Allen-Bradley

Logix5000 Controllers Motion Instructions

Catalog Numbers 1756-L1M1, 1756-L1M2, 1756-L1M3, 1756-L55M12, 1756-L55M13, 1756-L55M14, 1756-L55M16, 1756-L55M22, 1756-L55M23, 1756-L55M24, 1756-L60M03SE, 1756-L61, 1756-L62, 1756-L63, 1756-L64, 1768-L43, 1789-L60, 1789-20D

Reference Manual

Rockwell Automation

Important User Information

Solid state equipment has operational characteristics differing from those of electromechanical equipment. Safety Guidelines for the Application, Installation and Maintenance of Solid State Controls (publication SGI-1.1 available from your local Rockwell Automation sales office or online at http://literature.rockwellautomation.com) describes some important differences between solid state equipment and hard-wired electromechanical devices. Because of this difference, and also because of the wide variety of uses for solid state equipment, all persons responsible for applying this equipment must satisfy themselves that each intended application of this equipment is acceptable.

In no event will Rockwell Automation, Inc. be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular installation, Rockwell Automation, Inc. cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by Rockwell Automation, Inc. with respect to use of information, circuits, equipment, or software described in this manual.

Reproduction of the contents of this manual, in whole or in part, without written permission of Rockwell Automation, Inc., is prohibited.

Throughout this manual, when necessary, we use notes to make you aware of safety considerations.

WARNING	Identifies information about practices or circumstances that can cause an explosion in a hazardous environment, which may lead to personal injury or death, property damage, or economic loss.
IMPORTANT	Identifies information that is critical for successful application and understanding of the product.
ATTENTION	Identifies information about practices or circumstances that can lead to personal injury or death, property damage, or economic loss. Attentions help you identify a hazard, avoid a hazard, and recognize the consequence
SHOCK HAZARD	Labels may be on or inside the equipment, for example, a drive or motor, to alert people that dangerous voltage may be present.
BURN HAZARD	Labels may be on or inside the equipment, for example, a drive or motor, to alert people that surfaces may reach dangerous temperatures.

Where to Find an Instruction

Use this locator to find the reference details about Logix instructions (the grayed-out instructions are available in other manuals). This locator also lists which programming languages are available for the instructions.

If the locator lists	The instruction is documented in
a page number	this manual
general	Logix5000 Controllers General Instructions Reference Manual, 1756-RM003
process	Logix5000 Controllers Process Control and Drives Instructions Reference Manual, 1756-RM006
phase	Logix5000 Controllers PhaseManager User Manual, LOGIX-UM001

Instruction	Location	Languages
ABL ASCII Test For Buffer Line	general	relay ladder structured text
ABS Absolute Value	general	relay ladder structured text function block
ACB ASCII Chars in Buffer	general	relay ladder structured text
ACL ASCII Clear Buffer	general	relay ladder structured text
ACOS Arc Cosine	general	structured text
ACS Arc Cosine	general	relay ladder function block
ADD Add	general	relay ladder structured text function block
AFI Always False Instruction	general	relay ladder
AHL ASCII Handshake Lines	general	relay ladder structured text
ALMA Analog Alarm	general	relay ladder structured text function block
ALM Alarm	process	structured text function block
ALMD Digital Alarm	general	relay ladder structured text function block
AND Bitwise AND	general	relay ladder structured text function block
ARD ASCII Read	general	relay ladder structured text
ARL ASCII Read Line	general	relay ladder structured text

Instruction	Location	Languages
ASIN Arc Sine	general	structured text
ASN Arc Sine	general	relay ladder function block
ATAN Arc Tangent	general	structured text
ATN Arc Tangent	general	relay ladder function block
AVE File Average	general	relay ladder
AWA ASCII Write Append	general	relay ladder structured text
AWT ASCII Write	general	relay ladder structured text
BAND Boolean AND	general	structured text function block
BNOT Boolean NOT	general	structured text function block
BOR Boolean OR	general	structured text function block
BRK Break	general	relay ladder
BSL Bit Shift Left	general	relay ladder
BSR Bit Shift Right	general	relay ladder
BTD Bit Field Distribute	general	relay ladder
BTDT Bit Field Distribute with Target	general	structured text function block
BTR Message	general	relay ladder structured text

Instruction	Location	Languages
BTW Message	general	relay ladder structured text
BXOR Boolean Exclusive OR	general	structured text function block
CLR Clear	general	relay ladder structured text
CMP Compare	general	relay ladder
CONCAT String Concatenate	general	relay ladder structured text
COP Copy File	general	relay ladder structured text
COS Cosine	general	relay ladder structured text function block
CPS Synchronous Copy File	general	relay ladder structured text
CPT Compute	general	relay ladder
CTD Count Down	general	relay ladder
CTU Count Up	general	relay ladder
CTUD Count Up/Down	general	structured text function block
D2SD Discrete 2-State Device	process	structured text function block
D3SD Discrete 3-State Device	process	structured text function block
DDT Diagnostic Detect	general	relay ladder
DEDT Deadtime	process	structured text function block
DEG Degrees	general	relay ladder structured text function block
DELETE String Delete	general	relay ladder structured text
DERV Derivative	process	structured text function block
DFF D Flip-Flop	process	structured text function block
DIV Divide	general	relay ladder structured text function block
DTOS DINT to String	general	relay ladder structured text

Instruction	Location	Languages
DTR Data Transitional	general	relay ladder
EOT End of Transition	general	relay ladder structured text
EQU Equal to	general	relay ladder structured text function block
ESEL Enhanced Select	process	structured text function block
EVENT Trigger Event Task	general	relay ladder structured text
FAL File Arithmetic and Logic	general	relay ladder
FBC File Bit Comparison	general	relay ladder
FFL FIFO Load	general	relay ladder
FFU FIFO Unload	general	relay ladder
FGEN Function Generator	process	structured text function block
FIND Find String	general	relay ladder structured text
FLL File Fill	general	relay ladder
FOR For	general	relay ladder
FRD Convert to Integer	general	relay ladder function block
FSC File Search and Compare	general	relay ladder
GEQ Greater than or Equal to	general	relay ladder structured text function block
GRT Greater Than	general	relay ladder structured text function block
GSV Get System Value	general	relay ladder structured text
HLL High/Low Limit	process	structured text function block
HPF High Pass Filter	process	structured text function block
ICON Input Wire Connector	process	function block
INSERT Insert String	general	relay ladder structured text

Instruction	Location	Languages
INTG Integrator	process	structured text function block
IOT Immediate Output	general	relay ladder structured text
IREF Input Reference	process	function block
JKFF JK Flip-Flop	process	structured text function block
JMP Jump to Label	general	relay ladder
JSR Jump to Subroutine	general	relay ladder structured text function block
JXR Jump to External Routine	general	relay ladder
LBL Label	general	relay ladder
LDL2 Second-Order Lead Lag	process	structured text function block
LDLG Lead-Lag	process	structured text function block
LEQ Less Than or Equal to	general	relay ladder structured text function block
LES Less Than	general	relay ladder structured text function block
LFL LIFO Load	general	relay ladder
LFU LIFO Unload	general	relay ladder
LIM Limit	general	relay ladder function block
LN Natural Log	general	relay ladder structured text function block
LOG Log Base 10	general	relay ladder structured text function block
LOWER Lower Case	general	relay ladder structured text
LPF Low Pass Filter	process	structured text function block
MAAT Motion Apply Axis Tuning	218	relay ladder structured text
MAFR Motion Axis Fault Reset	47	relay ladder structured text
MAG Motion Axis Gear	87	relay ladder structured text

Instruction	Location	Languages
MAHD Motion Apply Hookup Diagnostics	229	relay ladder structured text
MAH Motion Axis Home	60	relay ladder structured text
MAJ Motion Axis Jog	65	relay ladder structured text
MAM Motion Axis Move	75	relay ladder structured text
MAOC Motion Arm Output Cam	186	relay ladder structured text
MAPC Motion Axis Position Cam	115	relay ladder structured text
MAR Motion Arm Registration	176	relay ladder structured text
MASD Motion Axis Shutdown	37	relay ladder structured text
MAS Motion Axis Stop	50	relay ladder structured text
MASR Motion Axis Shutdown Reset	40	relay ladder structured text
MATC Motion Axis Time Cam	138	relay ladder structured text
MAVE Moving Average	process	structured text function block
MAW Motion Arm Watch	170	relay ladder structured text
MAXC Maximum Capture	process	structured text function block
MCCP Motion Calculate Cam Profile	109	relay ladder structured text
MCD Motion Change Dynamics	98	relay ladder structured text
MCR Master Control Reset	general	relay ladder
MCT Motion Coordinated Transform	338	relay ladder structured text
MCTP Motion Calculate Transform Position	350	relay ladder structured text
MDF Motion Direct Drive Off	45	relay ladder structured text
MDOC Motion Disarm Output Cam	213	relay ladder structured text

Instruction	Location	Languages
MDO Motion Direct Drive On	42	relay ladder structured text
MDR Motion Disarm Registration	183	relay ladder structured text
MDW Motion Disarm Watch	174	relay ladder structured text
MEQ Mask Equal to	general	relay ladder structured text function block
MGSD Motion Group Shutdown	161	relay ladder structured text
MGS Motion Group Stop	156	relay ladder structured text
MGSP Motion Group Strobe Position	166	relay ladder structured text
MGSR Motion Group Shutdown Reset	164	relay ladder structured text
MID Middle String	general	relay ladder structured text
MINC Minimum Capture	process	structured text function block
MOD Modulo	general	relay ladder structured text function block
MOV Move	general	relay ladder
MRAT Motion Run Axis Tuning	223	relay ladder structured text
MRHD Motion Run Hookup Diagnostics	234	relay ladder structured text
MRP Motion Redefine Position	103	relay ladder structured text
MSF Motion Servo Off	34	relay ladder structured text
MSG Message	general	relay ladder structured text
MSO Motion Servo On	31	relay ladder structured text
MSTD Moving Standard Deviation	process	structured text function block
MUL Multiply	general	relay ladder structured text function block
MUX Multiplexer	process	function block

Instruction	Location	Languages
MVM Masked Move	general	relay ladder
MVMT Masked Move with Target	general	structured text function block
NEG Negate	general	relay ladder structured text function block
NEQ Not Equal to	general	relay ladder structured text function block
NOP No Operation	general	relay ladder
NOT Bitwise NOT	general	relay ladder structured text function block
NTCH Notch Filter	process	structured text function block
OCON Output Wire Connector	process	function block
ONS One Shot	general	relay ladder
OR Bitwise OR	general	relay ladder structured text function block
OREF Output Reference	process	function block
OSFI One Shot Falling with Input	general	structured text function block
OSF One Shot Falling	general	relay ladder
OSRI One Shot Rising with Input	general	structured text function block
OSR One Shot Rising	general	relay ladder
OTE Output Energize	general	relay ladder
OTL Output Latch	general	relay ladder
OTU Output Unlatch	general	relay ladder
PATT Attach to Equipment Phase	phase	relay ladder structured text
PCLF Equipment Phase Clear Failure	phase	relay ladder structured text
PCMD Equipment Phase Command	phase	relay ladder structured text

Instruction	Location	Languages
PDET Detach from Equipment Phase	phase	relay ladder structured text
PFL Equipment Phase Failure	phase	relay ladder structured text
PIDE Enhanced PID	process	structured text function block
PID Proportional Integral Derivative	general	relay ladder structured text
PI Proportional + Integral	process	structured text function block
PMUL Pulse Multiplier	process	structured text function block
POSP Position Proportional	process	structured text function block
POVR Equipment Phase Override Command	phase	relay ladder structured text
PPD Equipment Phase Paused	phase	relay ladder structured text
PRNP Equipment Phase New Parameters	phase	relay ladder structured text
PSC Phase State Complete	phase	relay ladder structured text
PXRQ Equipment Phase External Request	phase	relay ladder structured text
RAD Radians	general	relay ladder structured text function block
RESD Reset Dominant	process	structured text function block
RES Reset	general	relay ladder
RET Return	general	relay ladder structured text function block
RLIM Rate Limiter	process	structured text function block
RMPS Ramp/Soak	process	structured text function block
RTO Retentive Timer On	general	relay ladder
RTOR Retentive Timer On with Reset	general	structured text function block
RTOS REAL to String	general	relay ladder structured text

Instruction	Location	Languages
SBR Subroutine	general	relay ladder structured text function block
SCL Scale	process	structured text function block
SCRV S-Curve	process	structured text function block
SEL Select	process	function block
SETD Set Dominant	process	structured text function block
SFP SFC Pause	general	relay ladder structured text
SFR SFC Reset	general	relay ladder structured text
SIN Sine	general	relay ladder structured text function block
SIZE Size In Elements	general	relay ladder structured text
SNEG Selected Negate	process	structured text function block
SOC Second-Order Controller	process	structured text function block
SQI Sequencer Input	general	relay ladder
SQL Sequencer Load	general	relay ladder
SQO Sequencer Output	general	relay ladder
SQR Square Root	general	relay ladder function block
SQRT Square Root	general	structured text
SRT File Sort	general	relay ladder structured text
SRTP Split Range Time Proportional	process	structured text function block
SSUM Selected Summer	process	structured text function block
SSV Set System Value	general	relay ladder structured text
STD File Standard Deviation	general	relay ladder
STOD String To DINT	general	relay ladder structured text

Instruction	Location	Languages
STOR String To REAL	general	relay ladder structured text
SUB Subtract	general	relay ladder structured text function block
SWPB Swap Byte	general	relay ladder structured text
TAN Tangent	general	relay ladder structured text function block
TND Temporary End	general	relay ladder
TOD Convert to BCD	general	relay ladder function block
TOFR Timer Off Delay with Reset	general	structured text function block
TOF Timer Off Delay	general	relay ladder
TONR Timer On Delay with Reset	general	structured text function block
TON Timer On Delay	general	relay ladder
TOT Totalizer	process	structured text function block
TRN Truncate	general	relay ladder function block
TRUNC Truncate	general	structured text
UID User Interrupt Disable	general	relay ladder structured text
UIE User Interrupt Enable	general	relay ladder structured text
UPDN Up/Down Accumulator	process	structured text function block
UPPER Upper Case	general	relay ladder structured text
XIC Examine If Closed	general	relay ladder
XIO Examine If Open	general	relay ladder
XOR Bitwise Exclusive OR	general	relay ladder structured text function block
XPY X to the Power of Y	general	relay ladder structured text function block

Table of Contents

Chapter 5

Motion Event Instructions (MAW, MDW, MAR, MDR, MAOC, MDOC)

Chapter 6

Motion Configuration Instructions (MAAT, MRAT, MAHD, MRHD)

Chapter 7

Motion Coordinated Instructions (MCLM, MCCM, MCCD, MCS, MCSD, MCT, MCTP, MCSR)

Chapter 8

Tune an S-curve Profile

Chapter 9

Troubleshoot Axis Motion

Introduction

This manual is one of several Logix5000-based instruction manuals.

Task/Goal	Documents
Program the controller for sequential applications	Logix5000 Controllers General Instructions Reference Manual, publication 1756-RM003
Program the controller for process or drives applications	Logix5000 Controllers Process Control and Drives Instructions Reference Manual, publication 1756-RM006
Program the controller for motion applications **You are here**	Logix5000 Controllers Motion Instructions Reference Manual, publication 1756-RM007
Program the controller to use equipment phases	PhaseManager User Manual, publication LOGIX-UM001
Import a text file or tags into a project	Logix5000 Controllers Import/Export Reference Manual, publication 1756-RM084
Export a project or tags to a text file	
Convert a PLC-5 or SLC 500 application to a Logix5000 application	Logix5550 Controller Converting PLC-5 or SLC 500 Logic to Logix5550 Logic Reference Manual, publication 1756-6.8.5

You can use these Logix5000 controllers for motion control:

- 1756 ControlLogix controllers
- 1768 CompactLogix ontrollers (available in the future)
- 1789 SoftLogix5800 controllers
- 20D PowerFlex 700S with DriveLogix controllers

If you have a PowerFlex 700S Drive with DriveLogix controller

You can't use these instructions with a DriveLogix controller:

- Motion Direct Drive On (MDO)
- Motion Direct Drive Off (MDF)
- Motion Apply Axis Tuning (MAAT)
- Motion Run Axis Tuning (MRAT)
- Motion Apply Hookup Diagnostics (MAHD)
- Motion Run Hookup Diagnostics (MRHD)

Who Should Use This Manual

This document provides a programmer with details about the motion instructions that are available for a Logix5000 controller. You should already be familiar with how the Logix5000 controller stores and processes data.

Novice programmers should read all the details about an instruction before using the instruction. Experienced programmers can refer to the instruction information to verify details.

Purpose of This Manual

This manual provides information about each motion instruction.

This section	Provides this type of information
Instruction name	Identifies the instruction. Defines whether the instruction is an input or an output instruction.
Operands	Lists all the operands of the instruction.
Structured Text	Describes the use of operands in Structured Text format.
Motion Instruction structure	Lists control status bits and values, if any, of the instruction.
Description	Describes the instruction's use. Defines any differences when the instruction is enabled and disabled, if appropriate.
Arithmetic status flags	Defines whether or not the instruction affects arithmetic status flags.
Fault conditions	Defines whether or not the instruction generates minor or major faults. if so, defines the fault type and code.
Error Codes	Lists and defines the applicable error codes.
Status Bits	Lists affected status bits, their states, and definitions.
Example	Provides at least one programming example. Includes a description explaining each example.

Sequential Function Chart (SFC)

A Sequential Function Chart is a flowchart that controls your machine or process. SFC uses steps and transitions to perform specific operations or actions. You can use SFC to:

- Organize the functional specification of your system.
- Program and control your system as a series of steps and transitions.

You gain the following advantages by using Sequential Function Chart (SFC).

- Graphical division of processes into major logic pieces.
- Faster repeated execution of individual pieces of your logic.
- A more simple screen display.
- Time to design and debug your program is reduced.
- Troubleshooting is faster and easier.
- Direct access to the point in the logic where the machine faulted.
- Easier to enhance and update.

For more detailed information about how to program and use SFC, see Logix5000 Controllers Common Procedures Manual, publication 1756-PM001.

