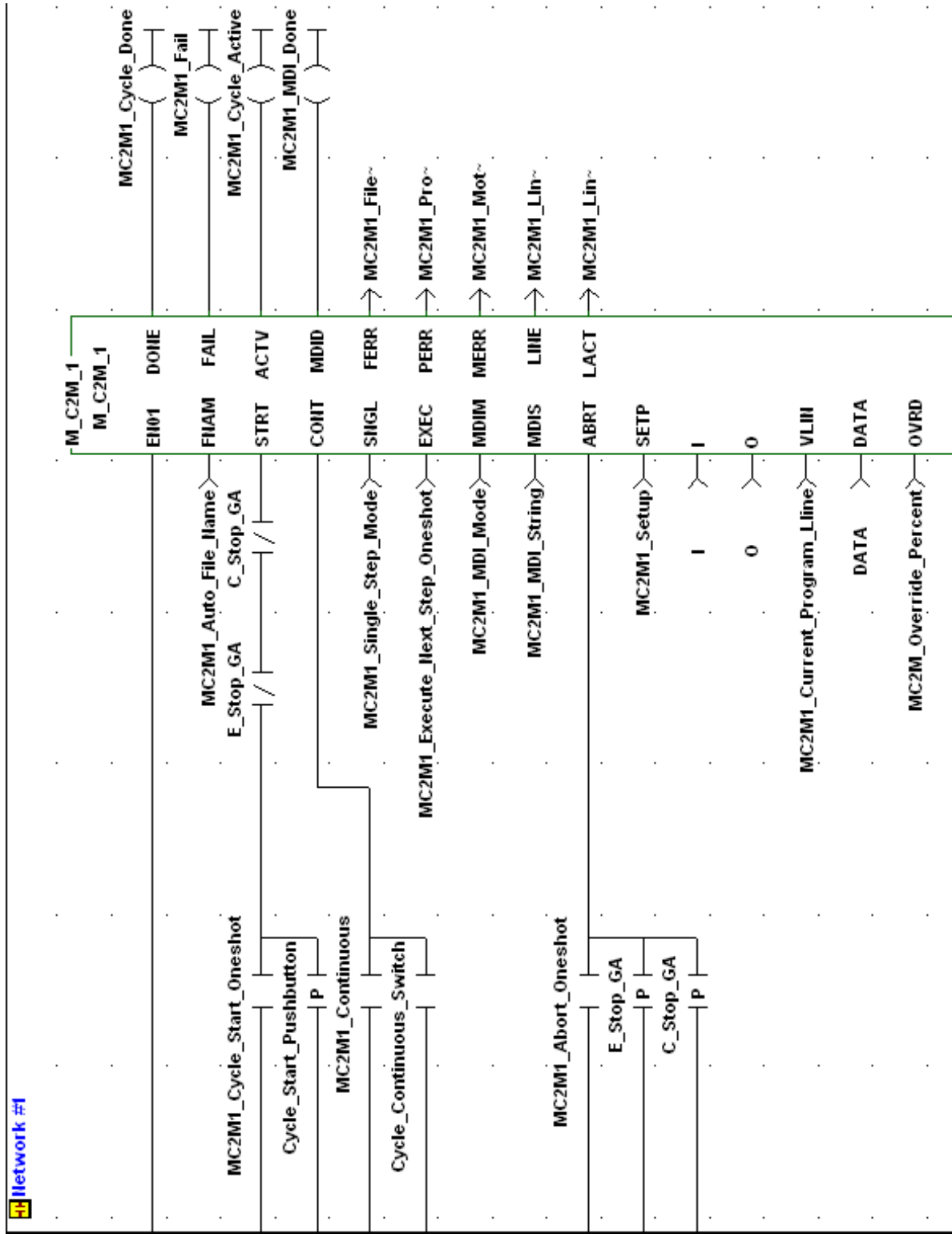
Code	Type	Use	Description
M10x	Modal	Turn On Output	Turn on user output 1 to 9, all outputs cleared by abort or end of program when not continuous
M20x		Turn Off Output	Turn off user output 1 to 9. M10x and M20x instructions programmed in the same line as motion (G00..G03) will cause the specified output to turn on/off when the queued move becomes active

M10x and M20x are the modal user output control group. M20x, outputs off, is the default state. Once selected the output state is maintained and does not need to be specified again until you wish to change it. It will be reset to M20x, outputs off, after M30 end of program when not in continuous mode or when program execution is aborted. If an M10x, Turn On, or an M20x, Turn Off, is specified in a line without motion it will take effect immediately. If specified in a line with motion, it will take effect when the move becomes active. Multiple M10x's and M20x's may be programmed in a single line. Outputs 1 to 9 are presented to the user application via the O input of M_C2M_1.

Code	Type	Use	Description
M50x		Wait for Input On	Wait until specified input (x = 1 to 9) is on
M60x		Wait for Input Off	Wait until specified input (x = 1 to 9) is off
			M50x and M60x instructions programmed in the same line as motion (G00..G03) will cause the queuing of the specified move to be delayed until the wait for on/off states are satisfied

M50x, Wait of Input On, and M60x, Wait for Input Off, are non-modal and effect only the line they are programmed in. If the line contains M50x and M60x instructions, the rest of the line will not be executed until all of the M50x and M60x wait for inputs are satisfied.

M_C2M_1 ASFB Template



INPUTS

Input	Description
EN01	Enable - must be energized at all times.
FNAM	Filename string – Name of program file to be executed. Typically “RAMDISK:<filename.txt>\$00. The string must be terminated by \$00.
STRT	Pulse to start program execution - Modal data is reset.
CONT	Energize to repeat program execution continuously.
SNGL	Energize to enter single step mode. After using single step mode deenergizing SNGL will cause execution to continue. When entering single step mode all pre-processed motion (up to three moves) will execute before execution is stalled. When in single step mode the VLIN will show the instruction that will be executed when EXEC is pulsed. When in single step mode, pulse MDIM to activate Manual Data Input Mode.
EXEC	Pulse to execute next instruction when in single step mode. Pulse to execute instruction in MDIS when in MDI (Manual Data Input) mode.
MDIM	Energize to select Manual Data Input mode. While in MDI mode pulse EXEC to execute the instruction specified by MDIS.
MDIS	Manual Data Input String. M and G code instruction to be executed when EXEC is pulsed when in MDI mode.
ABRT	Pulse to abort program execution. When program execution is aborted all user outputs (O.O1 to O.O9) will be deenergized, M03 will turn off, SDAT will be cleared to zero, and all axes motion will be aborted.
SETP	See the table below for a description of the Setup data structure.
I	User Input structure I.I1 to I.I9 corresponding to Wait for Input On M501 to M509 and Wait for Input Off M601 to M609, respectively.
O	User Output structure O.O1 to O.O9 corresponding to turn On Output M101 to M109 and Turn Off Output M201 to M209, respectively.
VLIN	String which will show the current program line being executed. Note that due to preprocessing VLIN can be up to three lines ahead of actual application motion.
DATA	M_C2M_1 Data Structure. Provides the data for the active program line.
OVRD	Path feedrate override. Specify from 0 to 255 percent of programmed (or Rapid) feedrate.

SETUP DATA STRUCTURE - The setup data structure defines the operation of this application of M_C2M_1. Values should be specified as initial values in Software Declarations and not changed while running.

Name	Type	Description
Setup	STRUCT	Setup Data Structure
.X_ACTIVE	BOOL	X, Y, Z, W, and B_ACTIVE are set to 1 to indicate axis is active in this application.
.X_DG2R	INT	
.X_Axis_Number	USINT	X, Y, Z, W, A and B_AXIS_Number are set to 1 to 16 to indicate each axis' servo setup axis number.
.Y_ACTIVE	BOOL	
.Y_DG2R	INT	
.Y_Axis_Number	USINT	
.Z_ACTIVE	BOOL	X, Y, Z, W, A and B_DG2R are set to indicate the number of digits to the right of the implied decimal point. See Implied Decimal Point Data section below.
.Z_DG2R	INT	
.Z_Axis_Number	USINT	
.W_ACTIVE	BOOL	
.W_DG2R	INT	
.W_Axis_Number	USINT	
.A_ACTIVE	BOOL	
.A_DG2R	INT	
.A_Axis_Number	USINT	
.B_ACTIVE	BOOL	
.B_DG2R	INT	
.B_Axis_Number	USINT	Indicate the number of digits to the right of the implied decimal point. See Implied Decimal Point Data section below.
.I_DG2R	INT	
.J_DG2R	INT	
.K_DG2R	INT	
.F_DG2R	INT	
.S_DG2R	INT	The feedrate used for G00 Rapid moves
.RAPID	DINT	
.BNDW	DINT	The Circular Endpoint on circle bandwidth. See PiCPro Function Block Help for M_SCRVLC for further information.
.PATH	USINT	Typically set to 1. Set to 2,3 or 4 for applications running up to four simultaneous M_C2M_1 instances. See Interpolation Paths section below for more information.

.ACCEL	LREAL	Path S-Curve Acceleration/Deceleration definition. See PiCPro Function Block Help for M_SCRVLC for more information.
.JERK	LREAL	
.MAXF	DINT	
.X Offset	DINT	X axis absolute endpoint offset value
.X Offset_Ignore	BOOL	Set to ignore X axis endpoint offset
.Y Offset	DINT	Y axis absolute endpoint offset value
.Y Offset_Ignore	BOOL	Set to ignore Y axis endpoint offset
.Z Offset	DINT	Z axis absolute endpoint offset value
.Z Offset_Ignore	BOOL	Set to ignore Z axis endpoint offset
.W Offset	DINT	W axis absolute endpoint offset value
.W Offset_Ignore	BOOL	Set to ignore W axis endpoint offset
.A Offset	DINT	A axis absolute endpoint offset value
.A Offset_Ignore	BOOL	Set to ignore A axis endpoint offset
.B Offset	DINT	B axis absolute endpoint offset value
.B Offset_Ignore	BOOL	Set to ignore B axis endpoint offset
.Structure_Check_Constant	DINT	Must have initial value of +12345. Used to verify that the application Setup structure matches the M_C2M_1 structure.
END_STRUCT		

The offset data provides the application with the ability to identify offset values for X, Y, Z, W, A, and B which will be added to all absolute (G90) position endpoints. This includes G00, G01, G02 and G03 move endpoints. Incremental mode (G91) distances are not affected. The offset values are applied when the corresponding "ignore" flag is not set.

INPUT DATA STRUCTURE - The input data structure “I” allows integration of user inputs with the execution of the program.

Name	Type	Description
I	STRUCT	Input Data Structure
.I1	BOOL	Input 1, M501 Wait for Input On , M601 Wait for Input Off
.I2	BOOL	Input 2, M502 Wait for Input On , M602 Wait for Input Off
.I3	BOOL	Input 3, M503 Wait for Input On , M603 Wait for Input Off
.I4	BOOL	Input 4, M504 Wait for Input On , M604 Wait for Input Off
.I5	BOOL	Input 5, M505 Wait for Input On , M605 Wait for Input Off
.I6	BOOL	Input 6, M506 Wait for Input On , M606 Wait for Input Off
.I7	BOOL	Input 7, M507 Wait for Input On , M607 Wait for Input Off
.I8	BOOL	Input 8, M508 Wait for Input On , M608 Wait for Input Off
.I9	BOOL	Input 9, M509 Wait for Input On , M609 Wait for Input Off
.M01_Input	BOOL	If ON enter single step mode when M01 programmed
END_STRUCT		

OUPUT DATA STRUCTURE - The input data structure “O” allows integration of user outputs with the execution of the program.

Name	Type	Description
O	STRUCT	Output Data Structure
.01	BOOL	Output 1, M101 Turn On Output, M201 Turn Off Output
.02	BOOL	Output 2, M102 Turn On Output, M202 Turn Off Output
.03	BOOL	Output 3, M103 Turn On Output, M203 Turn Off Output
.04	BOOL	Output 4, M104 Turn On Output, M204 Turn Off Output
.05	BOOL	Output 5, M105 Turn On Output, M205 Turn Off Output
.06	BOOL	Output 6, M106 Turn On Output, M206 Turn Off Output
.07	BOOL	Output 7, M107 Turn On Output, M207 Turn Off Output
.08	BOOL	Output 8, M108 Turn On Output, M208 Turn Off Output
.09	BOOL	Output 9, M109 Turn On Output, M209 Turn Off Output
END_STRUCT		

DATA STRUCTURE - The M_C2M_1 Data Structure "DATA" provides the application with a view of all of the program data variables and flags.

Name	Type	Description
.X_Dat	DINT	X Command data value.
.Y_Dat	DINT	Y Command data value.
.Z_Dat	DINT	Z Command data value.
.W_Dat	DINT	W Command data value.
.A_Dat	DINT	A Command data value.
.B_Dat	DINT	B Command data value.
.F_Dat	DINT	F Command data value.
.S_Dat	DINT	S Command data value.
.N_Dat	DINT	N Command data value.
.I_Dat	DINT	I Command data value.
.J_Dat	DINT	J Command data value.
.K_Dat	DINT	K Command data value.
.L_Dat	DINT	L Command data value.
.G00	BOOL	G00 Command is active.
.G01	BOOL	G01 Command is active.
.G02	BOOL	G02 Command is active.
.G03	BOOL	G03 Command is active.
.G17	BOOL	G17 Command is active.
.G18	BOOL	G18 Command is active.
.G19	BOOL	G19 Command is active.
.G90	BOOL	G90 Command is active.
.G91	BOOL	G91 Command is active.
.M00_M01_Program_Pause	BOOL	M00/M01 Command is active.
.M03	BOOL	M03 Command is active.
.M04	BOOL	M04 Command is active.
.M05	BOOL	M05 Command is active.
.M101	BOOL	M101 Command is active.
.M102	BOOL	M102 Command is active.
.M103	BOOL	M103 Command is active.
.M104	BOOL	M104 Command is active.
.M105	BOOL	M105 Command is active.
.M106	BOOL	M106 Command is active.
.M107	BOOL	M107 Command is active.
.M108	BOOL	M108 Command is active.
.M109	BOOL	M109 Command is active.
.Wait_for_Input	BOOL	Wait_for_Input Command is active.

.Wait_for_Time_Delay	BOOL	Wait_for_Time_Delay Command is active.
.Time_Left	UDINT	Time left for the Wait_for_Time_Delay Command.
.Wait_for_In_Position	BOOL	Wait_for_In_Position_Command is active.
.Wait_for_Servo_Queue	BOOL	Wait_for_Servo_Queue Command is active.
.Structure_Check_Constant	DINT	Must have initial value of 12345. Used to verify that the Data structure matches the M_C2M_1 structure.

M_C2M_1 OUTPUTS

Outputs	Description
DONE	Indicates that the program execution has completed successfully or was aborted by and ABRT request.
FAIL	Indicates that an error occurred while trying to execute the program. The type of error is indicated by FERR, PERR and MERR as described below. When FAIL occurs all user outputs will be reset, all axes motion aborted and the program file will be closed.
ACTV	Active indicates that program execution is in progress
MDID	MDI mode done - set on entry to MDI mode and when execution of MDIS completes during MDI mode operation.
FERR	File read error - Using PiCPro Online Help, refer to I/O Function Block Error Codes under Error Codes for a description of these errors.
PERR	Program Error - See the Program Error table below for a description of these error codes.
MERR	Motion Error - See PiCPro Function Block Help for M_SCRVLC and refer to the ERR output for a description of error codes.
LINE	Indicates the Line number.
LACT	Indicates that the Line is active.

PERR Program Errors - This table provides a description of errors that will be reported if an improperly formatted program is encountered.

Error Number	Description
7001	CRLF line terminator not found
7002	Unrecognizable Field Code (i.e. not N,X,Y...M)
7003	Unrecognizable G Code
7004	Unrecognizable Mxx Code
7005	Bad data for N Code
7006	Missing CRLF line terminator (line wider than 128 characters)
7007	Missing LF line terminator
7008	Missing CRLF line terminator
7009	End of File missing CRLF line terminator
7010	End of File missing CRLF line terminator
7011	Poorly formed data
7012	SETP structure doesn't match M_C2M's SETP structure
7013	Multiple G00, G01, G02, G03, G92, codes in single line
7014	Incomplete line programmed, i.e. "G03 G17X1 Y1 11", J data word is missing so move could not be executed
7015	% Start of program not found
7016	SETP.Application_Name does not match application name specified at beginning of program (e.g. %/Saw Type 1/)

IMPLIED DECIMAL POINT DATA - Implied decimal point data accommodates the fact that position and feedrate information used with PiCPro motion control programming is stored in 32-bit double integer variables. The M and G code program will need to specify position and feedrate information with a decimal point. In the Setup data structure input to the M_C2M_1 ASFB the digits-to-right (i.e. X_DG2R) specified for each program code is used to scale data appropriately to the needs of the PiCPro motion control instructions. The table below shows the effect of setting the DG2R precision to various values.

Digits-to-Right	Data in Program Line	Data Delivered to Motion Function
3	X123	123000
3	X.1	100
3	X1.2	1200
3	X1	1000
3	X1.0002	1000
4	X123	1230000
4	X.1	1000
4	X1.2	12000
4	X1	10000
4	X1.0002	10002

Scaling from programmed units to machine servo feedback units is defined when programming the application specific servo setup data using PiCPro.

Path Positioning Using M_C2M_1

When G00 - Rapid is active the axes specified in the line will be move to the end-point (G90 absolute) or the incremental distance (G91 incremental) specified by the X, Y, Z, W, A and B data words at the rate specified by SETUP.RAPID. Unprogrammed axes will not move.. .

G00 Rapid Mode

G00 - The axes specified in the line will be move to the endpoint (G90 absolute) or the incremental distance (G91 incremental) specified by the X, Y, Z, W, A and B data words at the rate specified by SETUP.RAPID. Unprogrammed axes will not move.

G00 Rapid Mode Example

```
%/Start of Program/  
N1000 G90 G00 X10.0 Y5.0 /Position X to 10.0 and Y to 5.0 at rapid rate /  
N1010 X15 Y0 G09 /Position X to 10 and Y to 0 at rapid rate, decel to zero/  
N1020 G91 Y1 /Move Y incrementally 1 at rapid rate/  
N1030 G09 Y1 /Move Y incrementally 1 at rapid rate, decel to zero /
```

G01 Linear Interpolation

G01 - The axes specified in the line will be move to the endpoint (G90 absolute) or the incremental distance (G91 incremental) specified by the X, Y, Z, W, A and B data words at the rate specified by F using linear interpolation. Unprogrammed axes will not move.

G01 Linear Interpolation Example

```
%/Start of Program/  
N1000 G90 G01 X10.0 Y5.0 F100.0 /Position X to 10.0 and Y to 5.0 at a path rate of 100 /  
N1010 X15 Y0 G09 /Position X to 10 and Y to 0 at a path rate of 100, decel to zero/  
N1020 G91 Y1 /Move Y incrementally 1 at a path rate of 100/  
N1030 G09 Y1 /Move Y incrementally 1 at path rate of 100, decel to zero /
```

G02 Circular Clockwise and G03 Counter Clockwise Circular Interpolation

Plane	Mode	Description
G17, XY	G02, G03	Use circular interpolation to move to X and Y endpoints (incremental or absolute based on G90/G91), I and J centerpoints (always incremental from start of circle) at F modal path feedrate. If Z, W, A, or B are programmed in the same line they will be moved in a third axis departure move and arrive at their programmed position simultaneously with X and Y.
G18, XZ	G02, G03	Use circular interpolation to move to X and Z endpoints (incremental or absolute based on G90/G91), I and K centerpoints (always incremental from start of circle) at F modal path feedrate. If Y, W, A or B are programmed in the same line they will be moved in a third axis departure move and arrive at their programmed position simultaneously with X and Z.
G19, YZ	G02, G03	Use circular interpolation to move to Y and Z endpoints (incremental or absolute based on G90/G91), J and K centerpoints (always incremental from start of circle) at F modal path feedrate. If X, W, A or B are programmed in the same line they will be moved in a third axis departure move and arrive at their programmed position simultaneously with Y and Z.

G02 Clockwise and G03 Counter Clockwise Circular Interpolation Examples

%/Start of Program/

G90 /Select Absolute Positioning Mode

F800 /Specify path feedrate of 800 /

G01 X8.000 Y0.000

G01 X16.472 Y0.000

G03 X17.472 Y1.000 I0.000 J1.000 / Circular Counter Clockwise /

G01 X17.472 Y11.707

G03 X9.472 Y19.707 I-8.000 J0.000

G01 X1.000 Y19.707

G03 X0.000 Y18.707 I0.000 J-1.000

G01 X0.000 Y8.000

G03 X8.000 Y0.000 I8.000 J0.000

G09

Coordinating User Outputs with Motion

To turn outputs on and off in step with servo axis positioning program M10x and M20x instructions in the same line as the desired motion.

%/Start of Program/

N1000 M101 / Immediately Turn on Output 1 /

N1010 G04 F1.5 /Wait for 1.5 seconds

N1020 M102 / Immediately Turn off Output 1 /

N1030 M101 G91 G01 F100.0 X1.00 / Output 1 turns on when this move begins/

N1040 M102 X2.0 / Output 2 turns on when this move begins /

N1050 X3.0 G09

N1060 M103 / Output 3 turns on when the move in N1050 completes /

Coordinating Motion with User Inputs

The M50x Wait for Input On and M60x Wait for Input Off instructions are used to coordinate program execution with the state of user application inputs.

%/Start of Program/

N1000 M501 / Program execution stalls until user input 1 is on /

N1010 M502 G91 G01 F100.0 X1.00 / When user input 2 is on start move of 1 /

N1010 M503 G04 F1.5 / When user input 3 is on begin delay of 1.5 seconds /

Effect of Motion Que and Program Execution

To provide continuous path motion a queuing system is used to buffer one move which will blend with the currently active move with no deceleration of the servo axes. This queuing system requires that program lines be read and executed while motion started by previous lines is completed. This will lead to the program line display, VLIN, showing the line currently being parsed and queued and this line may be many lines after the line which started the current motion.

Instruction

Action

N1000 G90 G01 X100

Starts move of X to 100

N1010 G90 G01 X200

Queues move of X to 200

N1030 G90 G01 X300

Waits until queue move in N1000 completes which will make room on the queue

Any lines between N1000 and N1030 would execute immediately. Including G09 decel to zero changes the execution as described below.

Instruction

Action

N1000 G90 G01 X100 G09

Starts move of X to 100, and wait till in position

N1010 G90 G01 X200 G09

Start move of X to 200 and wait till in position

N1030 G90 G01 X300 G09

Start move of X to 300 and wait till in position

Using the G09 stalls execution until the move in the line completes

Adding user outputs to the same examples also shows the effect of the queue

Instruction	Action
N1000 G90 G01 X100 M101 M102	Starts move of X to 100, turn on output 1 Output 2 turns on while move to 100 is occurring
N1010 G90 G01 X200 M103 M104	Queues move of X to 200, output 3 will turn on when this move becomes active Output 4 turns on while move to 100 in N1000 is occur- ing. Output 4 will turn on before output 3
N1030 G90 G01 X300	Waits until queue move in N1000 completes which will make room on the queue.

Line Execution

Programs are executed a line at a time. In a line containing multiple instructions the order of execution is based on the type of instruction, not the order of its occurrence within the line. Line execution is performed in the following order:

- 1 - If the line contains any Wait for Input On or Off instructions execution will wait until all of the conditions have been satisfied.
- 2 - If the line contains a G04 dwell instruction execution will delay until the time specified by F passes.
- 3 - If the line contains a G00 to G03 motion instruction execution will wait until the servo queue is ready to accept the next move.

Lines Containing Incomplete Motion Instructions

Incomplete motion instructions that can be detected will result in FAIL being set with PERR = 7014 and the move will not be executed. An example of a line containing an incomplete motion instruction would be “G03 G17 X1 Y2 I3”. In this case the J data word specifying the Y axis centerpoint is missing resulting in the M&G program execution being ignored.

Simultaneous Multiple Paths

M_C2M_1 will support up to four completely independent motion programs on four separate interpolators. To do this the users application must have four separate instances of M_C2M_1. The SETUP.PATH should be set to 1,2,3 and 4 for instance 1 to 4, respectively. The table below describes the servo axis numbering in the applications servo setup data that must be used for each path.

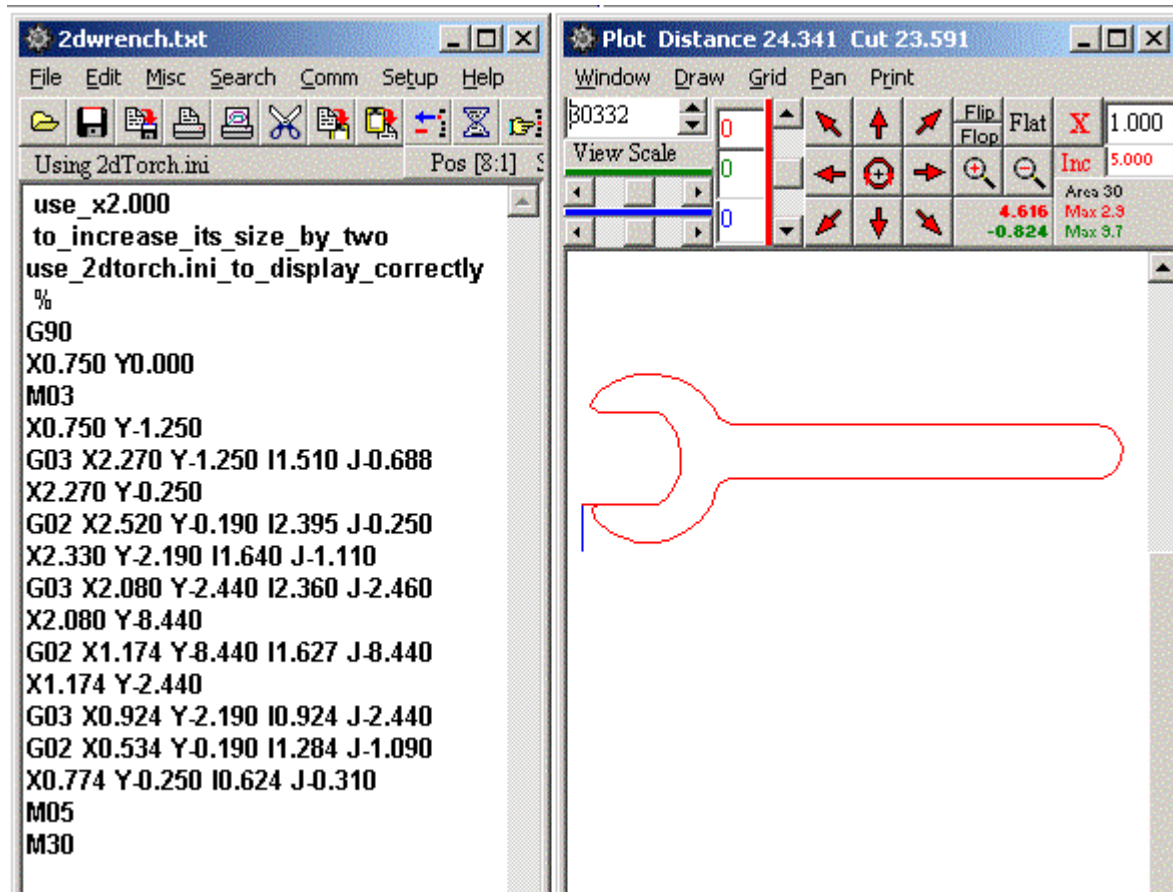
Program File Structure

Program lines can contain up to 126 characters and must be terminated with a carriage return (\$0D) and line feed (\$0A). Fields within the line must be combinations of letters followed by values, i.e. X-123.456, no space can occur between the letter and the value. One or more spaces must occur between fields, i.e. X123 Y245 is valid, X1Y2 is invalid.

The last line of the program must contain a carriage return (\$0D) and line feed (\$0A).

Using M_C2M_1 with Third Party Cad-to-Motion Tools

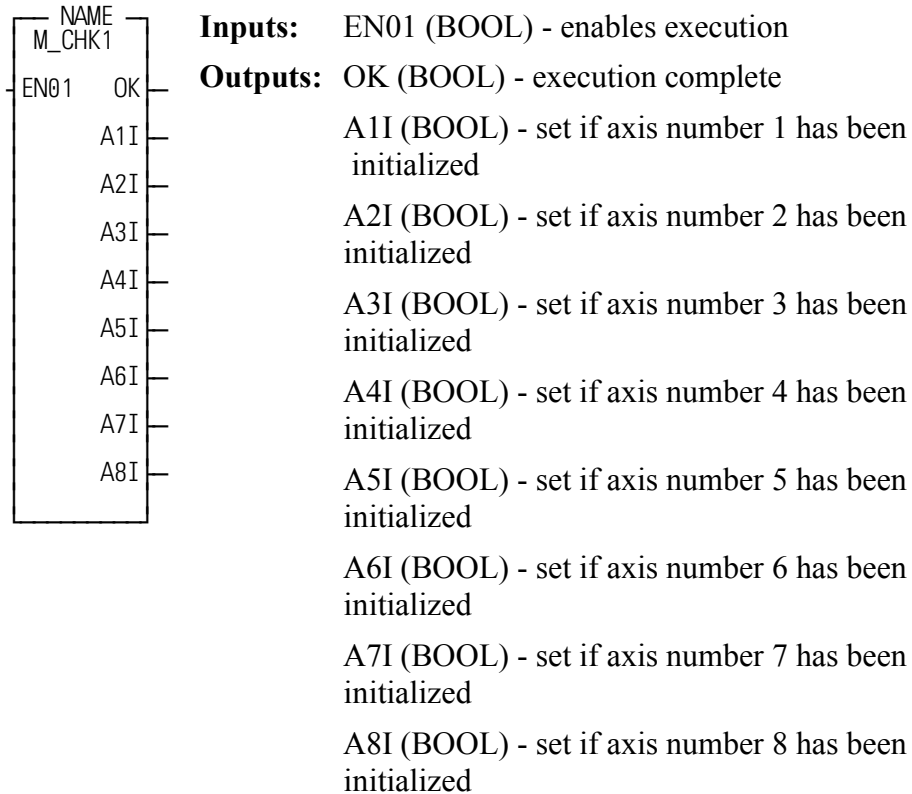
M_C2M_1, the Cad2Motion ASFB can be used to translate the output of a third-party DXF to Ascii file conversion program like Gcode2000 to machine and motion control. For more information log onto http://members.aol.com/_ht_a/gcode/decode/index.htm.



M_CHK1

Check for Servo Axis Initialized

USER/M_INIT



```
<<INSTANCE NAME>>:M_CHK1(EN01 := <<BOOL>>, OK => <<BOOL>>,
A1I => <<BOOL>>, A2I => <<BOOL>>, A3I => <<BOOL>>, A4I =>
<<BOOL>>, A5I => <<BOOL>>, A6I => <<BOOL>>, A7I => <<BOOL>>,
A8I => <<BOOL>>);
```

This function block checks to see which servo axes numbered from 1 to 8 have been initialized by the user's servo setup function

The OK output of the STRTSERV function should be wired directly to the enable (EN01) input of this function.

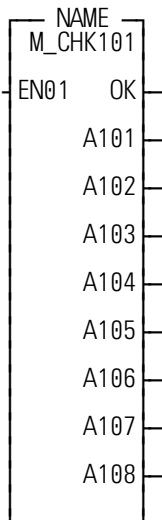
The outputs of this function will remain set even after the function is no longer enabled.

The outputs of this function can be used to ensure the correct setup information has been used. They can also be used as conditional contacts to qualify other motion functions.

M_CHK101

Check for Servo Axis Initialized

USER/M_INIT



Inputs: EN01 (BOOL) - enables execution

Outputs: OK (BOOL) - execution complete

A101 (BOOL) - set if axis number 101 has been initialized (servo axis 17).

A102 (BOOL) - set if axis number 102 has been initialized (servo axis 18).

A103 (BOOL) - set if axis number 103 has been initialized (servo axis 19).

A104 (BOOL) - set if axis number 104 has been initialized (servo axis 20).

A105 (BOOL) - set if axis number 105 has been initialized (servo axis 21).

A106 (BOOL) - set if axis number 106 has been initialized (servo axis 22).

A107 (BOOL) - set if axis number 107 has been initialized (servo axis 23).

A108 (BOOL) - set if axis number 108 has been initialized (servo axis 24).

```
<<INSTANCE NAME>>:M_CHK101(EN01 := <<BOOL>>, OK =>
<<BOOL>>, A101 => <<BOOL>>, A102 => <<BOOL>>, A103 =>
<<BOOL>>, A104 => <<BOOL>>, A105 => <<BOOL>>, A106 =>
<<BOOL>>, A107 => <<BOOL>>, A108 => <<BOOL>>);
```

This function block checks to see which servo axes numbered from 101 to 108 (servo axes 17 to 24) have been initialized by the user's servo setup function.

The OK output of the STRTSERV function should be wired directly to the enable (EN01) input of this function.

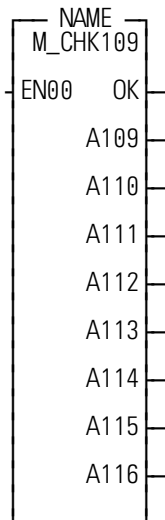
The outputs of this function will remain set even after the function is no longer enabled.

The outputs of this function can be used to ensure the correct setup information has been used. They can also be used as conditional contacts to qualify other motion functions.

M_CHK109

Check for Servo Axis Initialized

USER/M_INIT



Inputs: EN01 (BOOL) - enables execution

Outputs: OK (BOOL) - execution complete

A109 (BOOL) - set if axis number 109 has been initialized (servo axis 25).

A110 (BOOL) - set if axis number 110 has been initialized (servo axis 26).

A111 (BOOL) - set if axis number 111 has been initialized (servo axis 27).

A112 (BOOL) - set if axis number 112 has been initialized (servo axis 28).

A113 (BOOL) - set if axis number 113 has been initialized (servo axis 29).

A114 (BOOL) - set if axis number 114 has been initialized (servo axis 30).

A115 (BOOL) - set if axis number 115 has been initialized (servo axis 31).

A116 (BOOL) - set if axis number 116 has been initialized (servo axis 32).

```
<<INSTANCE NAME>>:M_CHK109(EN01 := <<BOOL>>, OK => <<BOOL>>  
A109 => <<BOOL>>, A110 => <<BOOL>>, A111 => <<BOOL>>, A112 =>  
<<BOOL>>, A113 => <<BOOL>>, A114 => <<BOOL>>, A115 =>  
<<BOOL>>, A116 => <<BOOL>>);
```

This function block checks to see which servo axes numbered from 109 to 116 (servo axes 25 to 32) have been initialized by the user's servo setup function.

The OK output of the STRTSERV function should be wired directly to the enable (EN01) input of this function.

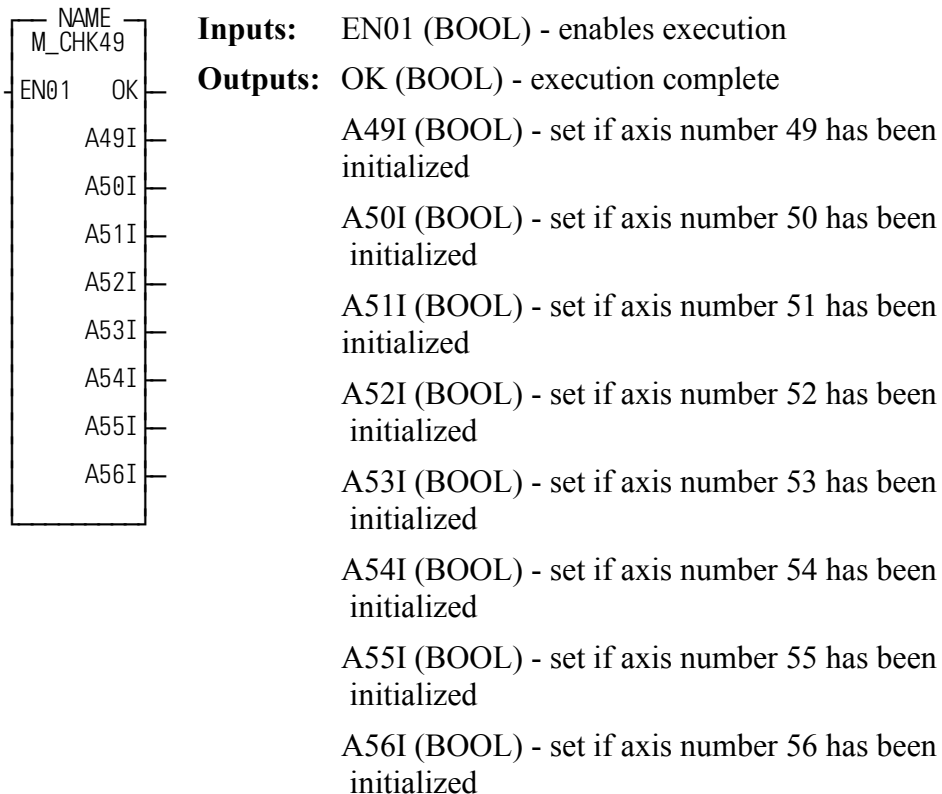
The outputs of this function will remain set even after the function is no longer enabled.

The outputs of this function can be used to ensure the correct setup information has been used. They can also be used as conditional contacts to qualify other motion functions.

M_CHK49

Check for Digitizing Axis Initialized

USER/M_INIT



```
<<INSTANCE NAME>>:M_CHK49(EN01 := <<BOOL>>, OK => <<BOOL>>  
A49I => <<BOOL>>, A50I => <<BOOL>>, A51I => <<BOOL>>, A52I =>  
<<BOOL>>, A53I => <<BOOL>>, A54I => <<BOOL>>, A55I =>  
<<BOOL>>, A56I => <<BOOL>>);
```

This function block checks to see which digitizing axes numbered from 49 to 56 have been initialized by the user's servo setup function.

The OK output of the STRTSERV function should be wired directly to the enable (EN01) input of this function.

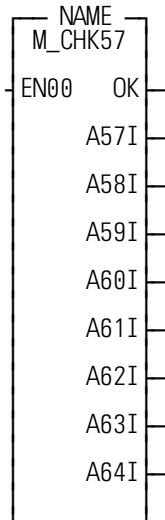
The outputs of this function will remain set even after the function is no longer enabled.

The outputs of this function can be used to ensure the correct setup information has been used. They can also be used as conditional contacts to qualify other functions.

M_CHK57

Check for Digitizing Axis Initialized

USER/M_INIT



Inputs: EN00 (BOOL) - enables execution

Outputs: OK (BOOL) - execution complete

A57I (BOOL) - set if axis number 57 has been initialized

A58I (BOOL) - set if axis number 58 has been initialized

A59I (BOOL) - set if axis number 59 has been initialized

A60I (BOOL) - set if axis number 60 has been initialized

A61I (BOOL) - set if axis number 61 has been initialized

A62I (BOOL) - set if axis number 62 has been initialized

A63I (BOOL) - set if axis number 63 has been initialized

A64I (BOOL) - set if axis number 64 has been initialized

```
<<INSTANCE NAME>>:M_CHK57(EN00 := <<BOOL>>, OK => <<BOOL>>  
A57I => <<BOOL>>, A58I => <<BOOL>>, A59I => <<BOOL>>, A60I =>  
<<BOOL>>, A61I => <<BOOL>>, A62I => <<BOOL>>, A63I =>  
<<BOOL>>, A64I => <<BOOL>>);
```

This function block checks to see which digitizing axes numbered from 57 to 64 have been initialized by the user's servo setup function.

The OK output of the STRTSERV function should be wired directly to the enable (EN00) input of this function.

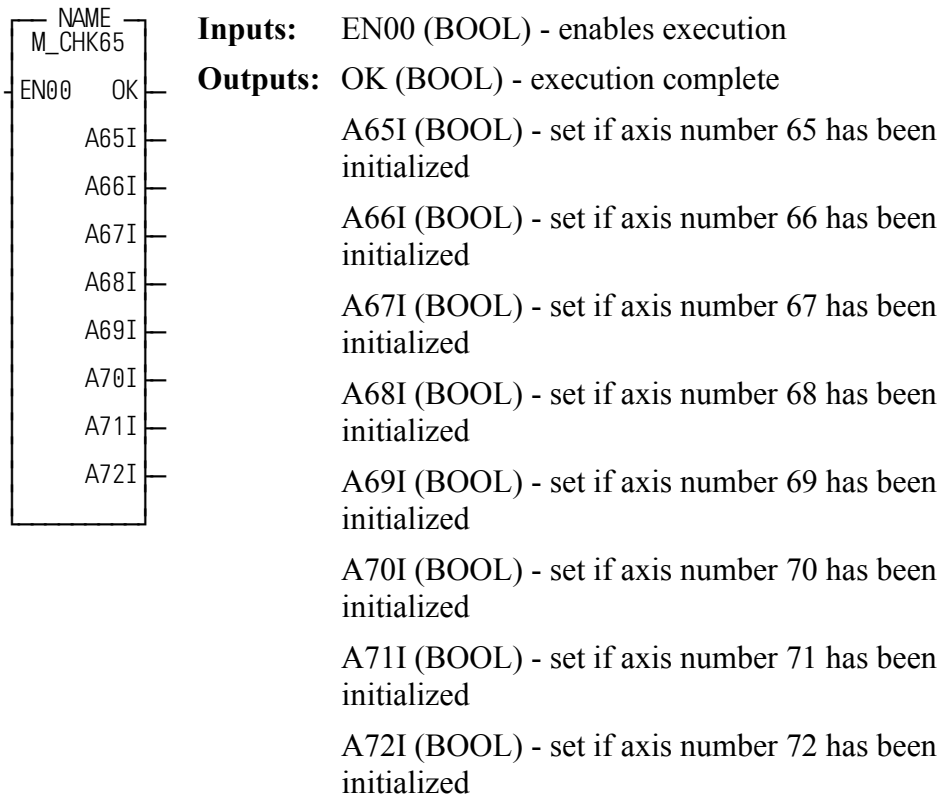
The outputs of this function will remain set even after the function is no longer enabled.

The outputs of this function can be used to ensure the correct setup information has been used. They can also be used as conditional contacts to qualify other functions.

M_CHK65

Check for Digitizing Axis Initialized

USER/M_INIT



```
<<INSTANCE NAME>>:M_CHK65(EN00 := <<BOOL>>, OK => <<BOOL>>  
A65I => <<BOOL>>, A66I => <<BOOL>>, A67I => <<BOOL>>, A68I =>  
<<BOOL>>, A69I => <<BOOL>>, A70I => <<BOOL>>, A71I =>  
<<BOOL>>, A72I => <<BOOL>>);
```

This function block checks to see which digitizing axes numbered from 65 to 72 have been initialized by the user's servo setup function.

The OK output of the STRTSERV function should be wired directly to the enable (EN00) input of this function.

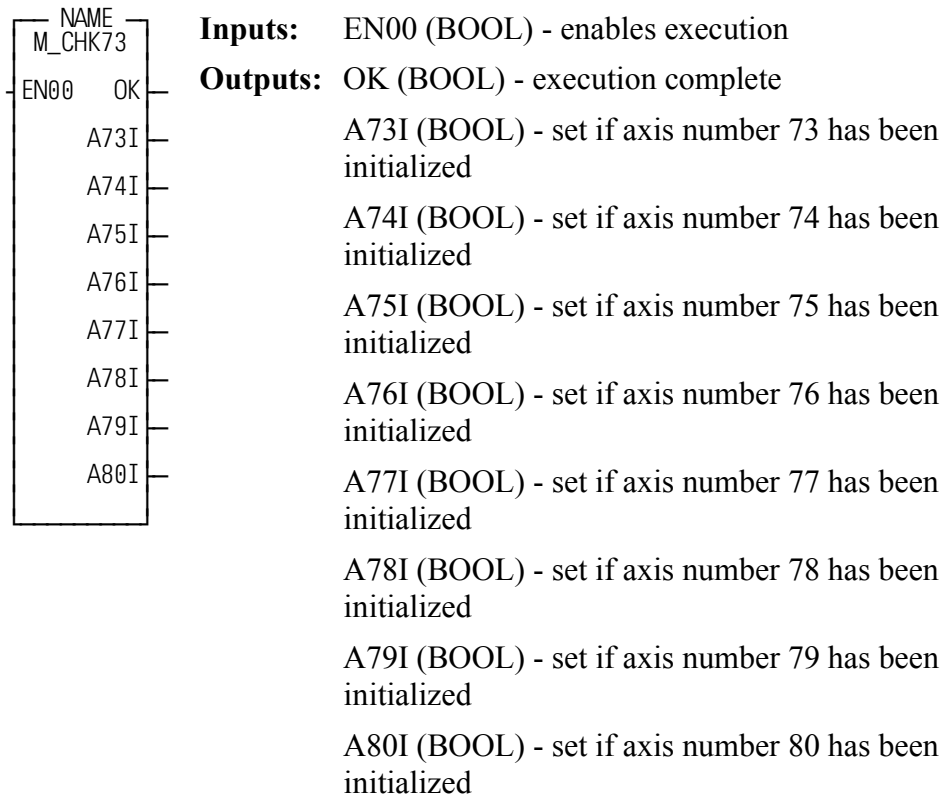
The outputs of this function will remain set even after the function is no longer enabled.

The outputs of this function can be used to ensure the correct setup information has been used. They can also be used as conditional contacts to qualify other functions.

M_CHK73

Check for Digitizing Axis Initialized

USER/M_INIT



```
<<INSTANCE NAME>>:M_CHK73(EN00 := <<BOOL>>, OK => <<BOOL>>  
A73I => <<BOOL>>, A74I => <<BOOL>>, A75I => <<BOOL>>, A76I =>  
<<BOOL>>, A77I => <<BOOL>>, A78I => <<BOOL>>, A79I =>  
<<BOOL>>, A80I => <<BOOL>>);
```

This function block checks to see which digitizing axes numbered from 73 to 80 have been initialized by the user's servo setup function.

The OK output of the STRTSERV function should be wired directly to the enable (EN00) input of this function.

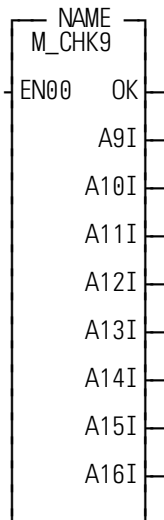
The outputs of this function will remain set even after the function is no longer enabled.

The outputs of this function can be used to ensure the correct setup information has been used. They can also be used as conditional contacts to qualify other functions.