Conventions and Related Terms

Set and clear

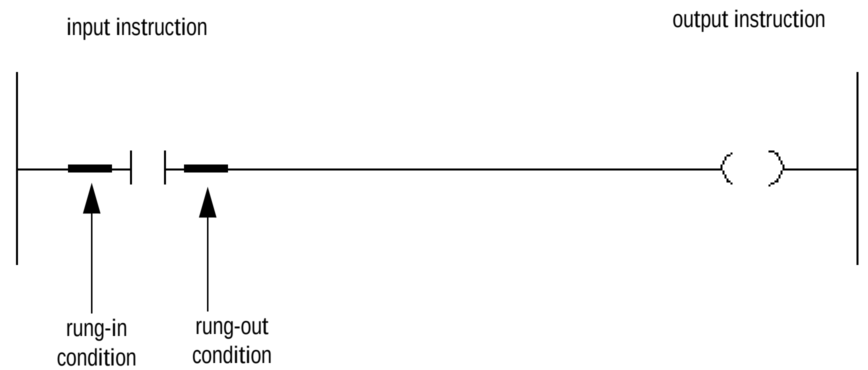
This manual uses set and clear to define the status of bits (booleans) and values (non-booleans):

This term	Means
set	the bit is set to 1 (ON) a value is set to any non-zero number
clear	the bit is cleared to 0 (OFF) all the bits in a value are cleared to 0

An instruction executes faster and requires less memory if all the operands of the instruction use the same optimal data type, typically DINT or REAL.

Rung condition

The controller evaluates ladder instructions based on the rung condition preceding the instruction (rung-condition-in). Based on the rung-condition-in and the instruction, the controller sets the rung condition following the instruction (rung-condition-out), which in turn, affects any subsequent instruction.



If the rung-in condition to an input instruction is true, the controller evaluates the instruction and sets the rung-out condition based on the results of the instruction. If the instruction evaluates to true, the rung-out condition is true; if the instruction evaluates to false, the rung-out condition is false.

Motion Concepts

Introduction

This chapter covers concepts that are common to all the motion instructions.

For This Information	See Page
Instruction Timing	17
Program a Velocity Profile	22
Choose a Command	26

Instruction Timing

Motion instructions use three types of timing sequences.

Timing type	Description
Immediate	The instruction completes in one scan.
Message	The instruction completes over several scans because the instruction sends messages to the servo module.
Process	The instruction could take an indefinite amount of time to complete.

Immediate Type Instructions

Immediate type motion instructions execute to completion in one scan. If the controller detects an error during the execution of these instructions, the error status bit sets and the operation ends.

Examples of immediate type instructions include the:

- Motion Change Dynamics (MCD) instruction
- Motion Group Strobe Position (MGSP) instruction

Immediate instructions work as follows:

1. When the rung that contains the motion instruction becomes true, the controller:

   - Sets the enable (.EN) bit.
   - Clears the done (.DN) bit.
   - Clears the error (.ER) bit.

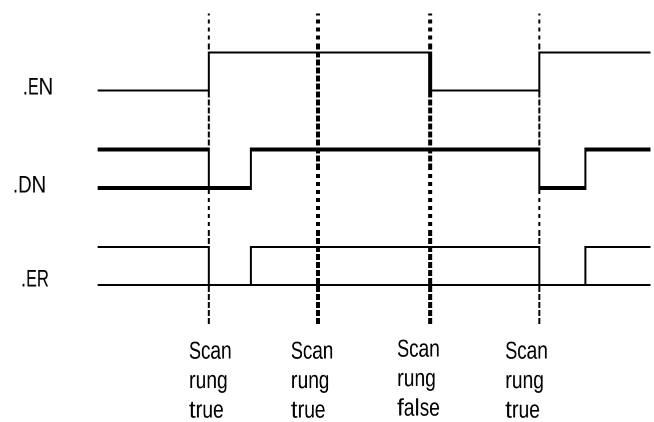
   The controller executes the instruction completely.

2.

If the controller	Then
Does not detect an error when the instruction executes	The controller sets the .DN bit.
Detects an error when the instruction executes	The controller sets the .ER bit and stores an error code in the control structure.

3. The next time the rung becomes false after either the .DN or .ER bit sets, the controller clears the .EN bit.

4. The controller can execute the instruction again when the rung becomes true.



Immediate Type Instructions - Rung Conditions

Message Type Instructions

Message type motion instructions send one or more messages to the servo module.

Examples of message type instructions include the:

- Motion Direct Drive On (MDO) instruction
- Motion Redefine Position (MRP) instruction

Message type instructions work as follows:

1. When the rung that contains the motion instruction becomes true, the controller:

    - Sets the enable (.EN) bit.
    - Clears the done (.DN) bit.
    - Clears the error (.ER) bit.

2. The controller begins to execute the instruction by setting up a message request to the servo module.

    The remainder of the instruction executes in parallel to the program scan.

3. The controller checks if the servo module is ready to receive a new message.

4. The controller places the results of the check in the message status word of the control structure.

5. When the module is ready, the controller constructs and transmits the message to the module.

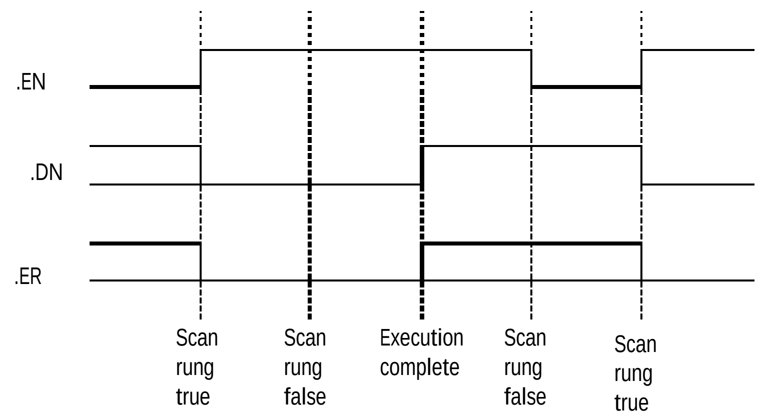
    This process may repeat several times if the instruction requires multiple messages.

6.

If the controller	Then
Does not detect an error when the instruction executes	The controller sets the .DN bit if all messaging to the module is completed.
Detects an error when the instruction executes	The controller sets the .ER bit and stores an error code in the control structure.

7. The next time the rung becomes false after either the .DN or .ER bit sets, the controller clears the .EN bit.

8. When the rung becomes true, the controller can execute the instruction again.



Message Type Instructions - Rung Conditions

Process Type Instructions

Process type motion instructions initiate motion processes that can take an indefinite amount of time to complete.

Examples of process type instructions include the:

- Motion Arm Watch Position (MAW) instruction
- Motion Axis Move (MAM) instruction

Process type instructions work as follows:

1. When the rung that contains the motion instruction becomes true, the controller:
   - Sets the enable (.EN) bit.
   - Clears the done (.DN) bit.
   - Clears the error (.ER) bit.
   - Clears the process complete (.PC) bit.
   - Sets the in process (.IP) bit.
2. The controller initiates the motion process.

3.

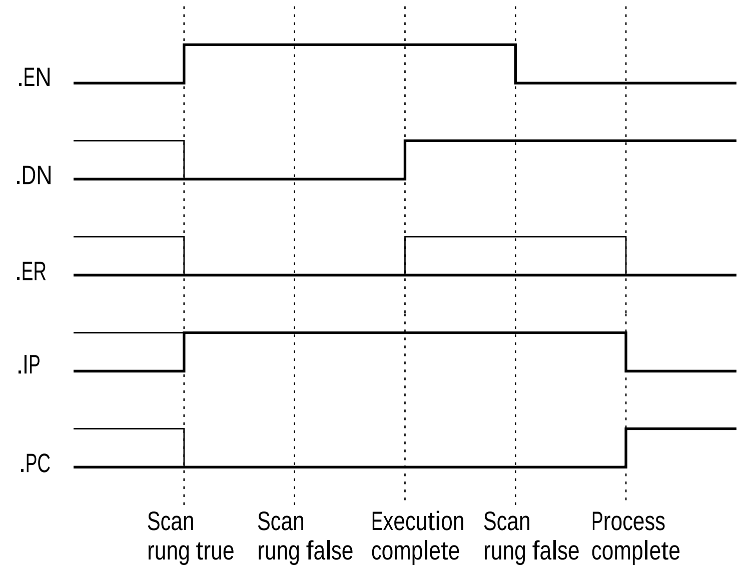
If	Then the controller
The controller does not detect an error when the instruction executes	• Sets the .DN bit. • Sets the in process (.IP) bit.
The controller detects an error when the instruction executes	• Sets the .ER bit. • Stores an error code in the control structure.
The controller detects another instance of the motion instruction	Clears the .IP bit for that instance.
The motion process reaches the point where the instruction can be executed again	Sets the .DN bit. For some process type instructions, like MAM, this occurs on the first scan. For others, like MAH, the.DN bit is not set until the entire homing process is complete.
One of the following occurs during the motion process: • The motion process completes • Another instance of the instruction executes • Another instruction stops the motion process • A motion fault stops the motion process	Clears the .IP bit.

4. After the initiation of the motion process, the program scan can continue.

   The remainder of the instruction and the control process continue in parallel with the program scan.

5. The next time the rung becomes false after either the .DN bit or the .ER bit sets, the controller clears the .EN bit.

6. When the rung becomes true, the instruction can execute again.



Process Type Instructions - Rung Conditions

Program a Velocity Profile

You can use either of these motion profiles for various instructions:

- **Trapezoidal** profile for linear acceleration and deceleration
- **S-curve** profiles for controlled jerk

For	See Page
Definition of Jerk	22
Choose a Profile	23
Use % of Time for the Easiest Programming of Jerk	24
Velocity Profile Effects	25
Jerk Rate Calculation	25

Definition of Jerk

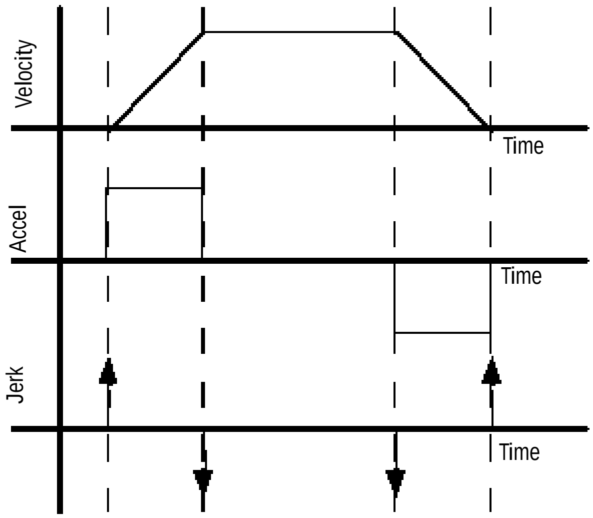
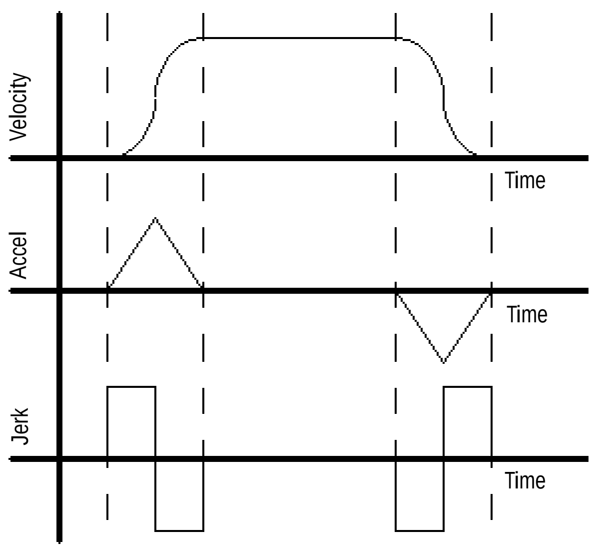
Jerk is the rate of change of acceleration or deceleration.

Example: If acceleration changes from 0 to 40 mm/s^2 in 0.2 seconds, the jerk is:

$(40\ mm/s^2 - 0\ mm/s^2)\ /\ 0.2\ s = 200\ mm/s^3$

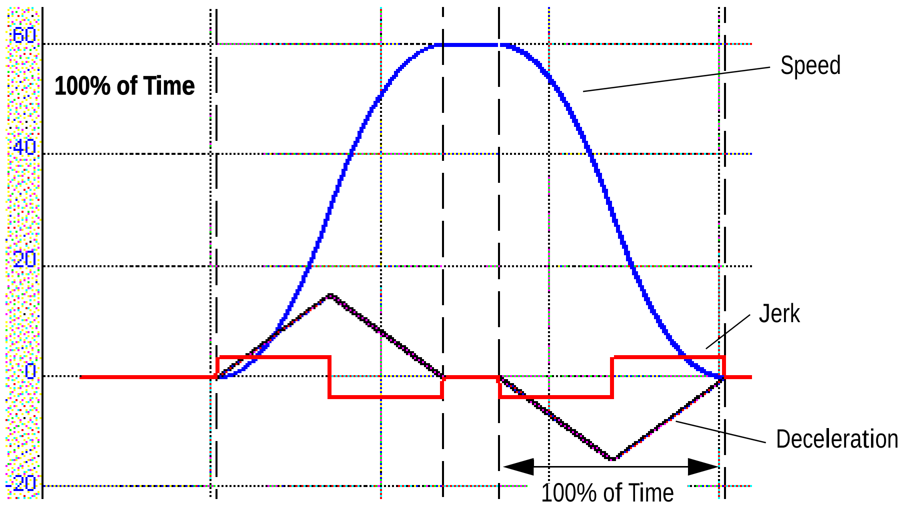
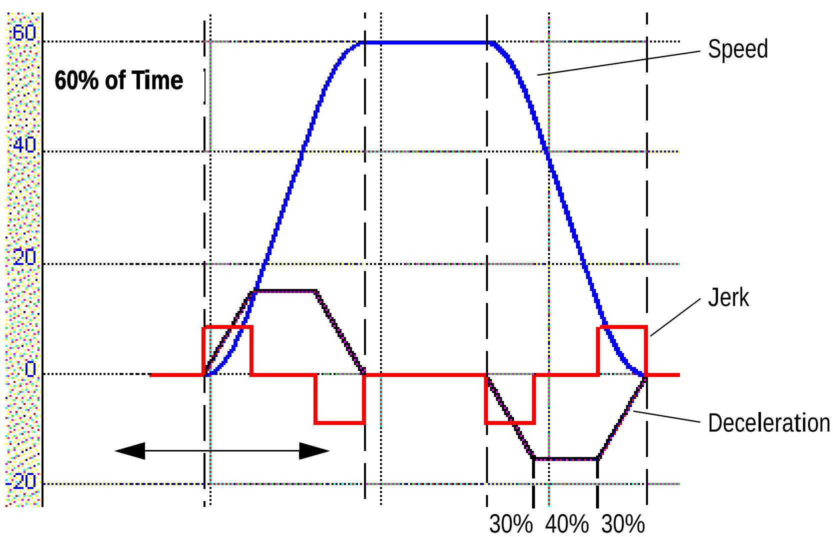
Choose a Profile

Consider cycle time and smoothness when you choose a profile.

If you want	Choose this Profile	Consideration
• Fastest acceleration and deceleration times • More flexibility in programming subsequent motion	Trapezoidal 	Jerk **doesn't** limit the acceleration and deceleration time: • The Acceleration and Deceleration rates control the maximum change in Velocity. • Your equipment and load get more stress than with an S-curve profile. • Jerk is considered infinite and is shown as a vertical line.
Smoother acceleration and deceleration that reduces the stress on the equipment and load	S-curve 	Jerk limits the acceleration and deceleration time: • It takes longer to accelerate and decelerate than a trapezoidal profile. • If the instruction uses an S-curve profile, the controller calculates acceleration, deceleration, and jerk when you start the instruction. • The controller calculates triangular acceleration and deceleration profiles.

Use % of Time for the Easiest Programming of Jerk

Use % of Time to specify how much of the acceleration or deceleration time has jerk. You don't have to calculate actual jerk values.

Example	Profile
100% of Time	At 100% of Time, the acceleration or deceleration changes the entire time that the axis speeds up or slows down. 
60% of Time	At 60% of Time, the acceleration or deceleration changes 60% of the time that the axis speeds up or slows down. The acceleration or deceleration is constant for the other 40%. 

Velocity Profile Effects

This table summarizes the differences between profiles:

Profile Type	**ACC/DEC** Time	**Motor** Stress	**Priority of Control** **Highest to Lowest**			
Trapezoidal	Fastest	Worst	Acc/Dec	Velocity	Position	
S-Curve	2X Slower	Best	Jerk	Acc/Dec	Velocity	Position

Jerk Rate Calculation

If the instruction uses or changes an S-curve profile, the controller calculates acceleration, deceleration, and jerk when you start the instruction.

Jerk is calculated as follows:

- Accel Jerk = (Max Accel^2 / Max Velocity) * (200 / % of Time - 1)
- Decel Jerk = (Max Decel^2 / Max Velocity) * (200 / % of Time - 1)

Which revision do you have?

- 15 or earlier – % of Time is fixed at 100.
- 16 or later – % of Time defaults to 100 but you can change it.

Choose a Command

Use this table to choose an instruction and see if it is available as a Motion Direct Command:

If You Want To	And	Use This Instruction	Motion Direct Command
Change the state of an axis	Enable the servo drive and activate the axis servo loop.	MSO Motion Servo On	Yes
	Disable the servo drive and deactivate the axis servo loop.	MSF Motion Servo Off	Yes
	Force an axis into the shutdown state and block any instructions that initiate axis motion.	MASD Motion Axis Shutdown	Yes
	Transition an axis to the ready state. If all of the axes of a servo module are removed from the shutdown state as a result of this instruction, the OK relay contacts for the module close.	MASR Motion Axis Shutdown Reset	Yes
	Enable the servo drive and set the servo output voltage of an axis.	MDO Motion Direct Drive On	Yes
	Disable the servo drive and set the servo output voltage to the output offset voltage.	MDF Motion Direct Drive Off	Yes
	Clear all motion faults for an axis.	MAFR Motion Axis Fault Reset	Yes
Control axis position	Stop any motion process on an axis.	MAS Motion Axis Stop	Yes
	Home an axis.	MAH Motion Axis Home	Yes
	Jog an axis.	MAJ Motion Axis Jog	Yes
	Move an axis to a specific position.	MAM Motion Axis Move	Yes
	Start electronic gearing between 2 axes	MAG Motion Axis Gear	Yes
	Change the speed, acceleration, or deceleration of a move or a jog that is in progress.	MCD Motion Change Dynamics	Yes
	Change the command or actual position of an axis.	MRP Motion Redefine Position	Yes
	Calculate a Cam Profile based on an array of cam points.	MCCP Motion Calculate Cam Profile	No
	Start electronic camming between 2 axes.	MAPC Motion Axis Position Cam	No
	Start electronic camming as a function of time.	MATC Motion Axis Time Cam	No
	Calculate the slave value, slope, and derivative of the slope for a cam profile and master value.	MCSV Motion Calculate Slave Values	No

If You Want To	And	Use This Instruction	Motion Direct Command
Initiate action on all axes	Stop motion of all axes.	MGS Motion Group Stop	Yes
	Force all axes into the shutdown state.	MGSD Motion Group Shutdown	Yes
	Transition all axes to the ready state.	MGSR Motion Group Shutdown Reset	Yes
	Latch the current command and actual position of all axes.	MGSP Motion Group Strobe Position	Yes
Arm and disarm special event checking functions such as registration and watch position	Arm the watch-position event checking for an axis.	MAW Motion Arm Watch Position	Yes
	Disarm the watch-position event checking for an axis.	MDW Motion Disarm Watch Position	Yes
	Arm the servo-module registration-event checking for an axis.	MAR Motion Arm Registration	Yes
	Disarm the servo-module registration-event checking for an axis.	MDR Motion Disarm Registration	Yes
	Arm an output cam for an axis and output.	MAOC Motion Arm Output Cam	No
	Disarm one or all output cams connected to an axis.	MDOC Motion Disarm Output Cam	No
Tune an axis and run diagnostic tests for your control system. These tests include: • Motor/encoder hookup test • Encoder hookup test • Marker test	Use the results of an MAAT instruction to calculate and update the servo gains and dynamic limits of an axis.	MAAT Motion Apply Axis Tuning	No
	Run a tuning motion profile for an axis	MRAT Motion Run Axis Tuning	No
	Use the results of an MRHD instruction to set encoder and servo polarities.	MAHD Motion Apply Hookup Diagnostic	No
	Run one of the diagnostic tests on an axis.	MRHD Motion Run Hookup Diagnostic	No

If You Want To	And	Use This Instruction	Motion Direct Command
Control multi-axis coordinated motion	Start a linear coordinated move for the axes of coordinate system.	MCLM Motion Coordinated Linear Move	No
	Start a circular move for the for the axes of coordinate system.	MCCM Motion Coordinated Circular Move	No
	Change in path dynamics for the active motion on a coordinate system.	MCCD Motion Coordinated Change Dynamics	No
	Stop the axes of a coordinate system or cancel a transform.	MCS Motion Coordinated Stop	No
	Shutdown the axes of a coordinate system.	MCSD Motion Coordinated Shutdown	No
	Start a transform that links two coordinate systems together. This is like bi-directional gearing.	MCT Motion Coordinated Transform [1]	No
	Calculate the position of one coordinate system with respect to another coordinate system.	MCTP Motion Calculate Transform Position[1]	No
	Transition the axes of a coordinate system to the ready state and clear the axis faults.	MCSR Motion Coordinated Shutdown Reset	No

[1] You can only use this instruction with 1756-L6x controllers.

Chapter 2

Motion State Instructions

(MSO, MSF, MASD, MASR, MDO,MDF, MAFR)

ATTENTION

Tags used for the motion control attribute of instructions should only be used once. Re-use of the motion control tag in other instructions can cause unintended operation. This may result in damage to equipment or personal injury.

Introduction

Motion state control instructions directly control or change the operating states of an axis. The motion state instructions are:

If you want to	Use this instruction	Available in these languages
Enable the servo drive and activate the axis servo loop.	MSO	relay ladder structured text
Disable the servo drive and deactivate the axis servo loop.	MSF	relay ladder structured text
Force an axis into the shutdown operating state. Once the axis is in the shutdown operating state, the controller will block any instructions that initiate axis motion.	MASD	relay ladder structured text
Change an axis from an existing shutdown operating state to an axis ready operating state. If all of the axes of a servo module are removed from the shutdown state as a result of this instruction, the OK relay contacts for the module will close.	MASR	relay ladder structured text
Enable the servo drive and set the servo output voltage of an axis.	MDO	relay ladder structured text
Deactivate the servo drive and set the servo output voltage to the output offset voltage.	MDF	relay ladder structured text
Clear all motion faults for an axis.	MAFR	relay ladder structured text

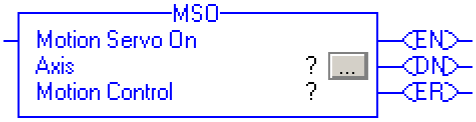
The five operating states of an axis are:

Operating State	Description
Axis ready	This is the normal power-up state of the axis. In this state: • the servo module drive enable output is inactive. • servo action is disabled. • no servo faults are present.
Direct drive control	This operating state allows the servo module DAC to directly control an external drive. In this state: • the servo module drive enable output is active. • position servo action is disabled.
Servo control	This operating state allows the servo module to perform closed loop motion. In this state: • the servo module drive enable output is active. • servo action is enabled. • the axis is forced to maintain the commanded servo position.
Axis faulted	In this operating state, a servo fault is present, and the status of the drive enable output, the action of the servo, and the condition of the OK contact depend on the faults and fault actions that are present.
Shutdown	This operating state allows the OK relay contacts to open a set of contacts in the E-stop string of the drive power supply. In this state: • the servo module drive enable output is inactive. • servo action is disabled. • the OK contact is open.

Motion Servo On (MSO)

Use the MSO instruction to activate the drive amplifier for the specified axis and to activate the axis' servo control loop.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Axis	AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.

Structured Text

MSO(Axis,MotionControl);

The operands are the same as those for the relay ladder MSO instruction.

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set when the axis' servo action has been successfully enabled and the drive enable and servo active status bits have been set.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.

Description: The Motion Servo On (MSO) instruction directly activates the drive and enables the configured servo loops associated with a physical servo axis. It can be used anywhere in a program, but should not be used while the axis is moving. If this is attempted, the MSO instruction generates an "Axis in Motion" error.

The MSO instruction automatically enables the specified axis by activating the drive and by activating the associated servo loop. The resulting state of the axis is referred to the Servo Control state.

The most common use of this instruction is to activate the servo loop for the specified axis in its current position in preparation for commanding motion.

To successfully execute a MSO instruction, the targeted axis must be configured as a Servo axis. If this condition is not met the instruction errors.

IMPORTANT The MSO instruction execution may take multiple scans to execute because it requires transmission of a message to the motion module and time for the drive output to stabilize and the servo loop to activate. The Done (.DN) bit is not set immediately, but only after the axis is in the Servo Control state.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes See Error Codes (ERR) for Motion Instructions.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions. The following Extended Error codes help to pinpoint the problem when the MSO instruction receives a Servo Message Failure (12) error message.

Extended Error Code (decimal)	Associated Error Code (decimal)	Meaning
Object Mode conflict (12)	SERVO_MESSAGE_FAILURE (12)	Axis is in shutdown.
Permission Denied (15)	SERVO_MESSAGE_FAILURE (12)	Enable input switch error. (SERCOS)
Device in wrong state (16)	SERVO_MESSAGE_FAILURE (12)	Device state not correct for action. (SERCOS)

MSO Changes to Status Bits

Axis Status Bits

Bit Name	State	Meaning
ServoActStatus	TRUE	Axis is in Servo Control state with the servo loop active.
DriveEnableStatus	TRUE	The axis drive enable output is active.

Motion Status Bits

None

Example: When the input conditions are true, the controller enables the servo drive and activates the axis servo loop configured by *axis1*.

Relay Ladder

MSO Ladder Example

Structured Text

MSO(Axis0,MSO_1);

Motion Servo Off (MSF)

Use the MSF instruction to deactivate the drive output for the specified axis and to deactivate the axis' servo loop.

IMPORTANT If you execute an MSF instruction while the axis is moving, the axis coasts to an uncontrolled stop.

Operands:

Relay Ladder

Operand	Type	Format	Description
Axis	AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis to perform action upon.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.

Structured Text

MSF(Axis,MotionControl);

The operands are the same as those for the relay ladder MSF instruction.

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set when the axis' servo action been successfully disabled and the drive enable and servo active status bits have both been cleared.
.ER (Done) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.

Description:

The Motion Servo Off (MSF) instruction directly and immediately turns off drive output and disables the servo loop on any physical servo axis. This places the axis in the Axis Ready state. The MSF instruction also disables any motion planners that may be active at the time of execution. The MSF instruction requires no parameters – simply enter or select the desired axis.

If the targeted axis does not appear in the list of available axes, the axis has not been configured for operation. Use the Tag Editor to create and configure a new axis.

You can use the MSF instruction to turn servo action OFF when you must move the axis by hand. Since the position continues to be tracked even with servo action OFF. When the servo loop is turned ON again, by the MSO instruction, the axis is again under closed-loop control, at the new position.

The axis stopping behavior varies depending upon the type of drive. In some cases the axis coasts to a stop and in other cases the axis decelerates to a stop using the drive's available stopping torque.

To execute an MSF instruction successfully, the targeted axis must be configured as a Servo axis. If this condition is not met, the instruction errs. IF you have an Axis Type of Virtual the instructions errors because with a Virtual Axis the servo action and drive enable status are forced to always be true. A Consumed axis data type also errors because only the producing controller can change the state of a consumed axis.

IMPORTANT	The MSF instruction execution may take multiple scans to execute because it requires transmission of a message to the motion module and time for the drive output and servo loop to be fully deactivated. The Done (.DN) bit is not set until this message has been successfully transmitted and the axis transitions to the Axis Ready state.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes See Error Codes (ERR) for Motion Instructions.

MSF Changes to Status Bits:

Axis Status Bits

Bit Name	State	Meaning
ServoActionStatus	FALSE	Axis is in Servo On state with the servo loop active.
DecelStatus	FALSE	Axis Drive Enable output is active.

Motion Status Bits

Bit Name	State	Meaning
AccelStatus	FALSE	Axis is not Accelerating.
DecelStatus	FALSE	Axis is not Decelerating.
MoveStatus	FALSE	Axis is not Moving.
JogStatus	FALSE	Axis is not Jogging.
GearingStatus	FALSE	Axis is not Gearing.
HomingStatus	FALSE	Axis is not Homing.
StoppingStatus	FALSE	Axis is not Stopping.
PositionCamStatus	FALSE	Axis is not Position Camming.
TimeCamStatus	FALSE	Axis is not Time Camming.
PositionCamPendingStatus	FALSE	Axis does not have a Position Cam Pending.
TimeCamPendingStatus	FALSE	Axis does not have a Time Cam Pending.
GearingLockStatus	FALSE	Axis is not in a Gear Locked condition.
PositionCamLockStatus	FALSE	Axis is not in a Cam Locked condition.

Example:

When the input conditions are true, the controller disables the servo drive and the axis servo loop configured by *Axis0*.

Relay Ladder

MSF Ladder Example

Structured Text

```
MSF(Axis0,MSF_1);
```

Motion Axis Shutdown (MASD)

Use the MASD instruction to force a specified axis into the Shutdown state. The Shutdown state of an axis is the condition where the drive output is disabled, servo loop deactivated, and any available or associated OK solid-state relay contacts open. The axis will remain in the Shutdown state until either an Axis or Group Shutdown Reset is executed.

Operands: *Relay Ladder*

Operand	Type	Format	Description
Axis	AXIS_FEEDBACK AXIS_VIRTUAL AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	The name of the axis to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.

Structured Text

```
MASD(Axis,MotionControl);
```

The operands are the same as those for the relay ladder MASD instruction.

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set when the axis have been successfully set to Shutdown state.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.

Description: The Motion Axis Shutdown (MASD) instruction directly and immediately disables drive output, disables the servo loop, and opens any associated OK contacts. This action places the axis into the Shutdown state.

Another action initiated by the MASD instruction is the clearing of all motion processes in progress and the clearing of all the motion status bits. Associated with this action, the command also clears all motion instruction IP bits that are currently set for the targeted axis.

The MASD instruction forces the targeted axis into the Shutdown state. One of the unique characteristics of the Shutdown state is that, when available, the OK solid state relay contact for the motion module or drive is Open. This feature can be used to open up the E-Stop string that controls main power to the drive system. Note that there is typically only one OK contact per motion module which means that execution of an MASD instruction for either axis associated with a given module opens the OK contact.

Another characteristic of the Shutdown state is that any instruction that initiates axis motion is blocked from execution. Attempts to do so result in an execution error. Only by executing one of the Shutdown Reset instructions can motion be successfully initiated.

To successfully execute a MASD instruction, the targeted axis must be configured as either a Servo or Feedback Only axis. If not, the instruction errs.

The axis remains in the shutdown state until either a Motion Axis Shutdown Reset (MASR) instruction or a Motion Group Shutdown Reset (MGSR) instruction executes.

IMPORTANT	The MASD instruction execution may take multiple scans to execute because it requires transmission of a message to the motion module. Thus, the Done (.DN) bit is not set until after this message is successfully transmitted and the axis is in the Shutdown state.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes See Error Codes (ERR) for Motion Instructions.

MASD Changes to Status Bits:

Axis Status Bits

Bit Name	State	Meaning
ServoActStatus	FALSE	• The axis is in the axis ready state. • The servo loop is inactive.
DriveEnableStatus	FALSE	The drive enable output is inactive.
ShutdownStatus	TRUE	The axis is in the shutdown state.

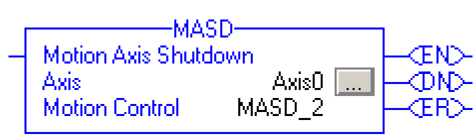
Motion Status Bits

Bit Name	State	Meaning
AccelStatus	FALSE	Axis is not Accelerating
DecelStatus	FALSE	Axis is not Decelerating
MoveStatus	FALSE	Axis is not Moving
JogStatus	FALSE	Axis is not Jogging
GearingStatus	FALSE	Axis is not Gearing
HomingStatus	FALSE	Axis is not Homing
StoppingStatus	FALSE	Axis is not Stopping
PositionCamStatus	FALSE	Axis is not Position Camming
TimeCamStatus	FALSE	Axis is not Time Camming
PositionCamPendingStatus	FALSE	Axis does not have a Position Cam Pending.
TimeCamPendingStatus	FALSE	Axis does not have a Time Cam Pending.
GearingLockStatus	FALSE	Axis is not in a Gear Locked condition
PositionCamLockStatus	FALSE	Axis is not in a Cam Locked condition

Example:

When the input conditions are true, the controller forces *axis1* into the shutdown operating state.

Relay Ladder



MASD Ladder Example

Structured Text

MASD(Axis0,MASD_1);

Motion Axis Shutdown Reset (MASR)

Use the MASR instruction to transition an axis from an existing Shutdown state to an Axis Ready state. All faults associated with the specified axis are automatically cleared. If, as a result of this instruction, all axes of the associated motion module are no longer in the Shutdown condition, the OK relay contacts for the module close.

Operands: *Relay Ladder*

Operand	Type	Format	Description
Axis	AXIS_FEEDBACK AXIS_VIRTUAL AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.

Structured Text

MASR(Axis,MotionControl);

The operands are the same as those for the relay ladder MASR instruction.

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set when the axis is successfully reset from Shutdown state.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.

Description: The Motion Axis Shutdown Reset (MASR) instruction clears all axis faults and takes the specified axis out of the Shutdown state. If the motion module supports an OK contact, and no other module axis is in the Shutdown state, the MASR instruction results in closure of the module's OK solid-state relay contact. Regardless of the OK contact condition, execution of the MASR places the axis into the Axis Ready state.

Just as the MASD instruction forces the targeted axis into the Shutdown state, the MASR instruction takes the axis out of the Shutdown state into the Axis Ready state. One of the unique characteristics of the Shutdown state is that any associated OK solid state relay contact for the motion module is Open. If, as a result of an

MASR instruction there are no axes associated with a given motion module in the Shutdown state, the OK relay contacts close as a result of the MASR. This feature can be used to close the E-Stop string that controls main power to the drive system and, thus, permit the customer to reapply power to the drive. Note that there is typically only one OK contact per motion module which means that execution of the MASR instruction may be required for all axes associated with a given module for the OK contact to close.

To successfully execute a MASR instruction, the targeted axis must be configured as either a Servo or Feedback Only axis. Otherwise, the instruction errs.

IMPORTANT	The MASR instruction execution may take multiple scans to execute because it requires transmission of a message to the motion module. Thus, the Done (.DN) bit is not set until after the message has been successfully transmitted.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes See Error Codes (ERR) for Motion Instructions.

Status Bits:

Bit Name	State	Meaning
ShutdownStatus	FALSE	The axis is not in the shutdown state.

Example: When the input conditions are true, the controller resets *axis1* from a previous shutdown operating state into an axis ready operating state.

Relay Ladder

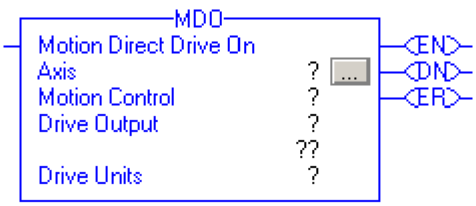
Structured Text

MASR(Axis0,MASR_1);

Motion Direct Drive On (MDO)

Use the MDO instruction in conjunction with motion modules that support an external analog servo drive interface, e.g. the 1756–M02AE or 1784-PM02AE servo module. This instruction activates the module's Drive Enable, enabling the external servo drive, and also sets the servo module's output voltage of the drive to the specified voltage level. The value for Drive Output may be specified in Volts or % of maximum axis' Output Limit.

Operands: *Relay Ladder*



Operand	Data Type	Description
Axis	Tag	Name of the axis to perform operation on.
Motion control	MOTION_ INSTRUCTION Tag	Structure used to access instruction status parameters.
Drive Output	REAL	Voltage to output in % of servo Output Limit or in Volts
Drive Units	Boolean	Units in which the Drive Output value is interpreted.