M_CHK9

Check for Servo Axis Initialized

USER/M_INIT



Inputs: EN00 (BOOL) - enables execution

Outputs: OK (BOOL) - execution complete

A9I (BOOL) - set if axis number 9 has been initialized

A10I (BOOL) - set if axis number 10 has been initialized

A11I (BOOL) - set if axis number 11 has been initialized

A12I (BOOL) - set if axis number 12 has been initialized

A13I (BOOL) - set if axis number 13 has been initialized

A14I (BOOL) - set if axis number 14 has been initialized

A15I (BOOL) - set if axis number 15 has been initialized

A16I (BOOL) - set if axis number 16 has been initialized

```
<<INSTANCE NAME>>:M_CHK9(EN00 := <<BOOL>>, OK => <<BOOL>>  
A9I => <<BOOL>>, A10I => <<BOOL>>, A11I => <<BOOL>>, A12I =>  
<<BOOL>>, A13I => <<BOOL>>, A14I => <<BOOL>>, A15I =>  
<<BOOL>>, A16I => <<BOOL>>);
```

This function block checks to see which servo axes numbered from 9 to 16 have been initialized by the user's servo setup function.

The OK output of the STRTSERV function should be wired directly to the enable (EN00) input of this function.

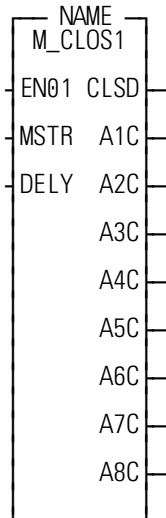
The outputs of this function will remain set even after the function is no longer enabled.

The outputs of this function can be used to ensure the correct setup information has been used. They can also be used as conditional contacts to qualify other motion functions.

M_CLOS1

Close Loop on Servo Axes 1 to 8

USER/M_INIT



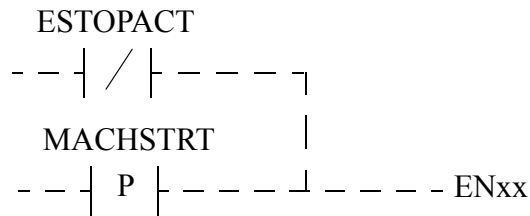
- Inputs:** EN01 (BOOL) - enables execution
MSTR (BOOL) - machine start input
DELY (TIME) - amount of time that will elapse after a positive transition of MSTR until the loops will be closed
- Outputs:** CLSD (BOOL) - one or more of axes 1 to 8 have their position loops closed
A1C (BOOL) - set when the loop is closed on axis 1
A2C (BOOL) - set when the loop is closed on axis 2
A3C (BOOL) - set when the loop is closed on axis 3
A4C (BOOL) - set when the loop is closed on axis 4
A5C (BOOL) - set when the loop is closed on axis 5
A6C (BOOL) - set when the loop is closed on axis 6
A7C (BOOL) - set when the loop is closed on axis 7
A8C (BOOL) - set when the loop is closed on axis 8

```
<<INSTANCE NAME>>:M_CLOS1(EN01 := <<BOOL>>, MSTR :=  
  <<BOOL>>, DELY := <<TIME>>, CLSD => <<BOOL>> A1C =>  
  <<BOOL>>, A2C => <<BOOL>>, A3C => <<BOOL>>, A4C => <<BOOL>>,  
  A5C => <<BOOL>>, A6C => <<BOOL>>, A7C => <<BOOL>>, A8C =>  
  <<BOOL>>);
```

This function block is used to reset the E-stop, C-stop, and programming errors on servo axes 1 through 8 when the machine start input is pulsed. It closes the loop on servo axes 1 through 8 after the machine start input is pulsed and a programmable time delay has elapsed. It drops the loop closed flag if an E-stop fault occurs.

This function block can be enabled every scan. If the enable input changes from ON to OFF during the time delay after machine start, the function block will abort the time delay and not close the position loops.

If there are conditions that should abort the sequence to close the position loops (such as an electrical E-stop condition during the time delay), then the enable should include both the positive transition of the machine start input and the current state of the electrical E-stop status as shown below.



The reason for these two input conditions is to provide the enable at the start of the time delay (with the P contact of the machine start signal) and to maintain the enable during the time delay as needed (with the NC contact for the electrical E-stop condition).

The MMC example applications located on the Applications CD (in the examples sub-directory) illustrate the recommended ladder logic for the E-stop handling of the M_CLOS1 application. Please refer to MMC2_EX.LDO for an example of M_CLOS1.

The machine start input must go through a positive transition (off to on) to reset the errors and close the loop.

The time at DELY is normally in the range from 500 ms to 2 sec.

On a positive transition of MSTR, this function will reset all E-stop, C-stop, and programming errors on axes 1 through 8.

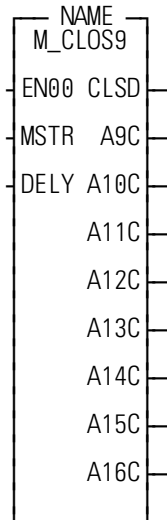
The positive transition of MSTR enables a timer with a preset time of DELY. After DELY has elapsed, the loops will be closed on axes 1 to 8. CLSD will be energized if one or more axes 1 to 8 have their position loops closed. The delay allows the drive some time to power up before it starts controlling the axis.

If an E-stop fault occurs on any of axis 1 to 8, its loop closed output (A1 to A8) will be dropped. CLSD is true as long as one or more of axes 1 to 8 have their position loops closed.

M_CLOS9

Close Loop on Servo Axes 9 to 16

USER/M_INIT



Inputs: EN00 (BOOL) - enables execution
MSTR (BOOL) - machine start input
DELY (TIME) - amount of time that will elapse after a positive transition of MSTR until the loops will be closed

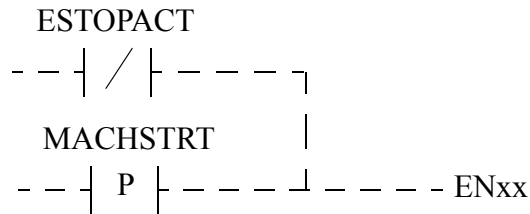
Outputs: CLSD (BOOL) - one or more of axis 9 to 16 have their position loops closed
A9C (BOOL) - set when the loop is closed on axis 9
A10C (BOOL) - set when the loop is closed on axis 10
A11C (BOOL) - set when the loop is closed on axis 11
A12C (BOOL) - set when the loop is closed on axis 12
A13C (BOOL) - set when the loop is closed on axis 13
A14C (BOOL) - set when the loop is closed on axis 14
A15C (BOOL) - set when the loop is closed on axis 15
A16C (BOOL) - set when the loop is closed on axis 16

```
<<INSTANCE NAME>>:M_CLOS9(EN00 := <<BOOL>>, MSTR :=  
<<BOOL>>, DELY := <<TIME>>, CLSD => <<BOOL>> A9C =>  
<<BOOL>>, A10C => <<BOOL>>, A11C => <<BOOL>>, A12C =>  
<<BOOL>>, A13C => <<BOOL>>, A14C => <<BOOL>>, A15C =>  
<<BOOL>>, A16C => <<BOOL>>);
```

This function block is used to reset the E-stop, C-stop, and programming errors on servo axes 9 through 16 when the machine start input is pulsed. It closes the loop on servo axes 9 through 16 after the machine start input is pulsed and a programmable time delay has elapsed. It drops the loop closed flag if an E-stop fault occurs.

This function block can be enabled every scan. If the enable input changes from ON to OFF during the time delay after machine start, the function block will abort the time delay and not close the position loops.

If there are conditions that should abort the sequence to close the position loops (such as an electrical E-stop condition during the time delay), then the enable should include both the positive transition of the machine start input and the current state of the electrical E-stop status as shown below.



The reason for these two input conditions is to provide the enable at the start of the time delay (with the P contact of the machine start signal) and to maintain the enable during the time delay as needed (with the NC contact for the electrical E-stop condition).

The MMC example applications located on the Applications CD (in the examples sub-directory) illustrate the recommended ladder logic for the E-stop handling of the M_CLOSx application. Please refer to MMC2_EX.LDO for an example of M_CLOS1 usage that applies to M_CLOS9 as well.

The machine start input must go through a positive transition (off to on) to reset the errors and close the loop.

The time at DELY is normally in the range from 500 ms to 2 sec.

On a positive transition of MSTR, this function will reset all E-stop, C-stop, and programming errors on axes 9 through 16.

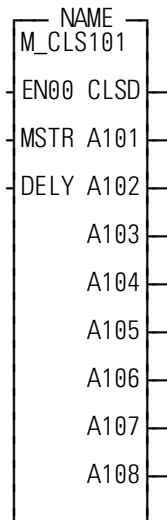
The positive transition of MSTR enables a timer with a preset time of DELY. After DELY has elapsed, the loops will be closed on axes 9 to 16. CLSD will be energized if one or more of axes 9 to 16 have their position loops closed. The delay allows the drive some time to power up before it starts controlling the axis.

If an E-stop fault occurs on any of axis 9 to 16, its loop closed output (A9 to A16) will be dropped. CLSD is true as long as one or more of axes 9 to 16 have their position loops closed.

M_CLS101

Close Loop on Servo Axes 101-108 (17th to 24th)

USER/M_INIT



Inputs: EN00 (BOOL) - enables execution
MSTR (BOOL) - machine start input
DELY (TIME) - amount of time that will elapse after a positive transition of MSTR until the loops will be closed

Outputs: CLSD (BOOL) - one or more of axes 101 to 108 have their position loops closed

A101 (BOOL) - set when the loop is closed on axis called 101 (the 17th defined axis)

A102 (BOOL) - set when the loop is closed on axis called 102 (the 18th defined axis)

A103 (BOOL) - set when the loop is closed on axis called 103 (the 19th defined axis)

A104 (BOOL) - set when the loop is closed on axis called 104 (the 20th defined axis)

A105 (BOOL) - set when the loop is closed on axis called 105 (the 21st defined axis)

A106 (BOOL) - set when the loop is closed on axis called 106 (the 22nd defined axis)

A107 (BOOL) - set when the loop is closed on axis called 107 (the 23rd defined axis)

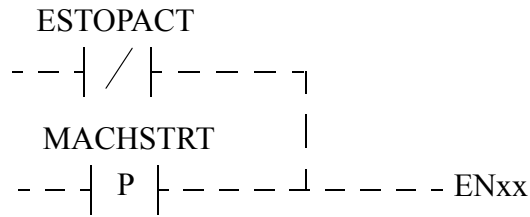
A108 (BOOL) - set when the loop is closed on axis called 108 (the 24th defined axis)

```
<<INSTANCE NAME>>:M_CLS101(EN00 := <<BOOL>>, MSTR :=  
<<BOOL>>, DELY := <<TIME>>, CLSD => <<BOOL>> A101 =>  
<<BOOL>>, A102 => <<BOOL>>, A103 => <<BOOL>>, A104 =>  
<<BOOL>>, A105 => <<BOOL>>, A106 => <<BOOL>>, A107 =>  
<<BOOL>>, A108 => <<BOOL>>);
```

This function block is used to reset the E-stop, C-stop, and programming errors on servo axes called 101 through 108 (the 17th to the 24th defined axes) when the machine start input is pulsed. It closes the loop on servo axes 101 through 108 after the machine start input is pulsed and a programmable time delay has elapsed. It drops the loop closed flag if an E-stop fault occurs.

This function block can be enabled every scan. If the enable input changes from ON to OFF during the time delay after machine start, the function block will abort the time delay and not close the position loops.

If there are conditions that should abort the sequence to close the position loops (such as an electrical E-stop condition during the time delay), then the enable should include both the positive transition of the machine start input and the current state of the electrical E-stop status as shown below.



The reason for these two input conditions is to provide the enable at the start of the time delay (with the P contact of the machine start signal) and to maintain the enable during the time delay as needed (with the NC contact for the electrical E-stop condition).

The MMC example applications located on the Applications CD (in the examples sub-directory) illustrate the recommended ladder logic for the E-stop handling of the M_CLOSx application. Please refer to MMC2_EX.LDO for an example of M_CLOS1 usage that applies to M_CLS101 as well.

The machine start input must go through a positive transition (off to on) to reset the errors and close the loop.

The time at DELY is normally in the range from 500 ms to 2 sec.

On a positive transition of MSTR, this function will reset all E-stop, C-stop, and programming errors on axes 101 to 108.

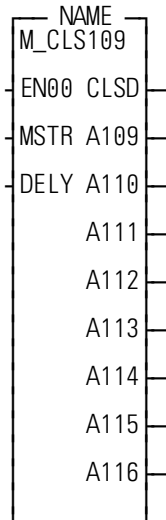
The positive transition of MSTR enables a timer with a preset time of DELY. After DELY has elapsed, the loops will be closed on axes 101 to 108. CLSD will be energized if one or more of axes 101 to 108 have their position loops closed. The delay allows the drive some time to power up before it starts controlling the axis.

If an E-stop fault occurs on any of axis 101 to 108, its loop closed output (A101 to A108) will be dropped. CLSD is true as long as one or more of axes 101 to 108 have their position loops closed.

M_CLS109

Close Loop on Servo Axes 109-116 (25th to 32nd)

USER/M_INIT



Inputs: EN00 (BOOL) - enables execution
MSTR (BOOL) - machine start input
DELY (TIME) - amount of time that will elapse after a positive transition of MSTR until the loops will be closed

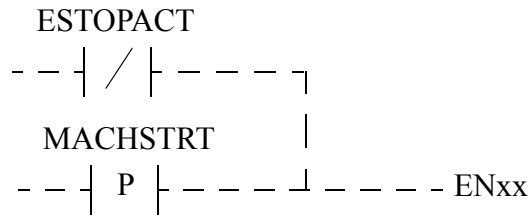
Outputs: CLSD (BOOL) - one or more of axes 109 to 116 have their position loops closed
A109 (BOOL) - set when the loop is closed on axis called 109 (the 25th defined axis)
A110 (BOOL) - set when the loop is closed on axis called 110 (the 26th defined axis)
A111 (BOOL) - set when the loop is closed on axis called 111 (the 27th defined axis)
A112 (BOOL) - set when the loop is closed on axis called 112 (the 28th defined axis)
A113 (BOOL) - set when the loop is closed on axis called 113 (the 29th defined axis)
A114 (BOOL) - set when the loop is closed on axis called 114 (the 30th defined axis)
A115 (BOOL) - set when the loop is closed on axis called 115 (the 31st defined axis)
A116 (BOOL) - set when the loop is closed on axis called 116 (the 32nd defined axis)

```
<<INSTANCE NAME>>:M_CLOS9(EN00 := <<BOOL>>, MSTR :=  
  <<BOOL>>, DELY := <<TIME>>, CLSD => <<BOOL>> A9C =>  
  <<BOOL>>, A10C => <<BOOL>>, A11C => <<BOOL>>, A12C =>  
  <<BOOL>>, A13C => <<BOOL>>, A14C => <<BOOL>>, A15C =>  
  <<BOOL>>, A16C => <<BOOL>>);
```

This function block is used to reset the E-stop, C-stop, and programming errors on servo axes called 109 through 116 (the 25th to the 32nd defined axes) when the machine start input is pulsed. It closes the loop on servo axes 109 through 116 after the machine start input is pulsed and a programmable time delay has elapsed. It drops the loop closed flag if an E-stop fault occurs.

This function block can be enabled every scan. If the enable input changes from ON to OFF during the time delay after machine start, the function block will abort the time delay and not close the position loops.

If there are conditions that should abort the sequence to close the position loops (such as an electrical E-stop condition during the time delay), then the enable should include both the positive transition of the machine start input and the current state of the electrical E-stop status as shown below.



The reason for these two input conditions is to provide the enable at the start of the time delay (with the P contact of the machine start signal) and to maintain the enable during the time delay as needed (with the NC contact for the electrical E-stop condition).

The MMC example applications located on the Applications CD (in the examples sub-directory) illustrate the recommended ladder logic for the E-stop handling of the M_CLOSx application. Please refer to MMC2_EX.LDO for an example of M_CLOS1 usage that applies to M_CLS109 as well.

The machine start input must go through a positive transition (off to on) to reset the errors and close the loop.

The time at DELY is normally in the range from 500 ms to 2 sec.

On a positive transition of MSTR, this function will reset all E-stop, C-stop, and programming errors on axes 109 to 116.

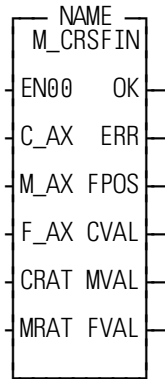
The positive transition of MSTR enables a timer with a preset time of DELY. After DELY has elapsed, the loops will be closed on axes 109 to 116. CLSD will be energized if one or more of axes 109 to 116 have their position loops closed. The delay allows the drive some time to power up before it starts controlling the axis.

If an E-stop fault occurs on any of axis 109 to 116, its loop closed output (A109 to A116) will be dropped. CLSD is true as long as one or more of axes 109 to 116 have their position loops closed.

M_CRSFIN

Coarse, Medium and Fine Resolver

USER/M_REF



Inputs: EN00 (BOOL) - enables execution
C_AX (USINT) - coarse resolver axis number
M_AX (USINT) - medium resolver axis number
F_AX (USINT) - fine resolver axis number
CRAT (DINT) - coarse to medium or coarse to fine ratio
MRAT (DINT) - medium to fine ratio

Outputs: OK (BOOL) - execution complete without error
ERR (INT) - error number
FPOS (DINT) - position that the fine axis has been part referenced to
CVAL (DINT) - value read from the coarse resolver
MVAL (DINT) - value read from the medium resolver
FVAL (BOOL) - value read from the fine resolver

```
<<INSTANCE NAME>>:M_CLS101(EN00 := <<BOOL>>, C_AX :=  
  <<USINT>>, M_AX := <<USINT>>, F_AX:= <<USINT>>, CRAT :=  
  <<DINT>>, MRAT := <<DINT>> OK => <<BOOL>> ERR => <<INT>>,  
  FPOS => <<DINT>>, CVAL => <<DINT>>, MVAL => <<DINT>>, FVAL =>  
  <<BOOL>>);
```

This function block reads coarse, medium, and fine resolvers and then part references the fine axis to the value calculated by using coarse, medium and fine. Three separate combinations of resolvers can be used: coarse, medium, and fine; coarse and fine; or medium and fine.

This function block should be one-shot after the axes have been initialized by the user's servo setup function.

The value entered at C_AX is the axis number for the coarse resolver, or 0 if you are not using a coarse resolver.

The value entered at M_AX is the axis number for the medium resolver, or 0 if you are not using a medium resolver.

The value entered at F_AX is the axis number for the fine resolver. This is also the axis that will be part referenced by this function block.

The value entered at CRAT is the coarse to medium ratio if coarse, medium and fine resolvers are being used, or the coarse to fine ratio if only coarse and fine resolvers are being used.

The value entered at MRAT is the medium to fine ratio.

The OK output indicates execution complete without error. If the OK output is not set, then an error has occurred and the error code will be stored in the ERR output. A listing of possible errors is shown below:

ERR	Description
0	No error
1	The OK from the READ_SV function for the fine axis was not set
2	The OK from the READ_SV function for the medium axis was not set
3	The OK from the READ_SV function for the coarse axis was not set
4	M_AX and C_AX inputs are both zero
5	M_AX was non-zero, but MRAT was zero
6	C_AX was non-zero, but CRAT was zero
7	The fine axis is moving or drifting
8	The medium axis is moving or drifting
9	The coarse axis is moving or drifting
10	The fine axis position was not between 0 and 3999
11	The medium axis position was not between 0 and 3999
12	The coarse axis position was not between 0 and 3999
13	An error occurred in the calculations for coarse, medium and fine
14	An error occurred in the calculations for coarse and fine
15	An error occurred in the calculations for medium and fine
16	The OK from the part reference function for F_AX did not get set

The FPOS output will show the final value that the fine axis has been part reference to.

The CVAL output will show the value read from the coarse resolver.

The MVAL output will show the value read from the medium resolver.

The FVAL output will show the value read from the fine resolver.

M_DATCAP

Captures Axis Information

USER/M_DATA

NAME
M_DATCAP
EN00 IDNE
INIT IERR
SRCE ELEM
QTY CDNE
SIZE SNDE
STRT SFAL
ONCE SERR
SEND
RDSK
SDIR
FILE

Inputs: EN00 (BOOL) - enables execution
INIT (BOOL) - initializes data capture memory area
SRCE (STRUCT(0..7)) - defines axis number and variable number to capture
QTY (USINT) - defines the number of variables to capture (This is the same as the number of elements used in the SRCE array.)
SIZE (UINT) - defines the number of samples to be captured
STRT (BOOL) - starts data capture
ONCE (BOOL) - set to capture data once; reset to capture data continuously
SEND (BOOL) - starts save of captured data to RAMDISK or workstation
RDSK (BOOL) - set if data will be saved to the RAMDISK or reset if data will be saved to the workstation
SDIR (STRING) - the subdirectory on the workstation or RAMDISK to send the data to (an eight character maximum)
FILE (STRING) - the file name that the data will be saved as (a 12 character maximum)

Outputs: IDNE (BOOL) - initialization complete without error
IERR (USINT) - error number that occurred during initialization
ELEM (UINT) - the element number currently being captured
CDNE (BOOL) - capture done
SDNE (BOOL) - file send done
SFAL (BOOL) - file send failed
SERR (INT) - error number that occurred during file send

```

<<INSTANCE NAME>>:M_DATCAP (EN00 := <<BOOL>>, INIT :=
  <<BOOL>>, SRCE := <<MEMORY AREA>>, QTY:= <<USINT>>, SIZE :=
  <<UINT>>, STRT := <<BOOL>>, ONCE := <<BOOL>> SEND :=
  <<BOOL>>, RDSK := <<BOOL>>, SDIR := <<STRING>>, FILE :=
  <<STRING>>, IDNE => <<BOOL>>, IERR => <<USINT>>, ELEM =>
  <<UINT>>, CDNE => <<BOOL>>, SDNE => <<BOOL>>, SFAL =>
  <<BOOL>>, SERR => <<INT>>);

```

This function block is considered obsolete. It requires the CAP2ASC.EXE DOS utility to extract the data captured. The M_DATCPT function block performs the same data capture operations as M_DATCAP with the same function block inputs and outputs except M_DATCPT creates an output file that is already in a directly viewable ASCII text format (it is a tab-delimited variable format).

This function block captures axis information on an interrupt basis and stores it in a structure. The structure can then be written out to a binary file on the RAMDISK or the workstation. In order to manipulate the data, convert this binary file to an ASCII text file using the CAP2ASC.exe which is included with the Motion ASFB examples. On your PC, type:

CAP2ASC filename

where filename is the name you assigned to the binary file. You can then view and/or edit this ASCII file using a text editing program or import it into a spreadsheet.

The EN00 input of this function block should be set every scan.

On a positive transition of the INIT input, the values entered at the SRCE input are examined. The SRCE input is an array of structures and must have the following members:

Name	Data Type	Definition
SRCE	STRUCT(0..7)	Defines axis and variables to capture
.AXIS	USINT	Defines the axis to capture data for
.VAR	USINT	Defines the variable to capture

The SRCE(X).VAR input must be one of the following values:

SRCE(X).VAR Definition

- | | |
|----|---|
| 1 | Actual Position (variable 1 of READ_SV)* |
| 2 | Fast input occurred |
| 3 | Commanded position (variable 3 of READ_SV)* |
| 4 | Position error (variable 4 of READ_SV)* |
| 5 | Filter error (variable 5 of READ_SV)* |
| 6 | Command change (variable 6 of READ_SV)* |
| 7 | Position change (variable 7 of READ_SV)* |
| 8 | Feedback position (variable 8 of READ_SV)* |
| 9 | Prefilter commanded |
| 10 | Prefilter command change |
| 11 | Remaining master offset |
| 12 | Remaining slave offset |

* The variables in the READ_SV function are reported in ladder units (LU). The variables in DATCAP function block are reported in feedback units (FU).

If an error is found at the SRCE input, then IDNE will not be set and IERR will hold a number describing the error that occurred. If no errors are found at the SRCE input, then IDNE will be set. A listing of possible errors at IERR are shown below:

IERR Description

- | | |
|---|--|
| 0 | No error |
| 1 | The function block has not stopped capturing data from a previous data capture initialization. |
| 2 | An axis number in the structure is invalid |
| 3 | The limit of eight variables in the array of structures has been exceeded. |
| 4 | Parameter number in the structure is out of range. |
| 5 | The initialization was done before the STRTSERV function was called. |
| 6 | Reserved |
| 7 | Reserved |
| 8 | Reserved |
| 9 | The total number of bytes to capture exceeds 7992. |

The QTY input defines the number of variables that will be captured. This is the same as the number of array elements used in the SRCE input.

The SIZE input defines the number of samples to capture.

When the STRT input is on, if ONCE is also on, the data will be captured once. When the STRT input is on, if ONCE is off, then the data will be captured continuously until the STRT input drops.

While the data is being captured, the ELEM output will show the current element number being captured. When data capture is complete, the CDNE output will be set.

Once the data has been captured, it can be sent to a file on the RAMDISK or workstation. The data will be sent when the SEND input is pulsed. If the RDSK input is ON when the SEND input is pulsed, then the captured data will be sent to the PiC900 RAMDISK. If the RDSK input is OFF when the SEND input is pulsed, then the captured data will be sent to the workstation C: drive.

The file will be saved with the name entered at FILE. This must be of the form FILENAME.EXT.

The SDIR input defines the subdirectory where the file will be located. The subdirectory must not exceed eight characters.

When the file has been successfully sent, the SDNE output will be set. If an error occurred in writing the file, then SFAL will be set and SERR will contain a number describing the error that occurred. A list of errors is shown below:

SERR	Description
0	No error
1 to 99	Error occurred on file open
101 to 199	Error occurred on file write
201 to 299	Error occurred on file write
301 to 399	Error occurred on file write
401 to 499	Error occurred on file write
501 to 599	Error occurred on file close

M_DATCPT

Capture Axis data to file

USER/M_DATA

NAME	
M_DATCPT	
EN00	IDNE
INIT	IERR
SRCE	ELEM
QTY	CDNE
SIZE	SDNE
STRT	SFAL
ONCE	SERR
SEND	
RDSK	
SDIR	
FILE	

Inputs: EN00 (BOOL) - enables execution
INIT (BOOL) - initializes data capture memory area
SRCE (STRUCT(0..7)) - defines axis number and variable number for each item to capture
QTY (USINT) - number of variables to capture (This is the same as the number of elements used in the SRCE array.)
SIZE (UINT) - number of samples to be captured
STRT (BOOL) - starts the data capture
ONCE (BOOL) - set to capture data once; reset to capture data continuously
SEND (BOOL) - starts save of captured data to specified file
RDSK (BOOL) - set if data will be saved to the RAMDISK or reset if data will be saved to the PC hard disk.
SDIR (STRING) - name of subdirectory (an eight character maximum)
FILE (STRING) - the file name that the data will be saved as (8.3 format)

Outputs: IDNE (BOOL) - initialization complete without error
IERR (USINT) - initialization error number
ELEM (UINT) - the element number currently being captured
CDNE (BOOL) - capture done
SDNE (BOOL) - file send done
SFAL (BOOL) - file send failed
SERR (INT) - error number that occurred during file send

```

<<INSTANCE NAME>>:M_DATCPT (EN00 := <<BOOL>>, INIT :=
  <<BOOL>>, SRCE := <<MEMORY AREA>>, QTY:= <<USINT>>, SIZE :=
  <<UINT>>, STRT := <<BOOL>>, ONCE := <<BOOL>> SEND :=
  <<BOOL>>, RDSK := <<BOOL>>, SDIR := <<STRING>>, FILE :=
  <<STRING>>, IDNE => <<BOOL>>, IERR => <<USINT>>, ELEM =>
  <<UINT>>, CDNE => <<BOOL>>, SDNE => <<BOOL>>, SFAL =>
  <<BOOL>>, SERR => <<INT>>);

```

This function block captures axis information on an interrupt basis and stores it in a structure. The structure can then be written out to a text file on the RAMDISK or the workstation. This text file is directly viewable with any text editor. It is also tab delimited so its possible to import it into some spreadsheet applications. This function block provides simpler control of CAPTINIT and CAPSTAT. Both of these standard functions are documented in the PiCPro Function Block Reference Guide and in on-line Help.

The EN00 input of this function block should be set every scan.

On a positive transition of the INIT input, the values entered at the SRCE input are examined. The SRCE input is an array of structures and must have the following members:

Name	Data Type	Definition
SRCE	STRUCT(0..7)	Defines axis and variables to capture
.AXIS	USINT	Defines the axis to capture data for
.VAR	USINT	Defines the variable to capture

The SRCE(X).VAR input must be one of the following values:

SRCE(X).VAR	Definition
1	Actual Position (variable 1 of READ_SV)*
2	Fast input occurred
3	Commanded position (variable 3 of READ_SV)*
4	Position error (variable 4 of READ_SV)*
5	Filter error (variable 5 of READ_SV)*
6	Command change (variable 6 of READ_SV)*
7	Position change (variable 7 of READ_SV)*
8	Feedback position (variable 8 of READ_SV)*
9	Prefilter commanded position
10	Prefilter command change
11	Remaining master offset
12	Remaining slave offset

- 13 Command change (variable 6 of READ_SV)
- 14 Position change (variable 7 of READ_SV)
- 15 Prefilter command change

* The variables in the READ_SV function are reported in ladder units (LU). The variables in DATCAP function block are reported in feedback units (FU).

If an error is found at the SRCE input, then IDNE will not be set and IERR will hold a number describing the error that occurred. If no errors are found at the SRCE input, then IDNE will be set. A listing of possible errors at IERR are shown below:

IERR Description

- 0 No error
- 1 The function block has not stopped capturing data from a previous data capture initialization.
- 2 An axis number in the structure is invalid
- 3 The limit of eight variables in the array of structures has been exceeded.
- 4 Parameter number in the structure is out of range.
- 5 The initialization was done before the STRTSERV function was called.
- 6 Reserved
- 7 Reserved
- 8 Reserved
- 9 The total number of bytes to capture exceeds 7992.

The QTY input defines the number of variables that will be captured. This is the same as the number of array elements used in the SRCE input.

The SIZE input defines the number of samples to capture.

When the STRT input is on, if ONCE is also on, the data will be captured once. When the STRT input is on, if ONCE is off, then the data will be captured continuously until the STRT input drops.

While the data is being captured, the ELEM output will show the current element number being captured. When data capture is complete, the CDNE output will be set.

Once the data has been captured, it can be sent to a file on the RAMDISK or workstation. The data will be sent when the SEND input is pulsed. If the RDSK input is ON when the SEND input is pulsed, then the captured data will be sent to the PiC900 RAMDISK. If the RDSK input is OFF when the SEND input is pulsed, then the captured data will be sent to the workstation C: drive.

The file will be saved with the name entered at FILE. This must be of the form FILENAME.EXT.

The SDIR input defines the subdirectory where the file will be located. The subdirectory must not exceed eight characters.

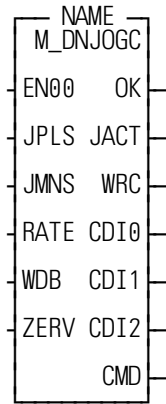
When the file has been successfully sent, the SDNE output will be set. If an error occurred in writing the file, then SFAL will be set and SERR will contain a number describing the error that occurred. A list of errors is shown below:

SERR	Description
0	No error
1 to 99	Error occurred on file open. (See Appendix B in the online help)
101 to 199	Error occurred on file write
201 to 299	Error occurred on file close

M_DNJOGC

Jog DeviceNet Axis

USER/M_DEVNET



Inputs: EN00 (BOOL) - enables execution
JPLS (BOOL) - jog in the PLUS direction (CW)
JMNS (BOOL) - jog in the MINUS direction (CCW)
RATE (DINT) - rate or velocity (programmed as RPM * 65536)
WDB (BOOL) - DeviceNet write data busy flag
ZERV (BOOL) - axis zero velocity - axis has stopped

Outputs: OK (BOOL) - function block is active
JACT (BOOL) - axis jog is active
WRC (BOOL) - write data/command to the drive
CDI0 (BOOL) - command data index - bit 0
CDI1 (BOOL) - command data index - bit 1
CDI2 (BOOL) - command data index - bit 2
CMD (DWORD) - command data value

```
<<INSTANCE NAME>>:M_DNJOGC (EN00 := <<BOOL>>, JPLS :=  
  <<BOOL>>, JMNS := <<BOOL>>, RATE:= <<DINT>>, WDB :=  
  <<BOOL>>, ZERV := <<BOOL>>, OK => <<BOOL>>, JACT =>  
  <<BOOL>>, WRC => <<BOOL>>, CDI0 => <<BOOL>>, CDI1 =>  
  <<BOOL>>, CDI2 => <<BOOL>>, CMD => <<DWORD>>);
```

This function block is used to allow a manual jog (move at a velocity) of a Centurion DeviceNet Drive axis.

Before this function block can be used, the axis must be enabled and placed into servo lock.

If the enable is active, triggering job plus (JPLS) or jog minus (JMNS) input will cause the specified DeviceNet axis to move at the indicated rate in the corresponding direction. When the input is deactivated, motion will stop.

This function block should be used only to allow an operator to manually move an axis on a machine. It is not designed for any other purpose.

Important - If the enable is disabled while a move is underway the axis will continue to move until the jog switch is deactivated.

The JPLS input enables a move in the positive direction for the selected axis. The JMNS input enables a move in the negative direction for the selected axis.

Rate is programmed in $\text{RPM} * 65,536$. An example: for 100 RPM, Rate = 6553600. If both the JPLS and the JMNS inputs are set; motion will stop until both inputs are dropped and one is again selected.

M_DNPOSC

Move DN Axis to Position

USER/M_DEVNET

NAME	
M_DNPOSC	
EN00	OK
STRT	STAT
RATE	STRI
POS	WRC
ABSO	CDI0
FDBK	CDI1
WDB	CDI2
ZERV	CDI3
INPO	CDI4
HOME	CMD

Inputs: EN00 (BOOL) - enables execution
STRT (BOOL) - start the axis move
RATE (DINT) - rate or velocity (programmed as RPM * 65536)
POS (DINT) - command position in FU
ABSO (BOOL) - absolute or incremental position (set indicates absolute)
FDBK (DWORD) - actual position (feedback) from the drive
WDB (BOOL) - DeviceNet write data busy flag
ZERV (BOOL) - axis is at zero velocity - axis has stopped
INPO (BOOL) - axis is in position - axis is at its commanded position
HOME (BOOL) - axis is homed

Outputs: OK (BOOL) - function block is active
STAT (INT) - axis status value
STRI (BOOL) - start move indicator
WRC (BOOL) - write data/command to the drive
CDI0 (BOOL) - command data index - bit 0
CDI1 (BOOL) - command data index - bit 1
CDI2 (BOOL) - command data index - bit 2
CDI3 (BOOL) - command data index - bit 3
CDI4 (BOOL) - command data index - bit 4 (not used)
CMD (DWORD) - command data value

```
<<INSTANCE NAME>>:M_DNPOSC (EN00 := <<BOOL>>, STRT :=  
<<BOOL>>, RATE := <<DINT>>, POS := <<DINT>>, ABSO := <<BOOL>>,  
ABSO := <<BOOL>>, FDBK := <<DWORD>>, WDB := <<BOOL>>, ZERV  
:= <<BOOL>>, INPPO := <<BOOL>>, HOME := <<BOOL>>, OK =>  
<<BOOL>>, STAT => <<INT>>, STRI => <<BOOL>>, WRC => <<BOOL>>,  
CDI0 => <<BOOL>>, CDI1 => <<BOOL>>, CDI2 => <<BOOL>>; CDI3 =>  
<<BOOL>>, CDI4 => <<BOOL>>, CMD => <<BOOL>>);
```

This function block is used to allow a position / index move with a Centurion DeviceNet Drive axis.

Before this function block can be used, the axis must be enabled and placed into servo lock.

If the enable is active, triggering (STRT) input will cause the specified DeviceNet axis to move at the indicated rate to the position endpoint (POS). The axis will travel an incremental distance if the ABSO input is deactivated. The axis will travel to an absolute position if the ABSO input is activated.

Important - If the enable is disabled while a move is underway, the axis will continue to move until it has reached its endpoint.

The Position command (POS) is entered in feedback counts. (Example: for an 8000 counts/rev encoder and an incremental move, Position = 16000 will result in a move of 2 revolutions).

Rate is programmed in RPM * 65,536. For example, for 100 RPM, Rate = 6553600.

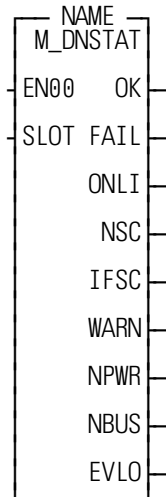
The axis status (STAT) will indicate the status of the axis based on the following code:

- 1 = Axis is Positioning
- 2 = Absolute mode: the command is equal to current position
- 3 = Incremental mode: the command is equal to zero
- 4 = Rate is equal to zero
- 5 = Absolute mode and axis is not Homed

M_DNSTAT

DeviceNet Module Status

USER/M_DEVNET



Inputs: EN00 (BOOL) - enables execution
SLOT (USINT) - slot number for the DeviceNet module

Outputs: OK (BOOL) - execution complete
FAIL (BOOL) - failure getting the DeviceNet status
ONLI (BOOL) - DeviceNet module is online
NSC (BYTE) - DeviceNet Network Status Code
IFSC (BYTE) - DeviceNet Interface Status Code
WARN (BOOL) - DeviceNet communication error warning
NPWR (BOOL) - No DeviceNet bus power
NBUS (BOOL) - No DeviceNet bus connection
EVLO (BOOL) - DeviceNet event was lost due to full event queue

```
<<INSTANCE NAME>>:M_DNSTAT (EN00 := <<BOOL>>, SLOT :=  
<<USINT>>, OK => <<BOOL>>, FAIL => <<BOOL>>, ONLI =>  
<<BOOL>>, NSC => <<BYTE>>, IFSC => <<BYTE>>, WARN =>  
<<BOOL>>, NPWR => <<BOOL>>, NBUS => <<BOOL>>, EVLO =>);
```

This function block obtains the DeviceNet network and interface status conditions. Those conditions are presented in outputs as bytes and booleans.

ONLI is set if the DeviceNet module is communicating with nodes.

NSC is the status of the DeviceNet module network interface.

0 = network interface is offline.

1 = network interface is offline due to a network fault.

2 = network interface is offline due to a configuration fault.

3 = network interface is online and no faults are detected.

4 = network interface is online but one or more network services have failed.

5 = network interface is online and is exchanging data; no faults are detected.

6 = network interface is online and is exchanging data; one or more network services is receiving an idle indication; no faults are detected.

7 = network interface is online but one or more previously active network services have been suspended; no faults are detected.

IFC is the status of the DeviceNet module data exchange interface.

0 = data exchange interface is closed.

1 = data exchange interface is open

2 = data exchange interface is faulted due to a "heartbeat" timeout.

WARN is set when the communication warning threshold has been exceeded.

NPWR is set when DeviceNet bus power is not present.

NBUS is set when DeviceNet bus is not connected.

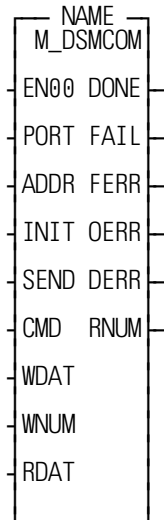
EVLO is set when an event was lost due to a full event queue in the DeviceNet module. This flag is cleared when the DeviceNet interface is closed (FB_CLS).

For more information regarding how this information is gathered or the meaning of any of the outputs, consult the FB_STA function description.

M_DSMCOM

Centurion DSM Serial Communication

USER/M_DRVCOM



Inputs: EN00 (BOOL) - enables execution
PORT (STRING) - identifies the serial communication port
ADDR (USINT) - identifies the Centurion servo drive address
INIT (BOOL) - (one-shot) initializes M_DSMCOM
SEND (BOOL) - (one-shot) executes read or write command
CMD (UINT) - command to execute
WDAT (memory area) source of data for the write command
memory area is a STRING, ARRAY, or STRUCTURE
WNUM (USINT) - number of bytes of data in WDAT
RDAT (memory area) - destination of data returned by the read command
memory area is a STRING, ARRAY, or STRUCTURE

Outputs: DONE (BOOL) - command executed without error
FAIL (BOOL) - command encountered an error
FERR (UINT) - PiC format error number
OERR (UINT) - operation error number
DERR (UINT) - Centurion drive error number
RNUM (USINT) - number of bytes of data in RDAT

```
<<INSTANCE NAME>>:M_DSMCOM (EN00 := <<BOOL>>, PORT :=  
<<STRING>>, ADDR := <<USINT>>, INIT := <<BOOL>>, SEND :=  
<<BOOL>>, CMD := <<UINT>>, WDAT := <<MEMORY AREA>>, DONE  
=> <<BOOL>>, FAIL => <<BOOL>>, FERR => <<UINT>>, DERR =>  
<<UINT>>, RNUM => <<USINT>>);
```

The M_DSMCOM function block allows the PiC to interface with from 1 to 32 Centurion DSM100 servo drives via RS232 or RS422/RS485 serial communication links. With this function block, various drive parameters can be read and written. These parameters are listed in Appendix A.

Inputs

The EN00 input of this function block should be set every scan.

The PORT input identifies the serial communication port. If the PiC user port is used, the reserved name USER:\$00 is entered. If a serial communication module is used, the name assigned to the port by the ASSIGN function block should be entered. The string can be no longer than 10 characters, with up to eight characters for the name followed by a ":" and the null character"\$00".

The ADDR input identifies the Centurion servo drive address. The drive address is set using the sixteen position rotary addressing switch on the drive or via software using DSMPro. The range is 0 to 32.

The INIT input initializes the M_DSMCOM function block. The DONE output will be set when the initialization has successfully completed. This initialization must be executed before a read or write is executed.

The SEND input executes a read or write command.

To execute a read command:

1. Move the command number into the CMD input.
2. One-shot the SEND input.

When the DONE output goes high:

- RDATA will hold the data read.
- RNUM will hold the number of bytes of data read.

To execute a write command:

1. Move the command number into the CMD input.
2. Move the data to write into the WDATA input.
3. Move the number of bytes of data into the WNUM input.
4. One-shot the SEND input.

When the DONE output goes high, the command is complete.

NOTE: Never send a new command until any previous command or initialization has completed. Completion is indicated by the DONE (or FAIL) output going high.

The CMD input specifies which read or write command to execute. See Appendix A for a list of all the available commands.

The WDAT input is the data to be written to the drive. The type and number of data depends on the write command being executed. There are two ways to handle the data to this input:

1. If your application will only be writing one specific command or different commands that are all the same data type, use a structure whose member(s) is/are the correct data type(s) to be sent.
For example, the write command 0DDH Analog Output Write Value expects an unsigned byte value followed by a signed word value. With this command, you could enter a structure at the WDAT input whose first member is an USINT and whose second member is an INT.
2. If your application will be writing different commands that are different data types, use a structure with one member that is the largest data type and use the PiCPro datatype conversion functions to convert any data to the data type of the structure member before sending the data.

The WNUM input is the number of bytes of data in WDAT.

The RDAT input is the data read from the drive. Following the successful completion of a read command, the memory area pointed to by the RDAT input holds the data read from the drive. The RNUM output will indicate the number of bytes of data read. The type and number of data depends on the read command being executed. Again, there are two ways to handle this data.

1. If your application will only be reading one specific command or different commands that are all the same data type, use a structure whose member(s) is/are the correct data type(s) to be sent.
For example, the read command 042H Gear Ratio reads two signed word values. With this command, you could enter a structure at the RDAT input with two INT members.
2. If your application will be reading different commands that are different data types, use a structure with one member that represents the largest data type and use the PiCPro conversion functions to convert any data to its correct data type after reading it.

Outputs

The DONE output will be set if the initialization or a read or write command is completed successfully. The FAIL output will be set if an error occurs during the execution of the initialization or a read or write command.

The FERR output will identify errors encountered by the M_DSMCOM function block when using the PiC serial communications function blocks. The OERR output will identify errors detected when a read or write command is executed. They are described below.

OERR Description

0 No error

- 1 Checksum error - invalid checksum in the drive response
- 2 Timeout error - drive did not respond in time
- 3 Read or write attempted before initialization
- 4 Invalid PORT name
- 5 CMD input out of range
- 6 ADDR input out of range
- 7 WNUM input out of range
- 8 Invalid address in drive response
- 9 Invalid function in drive response
- 10 Invalid data in drive response
- 11 Invalid drive response

The DERR output will identify errors reported by the Centurion drive in a response to a command. They are described below.

DERR Description

- 0 No error
- 1 Invalid data
- 2 Command not enabled
- 3 EEPROM write error
- 4 Data accepted after limiting to minimum
- 5 Data accepted after limiting to maximum
- 6 Command disabled when drive is enabled
- 7 Flash programming error
- 8 Invalid function code
- 9 Command disabled when drive is disabled

The RNUM output indicates the number of bytes of data in RDAT after a read command has executed.

Application Notes

1. The M_DSMCOM function block must only be entered in the LDO once for each serial port being used.
2. A read or write command must not be attempted until the function block initialization is complete.
3. A read or write command must not be attempted until a previous read or write command is complete.
4. If no data is being sent with a command (which is the normal mode for most read commands), the WNUM input must be zero.

Connections

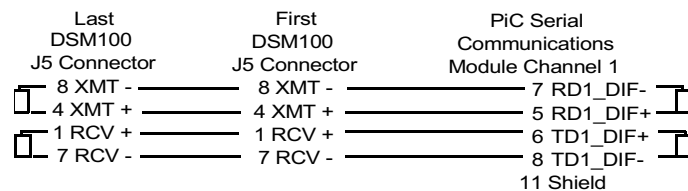
RS232 Connections


In single drive applications where the communications link is less than 50 feet, a three wire RS232 serial communication link may be used. The pinout is shown below.

Drive J5 Serial Port 9-pin D Connector	PiC User Port 10-pin Screw Terminal Connector
2 RCV	10 TD
3 XMT	9 RD
5 COM	8 GRD

RS422/RS485 Connections

Typically, the M_DSMCOM function block will be used with RS422/RS485 serial communication. RS422/RS485 provides superior noise immunity, allows communication links greater than 50 feet, and allows multiple drive connections to one PiC. A four wire daisy chain connection is made between a PiC Serial Communications Module and the DSM100 drives.



 Represents 100 ohm resistors which must be installed at each end of the daisy chain connection.

Example LDO with the M_DSMCOM Function Block

Please refer to the example ASFB M_DSM_EX.LDO ladder.