Structured Text

MDO(Axis,MotionControl, DriveOutput,DriveUnits);

The operands are the same as those for the relay ladder MDO instruction.

For the operands that require you to select from available options, enter your selection as:

This operand	Has these options which you...	
	enter as text	**or enter as a number**
DriveUnits	volts	0
	percent	1

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set when the axis' drive enable bit is activated and the specified analog output is successfully applied.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you entered a Drive Output value that was too large.

Description: For motion module's with an external servo drive interface, like the 1756–M02AE or 1784-PM02AE, the Motion Direct Drive On (MDO) instruction can be used to directly enable the Drive Enable output of the axis and set the analog output to a specified level determined by the Drive Output parameter. The Drive Output parameter can be expressed as a voltage, or as a percent of the maximum configured output voltage value given by the Output Limit attribute.

The MDO instruction can only be used on a physical axis whose Axis Type is configured for Servo. The instruction only executes when the axis' is in the Axis Ready state, i.e., servo action is OFF. The resulting state of the axis is referred to as the Drive Control state.

The MDO instruction automatically enables the specified axis by activating the appropriate Drive Enable output before setting the servo module's analog output to the specified voltage value. There is, typically, a 500 msec delay between the activation of the drive enable output and the setting of the analog output to the specified level to allow the drive's power structure to stabilize. To minimize drift during this drive enabling delay, the output voltage to the drive is set to the Output Offset attribute value (default is zero). Thereafter the output voltage is given by the specified Drive Output value of the MDO instruction and indicated by the Servo Output status attribute value.

The 16-bit DAC hardware associated with various Logix servo modules limits the effective resolution of the Direct Drive Motion Control to 305 µV or 0.003%. In the case of Direct Drive operation, the module's servo loop is inactive and bypassed. The Motion Direct Drive On instruction is only affected by the Servo Output Polarity configuration bit, the Output Offset, and Output Limit attributes for the axis. In the case where Output Limit configuration value is reduced below the current output voltage value, the Servo Output value is automatically clamped to the Output Limit value.

The most common use of this instruction is to provide an independent programmable analog output as an open loop speed reference for an external drive or for testing an external servo drive for closed loop operation.

To successfully execute a MDO instruction, the targeted axis must be configured as a Servo axis and be in the Axis Ready state, with servo action off. If these conditions are not met the instruction errs.

IMPORTANT	The MDO instruction execution may take multiple scans to execute because it requires transmission of a message to the motion module and time for the drive output to stabilize. The Done (.DN) bit is not set until after the axis is in the Drive Control state.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes See Error Codes (ERR) for Motion Instructions.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions. The following Extended Error codes help to pinpoint the problem when the MDO instruction receives a Servo Message Failure (12) error message.

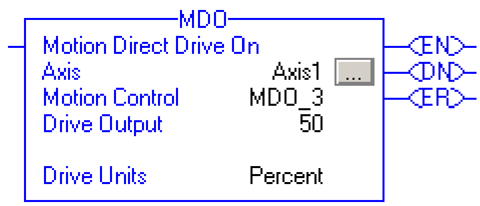
Extended Error Code (decimal)	Associated Error Code (decimal)	Meaning
Object Mode conflict (12)	SERVO_MESSAGE_FAILURE (12)	Axis is in shutdown.

Status Bits: *MDO Changes to Status Bits*

Bit Name	State	Meaning
DriveEnableStatus	TRUE	Axis is in Drive Control state with the Drive Enable output active.

Example: When the input conditions are true, the controller activates the servo drive for *axis1* and sets the servo output voltage of *axis1*. In this example, the output is 2% of the output value.

Relay Ladder



MDO Ladder Example

Structured Text

MDO(Axis0,MDO_1,4,percent);

Motion Direct Drive Off (MDF)

Use the MDF instruction to deactivate the servo drive and to set the servo output voltage to the output offset voltage. The output offset voltage is the output voltage that generates zero or minimal drive motion. You can specify this value during axis configuration.

Operands: *Relay Ladder*

Operand	Data Type	Description
Axis	Tag	Name of the axis to perform operation on.
Motion control	MOTION_ INSTRUCTION Tag	Structure used to access instruction status parameters.

Structured Text

```
MDF(Axis,MotionControl);
```

The operands are the same as those for the relay ladder MDF instruction.

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set when the axis' drive signals have been successfully disabled and the drive enable status bit is cleared.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.

Description: For motion module's with an external servo drive interface, e.g. the 1756–M02AE, the Motion Direct Drive Off (MDF) instruction directly disables the motion module Drive Enable output of the specified physical axis and also "zeroes" the modules' servo output to the external drive by applying the configured Output Offset value.

The MDF instruction is used to stop motion initiated by a preceding MDO instruction and transition the axis from the Direct Drive Control state back to the Axis Ready state.

To successfully execute an MDF instruction, the targeted axis must be configured as either a Servo or Feedback Only axis. Otherwise, the instruction errors.

IMPORTANT The MDF instruction execution may take multiple scans to execute because it requires transmission of a message to the motion module. The Done (.DN) bit is not set until after the message has been successfully transmitted.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes See Error Codes (ERR) for Motion Instructions.

MDF Changes to Status Bits: *Axis Status Bits*

Bit Name	State	Meaning
DriveEnableStatus	FALSE	Axis is in Axis Ready state with the Drive Enable output now active.

Example: When the input conditions are true, the controller deactivates the servo drive for *axis1* and sets the servo output voltage of *axis_* to the output offset value.

Relay Ladder

MDF Ladder Example

Structured Text

```
MDF(Axis0,MDF_1);
```

Motion Axis Fault Reset (MAFR)

Use the MAFR instruction to clear all motion faults for an axis. This is the only method for clearing axis motion faults.

IMPORTANT The MAFR instruction removes the fault status, but does not perform any other recovery, such as enabling servo action. In addition, when the controller removes the fault status, the condition that generated the fault(s) may still exist. If the condition is not corrected before using the MAFR instruction, the axis immediately faults again.

Operands: *Relay Ladder*

Operand	Type	Format	Description
Axis	AXIS_FEEDBACK AXIS_VIRTUAL AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.

Structured Text

MAFR(Axis,MotionControl);

The operands are the same as those for the relay ladder MAFR instruction.

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set when the axis' faults have been successfully cleared.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.

Description: The Motion Axis Fault Reset (MAFR) instruction directly clears the specified fault status on the specified axis. It does not correct the condition that caused the error. If the condition is not corrected prior to executing the MAFR instruction the axis could immediately fault again giving the appearance that the fault status was not reset.

This instruction is most commonly used as part of a fault handler program, which provides application specific fault action in response to various potential motion control faults. Once the appropriate fault action is taken, the MAFR instruction can be used to clear all active fault status bits.

To successfully execute a MAFR instruction, the targeted axis must be configured as either a Servo or Feedback Only axis. Otherwise, the instruction errors.

IMPORTANT The MAFR instruction execution may take multiple scans to execute because it requires transmission of a message to the motion module. The Done (.DN) bit is not set until after this message has been successfully transmitted. There is no guarantee that all faults are cleared by this instruction as one or more faults may be the result of a persistent condition.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes See Error Codes (ERR) for Motion Instructions.

MAFR Changes to Status Bits: None

MAFR Example: When the input conditions are true, the controller clears all motion faults for *axis1*.

Relay Ladder

MAFR Ladder Example

Structured Text

MAFR(Axis0,MAFR_1);

Motion Move Instructions

(MAS, MAH, MAJ, MAM, MAG, MCD, MRP, MCCP, MAPC, MATC, MCSV)

Introduction

Use the Motion Move instructions to control axis position.

If You Want To	Use This Instruction	Available In These Languages
Stop any motion process on an axis	Motion Axis Stop (MAS)	relay ladder structured text
Home an axis	Motion Axis Home (MAH)	relay ladder structured text
Jog an axis	Motion Axis Jog (MAJ)	relay ladder structured text
Move an axis to a specific position	Motion Axis Move (MAM)	relay ladder structured text
Start electronic gearing between 2 axes	Motion Axis Gear (MAG)	relay ladder structured text
Change the speed, acceleration, or deceleration of a move or a jog that is in progress	Motion Change Dynamics (MCD)	relay ladder structured text
Change the command or actual position of an axis	Motion Redefine Position (MRP)	relay ladder structured text
Calculate a Cam Profile based on an array of cam points	Motion Calculate Cam Profile (MCCP)	relay ladder structured text
Start electronic camming between 2 axes	Motion Axis Position Cam (MAPC)	relay ladder structured text
Start electronic camming as a function of time	Motion Axis Time Cam (MATC)	relay ladder structured text
Calculate the slave value, slope, and derivative of the slope for a cam profile and master value	Motion Calculate Slave Values (MCSV)	relay ladder structured text

Motion Axis Stop (MAS)

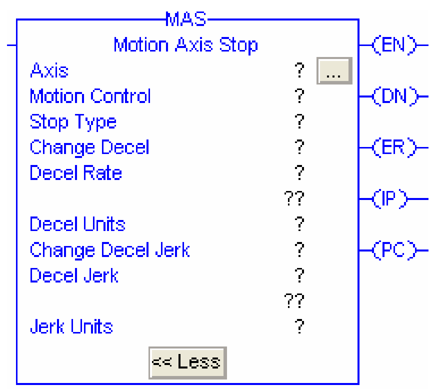
Use the MAS instruction to stop a specific motion process on an axis or to stop the axis completely.

ATTENTION

Use a motion control tag only once. Do not re-use it in another instruction. Otherwise, you can cause unexpected equipment motion and injure people.

Operands

Relay Ladder



<table>
<tr><th>Operand</th><th>Type</th><th>Format</th><th colspan="2">Description</th></tr>
<tr><td>Axis</td><td>AXIS_VIRTUAL
AXIS_GENERIC
AXIS_SERVO
AXIS_SERVO_DRIVE</td><td>Tag</td><td colspan="2">Name of the axis</td></tr>
<tr><td>Motion Control</td><td>MOTION_INSTRUCTION</td><td>Tag</td><td colspan="2">Control tag for the instruction</td></tr>
<tr><td rowspan="3">Stop Type</td><td rowspan="3">DINT</td><td rowspan="3">Immediate</td><td>To stop</td><td>Choose this Stop Type</td></tr>
<tr><td>All motion in process for this axis</td><td>All (0)</td></tr>
<tr><td>Only a certain type of motion but leave other motion processes running</td><td>Choose the type of motion that you want to stop:
Jog (1)
Move (2)
Gear (3)
Home (4)
Tune (5)
Test (6)
Position Cam (7)
Time Cam (8)
Master Offset Move (9)
The axis could still be moving when the MAS instruction is complete.</td></tr>
</table>

Operand	Type	Format	Description	
Change Decel	DINT	Immediate	**If you want to**	**Then choose**
			Use the Maximum Deceleration rate of the axis	No (0)
			Specify the deceleration rate	Yes (1)
Decel Rate	REAL	Immediate or tag	**Important**: The axis could overshoot its target position if you reduce the deceleration while a move is in process. Deceleration rate of the axis. The instruction uses this value only if Change Decel is Yes.	
Decel Units	DINT	Immediate	Which units do you want to use for the Decel Rate? • Units per sec^2 (0) • % of Maximum (1)	
Change Decel Jerk	DINT	Immediate	**If you want to**	**Then choose**
			Use the Maximum Deceleration Jerk rate of the axis	No (0)
			Program the deceleration jerk rate	Yes (1)
Decel Jerk	REAL	Immediate or tag	**Important**: The axis could overshoot its target position if you reduce the deceleration jerk while a move is in process. Deceleration jerk rate of the axis. Use these values to get started. • Decel Jerk = 100 • Jerk Units = % of Time (2) You can also enter the jerk rates in these Jerk Units. • Units per sec^3 (0) • % of Maximum (1)	
Jerk Units	DINT	Immediate		

Structured Text

```
MAS(Axis, MotionControl,
StopType, ChangeDecel,
DecelRate, DecelUnits,
ChangeDecelJerk,
DecelJerk, JerkUnits);
```

The structured text operands are the same as the relay ladder operands.

This operand	**Has these options which you can**	
	enter as text	**or enter as a number**
Stop Type	all	0
	jog	1
	move	2
	gear	3
	home	4
	tune	5
	test	6
	timecam	7
	positioncam	8
	masteroffsetmove	9
Change Decel	no	0
	yes	1
Decel Units	unitspersec2	0
	%ofmaximum	1
Change Decel Jerk	no	0
	yes	1
Jerk Units	unitspersec3	0
	%ofmaximum	1
	%oftime	2

MOTION_INSTRUCTION Data Type

To See If	Check If This Bit is Set	Data Type	Notes
A false-to-true transition caused the instruction to execute	EN	BOOL	The EN bit stays set until the process is complete and the rung goes false.
The stop was successfully initiated	DN	BOOL	
An error happened	ER	BOOL	
The axis is stopping	IP	BOOL	Any of these actions end the MAS instruction and clear the IP bit: • The axis is stopped • Another MAS instruction supersedes this MAS instruction • Shutdown command • Fault Action
The axis stopped	PC	BOOL	The PC bit stays set until the rung makes a false-to-true transition.

Description

Use the Motion Axis Stop (MAS) instruction when you want a decelerated stop for any controlled motion in process for the axis. The instruction stops the motion without disabling the servo loop. Use the instruction to:

- stop a specific motion process such as jogging, moving, or gearing
- stop the axis completely
- abort a test or tune process initiated by an MRHD or MRAT instruction

Which type of profile does the MAS instruction use?

If the Stop Type is	Then the MAS instruction use this profile
Jog	Same type of profile as the Motion Axis Jog (MAJ) instruction that started the jog
Move	Same type of profile as the Motion Axis Move (MAM) instruction that started the move
None of the above	Trapezoidal

Example

Suppose you use an MAJ instruction with an S-curve profile to start a jog. Then you use an MAS instruction with a Stop Type of Jog to stop the jog. In that case, the MAS instruction uses an S-curve profile to stop the jog.

Programming Guidelines

ATTENTION

If You Use An S-curve Profile

Be careful if you change the speed, acceleration, deceleration, or jerk while an axis is accelerating or decelerating along an S-curve profile. You can cause an axis to **overshoot its speed or reverse direction**.

For more information, see Troubleshoot Axis Motion on page 9-367.

Guideline	Details	
• **In relay ladder, toggle the rung condition each time you want to execute the instruction.**	This is a transitional instruction: • In relay ladder, toggle the rung-condition-in from cleared to set each time you want to execute the instruction.	

• **In structured text, condition the instruction so that it only executes on a transition.**	In structured text, instructions execute each time they are scanned.
	• Condition the instruction so that it only executes on a transition. Use either of these methods:
	  • qualifier of an SFC action
	  • structured text construct
	For more information, see Appendix C.	

• **Choose whether to stop all motion or only a specific type of motion.**	**If you want to stop**	**Then choose this Stop Type**
	All motion in process for this axis	All The instruction uses a trapezoidal profile and stops the axis.
	Stop only a certain type of motion but leave other motion processes running	The type of motion that you want to stop The axis could still be moving when the MAS instruction is complete. The instruction uses an S-curve profile to stop the axis only if: • The Stop Type is Jog or Move, and • The jog or move used an S-curve profile.
	Example: Suppose your axis is executing both a jog and a move at the same time. And suppose you want to stop only the jog but leave the move running. In that case, choose a Stop Type of Jog.	
• **To stop gearing or camming, select the slave axis.**	To stop a gearing or position camming process, enter the **slave** axis to turn off the specific process and stop the axis. If the master axis is a servo axis, you can stop the master axis which in turn stops the slave without disabling the gearing or position camming.	
• **To stop a Master Offset move, enter the slave axis but use master units.**	To stop an Absolute or Incremental Master Offset move: • For Axis, enter the slave axis. • For Deceleration and Jerk, enter the values and units for the master axis.	

Guideline	Details
• **Be careful if the instruction changes motion parameters.**	When you execute an MAS instruction, the axis uses the new deceleration and jerk rates for the motion that's already in process. This can cause an axis to overshoot its speed, overshoot its end position, or reverse direction. S-curve profiles are more sensitive to parameter changes. For more information, see Troubleshoot Axis Motion on page 9-367.
• **Use the jerk operands for S-curve profiles.**	Use the jerk operands when • The Stop Type is Jog or Move. • The jog or move uses an S-curve profile. Under those conditions, the instruction uses an S-curve profile to stop the axis. The instruction uses a constant deceleration rate for all other types of stops. You must fill in the jerk operands regardless of the type of stop.
• **Use % of Time for the easiest programming and tuning of jerk.**	For an easy way to program and tune jerk, enter it as a % of the acceleration or deceleration time. For more information, see: • Program a Velocity Profile on 1-25 • Tune an S-curve Profile on page 8-363.

Arithmetic Status Flags

not affected

Fault Conditions

none

Error Codes

See Error Codes (ERR) for Motion Instructions on page A-383.

Extended Error Codes

Use Extended Error Codes (EXERR) for more information about an error.

<table>
<tr><th rowspan="2">If ERR is</th><th rowspan="2">And EXERR is</th><th colspan="2">Then</th></tr>
<tr><th>Cause</th><th>Corrective Action</th></tr>
<tr><td>13</td><td>Varies</td><td>An operand is outside its range.</td><td>The EXERR is the number of the operand that is out of range. The first operand is 0.

For example, if EXERR = 4, then check the Decel Rate.
<table><tr><th>EXERR</th><th>MAS Operand</th></tr><tr><td>0</td><td>Axis</td></tr><tr><td>1</td><td>Motion Control</td></tr><tr><td>2</td><td>Stop Type</td></tr><tr><td>3</td><td>Change Decel</td></tr><tr><td>4</td><td>Decel Rate</td></tr></table></td></tr>
</table>

Changes to Status Bits

Motion Status Bits

If the Stop Typ is	Then		
NOT All	The instruction clears the Motion Status bit for the motion process that you stopped.		
All	The instruction clears all Motion Status bits.		
	Bit	**Status**	**Meaning**
	MoveStatus	FALSE	Axis is not Moving
	JogStatus	FALSE	Axis is not Jogging
	GearingStatus	FALSE	Axis is not Gearing
	HomingStatus	FALSE	Axis is not Homing
	StoppingStatus	TRUE	Axis is Stopping
	PositionCamStatus	FALSE	Axis is not Position Camming
	TimeCamStatus	FALSE	Axis is not Time Camming
	PositionCamPendingStatus	FALSE	Axis does not have a Position Cam Pending.
	TimeCamPendingStatus	FALSE	Axis does not have a Time Cam Pending.
	GearingLockStatus	FALSE	Axis is not in a Gear Locked condition
	PositionCamLockStatus	FALSE	Axis is not in a Cam Locked condition

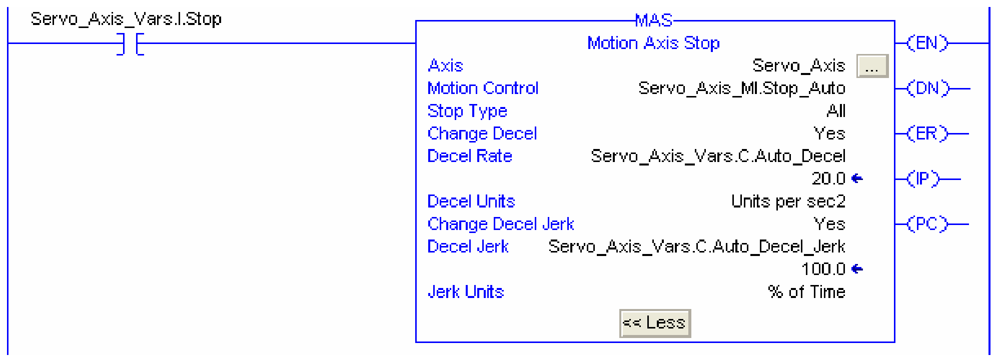
Example 1 When *Servo_Axis_Vars.I.Stop* turns on

Stop all motion on *Servo_Axis*.

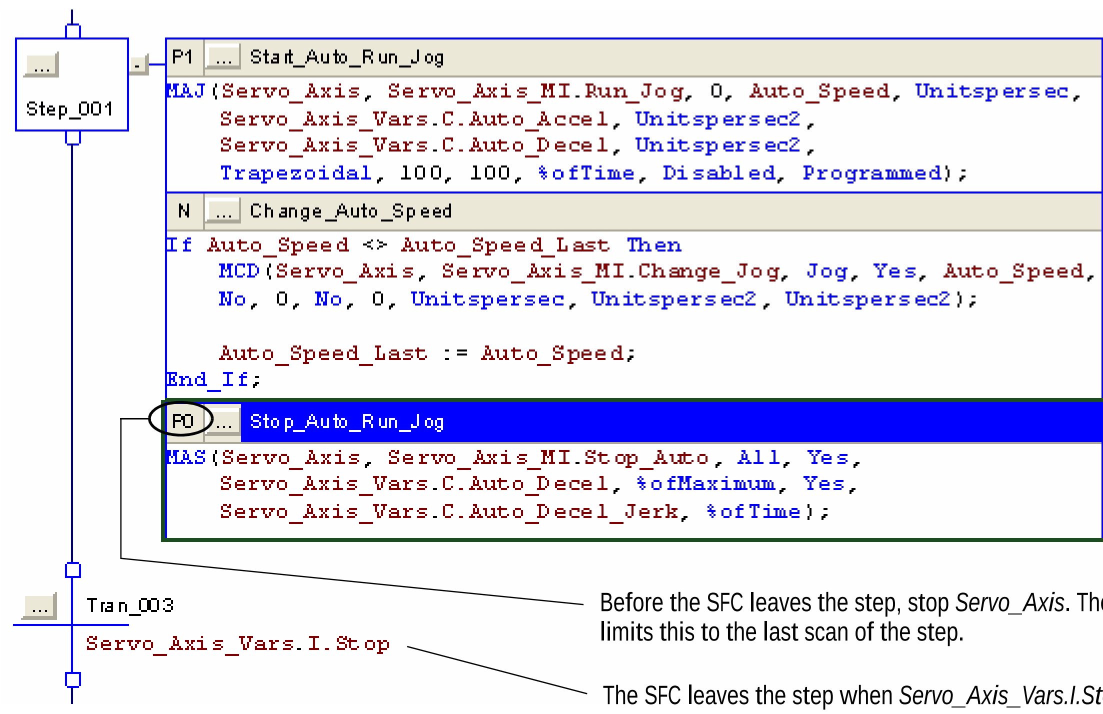
Decelerate at 20.0 units per sec2.

The instruction doesn't use the Decel Jerk value. Since the Stop Type is all, the instruction uses a trapezoidal profile to stop the axis.

Relay Ladder



Structured Text



Stop

Before the SFC leaves the step, stop *Servo_Axis*. The P0 qualifier limits this to the last scan of the step.

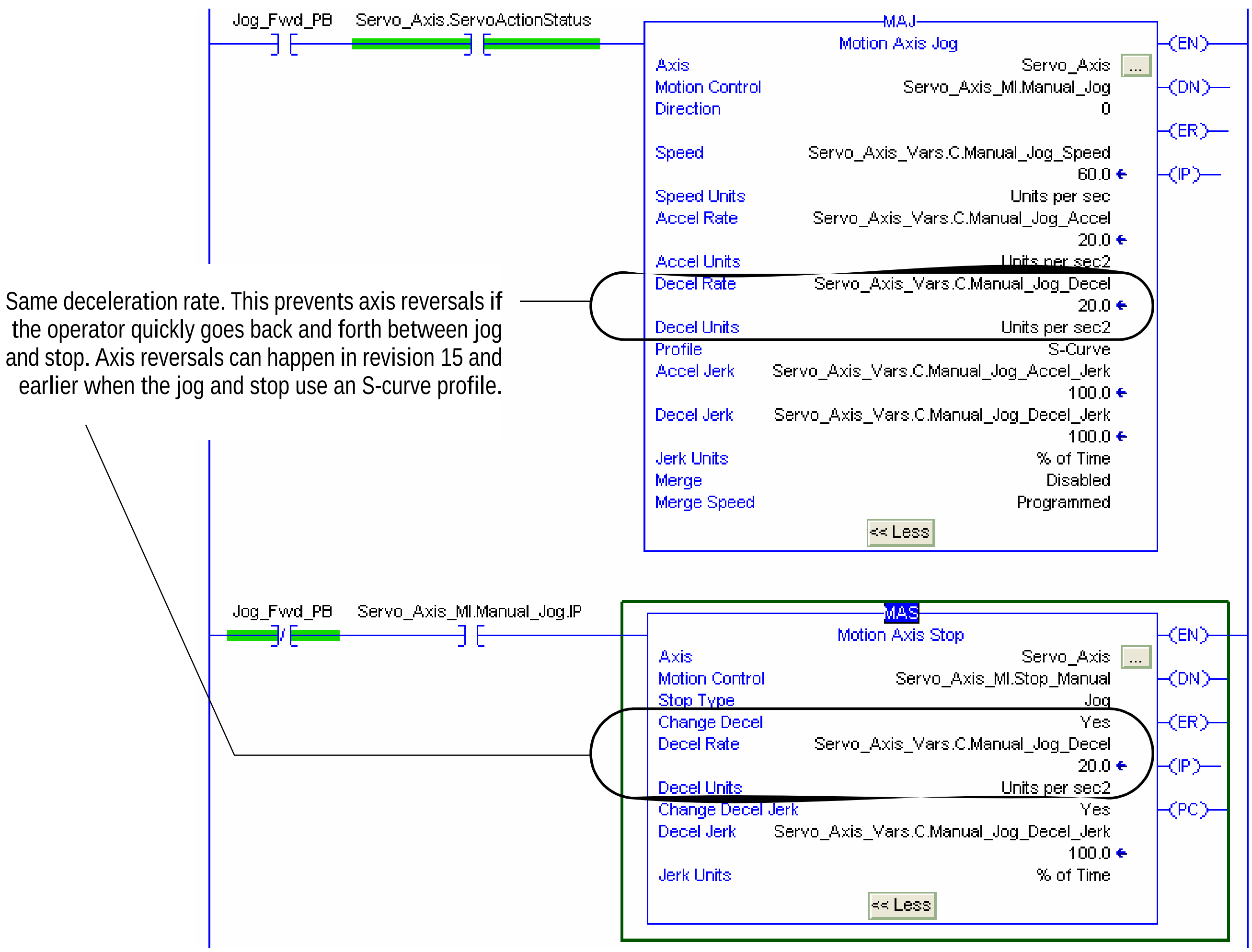
The SFC leaves the step when *Servo_Axis_Vars.I.Stop* turns on.

Example 2 The operator uses a pushbutton to jog an axis. The pushputton turns the *Jog_Fwd_PB* tag on and off. When the operator releases the button, the MAS instruction stops the axis.

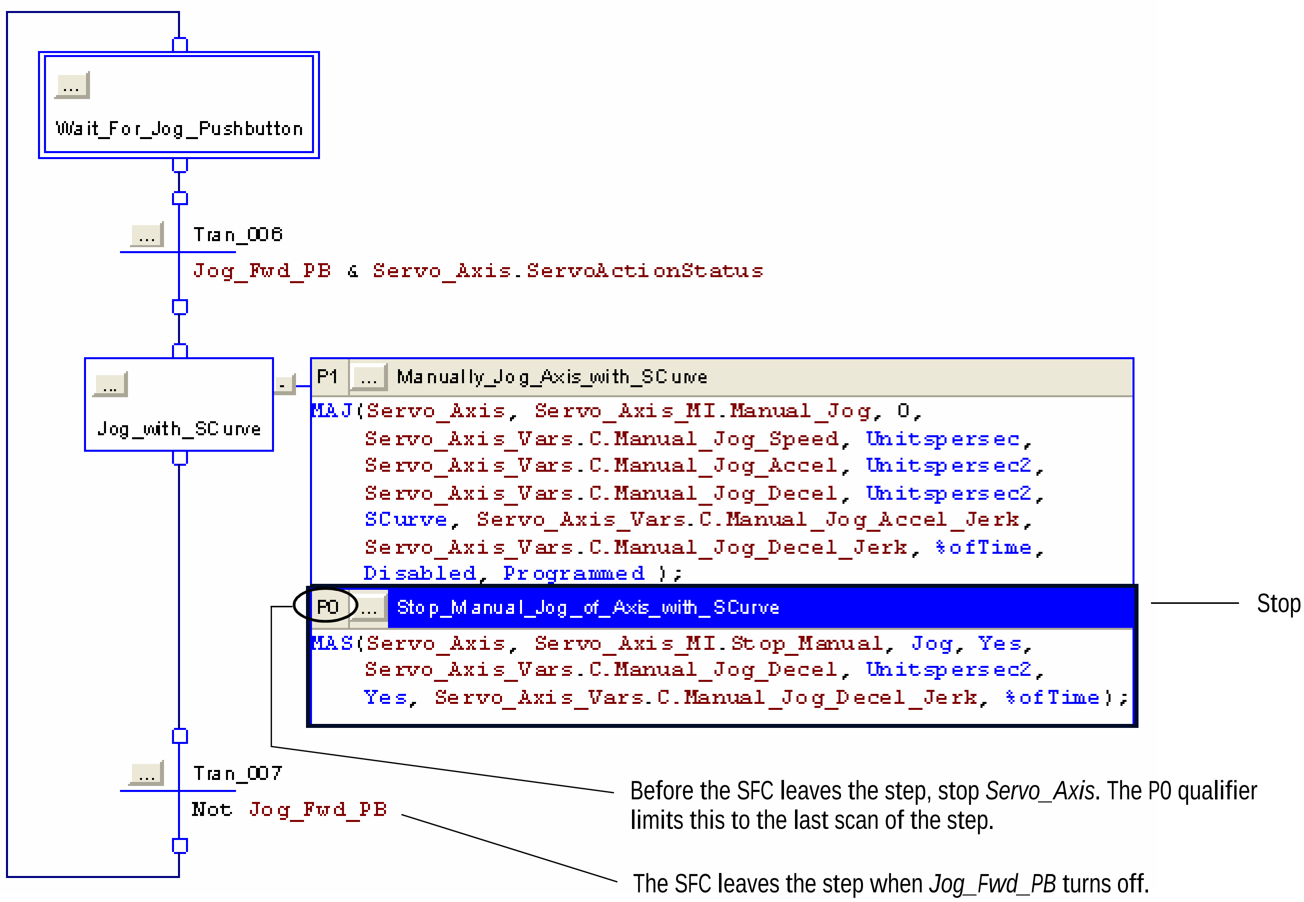
The MAS instruction uses an S-curve profile to stop the axis because:

- The MAJ instruction uses an S-curve profile.
- The Stop Type is Jog for the MAS instruction.

Relay Ladder



Structured Text



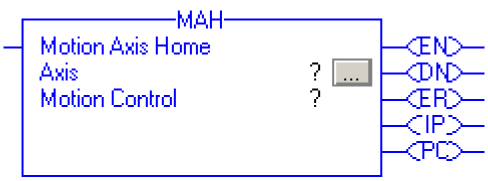
Motion Axis Home (MAH)

Use the MAH instruction to home an axis. Two different homing modes can be selected during axis configuration: Active or Passive. If an Active homing sequence is selected, the axis executes the configured Home Sequence Type and establishes an absolute axis position. If Passive homing is selected, however, no specific homing sequence is executed and the axis is left waiting for the next marker pulse to establish the home position.

ATTENTION

Use a motion control tag only once. Do not re-use it in another instruction. Otherwise, you can cause unexpected equipment motion and injure people.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Axis	AXIS_FEEDBACK AXIS_VIRTUAL AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.

Structured Text

```
MAH(Axis,MotionControl);
```

The operands are the same as those for the relay ladder MAH instruction.

MOTION_INSTRUCTION Data Type

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set when axis home has been successfully completed or is aborted.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.
.IP (In Process) Bit 27	It is set on positive rung transition and cleared after the Motion Home Axis is complete, or terminated by a stop command, shutdown, or a servo fault
.PC (Process Complete) Bit 26	It is set when axis home is successfully completed.

Description: The Motion Axis Home (MAH) instruction is used to calibrate the absolute position of the specified axis. For axes that are configured as type Servo, the axis can be homed using Active, Passive, or Absolute Homing Mode configuration. For Feedback Only axes, only Passive and Absolute homing modes are available. Absolute Homing Mode requires the axis to be equipped with absolute feedback device.

Active Homing

When the axis Homing Mode is configured as Active, the physical axis is first activated for servo operation. As part of this process all other motion in process is canceled and appropriate status bits cleared. The axis is then homed using the configured Home Sequence which may be Immediate, Switch, Marker, or Switch-Marker. The later three Home Sequences result in the axis being jogged in the configured Home Direction and then after the position is re-defined based on detection of the home event, the axis is automatically moved to the configured Home Position.

IMPORTANT	WHen unidirectional active homing is performed on a rotary axis and the Home Offset value is less than the deceleration distance when the home event is detected, the control moves the axis to the unwind position of zero. this ensures that the resulting move to the Home Position is unidirectional.

Passive Homing

When the axis Homing Mode is configured as Passive, the MAH instruction re-defines the actual position of a physical axis on the next occurrence of the encoder marker. Passive homing is most commonly used to calibrate Feedback Only axes to their markers, but can also be used on Servo axes. Passive homing is identical to active homing to an encoder marker except that the motion controller does not command any axis motion.

After initiating passive homing, the axis must be moved past the encoder marker for the homing sequence to complete properly. For closed-loop Servo axes, this may be accomplished with a MAM or MAJ instruction. For physical Feedback Only axes, motion cannot be commanded directly by the motion controller, and must be accomplished via other means.

Absolute Homing

If the motion axis hardware supports an absolute feedback device, Absolute Homing Mode may be used. The only valid Home Sequence for an absolute Homing Mode is “immediate”. In this case, the absolute homing process establishes the true absolute position of the axis by applying the configured Home Position to the reported position of the absolute feedback device. Prior to execution of the absolute homing process via the MAH instruction, the axis must be in the Axis Ready state with the servo loop disabled.

To successfully execute a MAH instruction on an axis configured for Active homing mode, the targeted axis must be configured as a Servo Axis Type. To successfully execute an MAH instruction, the targeted axis must be configured as either a Servo or Feedback Only axis. If any of these conditions are not met the instruction errs.

IMPORTANT When the MAH instruction is initially executed, the In process .IP bit is set and the Process Complete (.PC) bit is cleared. The MAH instruction execution may take multiple scans to execute because it requires transmission of multiple messages to the motion module. Thus, the Done (.DN) bit, is not set until after these messages have been successfully transmitted. The In process .IP bit is cleared and the Process Complete (.PC) bit is set at the same time that the Done (.DN) bit is set.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes: See Error Codes (ERR) for Motion Instructions on page A-383.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions. The following Extended Error codes help to pinpoint the problem

when the MAH instruction receives a Servo Message Failure (12) error message or Illegal Homing Configuration (41).

Associated Error Code (decimal)	Extended Error Code (decimal)	Meaning
SERVO_MESSAGE_FAILURE (12)	Process terminated on request (1)	Home execution followed by an instruction to shutdown/disable drive, or a motion stop instruction or a Processor change requests a cancel of Home.
SERVO_MESSAGE_FAILURE (12)	No Resource (2)	Not enough memory resources to complete request. (SERCOS)
SERVO_MESSAGE_FAILURE (12)	Object Mode conflict (12)	Axis is in shutdown.
SERVO_MESSAGE_FAILURE (12)	Permission denied (15)	Enable input switch error. (SERCOS)
SERVO_MESSAGE_FAILURE (12)	Device in wrong state (16)	Redefine Position, Home, and Registration 2 are mutually exclusive (SERCOS), device state not correct for action. (SERCOS)
ILLEGAL_HOMING_CONFIG (41)	Home sequence (4)	You have an absolute homing instruction when the Homing sequence is not immediate.
ILLEGAL_HOMING_CONFIG (41)	Home speed of zero (6)	Home speed cannot be zero.
ILLEGAL_HOMING_CONFIG (41)	Home return speed of zero (7)	The Home Return Speed cannot be zero.