M_DW2BOO

Convert DWORD to BOOLs

USER/M_COMMON

NAME	
M_DW2BOO	
EN00	OK
IN	00
	01
	02
	03
	04
	05
	06
	07
	08
	09
	010
	011
	012
	013
	014
	015
	016
	017
	018
	019
	020
	021
	022
	023
	024
	025
	026
	027
	028
	029
	030
	031

Inputs: EN01 (BOOL) - enables execution
IN (DWORD) - the data to convert

Outputs: OK (BOOL) - execution complete
O0 (BOOL) - bit 0 of IN (least significant bit of IN)
O1 (BOOL) - bit 1 of IN
O2 (BOOL) - bit 2 of IN
O3 (BOOL) - bit 3 of IN
O4 (BOOL) - bit 4 of IN
O5 (BOOL) - bit 5 of IN
O6 (BOOL) - bit 6 of IN
O7 (BOOL) - bit 7 of IN
O8 (BOOL) - bit 8 of IN
O9 (BOOL) - bit 9 of IN
O10 (BOOL) - bit 10 of IN
O11 (BOOL) - bit 11 of IN
O12 (BOOL) - bit 12 of IN
O13 (BOOL) - bit 13 of IN
O14 (BOOL) - bit 14 of IN
O14 (BOOL) - bit 15 of IN
O16 (BOOL) - bit 16 of IN
O17 (BOOL) - bit 17 of IN
O18 (BOOL) - bit 18 of IN
O19 (BOOL) - bit 19 of IN
O20 (BOOL) - bit 20 of IN
O21 (BOOL) - bit 21 of IN
O22 (BOOL) - bit 22 of IN
O23 (BOOL) - bit 23 of IN
O24 (BOOL) - bit 24 of IN
O25 (BOOL) - bit 25 of IN
O26 (BOOL) - bit 26 of IN
O27 (BOOL) - bit 27 of IN
O28 (BOOL) - bit 28 of IN
O29 (BOOL) - bit 29 of IN
O30 (BOOL) - bit 30 of IN
O31 (BOOL) - bit 31 of IN (most significant bit)

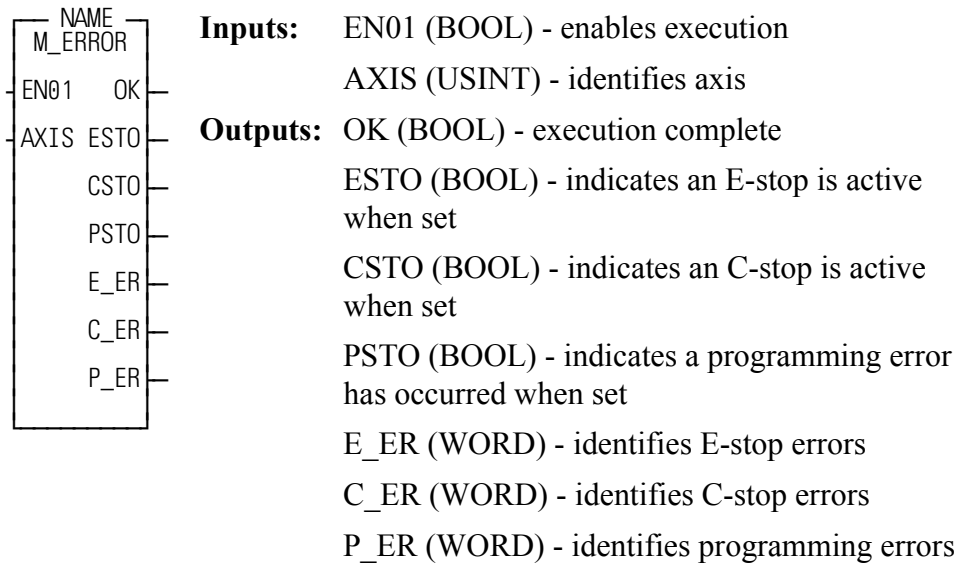
```
<<INSTANCE NAME>>:M_DW2BOO(EN00 := <<BOOL>>, IN :=  
<<DWORD>>, O0 => <<BOOL>> O1 => <<BOOL>>, O2 => <<BOOL>>, O3  
=> <<BOOL>>, O4 => <<BOOL>>, O5 => <<BOOL>>, O6 => <<BOOL>>,  
O7 => <<BOOL>>, O8 => <<BOOL>>, O8 => <<BOOL>> O9 =>  
<<BOOL>>, O10 => <<BOOL>>, O11 => <<BOOL>>, O12 => <<BOOL>>,  
O13 => <<BOOL>>, O14 => <<BOOL>>, O15 => <<BOOL>>, O16 =>  
<<BOOL>>, O17 => <<BOOL>> O18 => <<BOOL>>, O19 => <<BOOL>>,  
O20 => <<BOOL>>, O21 => <<BOOL>>, O22 => <<BOOL>>, O23 =>  
<<BOOL>>, O24 => <<BOOL>>, O25 => <<BOOL>>, O26 => <<BOOL>>  
O27 => <<BOOL>>, O28 => <<BOOL>>, O29 => <<BOOL>>, O30 =>  
<<BOOL>>, O31 => <<BOOL>>);
```

This function block converts a DWORD to 32 BOOLS.

M_ERROR

Axis Error Checking

USER/M_DATA



```
<<INSTANCE NAME>>:M_ERROR(EN01 := <<BOOL>>, AXIS :=  
<<USINT>>, OK => <<BOOL>> ESTO => <<BOOL>>, CSTO =>  
<<BOOL>>, PSTO => <<BOOL>>, E_ER => <<WORD>>, C_ER =>  
<<WORD>>, P_ER => <<WORD>>);
```

This function block is used to report **servo** E-stop, C-stop and programming error conditions in the ladder. These conditions may be caused by the servo software or defined by the programmer. If defined by the programmer they will be triggered using the E-STOP or C_STOP functions. All of these errors for the defined axis are reported in one location

The enable input of this function should be directly connected to the rail with a wire, causing this function block to be executed each scan.

The boolean outputs can be used as flags in the ladder to report error conditions.

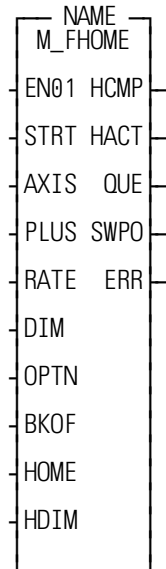
The word outputs can be converted to a HEX display by using the Module Monitor Edit View List command and inserting the variables. An option will be given on the format to display them. The variable's value during animation will be displayed in HEX format if the variable provided has 16#0 for its initial value. The default format during animation is decimal.

After monitoring them in HEX, referring to the tables in the manual of functions E_ERRORS, C_ERRORS and P_ERRORS will help identify the exact problem.

M_FHOME

Performs a Home Cycle using a Fast Reference

USER/M_REF



Inputs: EN01 (BOOL) - enables execution
STRT (BOOL) - enables the home cycle
AXIS (USINT) - identifies axis
PLUS (BOOL) - indicates direction of home cycle
RATE (UDINT) - feedrate at which motion occurs (entered in LU/MIN)
DIM (DINT) - reference dimension for the nearest resolver null or the next encoder index mark when the reference switch is set (entered in LUs)
OPTN (WORD) - provides referencing options (0 or 1) 0=No option 1=Ignore index or null
BKOF (BOOL) - selects backoff of reference switch option
HOME (BOOL) - selects homing after referencing option
HDIM (DINT) - home location to move to after reference is complete

Outputs: HCMP (BOOL) - home cycle is complete
HACT (BOOL) - home cycle is being executed
QUE (USINT) - number of move for queue
SWPO (DINT) - distance in feedback units (FUs) from the reference switch to the index mark of an encoder or the null of a resolver
ERR (BYTE) - report an error 1-4 if input data is invalid

```
<<INSTANCE NAME>>:M_FHOME(EN01 := <<BOOL>>, STRT :=  
<<USINT>>, AXIS := <<USINT>> PLUS := <<BOOL>>, RATE :=  
<<UDINT>>, DIM := <<DINT>>, OPTN := <<WORD>>, BKOF :=  
<<BOOL>>, HOME := <<BOOL>>, HDIM := <<DINT>>, HCMP =>  
<<BOOL>>, HACT => <<BOOL>>, QUE => <<USINT>>, SWPO =>  
<<DINT>>, ERR => <<BYTE>>, );
```

This function block performs a fast reference cycle on an axis, followed by a homing (position) move to a designated location.

Before this function can be used, the axis must be initialized and the position loop closed.

The reference cycle will cause the selected axis to move in the designated direction until the reference switch is sensed. In a fast reference this reference switch is wired to the fast input of the selected axis on the feedback module in the PiC900. When the fast input occurs, the position of the axis is latched by the hardware on the encoder module independent of the ladder scan. When the reference switch is sensed the axis will reference (assign a value) to the next index mark of an encoder or the nearest null of a resolver. After the value is assigned, the axis will decelerate to a stop and set the reference done flag.

If the HOME input is on when the reference done has been sensed, the home move will automatically be triggered to position the axis at a desired location.

If the BKOF input is on when the reference is requested, and the axis is on the reference switch, the axis will move in the opposite direction until the reference switch opens and will then move back onto the reference switch. If the BKOF input is not on the axis will move in the specified direction until it sees an off to on transition of the limit switch.

This function block is used to perform a fast reference, immediately followed by a position move to a selected home position. It should be executed every scan unless a home cycle will only be performed when the machine is started. In that case a normally closed contact of the output of HCMP may be used.

The inputs to this function block are basically the same as for the FAST_REF function. There are three additional inputs listed below.

The BKOF input selects the backoff reference switch option.

The HOME input selects the homing after referencing option.

The HDIM input assigns the home dimension to move to.

If the axis is sitting on the limit switch when the home cycle is requested, and the BKOF input is on, the axis will move in the opposite direction of that indicated by the PLUS input until the switch opens and then will complete the home cycle in the normal manner.

The SWPO output is used to determine if the reference switch location will allow for repeatable referencing. If the reference switch is not properly located in relationship to the index marker of an encoder or the null of a resolver it could possibly reference a revolution off. To prevent this, the value reported by this output should be as follows:

- For an encoder system the value of this output should be greater than 25% and less than 75% of the total counts (FUs) per revolution. Example: For 8000 FUs/Rev, the value should be >2000 and <6000.
- For a resolver system the value of this output should be less than 25% or greater than 75% of the total counts (FUs) per revolution. Example: For 4000 FUs/Rev, the value should be <1000 or >3000.

If the value is out of range either the reference switch will have to be moved or the transducer coupling shifted.

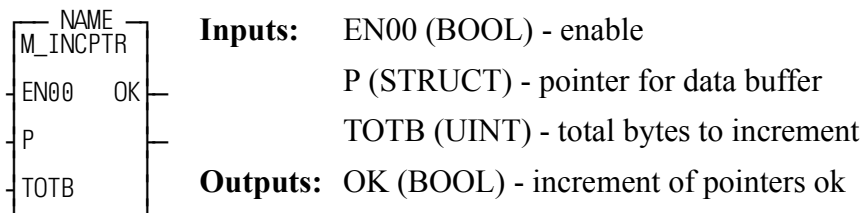
The ERR output indicates that invalid data was entered on one of the inputs. The possible errors are listed below:

ERR	Description
0	No error
1	The queue was not empty when the reference was requested
2	An error occurred in backing off of the reference switch
3	An error occurred in referencing
4	An error occurred in homing

M_INCPTR

Increment buffer pointers

USER/M_DATA



This function block increments the buffer pointers for M_DATCPT.

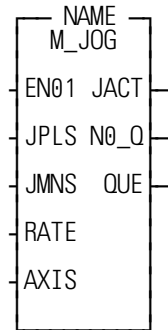
```
<<INSTANCE NAME>>:M_INCPTR (EN00 := <<BOOL>>, P := <<MEMORY AREA>>, TOTB := <<UINT>> OK => <<BOOL>>);
```

NOTES

M_JOG

Jogs a Closed Loop Axis

USER/M_MOVE



- Inputs:** EN01 (BOOL) - enables execution
JPLS (BOOL) - enables a jog in the plus direction
JMNS (BOOL) - enables a jog in the minus direction
RATE (UDINT) - feedrate at which motion occurs (entered in LU/MIN)
AXIS (USINT) - identifies axis
- Outputs:** JACT (BOOL) - indicates jogging is active when set; indicates no motion is occurring when not set
NO_Q (BOOL) - active queue for the specified axis was not available
QUE (USINT) - number of move for queue

```
<<INSTANCE NAME>>:M_JOG(EN01 := <<BOOL>>, JPLS := <<BOOL>>, JMNS := <<BOOL>>, RATE := <<UDINT>>, AXIS := <<USINT>>, JACT => <<BOOL>>), NO_Q => <<BOOL>>, QUE => <<USINT>>);
```

This function block is designed to simplify the task of doing a manual jog (velocity) move on a closed loop axis. The manual jog is defined as a move that would be triggered by the operator physically pressing a switch or a button to move an axis on the machine to a different location, without actually running a cycle.

Before this function block can be used, the axis must be initialized and placed in servo lock. If the enable input is active, triggering the jog plus (JPLS) or jog minus (JMNS) input will cause the specified axis to move at the indicated rate in the corresponding direction. When the input is deactivated motion will stop.

This function block is used to jog an axis that has been initialized and placed in servo lock with the close loop function. It checks the queue of the selected axis to be certain that no other moves are being executed. This function block should be used to allow the operator to manually move an axis on the machine. It is not designed for any other purpose.

IMPORTANT

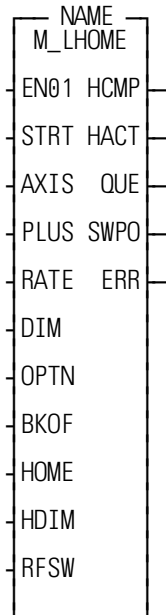
If the enable is disabled while a move is under way, the move will end.

The JPLS input enables a move in the positive direction for the selected axis. The JMNS input enables a move in the negative direction for the selected axis. If both the JPLS and JMNS inputs are set, motion will stop until one of them is dropped. At that time motion will resume in the direction still selected.

M_LHOME

Performs a Home Cycle using a Ladder Reference

USER/M_REF



Inputs: EN01 (BOOL) - enables execution
STRT (BOOL) - enables the home cycle
AXIS (USINT) - identifies axis
PLUS (BOOL) - indicates direction of home cycle
RATE (UDINT) - feedrate at which motion occurs (entered in LU/MIN)
DIM (DINT) - reference dimension for the nearest resolver null or the next encoder index mark when the reference switch is set (entered in LUs)
OPTN (WORD) - provides referencing options (0 or 1) 0=No option 1=Ignore index or null
BKOF (BOOL) - selects backoff of reference switch option
HOME (BOOL) - selects homing after referencing option
HDIM (DINT) - home location to move to after reference is complete
RFSW (BOOL) - reference switch on axis

Outputs: HCMP (BOOL) - home cycle is complete
HACT (BOOL) - home cycle is being executed
QUE (USINT) - number of move for queue
SWPO (DINT) - distance in feedback units (FUs) from the reference switch to the index mark of an encoder or the null of a resolver.
ERR (BYTE) - report an error 1-4 if input data is invalid

```
<<INSTANCE NAME>>:M_LHOME(EN01 := <<BOOL>>, STRT :=  
<<BOOL>>, AXIS := <<USINT>>, PLUS := <<BOOL>>, RATE :=  
<<UDINT>>, DIM := <<DINT>>, OPTN := <<WORD>>, BKOF :=  
<<BOOL>>, HOME := <<BOOL>> HDIM := <<DINT>>, RFSW :=  
<<BOOL>>, HCMP => <<BOOL>>, HACT => <<BOOL>>, QUE =>  
<<USINT>>, SWPO <<DINT>>, ERR => <<BYTE>>);
```

This function block performs a ladder reference cycle on an axis, followed by a homing (position) move to a designated location.

Before this function block can be used, the axis must be initialized and the position loop closed.

The reference cycle will cause the selected axis to move in the designated direction until the reference switch is sensed. In a ladder reference this reference switch is wired to an input module in the PiC900 and updated each scan of the ladder. When the reference switch is sensed the axis will reference (assign a value) to the next index mark of an encoder or the nearest null of a resolver. After the value is assigned the axis will decelerate to a stop and set the reference done flag.

If the HOME input is on when the reference done has been sensed the home move will automatically be triggered to position the axis at a desired location.

If the BKOF input is on when the reference is requested and if the axis is on the reference switch the axis will move in the opposite direction until the reference switch opens, and will then move back onto the reference switch. If the BKOF input is not on the axis will move in the specified direction until it sees an off to on transition of the limit switch.

This function block is used to perform a ladder reference, immediately followed by a position move to a selected home position. It should be executed every scan unless a home cycle will only be performed when the machine is started. In that case a normally closed contact of the output of HCMP may be used.

The inputs to this function block are similar to those of the FAST_REF function. There are four additional inputs listed below.

The BKOF input selects the backoff reference switch option.

The HOME input selects the homing after referencing option.

The HDIM input assigns the home dimension to move to.

The RFSW input is the reference switch.

If the axis is sitting on the limit switch when the home cycle is requested, and the BKOF input is on, the axis will move in the opposite direction of that indicated by the PLUS input until the switch opens and then will complete the home cycle in the normal manner.

The SWPO output is used to determine if the reference switch location will allow for repeatable referencing. If the reference switch is not properly located in relationship to the index marker of an encoder or the null of a resolver it could possibly reference a revolution off. To prevent this, the value reported by this output should be as follows:

- For an encoder system the value of this output should be greater than 25% and less than 75% of the total counts (FUs) per revolution. Example: For 8000 FUs/Rev, the value should be >2000 and <6000.

- For a resolver system the value of this output should be less than 25% or greater than 75% of the total counts (FUs) per revolution. Example: For 4000 FUs/Rev, the value should be <1000 or >3000.

If the value is out of range either the reference switch will have to be moved or the transducer coupling shifted.

The ERR output indicates that invalid data was entered on one of the inputs. The possible errors are listed below:

ERR	Description
0	No error
1	The queue was not empty when the reference was requested
2	An error occurred in backing off of the reference switch
3	An error occurred in referencing
4	An error occurred in homing

M_LINCIR

Performs Linear and Circular Moves

USER/M_MOVE

NAME	
M_LINCIR	
EN01	QUED
STRT	ERR
INC	
TIME	
RATE	
CCW	
LIN	
CIRC	
DEP	
NDPT	
CEN1	
CEN2	
BNDW	
OVRD	
PATH	

Inputs: EN01 (BOOL) - enables execution
STRT (BOOL) - enables the coordinated move
INC (WORD) - defines incremental or absolute mode (0=absolute, 1=incremental)
TIME (BOOL) - defines if move is feedrate or time of move (0=feedrate, 1=time of move)
RATE (DINT) - feedrate or time of move
CCW (BOOL) - defines direction of circular move (0=clockwise, 1=counter-clockwise)
LIN (WORD) - defines which axes to move in a linear mode
CIRC (WORD) - defines which axes to move in a circular mode
DEP (WORD) - defines which axes to move in a simultaneous endpoint arrival mode
NDPT (DINT(0..16)) - endpoints or distances to move
CEN1 (DINT) - circle center for lowest numbered circular axis
CEN2 (DINT) - circle center for highest numbered circular axis
BNDW (DINT) - circular endpoint bandwidth
OVRD (USINT) - feedrate override percentage
PATH (USINT) - path number

Outputs: QUED (BOOL) - move was queued without error
ERR (INT) - error number describing error that occurred when the move was queued

```
<<INSTANCE NAME>>:M_LINCIR(EN01 := <<BOOL>>, STRT :=  
<<BOOL>>, INC := <<WORD>>, TIME := <<BOOL>>, RATE :=  
<<DINT>>, CCW := <<BOOL>>, LIN := <<WORD>>, CIRC :=  
<<WORD>>, DEP := <<WORD>>, NDPT := <<DINT>>, CEN1 :=  
<<DINT>>, CEN2 := <<DINT>>, BNDW := <<DINT>>, OVRD :=  
<<USINT>>, PATH := <<USINT>>, QUED => <<BOOL>>, ERR =>  
<<INT>>);
```

This function block performs linear, circular, or third axis departure (simultaneous endpoint arrival) moves on a set of axes.

Before this function can be used, the axes must be initialized, the position loop must be closed, and a queue must be available on all axes to be used in the move.

This function block provides the interface from the application .LDO to the `RATIO_RL` and `CORD2RL` functions in order to perform linear coordinated, circular, and third axis departure (simultaneous endpoint arrival) motions.

Up to four separate paths of coordinated motion can be controlled. Each path of motion requires a separate instantiation of the `M_LINCIR` function block. Each path must control a unique set of axes. Only one `M_LINCIR` function block per path can be used within the application .LDO.

This function block can control up to 16 axes.

The `EN01` input of this function block must be set every scan.

The `STRT` input must be one-shot. When it is one-shot, the function block will start the coordinated move, or enter it in the queue for the axes. It is the user's responsibility to ensure that there is a queue available on all of the axes involved in the move before pulsing this input.

The `INC` input defines whether each axis should move in the absolute or incremental mode. One bit of this `WORD` is reserved for each of the sixteen possible axes. Bit 0 is set if axis 1 is incremental, or reset if axis 1 is absolute, bit 1 is set if axis 2 is incremental, reset if axis 2 is absolute, etc.

The `TIME` input defines whether the move should be executed as a path feedrate move or a time of move. This input should be reset for path feedrate, or set for time of move.

If the `TIME` input is reset, then the `RATE` input is the path feedrate for the move in ladder units/minute. If the `TIME` input is set, then the `RATE` input is the time for the move in milliseconds.

The `CCW` input is only used for circular moves. If it is reset, then the move is clockwise, if it is set, then the move is counter-clockwise.

The `LIN` input defines which axes in the move are to be moved in a linear mode. One bit of the `WORD` is reserved for each of the sixteen axes. The bit must be set for the axis to do a linear move. Axes who have their bits set will be included in the calculations for the path feedrate.

The `CIRC` input defines which axes in the move are to be moved in a circular mode. One bit of the `WORD` is reserved for each of the sixteen axes. The bit must be set for the axis to do a circular move. Axes who have their bits set will be included in the calculations for the path feedrate.

The `DEP` input defines which axes in the move are to be moved in a simultaneous endpoint arrival mode. One bit of the `WORD` is reserved for each of the sixteen axes. The bit must be set for the axis to move. Axes who have their bits set will not be included in the calculations for the path feedrate, but they will arrive at their endpoints simultaneously with the axes that are.

The LIN, CIRC, and DEP words may never have the same bits set in them at a time. You must always set a bit for every axis ever used in the path, even if the axis is not to move in this particular move. In this case, you would set either the LIN or DEP bit for the axis, set the INC bit for the axis, and program an endpoint of zero for the axis.

The NDPT array holds the endpoints for the axes used in the move. The 0th element is not used. If the INC bit is set for the axis, this is the distance to move, if the INC bit is reset for the axis, then this is the position to move to. The endpoints are entered in ladder units.

The CEN1 and CEN2 inputs define the circle centers if a circular move is being performed. The CEN1 input is the center for the lowest numbered circular axis, and the CEN2 input is the center for the highest numbered circular axis. The centers are always programmed as an incremental distance from the starting point of the circle, even if the INC bit for the axes is not set. The centers are entered in ladder units. For example, if a circle were being done with axes 4 and 6, then CEN1 would be the center for axis 4, and CEN2 would be the center for axis 6.

The BNDW input defines a bandwidth for circular moves. When a circular move is requested, the distance from the start point to the center point and the distance from the endpoint to the center point are compared for both axes. If these distances differ by more than the bandwidth entered here, then the move will not execute and error 14 will be returned on the ERR output. This bandwidth is entered in ladder units.

The OVRD input defines the feedrate override value. This can be changed at any time, even if the STRT input is not energized. This adjusts the actual feedrate or time to be from 0 to 255 percent of the programmed feedrate or time.

The PATH input defines the number of the path. Up to four totally independent paths of coordinated motion can be defined. This must be a number from 1 to 4. This should not be changed once it is set.

The QUED output will be set for one scan when STRT is pulsed and the move has been successfully queued on all axes defined. If an error occurred in queueing the move, this output will be reset when STRT is pulsed, and an error code will be stored in the ERR output.

The ERR output will be non-zero if an error occurs in queueing a move. A list of error codes is shown on the following table.

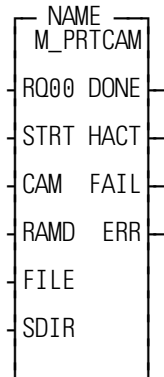
Note: WRITE_SV variable 25 Fast Queuing is enabled for the selected axes when STRT is set. Fast queuing will remain on for those axes until turned off by you.

ERR	Description
0	No error
1	No bits were set in the LIN, CIRC, or DEP WORDs
2	The same bit was set in the LIN and CIRC WORDs
3	The same bit was set in the DEP and CIRC WORDs
4	The same bit was set in the LIN and DEP WORDs
5	The number of bits set in the CIRC WORD was not 0 or 2
6	Not used
7	Not used
8	The time of move or feedrate was negative
9	The time of move or feedrate was zero
10	The feedrate was too high or the time was too low to calculate
11	The feedrate was too low or the time was too high to calculate
12	An axis that was selected was not initialized by the servo setup function
13	The STRTSRV function has not been called
14	Endpoint not on circle
1XX	When the distance to move was converted to feedback units, it was too positive to fit into 32 bits. XX = Axis number
2XX	When the distance to move was converted to feedback units, it was too negative to fit into 32 bits. XX = Axis number
3XX	The path feedrate or time entered causes an axis to exceed its velocity limit from servo setup. XX = Axis number
32766	The time axis could not be started
32767	One of the OKs on the RATIO_RL functions did not get set

M_PRTCAM

Creates a RATIOCAM text file

USER/M_DATA



Inputs: RQ00 (BOOL) - request file generation and write
CAM (STRUC 0..) - CAM structure input of the RATIOCAM function
RAMD (BOOL) - If enabled, allows file to be written to the RAMDISK. If disabled, file is written to the PC running PiCPro.
FILE (STRING) - name of the file
SDIR (STRING) - identifies the subdirectory where the file will be written to.

Outputs: DONE (BOOL) - set when the file is generated and written, reset when RQ00 goes on.
FAIL (BOOL) - set if an error occurs and reset when RQ00 goes on.
ERR (INT) - number of error that occurred. These errors are defined in Appendix B.

```
<<INSTANCE NAME>>:M_PRTCAM(RQ00 := <<BOOL>>, CAM :=  
<<MEMORY AREA>>, RAMD := <<BOOL>>, FILE := <<STRING>>  
SDIR := <<STRING>>, DONE => <<BOOL>>, FAIL => <<BOOL>> ERR  
=> <<INT>>);
```

This function block creates a text file for a RATIOCAM CAM structure. The file can be created on either the RAMDISK in the PiC or on the PC running PiCPro. A positive transition of RQ00 requests that the data specified by the CAM input be converted to ASCII code, concatenated, and written to the RAMDISK or to the PiCPro port. The CAM input is an array of structures and must have the following members:

Name	Data Type	Definition
CAM	STRUCT (0..998)	The structure of the RATIOCAM profile
.M	INT	Master segment size
.S	INT	Slave segment size

The FILE input requires a string data type variable with the filename as an initial value. The format is "FILENAME.EXT".

The SDIR input requires a string data type. A subdirectory is not required if you are writing the file to the RAMDISK. If you are writing the file to a PC running PiCPro, then the SDIR is required. It must contain the drive and subdirectory path. The following are examples showing the drive and subdirectory path:

C: indicates that the file will be written to C:Filename.ext.

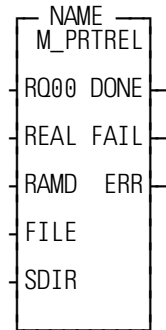
C:\PRT_CAM indicates the file will be written to the directory

C:\PRT_CAM\Filename.ext.

M_PRTREL

Creates a *RATIO_RL* text file

USER/M_DATA



Inputs: RQ00 (BOOL) - request file generation and write
REAL (STRUC 0..) - REAL structure input of the RATIO_RL function
RAMD (BOOL) - If enabled, allows file to be written to the RAMDISK. If disabled, file is written to the PC running PiCPro.
FILE (STRING) - name of the file
SDIR (STRING) - identifies the subdirectory where the file will be written to.

Outputs: DONE (BOOL) - set when the file is generated and written, reset when RQ00 goes on.
FAIL (BOOL) - set if an error occurs and reset when RQ00 goes on.
ERR (INT) - number of error that occurred. These errors are defined in Appendix B.

```
<<INSTANCE NAME>>:M_PRTREL(RQ00 := <<BOOL>>, REAL :=  
<<MEMORY AREA>>, RAMD := <<BOOL>>, FILE := <<(STRING)>>,  
SDIR := <<STRING>>, DONE => <<BOOL>>, FAIL => <<BOOL>>,  
ERR => <<INT>>);
```

This function block creates a text file for a *RATIO_RL* structure. The file can be created on the RAMDISK in the PiC or on the PC running PiCPro. A positive transition of RQ00 requests that the data specified by the REAL input be converted to ASCII code, concatenated, and written to the RAMDISK or to the PiCPro port. The REAL input is an array of structures and must have these members:

Name	Data Type	Definition
REAL	STRUCT (0..998)	The structure of the RATIO_RL profile
.M	DINT	Master segment size
.S	DINT	Slave segment size
.LEN	LREAL	Length or K1
.AMPL	LREAL	Amplitude or K2
.STANGL	LREAL	Start angle or K3
.SPARE	LREAL	Spare for future use
.FLAGS	DWORD	Flags

The FILE input requires a string data type variable with the filename as an initial value. The format is "FILENAME.EXT". The SDIR input requires a string data type. A subdirectory is not required if you are writing the file to the RAMDISK. If you are writing the file to a PC running PiCPro, then the SDIR is required. It must contain the drive and subdirectory path. The following are examples showing the drive and subdirectory path:

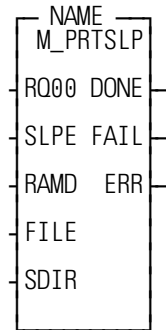
C: indicates that the file will be written to C:Filename.ext.

C:\PRT_CAM indicates the file will be written to the directory C:\PRT_CAM\Filename.ext.

M_PRTSLP

Creates a RATIOSLP text file

USER/M_DATA



Inputs: RQ00 (BOOL) - request file generation and write
SLPE (STRUC 0..) - SLPE structure input of the RATIOSLP function
RAMD (BOOL) - If enabled, allows file to be written to the RAMDISK. If disabled, file is written to the PC running PiCPro.
FILE (STRING) - name of the file
SDIR (STRING) - identifies the subdirectory where the file will be written to.

Outputs: DONE (BOOL) - set when the file is generated and written, reset when RQ00 goes on.
FAIL (BOOL) - set if an error occurs and reset when RQ00 goes on.
ERR (INT) - number of error that occurred. These errors are defined in Appendix B.

```
<<INSTANCE NAME>>:M_PRTSLP(RQ00 := <<BOOL>>, SLPE :=  
  <<MEMORY AREA>> RAMD := <<BOOL>>, FILE := <<STRING>> SDIR  
  := <<STRING>>, DONE => <<BOOL>>, FAIL => <<BOOL>>, ERR =>  
  <<INT>>);
```

This function block creates a text file for a RATIOSLP structure. The file can be created on either the RAMDISK in the PiC or on the PC running PiCPro.

A positive transition of RQ00 requests that the data specified by the SLPE input be converted to ASCII code, concatenated, and written to the RAMDISK or to the PiCPro port.

The REAL input is an array of structures and must have the following members:

Name	Data Type	Definition
SLPE	STRUCT (0..998)	The structure of the RATIOSLP profile
.M	INT	Master segment size
.S	INT	Slave segment size
.SLP	DINT	Slope of segment
.SR	DINT	Start ratio
.FLAGS	DWORD	Default flags

The FILE input requires a string data type variable with the filename as an initial value. The format is "FILENAME.EXT". The SDIR input requires a string data type. A subdirectory is not required if you are writing the file to the RAMDISK. If you are writing the file to a PC running PiCPro, then the SDIR is required. It must contain the drive and subdirectory path. The following are examples showing the drive and subdirectory path:

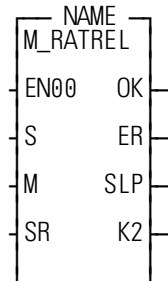
C: indicates that the file will be written to C:Filename.ext.

C:\PRT_CAM indicates the file will be written to the directory C:\PRT_CAM\Filename.ext.

M_RATREL

Calculates Ending Ratio and Slope

USER/M_DATA



Inputs: EN01 (BOOL) - enables execution
S (DINT) - slave distance
M (DINT) - master distance
SR (LREAL) - starting ratio

Outputs: OK (BOOL) - execution complete
ER (LREAL) - ending ratio
SLP (LREAL) - slope

K2 (LREAL) - slope output divided by 2

```
<<INSTANCE NAME>>:M_RATREL(EN01 := <<BOOL>>, S :=  
<<DINT>>, M := <<DINT>>, SR := <<LREAL>>, OK => <<BOOL>>, ER  
=> <<LREAL>>, SLP => <<LREAL>>, K2 => <<LREAL>>);
```

This function block calculates the ending ratio, slope, and K2 (slope/2) used in the ratio real structure from the master distance, slave distance, and starting ratio.

This function block calculates the ending ratio and slope to be used with the RATIO_RL structure as one segment of the RATIO_RL profile. Refer to the documentation in the PiCPro Online Help regarding RATIO_RL for more information.

The slave and master segments (S and M) are entered in feedback units.

The starting ratio for the first segment of a RATIO_RL profile is normally zero. The starting ratio is called LEN or K1 in the ratio real documentation.

The formulas used by this function for calculation are as follows:

$$ER = (2S / M) - SR$$

$$SLP = (ER - SR) / M$$

$$K2 = SLP / 2$$

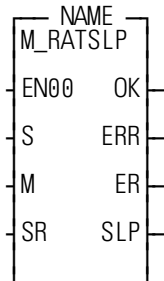
where ER is the ending ratio, SR is the starting ratio, S is the slave distance, M is the master distance, SLP is the slope, and K2 is the slope divided by 2. K2 is the AMPL structure member of the RATIO_RL REAL structure for a linear move.

The ending ratio is not an input to the RATIO_RL structure. However the ending ratio of one segment is normally used as the starting ratio of the next segment.

M_RATSLP

Calculates Ending Ratio and Slope

USER/M_DATA



Inputs: EN01 (BOOL) - enables execution
S (INT) - slave distance
M (INT) - master distance
SR (DINT) - starting ratio

Outputs: OK (BOOL) - execution complete without error
ERR (INT) - error number
ER (DINT) - ending ratio
SLP (DINT) - slope

```
<<INSTANCE NAME>>:M_RATREL(EN01 := <<BOOL>>, S := <<INT>>,
M := <<INT>>, SR := <<DINT>>, OK => <<BOOL>>, ERR => <<INT>>,
ER => <<DINT>>, SLP => <<DINT>>);
```

This function block calculates the ending ratio and slope used in the ratio slope structure from the master distance, slave distance, and starting ratio.

This function block calculates the ending ratio and slope to be used with the RATIOSLP structure as one segment of the RATIOSLP profile. Refer to the documentation in the PiCPro Online Help regarding RATIOSLP for more information. The slave and master segments (S and M) are entered in feedback units.

The starting ratio for the first segment of a slope profile is normally zero. Non zero starting ratios must already be multiplied by the scaling factor of 16777216 before being used as an input to this function.

The formulas used by this function for calculation are as follows:

$$ER = (2S / M) - SR$$

$$SLP = (ER - SR) / M$$

where ER is the ending ratio, SR is the starting ratio, S is the slave distance, M is the master distance, and SLP is the slope.

The ending ratio and slope that are outputs of this function have been multiplied by the scaling factor of 16777216. The ending ratio is not an input to the RATIOSLP structure. However, the ending ratio of one segment is normally used as the starting ratio of the next segment.

ERR Description

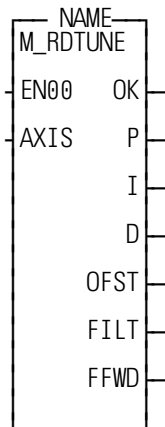
- 1 The calculation for ER failed when S was between -64 and +63 (inclusive)
- 2 The calculation for ER failed when S was less than -64 or greater than +63
- 3 The calculation for SLP failed

Note: An M value of zero results in an error due to an attempt to divide by 0. No master distance can have a value of zero in a RATIOSLP profile.

M_RDTUNE

Reads tuning parameters

USER/M_DATA



Inputs: EN00 (BOOL) - enables execution

AXIS (USINT) - identifies axis

Outputs: OK (BOOL) - execution complete

P (DINT) - proportional gain

I (DINT) - integral gain

D (DINT) - derivative gain

OFST (DINT) - analog output offset

FILT (DINT) - slow speed filter value

FFWD (DINT) - feedforward percentage

```
<<INSTANCE NAME>>:M_RDTUNE(EN00 := <<BOOL>>, AXIS :=  
<<USINT>>, OK => <<BOOL>>, P => <<DINT>>, I => <<DINT>>, D =>  
<<DINT>>, OFST => <<DINT>>, FILT => <<DINT>>, FFWD =>  
<<DINT>>);
```

This function block allows you to read all six tuning parameters from the TUNERead function in a single function.

This function block requires the numeric processor or a 486 DX processor.

The proportional gain for AXIS will be returned in P. P is in ladder units per minute per ladder unit of following error (LU / MIN / LUFE).

The integral gain for AXIS will be returned in I. I is in ladder units per minute per ladder units of following error times minutes (LU / MIN / LUFE * MIN).

The derivative gain for AXIS will be returned in D. D is in ladder units per minute per ladder unit of following error per minute (LU / MIN / LUFE / MIN).

The analog output offset voltage for AXIS will be returned in OFST. OFST is in millivolts.

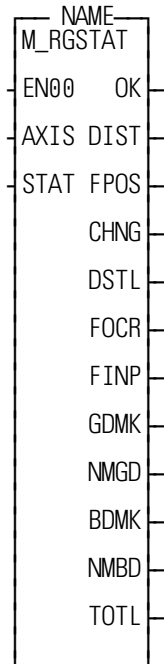
The slow speed filter value for AXIS will be returned in FILT. FILT is in milliseconds.

The feedforward percentage for AXIS will be returned in FFWD. FFWD will be from 0 to 100.

M_RGSTAT

Returns Registration Data

USER/M_DATA



Inputs: EN00 (BOOL) - enables execution
AXIS (USINT) - axis number
STAT (WORD) - status word from STATUSSV function

Outputs: OK (BOOL) - execution complete
DIST (DINT) - fast input distance
FPOS (DINT) - fast input position
CHNG (DINT) - registration/referencing position change
DSTL (BOOL) - indicates distance plus tolerance has been exceeded
FOCR (BOOL) - fast input occurred
FINP (BOOL) - fast input on
GDMK (BOOL) - good mark detected
NMGD (DINT) - number of good registration marks
BDMK (BOOL) - bad mark detected
NMBD (DINT) - number of bad registration marks
TOTL (DINT) - total number of fast inputs that have occurred

```
<<INSTANCE NAME>>:M_RGSTAT(EN00 := <<BOOL>>, AXIS :=  
<<USINT>>, STAT := <<WORD>>, OK => <<BOOL>>, DIST =>  
<<DINT>>, FPOS => <<DINT>>, CHNG => <<BOOL>>, DSTL =>  
<<BOOL>>, FOCR => <<BOOL>>, FINP => <<BOOL>>, GDMK =>  
<<BOOL>>, NMGD => <<DINT>>, BDMK => <<BOOL>>, NMBD =>  
<<DINT>>, TOTL => <<DINT>>);
```

This function block obtains information about registration. The information gathered is distance between fast inputs, fast input position, registration reference change, number of good marks, number of bad marks, total number of marks, and the state of STATUSSV flags.

This function block should be enabled every scan.

The input at AXIS determines which axis the output information is for. AXIS can be a closed loop or digitizing axis.

The STAT input is the status word read from the STATUSSV function. STATUSSV can only be called once per scan, so its output is used as an input to this function.

The OK output will not be set if the axis has not been initialized.

The DIST output is the distance between the most recent fast input and the previous fast input in ladder units.

The FPOS output is the actual position of the axis at the point where the most recent fast input occurred in ladder units.

The CHNG output is the amount the position of the axis has changed in ladder units due to registration or the last machine reference.

The DSTL output will be set if the distance from the last mark exceeds the value of $DIST + TOLR$ whether or not a mark has occurred. It will be reset when any mark occurs.

The FOCR output will be set if a fast input has occurred since the last time the STATUSSV function was called.

The FINP output is set if the fast input is on, and reset if the fast input is off.

The GDMK output will be set if a good mark has been detected since the last time the STATUSSV function was called.

The NMGD output holds the total number of good registration marks that have been detected.

The BDMK output will be set if a bad mark has been detected since the last time the STATUSSV function was called.

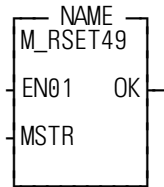
The BAD output holds the number of bad registration marks that have been detected.

The TOTL output holds the total number of fast input transitions that have occurred.

M_RSET49

Reset Errors on Digitizing Axes 49 to 56

USER/M_INIT



Inputs: EN01 (BOOL) - enables execution
MSTR (BOOL) - machine start input

Outputs: OK (BOOL) - execution complete

```
<<INSTANCE NAME>>:M_RSET49(EN01 := <<BOOL>>, MSTR :=  
<<BOOL>>, OK => <<BOOL>>);
```

This function block is used to reset the E-stop errors on digitizing axes 49 through 56 when the machine start input is pulsed.

This function block should be enabled every scan.

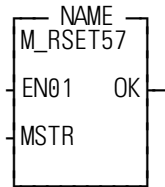
The machine start input must go through a positive transition (off to on) to reset the errors.

On a positive transition of MSTR, this function will reset all E-stop errors on axes 49 through 56.

M_RSET57

Reset Errors on Digitizing Axes 57 to 64

USER/M_INIT



Inputs: EN01 (BOOL) - enables execution
MSTR (BOOL) - machine start input

Outputs: OK (BOOL) - execution complete

```
<<INSTANCE NAME>>:M_RSET57(EN01 := <<BOOL>>, MSTR :=  
<<BOOL>>, OK => <<BOOL>>);
```

This function block is used to reset the E-stop errors on digitizing axes 57 through 64 when the machine start input is pulsed.

This function block should be enabled every scan.

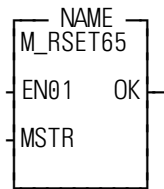
The machine start input must go through a positive transition (off to on) to reset the errors.

On a positive transition of MSTR, this function will reset all E-stop errors on axes 57 through 64.

M_RSET65

Reset Errors on Digitizing Axes 65 to 72

USER/M_INIT



Inputs: EN01 (BOOL) - enables execution
MSTR (BOOL) - machine start input

Outputs: OK (BOOL) - execution complete

```
<<INSTANCE NAME>>:M_RSET65(EN01 := <<BOOL>>, MSTR :=  
<<BOOL>>, OK => <<BOOL>>);
```

This function block is used to reset the E-stop errors on digitizing axes 65 through 72 when the machine start input is pulsed.

This function block should be enabled every scan.

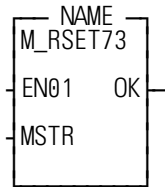
The machine start input must go through a positive transition (off to on) to reset the errors.

On a positive transition of MSTR, this function will reset all E-stop errors on axes 65 through 72.

M_RSET73

Reset Errors on Digitizing Axes 73 to 80

USER/M_INIT



Inputs: EN01 (BOOL) - enables execution
MSTR (BOOL) - machine start input

Outputs: OK (BOOL) - execution complete

```
<<INSTANCE NAME>>:M_RSET73(EN01 := <<BOOL>>, MSTR :=  
<<BOOL>>, OK => <<BOOL>>);
```

This function block is used to reset the E-stop errors on digitizing axes 73 through 80 when the machine start input is pulsed.

This function block should be enabled every scan.

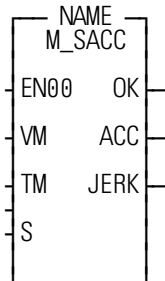
The machine start input must go through a positive transition (off to on) to reset the errors.

On a positive transition of MSTR, this function will reset all E-stop errors on axes 73 through 80.

M_SACC

Calculate ACC and JERK values with ACC_JERK

USER/M_MOVE



Inputs: EN00 (BOOL) - enables execution (one-shot)
VM (UDINT) - maximum velocity of move in ladder units/min
TM (REAL) - total time to reach velocity VM if axis starts from rest
S (USINT) - percentage of time spent in constant jerk

Outputs: OK (BOOL) - execution completed without error
ACC (LREAL) - maximum acceleration rate in ladder units or counts/min/sec
JERK (LREAL) - constant jerk in ladder units or counts/min/sec²

```
<<INSTANCE NAME>>:M_SACC(EN00 := <<BOOL>>, VM :=  
<<UDINT>>, TM :=<<REAL>>, S := <<USINT>>, OK => <<BOOL>>),  
ACC => <<LREAL>>), JERK => <<LREAL>>);
```

This function block is used to calculate the ACC and JERK values to be used with the ACC_JERK function.

Note: This function block is not intended to be used directly with the SCURVE or M_SCRVLC because the units for those functions are different (e.g ACC is Counts/min/min).

Inputs:

The EN00 input of this function block would normally be one-shot.

The VM input is set to the maximum velocity for the servo or time axis move to be executed.

The TM input sets the total time to reach velocity VM if the axis starts from rest. Typical values might be 0.1 seconds or 10 seconds. This value must be positive or the OK will not be set.

The S input sets the percentage of time spent in constant jerk. A value of 80(%) percent means 40% of the acceleration time spent in constant jerk, 20% in constant acceleration and another 40% in constant jerk. This value must be set to 100 or less or the OK will not be set.

Outputs:

The OK is set if the input values are within range and the output values were calculated.

The ACC output is the maximum acceleration rate for the axis expressed in ladder units/min/sec for a servo axis and counts/min/sec for a time axis.

The JERK output is the constant jerk in ladder units/min/sec for a servo axis and counts/min/sec² for a time axis.