For the Error Code 54 – Maximum Deceleration Value is Zero, if the Extended Error returns a positive number (0-*n*) it is referring to the offending axis in the coordinate system. Go to the Coordinate System Properties General Tab and look under the Brackets ([ ])column of the Axis Grid to determine which axis has a Maximum Deceleration value of 0. Click on the ellipsis button next to the offending axis to access the Axis Properties screen. Go to the Dynamics tab and make the appropriate change to the Maximum Deceleration Value. If the Extended Error number is -1, this means the Coordinate System has a Maximum Deceleration Value of 0. Go to the Coordinate System Properties Dynamics Tab to correct the Maximum Deceleration value.

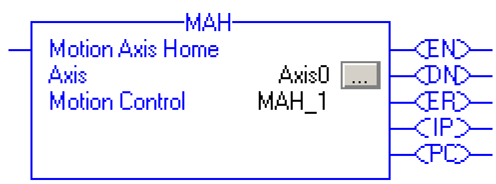
Status Bits: *MAH Changes to Status Bits*

Bit Name	State	Meaning
HomingStatus	TRUE	Axis is Homing
JogStatus	FALSE	Axis is no longer Jogging*
MoveStatus	FALSE	Axis is no longer Moving*
GearingStatus	FALSE	Axis is no longer Gearing
StoppingStatus	FALSE	Axis is no longer Stopping

During portions of the active homing sequence these bits may be set and cleared. The MAH instruction uses the Move and Jog motion profile generators to move the axis during the homing sequence. This also means that any disruption in the Move or Jog motion profiles due to other motion instructions can affect the successful completion of the MAH initiated homing sequence.

If in Passive homing mode, the MAH instruction simply sets the Homing Status bit.

Example: *Relay Ladder*



MAH Ladder Example

Structured Text

MAH(Axis0,MAH_1);

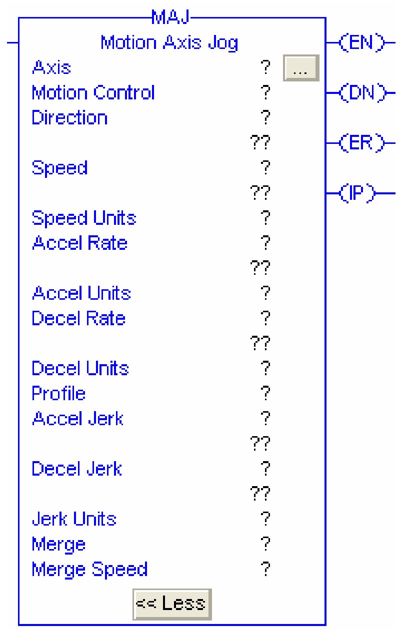
Motion Axis Jog (MAJ)

Use the MAJ instruction to move an axis at a constant speed until you tell it to stop.

ATTENTION

Use a motion control tag only once. Do not re-use it in another instruction. Otherwise, you can cause unexpected equipment motion and injure people.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Axis	AXIS_VIRTUAL AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	Tag	Name of the axis to jog
Motion Control	MOTION_INSTRUCTION	Tag	Control tag for the instruction
Direction	DINT	Immediate Tag	**For this jog direction** — **Enter** Forward — 0 Reverse — 1
Speed	REAL	Immediate Tag	Speed to move the axis in Speed Units.
Speed Units	DINT	Immediate	Which units do you want to use for the Speed? • Units per sec (0) • % of Maximum (1)

Operand	Type	Format	Description
Accel Rate	REAL	Immediate Tag	Acceleration rate of the axis in Accel Units
Accel Units	DINT	Immediate	Which units do you want to use for the Accel Rate? • Units per sec^2 (0) • % of Maximum (1)
Decel Rate	REAL	Immediate Tag	Deceleration rate of the axis in Deceleration Units.
Decel Units	DINT	Immediate	Which units do you want to use for the Decel Rate? • Units per sec^2 (0) • % of Maximum (1)
Profile	DINT	Immediate	Select the velocity profile to run the jog: • Trapezoidal (0) • S-Curve (1) For more information, see Program a Velocity Profile on page 1-22.
Accel Jerk	REAL	Immediate or tag	The instruction only uses the jerk operands if the Profile is S-curve. You must always fill them in however. • Accel Jerk is the acceleration jerk rate for the axis. • Decel Jerk is the deceleration jerk rate for the axis. Use these values to get started. • Accel Jerk = 100 • Decel Jerk = 100 • Jerk Units = % of Time (2) You can also enter the jerk rates in these Jerk Units. • Units per sec^3 (0) • % of Maximum (1)
Decel Jerk	REAL	Immediate or tag	
Jerk Units	DINT	Immediate	
Merge	DINT	Immediate	Do you want to turn all current axis motion into a pure jog governed by this instruction regardless of the motion instructions currently in process? • NO — Choose Disabled (0) • YES — Choose Enabled (1)
Merge Speed	DINT	Immediate	If Merge is Enabled, which speed do you want to jog at? • Speed of this instruction — Choose Programmed (0) • Current speed of the axis — Choose Current (1)

```
MAJ(Axis, MotionControl,
Direction, Speed,
SpeedUnits, AccelRate,
AccelUnits, DecelRate,
DecelUnits, Profile,
AccelJerk, DecelJerk,
JerkUnits, Merge,
MergeSpeed);
```

Structured Text

The structured text operands are the same as the relay ladder operands.

This operand	Has these options which you enter as text	or enter as a number
SpeedUnits	unitspersec	0
	%ofmaximum	1
AccelUnits	unitspersec2	0
	%ofmaximum	1
DecelUnits	unitspersec2	0
	%ofmaximum	1
Profile	trapezoidal	0
	scurve	1
Jerk Units	unitspersec3	0
	%ofmaximum	1
	%oftime	2
Merge	disabled	0
	enabled	1
MergeSpeed	programmed	0
	current	1

MOTION_INSTRUCTION Data Type

To See If	Check If This Bit is Set	Data Type	Notes
A false-to-true transition caused the instruction to execute	EN	BOOL	The EN bit stays set until the process is complete and the rung goes false.
The jog was successfully initiated	DN	BOOL	
An error happened	ER	BOOL	
The axis is jogging	IP	BOOL	Any of these actions stop this jog and clear the IP bit: • Another MAJ instruction supersedes this MAJ instruction • MAS instruction • Merge from another instruction • Shutdown command • Fault Action

Description: Use the MAJ instruction to move an axis at a constant speed without regard to position.

Programming Guidelines

ATTENTION

If You Use An S-curve Profile

Be careful if you change the speed, acceleration, deceleration, or jerk while an axis is accelerating or decelerating along an S-curve profile. You can cause an axis to **overshoot its speed or reverse direction**.

For more information, see Troubleshoot Axis Motion on page 9-367.

<table>
<tr><th>Guideline</th><th>Details</th></tr>
<tr><td>• In relay ladder, toggle the rung condition each time you want to execute the instruction.</td><td>This is a transitional instruction:
• In relay ladder, toggle the rung-condition-in from cleared to set each time you want to execute the instruction.</td></tr>
<tr><td>• In structured text, condition the instruction so that it only executes on a transition.</td><td>In structured text, instructions execute each time they are scanned.
• Condition the instruction so that it only executes on a transition. Use either of these methods:
  • qualifier of an SFC action
  • structured text construct
For more information, see Appendix C.</td></tr>
<tr><td>• Use the jerk operands for S-curve profiles.</td><td>Use the jerk operands when the instruction uses an S-curve profile.
You must fill in the jerk operands regardless of the profile.</td></tr>
<tr><td>• Use % of Time for the easiest programming and tuning of jerk.</td><td>For an easy way to program and tune jerk, enter it as a % of the acceleration or deceleration time.
For more information, see:
• Program a Velocity Profile on page 1-22
• Tune an S-curve Profile on page 8-363.</td></tr>
<tr><td>• Use Merge to cancel the motion of other instructions.</td><td>How you want to handle any motion that’s already in process.?
<table>
<tr><th>If you want to</th><th>And you want to</th><th>Then set</th></tr>
<tr><td>Add the jog to any motion already in process</td><td>→</td><td>Merge = Disabled
Merge Speed = Programmed
The instruction ignores Merge Speed but you must fill it in anyway.</td></tr>
<tr><td rowspan="2">End the motion from other instructions and just jog</td><td>Jog at the Speed that you set in this instruction</td><td>Merge = Enabled
Merge Speed = Programmed</td></tr>
<tr><td>Jog at the speed that the axis is already moving at</td><td>Merge = Enabled
Merge Speed = Current
The instruction ignores the value that you put in the Speed operand.</td></tr>
</table></td></tr>
</table>

Guideline	Details
• **Be careful if you start another jog while the axis is already jogging.**	If you start a new MAJ instruction while one is already in process, you can cause: • an accelerating axis to overshoot its speed • a decelerating axis to reverse direction (revision 15 and earlier) This happens if the MAJ instructions use an S-curve profile. **Reason**: The new MAJ instruction cancels the old MAJ instruction. The axis uses the speed, acceleration, deceleration, and jerk of the new instruction. For more information, see Troubleshoot Axis Motion on page 9-367.
• **Use an MAS instruction to stop the jog.**	See the examples.
• **Use an MCD instruction to change the speed while jogging.**	See Example 1 on page 3-71.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes See Error Codes (ERR) for Motion Instructions on page A-383.

Extended Error Codes Use Extended Error Codes (EXERR) for more information about an error.

If ERR is	And EXERR is	Then	
		Cause	**Corrective Action**
13	Varies	An operand is outside its range.	The EXERR is the number of the operand that is out of range. The first operand is 0. For example, if EXERR = 3, then check the Speed. **EXERR** – **Operand** 0 – Axis 1 – Motion Control 2 – Direction 3 – Speed
54	-1	The coordinate system has a Maximum Deceleration of 0.	Go to the Properties for the coordinate system and set a Maximum Deceleration.
	0 or more	An axis in the coordinate system has a Maximum Deceleration of 0.	1. Open the Properties for the coordinate system. 2. Use the EXERR value to see which axis has the Maximum Deceleration of 0. 3. Set the Maximum Deceleration for the axis.

Changes to Status Bits: *Motion Status Bits*

If Merge is	Then the instruction changes these bits		
	Bit Name	**State**	**Meaning**
Disabled	JogStatus	TRUE	The axis is Jogging.
Enabled	JogStatus	TRUE	The axis is Jogging.
	MoveStatus	FALSE	The axis is no longer Moving.
	GearingStatus	FALSE	The axis is no longer Gearing.

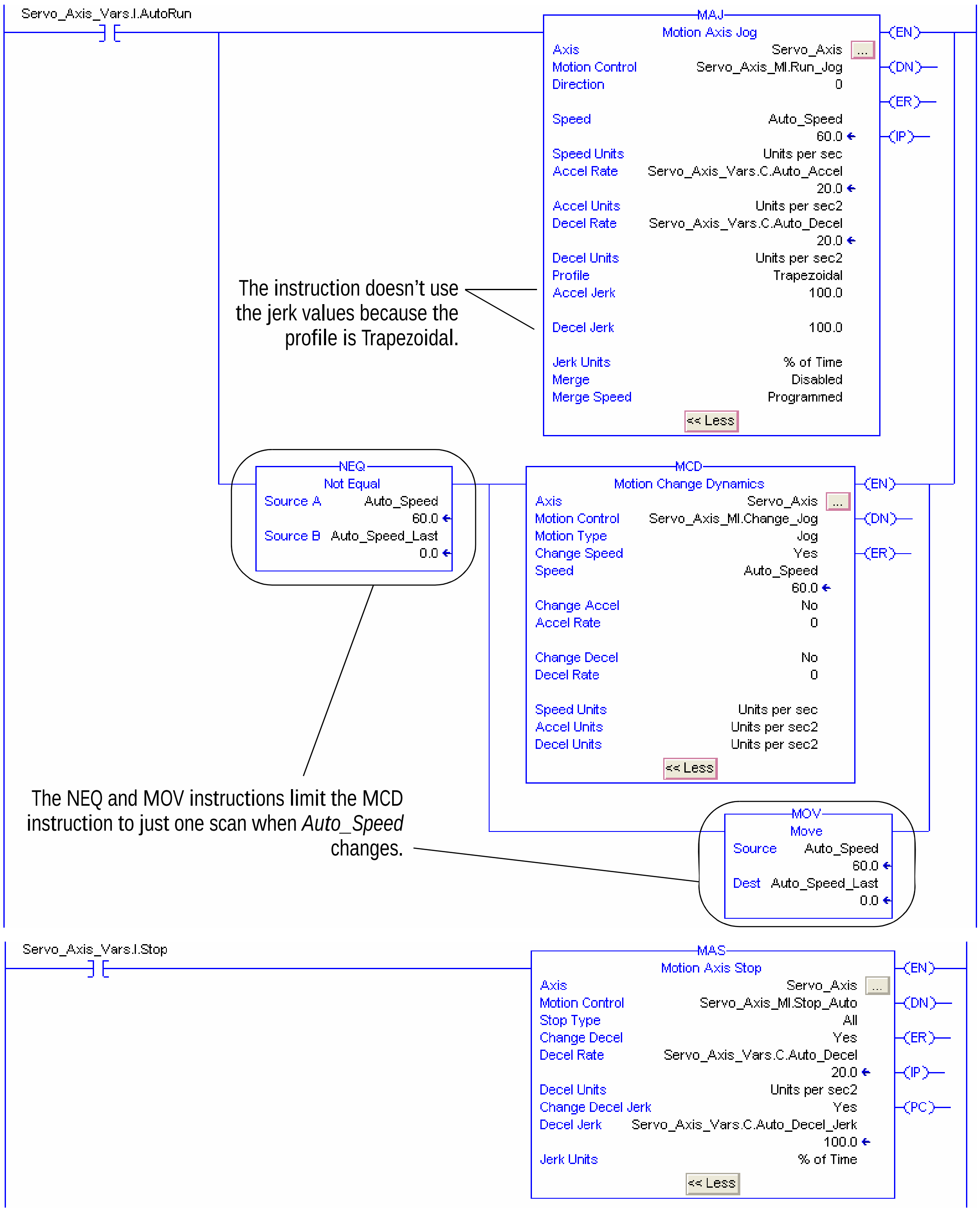
Example 1 *Jog with Speed Change—Relay Ladder*

When *Servo_Axis_Vars.I.AutoRun* turns on

Run *Servo_Axis* at *Auto_Speed*.

If *Auto_Speed* changes then change the speed of the jog to the new value of *Auto_Speed*.

When *Servo_Axis_Vars.I.Stop* turns on, stop *Servo_Axis*.



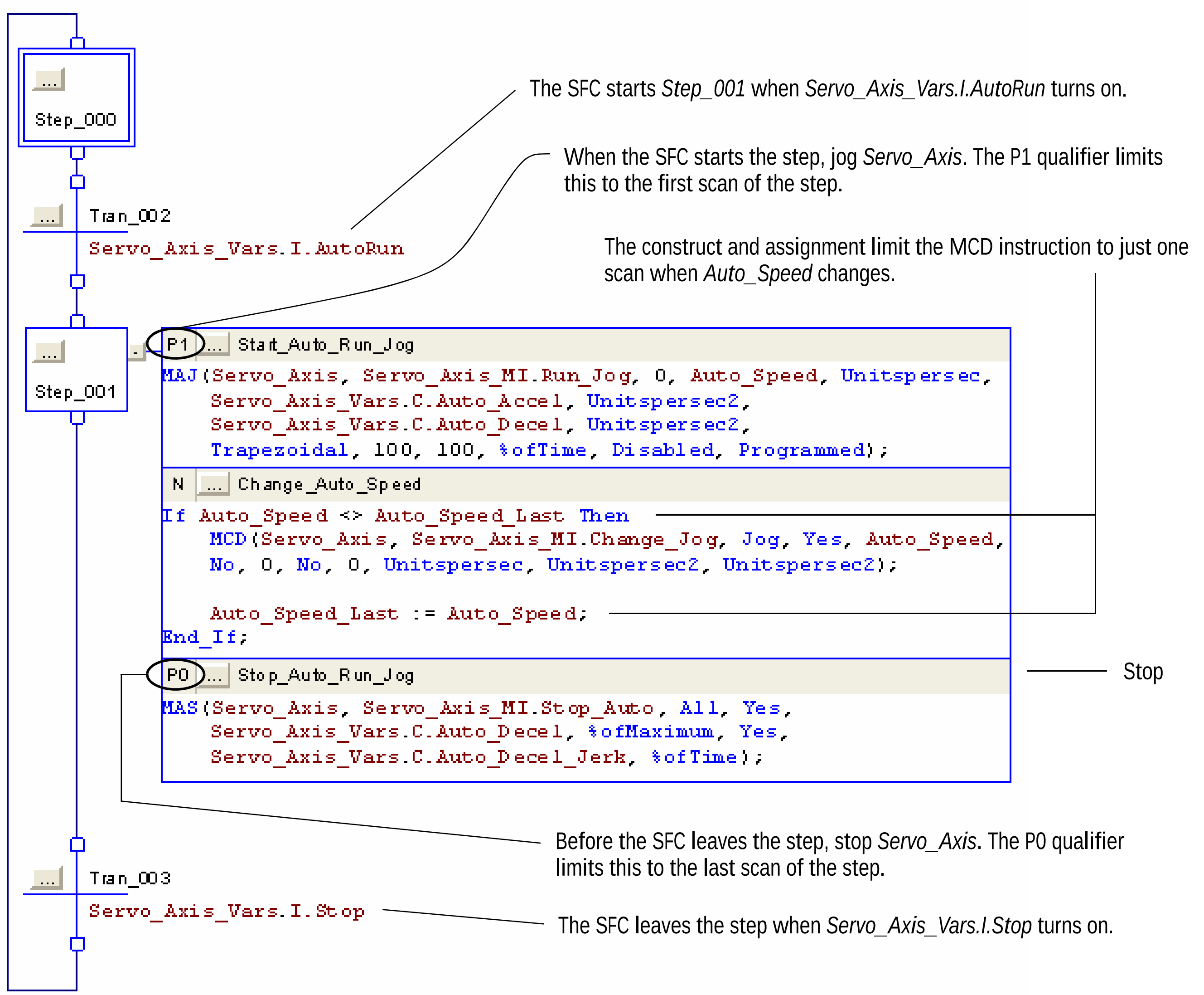
Jog with Speed Change—Structured Text

When *Servo_Axis_Vars.I.AutoRun* turns on

Run *Servo_Axis* at *Auto_Speed*.