M_SCRVLC

Performs Linear and Circular Moves with S-Curve

USER/M_MOVE

NAME	
M_SCRVLC	
EN01	QUED
STRT	ERR
INC	
TIME	
RATE	
CCW	
LIN	
CIRC	
DEP	
NDPT	
CEN1	
CEN2	
BNDW	
OVRD	
PATH	
ACCL	
JERK	
MAXF	

Inputs:

- EN01 (BOOL) - enables execution
- STRT (BOOL) - enables the coordinated move
- INC (WORD) - defines incremental or absolute mode for up to 16 axes (0=absolute, 1=incremental)
- TIME (BOOL) - defines if move is feedrate or time of move (0=feedrate, 1=time of move)
- RATE (DINT) - feedrate or time of move
- CCW (BOOL) - defines direction of circular move (0=clockwise, 1=counter-clockwise)
- LIN (WORD) - defines which axes to move in a linear mode
- CIRC (WORD) - defines which axes to move in a circular mode
- DEP (WORD) - defines which axes to move in a simultaneous endpoint arrival mode
- NDPT (DINT(0..16)) - endpoints or distances to move
- CEN1 (DINT) - circle center for lowest numbered circular axis
- CEN2 (DINT) - circle center for highest numbered circular axis
- BNDW (DINT) - circular endpoint bandwidth
- OVRD (USINT) - feedrate override percentage
- PATH (USINT) - path number
- ACCL (LREAL) - path acceleration in ladder units/min²
- JERK (LREAL) - path jerk in ladder units/min³
- MAXF (DINT) - maximum path feedrate in ladder units/minute

Outputs:

- QUED (BOOL) - move was queued without error
- ERR (INT) - error number describing error that occurred when the move was queued

```

<<INSTANCE NAME>>:M_SCRVLC(EN01 := <<BOOL>>, STRT :=
<<BOOL>>, INC := <<WORD>>, TIME := <<BOOL>>, RATE :=
<<DINT>>, CCW := <<BOOL>>, LIN := <<WORD>>, CIRC :=
<<WORD>>, DEP := <<WORD>>, NDPT := <<DINT (0..16)>>, CEN1 :=
<<DINT>>, CEN2 := <<DINT>>, BNDW := <<DINT>>, OVRD :=
<<USINT>>, PATH := <<USINT>>, ACCL := <<LREAL>>, JERK :=
<<LREAL>>, MAXF := <<DINT>>, QUED => <<BOOL>>, ERR =>
<<INT>>);

```

The M_SCRVLC function block provides the interface from the application .LDO to the RATIO_RL function in order to perform linear coordinated, circular, or third axis departure (simultaneous endpoint arrival) moves with S-curve acceleration and deceleration. Before this function can be used, the axes must be initialized, the position loop must be closed, and a queue must be available on all axes to be used in the move.

Up to four separate paths of coordinated motion can be controlled. Each path of motion requires a separate instantiation of the M_SCRVLC function block. Each path must control a unique set of axes. Only one M_SCRVLC function block per path can be used with the application .LDO.

This function block can control up to 16 axes.

Note: This function block requires a numeric processor or a 486 DX processor in the PiC900 and version 6.2 or higher of PiCPro.

Inputs

The EN01 input of this function block must be set every scan.

The STRT input must be one-shot. When it is one-shot, the function block will start the coordinated move, or enter it in the queue for the axes. It is the user's responsibility to ensure that there is a queue available on all of the axes involved in the move before pulsing this input.

The INC input defines whether each axis should move in the absolute or incremental mode. One bit of this WORD is reserved for each of the sixteen possible axes. Bit 0 is set if axis 1 is incremental, or reset if axis 1 is absolute, bit 1 is set if axis 2 is incremental, reset if axis 2 is absolute, etc.

The TIME input defines whether the move should be executed as a path feedrate move or a time of move. This input should be reset for path feedrate, or set for time of move.

If the TIME input is reset, then the RATE input is the path feedrate for the move in ladder units/minute. If the TIME input is set, then the RATE input is the time for the move in milliseconds.

The RATE is the path feedrate or the time for the move to execute depending on the TIME input.

The CCW input is only used for circular moves. If it is reset, then the move is clockwise, if it is set, then the move is counter-clockwise.

The LIN input defines which axes in the move are to be moved in a linear mode. One bit of the WORD is reserved for each of the sixteen axes. The bit must be set for the axis to do a linear move. Axes who have their bits set will be included in the calculations for the path feedrate.

The CIRC input defines which axes in the move are to be moved in a circular mode. One bit of the WORD is reserved for each of the sixteen axes. The bit must be set for the axis to do a circular move. Axes who have their bits set will be included in the calculations for the path feedrate.

The DEP input defines which axes in the move are to be moved in a simultaneous endpoint arrival mode. One bit of the WORD is reserved for each of the sixteen axes. The bit must be set for the axis to move. Axes who have their bits set will not be included in the calculations for the path feedrate, but they will arrive at their endpoints simultaneously with the axes that are.

Note: The LIN, CIRC, and DEP words may never have the same bits set in them at a time. You must always set a bit for every axis ever used in the path, even if the axis is not to move in this particular move. In this case, you would set either the LIN or DEP bit for the axis, set the INC bit for the axis, and program an endpoint of zero for the axis.

The NDPT array holds the endpoints for the axes used in the move. The 0th element is not used. If the INC bit is set for the axis, this is the distance to move, if the INC bit is reset for the axis, then this is the position to move to. The endpoints are entered in ladder units.

The CEN1 and CEN2 inputs define the circle centers if a circular move is being performed. The CEN1 input is the center for the lowest numbered circular axis, and the CEN2 input is the center for the highest numbered circular axis. The centers are always programmed as an incremental distance from the starting point of the circle, even if the INC bit for the axes is not set. The centers are entered in ladder units. For example, if a circle were being done with axes 4 and 6, then CEN1 would be the center for axis 4, and CEN2 would be the center for axis 6.

The BNDW input defines a bandwidth for circular moves. When a circular move is requested, the distance from the start point to the center point and the distance from the endpoint to the center point are compared for both axes. If these distances differ by more than the bandwidth entered here, then the move will not execute and error 14 will be returned on the ERR output. This bandwidth is entered in ladder units.

The OVRD input defines the feedrate override value. This can be changed at any time, even if the STRT input is not energized. This adjusts the actual feedrate or time to be from 0 to 255 percent of the programmed feedrate or time.

The PATH input defines the number of the path. Up to four totally independent paths of coordinated motion can be defined. This must be a number from 1 to 4. This should not be changed once it is set.

The ACCL is the path acceleration in ladder units/min². The JERK is the path jerk in ladder units/min³. The MAXF is the maximum path feedrate in ladder units/min. This should not be changed once it is set.

Outputs

The QUED output will be set for one scan when STRT is pulsed and the move has been successfully queued on all axes defined. If an error occurred in queueing the move, this output will be reset when STRT is pulsed, and an error code will be stored in the ERR output.

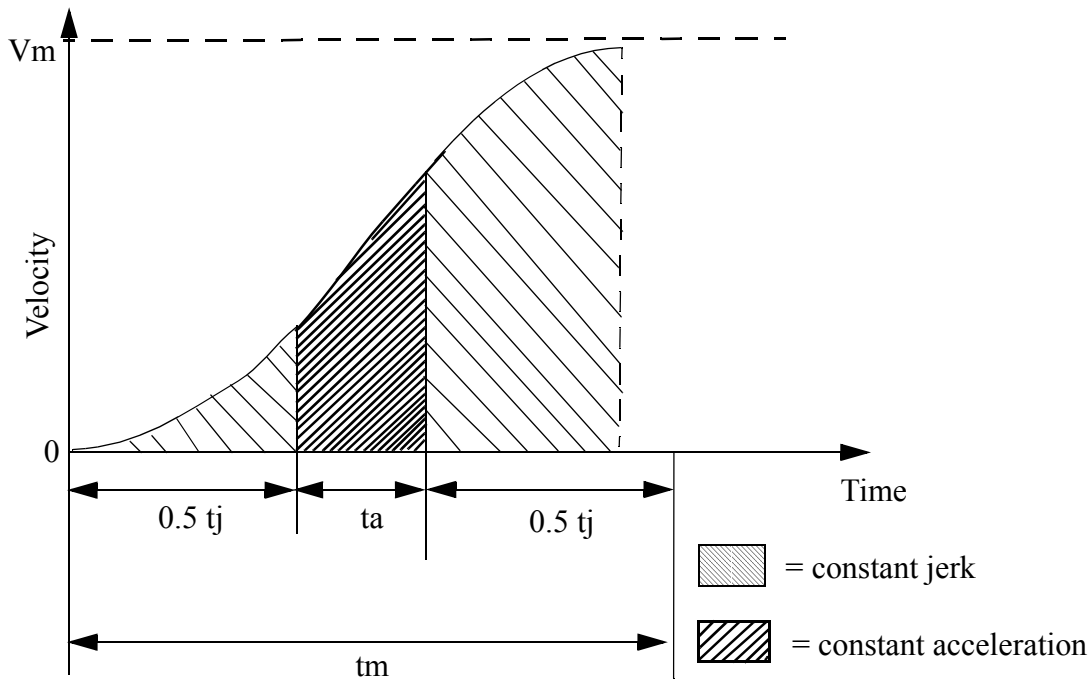
The ERR output will be non-zero if an error occurs in queueing a move. A list of error codes is shown below:

ERR	Description
0	No error
1	No bits were set in the LIN, CIRC, or DEP WORDs
2	The same bit was set in the LIN and CIRC WORDs
3	The same bit was set in the DEP and CIRC WORDs
4	The same bit was set in the LIN and DEP WORDs
5	The number of bits set in the CIRC WORD was not 0 or 2
6	Not used
7	Not used
8	The time of move or feedrate was negative
9	The time of move or feedrate was zero
10	The feedrate was too high or the time was too low to calculate
11	The feedrate was too low or the time was too high to calculate
12	An axis that was selected was not initialized by the servo setup function
13	The STRTSRV function has not been called
14	Endpoint not on circle
1XX	When the distance to move was converted to feedback units, it was too positive to fit into 32 bits. XX = Axis number
2XX	When the distance to move was converted to feedback units, it was too negative to fit into 32 bits. XX = Axis number
3XX	The path feedrate or time entered causes an axis to exceed its velocity limit from servo setup. XX = Axis number
32766	The time axis could not be started
32767	One of the OKs on the RATIO_RL functions did not get set or the OK on the time axis distance move did not get set

Calculating ACCL and JERK

This section explains how to calculate the ACCL and JERK inputs for the function block.

The drawing below illustrates an S-curve acceleration.



V_m = Maximum path velocity

t_m = The total time it takes to get to velocity V_m if the axis starts at 0.

s = The percentage of time (t_m) spent in constant jerk.

From 0 to t_1 , the axis will be in constant jerk

From t_1 to t_2 , the axis will be in constant acceleration.

From t_2 to t_m , the axis will again be in constant jerk.

The formulas below show the relationship between t_m , t_1 , t_2 , and s .

$$t_1 = t_m - t_2 = \left(\frac{1}{2} \times s \times t_m \right)$$

$$t_2 = t_m - \left(\frac{1}{2} \times s \times t_m \right)$$

For a 10% S-curve, 10% of the time (t_m) is spent in constant jerk.

This means that $s = 0.1$.

For a 20% S-curve, 20% of the time (t_m) is spent in constant jerk.

This means that $s = 0.2$, etc.

If you know V_m , t_m , and s , then you can calculate jerk and acceleration using the following formulas.

$$JERK = \frac{2 \times V_m}{s \times t_m^2 (1 - 0.5 \times s)}$$

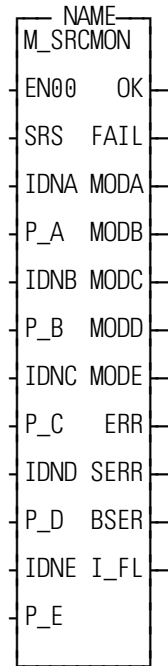
$$ACCL = \frac{V_m}{t_m \times (1 - 0.5 \times s)}$$

The units for JERK are ladder units per minute³; therefore, V_m is in ladder units per minute and t_m is in minutes. The units for ACCL are ladder units per minute².

M_SRCMON

Monitors up to five SERCOS IDNs

USER/M_SERCOS



Inputs: EN00 (BOOL) - enables execution
SRS (STRUCT) - slot, ring, and slave to monitor
IDNA (UINT) - number of first IDN to monitor
P_A (BOOL) - set for Product IDN, reset for System IDN
IDNB (UINT) - number of second IDN to monitor
P_B (BOOL) - set for Product IDN, reset for System IDN
IDNC (UINT) - number of third IDN to monitor
P_C (BOOL) - set for Product IDN, reset for System IDN
IDND (UINT) - number of fourth IDN to monitor
P_D (BOOL) - set for Product IDN, reset for System IDN
IDNE (UINT) - number of fifth IDN to monitor
P_E (BOOL) - set for Product IDN, reset for System IDN

Outputs: OK (BOOL) - execution complete
FAIL (BOOL) - execution failure
MODA (REAL) - value of first IDN
MODB (REAL) - value of second IDN
MODC (REAL) - value of third IDN
MODD (REAL) - value of fourth IDN
MODE (REAL) - value of fifth IDN
ERR (INT) - SERCOS error*
SERR (UINT) - SERCOS slave error*
BSER (INT) - SERCOS block specific error*
I_FL (UINT) - indicates the IDN that failed (1 through 5 corresponding to A through E) if an error occurs during a read
*See error tables at end of the M_SRCWTL function block section.

```

<<INSTANCE NAME>>:M_SRCMON(EN00 := <<BOOL>>, SRS :=
  <<MEMORY AREA>>, IDNA := <<UINT>>, P_A := <<BOOL>>, IDNB
  := <<UINT>>, P_B := <<BOOL>>, IDNC := <<UINT>>, P_C :=
  <<BOOL>>, IDND := <<UINT>>, P_D := <<BOOL>>, IDNE :=
  <<UINT>>, P_E := <<BOOL>>, OK => <<BOOL>>, FAIL =>
  <<BOOL>>, MODA => <<REAL>>, MODB => <<REAL>>, MODC =>
  <<REAL>>, MODD => <<REAL>>, MODE => <<REAL>>, ERR =>
  <<INT>>, SERR => <<UINT>>, BSER => <<INT>>, I_FL =>
  <<UINT>>);

```

The M_SRCMON function block monitors up to five SERCOS IDNs for a single SERCOS slave. The operation data for each IDN is continuously read as long as the EN00 input is energized.

The IDNA through IDNE inputs can be used or left blank. When the EN00 input transitions from off to on, the attributes of each IDN are read and saved in the function block. These attributes are used to scale the data being monitored into engineering units for the output. If the IDNA through IDNE inputs are changed while monitoring, the EN00 input must be dropped and then re-energized so that the attributes for each IDN are read again.

The SRS input is used to indicate which SERCOS slave to monitor. Slot, ring, and slave are used instead of an axis number so that this function block can be used in phase 2 initialization if desired. The SRS structure must be declared as follows:

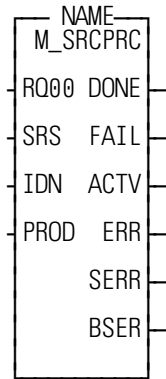
Name	Data Type	Definition
SRS	STRUCT	
.SLOT	UINT	Slot number of the SERCOS module
.RING	UINT	Ring number on the module
.SLAVE	UINT	Slave number on the ring
	END_STRUCT	

If FAIL is set, ERR or BSER will be non-zero indicating the type of error. If ERR = 128 indicating Slave Error, SERR will be non-zero indicating the type of slave error.

M_SRCPRC

Executes SERCOS procedure command function

USER/M_SERCOS



Inputs: RQ00 (BOOL) - requests execution of a procedure command function (one-shot)
SRS (STRUCT) - slot, ring, and slave number of the SERCOS slave to execute the procedure command function
IDN (UINT) - IDN number of procedure command function
PROD (BOOL) - set for Product IDN, reset for System IDN

Outputs: DONE (BOOL) - procedure command function complete
FAIL (BOOL) - procedure command function failure
ACTV (BOOL) - set while the procedure command function is active
ERR (INT) - SERCOS error*
SERR (UINT) - SERCOS slave error*
BSER (INT) - SERCOS block specific error*
*See error tables at end of the M_SRCWTL function block section.

```
<<INSTANCE NAME>>:M_SRCPRC(RQ00 := <<BOOL>>, SRS :=  
  <<MEMORY AREA>>, IDN := <<UINT>>, PROD := <<BOOL>>, DONE  
  => <<BOOL>>, ACTV => <<BOOL>>, ERR => <<INT>>, SERR =>  
  <<UINT>>, BSER => <<INT>>);
```

The M_SRCPRC function block executes a SERCOS procedure command function for a single SERCOS slave. The RQ00 input of this function block should be one-shot to initiate the procedure command function. While the procedure command function is executing within the SERCOS slave, the ACTV output will be set. If the procedure command function completes without error, the DONE output will be set and the ACTV output will be reset. If the procedure command function fails, the FAIL output will be set and the ACTV output will be reset. The DONE or FAIL output will remain set until the RQ00 input is one-shot again.

The SRS input is used to indicate which SERCOS slave is to execute the procedure command function. Slot, ring, and slave are used instead of an axis number so that this function can be used in phase 2 initialization if desired. The SRS structure must be declared as shown in the following table:

Name	Data Type	Definition
SRS	STRUCT	
.SLOT	UINT	Slot number of the SERCOS module
.RING	UINT	Ring number on the module
.SLAVE	UINT	Slave number on the ring
	END_STRUCT	

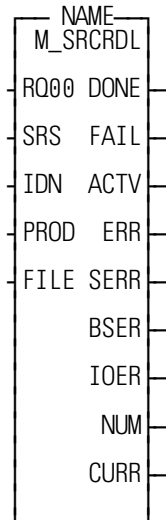
If FAIL is set, ERR or BSER will be non-zero indicating the type of error.

If ERR = 128 indicating slave error, SERR will be non-zero indicating the type of slave error.

M_SRCRDL

Reads SERCOS IDNs

USER/M_SERCOS



Inputs: RQ00 (BOOL) - requests execution (one-shot)
SRS (STRUCT) - slot, ring, and slave number
IDN (UINT) - IDN number that will return a list of IDNs
PROD (BOOL) - set for Product IDN, reset for System IDN
FILE (STRING [80]) - filename of the file to save to

Outputs: DONE (BOOL) - execution complete
FAIL (BOOL) - execution failed
ACTV (BOOL) - set while executing
ERR (INT) - SERCOS error*
SERR (UINT) - SERCOS slave error*
BSER (INT) - SERCOS block specific error*
IOER (INT) - I/O function block error (See Appendix B in the online help.)
NUM (UINT) - number of IDNs in the list
CURR (UINT) - current member being read

*See error tables at end of the M_SRCWTL function block section.

```
<<INSTANCE NAME>>:M_SRCRDL(RQ00 := <<BOOL>>, SRS :=  
<<MEMORY AREA>>, IDN := <<UINT>>, PROD := <<BOOL>>, FILE  
:= <<STRING>>, DONE => <<BOOL>>, FAIL => <<BOOL>>, ACTV =>  
<<BOOL>>, ERR => <<INT>>, SERR => <<UINT>> BSER => <<INT>>,  
IOER => <<INT>>, NUM => <<UINT>>, CURR => <<UINT>>);
```

The M_SRCRDL function block reads a list of up to 400 IDNs and saves the list to the PiC RAMDISK or workstation as an ASCII file along with the name, units, and operation data limits for each IDN in the list. Each IDN appears in a single line in the file. The data for each IDN is separated by tabs. This function block can be used in conjunction with M_SRCWTL to read and write lists of IDNs to and from a SERCOS slave.

The IDN number specified with the IDN and PROD inputs must return a list of IDNs in order to use this function block.

The RQ00 input must be one-shot. While the function block is reading the list of IDNs, the ACTV output will be set. If the read completes without error, the DONE output will be set and the ACTV output will be reset. If an error occurs during reading, the FAIL output will be set and the ACTV output will be reset. The DONE or FAIL output will remain set until the RQ00 input is one-shot again.

The NUM output indicates the total number of IDNs that exist in the list being read. The CURR output indicates the current member of the list being read and will range from 0 to NUM.

The SRS input is used to indicate from which SERCOS slave the list of IDNs will be read. Slot, ring, and slave are used instead of an axis number so that this function block can be used in phase 2 initialization if desired. The SRS structure must be declared as follows:

Name	Data Type	Definition
SRS	STRUCT	
.SLOT	UINT	Slot number of the SERCOS module
.RING	UINT	Ring number on the module
.SLAVE	UINT	Slave number on the ring
	END_STRUCT	

FILE is a string containing the full file specification of the file in which the list of IDNs is saved. This string must be terminated by the null character \$00, (i.e. RAMDISK:\IDNFILE.DAT\$00).

If FAIL is set ERR, BSER, or IOER will be non-zero indicating the type of error. If ERR = 128 indicating Slave Error, SERR will be non-zero indicating the type of slave error.

M_SRCWT

Writes and reads SERCOS IDNs

USER/M_SERCOS

NAME	
M_SRCWT	
RQ00	DONE
SRS	FAIL
IDNA	ACTV
P_A	ERR
WODA	SERR
IDNB	BSER
P_B	FIDN
WODB	RODA
IDNC	RODB
P_C	RODC
WODC	RODD
IDND	RODE
P_D	
WODD	
IDNE	
P_E	
WODE	

Inputs:

- RQ00 (BOOL) - requests execution (one-shot)
- SRS (STRUCT) - slot, ring, and slave number
- IDNA (UINT) - number of first IDN to write
- P_A (BOOL) - set for Product IDN, reset for System IDN
- WODA (REAL) - value of operation datum for IDNA
- IDNB (UINT) - number of second IDN to write
- P_B (BOOL) - set for Product IDN, reset for System IDN
- WODB (REAL) - value of operation datum for IDNB
- IDNC (UINT) - number of third IDN to write
- P_C (BOOL) - set for Product IDN, reset for System IDN
- WODC (REAL) - value of operation datum for IDNC
- IDND (UINT) - number of fourth IDN to write
- P_D (BOOL) - set for Product IDN, reset for System IDN
- WODD (REAL) - value of operation datum for IDND
- IDNE (UINT) - number of fifth IDN to write
- P_E (BOOL) - set for Product IDN, reset for System IDN
- WODE (REAL) - value of operation datum for IDNE

Outputs:

- DONE (BOOL) - set when the writes and reads are complete
- FAIL (BOOL) - set if write or read fails
- ACTV (BOOL) - set when operation is in process
- ERR (INT) - SERCOS error*
- SERR (UINT) - SERCOS slave error*
- BSER (INT) - SERCOS block specific error*
- FIDN (UINT) - the IDN the operation failed on
- RODA (REAL) - value of operation datum read back from IDNA
- RODB (REAL) - value of operation datum read back from IDNB
- RODC (REAL) - value of operation datum read back from IDNC
- RODD (REAL) - value of operation datum read back from IDND
- RODE (REAL) - value of operation datum read back from IDNE

*See error tables at end of M_SRCWTL function block section.

```

<<INSTANCE NAME>>:M_SRCWT(RQ00 := <<BOOL>>, SRS :=
  <<MEMORY AREA>>, IDNA := <<UINT>>, P_A := <<BOOL>>, WODA
  := <<REAL>>, IDNB := <<UINT>>, P_B := <<BOOL>>, WODB :=
  <<REAL>>, IDNC := <<UINT>>, P_C := <<BOOL>> WODC :=
  <<REAL>>, IDND := <<UINT>>, P_D := <<BOOL>>, WODD :=
  <<REAL>>, IDNE := <<UINT>>, P_E := <<BOOL>>, WODE :=
  <<REAL>>, DONE => <<BOOL>>, FAIL => <<BOOL>>, ACTV =>
  <<BOOL>>, ERR => <<INT>>, SERR => <<UINT>> BSER => <<INT>>,
  FIDN => <<UINT>>, RODA => <<REAL>>, RODB => <<REAL>>,
  RODC => <<REAL>>, RODD => <<REAL>>, RODE => <<REAL>>);

```

The M_SRCWT function block writes and reads up to five SERCOS IDNs.

The M_SRCWT function block will write and read back operation data to a maximum of five IDNs on a SERCOS slave. The operation data for each IDN is written and read once when the RQ00 input is energized.

The IDNA through IDNE inputs can be used or left blank. When the RQ00 input transitions from off to on, the attributes of each IDN are read and saved in the function block. These attributes are used to scale the data at the input to the correct units for the SERCOS slave. After the attributes are read the operation data is written and read back again to verify that the write was successful. While this process is happening, the ACTV output will remain set. If the process completes without error, the DONE output will be set and the ACTV output will be reset. If an error occurs, the FAIL output will be set and the ACTV output will be reset.

The RQ00 input must be one-shot each time you wish to write data to the SERCOS slave. A second request cannot be made while the first one is still active. If this happens, the second request will be ignored.

The SRS input is used to indicate which SERCOS slave to write to. Slot, ring, and slave are used instead of an axis number so that this function can be used in phase 2 initialization if desired. The SRS structure must be declared as follows:

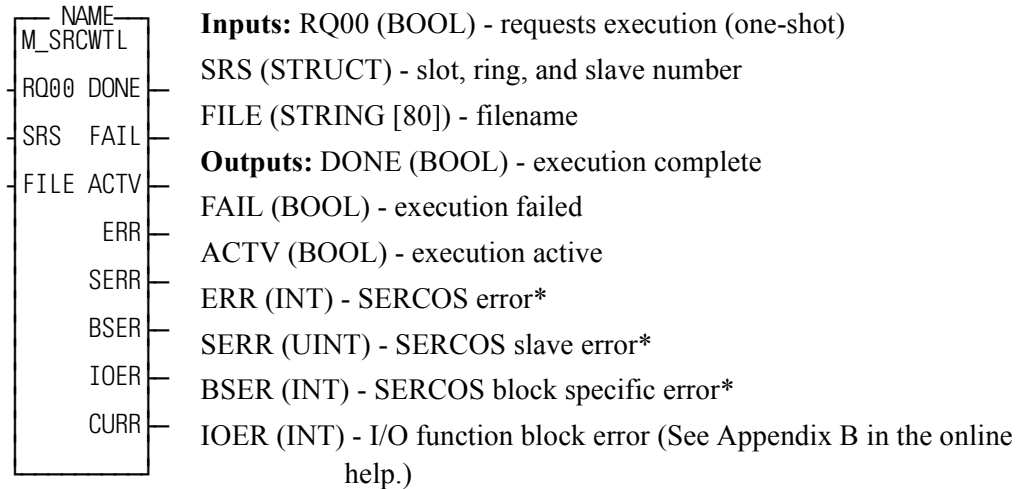
Name	Data Type	Definition
SRS	STRUCT	
.SLOT	UINT	Slot number of the SERCOS module
.RING	UINT	Ring number on the module
.SLAVE	UINT	Slave number on the ring
	END_STRUCT	

If FAIL is set, ERR or BSER will be non-zero indicating the type of error. If ERR = 128 indicating Slave Error, SERR will be non-zero indicating the type of slave error.

M_SRCWTL

Writes SERCOS IDNs

USER/M_SERCOS



*See error tables at end of the this function block section.

```
<<INSTANCE NAME>>:M_SRCWTL(RQ00 := <<BOOL>>, SRS := <<MEM-  
ORY AREA>>, FILE := <<STRING>>, DONE => <<BOOL>>, FAIL =>  
<<BOOL>>, ACTV => <<BOOL>>, ERR => <<INT>>, SERR => <<UINT>>  
BSER => <<INT>>, IOER => <<INT>>, CURR => <<UINT>>);
```

The M_SRCWTL function block writes a list of SERCOS IDNs.

The M_SRCWTL function block reads a list of IDNs from an ASCII file on the PiC RAMDISK or workstation and writes the operation data from the list to a SERCOS slave. The ASCII file must be of the same format used for the M_SRCRDL function block. M_SRCWTL can be used in conjunction with M_SRCRDL to read and write lists of IDNs to and from a SERCOS slave.

The RQ00 input to this function must be one-shot. While the function block is writing the list of IDNs, the ACTV output will be set. If the write completes without error, the DONE output will be set and the ACTV output will be reset. If an error occurs during the write, the FAIL output will be set and the ACTV output will be reset. The DONE or FAIL output will remain set until the RQ00 input is one-shot again.

The CURR output indicates the current IDN being written to the SERCOS slave. This will continually update while the function block is active.

The SRS input is used to indicate which SERCOS slave the list of IDNs will be written to. Slot, ring, and slave are used instead of an axis number so that this function can be used in phase 2 initialization if desired. The SRS structure must be declared as shown in the following table:

Name	Data Type	Definition
SRS	STRUCT	
.SLOT	UINT	Slot number of the SERCOS module
.RING	UINT	Ring number on the module
.SLAVE	UINT	Slave number on the ring
	END_STRUCT	

FILE is a string containing the full file specification of the file in which the list of IDNs is saved. This string must be terminated by the null character \$00 (i.e. RAM-DISK:\IDNFILE.DAT\$00).

If FAIL is set ERR, BSER, or IOER will be non-zero indicating the type of error. If ERR = 128 indicating Slave Error, SERR will be non-zero indicating the type of slave error.

M_SERCOS Function Block Errors

There are three types of error outputs that can appear on the *M_SERCOS* function blocks. They are described in the three tables that follow.

ERR Output

Table 1 contains the list of SERCOS errors that can appear at the ERR output of the *M_SERCOS* function blocks.

Table 1 - List of ERR Codes

Err # Description

- 0 No error
- 1 IDN queue was busy when called.
- 2 Quantity specified in the .AVAIL structure member is not large enough for received data.
- 3 Axis is not initialized, is not a SERCOS axis, or the slot/ring/slave specification is incorrect.
- 4 Invalid data in DATA input structure
- 5 Error reset function could not be completed.
- 6 SERCOS ring 1 busy*
- 7 SERCOS ring 2 busy*
- 8 SERCOS ring 1 configuration size error**
- 9 SERCOS ring 2 configuration size error**
- 10 Function block enabled while already in process
- 11 Bit 3 or bit 8 set in the procedure command acknowledgment (data status) Either operation data invalid or procedure command error
- 12 Not enough pool memory available
- 13 Change bit in status word was zero after reference complete.
- 14 The IDN queue was cleared during an IDN transfer, typically caused by calling the SC_INIT function while an IDN is being read or written.
- 15 SERCOS module is unavailable for IDN transfer because the phase-to-phase transition in progress is between phase 2 and phase 4.
- 16 Slave response timed out
- 17 The SERCOS module did not receive an expected AT response. SERCOS cable may be disconnected.
- 18 Number of SERCOS slots equals zero.
- 19 The SERCOS module did not receive an expected MDT response. SERCOS cable may be disconnected.
- 20 Phase 0 detected that the ring is not complete. The optic cable could be open or drive turned off.
- 21 The SERCOS module firmware is outdated for the features requested from a newer version of the motion library.
- 22 The SERCOS module firmware is a newer version and the motion library is outdated and unable to interface.
- 23 The data (user function) is outdated for the features requested from the library or the SERCOS module firmware.
- 24 The data is a newer version and the library is unable to interface.
- 25 A two-ring SERCOS module was specified in SERCOS setup but the module is a one-ring SERCOS module.
- 30 The drive status word (bit 13=1) indicates an error.

- 31 An E-stop condition exists for this axis in the PiC900.
- 32 Incorrect phase number, contact G&L Motion Control
- 33 Incorrect address error, contact G&L Motion Control
- 34 Incorrect AT number error, contact G&L Motion Control
- 35 Variable 48 is set to 1 and you attempt to close the loop
- 36 OPTN input is invalid.
- 48 Service channel not ready when attempt to send/receive non-cyclic data
- 49 No data to send or receive
- 50 The value of the .SIZE member of the TASK input structure does not match the byte count in the SERCOS module.
- 51 The value of the .SIZE member of the MAIN input structure does not match the byte count in the SERCOS module.
- 65 Error occurred calculating when MDT should occur.
- 66 Error occurred calculating when drive data valid.
- 67 Error occurred calculating when feedback data valid.
- 68 Error occurred calculating total time required for communication cycle.
- 69 Error occurred calculating cyclic data memory for SERCON processor.
- 70 Error occurred calculating cyclic data memory for internal memory map.
- 71 Error occurred calculating service channel memory map.
- 72 Incorrect ring error, contact G&L Motion Control
- 73 Incorrect AT count error, contact G&L Motion Control
- 74 CPU on SERCOS module has too many tasks during update.
- 128 Slave error occurred. Read SERR output to identify error. The SLV output indicates the slave number.
- 136 Slave will not respond in phase 1. The SLV output indicates the slave number.
- 144 Procedure command error - The slave number can be viewed at the SLV output and the IDN number at the IDN output.

*This busy error may occur if the SC_INIT function is not one-shotted and a second store operation is attempted before the first one is done.

**This size error will occur if too many IDNs are defined in the SERCOS setup data.

SERR Output

Table 2 contains the list of slave errors that can appear at the SERR output of M_*SERCOS* function blocks.

Table 2 - List of SERR Error Codes

SERR #	Description
0	No error
4097	This IDN does not exist.
4105	The data for this IDN may not be accessed.
8193	The name does not exist
8194	The name transmission is too short
8195	The name transmission is too long
8196	The name may not be changed
8197	The name is write-protected
12290	The attribute transmission is too short
12291	The attribute transmission is too long
12292	The attribute may not be changed
12293	The attribute is write-protected at this time
16385	The units do not exist
16386	The units transmission is too short
16387	The units transmission is too long
16388	The units may not be changed
16389	The units are write-protected at this time
20481	The minimum value does not exist
20482	The minimum value transmission is too short
20483	The minimum value transmission is too long
20484	The minimum value may not be changed
20485	The minimum value is write-protected
24577	The maximum value does not exist
24578	The maximum value transmission is too short
24579	The maximum value transmission is too long
24580	The maximum value may not be changed
24581	The maximum value is write-protected
28674	The data is too short.
28675	The data is too long
28676	The data may not be changed.
28677	The data is write-protected at this time.
28678	The data is smaller than the minimum value.
28679	The data is larger than the maximum value.
28680	The bit pattern for this IDN is invalid.

BSER Output

Table 3 contains the list of block specific errors that can appear at the BSER output of M_ *SERCOS* function blocks.

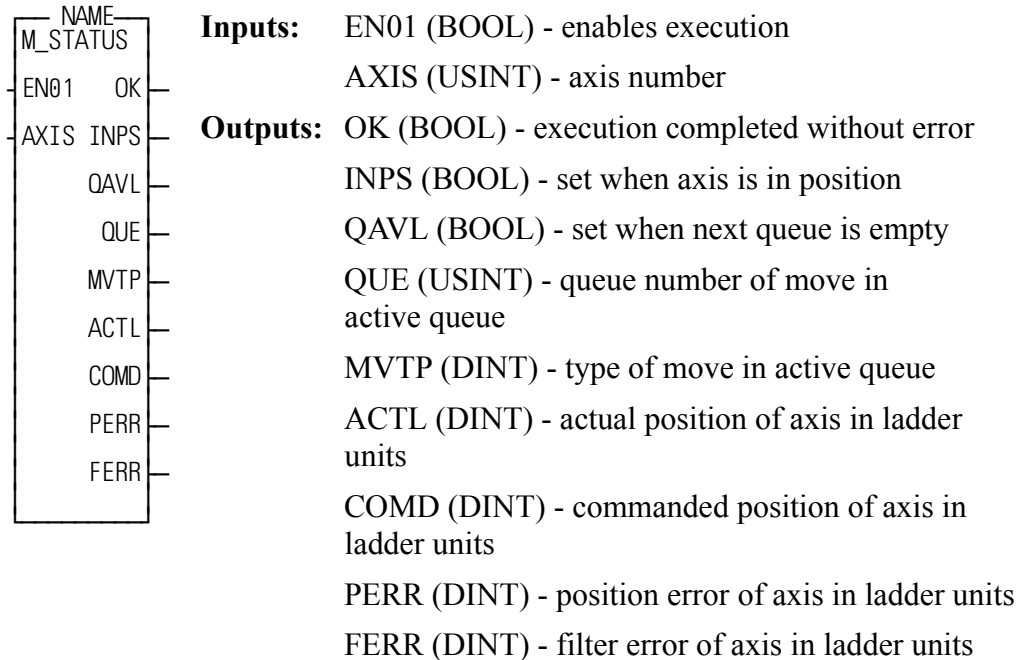
Table 3 - Block Specific Error Codes

BSER #	Description
0	No error
1	Request to execute but not in phase 2 or 4
2	IDN is a procedure command
3	Data is variable length
4	Data is reserved
5	IDN is not a procedure command

M_STATUS

Return Axis Data

USER/M_DATA



```
<<INSTANCE NAME>>:M_STATUS(EN01 := <<BOOL>>, AXIS :=  
  <<USINT>>, OK => <<BOOL>>, INPS => <<BOOL>>, QAVL =>  
  <<BOOL>>, QUE => <<USINT>>, MVTP => <<DINT>> ACTL =>  
  <<DINT>>, COMD => <<DINT>>, PERR => <<DINT>>, FERR =>  
  <<DINT>>);
```

This function block obtains information for a digitizing, time, or closed loop axis. It returns the in position flag, the queue available flag, the active queue number, the active move type, the actual position, the commanded position, the position error, and the filter error for the axis. This function block should be enabled every scan.

The input at AXIS determines which axis the output information is for.

The INPS output is set whenever the following error of the axis is within the in position limit entered in servo setup. It will be reset while the axis is in motion.

The QAVL output is set whenever the next queue or both the next and active queues are empty. When set it means another move can be put in the axis queue.

The QUE output holds the queue number of the move in the active queue. The queue number is assigned to each move when the move function is enabled. If no moves are active, the QUE number will be 0.

The MVTP output holds the type of the move in the active queue. If no move is active, this will be 0. The moves types are defined below:

MVTP	Description
11	POSITION
12	DISTANCE
14	VEL_STRT
16	FAST_REF or LAD_REF
18	RATIOPRO
20	RATIOSYN or RATIO_GR
22	RATIOCAM
23	RATIOSLP
24	RATIO_RL

The ACTL output holds the actual position of the axis in ladder units.

The COMD output holds the commanded position of the axis in ladder units.

The PERR output holds the position error of the axis in ladder units.

The FERR output holds the filter error of the axis in ladder units.

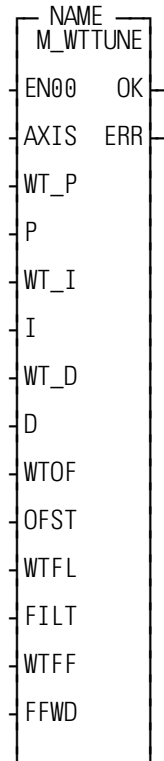
This function block can be used for a digitizing axis, a time axis, or a closed loop axis. If used for a digitizing axis only the ACTL and COMD outputs are used and there is no need to enter variables for the INPS, QAVL, QUE, MVTP, PERR, or FERR outputs. If used for a time axis, only the ACTL output is used and there is no need to enter variables for the INPS, QAVL, QUE, MVTP, COMD, PERR, and FERR outputs.

The OK output will not be set if the axis has not been initialized.

M_WTTUNE

Writes tuning parameters

USER/M_DATA



Inputs: EN00 (BOOL) - enables execution
AXIS (USINT) - identifies axis
WT_P (BOOL) - enables write of proportional gain
P (DINT) - proportional gain
WT_I (BOOL) - enables write of integral gain
I (DINT) - integral gain
WT_D (BOOL) - enables write of derivative gain
D (DINT) - derivative gain
WTOF (BOOL) - enables write of analog output offset
OFST (DINT) - analog output offset
WTFL (BOOL) - enables write of slow speed filter value
FILT (DINT) - slow speed filter value
WTFF (BOOL) - enables write of feedforward percentage
FFWD (DINT) - feedforward percentage

Outputs: OK (BOOL) - execution complete
ERR (INT) - error number

```
<<INSTANCE NAME>>:M_WTTUNE(EN00 := <<BOOL>>, AXIS :=  
  <<USINT>>, WT_P := <<BOOL>>, P := <<DINT>>, WT_I := <<BOOL>>, I  
  := <<DINT>>, WT_D := <<BOOL>> D := <<DINT>>, WTOF := <<BOOL>>,  
  OFST := <<DINT>>, WTFL := <<BOOL>>, FILT := <<DINT>>, WTFF :=  
  <<BOOL>>, FFWD := <<DINT>>, OK => <<BOOL>>, ERR => <<INT>>);
```

This function block allows you to write all six tuning parameters from the TUNE-WRIT function in a single function.

This function block requires the numeric processor or a 486 DX processor.

The EN00 input of this function should be set every scan.

The AXIS input identifies which axis to write data to. It must be between 1 and 16 or between 101 and 116, inclusive.

Note: LU = ladder units, MIN = minutes, LUFU = ladder units of following error.

When the WT_P input is set, the proportional gain of AXIS will be changed to the value entered at P. P is in LU / MIN / LUFÉ and must be between 0 and 20000.

When the WT_I input is set, the integral gain of AXIS will be changed to the value entered at I. I is in LU / MIN / LUFÉ * MIN. I must be from 0 to 32000.

When the WT_D input is set, the derivative gain of AXIS will be changed to the value entered at D. D is in LU / MIN / LUFÉ / MIN.

When the WTOF input is set, the analog output offset voltage of AXIS will be changed to the value entered at OFST. OFST must be from -10000 to +10000 millivolts.

When the WTFL input is set, the slow speed filter value of AXIS will be changed to the value entered at FILT. FILT must be from 0 to 10000 milliseconds.

When the WTFF input is set, the feedforward percentage of AXIS will be changed to the value entered at FFWD. FFWD must be from 0 to 100%.

The WT_P, WT_I, WT_D, WTOF, WTFL and WTFF inputs can be one-shot. The parameters will remain changed until the axis is re-initialized or until this function block or the TUNEWRIT function is called again for AXIS.

The OK output will be set if the function executes without error. If an error occurs, OK will not be set and ERR will hold a number describing the error that occurred. A listing of errors is shown below:

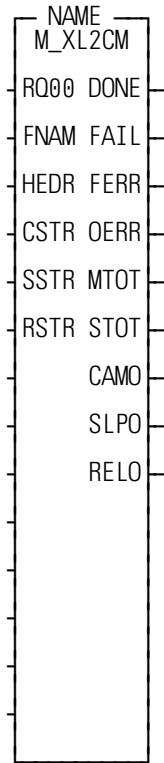
ERR	Description
------------	--------------------

0	No error
1	Tried to change P for AXIS number that was not initialized or is out of range
3	Data for P is out of range or can not be calculated
101	Tried to change I for AXIS number that was not initialized or is out of range
103	Data for I is out of range or can not be calculated
201	Tried to change D for AXIS number that was not initialized or is out of range
203	Data for D is out of range or can not be calculated
301	Tried to change OFST for AXIS number that was not initialized or is out of range
303	Data for OFST is out of range or can not be calculated
401	Tried to change FILT for AXIS number that was not initialized or is out of range
403	Data for FILT is out of range or can not be calculated
501	Tried to change FFWD for AXIS number that was not initialized or is out of range
503	Data for FFWD is out of range or can not be calculated

M_XL2CM

Excel to Cam Profile

USER/M_XL2CM



Inputs: RQ00 (BOOL) - one-shot to start conversion
FNAM (STRING) - file name of csv file that is to be converted
HEDR (STRUCT) - structure containing information about the profile such as master axis, scaling
CSTR (STRUCT) - array of structures used to hold RATIOCAM profile data
SSTR (STRUCT) - array of structures used to hold RATIOSLP profile data
RSTR (STRUCT) - array of structures used to hold RATIO_RL profile data

Outputs: DONE (BOOL) - file conversion completed without error
FAIL (BOOL) - file conversion failed
FERR (INT) - error number from OPEN, READ, or CLOSE functions
OERR (INT) - error number from operation of function
MTOT (DINT) - Total master distance moved in the profile (feedback units)
STOT (DINT) - Total slave distance moved in the profile (feedback units)
CAMO (BOOL) - set if the profile type is RATIOCAM
SLPO (BOOL) - set if the profile type is RATIOSLP
RELO (BOOL) - set if the profile type is RATIO_RL

<<INSTANCE NAME>>:M_XL2CM(RQ00 := <<BOOL>>, FNAM := <<STRING>>, HEDR := <<STRUCT>>, CSTR := <<STRUCT>>, SSTR :=

```

<<STRUCT>>, RSTR := <<STRUCT>>, DONE => <<BOOL>>, FAIL =>
<<BOOL>>, FERR => <<INT>>, OERR => <<INT>>, MTOT => <<DINT>>,
STOT => <<DINT>>, CAMO => <<BOOL>>, SLPO => <<BOOL>>, RELO
=> <<BOOL>>);

```

This Excel to Cam application specific function block is used to convert an ASCII CSV file in the correct format to a data structure that can be called directly by RATIOCAM, RATIOSLP or RATIO_RL.

The CSV files must be created using the Excel spreadsheet that is included with this function (the default spreadsheet is called Cam Profile Design Tool.XLS). Using Excel, enter profile information such as master axis, slave axis, and scaling. You then select the profile type (RATIOCAM, RATIOSLP or RATIO_RL) and enter the point pairs. The spreadsheet allows you to see a graph of the profile. Once you have entered the data, you export the appropriate data to a CSV file. Instructions on how to do this are included with the spreadsheet. The resulting CSV file is copied to the control's RAMDISK so this function can read it.

The string specified at the FNAM input must include the entire file path (e.g. RAMDISK:File.CSV\$00).

The HEDR structure contains the following members, which are all read from the csv file:

```

.VERSION STRING[8] - the version number
.COMMENT STRING[40] - the comment
.SLAVE_AXIS USINT - axis number of the slave
.SLAVE_LABEL STRING[32] - slave description
.MASTER_AXIS USINT - axis number of the master .MASTER_LABEL
STRING[32] - master description
.SLAVE_FU DINT - slave scaling info
.SLAVE_LU DINT - slave scaling info
.MASTER_FU DINT - master scaling info
.MASTER_LU DINT - master scaling info
.PROFILE_TYPE STRING[32] - cam, slope or real .NUMBER_OF_SEGS INT -
number of segments in the profile

```

The CSTR, SSTR and RSTR inputs are all arrays of structures. The structure format is the same as that specified by the associated function: RATIOCAM, RATIOSLP and RATIO_RL respectively. In the example program each data structure is declared to be the maximum size so no data overruns occur. You can make the structures smaller depending on your application.

Either the DONE or the FAIL output will be set when the conversion is complete. If the FAIL output is set, either the FERR or the OERR output will indicate an error number. The FERR output is used to indicate an error from the OPEN, READ or CLOSE function and the error numbers can be found in Appendix B of the online help. The OERR output indicates an error inside the function and the codes are as follows:

The MTOT and STOT outputs indicate the total master and slave distance moved over the entire profile.

The CAMO, SLPO and RELO outputs are used to indicate the profile type. Only one of them will be set when the DONE is set. CAMO indicates RATIOCAM, SLPO indicates RATIOSLP and RELO indicates RATIO_RL. These outputs are used in the example program to decide which function to call after the profile data is read.