If *Auto_Speed* changes then change the speed of the jog to the new value of *Auto_Speed*.

When *Servo_Axis_Vars.I.Stop* turns on, stop *Servo_Axis*.



Example 2 *Jog Forward and Reverse with S-curve—Relay Ladder*

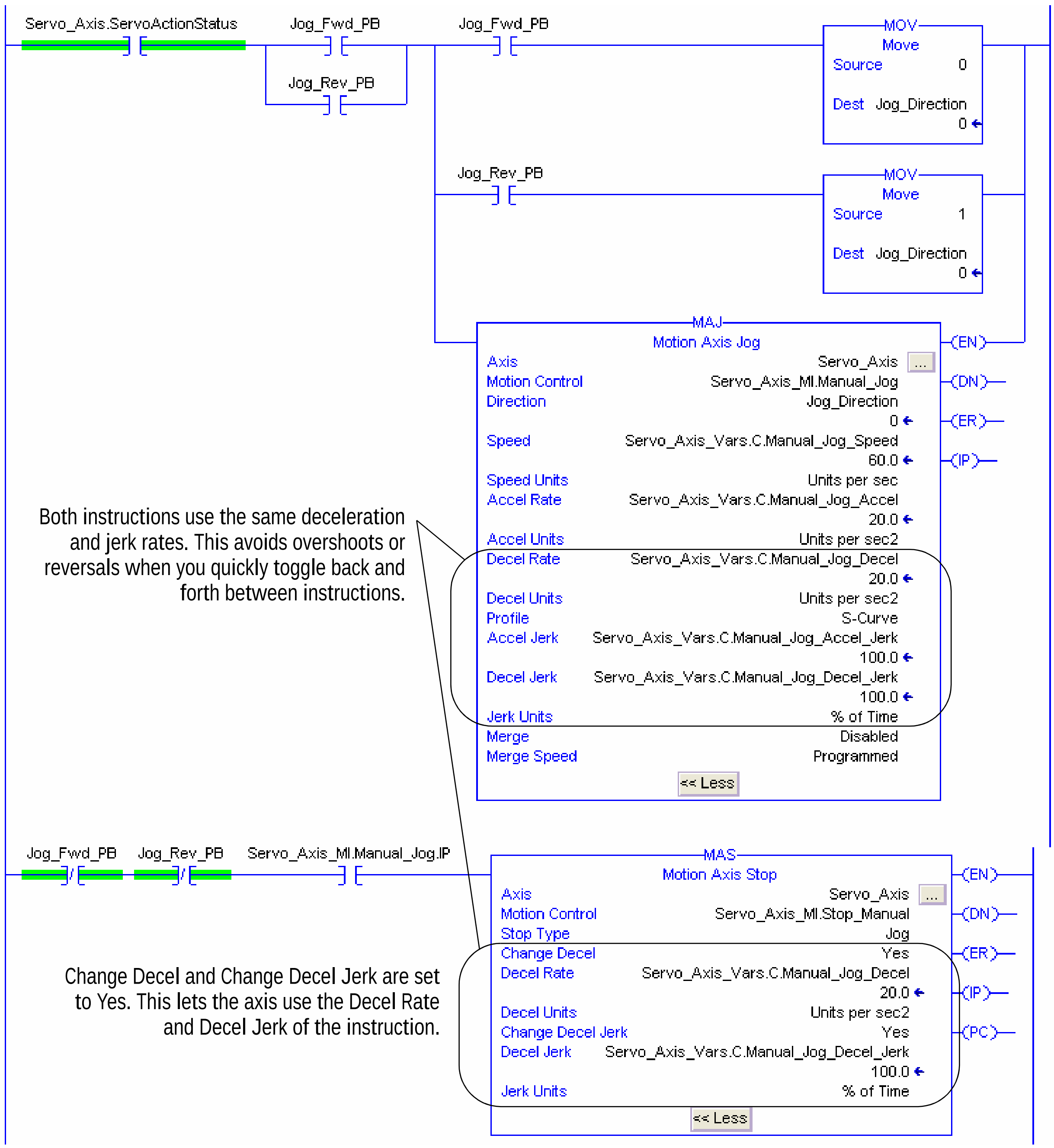
When the servo loop is enabled

And *Jog_Fwd_PB* or *Jog_Rev_PB* turn on

Set *Jog_Direction*.

Run *Servo_Axis* at *Servo_Axis_Vars.C.Manual_Jog_Speed*.

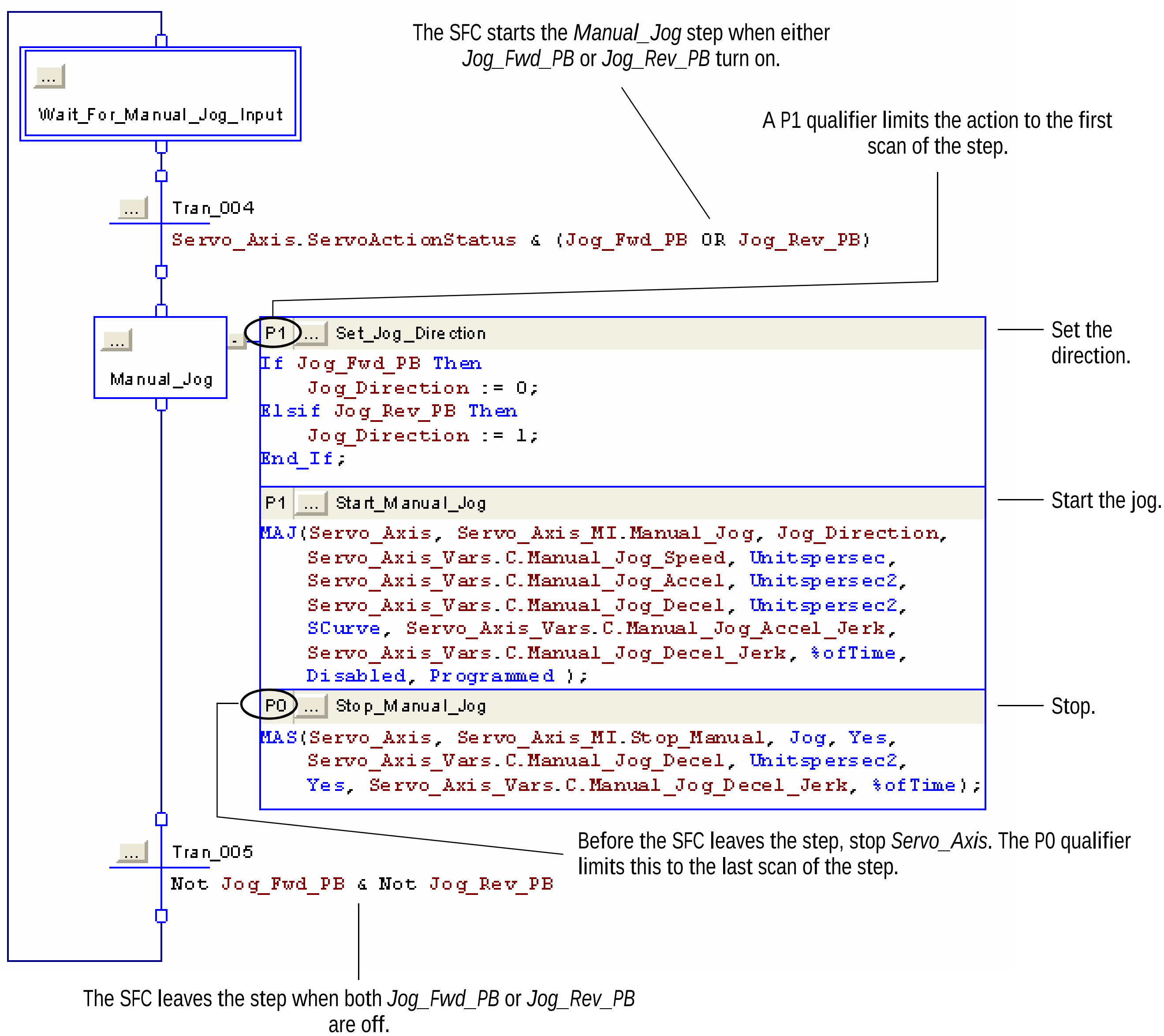
When *Jog_Fwd_PB* and *Jog_Rev_PB* are off, stop *Servo_Axis*.



Jog Forward and Reverse with S-curve—Structured Text

When the servo loop is enabled
And *Jog_Fwd_PB* or *Jog_Rev_PB* turn on
  Set *Jog_Direction*.
  Run *Servo_Axis* at *Servo_Axis_Vars.C.Manual_Jog_Speed*.

When *Jog_Fwd_PB* and *Jog_Rev_PB* are off, stop *Servo_Axis*.



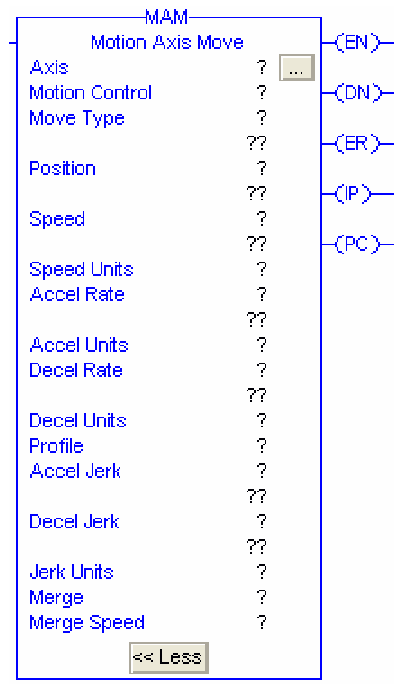
Motion Axis Move (MAM)

Use the Motion Axis Move (MAM) instruction to move an axis to a specified position.

ATTENTION

Use a motion control tag only once. Do not re-use it in another instruction. Otherwise, you can cause unexpected equipment motion and injure people.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Axis	AXIS_VIRTUAL AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	Tag	Name of the axis For an Absolute or Incremental Master Offset move, enter the slave axis.
Motion Control	MOTION_INSTRUCTION	Tag	Control tag for the instruction

<table>
<tr><th>Operand</th><th>Type</th><th>Format</th><th>Description</th></tr>
<tr><td>Move Type</td><td>DINT</td><td>Immediate
Tag</td><td>
<table>
<tr><th>To</th><th>Use This Move Type</th><th>And enter</th></tr>
<tr><td>Move an axis to an absolute position</td><td>Absolute</td><td>0</td></tr>
<tr><td>Move an axis a specified distance from where it is now</td><td>Incremental</td><td>1</td></tr>
<tr><td>Move a Rotary axis to an absolute position in the shortest direction regardless of its current position</td><td>Rotary Shortest Path</td><td>2</td></tr>
<tr><td>Move a Rotary axis to an absolute position in the positive direction regardless of its current position</td><td>Rotary Positive</td><td>3</td></tr>
<tr><td>Move a Rotary axis to an absolute position in the negative direction regardless of its current position</td><td>Rotary Negative</td><td>4</td></tr>
<tr><td>Off set the master value of a position cam to an absolute position</td><td>Absolute Master Offset</td><td>5</td></tr>
<tr><td>Off set the master value of a position cam by an incremental distance</td><td>Incremental Master Offset</td><td>6</td></tr>
</table>
See Choose a Move Type for a Rotary Axis on page 3-82 for more information about rotary moves.</td></tr>
<tr><td>Position</td><td>REAL</td><td>Immediate
Tag</td><td>Absolute position or incremental distance for the move
<table>
<tr><th>For this Move Type</th><th>Enter this Position value</th></tr>
<tr><td>Absolute</td><td>Position to move to</td></tr>
<tr><td>Incremental</td><td>Distance to move</td></tr>
<tr><td>Rotary Shortest Path</td><td rowspan="3">Position to move to. Enter a positive value that is less than the Position Unwind value.</td></tr>
<tr><td>Rotary Positive</td></tr>
<tr><td>Rotary Negative</td></tr>
<tr><td>Absolute Master Offset</td><td>Absolute offset position</td></tr>
<tr><td>Incremental Master Offset</td><td>Incremental offset distance</td></tr>
</table></td></tr>
<tr><td>Speed</td><td>REAL</td><td>Immediate
Tag</td><td>Speed to move the axis in Speed Units.</td></tr>
<tr><td>Speed Units</td><td>DINT</td><td>Immediate</td><td>Which units do you want to use for the Speed?
• Units per sec (0)
• % of Maximum (1)</td></tr>
<tr><td>Accel Rate</td><td>REAL</td><td>Immediate
Tag</td><td>Acceleration rate of the axis in Accel Units</td></tr>
<tr><td>Accel Units</td><td>DINT</td><td>Immediate</td><td>Which units do you want to use for the Accel Rate?
• Units per sec^2 (0)
• % of Maximum (1)</td></tr>
</table>

Operand	Type	Format	Description
Decel Rate	REAL	Immediate Tag	Deceleration rate of the axis in Deceleration Units.
Decel Units	DINT	Immediate	Which units do you want to use for the Decel Rate? • Units per sec^2 (0) • % of Maximum (1)
Profile	DINT	Immediate	Select the velocity profile to run for the move: • Trapezoidal (0) • S-Curve (1) For more information, see Program a Velocity Profile on page 1-22.
Accel Jerk	REAL	Immediate Tag	The instruction only uses the jerk operands if the Profile is S-curve. You must always fill them in however. • Accel Jerk is the acceleration jerk rate for the axis. • Decel Jerk is the deceleration jerk rate for the axis. Use these values to get started. • Accel Jerk = 100 • Decel Jerk = 100 • Jerk Units = % of Time (2) You can also enter the jerk rates in these Jerk Units. • Units per sec^3 (0) • % of Maximum (1)
Decel Jerk	REAL	Immediate Tag	
Jerk Units	DINT	Immediate	
Merge	DINT	Immediate	Do you want to turn all current axis motion into a pure move governed by this instruction regardless of the motion instructions currently in process? • NO — Choose Disabled (0) • YES — Choose Enabled (1)
Merge Speed	DINT	Immediate	If Merge is Enabled, which speed do you want to move at? • Speed of this instruction — Choose Programmed (0) • Current speed of the axis — Choose Current (1)

Structured Text

```
MAM(Axis, MotionControl,
MoveType, Position, Speed,
SpeedUnits, AccelRate,
AccelUnits, DecelRate,
DecelUnits, Profile,
AccelJerk, DecelJerk,
JerkUnits, Merge,
MergeSpeed);
```

The operands are the same as the relay ladder instruction.

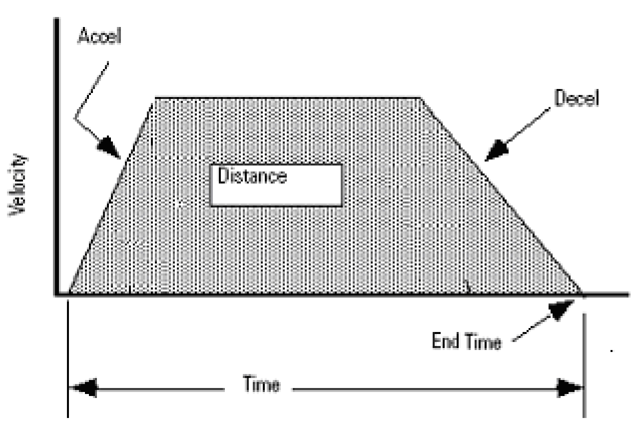
This operand	**Has these options which you**	
	enter as text	**or enter as a number**
SpeedUnits	unitspersec %ofmaximum	0 1
AccelUnits	unitspersec2 %ofmaximum	0 1
DecelUnits	unitspersec2 %ofmaximum	0 1
Profile	trapezoidal scurve	0 1
JerkUnits	unitspersec3 %ofmaximum %oftime	0 1 2
Merge	disabled enabled	0 1
MergeSpeed	programmed current	0 1

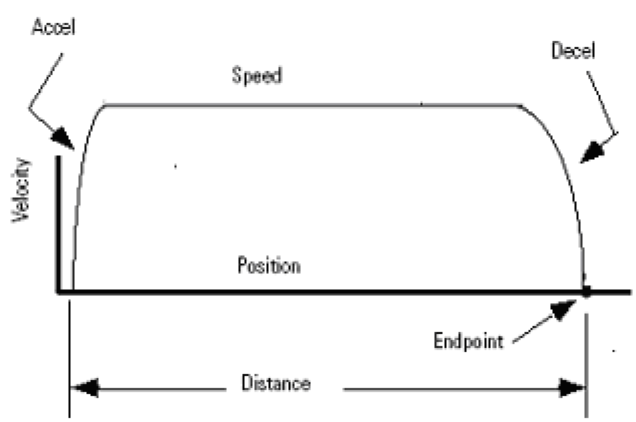
MOTION_INSTRUCTION Data Type

To See If	Check If This Bit is Set	Data Type	Notes
A false-to-true transition caused the instruction to execute	EN	BOOL	The EN bit stays set until the process is complete and the rung goes false.
The move was successfully initiated	DN	BOOL	
An error happened	ER	BOOL	
The axis is moving to the end Position	IP	BOOL	Any of these actions stop this move and clear the IP bit: • The axis gets to the end Position • Another MAM instruction supersedes this MAM instruction • MAS instruction • Merge from another instruction • Shutdown command • Fault Action
The axis is at the end Position	PC	BOOL	• The PC bit stays set until the rung makes a false-to-true transition. • The PC bit stays cleared if some other action stops the move before the axis gets to the end Positon.

Description: The Motion Axis Move (MAM) instruction moves an axis to either a specified absolute position or by a specified incremental distance. The MAM instruction can also produce other special types of moves.

Example: Trapezoidal move starting from standstill





Programming Guidelines

ATTENTION

If You Use An S-curve Profile

Be careful if you change the speed, acceleration, deceleration, or jerk while an axis is accelerating or decelerating along an S-curve profile. You can cause an axis to **overshoot its speed or reverse direction**.

For more information, see Troubleshoot Axis Motion on page 9-367.