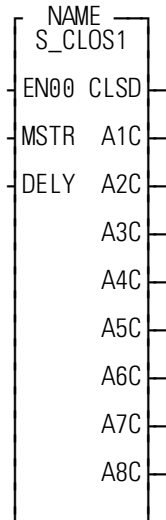
OERR DESCRIPTION:

ERR	Description
7001	Invalid version number in csv file
7002	Invalid comment line
7003	Invalid slave axis number and label
7004	Invalid master axis number and label
7005	Invalid slave scaling
7006	Invalid master scaling
7007	Invalid profile type found in file
7008	Invalid number of segments
7009	Invalid column labels line in csv file

S_CLOS1

Close Loop on SERCOS Servo Axes 1 to 8

USER/S_ASFB



Inputs: EN00 (BOOL) - enables execution
MSTR (BOOL) - machine start input
DELY (TIME) - amount of time that will elapse after a positive transition of MSTR until the loops will be closed

Outputs: CLSD (BOOL) - one or more of axes 1 to 8 have their position loops closed
A1C (BOOL) - set when the loop is closed on axis 1
A2C (BOOL) - set when the loop is closed on axis 2
A3C (BOOL) - set when the loop is closed on axis 3
A4C (BOOL) - set when the loop is closed on axis 4
A5C (BOOL) - set when the loop is closed on axis 5
A6C (BOOL) - set when the loop is closed on axis 6
A7C (BOOL) - set when the loop is closed on axis 7
A8C (BOOL) - set when the loop is closed on axis 8

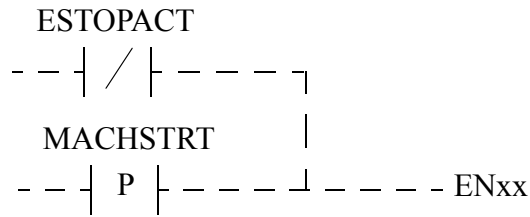
```
<<INSTANCE NAME>>:S_CLOS1(EN00 := <<BOOL>>, MSTR :=  
  <<BOOL>>, DELAY := <<TIME>>, CLSD => <<BOOL>>, A1C =>  
  <<INT>>, A2C => <<BOOL>>, A3C => <<BOOL>>, A4C => <<BOOL>>,  
  A5C => <<BOOL>>, A6C => <<BOOL>>, A7C => <<BOOL>>, A8C =>  
  <<BOOL>>);
```

This function block is a replacement for M_CLOS1 for SERCOS axes. It is not for analog controlled axes.

This function block is used to reset the E-stop, C-stop, and programming errors on SERCOS servo axes 1 through 8 when the machine start input is pulsed. It also sends a class one diagnostics fault reset to the SERCOS drive. It closes the loop on SERCOS servo axes 1 through 8 after the machine start input is pulsed and a programmable time delay has elapsed. If there are no E-stop faults and the drive is enabled the loop will be closed and the closed output will be energized.

This function block can be enabled every scan. If the enable input changes from ON to OFF during the time delay after machine start, the function block will abort the time delay and not close the position loops.

If there are conditions that should abort the sequence to close the position loops (such as an electrical E-stop condition during the time delay), then the enable should include both the positive transition of the machine start input and the current state of the electrical E-stop status as shown below.



The reason for these two input conditions is to provide the enable at the start of the time delay (with the P contact of the machine start signal) and to maintain the enable during the time delay as needed (with the NC contact for the electrical E-stop condition).

The MMC example applications located on the Applications CD (in the examples sub-directory) illustrate the recommended ladder logic for the E-stop handling of the S_CLOS1 application. Please refer to MMC4_SOI.LDO for an example of S_CLOS1.

The machine start input must go through a positive transition (off to on) to reset the errors and close the loop.

The time DELY is normally in the range from 500 ms to 2 seconds.

On a positive transition of MSTR, this function will send a procedure command to the SERCOS drive to reset class one diagnostic errors on axes 1 through 8.

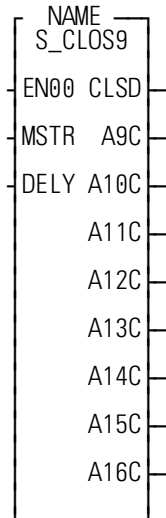
The positive transition of MSTR enables a timer with a preset time of DELY. After DELY has elapsed, all E-stop, C-stop, and programming errors are reset on axes 1 through 8. If the E-stops are reset and the SERCOS drive is enabled, the loops will be closed on axes 1 to 8. CLSD will be energized if one or more of axes 1 to 8 have their position loops closed.

If an E-stop fault occurs on an axis 1 to 8, its loop closed output (A1 to A8) will be dropped. CLSD is true as long as one or more of axes 1 to 8 have their position loops closed.

S_CLOS9

Close Loop on SERCOS Servo Axes 9 to 16

USER/S_ASFB



Inputs: EN00 (BOOL) - enables execution
MSTR (BOOL) - machine start input
DELY (TIME) - amount of time that will elapse after a positive transition of MSTR until the loops will be closed

Outputs: CLSD (BOOL) - one or more of axes 9 to 16 have their position loops closed
A9C (BOOL) - set when the loop is closed on axis 9
A10C (BOOL) - set when the loop is closed on axis 10
A11C (BOOL) - set when the loop is closed on axis 11
A12C (BOOL) - set when the loop is closed on axis 12
A13C (BOOL) - set when the loop is closed on axis 13
A14C (BOOL) - set when the loop is closed on axis 14
A15C (BOOL) - set when the loop is closed on axis 15
A16C (BOOL) - set when the loop is closed on axis 16

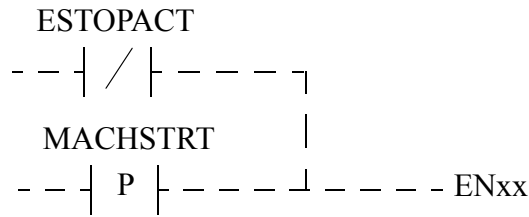
```
<<INSTANCE NAME>>:S_CLOS9(EN00 := <<BOOL>>, MSTR :=  
<<BOOL>>, DELAY := <<TIME>>, CLSD => <<BOOL>>, A9C =>  
<<INT>>, A10C => <<BOOL>>, A11C => <<BOOL>>, A12C =>  
<<BOOL>>, A13C => <<BOOL>>, A14C => <<BOOL>>, A15C =>  
<<BOOL>>, A16C => <<BOOL>>);
```

This function block is a replacement for M_CLOS9 for SERCOS axes. It is not for analog controlled axes.

This function block is used to reset the E-stop, C-stop, and programming errors on SERCOS servo axes 9 through 16 when the machine start input is pulsed. It also sends a class one diagnostics fault reset to the SERCOS drive. It closes the loop on SERCOS servo axes 9 through 16 after the machine start input is pulsed and a programmable time delay has elapsed. If there are no E-stop faults and the drive is enabled the loop will be closed and the closed output will be energized.

This function block can be enabled every scan. If the enable input changes from ON to OFF during the time delay after machine start, the function block will abort the time delay and not close the position loops.

If there are conditions that should abort the sequence to close the position loops (such as an electrical E-stop condition during the time delay), then the enable should include both the positive transition of the machine start input and the current state of the electrical E-stop status as shown below.



The reason for these two input conditions is to provide the enable at the start of the time delay (with the P contact of the machine start signal) and to maintain the enable during the time delay as needed (with the NC contact for the electrical E-stop condition).

The MMC example applications located on the Applications CD (in the examples sub-directory) illustrate the recommended ladder logic for the E-stop handling of the S_CLOSx application. Please refer to MMC4_SOI.LDO for an example of S_CLOSx.

The machine start input must go through a positive transition (off to on) to reset the errors and close the loop.

The time DELY is normally in the range from 500 ms to 2 seconds.

On a positive transition of MSTR, this function will send a procedure command to the SERCOS drive to reset class one diagnostic errors on axes 9 through 16.

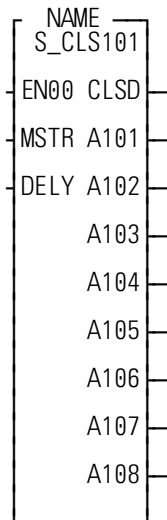
The positive transition of MSTR enables a timer with a preset time of DELY. After DELY has elapsed, all E-stop, C-stop, and programming errors are reset on axes 9 through 16. If the E-stops are reset and the SERCOS drive is enabled, the loops will be closed on axes 9 to 16. CLSD will be energized if one or more of axes 9 to 16 have their position loops closed.

If an E-stop fault occurs on an axis 9 to 16, its loop closed output (A9 to A16) will be dropped. CLSD is true as long as one or more of axes 9 to 16 have their position loops closed.

S_CLS101

Close Loop on SERCOS Servo Axes 101 to 108

USER/S_ASFB



Inputs: EN00 (BOOL) - enables execution
MSTR (BOOL) - machine start input
DELY (TIME) - amount of time that will elapse after a positive transition of MSTR until the loops will be closed

Outputs: CLSD (BOOL) - one or more of axes 101 to 108 have their position loops closed
A101 (BOOL) - set when the loop is closed on axis 101
A102 (BOOL) - set when the loop is closed on axis 102
A103 (BOOL) - set when the loop is closed on axis 103
A104 (BOOL) - set when the loop is closed on axis 104
A105 (BOOL) - set when the loop is closed on axis 105
A106 (BOOL) - set when the loop is closed on axis 106
A107 (BOOL) - set when the loop is closed on axis 107
A108 (BOOL) - set when the loop is closed on axis 108

```
<<INSTANCE NAME>>:S_CLOS101(EN00 := <<BOOL>>, MSTR :=  
<<BOOL>>, DELAY := <<TIME>>, CLSD => <<BOOL>>, A101 =>  
<<INT>>, A102 => <<BOOL>>, A103 => <<BOOL>>, A104 => <<BOOL>>,  
A105 => <<BOOL>>, A106 => <<BOOL>>, A107 => <<BOOL>>, A108 =>  
<<BOOL>>);
```

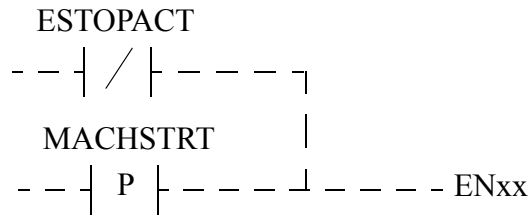
This function block is a replacement for M_CLS101 for SERCOS axes. It is not for analog controlled axes.

This function block is used to reset the E-stop, C-stop, and programming errors on SERCOS servo axes 101 through 108 when the machine start input is pulsed. It also sends a class one diagnostics fault reset to the SERCOS drive. It closes the loop on SERCOS servo axes 101 through 108 after the machine start input is pulsed and a programmable time delay has elapsed. If there are no E-stop faults

and the drive is enabled the loop will be closed and the closed output will be energized.

This function block can be enabled every scan. If the enable input changes from ON to OFF during the time delay after machine start, the function block will abort the time delay and not close the position loops.

If there are conditions that should abort the sequence to close the position loops (such as an electrical E-stop condition during the time delay), then the enable should include both the positive transition of the machine start input and the current state of the electrical E-stop status as shown below.



The reason for these two input conditions is to provide the enable at the start of the time delay (with the P contact of the machine start signal) and to maintain the enable during the time delay as needed (with the NC contact for the electrical E-stop condition).

The machine start input must go through a positive transition (off to on) to reset the errors and close the loop.

The time DELY is normally in the range from 500 ms to 2 seconds.

On a positive transition of MSTR, this function will send a procedure command to the SERCOS drive to reset class one diagnostic errors on axes 101 through 108.

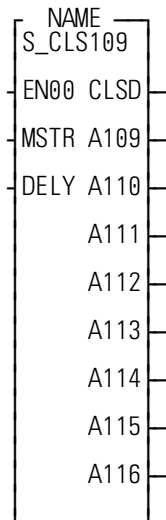
The positive transition of MSTR enables a timer with a preset time of DELY. After DELY has elapsed, all E-stop, C-stop, and programming errors are reset on axes 101 through 108. If the E-stops are reset and the SERCOS drive is enabled, the loops will be closed on axes 101 to 108. CLSD will be energized if one or more of axes 101 to 108 have their position loops closed.

If an E-stop fault occurs on an axis 101 to 108, its loop closed output (A101 to A108) will be dropped. CLSD is true as long as one or more of axes 101 to 108 have their position loops closed.

S_CLS109

Close Loop on SERCOS Servo Axes 109 to 116

USER/S_ASFB



Inputs: EN00 (BOOL) - enables execution
MSTR (BOOL) - machine start input
DELY (TIME) - amount of time that will elapse after a positive transition of MSTR until the loops will be closed

Outputs: CLSD (BOOL) - one or more of axes 109 to 116 have their position loops closed
A109 (BOOL) - set when the loop is closed on axis 109
A110 (BOOL) - set when the loop is closed on axis 110
A111 (BOOL) - set when the loop is closed on axis 111
A112 (BOOL) - set when the loop is closed on axis 112
A113 (BOOL) - set when the loop is closed on axis 113
A114 (BOOL) - set when the loop is closed on axis 114
A115 (BOOL) - set when the loop is closed on axis 115
A116 (BOOL) - set when the loop is closed on axis 116

```
<<INSTANCE NAME>>:S_CLOS109(EN00 := <<BOOL>>, MSTR :=  
  <<BOOL>>, DELAY := <<TIME>>, CLSD => <<BOOL>>, A109 =>  
  <<INT>>, A110 => <<BOOL>>, A111 => <<BOOL>>, A112 => <<BOOL>>,  
  A113 => <<BOOL>>, A114 => <<BOOL>>, A115 => <<BOOL>>, A116 =>  
  <<BOOL>>);
```

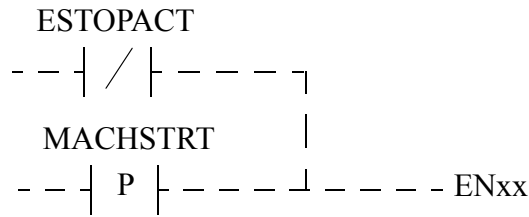
This function block is a replacement for M_CLS109 for SERCOS axes. It is not for analog controlled axes.

This function block is used to reset the E-stop, C-stop, and programming errors on SERCOS servo axes 109 through 116 when the machine start input is pulsed. It also sends a class one diagnostics fault reset to the SERCOS drive. It closes the loop on SERCOS servo axes 109 through 116 after the machine start input is pulsed and a programmable time delay has elapsed. If there are no E-stop faults

and the drive is enabled the loop will be closed and the closed output will be energized.

This function block can be enabled every scan. If the enable input changes from ON to OFF during the time delay after machine start, the function block will abort the time delay and not close the position loops.

If there are conditions that should abort the sequence to close the position loops (such as an electrical E-stop condition during the time delay), then the enable should include both the positive transition of the machine start input and the current state of the electrical E-stop status as shown below.



The reason for these two input conditions is to provide the enable at the start of the time delay (with the P contact of the machine start signal) and to maintain the enable during the time delay as needed (with the NC contact for the electrical E-stop condition).

The machine start input must go through a positive transition (off to on) to reset the errors and close the loop.

The time DELY is normally in the range from 500 ms to 2 seconds.

On a positive transition of MSTR, this function will send a procedure command to the SERCOS drive to reset class one diagnostic errors on axes 109 through 116.

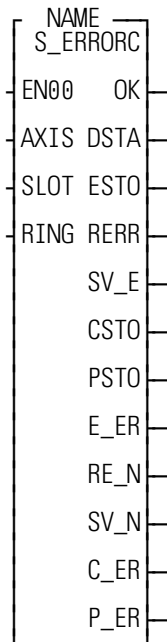
The positive transition of MSTR enables a timer with a preset time of DELY. After DELY has elapsed, all E-stop, C-stop, and programming errors are reset on axes 109 through 116. If the E-stops are reset and the SERCOS drive is enabled, the loops will be closed on axes 109 to 116. CLSD will be energized if one or more of axes 109 to 116 have their position loops closed.

If an E-stop fault occurs on an axis 109 to 116, its loop closed output (A109 to A116) will be dropped. CLSD is true as long as one or more of axes 109 to 116 have their position loops closed.

S_ERRORC

Axis Error Checking Centurion SERCOS Drives

USER/S_ASFB



Inputs: EN00 (BOOL) - enables execution
AXIS (USINT) - identifies SERCOS axis
SLOT (USINT) - slot number for the SERCOS module
RING (USINT) - ring the axis is connected to

Outputs: OK (BOOL) - execution complete
DSTA (USINT) - indicates the drive status
ESTO (BOOL) - indicates an E-stop is active when set
RERR (BOOL) - indicates a ring error
SV_E (BOOL) - indicates a slave (drive) error
CSTO (BOOL) - indicates a C-stop is active when set
PSTO (BOOL) - indicates a programming error has occurred
E_ER (WORD) - identifies E-stop errors
RE_N (INT) - identifies ring error number
SV_N (UINT) - identifies slave (drive) error number
C_ER (WORD) - identifies C-stop errors
P_ER (WORD) - identifies programming errors

```
<<INSTANCE NAME>>:S_ERRORC(EN00 := <<BOOL>>, AXIS :=  
<<USINT>>, SLOT := <<USINT>>, RING := <<USINT>>, OK =>  
<<BOOL>>, DSTA => <<USINT>>, ESTO => <<BOOL>>, RERR =>  
<<BOOL>>, SV_E => <<BOOL>>, CSTO => <<BOOL>>, PSTO =>  
<<BOOL>>, E_ER => <<WORD>>, RE_N => <<INT>>, SV_N  
=><<UINT>>, C_ER => <<WORD>>, P_ER => <<WORD>>);
```

This function block is a replacement for M_ERROR for a SERCOS axis with a Centurion drive. It is not for an analog controlled axis.

This function block is used to report errors that occur on a SERCOS servo axis. The types of errors include ring errors, drive errors, E-stop, C-stop and programming errors. These conditions may be caused by the SERCOS hardware, SERCOS drive, servo software or the ladder programming. If defined by the programmer, they will be triggered using the E-STOP or C_STOP functions. All of these errors for the defined axis are reported by this one function block.

The enable input of this function should be directly connected to the rail with a wire, causing this function block to be executed each scan.

The boolean outputs can be used as flags in the ladder to report error conditions.

The E_ER, C_ER and P_ER word outputs can be converted to HEX display by using the Module Monitor Edit View List command and inserting the variables. Alternately, they can be given an initial value of 16#0 for a hex value during animation. After monitoring them in HEX, refer to the tables in the manual of functions E_ERRORS, C_ERRORS and P_ERRORS to help identify the exact problem. The RE_N value (ring error number) value can be identified by referring to the SCR_ERR function in the Function/Function Block Reference Guide. Refer to the SERCOS drive manual for the description of errors occurring on the SV_N value (drive error number).

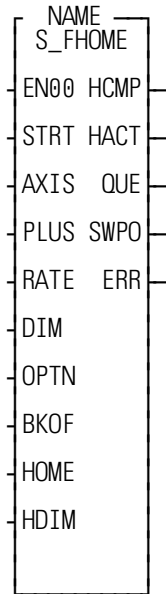
The DSTA, drive status output reflects the state the drive is currently at in decimal format. It is defined by the upper two bits of the SERCOS drive status word. The four possible states are:

- 0- drive not ready for power-up, internal checks not yet concluded successfully
- 1- drive logic OK and ready for main power
- 2 - drive ready and main power applied but not enabled, no torque to motor
- 3 - drive is enabled with torque applied to motor

S_FHOME

Performs a SERCOS Home Cycle using a Fast Reference

USER/S_ASFB



Inputs: EN00 (BOOL) - enables execution
STRT (BOOL) - enables the home cycle
AXIS (USINT) - identifies SERCOS axis
PLUS (BOOL) - indicates direction of home cycle
RATE (UDINT) - feedrate at which motion occurs (entered in LU/MIN)
DIM (DINT) - reference dimension for the nearest resolver null or the next encoder index mark when the reference switch is set (entered in LUs)
OPTN (WORD) - provides referencing options (0 or 1) 0 = no option, 1 = Ignore index or null
BKOF (BOOL) - selects backoff of reference switch option
HOME (BOOL) - selects homing after referencing option
HDIM (DINT) - home location to move after reference is complete

Outputs: HCMP (BOOL) - home cycle is complete
HACT (BOOL) - home cycle is being executed
QUE (USINT) - number of moves for queue
SWPO (DINT) - distance in feedback units (FUs) from the reference switch to the index mark of an encoder or the null of a resolver
ERR (BYTE) - report an error 1-4 if input data is invalid

```
<<INSTANCE NAME>>:S_FHOME(EN00 := <<BOOL>>, STRT :=  
<<BOOL>>, AXIS := <<USINT>>, PLUS := <<BOOL>>, RATE :=  
<<UDINT>>, DIM := <<DINT>>, OPTN := <<WORD>>, BKOF :=  
<<BOOL>>, HOME := <<BOOL>>, HDIM := <<DINT>>, HCMP =>  
<<BOOL>>, HACT => <<BOOL>>, QUE => <<USINT>>, SWPO =>  
<<DINT>>, ERR => <<BYTE>>);
```

This function block is a replacement for M_FHOME for a SERCOS axis. It is not for an analog controlled axis.

This function block performs a fast reference cycle on a SERCOS axis, followed by a homing (position) move to a designated location.

Before this function can be used, the SERCOS axis must be initialized and the position loop must be closed.

The reference cycle will cause the selected SERCOS axis to move in the designated direction until the reference switch is sensed.

In the Centurion SERCOS drive the reference switch is wired to the input number two of the selected axis on the Centurion drive. This function block uses SCA_RFIT to initialize the SERCOS drive's fast input for the reference cycle and to direct the SERCOS drive to latch the position upon that input.

When the fast input occurs, the position of the axis is latched by the hardware in the drive independent of the ladder scan.

When the reference switch is sensed, the axis will reference (assign a value) to the next index mark of an encoder or the nearest null of a resolver. After the value is assigned, the axis will decelerate to a stop and set the reference done flag.

If the HOME input is on when the reference done has been sensed, the home move will automatically be triggered to position the axis at a desired location.

If the BKOF input is on when the reference is requested, and the axis is on the reference switch, the axis will move in the opposite direction of that indicated by the PLUS input until the switch opens and then will complete the home cycle in the normal manner. If the BKOF input is not on the axis will move in the specified direction until it sees an off to on transition of the limit switch.

This function block is used to perform a fast reference, immediately followed by a position move to a selected home position. It should be executed every scan unless a home cycle will only be performed when the machine is started. In that case a normally closed contact of the output of HCMP may be used.

The SWPO output is used to determine if the reference switch location will allow for repeatable referencing. If the reference switch is not properly located in relationship to the index marker of an encoder or the null of a resolver it could possibly reference a revolution off. To prevent this, the value reported by this output should be as follows:

- For an encoder system the value of this output should be greater than 25% and less than 75% of the total counts (FUs) per revolution. Example: For 8000 FUs/ Rev, the value should be >2000 and <6000.
- For a resolver system the value of this output should be less than 25% or greater than 75% of the total counts (FUs) per revolution. Example: For 4000 FUs/ Rev, the value should be <1000 or >3000.

If the value is out of range either the reference switch will have to be moved or the transducer coupling shifted. The ERR output indicates that invalid data was entered on one of the inputs. The possible errors are listed in the following table:

ERR Description

- | | |
|---|--|
| 0 | No error |
| 1 | The queue was not empty when the reference was requested |
| 2 | An error occurred in backing off of the reference switch |
| 3 | An error occurred in referencing |
| 4 | An error occurred in homing |
| 5 | An error occurred within the SERCOS drive, either during the initialization of the SERCOS drive (its probe input) or during the monitoring of the SERCOS drive while it is referencing. The SERCOS ring and slave error values can be obtained by animating this function block after the error. |

S_IO_C

Inputs/Outputs Centurion DSM SERCOS Drive

USER/S_ASFB

NAME	
S_IO_C	
EN00	OK
AXIS	RST
FOT	ENAB
RDY1	DIN1
BRK1	DIN2
OUT1	DIN3
OUT2	DIN4
OUT3	DRVR
OUT4	DOT1
	DOT2
	DOT3
	DOT4

Inputs: EN00 (BOOL) - enables execution
AXIS (USINT) - identifies SERCOS axis
FOT (BOOL) - force the desired output states to drive outputs
RDY1 (BOOL) - desired state of the drive ready output
BRK1 (BOOL) - desired state of the drive brake output
OUT1 (BOOL) - desired state of the drive output one
OUT2 (BOOL) - desired state of the drive output two
OUT3 (BOOL) - desired state of the drive output three
OUT4 (BOOL) - desired state of the drive output four

Outputs: OK (BOOL) - execution complete
RST (BOOL) - state of the drive reset input
ENAB (BOOL) - state of the drive enable input
DIN1 (BOOL) - state of the drive input one
DIN2 (BOOL) - state of the drive input two
DIN3 (BOOL) - state of the drive input three
DIN4 (BOOL) - state of the drive input four
DRVR (BOOL) - state of the drive ready output
DRVB (BOOL) - state of the drive brake output
DOT1 (BOOL) - state of the drive output one
DOT2 (BOOL) - state of the drive output two
DOT3 (BOOL) - state of the drive output three
DOT4 (BOOL) - state of the drive output four

```
<<INSTANCE NAME>>:S_IO_C(EN00 := <<BOOL>>, AXIS := <<USINT>>,
FOT := <<BOOL>>, RDY1 := <<BOOL>>, BRK1 := <<BOOL>>, OUT1 :=
<<BOOL>>, OUT2 := <<BOOL>>, OUT3 := <<BOOL>>, OUT4 :=
<<BOOL>>, OK => <<BOOL>>, RST => <<BOOL>>, ENAB =>
<<BOOL>>, DIN1 => <<BOOL>>, DIN2 => <<BOOL>>, DIN3 =>
<<BOOL>>, DIN4 => <<BOOL>>, DRVR => <<BOOL>>, DRVB =>
```

```
<<BOOL>>, DOT1 => <<BOOL>>, DOT2 => <<BOOL>>, DOT3 =>  
<<BOOL>>, DOT4 => <<BOOL>>);
```

The SERCOS drive I/O function block provides ladder access to the digital inputs and outputs of a Centurion DSM SERCOS servo drive through the SERCOS service channel. It is not intended for analog or non-Centurion DSM SERCOS drives.

The Digital Output Override IDN P0036 must be set in the drive to use this feature. Refer to the Centurion SERCOS Drives IDN Manual.

The EN enable input should always be enabled allowing the function block to be executed every ladder scan. When the input is enabled the function block will wait until the SERCOS and Servo axis setup files have been initiated before any service channel activity begins. Once initialized, the drive ID is read to verify it is a Centurion SERCOS drive, and if so, the drive inputs and outputs are then continuously read.

When the FOT is enabled the state of the function block RDY1, BRK1, OUT1, OUT2, OUT3 and OUT4 inputs will be written to the drive. After a successful write, the function block outputs will reflect the desired outputs. While the FOT is enabled, the function block inputs are evaluated and any state change will be sent to the drive. The new drive outputs will again be reflected at the function block outputs.

NOTE

If the FOT input is disabled, all drive outputs are set to zero.

The OK output will be energized only if the axis is initialized, the drive is of type Centurion DSM SERCOS, and the drive I/O reads and writes are successful. If any failures occur, the OK output will not be energized. If a failure occurs while reading or writing I/O, the related function block outputs are set to zero.

NOTE

If an I/O read or write failure occurs, the related function block outputs may not reflect the actual state of the drive inputs and outputs

S_LHOME

Perform a SERCOS Home Cycle using a Ladder Reference

USER/S_ASFB

NAME		
S_LHOME		
EN00	HCMP	Inputs: EN00 (BOOL) - enables execution
STRT	HACT	STRT (DINT) - move from reference switch and move back to home position after referencing.
AXIS	QUE	AXIS (USINT) - identifies SERCOS axis
PLUS	SWPO	PLUS (BOOL) - indicates direction of home cycle
RATE	ERR	RATE (UDINT) - feedrate at which motion occurs (entered in LU/MIN)
DIM		DIM (DINT) - reference dimension for the nearest resolver null or the next encoder index mark when the reference switch is set (entered in LUs)
OPTN		OPTN (WORD) - provides referencing options (0 or 1) 0 = No option, 1 = Ignore index or null
BKOF		BKOF (BOOL) - selects backoff of reference switch option
HOME		HOME (BOOL) - selects homing after referencing option
HDIM		HDIM (DINT) - home location to move to after reference is complete
RFSW		RFSW (BOOL) - references switch on axis
		Outputs: HCMP (BOOL) - home cycle is complete
		HACT (BOOL) - home cycle is being executed
		QUE (USINT) - number of move for queue
		SWPO (DINT) - distance in feedback unit (FUs) from the reference switch to the index mark of an encoder or the null of a resolver
		ERR (BYTE) - report an error 1-4 if input data is invalid

```
<<INSTANCE NAME>>:S_LHOME(EN00 := <<BOOL>>,
  STRT := <<DINT >> AXIS := <<USINT>>, PLUS := <<BOOL>>, RATE :=
  <<UDINT>>, DIM := <<DINT>>, OPTN := <<WORD>>, BKOF :=
  <<BOOL>>, HOME := <<BOOL>>, HDIM := <<DINT>>, RFSW :=
  <<BOOL>>, HCMP => <<BOOL>>, HACT => <<BOOL>>, QUE =>
  <<USINTL>>, SWPO => <<DINT>>, ERR => <<BYTE>>);
```

This function block is a replacement for M_LHOME for a SERCOS axis. It is not for an analog controlled axis.

This function block performs a ladder reference cycle on a SERCOS axis, followed by a homing (position) move to a designated location.

Before this function block can be used, the SERCOS axis must be initialized and the position loop closed.

The reference cycle will cause the selected SERCOS axis to move in the designated direction until the reference switch is sensed. This function block uses SCA_RFIT to direct the drive to ignore the fast input for the reference and to monitor the position while the ladder checks for the reference switch. In a ladder reference, this reference switch is wired to an input in the MMC or in an input module within the PiC rack and updated each scan of the ladder. When the reference switch is sensed the SERCOS axis will reference (assign a value) to the next index mark of an encoder or the nearest null of a resolver. After the value is assigned the axis will decelerate to a stop and set the reference done flag.

If the HOME input is on when the reference done has been sensed the home move will automatically be triggered to position the SERCOS axis at a desired location.

If the BKOF input is on when the reference is requested and if the axis is on the reference switch, the axis will move in the opposite direction of that indicated by the PLUS input until the reference switch opens, and then will complete the home cycle in the normal manner. If the BKOF input is not on, the axis will move in the specified direction until it sees an off to on transition of the limit switch.

This function block is used to perform a ladder reference, immediately followed by a position move to a selected home position. It should be executed every scan unless a home cycle will only be performed when the machine is started. In that case a normally closed contact of the output of HCMP may be used.

The SWPO output is used to determine if the reference switch location will allow for repeatable referencing. If the reference switch is not properly located in relationship to the index marker of an encoder or the null of a resolver it could possibly reference a revolution off. To prevent this, the value reported by this output should be as follows:

- For an encoder system the value of this output should be greater than 25% and less than 75% of the total counts (FUs) per revolution. Example: For 8000 FUs/ Rev, the value should be >2000 and <6000.
- For a resolver system the value of this output should be less than 25% or greater than 75% of the total counts (FUs) per revolution. Example: For 4000 FUs/ Rev, the value should be <1000 or >3000.

If the value is out of range either the reference switch will have to be moved or the transducer coupling shifted.

The ERR output indicates that invalid data was entered on one of the inputs. The possible errors are listed below:

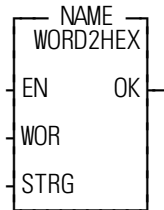
ERR Description

- 0 No error
- 1 The queue was not empty when the reference was requested
- 2 An error occurred in backing off of the reference switch
- 3 An error occurred in referencing
- 4 An error occurred in homing
- 5 An error occurred within the SERCOS drive, either during the initialization of the SERCOS drive (its probe input) or during the monitoring of the SERCOS drive while it is referencing. The SERCOS ring and slave error values can be obtained by animating this function block after the error.

WORD2HEX

Converts a word to a hex value

USER/M_COMMON



Inputs: EN (BOOL) - enables execution
WORD (WORD) - value to convert
STRG (STRING) - Converted value

Outputs: OK (BOOL) - execution complete

```
<<INSTANCE NAME>>:WORD2HEX(EN00 := <<BOOL>>, WORD := <<  
WORD>> STRG := <<STRING>>, OK => <<BOOL>>);
```

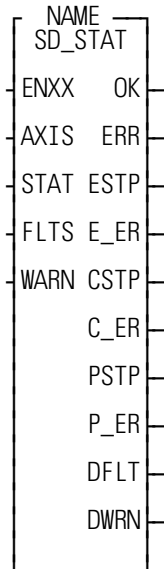
This function block places the hexadecimal notation of the value at WORD into the string at STRG.

Example: If 26,854 is entered at the WORD input, 68E6 will be reported at STRG.

SD_STAT

Axis/Drive Status, Fault/Warning Information for MMC Digital Smart Drive

USER/SD_DATA



Inputs: ENxx (BOOL) - enables execution
AXIS (USINT) - identifies axis
STAT (STRUCT) - provides axis and drive status information
FLTS (STRUCT) - provides axis and drive fault information
WARN (STRUCT) - provides axis and drive warning information

Outputs:
OK (BOOL) - execution complete
ERR (INT) - reports an error number, see error listing below
ESTP (BOOL) - indicates an E-Stop is active
E_ER (WORD) - identifies E-Stop errors
CSTP (BOOL) - indicates a C-Stop is active
C_ER (WORD) - identifies C-Stop errors
PSTP (BOOL) - indicates a P-Stop is active
P_ER (WORD) - identifies P-Stop errors
DFLT (UINT) - identifies Drive Fault Code
DWRN (UINT) - identifies Drive Warning Code

```
<<INSTANCE  
NAME>>:SD_STAT(ENXX:=1,Axis=<<USINT>>,Stat=<<MEMORY  
AREA>>,Flts:=MEMORY AREA>>,Warn:=MEMORY  
AREA>>,OK=><<BOOL>>,ERR=><<INT>>,ESTP=><<BOOL>>,E_ER=<>>  
WORD>>,CSTP=><<BOOL>>,C_ER=><<WORD>>,PSIP=><<BOOL>>,P_  
ER=><<WORD>>,DFLT=><<UINT>>,DWRN=><<UINT>>);
```

This function block provides status and fault information using the READ_SV variables and other common Motion functions for an MMC Digital Servo Axis. The structure inputs for this ASFB are quite large and need to match the descriptions below to function correctly. Instead of re-entering these declarations we suggest that you either start with one of the Digital Smart Drive Examples included with the PicPro 15.0 or higher Applications CD or copy and paste the appropriate networks from the example into your ladder.

NOTE: The Stat.StructureSize, Flts.StructureSize, and Warn.StructureSize variables must set to the size of the structures in bytes. Use the SIZEOF function to determine the size of the data structure as shown in the example ladders. The ASFB compares the sizes of the structures with the internal structures to make sure that they match. If they don't match an error will be reported and the function will not execute.

INPUTS:

NAME	DATA TYPE	DESCRIPTION OR FUNCTION USED FOR DATA
ENxx	BOOL	Enables Execution
Axis	USINT	Axis Number
Stat	STRUCT	Value written to drive output 1, READ_SV variable 72
.InPosition	BOOL	IN_POS?
.QueAvailable	BOOL	Q_AVAIL?
.AcitiveQue	USINT	Q_NUMBER
.MoveType	DINT	READ_SV variable 2
.Actual Position	DINT	READ_SV variable 1
.CommandedPosition	DINT	READ_SV variable 3
.PositionError	DINT	READ_SV variable 4
.FilterError	DINT	READ_SV variable 5
.MotorCurrent	DINT	READ_SV variable 73
.MotorAvgCurrent	DINT	READ_SV variable 74
.MotorTemp	DINT	READ_SV variable 77
.StartupCommutation Complete	BOOL	READ_SV variable 67 Bit 0
.AtZeroSpeed	BOOL	READ_SV variable 67 Bit 1
.InSpeedWindow	BOOL	READ_SV variable 67 Bit 2
.UpToSpeed	BOOL	READ_SV variable 67 Bit 3
.AtPlusCurrentLimit	BOOL	READ_SV variable 67 Bit 4
.AtMinusCurrentLimit	BOOL	READ_SV variable 67 Bit 5
.DriveBusCharged	BOOL	READ_SV variable 67 Bit 6
.DriveEnabled	BOOL	READ_SV variable 67 Bit 7
.DriveReady	BOOL	READ_SV variable 67 Bit 8
.BrakeReleased	BOOL	READ_SV variable 67 Bit 9
.DriveFault	BOOL	READ_SV variable 67 Bit 10
.DriveWarning	BOOL	READ_SV variable 67 Bit 11
.ShuntOn	BOOL	READ_SV variable 67 Bit 12
.DriveReadyAndBusCharged	BOOL	READ_SV variable 67 Bit 13
Reserved _26_31	BOOL (0..5)	READ_SV variable 67 Bits 26-31
.HardwareEnableLine	BOOL	READ_SV variable 67 Bit 24
AuxiliaryFeedbackLossOfFeedback	BOOL	READ_SV variable 67 Bit 25
Reserved	BOOL (0..9)	READ_SV variable 67 Bits 14-23

StructureSize	UINT	Set to the Size of STAT Structure in Bytes END_STUCT
---------------	------	---

NAME	DATA TYPE	DESCRIPTION OR FUNCTION USED FOR DATA
------	-----------	---------------------------------------

Flts	STRUCT	Axis and Drive Faults
.EstopActive	BOOL	E_STOP?
.EstopErr	WORD	E_ERRORS
.LossOfFeedbackEstop	BOOL	E_ERRORS Bit 0
.ExcessPositionErrorEstop	BOOL	E_ERRORS Bit 1
.SlaveOverflowDeltaEstop	BOOL	E_ERRORS Bit 2
.UserLadderEstop	BOOL	E_ERRORS Bit 3
.SERCOSDriveEstop	BOOL	E_ERRORS Bit 4
.SERCOSCycDataEstop	BOOL	E_ERRORS Bit 5
.ASIU_Timeout	BOOL	E_ERRORS Bit 6
.DriveFaultEstop	BOOL	E_ERRORS Bit 7
.DriveCommFaultEstop	BOOL	E_ERRORS Bit 8
Reserved Estopbits	BOOL (0..5)	E_ERRORS Bits 9-14
.AnyEstopBitSet	BOOL	E_ERRORS Bit 15
.CstopActive	BOOL	C_STOP?
.CstopErr	WORD	C_ERRORS
.SWPositiveEndLimitCstop	BOOL	C_ERRORS Bit 0
.SWNegative EndLimitCstop	BOOL	C_ERRORS Bit 1
.UserLadderCstop	BOOL	C_ERRORS Bit 2
.MachRefDimCstop	BOOL	C_ERRORS Bit 3
.FeedrateCstop	BOOL	C_ERRORS Bit 4
.DistPosnDimCstop	BOOL	C_ERRORS Bit 5
.PartRefDimCstop	BOOL	C_ERRORS Bit 6
.PartRefCstop	BOOL	C_ERRORS Bit 7
Reserved Cstopbits	BOOL (0..6)	C_ERRORS Bits 8-14
.AnyCstopBitSet	BOOL	C_ERRORS Bit 15
.PstopActive	BOOL	P_STOP?
.PstopErr	WORD	P_ERRORS
.MasterStartPosnPstop	BOOL	P_ERRORS Bit 0
Reserved Bit 1 Pstop	BOOL	P_ERRORS Bit 1
Reserved Bit 2 Pstop	BOOL	P_ERRORS Bit 2
Reserved Bit 3 Pstop	BOOL	P_ERRORS Bit 3
Reserved Bit 4 Pstop	BOOL	P_ERRORS Bit 4
.MasterNotAvailPstop	BOOL	P_ERRORS Bit 5

.InvalidProfilePstop	BOOL	P_ERRORS Bit 6
.FastQuePstop	BOOL	P_ERRORS Bit 7
.SlaveDistPstop	BOOL	P_ERRORS Bit 8
.MasterDistPstop	BOOL	P_ERRORS Bit 9
.SlaveBeyondStrtPointPstop	BOOL	P_ERRORS Bit 10
.MasterBeyondStrtPointPstop	BOOL	P_ERRORS Bit 11
Reserved Bit 12 Pstop	BOOL	P_ERRORS Bit 12
Reserved Bit 13 Pstop	BOOL	P_ERRORS Bit 13
Reserved Bit 14 Pstop	BOOL	P_ERRORS Bit 14
.AnyPstopBitSet	BOOL	P_ERRORS Bit 15
.NoDriveFaults	BOOL	Set when no drive fault is active
.DriveMemoryFault_11	BOOL	READ_SV variable 68 Bit 0
.DriveBusOverVoltageFault_12	BOOL	READ_SV variable 68 Bit 1
.DrivePM1OverCurrentFault_13	BOOL	READ_SV variable 68 Bit 2
.DriveOverPowerFault	BOOL	READ_SV variable 68 Bit 3
.MotorTempFault	BOOL	READ_SV variable 68 Bit 4
.ContinuousCurrentFault	BOOL	READ_SV variable 68 Bit 5
.DriveHeatsinkTempFault_17	BOOL	READ_SV variable 68 Bit 6
Reserved1	BOOL	READ_SV variable 68 Bit 7
.DriveF1FeedbackFault_22	BOOL	READ_SV variable 68 Bit 8
.DriveAmbientTempFault_23	BOOL	READ_SV variable 68 Bit 9
.MotorCalculatedTempFault_24	BOOL	READ_SV variable 68 Bit 10
.DriveTimingFault_25	BOOL	READ_SV variable 68 Bit 11
.DriveInterfaceFault_26	BOOL	READ_SV variable 68 Bit 12
.UserSetFault_27	BOOL	READ_SV variable 68 Bit 13
.DriveF1CommunicationFault_31	BOOL	READ_SV variable 68 Bit 14
.OverSpeedFault_32	BOOL	READ_SV variable 68 Bit 15
.OverCurrentFault_33	BOOL	READ_SV variable 68 Bit 16
.ControlPanelDisconnectFault_34	BOOL	READ_SV variable 68 Bit 17
.DrivePowerModuleFault_35	BOOL	READ_SV variable 68 Bit 18
.DriveSetupDataFault_36	BOOL	READ_SV variable 68 Bit 19
Reserved2	BOOL	READ_SV variable 68 Bit 20
.DriveRelayFault_41	BOOL	READ_SV variable 68 Bit 21
.DrivePM2OverCurrentFault_42	BOOL	READ_SV variable 68 Bit 22
.DrivePMTemperatureFault_43	BOOL	READ_SV variable 68 Bit 23
.MotorGroundFault	BOOL	READ_SV variable 68 Bit 24
.DriveACInputOverVoltageFault_44	BOOL	READ_SV variable 68 Bit 25
.OvertravelPlusFault_46	BOOL	READ_SV variable 68 Bit 26
.OvertravelMinusFault_47	BOOL	READ_SV variable 68 Bit 27
.DigitalLinkCommunicationError_5	BOOL	READ_SV variable 68 Bit 28

1

.InvalidSwitchSettingFault_52	BOOL	READ_SV variable 68 Bit 29
Reserved3	BOOL	READ_SV variable 68 Bit 30
Reserved4	BOOL	READ_SV variable 68 Bit 31
.StuctureSize	UINT	Set to the Size of Flts Structure in Bytes
	END_STUC	
	T	

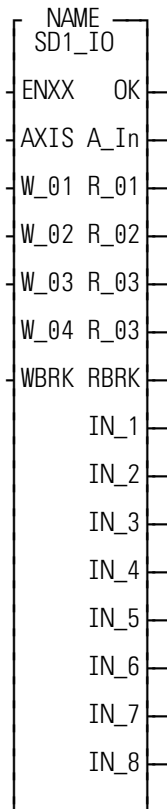
NAME	DATA TYPE	DESCRIPTION OR FUNCTION USED FOR DATA
Warn	STRUCT	Drive Warnings, see MMC Smart Drive and Digital MMC Control Hardware Manual for detailed descriptions
.Nowarnings	BOOL	Set when no drive warning is active
.DriveHeatsinkTempWarning_01	BOOL	READ_SV variable 69 Bit 0
.DriveAmbientTempWarning_02	BOOL	READ_SV variable 69 Bit 1
.MotorTempWarning_03	BOOL	READ_SV variable 69 Bit 2
.MotorCalculatedTempWarning_04	BOOL	READ_SV variable 69 Bit 3
.LossofFeedbackonAuxWarning_05	BOOL	READ_SV variable 69 Bit 4
Reserved	BOOL (0-.26)	READ_SV variable 69 Bits 5-31
.StructureSize	UINT	Set to the Size of Warn Structure in Bytes
		END STUCT

OUTPUTS:

NAME	DATA TYPE	DESCRIPTION OR FUNCTION USED FOR DATA
OK	BOOL	Execution complete without error
ERR	INT	Reports error number, see error listing below
ESTP	BOOL	Indicates an E-Stop is active
E_ER	WORD	Identifies E-Stop errors, see E_ERRORS
CSTP	BOOL	Indicates a C-Stop is active
C_ER	WORD	Identifies C-Stop errors; see C_ERRORS
PSTP	BOOL	Indicates a P-Stop is active
P_ER	WORD	Identifies P-Stop errors, see P_ERRORS
DFLT	UINT	Identifies Drive Fault Code, see the MMC Smart Drive and Digital MMC Control Hardware Manual for detailed descriptions
DWRN	UINT	Identifies Drive Warning Code, see the MMC Smart Drive and Digital MMC Control Hardware Manual for detailed description

ERROR CODES FOR ERR:

ERR	Error Description
1	Invalid value for Stat.StructureSize, the Stat structure has the wrong number of elements and/or the wrong data types
2	Invalid value for Flts.StructureSize, the Flts structure has the wrong number of elements and/or the wrong data types
3	Invalid value for Warn.StructureSize, the Warn structure has the wrong number of elements and/or the wrong data types
4	One or more of the internal motion functions failed, check to make sure that the axis has been initialized



Inputs: ENxx (BOOL) - enables execution
AXIS (USINT) - identifies axis
W_01 (BOOL) - value written to drive output 1
W_02 (BOOL) - value written to drive output 2
W_03 (BOOL) - value written to drive output 3
W_04 (BOOL) - value written to drive output 4
WBRK (BOOL) - value written to drive brake output

Outputs:

OK (BOOL) - execution complete
A_IN (DINT) - value of drive analog input
R-01 (BOOL) - current state of drive output 1
R-02 (BOOL) - current state of drive output 2
R-03 (BOOL) - current state of drive output 3
R-04 (BOOL) - current state of drive output 4
R-BRK (BOOL) - current state of drive output brake
IN_1 (BOOL) - current state of drive input 1
IN_2 (BOOL) - current state of drive input 2
IN_3 (BOOL) - current state of drive input 3
IN_4 (BOOL) - current state of drive input 4
IN_5 (BOOL) - current state of drive input 5
IN_6 (BOOL) - current state of drive input 6
IN_7 (BOOL) - current state of drive input 7
IN_8 (BOOL) - current state of drive input 8

```
<<INSTANCE NAME>>:SD_10(ENXX=1,Axis:=<<W_01:=<<BOOOL>>,
W02:=<<BOOOL>>,W_03:=<<BOOOL>>,W_04:=<<BOOOL>>,W_BRK:=<<BO
OOL>>,OK=><<BOOOL>>,A_In=><<DINT>>,R_01=><<BOOOL>>,R_02=><<B
OOL>>,R_03=><<BOOOL>>,R_04=><<BOOOL>>,R_BRK=><<BOOOL>>,In_1
=><<BOOOL>>,In_2=><<BOOOL>>,In_3=><<BOOOL>>,In_4=><<BOOOL>>,In_
5=><<BOOOL>>,In_6=><<BOOOL>>,In_7=><<BOOOL>>,In_8=><<BOOOL>>
```

This function block reads the current values of the analog and DC inputs and reads/writes the outputs for the MMC Digital Smart Drive.

NOTE: The MMC Application Input/Output assignment must be checked in the drive setup for the inputs and outputs to be read/written by this function. See READ_SV variables 70, 71, and 72 for more details.

INPUTS

Name	Data Type	Description or Function Used for Data
ENxx	BOOL	Enables Execution
Axis	USINT	Axis Number
W_01	BOOL	Value written to drive output 1, READ_SV variable 72
W_02	BOOL	Value written to drive output 2, READ_SV variable 72
W_03	BOOL	Value written to drive output 3, READ_SV variable 72
W_04	BOOL	Value written to drive output 4, READ_SV variable 72
WBRK	USINT	Value written to drive brake output, READ_SV variable 72

OUTPUTS:

Name	Data Type	Description or Function Used for Data
OK	BOOL	Execution complete without error
A_IN	DINT	Value of drive analog input, READ_SV variable 71
R_01	BOOL	Current state of drive output 1, READ_SV variable 72
R_02	BOOL	Current state of drive output 2, READ_SV variable 72
R_03	BOOL	Current state of drive output 3, READ_SV variable 72
R_04	BOOL	Current state of drive output 4, READ_SV variable 72
RBRK	BOOL	Current state of drive brake output, READ_SV variable 72
IN_1	BOOL	Current state of drive input 1, READ_SV variable 71
IN_2	BOOL	Current state of drive input 2, READ_SV variable 71
IN_3	BOOL	Current state of drive input 3, READ_SV variable 71
IN_4	BOOL	Current state of drive input 4, READ_SV variable 71
IN_5	BOOL	Current state of drive input 5, READ_SV variable 71
IN_6	BOOL	Current state of drive input 6, READ_SV variable 71
IN_7	BOOL	Current state of drive input 7, READ_SV variable 71
IN_8	BOOL	Current state of drive input 8, READ_SV variable 71

NOTES

APPENDIX A **M_DSMCOM** Commands

This appendix contains the commands that can be entered at the CMD input of the M_DSMCOM function block. These commands allow you to communicate with the DSM100 drive over the communications port. The tables that follow contain detailed descriptions of the commands, applicable values, responses, and exceptions.

Exception Responses

If a command is received by the drive without a communication error, but cannot be processed normally, an exception response is generated. The table below lists the possible exception responses.

Response Data	Exception Type	Description	Applicable Commands
01	Invalid Data	The command data parameter was unacceptable, and the parameter was not changed in the drive.	Non-Range Variable Commands, Low Level Commands
02	Command Not Enabled	The command is disabled and is dependent on another command for enabling.	Manufacturing, Firmware Upgrades
03	EEPROM Write Error	The command required a write to EEPROM, and the data was not able to be written.	All
04	Data Accepted After Limiting to Minimum	The command data was out of range, but was modified to the minimum value.	Range Variable Commands
05	Data Accepted After Limiting to Maximum	The command data was out of range, but was modified to the maximum value.	Range Variable Commands
06	Command Disabled When Drive is Enabled	The command cannot be complied with, because the drive is enabled.	All
07	Flash Programming Error	The command required the flash memory to be altered and an error occurred.	Flash Memory Altering Commands
08	Invalid Function Code	The master function code was not recognized by the drive.	All
09	Command Disabled When Drive is Disabled	The command cannot be complied with, because the drive is disabled.	All

Host Command Set

The tables below use the following symbols to specify data widths.

Note: Every byte in the specified data field is sent in the command encoded as two ASCII-hex characters):

Data	Signed	Unsigned
8-bit data	[c1]..[cn]	[b1]..[bn]
16-bit data	[s1]..[sn]	[w1]..[wn]
32-bit data	[L1]..[Ln]	[d1]..[dn]

On numeric parameters, the Range of Data Values field contains the range of values in user units and the resolution [denoted by (ϵ : xxxxxx)]. The field also contains the range of command hexadecimal values expected for the parameter.

In addition, the Units field contains the multiplier for converting the user units to the command hexadecimal values. These multipliers are presented in hexadecimal also.

Common Product Line Commands

These commands will remain consistent across product lines. The bit definitions on the Powerup Status command may change between products, but the command must return zero (00) on a successful powerup.

Parameter	Range of Data Values	Units	Com- mand	Com- mand Data	Respo nse Data	Excep tion Respo nses
Product Type Identifies the type of product. Currently only the BCM-03 is known. but is provided for future expansion.	[b1] - Type 0- BCM-03	-	Rea	00	-	[b1]
			d	0		
			Wri	-	-	-
			te			

<p>Powerup Status The status of the drive during power up testing. The bit definitions of the READ Powerup Status command may change between products, but the command will always return zero (00) on a successful powerup.</p>	<p>[b1] - Status 00 - Successful Power-Up 51 - Boot Block Checksum Error 52 - Non-Boot Block Checksum Error 53 - Uninitialized Personality EEPROM Error 54 - Personality EEPROM Read Error 55 - Personality EEPROM Data Corruption Error 56 - Main Processor Watchdog Error 57 - Sub Processor Watchdog Error 58 - Main Processor RAM Error 59 - Sub Processor RAM Error 60 - Uninitialized Service EEPROM Error 61 - Service EEPROM Read Error 62 - Service EEPROM Data Corruption Error 63 - Main Processor A/D Converter Error 64 - Sub Processor A/D Converter Error</p>	-	Read	00	-	[b1]
--	---	---	------	----	---	------

	65 - Analog1 Output Error 66 - Gate Array Error 67 - Analog2 Output Error 68 - Inter-Processor Communication Error 69 - Sub Processor Initialization Error 70 - Sub Processor SRAM Error 71 - Sub Processor Code Loading Error 72 - Sub Processor Startup Error 73 - Sub Processor Checksum Error 74 - Personality EEPROM Write Error 75 - Service EEPROM Write Error 76 - Software Clock Error 77 - Sub Processor Communication Checksum Error 78 - Sub Processor Sine Table Generation Error 79 - Personality Data Out Of Range 80 - Service Data Out Of Range 81 - Motor Block Checksum Error		Write	-	-	-	
Main Firmware Version The version number of the drive's main firmware.	[b1] - Major Version 0..255 (00..ff)	-	Read	002	-	[b1] [b2]	
	[b2] - Minor Revision 0..255 (00..ff)		Write		-	-	
Boot Firmware Version The version number of the drive's boot firmware.	[b1] - Major Version 0..255 (00..ff)	-	Read	003	-	[b1] [b2]	
	[b2] - Minor Revision 0..255 (00..ff)		Write		-	-	

General Commands

Parameter	Range of Data Values	Units	Com- mand	Com- mand Data	Respo nse Data	Excep- tion Respo nses
Reset Personality EEPROM Resets the personality EEPROM to its factory settings.	[No Data]	-	Read	-	-	
			Write	010	-	-
Drive Name Identifies the drive in a multidrop system.	[b1]..[b32] - Name Name is a 32-character string.	-	Read	011	-	[b1]..[b32]
			Write	012	[b1]..[b32]	-
Position Scale Value The position scale used by the host computer for scaling position variables. This information is not necessary for drive operation. The scale is a 32 bit IEEE floating point value.	[d1] - Value -3.4e+38..3.4e+38 (ε: 1.19e-7) (80000000..7ffffff)	units / count	Read	013	-	[d1]
			Write	014	[d1]	-
Position Scale Text The position scale text used by the host computer to identify the units of the position scale.	[b1]..[b8] - Name Name is an 8-character string.	-	Read	015	-	[b1]..[b8]
			Write	016	[b1]..[b8]	-
Velocity Scale Value The velocity scale used by the host computer for scaling velocity variables. This information is not necessary for drive operation. The scale is a 32 bit IEEE floating point value.	[d1] - Value -3.4e+38..3.4e+38 (ε: 1.19e-7)	units / RPM	Read	017	-	[d1]

	(80000000..7fffffff)		Write	018	[d1]	-	03
Velocity Scale Text The velocity scale text used by the host computer to identify the units of the velocity scale.	[b1]..[b8] - Name Name is an 8-character string.	-	Read	019	-	[b1]..[b8]	
			Write	01a	[b1]..[b8]	-	03
Acceleration Scale Value The acceleration scale used by the host computer for scaling acceleration variables. This information is not necessary for drive operation. The scale is a 32 bit IEEE floating point value.	[d1] - Value -3.4e+38..3.4e+38 (ε: 1.19e-7) (80000000..7fffffff)	units / RPM / second	Read	01b	-	[d1]	
			Write	01c	[d1]	-	03
Acceleration Scale Text The acceleration scale text used by the host computer to identify the units of the acceleration scale.	[b1]..[b8] - Name Name is an 8-character string.	-	Read	01d	-	[b1]..[b8]	
			Write	01e	[b1]..[b8]	-	03
Torque Scale Value The torque scale used by the host computer for scaling torque variables. This information is not necessary for drive operation. The scale is a 32 bit IEEE floating point value.	[d1] - Value -3.4e+38..3.4e+38 (ε: 1.19e-7) (80000000..7fffffff)	units / Amp	Read	01f	-	[d1]	
			Write	020	[d1]	-	03
Torque Scale Text The torque scale text used by the host computer to identify the units of the torque scale.	[b1]..[b8] - Name Name is an 8-character string.	-	Read	021	-	[b1]..[b8]	
			Write	022	[b1]..[b8]	-	03

Position Loop Commands

Parameter	Range of Data Values	Units	Com- mand	Com- mand Data	Respo- nse Data	Excep- tion Respo- nses
Position Loop Propor- tional Gain	[w1] - Gain 0.0..31.98 (ϵ : 7.8e-3) (0000..0fff)	in/min/ mil (\times 0080)	Rea d	03 0	-	[w1]
			Wri te	03 1	[w1]	-
Position Loop Integral Gain	[w1] - Gain 0..31.98 (ϵ : 7.8e-3) (0000..0fff)	-	Rea d	03 2	-	[w1]
			Wri te	03 3	[w1]	-
Position Loop Deriva- tive Gain	[w1] - Gain 0..31.98 (ϵ : 7.8e-3) (0000..0fff)	-	Rea d	03 4	-	[w1]
			Wri te	03 5	[w1]	-
Position Loop Feedfor- ward Gain	[w1] - Gain 0..200 (ϵ : 1) (0000..00c8)	-	Rea d	03 6	-	[w1]
			Wri te	03 7	[w1]	-
Integrator Zone Maximum position error which the integrator is still active. If the posi- tion error is greater than the I Zone, the integrator is reset	[w1] - Zone 0..32767 (ϵ : 1) (0000..7fff)	counts	Rea d	03 8	-	[w1]
			Wri te	03 9	[w1]	-
Position Window Size Maximum position error which allows the In Posi- tion flag to remain set.	[w1] - Size 0..32767 (ϵ : 1) (0000..7fff)	counts	Rea d	03 a	-	[w1]
			Wri te	03 b	[w1]	-
Position Window Time The minimum time which the position error must be less than the Position Window Size to set the In Position flag.	[b1] - Time 0..255 (ϵ : 1)	milli- sec- onds	Rea d	03 c	-	[b1]

	(00..ff)	(× 01)	Write	03	[b1]	-	03
Position Error Limit Minimum position error which allows the excess Position Error flag to remain clear	[d1] - Limit 1..2147483647 (ε: 1)	counts	Read	03	-	[d1]	
	(00000001..7ffffff)	(× 00000001)	Write	03	[d1]	-	03, 04, 05
Position Error Time The minimum time which the position error must be greater than the Position Error Limit to cause an Excess Position Error fault.	[w1] - Time 0..65535 (ε: 1)	milli-seconds	Read	04	-	[w1]	
	(00..ffff)	(× 0001)	Write	04	[w1]	-	03
Gear Ratio The ratio between the motor and master counts for following.	[s1] - Motor -32767..32767 (ε: 1) (8001..7fff)	counts	Read	04	-	[s1] [s2]	
	[s2] - Master 1..32767 (ε: 1) (0001..7fff)	master counts (× 0001)	Write	04	[s1] [s2]	-	03, 04
Master Rotation The rotation direction of the master encoder in follower mode, and the polarity of the direction input in the step/direction mode.	[b1] - Direction 0 - forward direction (TP: Normal) 1 - reverse direction (TP: Reverse)	-	Read	04	-	[b1]	
			Write	04	[b1]	-	01, 03, 06
Slew Rate The acceleration limit for the motor when used in a follower mode.	[d1] - Rate 0..2147483647 (ε: 1)	RPM/sec	Read	04	-	[d1]	
	(00000000..7ffffff)	(× 00000001)	Write	04	[d1]	-	03, 05

Slew Enable Determines if the slew rate is used in follower mode	[b1] - Flag 0 - Disabled (TP: Off) 1 - Enabled (TP: On)	-	Rea	04	-	[b1]	
			d	8			
			Wri	04	[b1]	-	01, 03
			te	9			

Velocity Loop Commands

Parameter	Range of Data Values	Units	Com- mand	Com- mand Data	Respo nse Data	Excep tion Respo nses
Velocity Loop Proportional Gain	[w1] - Gain 0..1000 (ε: 1) (0000..03e8)	-	Re ad	04a	-	[w1]
			Wri te	04b	[w1]	-
Velocity Loop Integral Gain	[w1] - Gain 0..1000 (ε: 1) (0000..03e8)	-	Re ad	04c	-	[w1]
			Wri te	04d	[w1]	-
Velocity Loop Derivative Gain	[s1] - Gain -1000..1000 (ε: 1) (fc18..03e8)	-	Re ad	04e	-	[s1]
			Wri te	04f	[s1]	-
Zero Speed Limit Maximum motor velocity which allows the ZeroSpeed flag to remain set.	[d1] - Limit 0..32767.99998 (ε: 1.53e-5) (00000000..7fffffff)	RPM	Re ad	050	-	[d1]
			Wri te	051	[d1]	-
Speed Window Size Maximum motor velocity error which allows the Speed Window flag to remain set.	[d1] - Limit 0..32767.99998 (ε: 1.53e-5) (00000000..7fffffff)	RPM	Re ad	052	-	[d1]
			Wri te	053	[d1]	-
Over Speed Limit Minimum motor velocity which causes the Over- speed fault to occur.	[d1] - Limit 0..32767.99998 (ε: 1.53e-5) (00000000..7fffffff)	RPM	Re ad	054	-	[d1]
			Wri te	055	[d1]	-

At Speed Limit Minimum motor velocity which causes the At Speed flag to be set	[d1] - Limit 0..32767.99998 (ε: 1.53e-5) (00000000..7fffffff)	RPM (× 00010000)	Re ad	056	-	[d1]	
			Wri te	057	[d1]	-	03, 05
Velocity Loop Update Period Velocity control loop execution period.	[b1] - Period 0 - 200 μsecond 1 - 400 μsecond 2 - 600 μsecond 3 - 800 μsecond 4 - 1000 μsecond 5 - 1200 μsecond 6 - 1400 μsecond 7 - 1600 μsecond	-	Re ad	058	-	[b1]	
			Wri te	059	[b1]	-	01, 03, 06
Velocity Error Limit Sets or returns the minimum velocity error which allows the Excess Velocity Error flag to remain clear.	[d1] - limit 0..32767.99998 (ε: 1.53e-5) (00000000..7fffffff)	RPM (× 00010000)	Re ad	05A	-	[d1]	03, 05
			Wri te	05B	[d1]	-	
Velocity Error Time Sets or returns the minimum time which the velocity error must be greater than the Velocity Error Limit to cause an Excess Velocity Error fault	[w1] - time 0..65535 (ε: 1.53e-5) (00.. ffff)	milli-second (x0001)	Re ad	05C	-	[w1]	03
			Wri te	05D	[w1]	-	

Torque Current Conditioning Commands

Parameter	Range of Data Values	Units	Command	Command Data	Response Data	Exception Responses	
Low Pass Filter Bandwidth Cutoff frequency of the low pass filter.	[w1] - Bandwidth 1..992 (ϵ : 1)	Hz	Read	070	-	[w1]	
	(0001..03e0)	(\times 0001)	Write	071	[w1]	-	03, 04, 05
Low Pass Filter Enable Determines if the low pass filter is used in the control loop.	[b1] - Flag 0 - Disabled (TP: Off)	-	Read	076	-	[b1]	
	1 - Enabled (TP: On)		Write	077	[b1]	-	01, 03
Software Positive Current Limit User specified positive current limit for the drive. The minimum of this value, the peak rating of the drive, the peak rating of the motor, and the +ILIMIT analog input is used as the limiting value.	[w1] - Limit 0..255.992 (ϵ : 7.8e-3)	Amps	Read	07a	-	[w1]	
	(0000..7fff)	(\times 0080)	Write	07b	[w1]	-	03, 05
Software Negative Current Limit User specified negative current limit for the drive. The minimum of this value, the peak rating of the drive, the peak rating of the motor, and the -ILIMIT analog input is used as the limiting value.	[w1] - Limit 0..255.992 (ϵ : 7.8e-3)	Amps	Read	07c	-	[w1]	
	(0000..7fff)	(\times 0080)	Write	07d	[w1]	-	03, 05

Continuous Current Limit User specified current faulting value. This parameter is provided to allow a faulting current value which is less than the capacity of the drive and motor.	[w1] - Limit 0..255.992 (ε: 7.8e-3)	Amps	Read	07e	-	[w1]	
	(0000..7fff)	(× 0080)	Write	07f	[w1]	-	03, 05
PWM Frequency Switching Disable Sets or returns the flag which indicates if the PWM frequency changes with the speed and current demands of the motor.	[b1] - Flag 00 - Enabled	-	Read	1A8	-	[b1]	
	01 - Disabled	-	Write	1A9	[b1]	-	01, 03

Motor Commands

Note: All Motor Commands other than *Motor ID* are disabled and return the exception response 02 unless Motor ID is set to 65535 (ffff).

Parameter (Motor Commands)	Range of Data Values	Units	Command	Command Data	Response Data	Exception Responses
Motor ID Identifies the motor in the drive's motor parameter table currently being used. The word is separated into various groups of bit fields to specify the encoder resolution, motor, type, and table ID. The setting 0 (0000) indicates that no motor has been selected, and the setting 65535 (ffff) indicates motor parameters were set Individually and not read from the drive's motor parameter table.	[w1] - Number 0..65535 (0000..ffff) BITS USAGE 15..12 Table ID 11..8 Encoder Resolution 7 Type (0 = synch., 1 = induct.) 6..0 Motor Number	-	Read 09	-	[w1]	
			Write 09	[w1]	-	03, 06
Encoder Lines The number of lines on the motor encoder.	[w1] - Lines 100..15000 (ε: 1) (0064..3a98)	lines/rev (× 0001)	Read 09	-	[w1]	
			Write 09	[w1]	-	01, 02, 03, 06
Maximum Motor Speed The minimum speed of the motor which causes an Overspeed fault.	[d1] - Speed 0..32767.99998 (ε: 1.53e-5) (00000000..7ffffff)	RPM (× 0001000)	Read 09	-	[d1]	
			Write 09	[d1]	-	01, 02, 03, 06
Motor Peak Current The peak current which the motor can handle.	[w1] - Current 0..255.992 (ε: 7.8e-3) (0000..7fff)	Amps (× 0080)	Read 09	-	[w1]	
			Write 09	[w1]	-	01, 02, 03, 06

Motor Continuous Current The continuous current which the motor can handle.	[w1] - Current 0..255.992 (ϵ : 7.8e-3)	Amps	Read 09 -	[w1]	
	(0000..7fff)	(\times 0080)	Write 09 [w1] -		01, 02, 03, 06
Torque Constant The sine wave torque constant of the motor.	[w1] - K_t 0.00024..15.9998 (ϵ : 2.44e-4)	N-m/ Amp	Read 09 -	[w1]	
	(0001..ffff)	(\times 1000)	Write 09 [w1] -		01, 02, 03, 06
Rotor Inertia J_m	[w1] - J_m 0.0156..1023.98 (ϵ : 1.56e-2)	kg- cm ²	Read 09 -	[w1]	
	(0001..ffff)	(\times 0040)	Write 09 [w1] -		01, 02, 03, 06
Back EMF Constant K_e	[w1] - K_e 0.0039..255.996 (ϵ : 3.91e-3)	Volts / 1000 RPM	Read 09 -	[w1]	
	(0001..ffff)	(\times 0100)	Write 09 [w1] -		01, 02, 03, 06
Winding Resistance The phase to phase resistance of the motor windings at 25° C.	[w1] - Resistance 0.0039..255.996 (ϵ : 3.91e-3)	Ohms	Read 0a -	[w1]	
	(0001..ffff)	(\times 0100)	Write 0a [w1] -		01, 02, 03, 06
Winding Inductance The phase to phase inductance of the motor windings.	[w1] - Inductance 0.0039..255.996 (ϵ : 3.91e-3)	mH	Read 0a -	[w1]	
	(0001..ffff)	(\times 0100)	Write 0a [w1] -		01, 02, 03, 06
Thermostat Flag Indicates if the motor contains a thermostat.	[b1] - Flag 0 - no thermostat present	-	Read 0a -	[b1]	
	1 - thermostat is present		Write 0a [b1] -		01, 02, 03, 06

Motor Commands (Continued)

Parameter (Motor Commands)	Range of Data Values	Units	Command	Command Data	Response Data	Exception Responses	
Commutation Type	[b1] - Type 0 - induction motor 1 - 6-step ABS/Index 2 - 8-step ABS/Index 3 - Hall/Index 4 - Hall/Hall	-	Read	0a6	-	[b1]	01, 02, 03, 06
			Write	0a7	[b1]	-	
Current Feedforward	[s1] - Value -127.996..127.996 (ε: 3.91e-3) (8001..7fff)	degree s/ kRPM (× 0100)	Read	0a8	-	[s1]	02, 03, 04, 06
			Write	0a9	[s1]	-	
Thermal Time Constant The thermal time constant for protecting the motor.	[w1] - Time 0..65535 (ε: 1) (0000..ffff)	sec- onds (× 0001)	Read	0aa	-	[w1]	02, 03, 06
			Write	0ab	[w1]	-	
Pole Count Number of poles.	[b1] - Poles 0 - 2 Poles 1 - 4 Poles 2 - 6 Poles 3 - 8 Poles	-	Read	0ac	-	[b1]	01, 02, 03, 06
			Write	0ad	[b1]	-	
Hall Offset The offset of the Hall-effect sensor relative to the rotor.	[w1] - Offset 0..359 (ε: 1) (0000..0167)	electri- cal degree s (× 0001)	Read	0ae	-	[w1]	01, 02, 03, 06
			Write	0af	[w1]	-	
Index Offset The offset of the motor encoder relative to the rotor.	[w1] - Offset 0..359 (ε: 1) (0000..0167)	electri- cal degree s (× 0001)	Read	0b0	-	[w1]	01, 02, 03, 06
			Write	0b1	[w1]	-	

Motor Table Information Information about the Motor Table in the drive. The information returned includes the number of synchronous motors and induction motors in the table, and the table ID number.	[b1] - Sync. Records 0..255 (00..ff) [b2] - Induction Records 0..255 (00..ff) [b3] - Table ID 0..31 (00..1f)		Read	0b2	-	[b1] ..[b3]	
			Write	-	-	-	
Motor Table Record Size Information about the Motor Table records in the drive. The information returned includes the synchronous motor and induction motor record sizes.	[w1] - Sync. Record Size 0..65535 (0000..ffff) [w2] - Induction Record Size 0..65535 (0000..ffff)		Read	0b3	-	[w1] ..[w2]	
			Write	-	-	-	
Motor Table Version Version of the Motor Table in the drive.	[b1] - Major Version 0..255 (00..ff) [b2] - Minor Revision 0..255 (00..ff)		Read	0b4	-	[b1].. b2]	
			Write	-	-	-	
Thermal Time Constant Enable Sets or returns the flag which indicates if the Thermal Time Constant is used for protecting the motor.	[b1] - Flag 00 - Disabled 01 - Enabled	-	Read	1A 6	-	[b1]	
			Write	1A 7	[b1]	-	02, 03, 06

Motor Commands (Continued)

Parameter	Range of Data Values	Units	Com- mand	Com- mand Data	Respo nse Data	Excep tion Respo nses
Motor Forward Direc- tion Flag Sets or returns the motor's forward direction when viewed from the shaft end.	[b1] - Flag 00 - clockwise 01 - counterclockwise		Rea d	1A A	-	[b1]
			Wri te	1A B	[b1]	-

Digital I/O Commands

Parameter	Range of Data Values	Units	Com- mand	Com- mand Data	Respo nse Data	Excep tion Respo nses
Digital Input Configuration Register Determines which flag is (or flags are) controlled by the specified digital input. If no bits are set for an input, it is unassigned. The Preset Select lines can be used together or separately to select the desired preset. Unassigned select lines are set to 0. The select codes are as follows: Preset C B A 0 0 0 0 1 0 0 1 2 0 1 0 3 0 1 1 4 1 0 0 5 1 0 1 6 1 1 0 7 1 1 1	[b1] - Input Number 0 - Input1 1 - Input2 2 - Input3 3 - Input4 [w2] - Flag Number Bit 0 - Torque Override (TP: TrqMode) Bit 1 - Integrator Inhibit (TP: IntInh) Bit 2 - Follower Enable (TP: FolEnab) Bit 3 - Forward Enable (TP: FClamp) Bit 4 - Reverse Enable (TP: RClamp) Bit 5 - Analog Override (TP: Override) Bit 6 - Preset Select Line A (TP: PreSelA) Bit 7 - Preset Select Line B (TP: PreSelB) Bit 8 - Preset Select Line C (TP: PreSelC)		Rea d	0c0	[b1] [w2]	
			Wri te	0c1	[b1] [w2]	01, 03, 06

Digital Output Configuration Register Determines which flag is (or flags are) monitored on the specified digital output. If no bits are set for an input, it is unassigned.	[b1] - Output Number 0 - Output1 1 - Output2 2 - Output3 3 - Output4 [w2] - Flag Number Bit 0 - In-Position (TP: InPos) Bit 1 - Within Position Window (TP: PosWin) Bit 2 - Zero Speed (TP: 0 Speed) Bit 3 - Within Speed Window (TP: SpdWin) Bit 4 - Positive ILimit (TP: +ILimit) Bit 5 - Negative ILimit (TP: -ILimit) Bit 6 - At Speed (TP: AtSpeed) Bit 7 - Drive Enabled (TP: DrvEnab) Bit 8 - DC Bus Charged (TP: BusChg) Bit 9 - Disabling Fault		Read	0c2	[b1]	[w2]	
			Write	0c3	[b1] [w2]		01, 03, 06
Override Digital Output Overrides the digital output control to allow the user to write the output bits directly.	[b1] - State 0 - Normal 1 - Override	-	Read	0c4	-	[b1]	
			Write	0c5	[b1]	-	01
Digital Output Write Mask Contains the bit pattern to write to the digital outputs when in override control.	[w1] - States Bit 0 - READY Output State Bit 1 - BRAKE Output State Bit 2 - OUTPUT1 Output State	-	Read	0c6	-	[w1]	

	Bit 3 - OUTPUT2 Output State Bit 4 - OUTPUT3 Output State Bit 5 - OUTPUT4 Output State		Write	0c7	[w1]	-	01
BRAKE Active Delay The time delay between disabling the drive, and activating the BRAKE output. Negative values indicate the time that the BRAKE is active before disabling the drive. BRAKE Inactive Delay The time delay between enabling the drive and deactivating the BRAKE output. Negative values indicate the time that the BRAKE is inactive before enabling the drive.	[s1] - Delay -32767..32767 (ε: 1) (8001..7fff)	milli-seconds (× 0001)	Read	0c8	-	[s1]	03, 04
			Write	0c9	[s1]	-	
	[s1] - Delay -32767..32767 (ε: 1) (8001..7fff)	milli-seconds (× 0001)	Read	0ca	-	[s1]	
			Write	0cb	[s1]	-	03, 04

Analog I/O Commands

Parameter (Analog I/O Commands)	Range of Data Values	Units	Command	Command Data	Response Data	Exception Responses	
COMMAND Velocity Offset The offset applied to the COMMAND analog input when being used for velocity command.	[s1] - Offset -10000..10000 (ε: 1) (d8f0..2710)	milli-volts (× 0001)	Read	0c	-	[s1]	
			Write	0cd	[s1]	-	03, 04

COMMAND Velocity Scale The scale applied to the COMMAND analog input when being used for velocity command.	[s1] - Scale -32767..32767 (ε: 1) (8001..7fff)	RPM / Volt (× 0001)	Read 0c - [s1]	
			Write 0c [s1] - 03, 04	
COMMAND Torque Offset The offset applied to the COMMAND analog input when being used for torque command.	[s1] - Offset -10000..10000 (ε: 1) (d8f0..2710)	milli-volts (× 0001)	Read 0d - [s1]	
			Write 0d [s1] - 03, 04	
COMMAND Torque Scale The scale applied to the COMMAND analog input when being used for torque command.	[s1] - Scale -127.996..127.996 (ε: 3.91e-3) (8001..7fff)	Amps / Volt (× 0100)	Read 0d - [s1]	
			Write 0d [s1] - 03, 04	

<p>Analog Output Configuration Register Determines which signal is monitored on the specified analog output.</p>	<p>[b1] - Output Number 0 - Output1 1 - Output2</p> <p>[b2] - Signal Number 0 - Current Command (TP: ICmd) 1 - Current Average Command (TP: Avg I) 2 - Current Positive Peak (TP: +IPeak) 3 - Current Negative Peak (TP: -IPeak) 4 - Positive ILimit (TP: +ILimit) 5 - Negative ILimit (TP: -ILimit) 6 - Motor Velocity (TP: MtrVel) 7 - Velocity Command (TP: VelCmd) 8 - Velocity Error (TP: VelErr) 9 - Motor Position (TP: MtrPos) 10 - Position Command Slew (TP: PosCmd)</p>		Read	0d4	[b1]	[b2]	
---	--	--	------	-----	------	------	--

	11 - Position Error (TP: Pos-Err) 12 - Position Peak Positive Error (TP: +PosEPk) 13 - Position Peak Negative Error (TP: -PosEPk) 20 - Master Position (TP: MstrPos) 21 - Position Loop Output (TP: [not avail]) 22 - Velocity Loop Output (TP: [not avail]) 23 - Filter Output (TP: [not avail]) 24 - Notch Output (TP: [not avail]) 25 - R Phase Current (TP: [not avail]) 26 - T Phase Current (TP: [not avail]) 27 - Torque Current (TP: [not avail]) 28 - Field Current (TP: [not avail]) 29 - Torque Voltage (TP: [not avail]) 30 - Field Voltage (TP: [not avail]) 31 - Scaled A/D Command Value (TP: [not avail]) 32 - Bus Voltage (TP: [not avail])		Write	0d5	[b1] [b2]		01, 03
Analog Offset The offset applied to the specified Analog output.	[b1] - Output Number 0 - Output1 1 - Output2		Read	0d6	[b1]	[s2]	
	[s2] - Offset -32767..32767 (ε: 1) (8001..7fff)	milli-volts (×0001)	Write	0d7	[b1] [s2]	-	01, 03, 04

Analog I/O Commands (Continued)

Parameter (Analog I/O Commands)	Range of Data Values	Units	Com- mand	Com- mand Data	Resp onse Data	Excep- tion Respo nses	
Analog Scale The scale applied to the specified Analog output.	[b1] - Output Number 0 - Output1 1 - Output2 [s2] - Scale -32767..32767 (ε: 1) (8001..7fff)	Drive Inter- nal Units (Depen- dent on selecte d sig- nal)	Rea d	0d 8	[b1]	[s2]	01, 03, 04
			Writ e	0d 9	[b1] [s2]	-	
Override Analog Outputs Overrides the analog output control to write the outputs directly.	[b1] - State 0 - Normal 1 - Override	-	Rea d	0d a	-	[b1]	01
			Writ e	0d b	[b1]	-	
Analog Output Write Value Contains the value to write to the analog outputs when in override cotrol	[b1] - Output Number 0 - Output1 1 - Output2 [s2] - Value -10000..10000 (ε: 1) (d8f00.2710)	milli- volts (x 0001)	Rea d	0d c	[b1]	[s2]	01
			Writ e	0d d	[b1] [s2]	-	

Serial Port Commands

Parameter	Range of Data Values	Units	Com- mand	Com- mand Data	Respo nse Data	Excep tion Respo nses
Serial Port Baud Rate The drive's serial port baud rate. If the baud rate is changed, it will not take effect until the drive is reset.	[b1] - Rate 0 - 1200 (TP: 1200) 1 - 2400 (TP: 2400)	-	Rea d	0d e	-	[b1]
	2 - 4800 (TP: 4800) 3 - 9600 (TP: 9600) 4 - 19200 (TP: 19200)		Wri te	0d f	[b1]	-
Serial Port Frame Format The drive's serial port baud rate. If the baud rate is changed, it will not take effect until the drive is reset.	[b1] - Frame 0 - 7 data bits, even parity, 1 stop bit (TP: 7D1SEP) 1 - 7 data bits, odd parity, 1 stop bit (TP: 7D1SOP)	-	Rea d	0e 0	-	[b1]
	2 - 8 data bits, no parity, 1 stop bit (TP: 8D1SNP) 3 - 8 data bits, even parity, 1 stop bit (TP: 8D1SEP) 4 - 8 data bits, odd parity, 1 stop bit (TP: 8D1SOP)		Wri te	0e 1	[b1]	-
Software Drive ID The ID used for drive addressing when the rotary DIP switch is set to position "F".	[b1] - ID 0..255	-	Rea d	0e 2	-	[b1]
	(00..ff)		Wri te	0e 3	[b1]	-

Operating Mode Commands

Parameter (Operating Mode Commands)	Range of Data Values	Units	Command	Command Data	Response Data	Exception Responses
Encoder Output Configuration Register The divisor for the motor encoder quadrature output.	[b1] - Divisor 0 - Divide by 1 (TP: ÷ by 1) 1 - Divide by 2 (TP: ÷ by 2) 2 - Divide by 4 (TP: ÷ by 4) 3 - Divide by 8 (TP: ÷ by 8)	-	Read	0f0	-	[b1]
			Write	0f1	[b1]	-
Command Source The signal used for the drive's command source.	[b1] - Source 0 - Analog COMMAND Input (TP: Analog) 1 - Presets (TP: Presets) 2 - Master Encoder (TP: AuxEnc) 3 - Step/Direction (TP: StepDir) 4 - Step+/Step- (TP: Step+/-)		Read	0f2	-	[b1]
			Write	0f3	[b1]	
Drive Mode The flag which determines if the velocity control loop is active.	[b1] - Mode 0 - Velocity (TP: Velocity) 1 - Torque (TP: Torque)		Read	0f4	-	[b1]
			Write	0f5	[b1]	
Velocity Preset The command velocity levels used when the drive is configured with Presets as the Command Source, and Velocity as the drive mode.	[b1] - Preset 0..7 (00..07) [L2] - Velocity - 32767.99998..32767.99998 (ε: 1.53e-5) (80000001..7ffffff)	RPM	Read	0f6	[b1]	[L2]
			Write	0f7	[b1] [L2]	-

Torque Preset The command torque levels used when the drive is configured with Presets as the Command Source, and Torque as the drive mode.	[b1] - Preset 0..7 (00..07) [s2] - Torque -255.992..255.992 (ε: 7.81e-3) (8001..7fff)	- Amps (× 0080)	Read	0f8	[b1]	[s2]	
			Write	0f9	[b1] [s2]	-	01, 03, 04
Analog Input Acceleration Limit The acceleration value used when the analog command input changes while the drive is in velocity mode.	[d1] - Rate 0..2147483647 (ε: 1) (00000000..7fffffff)	RPM/ sec (× 00000 001)	Read	0fa	-	[w1]	
			Write	0fb	[w1]	-	03, 05
Analog Input Deceleration Limit The deceleration value used when the analog command input changes while the drive is in velocity mode.	[d1] - Rate 0..2147483647 (ε: 1) (00000000..7fffffff)	RPM/ sec (× 00000 001)	Read	0fc	-	[d1]	
			Write	0fd	[d1]	-	03, 05
Preset Input Acceleration Limit The acceleration value used when changing between velocity presets. This limit is only used while the drive is in velocity mode and the Command Source is set to Preset input.	[d1] - Rate 0..2147483647 (ε: 1) (00000000..7fffffff)	RPM/ sec (× 00000 001)	Read	0fe	-	[d1]	
			Write	0ff	[d1]	-	03, 05

Preset Input Deceleration Limit The deceleration value used when changing between velocity presets. This limit is only used while the drive is in velocity mode and the Command Source is set to Preset input.	[d1] - Rate 0..2147483647 (ε: 1)	RPM/sec	Read	100	-	[d1]	
	(00000000..7fffffff)	(× 00000001)	Write	101	[d1]	-	03, 05
Tuning Direction Flag Sets or returns the flag which indicates the direction the motor rotates during tuning.	[b1] - Flag 00 - Bi-directional		Read	1A0	-	[b1]	
	01 - Forward 02 - Reverse		Write	1A1	[b1]	-	01, 03, 06

Operating Mode Commands (Continued)

Parameter (Operating Mode Commands)	Range of Data Values	Units	Command	Command Data	Response Data	Exception Responses
Analog Input Acceleration Limits Enable Sets or returns the flag which indicates that acceleration limits are enabled. This flag is only used while the drive is in velocity mode and the Command Source is set to analog COMMAND input.	[b1] - Flag 00 - Disabled 01 - Enabled		Read	1A 2	-	[b1]
			Write	1A 3	[b1]	-
Preset Acceleration Limits Enable Sets or returns the flag which indicates that acceleration limits are enabled. This flag is only used while the drive is in velocity mode and the Command Source is set to Preset input.	[b1] - Flag 00 - Disabled 01 - Enabled		Read	1A 4	-	[b1]
			Write	1A 5	[b1]	-
Change Direction Flag Sets or returns the flag which indicates if the normal direction has been changed (reversed).	[b1] - Flag 00 - Normal 01 - Reversed		Read	1A C	-	[b1]
			Write	1A D	[b1]	-

Alternative Operating Mode Commands

Parameter (Operating Mode Commands)	Range of Data Values	Units	Command	Command Data	Response Data	Exception Responses	
Operating Mode The operating mode for the drive. Usually, the drive is in Normal mode. The mode can be changed for tuning, encoder alignment, and encoder resolution detection.	[b1] - Mode 0 - Normal (TP: Normal) 1 - AutoTuning (TP: Auto) 2 - Manual Tuning (Velocity Step) (TP: Man Vel) 3 - Manual Tuning (Position Step) (TP: Man Pos) 4 - Encoder Alignment (TP: Align) 5 - Encoder Resolution Detection (TP: [not avail])		Read	102	-	[b1]	
			Write	103	[b1]		01, 06
Operating Mode Status Contains status bits for above alternative operating modes.	[w1] - Status Bit 0 - AutoTuning Complete Bit 1 - Encoder Alignment Complete Bit 2 - Motor Index Detected Bit 3 - Master Index Detected Bit 4 - Motor Encoder Resolution Determined Bit 5 - Master Encoder Resolution Determined Bit 6 - AutoTune Failed		Read	104	-	[w1]	
			Write	-	-	-	
Autotune Maximum Current The maximum current used in the autotuning algorithm.	[w1] - Current 0.0078..255.992 (ε: 7.81e-3) (0001..7fff)	Amps (× 0080)	Read	105	-	[w1]	
			Write	106	[w1]	-	03, 04, 05

Autotune Maximum Distance The maximum distance the motor can travel in the autotuning algorithm.	[d1] - Distance 1..2147483647 (ε: 1) (00000001..7ffffff)	Counts (×0001)	Read	107	-	[d1]	
			Write	108	[d1]	-	03, 04, 05
Manual Tune Position Period The period of the square wave used in the position step manual tuning mode.	[w1] - Period 1..32767 (ε: 1) (0001..7fff)	milli-seconds (×0001)	Read	109	-	[w1]	
			Write	10a	[w1]	-	03, 04, 05
Manual Tune Position Step The amplitude of the square wave used in the position step manual tuning mode.	[w1] - Amplitude 1..32767 (ε: 1) (0001..7fff)	Counts (×0001)	Read	10b	-	[w1]	
			Write	10c	[w1]	-	03, 04, 05
Manual Tune Velocity Period The period of the square wave used in the velocity step manual tuning mode.	[w1] - Period 1..32767 (ε: 1) (0001..7fff)	milli-seconds (×0001)	Read	10d	-	[w1]	
			Write	10e	[w1]	-	03, 04, 05
Manual Tune Velocity Step The amplitude of the square wave used in the velocity step manual tuning mode.	[d1] - Amplitude 0.000015..32767.99998 (ε: 1.53e-5) (00000001..7ffffff)	RPM (×0001000)	Read	10f	-	[d1]	
			Write	110	[d1]	-	03, 04, 05

Encoder Alignment Offset The offset of the motor encoder index pulse relative to the rotor phase location. This value is determined automatically in the Encoder Alignment Operating Mode and continually updates while in that mode. It can also be set when not in the Encoder Alignment Operating Mode by using the write command. (Note: The value in this parameter does not affect the commutation. It has to be set by using the Remove Alignment Offset command)	[s1] - Offset -180..179 (ε: 1) (ff4c..00b3)	electrical degrees (× 0001)	Read	11	-	[s1]	
			Write	11	[s1]	-	01

Alternative Operating Mode Commands (Continued)

Parameter (Alternative Operating Mode Com)	Range of Data Values	Units	Command	Command Data	Response Data	Exception Responses
Save Alignment Offset Corrects the encoder alignment by copying the Encoder Alignment Offset value into an encoder alignment compensation parameter. The alignment compensation parameter value is used in correcting the motor encoder input for commutation.	[No Data] (TP: ↑ to Rmv)		Read	-	-	
			Write	113	-	-
Motor Encoder Resolution The measured motor encoder counts between index pulses when in the Encoder Resolution Detection Operating Mode.	[w1] - Resolution 0..32767 (ε: 1) (0000..7fff)	counts (× 0001)	Read	114	-	[w1]
			Write	-	-	-
Master Encoder Resolution The measured master encoder counts between index pulses when in the Encoder Resolution Detection Operating Mode.	[w1] - Resolution 0..32767 (ε: 1) (0000..7fff)	counts (× 0001)	Read	115	-	[w1]
			Write	-	-	-
Motor Index Position The last recorded position of the motor encoder index.	[w1] - Position 0..65535 (ε: 1) (0000..ffff)	counts (× 0001)	Read	116	-	[w1]
			Write	-	-	-

Master Index Position The last recorded position of the master encoder index.	[w1] - Position 0..65535 (ε: 1) (0000..ffff)	mas- ter counts	Read	11 7	-	[w1]	
		(× 0001)	Write	-	-	-	

Runtime Command and Control Commands

Parameter	Range of Data Values	Units	Com- mand	Com- mand Data	Respo nse Data	Excep tion Respo nses
Reset Drive Resets the drive hardware and reboots the drive's processors.	[No Data] (TP: ↑to Reset)	-	Rea	-	-	
			d	-	-	
			Wri	12	-	06
			te	0	-	-
Software Drive Enable/Disable If set to Enable Drive and the ENABLE input is active, the drive is enabled. If set to Disable Drive or the ENABLE input is not active, the drive is disabled.	[b1] - State 0 - Disable Drive (TP: Dis-able) 1 - Enable Drive (TP: Enable)	-	Rea	12	-	[b1]
			d	1	-	-
			Wri	12	[b1]	01
			te	2	-	-
Torque Setpoint The torque command value used when the Drive Mode is Torque, and the Setpoint Control is Enabled.	[s1] - Torque -255.992..255.992 (ε: 7.81e-3) (8001..7fff)	Amps (× 0080)	Rea	12	-	[s1]
			d	3	-	-
			Wri	12	[s1]	04
			te	4	-	-
Velocity Setpoint The velocity command value used when the Drive Mode is Velocity, and the Setpoint Control is Enabled.	[L1] - Velocity -32767.99998..32767.99998 (ε: 1.53e-5) (80000001..7fffffff)	RPM (× 00010000)	Rea	12	-	[L1]
			d	5	-	-
			Wri	12	[L1]	04
			te	6	-	-

Setpoint Acceleration The acceleration value used when the Velocity Setpoint changes, and the Setpoint Control is Enabled.	[d1] - Rate 0..2147483647 (ε: 1) (00000000..7fffffff)	RPM/ sec (× 00000 001)	Rea	12	-	[d1]	
			d	7			
Setpoint Control Enables or disables the setpoint control.	[b1] - State 0 - Disable Setpoint Control (TP: Normal) 1 - Enable Setpoint Control (TP: CtlPanl)	-	Rea	12	-	[b1]	
			d	9			
Reset Faults Resets the fault detection circuitry.	[No Data] (TP: ↑toReset)	-	Rea	-	-	-	
			d				
			Wri	12	[d1]	-	03, 05
			te	8			
			Wri	12	[b1]	-	01
			te	a			
			Wri	12	-	-	
			te	b			

Runtime Status Commands

Parameter (Runtime Status Commands)	Range of Data Values	Units	Command	Command Data	Response Data	Exception Responses
Packed Drive Status The status of various flags in the drive. This status is repeatedly updated.	[d1] - Status Bit 0 - In-Position Bit 1 - Within Position Window Bit 2 - Zero Speed Bit 3 - Within Speed Window Bit 4 - Positive ILimit Bit 5 - Negative ILimit Bit 6 - At Speed Bit 7 - Drive Enabled Bit 8 - DC Bus Charged Bit 9 - Fault Disable Bit 10 - Fault Decel/Disable Bit 11 - Latched Fault Warning Bit 12 - Unlatched Fault Warning Bit 14 - Brake Active Bit 15 - Drive Ready Bit 16 - Torque Mode Bit 17- Integrator Inhibit Bit 18 - Follower Enable Bit 19 - Forward Clamp Bit 20 - Reverse Clamp Bit 21 - Analog Override Bit 22 - Preset Select Line A Bit 23 - Preset Select Line B Bit 24 - Preset Select Line C Bit 30 - Reset Faults Bit 31 - Enable Active	-	Read	13-14	-	[d1]
			Write	-	-	-

<p>Fault Status Identifies the present state of the possible fault conditions.</p> <p>If a specific Fault Group Mask Is set to unlatched warning, the appropriate bit is not latched in this register and may clear when the condition is removed.</p> <p>If the specific Fault Group Mask is <i>not</i> set to unlatched warning, the appropriate bit is latched in this register and will remain set until the drive is reset.</p>	<p>[d1] - Status</p> <p>Bit 0 - +24VDC Fuse Blown</p> <p>Bit 1 - +5VDC Fuse Blown</p> <p>Bit 2 - Encoder Power Fuse Blown</p> <p>Bit 3 - Motor Overtemperature</p> <p>Bit 4 - IPM Fault (Overtemperature/Overcurrent/Short Circuit)</p> <p>Bit 5 - Channel IM Line Break</p> <p>Bit 6 - Channel BM Line Break</p> <p>Bit 7 - Channel AM Line Break</p> <p>Bit 8 - Bus Undervoltage</p> <p>Bit 9 - Bus Overvoltage</p> <p>Bit 10 - Illegal Hall State</p> <p>Bit 11 - Sub processor Unused Interrupt</p>	-	Rea d	13 5	-	[d1]	
---	---	---	----------	---------	---	------	--

	Bit 12 - Main processor Unused Interrupt Bit 16 - Excessive Average Current Bit 17 - Overspeed Bit 18 - Excess Following Error Bit 19 - Motor Encoder State Error Bit 20 - Master Encoder State Error Bit 21 - Motor Thermal Pro- tection Bit 22 - IPM Thermal Pro- tection Bit 27 - Enabled with No Motor Selected Bit 28 - Motor Selection not in Table Bit 29 - Personality Write Error Bit 30 - Service Write Error Bit 31 - CPU Communica- tions Error	Write	-	-	-		
--	--	-------	---	---	---	--	--

Runtime Status Commands (Continued)

Parameter (Runtime Status Commands)	Range of Data Values	Units	Command	Command Data	Response Data	Exception Responses
<p>Run State Identifies the present state of the drive and possible fault conditions. The reported faults are only ones with Fault Mask values set to Disable Drive or Decel, Then Disable Drive.</p> <p>This command is added to support TouchPad background status polling operation. This implies that other products which use the touchpad will need to adhere to the following format.</p> <p>The state values 1..127 are reserved for fault indications. These values will cause the fault to be shown on the touchpad.</p>	<p>[c1] - State -01 - Drive Enabled 00 - Drive Ready 01 - +24VDC Fuse Blown 02 - +5VDC Fuse Blown 03 - Encoder Power Fuse Blown 04 - Motor Overtemperature 05 - IPM Fault (Overtemperature/Overcurrent/Short Circuit) 06 - Channel IM Line Break 07 - Channel BM Line Break 08 - Channel AM Line Break 09 - Bus Undervoltage 10 - Bus Overvoltage 11 - Illegal Hall State</p>	-	Read	136	-	[c1]

<p>The values 0..-128 are reserved for non-fault state information which is to be indicated, but not shown as a fault by the touchpad.</p>	<p>12 - Sub processor Unused Interrupt 13 - Main processor Unused Interrupt 17 - Excessive Average Current 18 - Overspeed 19 - Excess Following Error 20 - Motor Encoder State Error 21 - Master Encoder State Error 22 - Motor Thermal Protection 23 - IPM Thermal Protection 28 - Enabled with No Motor Selected 29 - Motor Selection not in Table 30 - Personality Write Error 31 - Service Write Error 32 - CPU Communications Error</p>		Write	-	-	-	
<p>Digital Input States Identifies the present state of the digital inputs.</p>	<p>[w1] - States Bit 0 - RESET FAULTS Input State Bit 1 - ENABLE Input State Bit 2 - INPUT1 Input State Bit 3 - INPUT2 Input State Bit 4 - INPUT3 Input State Bit 5 - INPUT4 Input State</p>	-	Read	13 7	-	[w1]	
<p>Digital Output States Identifies the present state of the digital outputs.</p>	<p>[w1] - States Bit 0 - READY Output State Bit 1 - BRAKE Output State Bit 2 - OUTPUT1 Output State Bit 3 - OUTPUT2 Output State Bit 4 - OUTPUT3 Output State Bit 5 - OUTPUT4 Output State</p>	-	Read	13 8	-	[w1]	
			Write	-	-	-	-

Runtime Data Commands

Parameter (Runtime Data Commands)	Range of Data Values	Units	Command	Command Data	Response Data	Exception Responses
Reset Peaks Resets the peak detection firmware for positive position error peak, negative position error peak, positive torque current, and negative current.	[No Data] (TP: ↑toReset)	-	Read	-	-	
			Write	140	-	-
COMMAND Input The command input value before scaling and offsetting.	[s1] - Value -10000.. 10000 (ε: 1) (d8f0..2710)	milli-volts (× 0001)	Read	141	-	[s1]
			Write	-	-	-
Positive ILimit Input The +ILimit input value.	[s1] - Value -255.992..255.992 (ε: 7.81e-3) (8001..7fff)	Amps (× 0080)	Read	142	-	[s1]
			Write	-	-	-
Negative ILimit Input The -ILimit input value.	[s1] - Value -255.992..255.992 (ε: 7.81e-3) (8001..7fff)	Amps (× 0080)	Read	143	-	[s1]
			Write	-	-	-
Analog Output The analog output values.	[b1] - Number 0 - Output 1 1 - Output 2 [s2] - Value -10000..10000 (ε: 1) (d8f0..2710)	milli-volts (× 0001)	Read	144	[b1]	[s2]
			Write	-	-	-
Motor Position The value of the motor encoder register.	[L1] - Value -2147483647..2147483647 (ε: 1)	Counts	Read	145	-	[L1]

	(80000001..7fffffff)	(× 00000 001)	Write	-	-	-	
Master Position The value of the master input register.	[L1] - Value -2147483647..2147483647 (ε: 1) (80000001..7fffffff)	Mas- ter Counts (× 00000 001)	Read	14	-	[L1]	
			Write	-	-	-	
Position Command The position command input to the position loop, which is the master position, after gearing and slew rate limiting.	[L1] - Value -2147483647..2147483647 (ε: 1) (80000001..7fffffff)	Counts (× 00000 001)	Read	14	-	[L1]	
			Write	-	-	-	
Position Error The difference between the Position Command and the Motor Position.	[L1] - Value -2147483647..2147483647 (ε: 1) (80000001..7fffffff)	Counts (× 00000 001)	Read	14	-	[L1]	
			Write	-	-	-	
Position Positive Peak Error The maximum amount the Position Command lead the Motor Position.	[L1] - Value 0..2147483647 (ε: 1) (00000000..7fffffff)	Counts (× 00000 001)	Read	14	-	[L1]	
			Write	-	-	-	
Position Negative Peak Error The maximum amount the Position Command lagged the Motor Position.	[L1] - Value -2147483647..0 (ε: 1) (80000001..00000000)	Counts (× 00000 001)	Read	14	-	[L1]	
			Write	-	-	-	

Velocity Command The command value to the velocity loop.	[L1] - Value - 32767.99998..32767.99998 (ε: 1.53e-5) (80000001..7ffffff)	RPM (× 00010 000)	Read	14	-	[L1]	
			Write	-	-	-	
Motor Velocity The feedback value to the velocity loop.	[L1] - Value - 32767.99998..32767.99998 (ε: 1.53e-5) (80000001..7ffffff)	RPM (× 00010 000)	Read	14	-	[L1]	
			Write	-	-	-	

Runtime Data Commands (Continued)

Parameter (Runtime Data Commands)	Range of Data Values	Units	Command	Command Data	Response Data	Exception Responses
Velocity Error The difference between Velocity Command and Motor Velocity.	[L1] - Value - 32767.99998..32767.99998 (ϵ : 1.53e-5) (80000001..7ffffff)	RPM (\times 00010 000)	Read	14	-	[L1]
			Write	-	-	
Current Command The command value to the current loop.	[s1] - Value -255.992..255.992 (ϵ : 7.81e-3) (8001..7fff)	Amps (\times 0080)	Read	14	-	[s1]
			Write	-	-	
Average Current The average value of the Current Command(?).	[s1] - Value -255.992..255.992 (ϵ : 7.81e-3) (8001..7fff)	Amps (\times 0080)	Read	14	f	[s1]
			Write	-	-	
Current Positive Peak The largest positive value of the Current Command(?).	[s1] - Value 0.0..255.992 (ϵ : 7.81e-3) (0000..7fff)	Amps (\times 0080)	Read	15	0	[s1]
			Write	-	-	
Current Negative Peak The largest negative value of the Current Command(?).	[s1] - Value -255.992..0.0 (ϵ : 7.81e-3) (8001..0000)	Amps (\times 0080)	Read	15	1	[s1]
			Write	-	-	
Bus Voltage The measured voltage of the DC bus.	[w1] - Value 0..32767 (ϵ : 1) (0000..7fff)	Volts (\times 0001)	Read	15	2	[w1]
			Write	-	-	
Field Current The calculated field current for induction motors.	[s1] - Value -255.992..255.992 (ϵ : 7.81e-3)	Amps	Read	15	3	[s1]

	(8001..7fff)	(× 0080)	Write	-	-	-	
Torque Current The calculated torque current.	[s1] - Value -255.992..255.992 (ε: 7.81e-3) (8001..7fff)	Amps (× 0080)	Read	15	-	[s1]	
			Write	-	-	-	
R-Phase Current The calculated R-Phase current.	[s1] - Value -255.992..255.992 (ε: 7.81e-3) (8001..7fff)	Amps (× 0080)	Read	15	-	[s1]	
			Write	-	-	-	
T-Phase Current The calculated T-Phase current.	[s1] - Value -255.992..255.992 (ε: 7.81e-3) (8001..7fff)	Amps (× 0080)	Read	15	-	[s1]	
			Write	-	-	-	
Field Voltage Command The field voltage command for induction motors.	[s1] - Value -255.992..255.992 (ε: 7.81e-3) (8001..7fff)	Volts (× 0080)	Read	15	-	[s1]	
			Write	-	-	-	
Torque Voltage Command The torque voltage command.	[s1] - Value -255.992..255.992 (ε: 7.81e-3) (8001..7fff)	Volts (× 0080)	Read	15	-	[s1]	
			Write	-	-	-	
Average Motor Current The average current seen by the motor.	[s1] - Value 0.0..255.992 (ε: 7.81e-3) (0000..7fff)	Amps (× 0080)	Read	15	-	[s1]	
			Write	-	-	-	

Runtime Data Collection Commands

Parameter (Runtime Data Collection Commands)	Range of Data Values	Units	Command	Command Data	Response Data	Exception Responses	
Channel 1 Source The signal values returned in channel 1 of Collected Data.	[b1] - Signal Number 0 - Current Command 1 - Current Average Command 2 - Current Positive Peak 3 - Current Negative Peak 4 - Positive ILimit 5 - Negative ILimit 6 - Motor Velocity 7 - Velocity Command 8 - Velocity Error 9 - Motor Position 10 - Position Command Slewed 11 - Position Error 12 - Position Peak Positive Error 13 - Position Peak Negative Error 20 - Master Position 21 - Position Loop Output 22 - Velocity Loop Output 23 - Filter Output 24 - Notch Output 25 - R Phase Current 26 - T Phase Current 27 - Torque Current 28 - Field Current 29 - Torque Voltage 30 - Field Voltage 31 - Scaled A/D Command Value 32 - Bus Voltage		Read	160	-	[b1]	
			Write	161	[b1]		01
Channel 2 Source The signal values returned in channel 2 of Collected Data.	[b1] - Signal Number See <i>Channel 1 Source</i> for selections		Read	162	-	[b1]	
			Write	163	[b1]		01

Trigger Source The signal used to trigger the data collection depending on the Trigger Mode.	[b1] - Signal Number See <i>Channel 1 Source</i> for selections		Read	16	-	[b1]	
			Write	16	[b1]		01
Timebase The time between samples returned in Collected Data.	[w1] - Value 0.0..13107.0 (ϵ : 0.2) (0000..ffff)	milli-seconds (\times 0005)	Read	16	-	[w1]	
			Write	16	[w1]	-	
Trigger Mode Determines how data is collected when Arm Triggering command is sent.	[b1] - Mode 0 - Trigger immediately 1 - Trigger on positive transition of trigger source 2 - Trigger on negative transition of trigger source.		Read	16	-	[b1]	
			Write	16	[b1]	-	01
Trigger Threshold The value which must be crossed on the Trigger Source when the Trigger Mode is set to trigger in a transition.	[s1] - Value (8001..7fff)	Drive Internal Units (Dependent on selected trigger source)	Read	16	-	[s1]	
			Write	16	[s1]	-	04
Arm Triggering Arms the data collection to begin collecting data when the next trigger event occurs.	[No Data]		Read	-	-	-	
			Write	16	-	-	

Trigger Status The status of the data collection process.	[b1] - Status 0 - Waiting for Trigger to Occur 1 - Triggered, Collecting Data 2 - Data Collection Complete		Read	16	-	[b1]	
			Write	-	-	-	

Runtime Data Collection Commands (Continued)

<p>Collected Data The data collected from the last trigger event.</p>	<p>[b1] - Group 0 - Channel 1, Samples 1 through 16 1 - Channel 1, Samples 17 through 32 2 - Channel 1, Samples 33 through 48 3 - Channel 1, Samples 49 through 64 4 - Channel 1, Samples 65 through 80 5 - Channel 1, Samples 81 through 96 6 - Channel 1, Samples 97 through 112 7 - Channel 1, Samples 113 through 128 8 - Channel 2, Samples 1 through 16 9 - Channel 2, Samples 17 through 32 a - Channel 2, Samples 33 through 48 b - Channel 2, Samples 49 through 64 c - Channel 2, Samples 65 through 80 d - Channel 2, Samples 81 through 96 e - Channel 2, Samples 97 through 112 f - Channel 2, Samples 113 through 128 [s2]..[s17] Requested data Error Code 4 - Data Accepted After Limiting to Minimum.</p>	<p>Read</p>	<p>16</p>	<p>[b1]</p>	<p>[s2].. [s17]</p>	<p></p>	<p></p>
		<p>Write</p>	<p>-</p>	<p>-</p>	<p>-</p>	<p>04</p>	<p></p>

NOTES

APPENDIX B Press Transfer ASFBS

Introduction

This set of Application Specific Function Blocks (ASFBS) is designed to generate a profile for a slave axis in a press application. As the master axis moves, the slave axis moves in, dwells, and moves out in one master rotation (i.e., 360 degrees). A variation that could be supported as well would be for the slave axis to move in and dwell for each master rotation and that motion is repeated several times before the slave moves out to its initial position.

This profile can have different shapes. It can be triangular (the slave accelerates and decelerates without achieving a constant velocity) or trapezoidal (the slave accelerates to a maximum velocity for a portion of its motion before it decelerates). The acceleration and deceleration can also be configured for an 'scurve' where the corners of the motion transitions are smoothed.

To obtain a slave axis profile for two slave moves for one master axis rotation, the M_PR2MV function block is called from the main application ladder. This function block has a number of inputs to direct the profile generation:

- RR - an array of structures configured in the format required by the RATIO_RL function block. This set of functions blocks is designed for a RATIO_RL application. RATIO_RL usage is detailed in the PiCPro function block reference guide. This structure has the following format:
- MAST_DIS - the distance of the master motion in this segment (in FU).
- SLAV_DIS - the distance of the slave motion in this segment (in FU).
- K1 - the K1 coefficient in the polynomial equation for RATIO_RL.
- K2 - the K2 coefficient in the polynomial equation for RATIO_RL.
- K3 - the K3 coefficient in the polynomial equation for RATIO_RL.
- SPARE - reserved for future use.
- FLAGS - indicate the execution of the polynomial function of RATIO_RL.

This RR input to the function block must be defined as an array of structures in the calling function (which is usually the main application ladder). The actual size required for this array will depend upon the type of profile required; an scurve profile will have more segments than a simpler constant acceleration profile. The number of segments within the profile will be as follows for each move: acceleration portion (1 for no scurve, 3 with scurve), constant velocity portion (0 or triangular, 1 for trapezoidal), deceleration portion (1 for no scurve, 3 with scurve), and dwell portion (1 if a dwell is required). For example, for the application of M_PR2MV, the size of the RR array **must be at least 17** to encompass the various combinations because 16 segments will be required.

The sizing of this array is very important. If the array is sized too small, run-time errors within the application are likely to occur (because other variables in PiC memory will be written during the calculations since the internal function blocks will assume enough memory has been allocated by the main application ladder).

- MOV1 and MOV2 - an input structure describes each of the two slave moves required. This structure provides the following information for each move:
 - STRT_ANG - the angle of the master axis at the start of the slave's move (in degrees).
 - STOP_ANG - the angle of the master axis at the end of the slave's move (in degrees).
 - SLV_MOVE - the distance of the slave move (specified in input units that can be scaled).
 - MAX_V - the maximum velocity that can be allowed for this slave move (specified as a ratio of slave FU to master FU).
 - PCT_J - the percent of maximum possible jerk to be used for this slave move (in a range of 0.0 to 100.0). A value of zero means no jerk, and therefore, no scurve.
 - PCT_A - the percent of maximum possible acceleration to be used for this slave move (in a range of 0.0 to 100.0).
 - TRI_ONLY - a boolean flag to indicate a triangular profile is desired.
 - SCURVE - a boolean flag to indicate the smoothed scurve accel/decel is desired.
 - MDST - the number of master feedback units in one cycle or rotation.
 - MSCL - the number of master feedback units per input unit in the input MOVx structure (i.e., the start and stop angles).
 - SSCL - the number of slave feedback units per input unit in the input MOVx structure (i.e., the slave distance moved between two master angles).
 - VLIM - the maximum allowable velocity for this master/slave application (specified as a ratio of slave FU to master FU). This limit is one that would reflect the inherent machine limitations. The individual move structures specify the maximum velocity that is desired for that specific move; that velocity for a move cannot exceed this VLIM value. This VLIM value would be one that is entered once for the application; the velocities for the individual moves could be specified via the user interface.

The input move structure can indicate the intent of a triangular slave move (TRI_ONLY). However, if the other parameters result in a trapezoidal profile achieving the required slave motion, this function block will generate the appropriate trapezoidal profile and it will set a boolean output that indicates this change in behavior. If the main ladder must get a triangular profile then it can take the appropriate actions, such as providing the user interface with a signal that the move parameters must be specified again. If the main ladder will tolerate either triangular or trapezoidal profile but it prefers the triangular profile then this is supported.

If the combination of parameters prevents the generation of a profile then the function block returns an appropriate error indicator. The main ladder must make sure that no errors were detected before trying to apply the generated profile.

The input move structure can direct the profile shape by specifying the percent of maximum acceleration (PCT_A). 100% of maximum acceleration would approximate a step function - immediately get to the maximum velocity for the slave's move (in most cases an unacceptable response for the slave). 0% of maximum acceleration would obtain the minimum slope for the slave's acceleration and still achieve the required slave motion. Values within this range obtain an intermediate behavior.

There is no separate deceleration rate provided as an input, so the deceleration portion of the profile will use the same parameters as the acceleration portion. However, there is an internal function block to generate the deceleration portion of the profile so it could be possible (but not supported at this time) for the generated profile to contain different acceleration and deceleration configurations.

The input move structure can indicate the intent of smoothed scurve acceleration and deceleration portions of the slave profile (SCURVE). This also requires a percentage of jerk to be specified (PCT_J). Maximum jerk (100%) would obtain no scurve behavior because there would be only constant acceleration. Minimum jerk (0.1%) would obtain the smoothest acceleration portion of the profile with no constant acceleration but with the highest peak acceleration rate.

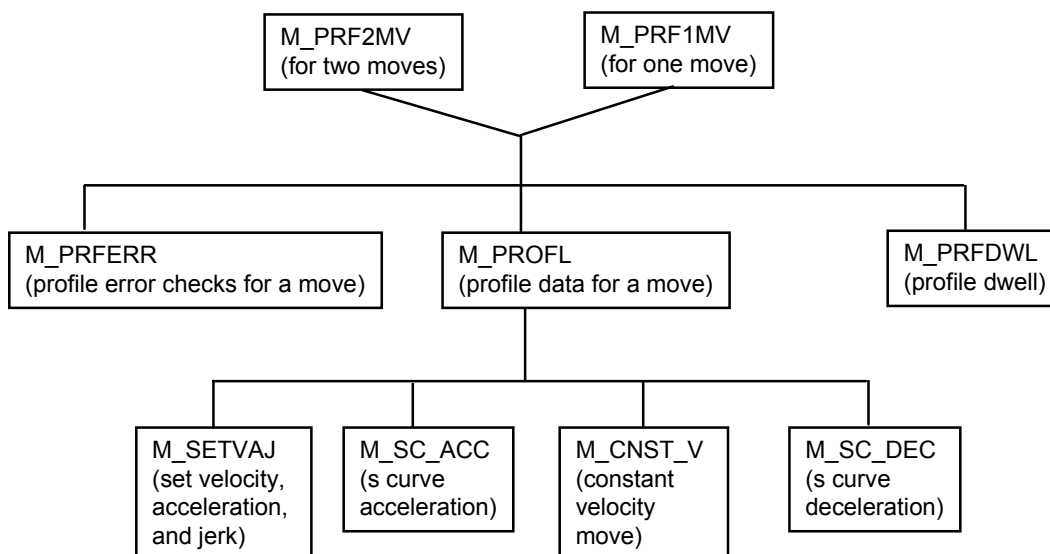
This is the set of function blocks whose purpose is to generate a slave profile for a press application.

- M_PRF2MV - this function block generates a slave axis profile for two slave moves for one master axis rotation. It will in turn call the M_PRFERR, M_PROFL and M_PRFDWL function blocks for each of the two slave axis moves in the profile.
- M_PRF1MV - this function block generates a slave axis profile for one slave move for one master axis rotation. It will in turn call the M_PRFERR, M_PROFL and M_PRFDWL function blocks to generate the profile. The M_PRF1MV function block has the same inputs as M_PRF2MV except that only one move is handled in the profile rather than two.
- M_PRFERR - this function block checks the validity of the input move's parameters.
- M_PROFL - this function block generates the portion of the profile when the slave is moving. It will in turn call the M_SETVAJ, M_SC_ACC, M_CNST_V and M_SC_DEC function blocks.
- M_PRFDWL - this function block generates the portion of the profile when the master axis is moving but the slave is not. This block is required because the RATIO_RL profile must account for all the master counts so that the profile can be repeated (i.e., for each master rotation, the slave performs the same profile). Therefore if the slave is moving only part of the time (which will occur in many press applications), then a portion of the profile contains the master's motion that has no corresponding slave motion. Also, because the real to integer calculations being performed during the generation of the profile might result in rounding, there could be a few counts of master or slave axis motion that could not be incorporated into the main part of the profile. Those remaining counts, if any, can be accounted for during this portion of the profile.
- M_SETVAJ - this function block calculates the acceleration, velocity and jerk to be used for this move. This function block also determines whether the move's parameters can support a triangular profile or whether it must be a trapezoidal profile.
- M_SC_ACC - this function block adds the acceleration portion of the profile into the main structure (for a RATIO_RL). Depending on the move's parameters, the acceleration will be constant acceleration or it will be an scurve (i.e., smoothed acceleration).
- M_CNST_V - this function block adds the constant velocity portion of the trapezoidal profile into the main structure (for a RATIO_RL). The constant velocity portion is the 'flat top' of the profile. A triangular profile does not have a constant velocity portion.

- M_SC_DEC - this function block adds the deceleration portion of the profile into the main structure (for a RATIO_RL). Depending on the move's parameters, the deceleration will be constant deceleration or it will be an s curve (i.e., smoothed deceleration).

If a specific application requires a different combination of slave moves for one (or more) master moves, these function blocks are the 'building blocks' for that application. The M_PRF1MV function block illustrates how to convert a defined move of a slave axis (i.e., the move structure) into a profile for RATIO_RL. Its contents can be merged into your application and then modified to concatenate other slave moves, each with its own definition specified in a move structure, into the single profile. Note that if a longer profile is to be generated you must make sure that the array of structures for the profile in the application is adequately sized.

The following flow chart shows the relationship of the function blocks to each other.



Note: The M_PRF2MV function block contains two each of M_PRFERR, M_PROFL, and M_PRFDWL. There is one of these function blocks for each of the two moves.

M_PRF2MV

2 slave moves for master

USER/M_PROFL

NAME	
M_PRF2MV	
EN00	OK
RR	ERR1
MOV1	TRP1
MOV2	AMX1
MDST	VMX1
MSCL	ERR2
SSCL	TRP2
VLIM	AMX2
	VMX2

- Inputs:**
- EN00 (BOOL) - enables execution (Typically one-shot)
 - RR (STRUCTURE) - array of structures to be used for RATIO_RL profile.
 - MOV1 (STRUCTURE) - structure containing input data for first move
 - MOV2 (STRUCTURE) - structure containing input data for second move
 - MDST (DINT) - master feedback units/cycle where cycle is divided into 360 degrees
 - MSCL (REAL) - master feedback units/input unit, typically set to 1.0
 - SSCL (REAL) - Slave feedback units/input unit
 - VLIM (REAL) - Maximum allowable velocity
- Outputs:**
- OK (BOOL) - execution completed without error
 - ERR1 (BYTE) - error number for first move
 - TRP1 (BOOL) - move 1 changed to a trapezoid to achieve move
 - AMX1 (REAL) - maximum acceleration rate calculated for move 1
 - VMX1 (REAL) - maximum velocity rate calculated for move 1
 - ERR2 (BYTE) - error number for second move.
 - TRP2 (BOOL) - move 2 changed to a trapezoid to achieve move
 - AMX2 (REAL) - maximum acceleration rate calculated for move 2
 - VMX2 (REAL) - maximum velocity rate calculated for move 2

```
<<INSTANCE NAME>>:M_PRF2MV(EN00 := <<BOOL>>, RR := <<MEMORY AREA>> MOV1 := <<MEMORY AREA>>, MOV2 := <<MEMORY AREA>>, MDST := <<DINT>>, MSCL := <<REAL>>, SSCL := <<REAL>>, VLIM := <<REAL>>, OK => <<BOOL>>, ERR1 => <<BYTE>>, TRP1 => <<BOOL>>, AMX1 => <<REAL>>, VMX1 => <<REAL>>, ERR2 =>
```

```
<<BYTE>>, TRP2 => <<BOOL>>, AMX2 => <<REAL>>, VMX2 =>
<<REAL>>);
```

The M_PRF2MV and M_PRF1MV application specific function blocks are used for applications where there is a cyclic master moving in one direction and the slave is performing a distance move (or two distance moves) and an optional dwell during the master cycle. The function blocks take as inputs the distance the slave should move and over what portion of the master cycle it should move and converts that information into a RATIO_RL profile structure. M_PRF1MV is used if there is one move (with or without a dwell) during a master cycle. M_PRF2MV is used if there are two moves (with or without dwell) during a master cycle. The slave moves can be different directions or the same. The slave moves can be trapezoidal or triangular, and the functions allow a move to be smoothed with an “S-Curve” -like acceleration by input selection.

The information given to the function about the move such as slave distance and portion of the master cycle to move is converted to an array of structures that is used as an input to the RATIO_RL function. The structure must be formatted as shown in Table 2-1 and must have an array size of at least 17 (0..16) because the function clears out the structure data before it calculates the profile. This structure is filled in by the function block so it is not necessary for you to enter any data into it.

Table 2-1. Ratio Real Structure

Name	Data Type	Definition
RR	STRUCTURE	Array of Structures to be used for profile
.MAST_DIS	DINT	Master move distance in feedback units
.SLAV_DIS	DINT	Slave move distance relative to Master in feedback units
.K1	LREAL	VELOCITY co-efficient for polynomial
.K2	LREAL	ACCELERATION co-efficient for polynomial (A/2)
.K3	LREAL	JERK co-efficient for polynomial (J/6)
.SPARE	LREAL	Spare
.FLAGS	DWORD	Move type flags. Bits 2 & 3 = 0 for polynomial

The function assumes a rotary cyclic master moving in a single direction. The value at the MDST input indicates the number of master feedback units for the cycle. This is divided into 360 degrees for a cycle. The master information for the profile is entered in degrees. The MSCL input is typically set to 1.

The slave information for the profile is entered in slave input units. The SSCL input defines the number of slave feedback units per slave input unit. For example, if you want to enter slave data in revolutions and there are 8000 feedback units per revolution, you would enter a value of 8000.0 at the SSCL input and a value of 0.25 for slave distance in the move structure to move one quarter of a rev.

The shape of the slave profile is determined by the input parameters at MOV1 and MOV2. Each slave move can have a zero-speed dwell after it. The format for the MOV1 and MOV2 structures is shown in Table 2-2. MOV2 has the same structure format as MOV1.

The VLIM input to the function block is used as a limit on the master/slave ratio that is allowed. This value is compared with the individual MAX_V inputs in the move structures and an error is set if the individual MAX_V inputs exceed VLIM. The maximum ratio can be multiplied by the speed of the master to find the maximum velocity of the slave axis. Using master/slave ratio as opposed to velocity is a better method for applications where the master can run at variable speeds - in this case if the master is moving slower the slave will do so as well.

Table 2-2. Move Structure

Name	Data Type	Definition
MOV1	STRUCTURE	Structure containing move's input data
.STRT_ANG	REAL	Angle of master axis at start of slave move
.STOP_ANG	REAL	Angle of master axis at end of slave move
.SLV_MOVE	REAL	Distance of slave move
.MAX_V	REAL	Maximum desired velocity of the slave axis
.PCT_J	REAL	Percent of maximum possible jerk to be used
.PCT_A	REAL	Percent of maximum possible accel to be used
.TRI_ONLY	BOOL	Triangular profile desired
.SCURVE	BOOL	Smoothed scurve acc/dec desired

The profile will be calculated as a triangle if possible. If it is not possible to achieve a triangular profile because of the limit on ratio (MAX_V) then the profile will become trapezoidal in shape with MAX_V as the limit for master/slave ratio.

Regardless of the value of STRT_ANG, the slave move will be put in the first segment of the RATIO_RL profile. Therefore, if you want the slave to start its move at a specific master position, you should use the MSTR input of the RATIO_RL profile.

The acceleration parameter for the move is set to a value between 0 and 100%. 0% acceleration is the slowest amount and the move will be as close to triangular as possible. 100% acceleration is the fastest amount and the move will be almost rectangular in shape.

Any profile can be “smoothed” by adjusting the percent jerk. This will not change the basic shape of the profile, but will change the acceleration and deceleration

portions of the move to resemble an “S-Curve”. The percent jerk parameter determines how much the acceleration (or deceleration) segment of the move will be smoothed. A 0% value is the most smoothing, and a 100% value will yield no smoothing. The maximum velocity of the move is not affected by “smoothing” nor is the average acceleration. It only affects how the acceleration (and deceleration) will be applied to obtain this velocity.

The ASFB checks the input data for errors. If an error is detected the profile will not be generated and the OK output will not be set. ERR1 and ERR2 are byte values. Each error that can occur is assigned a bit number in the byte. It is possible for more than one error to be set at a time. The bit assignments for ERR1 and ERR2 are shown in Table 2-3. The function BYT2BOOL is helpful in checking for specific errors.

Maximum velocity can be limited to allow an automatically adjusting profile which will be triangular until the maximum velocity is reached. It will then spread into a trapezoid using the minimum acceleration to achieve the move.

It is also possible to set up each move as a constant accel/decel triangular move or a trapezoidal move using operator inputs for the desired shape. Acceleration can be adjusted to change the shape from triangular to nearly rectangular by increasing the acceleration percent.

For any such profile, the acceleration can be "smoothed" by adjusting the jerk percent. This will not change the basic shape of the profile, but will change the acceleration and deceleration portions of the move to resemble an "S-Curve".

The percentage of jerk corresponds inversely to the portion of the Acceleration (or Deceleration) segment of the move which will be smoothed, until 100% equals no S-Curve. At minimum jerk, there is no constant accel portion. This will correspond to the highest acceleration rate. Maximum velocity during a move is not affected by “smoothing” or jerk, nor is the average acceleration. It only affects how the acceleration (and deceleration) will be applied to obtain this velocity.

The first portion of the ASFB checks the input data for any detectable errors. The bit assignments for ERR1 and ERR2 are shown in Table 2-3. The function BYT2BOOL is helpful in checking for specific errors.

Table 2-3. Error Definitions

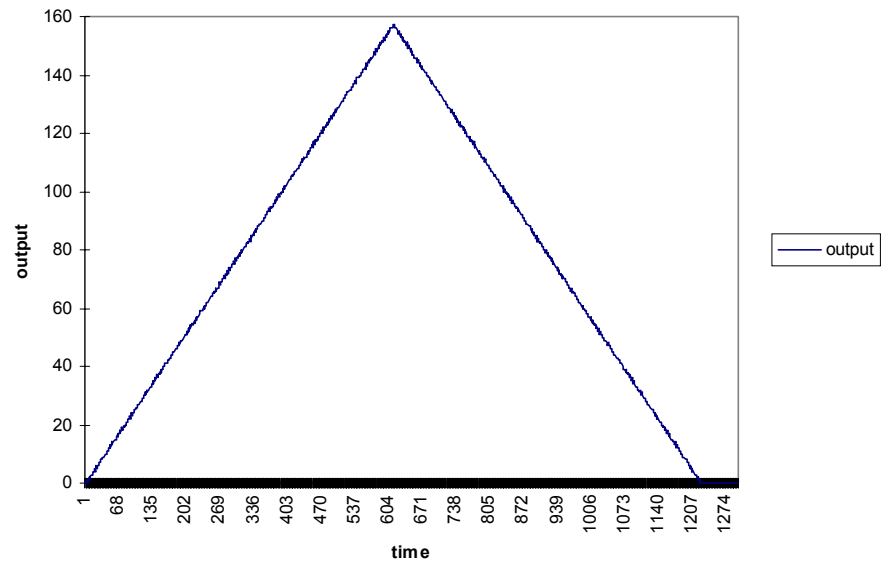
Fault Bit Number	Description
0	Starting angle is not within -180 to 360 degrees
1	Ending angle is not within -180 to 360 degrees
2	Acceleration percent value not within 0.0 to 100.0
3	Jerk percentage value not within 0.0 to 100.0
4	Desired velocity limit higher than allowed
5	Desired velocity limit is zero
6	Master Move 1 overlaps Master Move 2
7	Cannot set-up move with input parameters given

Example Profiles

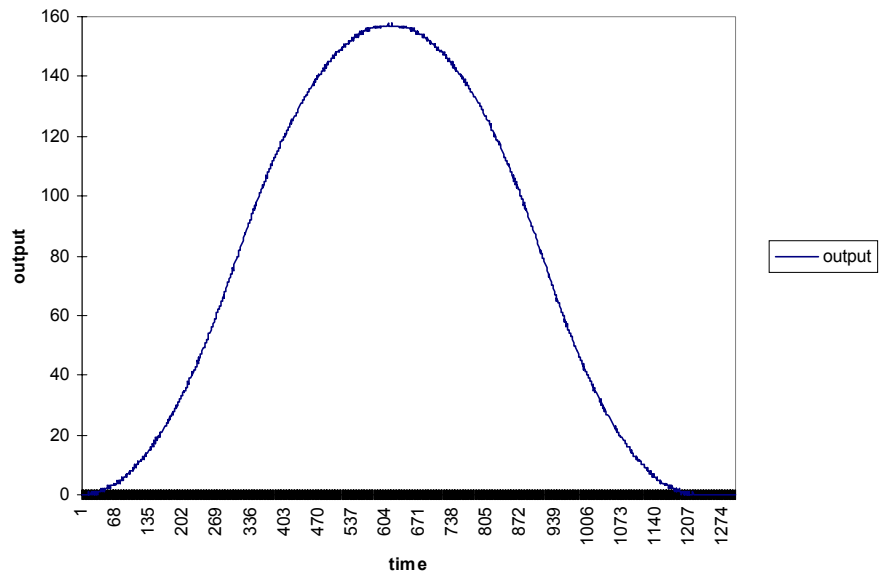
The following four examples illustrate the effects of jerk on a triangular profile. As can be seen, the lower the jerk percentage the smoother the profile.

- 1.** Triangular profile with no scurve (and 50% acceleration).
- 2.** Triangular profile with scurve and 5% jerk.
- 3.** Triangular profile with scurve and 50% jerk.
- 4.** Triangular profile with scurve and 95% jerk.

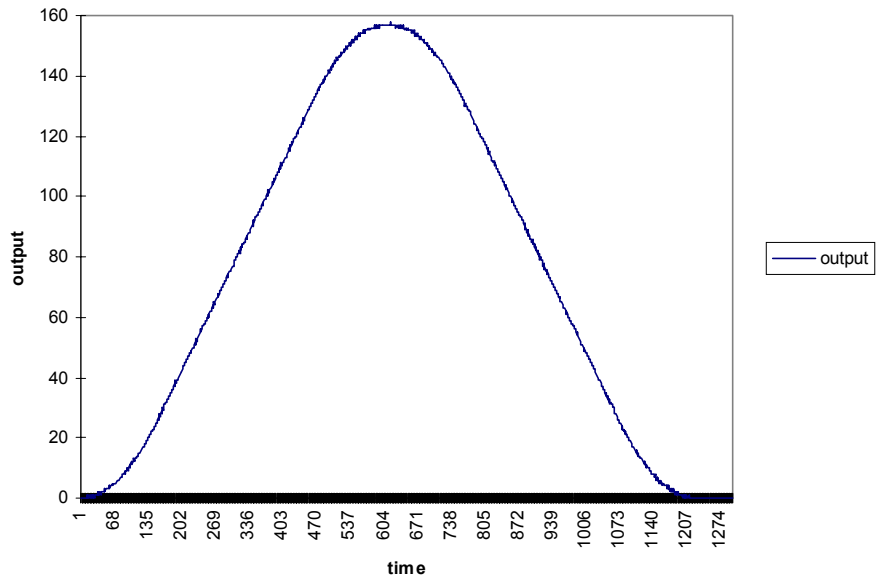
Figure 2-1. Triangular no scurve, 50% accel



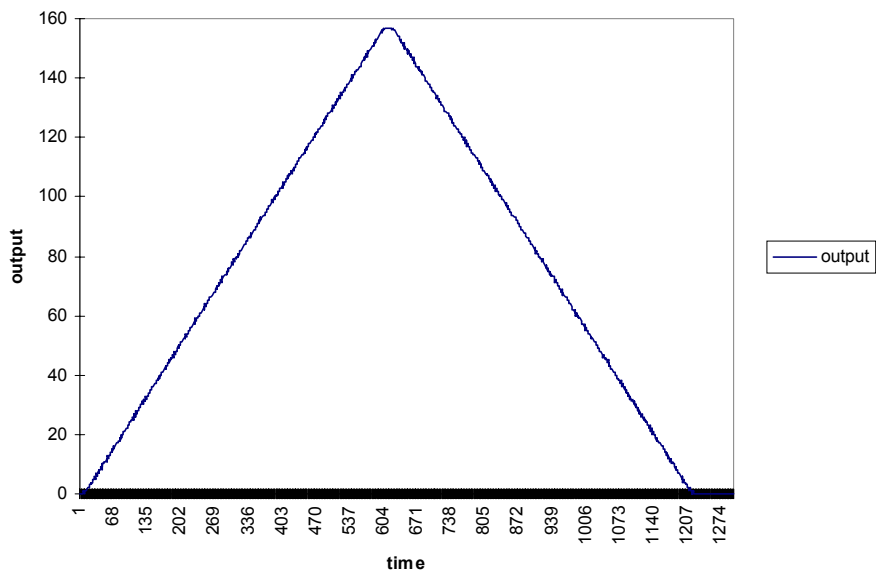
Triangular scurve, 50% accel, 5% jerk



Triangular scurve, 50% accel, 50% jerk



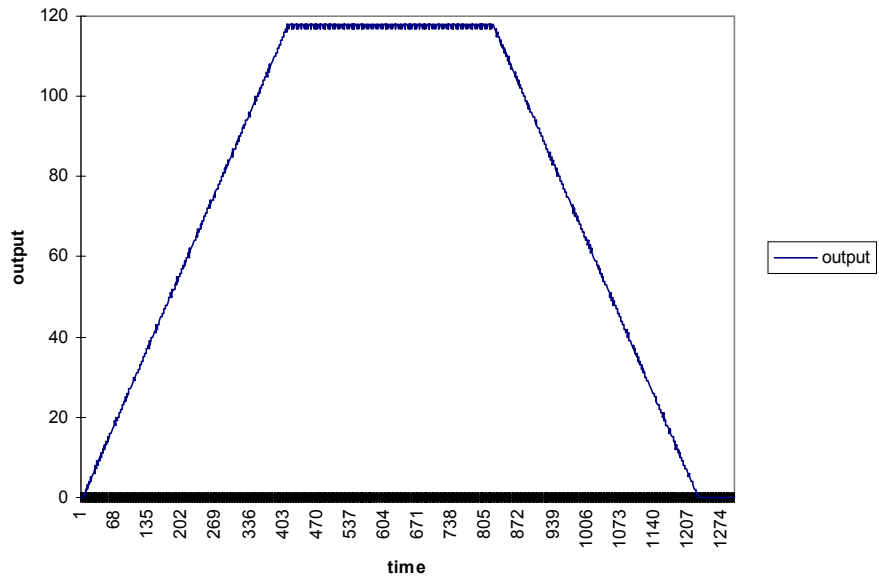
Triangular scurve, 50% accel, 95% jerk



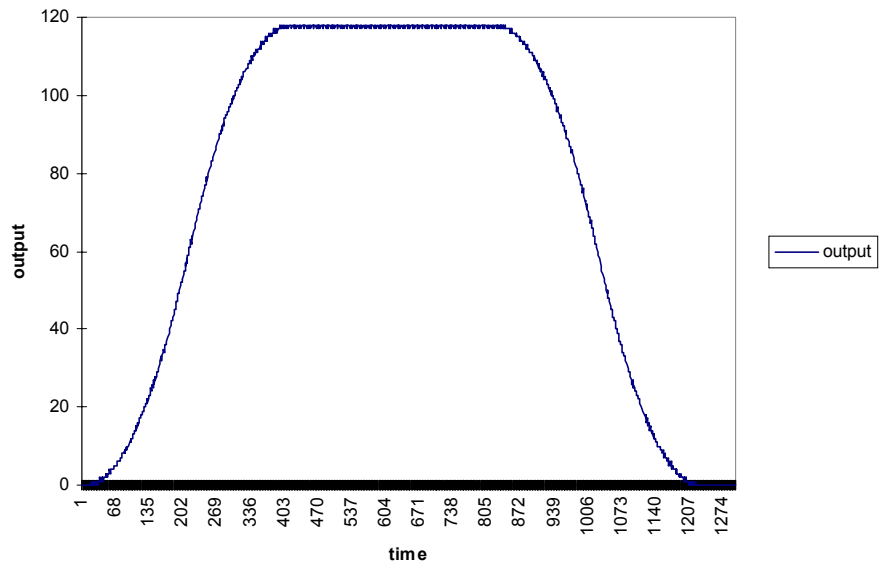
The following six examples illustrate the effects of acceleration and jerk on a trapezoidal profile. As can be seen, the higher the acceleration percentage the steeper the acceleration curve. Also, just as for the triangular profile, the lower the jerk percentage the smoother the profile.

1. Trapezoidal profile with no scurve and 50% acceleration.
2. Trapezoidal profile with scurve, 50% acceleration and 5% jerk.
3. Trapezoidal profile with scurve, 50% acceleration and 50% jerk.
4. Trapezoidal profile with scurve, 50% acceleration and 95% jerk.
5. Trapezoidal profile with scurve, 10% acceleration and 50% jerk.
6. Trapezoidal profile with scurve, 90% acceleration and 50% jerk.

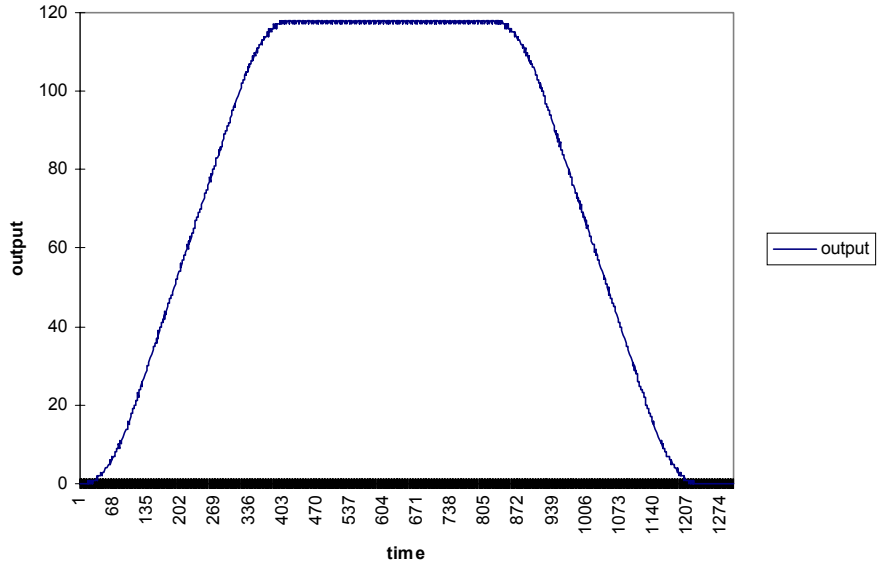
Trapezoidal no scurve, 50% accel



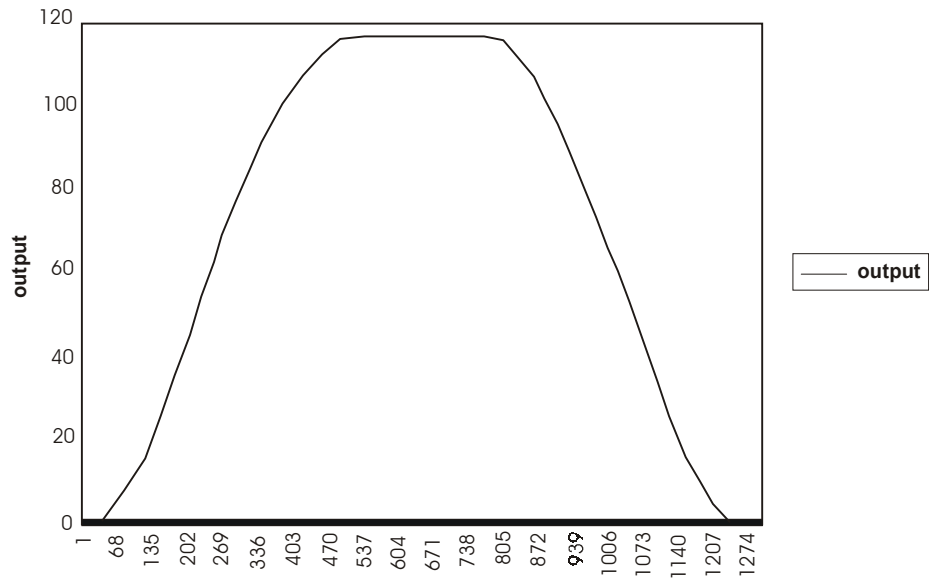
Trapezoidal scurve, 50% accel 5% jerk



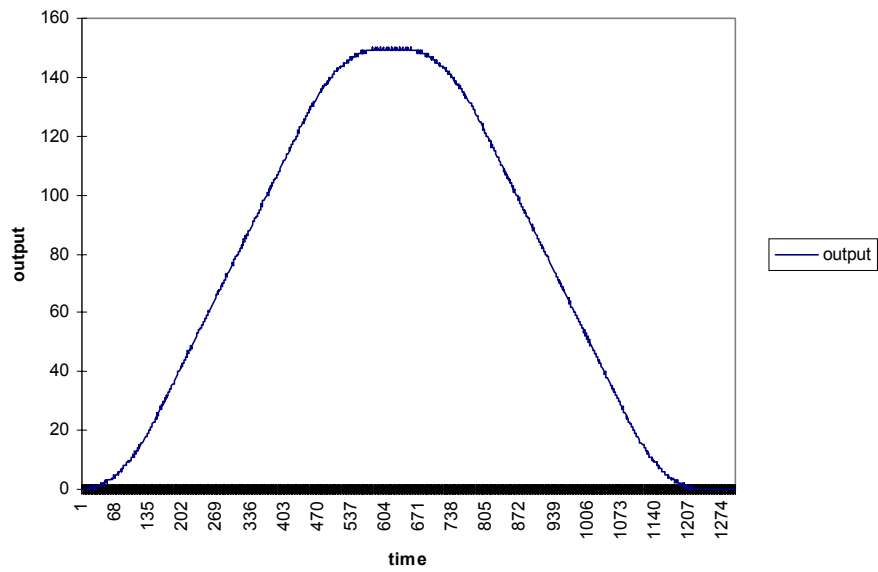
Trapezoidal scurve, 50% accel 50% jerk



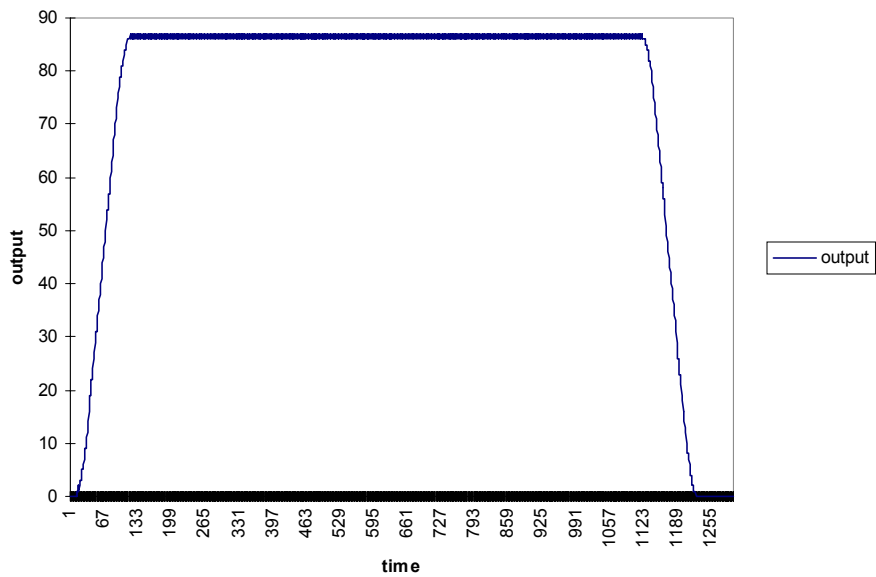
Trapezoidal scurve, 50% accel 95% jerk



Trapezoidal scurve, 10% accel 50% jerk



Trapezoidal scurve, 90% accel, 50% jerk



M_PRF1MV

One slave move for master

USER/M_PROFL

NAME	
M_PRF1MV	
EN00	OK
RR	ERR1
MOV1	TRP1
MDST	AMX1
MSCL	VMX1
SSCL	
VLIM	

Inputs: EN00 (BOOL) - enables execution (Typically one-shot)

RR (STRUCTURE) - Array of Structures to be used for profile.

MOV1 (STRUCTURE) - Structure containing 1st move's input data.

MDST (DINT) - Master feedback units/cycle.

MSCL (REAL) - Master feedback units/input unit.

SSCL (REAL) - Slave feedback units/input unit..

VLIM (REAL) - Maximum allowable velocity.

Outputs: OK (BOOL) - execution completed without error.

ERR1 (BYTE) - Error number for move.

TRP1 (BOOL) - Move changed to a trapezoid to achieve move.

AMX1 (REAL) - Maximum acceleration rate calculated for move.

VMX1 (REAL) - Maximum velocity rate calculated for move.

```
<<INSTANCE NAME>>:M_PRF1MV(EN00 := <<BOOL>>, RR := <<MEMORY AREA>> MOV1 := <<MEMORY AREA>>, MDST := <<DINT>>,
MCSL := <<REAL>>, SSCL := <<REAL>>, VLIM := <<REAL>>, OK =>
<<BOOL>>, ERR1 => <<BYTE>>, TRP1 => <<BOOL>>, AMX1 =>
<<REAL>>, VMX1 => <<REAL>>);
```

The M_PRF1MV function block sets up a single slave move in the master cycle.

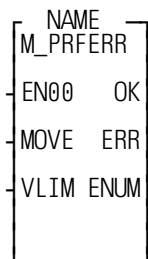
The M_PRF1MV function block does the same processing as M_PRF2MV except that it processes only one slave move for the master cycle rather than two slave moves. Refer to the M_PRF2MV description for more information.

Note: Fault bit number 6 (Table 2-3: Error Definitions) is not used by M_PRF1MV.

M_PRFERR

Check for profile errors

USER/M_PROFL



Inputs: EN00 (BOOL) - enables execution (Typically one-shot)
MOVE (STRUCT) - Structure containing 1st move's input data.
VLIM (REAL) - Maximum allowable velocity.

Outputs: OK (BOOL) - execution completed without error.
ERR (BOOL) - Error flag.
ENUM (BYTE) - Error number.

```
<<INSTANCE NAME>>:M_PRFERR(EN00 := <<BOOL>>, MOVE :=  
  <<MEMORY AREA>> VLIM := <<REAL>>, OK => <<BOOL>>, ERR =>  
  <<BOOL>>, ENUM => <<BYTE>>);
```

The M_PRFERR function block checks the validity of the move's parameters that have been passed into a function block.

This function block checks the validity of the move's parameters which have been passed into a function block. It is originally designed specifically for a rotary master and a linear slave axis. The format for the MOVE structure is shown in Table 2-4.

Table 2-4. MOVE Structure

Name	Data Type	Definition
MOVE	STRUCT	Structure of input data for move
.STRT_ANG	REAL	Angle of master axis at start of slave move
.STOP_ANG	REAL	Angle of master axis at end of slave move
.SLV_MOVE	REAL	Distance of slave move
.MAX_V	REAL	Maximum desired velocity of the slave axis
.PCT_J	REAL	Percent of maximum possible jerk to be used
.PCT_A	REAL	Percent of maximum possible accel to be used
.TRI_ONLY	BOOL	Triangular profile desired
.SCURVE	BOOL	Smoothed scurve acc/dec desired

If any of the values are invalid, then a fault is flagged (ERR). All faults found are coded into a byte which is passed to the calling ladder. Use BYT2BOOL to decode faults. Only the first 6 bits are set in this function block. The upper 2 bits are used and set in the calling ladder. The Error Definitions are shown in Table 2-5.

Table 2-5. Error Definitions

Fault Bit Number	Description
0	Starting angle is not within -180 to 360 degrees
1	Ending angle is not within -180 to 360 degrees
2	Acceleration percent value not within 0.0 to 100.0
3	Jerk percentage value not within 0.0 to 100.0
4	Desired velocity limit higher than allowed
5	Desired velocity limit is zero

M_PROFL

Make profile for 1 move

USER/M_PROFL

NAME	
M_PROFL	
EN00	OK
MDST	TRPZ
SDST	SEGS
RR	MRND
MAXV	SRND
MSCL	ERR
SSCL	AMAX
ACCP	VMAX
JERK	MAST
	SLAV

Inputs: EN00 (BOOL) - enables execution (Typically one-shot)
MDST (REAL) - Master distance for this move.
SDST (REAL) - Slave distance for this move.
RR (STRUCT) - Array of structures to be used for profile.
MAXV (REAL) - Maximum desired velocity.
MSCL (REAL) - Master feedback units/input unit.
SSCL (REAL) - Slave feedback units/input unit.
ACCP (REAL) - Percent of maximum possible accel to be used.
JERK (REAL) - Percent of maximum possible jerk to be used

Outputs: OK (BOOL) - execution completed without error.
TRPZ (BOOL) - Move changed to a trapezoid to achieve move.
SEGS (USINT) - Total number of structures used for this move.
MRND (DINT) - Master move rounding error detected in FUs.
SRND (DINT) - Slave move rounding error detected in FUs.
ERR (BOOL) - Cannot achieve the move with the given inputs.
AMAX (REAL) - Maximum acceleration rate calculated for move.
VMAX (REAL) - Maximum velocity rate calculated for move.
MAST (DINT) - Number of master feedback units used in move.
SLAV (DINT) - Number of Master feedback units used in move.

```

<<INSTANCE NAME>>:M_PROFL(EN00 := <<BOOL>>, MDST :=
  <<REAL>>, SDST := <<REAL>> RR := <<MEMORY AREA>>, MAXV :=
  <<REAL>>, MSCL := <<REAL>>, SSCL := <<REAL>>, ACCP :=
  <<REAL>>, JERK := <<REAL>>, OK => <<BOOL>>, TRPZ => <<BOOL>>,
  SEGS => <<USINT>>, MRND => <<DINT>>, SRND => <<SRND>>, ERR
  => <<BOOL>>, AMAX => <<REAL>>, VMAX => <<REAL>>, MAST =>
  <<DIST>>, SLAV => <<DINT>>);

```

The M_PROFL function block sets up one move

Maximum velocity can be limited to allow an automatically adjusting profile which will be triangular until the maximum velocity is reached. It will then spread into a trapezoid using the minimum acceleration to achieve the move.

It is also possible to set up each move as a constant accel/decel triangular move or a trapezoidal move using operator inputs for the desired shape. Acceleration can be adjusted to change the shape from triangular to nearly rectangular by increasing the acceleration percent.

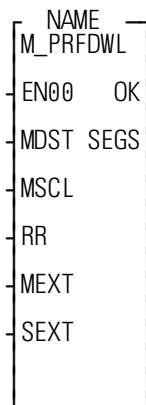
For any such profile, the acceleration can be “smoothed” by adjusting the jerk percent. This will not change the basic shape of the profile, but will change the acceleration and deceleration portions of the move to resemble an “S-Curve”. The percentage of jerk corresponds inversely to the portion of the Acceleration (or Deceleration) segment of the move which will be smoothed, until 100% equals no S-Curve. At minimum jerk, there is no constant accel portion. This will correspond to the highest acceleration rate. Maximum velocity during a move is not affected by “smoothing” or jerk, nor is the average acceleration. It only affects how the acceleration (and deceleration) will be applied to obtain this velocity.

The format for the RATIO_RL structure is shown in Table 2-1.

M_PRFDWL

Slave dwell in profile

USER/M_PROFL



Inputs: EN00 (BOOL) - enables execution (Typically one-shot)
MDST (REAL) - Master distance for dwell.
MSCL (REAL) - Master scale factor.
RR (STRUCTURE) - Array of structures used for the profile.
MEXT (DINT) - Extra master feedback units.
SEXT (DINT) - Extra slave feedback units.

Outputs: OK (BOOL) - execution completed without error
SEGS (USINT) - Total number of structures used for move.

```
<<INSTANCE NAME>>:M_PRFDWL(EN00 := <<BOOL>>, MDST :=  
  <<REAL>>, MSCL := <<REAL>> RR := <<MEMORY AREA>>, MEXT :=  
  <<DINT>>, SEXT := <<DINT>>, OK => <<BOOL>>, SEGS =>  
  <<USINT>>);
```

This M_PRFDWL function block takes the array of structures pointer and fills the next structure with the necessary data for a Slave dwell in the profile. The Master distance will be adjusted by any rounding errors detected when the preceding move was calculated. The only slave motion will be any slave rounding errors detected. These will be applied at the end of the dwell.

There is no acceleration or jerk or initial velocity in a dwell move.

M_SETVAJ

Set vel, acc, jerk values

USER/M_PROFL

NAME	
M_SETVAJ	
EN00	OK
VLMT	MAXV
MDST	MAXA
SDST	MAXJ
ACCP	MACC
JERK	TRPZ

Inputs: EN00 (BOOL) - enables execution (Typically one-shot)

VLMT (REAL) - Maximum desired velocity.

MDST (REAL) - Master distance for this move.

SDST (REAL) - Slave distance for this move.

ACCP (REAL) - Percent of maximum possible accel to be used.

JERK (REAL) - Percent of maximum possible jerk to be used.

Outputs: OK (BOOL) - execution completed without error.

MAXV (USINT) - Maximum velocity calculated for move.

MAXA (LREAL) - Maximum acceleration rate calculated for move.

MAXJ (LREAL) - Maximum jerk calculated for move.

MACC (LREAL) - Master distance during acceleration (and deceleration).

TRPZ (BOOL) - Move changed to a trapezoid to achieve move.

```
<<INSTANCE NAME>>:M_SETVAJ(EN00 := <<BOOL>>, VLMT :=
<<REAL>>, MDST := <<REAL>> SDST := <<REAL>>, ACCP :=
<<REAL>>, JERK := <<REAL>>, OK => <<BOOL>>, MAXV =>
<<USINT>>, MAXJ => <<LREAL>>, MACC => <<LREAL>>, TRPZ =>
<<BOOL>>);
```

The M_SETVAJ function block calculates the acceleration, maximum velocity, and jerk to be used for this move.

The M_SETVAJ ASFB calculates the acceleration, maximum velocity, and jerk to be used for this move. Separate calls can be used if acceleration and deceleration are to be different. The distance that the master will move during the acceleration (and deceleration) is also calculated. If Accel is not the same as decel, care must be used to avoid invalid setup of the Ratio Real. Use MAXV for K1 of Decel. It is easy to have s-curve on either the acceleration or deceleration rather than both. The value of this is to allow the axis to accel or decel faster when inertia is lower, and allow more time for critical moves (i.e., a larger portion of a triangular move for

the decel when concerned with a loaded part slipping out of the holding mechanism while the accel has no such concern).

MAXV is the maximum velocity that the profile will reach, and is the ending velocity for the accel portion of the move.

MACC is the distance that the Master axis will move during the accel.

For any such move profile, the acceleration can be "smoothed" by adjusting the jerk percent. This will change the acceleration (and deceleration) portions of the move to resemble an "S-Curve".

The percentage of jerk corresponds inversely to the portion of the Acceleration (or Deceleration) segment of the move which will be smoothed, until 100% equals no S-Curve. At minimum jerk, there is no constant accel portion. This will correspond to the highest acceleration rate. Maximum velocity during a move is not affected by "smoothing" or jerk, nor is the average acceleration. It only affects how the acceleration (and deceleration) will be applied to obtain this velocity.

M_SC_ACC

Acceleration segment

USER/M_PROFL

NAME	
M_SC_ACC	
EN00	OK
AMAX	SDST
VMAX	SEGS
JERK	SCUR
MDST	AERR
RR	VERR

- Inputs:** EN00 (BOOL) - enables execution (Typically one-shot)
- AMAX (LREAL) - Maximum acceleration rate calculated for the move.
- VMAX (LREAL) - Maximum velocity calculated for the move.
- JERK (LREAL) - Maximum jerk calculated for this move.
- MDST (LREAL) - Master distance during acceleration.
- RR (STRUCTURE) - Array of structures to be used for profile.
- Outputs:** OK (BOOL) - Execution of function completed without error.
- SDST (DINT) - Slave distance during acceleration.
- SEGS (USINT) - Total number of segments used for acceleration.
- SCUR (BOOL) - If set, acceleration uses an S curve move.
- AERR (BOOL) - If set, acceleration equals 0.
- VERR (BOOL) - If set, velocity equals 0..

```
<<INSTANCE NAME>>:M_SC_ACC(EN00 := <<BOOL>>, AMAX :=
<<LREAL>>, VMAX := <<LREAL>> JERK := <<LREAL>>, MDST :=
<<LREAL>>, RR := <<MEMORY AREA>>, OK => <<BOOL>>, SDST =>
<<DINT>>, SEGS => <<USINT>>, SCUR => <<BOOL>>, AERR =>
<<BOOL>>, VERR => <<BOOL>>);
```

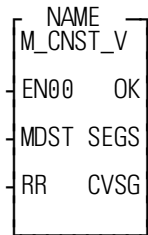
The M_SC_ACC function block adds the acceleration portion of the move to the profile.

The M_SC_ACC ASFB adds the necessary segments for the acceleration portion of the profile. If the JERK input is not equal to zero, there will be three segments for the acceleration. If the JERK input is zero, there will be one segment required.

M_CNST_V

Constant velocity segment

USER/M_PROFL



Inputs: EN00 (BOOL) - enables execution (Typically one-shot)
MDST (REAL) - Master distance for this move.
RR (STRUCT) - Array of Structures to be used for profile.

Outputs: OK (BOOL) - Execution of function completed without error.
SEGS (USINT) - Total number of segments used for profile.
CVSG (BOOL) - A valid Constant Velocity structure was used flag.

```
<<INSTANCE NAME>>:M_CNST_V(EN00 := <<BOOL>>, MDST :=  
  <<REAL>>, RR := <<MEMORY AREA>> OK => <<BOOL>>, SEGS =>  
  <<USINT>>, CVSG => <<BOOL>>);
```

The M_CNST_V function block fills a Ratio Real structure with the necessary distances and polynomial co-efficients for a Constant Velocity move. The initial velocity is filled by an earlier function call.

A constant velocity move requires the initial velocity, K1 to be non-zero, and the initial and final values of both acceleration, K2/2, and Jerk, K3/6, to be zero.

The Master Move Distance cannot be zero for a RATIO_RL segment, therefore if this would be the case, then no constant velocity move is set up.

M_SC_DEC

Deceleration segment

USER/M_PROFL

NAME	
M_SC_DEC	
EN00	OK
DMAX	SDST
VMAX	SEGS
JERK	SCUR
MDST	DERR
RR	VERR

- Inputs:** EN00 (BOOL) - enables execution (Typically one-shot)
- DMAX (LREAL) - Maximum acceleration rate calculated for the move.
- VMAX (LREAL) - Maximum velocity calculated for the move.
- JERK (LREAL) - Maximum jerk calculated for this move.
- MDST (LREAL) - Master distance during acceleration.
- RR (STRUCTURE) - Array of structures to be used for profile.
- Outputs:** OK (BOOL) - Execution of function completed without error.
- SDST (DINT) - Slave distance during acceleration.
- SEGS (USINT) - Total number of segments used for acceleration.
- SCUR (BOOL) - If set, acceleration uses an S curve move.
- DERR (BOOL) - If set, acceleration equals 0.
- VERR (BOOL) - If set, velocity equals 0.

```
<<INSTANCE NAME>>:M_SC_DEC(EN00 := <<BOOL>>, DMAX :=
<<LREAL>>, VMAX := <<LREAL>> JERK := <<LREAL>>, MDST :=
<<LREAL>>, RR := <<MEMORY AREA>>, OK => <<BOOL>>, SDST =>
<<DINT>>, SEGS => <<USINT>>, SCUR => <<BOOL>>, DERR =>
<<BOOL>>, VERR => <<BOOL>>);
```

The M_SC_DEC function block adds the deceleration portion of the move to the profile.

The M_SC_DEC ASFB adds the necessary segments for the deceleration portion of the profile. If the JERK input is not equal to zero, there will be three segments for the deceleration. If the JERK input is zero, there will be one segment required.

NOTES

APPENDIX C **Contents of the Applications CD**

The Applications CD contains the Motion ASFBs, General ASFBs, Ethernet ASFBs and Example Applications for the Standalone MMC and the MMC for PC.

The following manuals can be found in the folder Manuals on your CD: Motion ASFB Manual, General Purpose ASFB Manual and Ethernet ASFB Manual. They are PDF files and can be opened, read, and printed using Adobe Acrobat Reader. Acrobat Reader can be downloaded free of charge from Adobe at <http://www.adobe.com/supportservice/custsupport/download.html>.

Note: You MUST use Acrobat Reader 4.0 or newer to view these manuals.

The PiCPro Applications can be used with either PiCPro Professional Edition or MMC-Limited Edition. The **Examples** folder has example projects for each edition of PiCPro.

The default folders when installing the Applications CD are (where Vxx.x.x represents the version of the applications software) :

- C:\Program Files\G&L Motion Control\Applications Vxx.x.x\ASFB
- C:\Program Files\G&L Motion Control\Applications Vxx.x.x\Examples
- C:\Program Files\G&L Motion Control\Applications Vxx.x.x\Tools
- C:\Program Files\G&L Motion Control\Applications Vxx.x.x\QuickStart

Note: You should not put any other files or applications into these folders.

The **ASFB** folder contains the Motion, General and Ethernet ASFBs.

The **Tools** folder contains the ServoSetupAssistant spreadsheet.

The **Examples** folder contains 8 Standalone MMC examples or starter ladders and several simple examples for the Motion, General and Ethernet ASFBs. There are also 3 MMC for PC examples or starter ladders included, although these cannot be downloaded using the MMC-Limited Edition.

The following is a description of how the examples are named.

- MMCxxxxx Standalone MMC Application Example.
- MPCxxxxx MMC for PC Application Example.
- SRCxxxxx SERCOS Setup Example.
- SRVxxxxx Standalone MMC Servo Setup Example.
- SRVPxxxxx MMC for PC Servo Setup Example.
- ZSRMMC SERCOS Setup Example Library.
- ZSRVMMC Servo Setup Example Library.
- xxxxxx.CPA Cimrex Operator Interface Application Example.
- xxxxxx.GLC G&L DeviceNet Scanner Configuration Example.
- M_xxxxxx Motion ASFB Example.
- G_xxxxxx General ASFB Example.
- E_xxxxxx Ethernet ASFB Example.

- U_XXXXXX MMC Application Example UDFB.

The following are brief descriptions of each Standalone MMC example (see Appendix D for more information).

- M_C2M_EX. The Cad2Motion ASFB, translates an M and G code format ASCII file into servo motion. Many applications require description of their motion path using CAD software. Third party packages (such as Gcode2000) will convert the CAD package DXF output to M and G code text files. M_C2M will translate the M and G code file to servo motion. Examples of applications include glue laying and textiles cutting. The Cad2Motion ASFB is not intended for applications that involve metal cutting machine tools such as lathes and mills and therefore does not support features required by CNC applications such as cutter radius compensation, tool offsets and constant surface speed.
- MMC4_EX. This example can be used as a starting point for a 4 axis application. It contains the logic required to initialize the servo axes, close the servo loops, jog an axis, and home an axis. It also contains basic E-Stop and C-Stop logic. This example does not have any operator interface support. All of the machine functions are performed with discrete I/O.
- MMC2_EX. This example can be used as a starting point for a 2 axis application. It contains the logic required to initialize the servo axes, close the servo loops, jog an axis, and home an axis. It also contains basic E-Stop and C-Stop logic. This example does not have any operator interface support. All of the machine functions are performed with discrete I/O.
- MMC4_OI. This example can be used as a starting point for a 4 axis application. It contains the logic required to initialize the servo axes, close the servo loops, jog an axis, and home an axis. It also contains basic E-Stop and C-Stop logic. This example uses a Cimrex 30 as an operator interface. The Cimrex application is called MMC4_C30.
- MMC2_OI. This example can be used as a starting point for a 2 axis application. It contains the logic required to initialize the servo axes, close the servo loops, jog an axis, and home an axis. It also contains basic E-Stop and C-Stop logic. This example uses a Cimrex 30 or Cimrex 20 as an operator interface. The Cimrex applications are called MMC2_C30 and MMC2_C20 (for the Cimrex 30 or Cimrex 20 units respectively).

- **MMC4_SOI.** This example can be used as a starting point for a SERCOS application having an operator interface device. It contains the logic required to initialize the SERCOS configuration, initialize the servo axes, close the servo loops, jog an axis, and home an axis. It also contains basic E-Stop and C-Stop logic. This example uses a Cimrex 30 as an operator interface. The Cimrex application is called MMC4_SC30.
- **MMC4_PT.** This example shows how to apply the press transfer application specific function blocks. It contains the logic required to initialize the servo axes, close the servo loops, jog an axis, and home an axis. It also contains basic E-Stop and C-Stop logic. It also contains logic to configure and run the specified slave profile(s). This example uses a Cimrex 30 as an operator interface. The Cimrex application is called M4PT_C30.
- **MMC_STEP.** This example shows how to apply the stepper block I/O module. It contains the logic required to initialize the stepper block and to control it. A single UDFB (U_STPRC) provides the control logic for a stepper axis. The features supported by the UDFB include home (zero reference), jog, move a distance, move to a position, control-stop, emergency-stop and c-stop/e-stop reset.

MMC_DND. This example shows how to apply the Centurion DeviceNet Positioning MicroDSM drive. It contains the logic required to initialize and run the DeviceNet network. Two UDFBs provides the control logic for a DeviceNet axis to jog, move a distance or move to a position. A third UDFB provides the DeviceNet scanner status details. This example ladder and DeviceNet configuration are both referenced in the MicroDSM with DeviceNet installation manual.

The following is a brief description of each MMC for PC example (or starter ladder).

- **MPC4_EX.** This example can be used as a starting point for a 4 axis application. It contains the logic required to initialize the servo axes, close the servo loops, jog an axis, and home an axis. It also contains basic E-Stop and C-Stop logic. This example does not have any operator interface support. All of the machine functions are performed with discrete I/O. This example is almost identical to MMC4_EX.
- **MPC4_OI.** This example can be used as a starting point for a 4 axis application. It contains the logic required to initialize the servo axes, close the servo loops, jog an axis, and home an axis. It also contains basic E-Stop and C-Stop logic. This example uses a Cimrex C30 with an Ethernet module as an operator interface. The Cimrex application is called MPC4_C30. This example is almost identical to MMC4_OI.

- MPC4_SOI. This example can be used as a starting point for a SERCOS application having an operator interface device. It contains the logic required to initialize the SERCOS configuration, initialize the servo axes, close the servo loops, jog an axis, and home an axis. It also contains basic E-Stop and C-Stop logic. This example uses a Cimrex C30 with an Ethernet module as an operator interface. The Cimrex application is called MPC4_SC30. This example is almost identical to MMC4_SOI.

Note:For both of the MMC for PC with Cimrex examples, there will usually be a configuration change required before either works. The Cimrex configuration file calls out the Ethernet connection to the MMC for PC. However, the correct TCP/IP address of the PC hosting the MMC for PC is unknown. The actual host address must be verified in the Cimrex configuration before it is downloaded to the Cimrex unit.

APPENDIX D **Standalone MMC Examples**

Descriptions of the Standalone MMC Examples

The following is a description of the functionality provided with each Standalone MMC example.

The purpose of each example is to provide a foundation for a customer's own application. Each example provides all the basic features of most applications:

- Loading the servo configuration
- Closing the servo position loops
- Handling emergency stop and control stop conditions
- Jogging the axes
- Referencing (or homing) the axes

Typically, the ladder logic required for a specific application can be added to the end of an example.

Which example is the best starting point depends on the specific application (e.g. how many servo axes are required and will an operator interface device be used?).

Basic Application Examples

The following is a description of the two examples that lack an operator interface. They are MMC2_EX for a 2 axis MMC and MMC4_EX for a 4 axis MMC.

- The initial network with the STRTSERV function loads the servo setup configuration. Each example ladder includes a servo setup example (such as SRV2_EX for a 2 axis MMC or SRV4_EX for a 4 axis MMC). Each application must have its own servo setup configuration. Typically, this configuration is unique for every application as each could have different ratios of programming units (i.e., inches, millimeters, degrees, etc.) to the servo feedback units, different travel limits, different servo PID gains, etc.
- The M_CHK1 and M_CHK49 function blocks check which axes are configured within the loaded servo setup. These blocks make it easier to write sections of ladder code that support different numbers of axes; by checking the Booleans set here, the appropriate actions can be taken based on the axes that are configured.
- The M_CLOS1 function blocks will close the position loops for the configured servo axes. This action also includes resetting any internal e-stop or c-stop servo conditions (a servo e-stop will open the position loop).
- The M_STATUS and M_ERROR function blocks provide all the significant data for the servos in one place within the ladder. If the application is using the digitizing axis available with the MMC, there is a network that checks that axis for an e-stop condition (that can result from a 'loss of feedback' hardware error).

- The STATUSSV functions provide the ladder with information regarding the state of the respective servo fast inputs. Each axis has a DC input provided with a 'fast input' capability - so that (when enabled) the axis can latch the axis position at the instant the DC input occurs, based solely on the input condition rather than on the ladder scan time.
- The network with the ESTOPACT coil monitors for an emergency stop (or e-stop) condition. The purpose for this network is to ensure that if an e-stop condition is detected, all the servo axes will be e-stopped immediately.
- The network with the CSTOPACT coil monitors for a controlled stop (or c-stop) condition. The purpose for this network is to ensure that if a c-stop condition is detected, all the servo axes will be c-stopped immediately.
- The networks with the GEN_IO and AUX_IO labels provide a simple way to view all the MMC discrete input and output conditions using the ladder animation.
- The subsequent networks provide support for jogging and homing the axes. When jogging, only one axis can be selected and it can jog in one of three modes: at a constant velocity, to a specific incremental distance, or as a follower to a handwheel. Note that jogging will stop when a travel limit is reached because an e-stop will occur (in the ESTOPACT's coil network). However, if the Machine Start push-button is being pushed at the same time as the jog button, the operator can jog off a specific over-travel. For instance when in velocity mode, if the axis is on the plus over-travel switch, the axis cannot jog in the plus direction but it can jog in the minus direction.

The last networks in the basic examples provide support for the referencing (or homing) of the servo axes. The M_FHOME function block supports a homing sequence for the axis using the axis fast input as the reference switch. The M_LHOME function block supports a homing sequence for the axis using a specific discrete input as the reference switch. Both function blocks are provided for each servo axis in the ladder with M_FHOME actually connected to the rail. If the M_LHOME functionality is desired, then its code fragment (the function block and the surrounding elements) can be cut and pasted over the M_FHOME area.

Operator Interface Application Examples

The examples having operator interface support (MMC2_OI and MMC4_OI) have more ladder logic than the basic examples. Early in the ladder, the OI_SER function block enables the MMC variables (in the main ladder) that are given the G (global) attribute to be shared with the operator interface device (typically a Cimrex terminal) connected via an RS232 serial connection. Note that the global attribute is also used when the MMC is connected to an external PC via the Ethernet module and the MMC shares its data with the PC using the OPC Server technology.

There are a number of networks that update the state of the ALARM() Boolean array. This array is used to provide immediate indicators to the operator interface device regarding specific ladder conditions.

After the axis jogging and homing section of the ladder, there are several networks involved with the saving and retrieval of the axis setup data using a RAMDISK file. Rather than having the various motion rates or reference positions hard coded within the ladder logic, they can be set by the operator and saved in a RAMDISK file, AXSETUP.DAT.

The examples conclude with a few networks that are the basis for supporting other Cimrex features: trending or graphing and the control of the LEDs.

SERCOS Drive Interface Application Example

There is an example provided for the 8 axis SERCOS MMC unit. This example (MMC4_SOI) is an extension of the 4 axis MMC example with an operator interface. For the support of the SERCOS drives, several different ASFBs are used.

At the start of the ladder logic, the SC_INIT function initializes the SERCOS ring connected to the MMC. Each SERCOS application must have its own SERCOS setup configuration. The example uses SRC_MMC4 for its 4 SERCOS axis configuration. For an actual application, this SERCOS setup function must be replaced with the actual SERCOS setup configuration. The programmer's steps to do that are very similar to those for replacing the servo setup, except that steps 7 through 10 in the above procedure (when STARTING A NEW APPLICATION FROM AN EXAMPLE) are done with SERCOS setup, not servo setup. The example uses SRV_MMC4 for its servo setup configuration. For an actual application, this servo setup function must be replaced with the actual servo setup configuration, as previously described.

The S_CLOS1 function block will close the position loop for the configured SERCOS axes. This ASFB is different for SERCOS axes because the SERCOS drive itself might have error conditions to reset before the position loop can be closed.

The S_ERRORC function block provides all the significant error information for a specific SERCOS axis. This ASFB is different than M_ERROR because a SERCOS axis has additional error information.

The S_FHOME and S_LHOME function blocks provide support for the referencing of the SERCOS axes. They are different than M_FHOME and M_LHOME because the latching of the reference position is done within the SERCOS drive.

The S_IO_C function block is unique to a Centurion SERCOS drive application because the additional discrete input and output points at the SERCOS drive can be made available to the ladder.

Press Transfer Application Example

There is an example provided for the press transfer ASFB set within the motion ASFB package. This example (MMC4_PT) is an extension of the 4 axis MMC example with an operator interface. There is a detailed 'readme' file for this example's user interface and the accompanying ladder logic; that file is MMC4_PT.TXT.

Stepper Application Example

There is an example provided for the stepper block I/O module. This example (MMC_STEP) is a simple ladder to illustrate the control logic for a stepper module.

The UDFB U_STPRC handles the overhead required for a stepper control. This UDFB handles the stepper home or zero reference, jog, move a distance, move to a position, controlled-stop, emergency stop, and a reset of a c-stop or e-stop.

Centurion DeviceNet Positioning MicroDSM Drive Application Example

There is an example provided for the Centurion DeviceNet MicroDSM drive. This example (MMC_DND) is a simple example to illustrate the logic to control the DeviceNet scanner and to control the DeviceNet drive over DeviceNet.

The ASFB M_DNJOGC handles the overhead required for the jog motion. The ASFB M_DNPOSC handles the overhead required for the motion of either a move of an incremental distance or a move to an absolute position. Note that both of these ASFBs are for the Centurion DeviceNet MicroDSM drive. Other DeviceNet drives will have their own unique DeviceNet interface (i.e., the respective boolean, byte and long-word tag name assignments in the scanner's memory map).

The ASFB M_DNSTAT extracts the details for the DeviceNet module status into individual bytes and booleans.

This example ladder is referenced in the MicroDSM with DeviceNet installation manual. It serves as the programming example for the MicroDSM with DeviceNet drive. The MMC_DND.GLC DeviceNet scanner's configuration data file is also referenced in the same drive manual to illustrate how to define the tag names for the DeviceNet drive's input and output data.

Starting a New MMC Application from a Standalone MMC Example

The following is a description of how to start a new application from a Standalone MMC example.

To use one of the examples as a starting point for your application follow the steps below. The following procedure will use the MMC4_EX Professional Edition Project as the starting point. The target application will be called NEW_APPL and the servo setup library will be ZNEWAPPL; other appropriate names can replace these used in this procedure. The same steps can be followed with the PiCPro MMC Limited Edition.

Note:In the instructions below, xx.x and xx.x.x represent the current version of the software.

1. Start PiCPro Vxx.x Professional Edition.
2. File | Open, MMC4_EX Pro Edition.PRJ. This will open up the project.

3. Once the project is open, do a File | Save As and fill in the following fields:
 - Destination PRJ
C:\Program Files\G&L Motion Control\Applications
Vxx.x.x\New_Appl\New_Appl.prj
This will be the name and folder for the new project.
 - Destination G&L
C:\Program Files\G&L Motion Control\Applications
Vxx.x.x\New_Appl\New_Appl.G&L.
This is the file name of the compressed project. The compressed file can be stored on the MMC Flashdisk. This allows the end user the ability to open the project from the MMC without having the original source code.
 4. Change the destination folders for Main LDO, Lib Path, UDFB Source, Servo Source, and Servo Lib's from:
 - C:\Program Files\G&L Motion Control\Applications Vxx.x.x\Examples
 - To:
C:\Program Files\G&L Motion Control\Applications Vxx.x.x\New_Appl
 5. Close the Save As Project by clicking OK.
 6. Expand the Main .LDO and double click on MMC4_EX.LDO. This will open the main ladder and do a File | Save As with the new file name NEW_APPL.ldo.
 7. In network 5 (the network with the STRTSERV function), right click on SRV4_EX and then View Servo Setup. Do a File | Save As to SRV4_NEW.
 8. Make any necessary changes to the servo setup and then do a Compile | Make Function. When prompted for the library name for the servo function, change the name to ZNEWAPPL.
 9. Close servo setup.
 10. Replace the SRV4_EX function with the new servo setup function you just created by clicking on Ladder | Functions | ZNEWAPPL | SRV4_NEW.
 11. Close the NEW_APPL ladder window and save the changes when prompted.
 12. Go to back to the project window and delete MMC4_EX from the MAIN .LDO and add NEW_APPL.ldo.
 13. Do a File | Update Project Tree to update the project.
 14. Close the project. When prompted to compress the application select No. You do not need to compress the project until you are ready to store it on the flash disk.
 15. Using Windows Explorer delete the MMC4_*. * files, the SRV4*. * files and ZSRVMMC.LIB file from the new project folder:
C:\Program Files\G&L Motion Control\Applications Vxx.x.x\New_Appl
- Note:** If the application is using SERCOS-controlled servos, the above procedure is followed starting with MMC4_SOI with additional steps to replace the example SERCOS setup file.

NOTES

APPENDIX E **Servo Setup Assistant**

The ServoSetupAssistant is an Excel spreadsheet tool that makes creating your servo setup information a little bit easier. You specify machine parameters such as time base and precision in the ServoSetupAssistant spreadsheet. Based on these parameters the servo setup information for an axis is automatically calculated and can be imported into PiCPro using the Import button that is provided under the 'Axis Data' tab. The Servo Setup Assistant spreadsheet (ServoSetup Assistant.xls) is part of the Applications Software CD installation. It can be found in the Tools folder (C:\Program Files\G&L Motion Control\Applications Vxx.x.x\Tools, where xx.x.x represents the current version of the software). To use this spreadsheet:

1. Do a File Save As to save this spreadsheet as a different name to maintain the original. A different .XLS file should be made for each axis in your application.
2. Two sheets are provided for each type of axis that is allowed: Closed loop servo, Digitizing, and Closed loop SERCOS. The first sheet is used to input data, the second sheet holds the servo setup data that will be imported into PiCPro. You may delete the sheets for axes that are not being used. Double click on the tab for a sheet to change the name of the sheet to match the description of the axis.
3. Enter the data for the axis. The setup information is being automatically calculated on the next sheet as you enter. Enter only information with pull down boxes or data that appears in blue. When you are ready to save the information for an axis select the sheet called 'PiCPro 'X' Setup Data' (where X is Servo, Digitizing or SERCOS Servo). NOTE: Before you save the file in .CSV format, you must save the entire workbook by doing a File Save to keep your edits. If you do not do this, all formulas and changes will be lost when you save the file in .CSV format.
4. From the File menu select Save As and save the sheet under a different name, with file type .CSV (comma delimited variable). Excel will prompt you through the save process. All other sheets in the workbook and all formulas will be deleted in the .CSV file.
5. Close the files in Excel.
6. Start PiCPro. Import the data from the .CSV file by selecting the Import button under axis data.

NOTES

Index

A

Acceleration Scale Text A-6
Acceleration Scale Value A-6
ADDCKSUM 2-6
Alternative Mode Status A-31
Analog Input Acceleration Limit A-28
Analog Input Acceleration Limits Enable A-30
Analog Input Deceleration Limit A-28
Analog Offset A-24
Analog Output A-43
Analog Output Configuration Register A-23
Analog Scale A-25
Applications CD
 contents C-1
Arm Triggering A-49
ASFB 1-1
 using 1-3
At Speed Limit A-11
Autotune Maximum Current A-31
Autotune Maximum Distance A-32
Average Current A-46
Average Motor Current A-47
Average Time Constant A-16

B

Back EMF Constant A-15
Boot Firmware Version A-4
BRAKE Active Delay A-21
BRAKE Inactive Delay A-21
Bus Voltage A-46
BYTE2HEX 2-6

C

Change Direction Flag A-30
Channel 1 Source A-48
Channel 2 Source A-48
CHKCKSUM 2-7
Collected Data A-51
COMMAND Input A-43
Command Source A-27
COMMAND Torque Offset A-22
COMMAND Torque Scale A-22
COMMAND Velocity Offset A-21

COMMAND Velocity Scale A-22
Commands, Common Product Line A-2
Commutation Type A-16
Continuous Current Limit A-13
Current Command A-46
Current Feedforward A-16
Current Negative Peak A-46
Current Positive Peak A-46

D

Digital Input Configuration Register A-19
Digital Input States A-42
Digital Output Configuration Register A-20
Digital Output States A-42
Digital Output Write Mask A-20
Drive Mode A-27
Drive Name A-5
DWOR2HEX 2-7

E

Encoder Alignment Offset A-33
Encoder Lines A-14
Encoder Output Configuration Register A-27
exception responses A-1

F

Fault Status A-39
Field Current A-46
Field Voltage Command A-47
Firmware, Main Version A-4

G

Gear Ratio A-8

H

Hall Offset A-16
HEX2BYTE 2-8
HEX2DWORD 2-9
HEX2WORD 2-9

I

Index Offset A-16
Installation 1-1
Integrator Zone A-7

L

Low Pass Filter Bandwidth A-12

Low Pass Filter Enable A-12

M

M_C2M_1 2-10

M_CHK1 2-37

M_CHK101 2-38

M_CHK109 2-39

M_CHK49 2-40

M_CHK57 2-41

M_CHK65 2-42

M_CHK73 2-43

M_CHK9 2-44

M_CLOS1 2-45

M_CLOS9 2-47

M_CLS101 2-49

M_CLS109 2-51

M_CNST_V B-26

M_CRSFIN 2-53

M_DATCAP 2-55

M_DATCPT 2-59

M_DNJOGC 2-63

M_DNPOSC 2-65

M_DNSTAT 2-67

M_DSMCOM commands A-1

 alternative operating mode A-31

 analog I/O A-21

 common product line A-2

 digital I/O A-19

 exception response A-1

 general A-5

 host command set A-2

 motor A-14

 operating mode A-27

 position loop A-7

 runtime command and control A-36

 runtime data A-43

 runtime data collection A-48

 runtime status A-38

 serial port A-26

 torque current conditioning A-12

 velocity loop A-10

M_DW2BOO 2-74

M_ERROR 2-76

M_FHOME 2-77

M_INCPTR 2-79

M_JOG 2-81

M_LHOME 2-82

M_LINCIR 2-85

M_PRF1MV B-17

M_PRF2MV B-6

M_PRFDWL B-22

M_PRFERR B-18

M_PROFL B-20

M_PRTCAM 2-89

M_PRTREL 2-91

M_PRTSLP 2-93

M_RATREL 2-95

M_RATSLP 2-96

M_RDTUNE 2-98

M_RGSTAT 2-99

M_RSET49 2-101

M_RSET57 2-102

M_RSET65 2-103

M_RSET73 2-104

M_SACC 2-105

M_SC_ACC B-25

M_SC_DEC B-27

M_SCRVLC 2-107

M_SETVAJ B-23

M_SRCMON 2-113

M_SRCPRC 2-115

M_SRCRDL 2-117

M_SRCWT 2-119

M_SRCWTL 2-121

M_STATUS 2-127

M_WTTUNE 2-129

M_XL2CM 2-131

Manual Tune Position Period A-32

Manual Tune Position Step A-32

Manual Tune Velocity Period A-32

Manual Tune Velocity Step A-32

Master Encoder Resolution A-34

Master Index Position A-35

Master Position A-44

Maximum Motor Speed A-14

MMC application

 new from standalone MMC example D-4

Motor Continuous Current A-15

Motor Encoder Resolution A-34

Motor Forward Direction Flag A-18

Motor ID A-14

Motor Index Position A-34

Motor Peak Current A-14
 Motor Position A-43
 Motor Table Information A-17
 Motor Table Record Size A-17
 Motor Table Version A-17
 Motor Velocity A-45
N
 Negative Current Limit Input A-43
O
 Operating Mode A-31
 Over Speed Limit A-10
 Override Analog Outputs A-25
 Override Digital Output A-20
P
 Packed Drive Status A-38
 Pole Count A-16
 Position Command A-44
 Position Error A-44
 Position Error Limit A-8
 Position Error Time A-8
 Position Loop Derivative Gain A-7
 Position Loop Feedforward Gain A-7
 Position Loop Integral Gain A-7
 Position Loop Proportional Gain A-7
 Position Negative Peak Error A-44
 Position Positive Peak Error A-44
 Position Scale Text A-5
 Position Scale Value A-5
 Position Window Size A-7
 Position Window Time A-7
 Positive Current Limit Input A-43
 Powerup Status A-3
 Preset Acceleration Limits Enable A-30
 Preset Input Acceleration Limit A-28
 Preset Input Deceleration Limit A-29
 press transfer asfbs B-1
 Product Type A-2
R
 Reset Drive A-36
 Reset Faults A-37
 Reset Peaks A-43
 Reset Personality NVRAM A-5
 revision

history 1-2
 range 1-2
 Rotor Inertia A-15
 R-Phase Current A-47
 Run State A-41
S
 S_CLOS9 2-137
 S_CLS101 2-140
 S_CLS109 2-143
 S_ERRORC 2-146
 S_FHOME 2-149
 S_IO_C 2-152
 S_LHOME 2-154
 Save Alignment Offset A-34
 Serial Port Baud Rate A-26
 Serial Port Frame Format A-26
 servo setup assistant spreadsheet E-1
 Setpoint Acceleration A-37
 Setpoint Control A-37
 Slew Enable A-9
 Slew Rate A-8
 Software Drive Enable/Disable A-36
 Software Drive ID A-26
 Software Negative Current Limit A-12
 Software Positive Current Limit A-12
 Speed Window Size A-10
 standalone MMC examples D-1
 basic application D-1
 Centurion DeviceNet positioning Mi-
 coDSM drive application D-4
 descriptions D-1
 operator interface application D-2
 press transfer application D-3
 SERCOS drive interface application D-3
 stepper application D-4
T
 Thermal Time Constant Enable A-17
 Thermostat Flag A-15
 Timebase A-49
 Torque Constant A-15
 Torque Current A-47
 Torque Preset A-28
 Torque Scale Text A-6
 Torque Scale Value A-6

Torque Setpoint A-36
Torque Voltage Command A-47
T-Phase Current A-47
Trigger Mode A-49
Trigger Source A-49
Trigger Status A-50
Trigger Threshold A-49
Tuning Direction Flag A-29

V

Velocity Command A-45
Velocity Error A-46
Velocity Loop Derivative Gain A-10
Velocity Loop Integral Gain A-10

Velocity Loop Proportional Gain A-10
Velocity Loop Update Period A-11
Velocity Preset A-27
Velocity Scale Text A-6
Velocity Scale Value A-5
Velocity Setpoint A-36

W

Winding Inductance A-15
Winding Resistance A-15
WORD2HEX 2-157

Z

Zero Speed Limit A-10