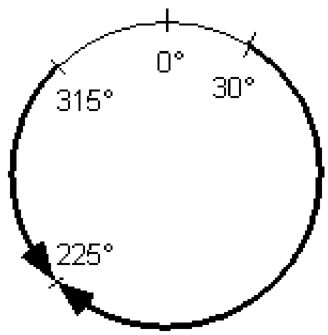
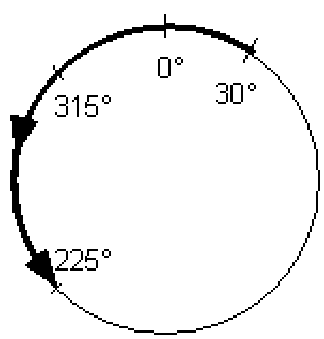
Guideline	Details
• **In relay ladder, toggle the rung condition each time you want to execute the instruction.**	This is a transitional instruction: • In relay ladder, toggle the rung-condition-in from cleared to set each time you want to execute the instruction.

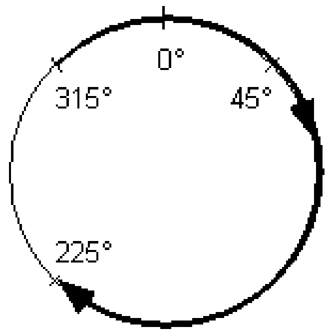
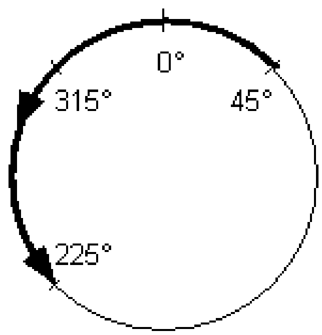
• **In structured text, condition the instruction so that it only executes on a transition.**	In structured text, instructions execute each time they are scanned.
	• Condition the instruction so that it only executes on a transition. Use either of these methods:
	  • qualifier of an SFC action
	  • structured text construct
	For more information, see Appendix C.

• **For a Master Offset move, enter the slave axis but use master units.**	Use an Absolute or Incremental Master Offset move to off set the master value of a position cam without actually changing the position of the master axis. This shifts the position cam profile along the master axis. • For Axis, enter the slave axis. • For Position, enter the absolute offset position or incremental offset distance • For Speed, Acceleration, Deceleration, and Jerk, enter them for the master axis. The instruction adds in the offset at the Speed, Acceleration, Deceleration, and Jerk values.
• **Use % of Time for the easiest programming and tuning of jerk.**	For an easy way to program and tune jerk, enter it as a % of the acceleration or deceleration time. For more information, see: • Program a Velocity Profile on page 1-22 • Tune an S-curve Profile on page 8-363.

<table>
<tr><th>Guideline</th><th>Details</th></tr>
<tr><td>• Use Merge to cancel the motion of other instructions.</td><td>
How you want to handle any motion that's already in process.?
<table>
<tr><th>If you want to</th><th>And you want to</th><th>Then set</th></tr>
<tr><td>Add the move to any motion already in process</td><td>→</td><td>Merge = Disabled
Merge Speed = Programmed
The instruction ignores Merge Speed but you must fill it in anyway.</td></tr>
<tr><td rowspan="2">End the motion from other instructions and just move</td><td>Move at the Speed that you set in this instruction</td><td>Merge = Enabled
Merge Speed = Programmed</td></tr>
<tr><td>Move at the speed that the axis is already moving at</td><td>Merge = Enabled
Merge Speed = Current
The instruction ignores the value that you put in the Speed operand.</td></tr>
</table>
Is This an Absolute or Incremental Master Offset Move?
If this is an Absolute or Incremental Master Offset move and Merge is Enabled, then:
• The move only ends an Absolute or Incremental Master Offset move that's already in process.
• The move doesn't affect any other motion that's already in process.
</td></tr>
<tr><td>• Use a second MAM instruction to change one that's already in process.</td><td>
You can change the position, speed, acceleration, or deceleration. The change immediately takes effect.
<table>
<tr><th>To change the position of an</th><th>Set up a second MAM instruction like this.</th></tr>
<tr><td>Absolute move</td><td>Either:
• Set the Move Type to Absolute and the Position to the new position.
• Set the Move Type to Incremental and set the Position to the distance to change the end position. The new end position is the old end position plus the new incremental distance.
In either case, the axis moves to the new position without stopping at the old position—including any required change of direction.</td></tr>
<tr><td>Incremental move</td><td>Either:
• Set the Move Type to Absolute and the Position to the new position. The axis goes directly to the new position without completing the incremental move.
• Set the Move Type to Incremental and set the Position to the additional distance. The axis moves the total of both incremental moves.</td></tr>
</table>
</td></tr>
<tr><td>• Combine a move with gearing for complex profiles and synchronization.</td><td>You can use a Motion Axis Gear (MAG) instruction together with an MAM instruction. This superimposes the gearing on top of the move or the move on top of the gearing.
Example: Superimpose an incremental move on top of electronic gearing for phase advance and retard control.</td></tr>
</table>

Choose a Move Type for a Rotary Axis

Move Type	Example	Description
Absolute	Absolute move to 225°. The direction depends on the starting position of the axis. 	With an Absolute move, the direction of travel depends on the current position of the axis and isn't necessarily the shortest path to the end position. Starting positions less than the end position result in motion in the positive direction, while starting positions greater than the end position result in motion in the negative direction. The specified position is interpreted trigonometrically and can be positive or negative. It can also be greater than the Position Unwind value. Negative position values are equivalent to their corresponding positive values and are useful when rotating the axis through 0. For example, –90° is the same as +270°. When the position is greater than the Position Unwind value, the axis moves through more than one revolution before stopping at an absolute position.
Incremental		The specified distance is interpreted trigonometrically and can be positive or negative. It can also be greater than the Position Unwind value. When the distance is greater than the Position Unwind value, the axis moves through more than one revolution before stopping.
Rotary Shortest Path	Rotary Shortest Path move from 30° to 225°. 	**Important**: Only use a Rotary Shortest Path move if the Positioning Mode of the axis is Rotary (Rotary axis). A Rotary Shortest Path move is a special type of absolute move for a Rotary axes. The axis: • moves to the specified Position in the **shortest** direction regardless of its current position • moves through 0° if needed. With a Rotary Shortest Path move, you: • can start the move while the axis is moving or standing still • **can't** move the axis more than one revolution with a single move.

Move Type	Example	Description
Rotary Positive	Rotary Positive move from 315° to 225°.  	**Important**: Only use a Rotary Positive move while the axis is standing still and not moving. Otherwise the axis could move in the wrong direction. A Rotary Positive move is a special type of absolute move for a Rotary axes. The axis: • moves to the specified Position in the **positive** direction regardless of its current position • moves through 0° if needed You can't move the axis more than one revolution with a single Rotary Shortest Path move.
Rotary Negative	Rotary Negative move from 45° to 225°.  	**Important**: Only use a Rotary Shortest Path move if • The Positioning Mode of the axis is Rotary (Rotary axis). • The axis is standing still and not moving. Otherwise the axis could move in the wrong direction. A Rotary Negative move is a special type of absolute move for a Rotary axes. The axis: • moves to the specified Position in the **negative** direction regardless of its current position • moves through 0° if needed You can't move the axis more than one revolution with a single Rotary Shortest Path move.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes: See Error Codes (ERR) for Motion Instructions on page A-383.

Extended Error Codes: Use Extended Error Codes (EXERR) for more information about an error.

If ERR is	And EXERR is	Then	
		Cause	**Corrective Action**
13	Varies	An operand is outside its range.	The EXERR is the number of the operand that is out of range. The first operand is 0. For example, if EXERR = 4, then check the Speed. **EXERR** / **Operand** 0 / Axis 1 / Motion Control 2 / Move Type 3 / Position 4 / Speed
54	-1	The coordinate system has a Maximum Deceleration of 0.	Go to the Properties for the coordinate system and set a Maximum Deceleration.
	0 or more	An axis in the coordinate system has a Maximum Deceleration of 0.	1. Open the Properties for the coordinate system. 2. Use the EXERR value to see which axis has the Maximum Deceleration of 0. 3. Set the Maximum Deceleration for the axis.

Changes to Status Bits: *Motion Status Bits*

If the Move Type is	And Merge is	Then the instruction changes these bits		
		Bit Name	**State**	**Meaning**
NOT Absolute Master Offset or Incremental Master Offset	Disabled	MoveStatus	TRUE	Axis is Moving
	Enabled	MoveStatus	TRUE	Axis is Moving
		JogStatus	FALSE	Axis is no longer Jogging
		GearingStatus	FALSE	Axis is no longer Gearing
Absolute Master Offset or Incremental Master Offset	→	MasterOffsetMoveStatus	TRUE	Axis is Offset

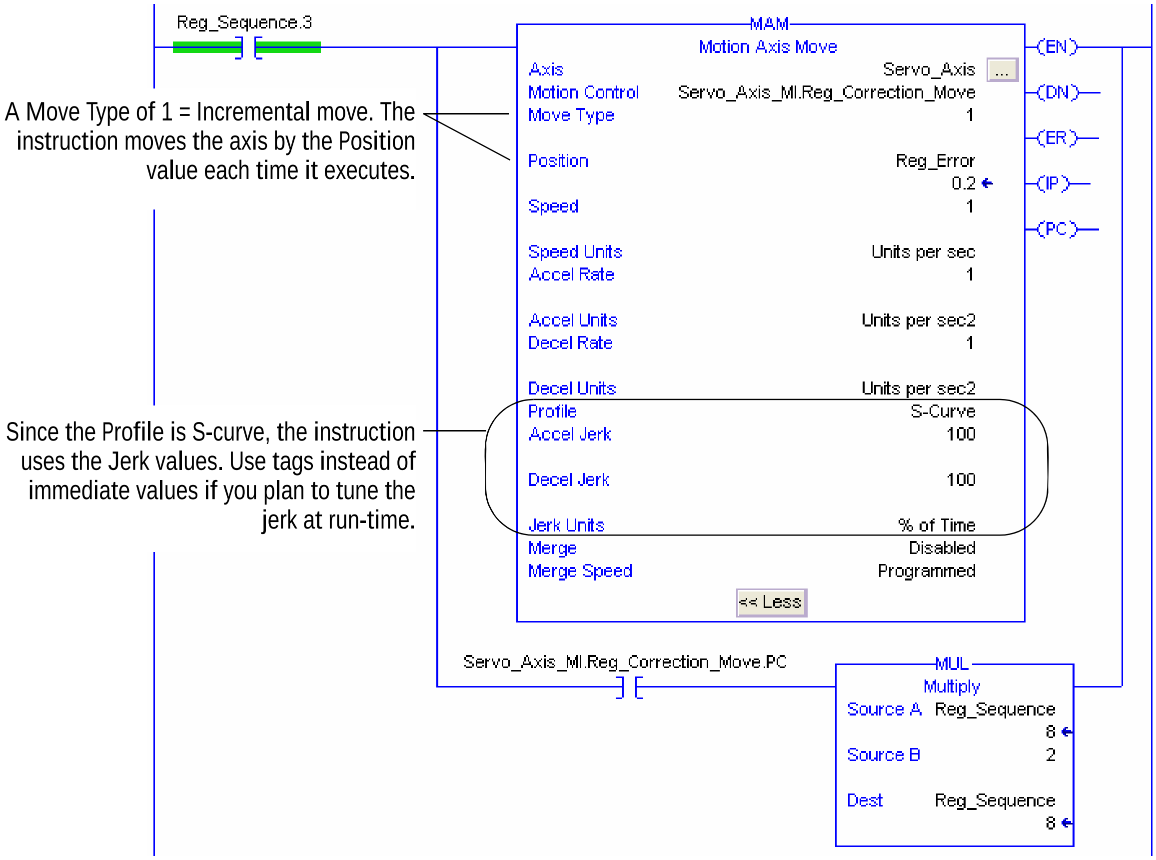
Example *Move—Relay Ladder*

This example uses the bit pattern of *Reg_Sequence* to step through the logic.

When *Reg_Sequence.3* turns on

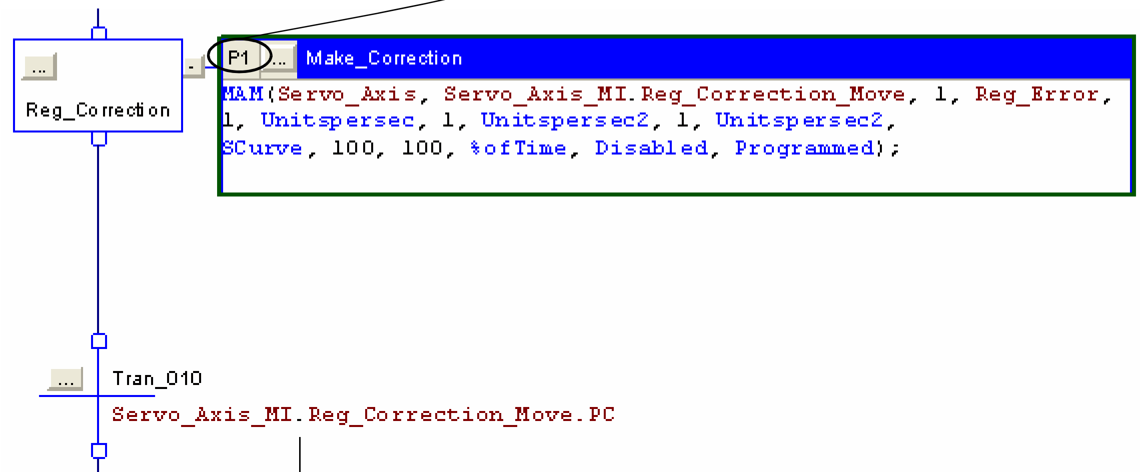
Move *Servo_Axis* the distance of *Reg_Error.*

When the move is complete, multiply *Reg_Sequence* by 2. This turns off bit 3 of *Reg_Sequence* and turns on bit 4.



Move—Structured Text

When the SFC starts the *Reg_Correction* step, move *Servo_Axis* the distance of *Reg_Error*. The P1 qualifier limits this to the first scan of the step.



The PC bit for the move turns on when the move is complete. The SFC leaves the step when the PC bit turns on.

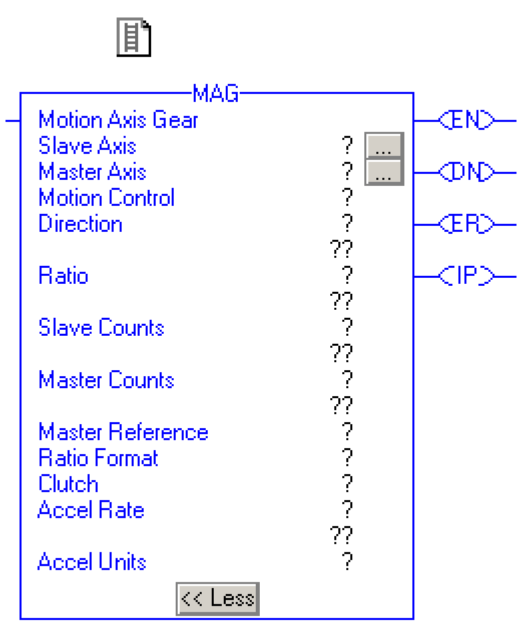
Motion Axis Gear (MAG)

The Motion Axis Gear (MAG) instruction provides electronic gearing between any two axes in a specified direction and at a specified ratio. When called, the specified Slave Axis is geared to the Master Axis at the specified Ratio (e.g., 1.345) or Slave Counts to Master Counts (e.g., 1:3). The MAG instruction supports specification of the gear ratio in one of two different formats, Real or Fractional, as determined by the Ratio Format input selection. The direction of Slave Axis motion relative to the Master Axis is defined by a very flexible Direction input parameter. The gearing direction may be explicitly set as the Same or Opposite or set relative to the current gearing direction as Reverse or Unchanged. Note, also, that the value for Ratio is sign sensitive. The Master Reference selection allows gearing input to be derived from either the Actual or Command position of the Master Axis. When the instruction's Clutch capability is activated the gearing instruction commands the slave axis to accelerate or decelerate at a controlled rate before Locking on to the master axis using the instructions Acceleration value much like the clutch of a car.

ATTENTION

Use a motion control tag only once. Do not re-use it in another instruction. Otherwise, you can cause unexpected equipment motion and injure people.

Operands: **Relay Ladder**



Operand	Type	Format	Description
Slave axis	AXIS_VIRTUAL AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis to perform operation on.
Master axis	AXIS_FEEDBACK AXIS_CONSUMED AXIS_VIRTUAL AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	The axis that the slave axis follows.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.

Operand	Type	Format	Description
Direction	UINT32	immediate or tag	The relative direction that the Slave axis tracks the Master Axis. Select one of following: 0 = slave axis moves in the same direction as the master axis 1 = slave axis moves in the opposite direction of its current direction 2 = slave axis reverses from current or previous 3 = slave axis to continue its current or previous direction
Ratio	REAL	immediate or tag	Signed Real value establishing the gear ratio in Slave User Units per Master User Unit.
Slave counts	UINT32	immediate or tag	Integer value representing slave counts used in specifying a Fractional gear ratio.
Master counts	UINT32	immediate or tag	Integer value representing master counts used in specifying a Fractional gear ratio.
Master reference	BOOLEAN	immediate	Sets the master position reference to either Command position or Actual position. 0 = Actual – slave axis motion is generated from the current position of the master axis as measured by its encoder or other feedback device. 1 = Command – slave axis motion is generated from the desired or commanded position of the master axis.
Ratio format	BOOLEAN	immediate	The desired ratio specification format. Select either: 0 = real gear ratio 1 = integer fraction of slave encoder counts to master encoder counts

Operand	Type	Format	Description
Clutch	BOOLEAN	immediate	When Clutch is enabled, motion control ramps the slave axis up to gearing speed at the instruction's defined Acceleration value. If not enabled, the Slave axis immediately locks onto the Master axis. If the Master Axis is currently moving this condition results in an abrupt uncontrolled acceleration event of the Slave Axis which can cause the axis to fault. Select either: 0 = enabled 1 = disabled
Accel rate	BOOLEAN	immediate or tag	Acceleration rate of the Slave Axis in % or Acceleration Units. It is applied when the Clutch feature is enabled.
Accel units	DINT	immediate	The units used to display the Acceleration value. Select either: 0 = units per sec^2 1 = % of maximum acceleration

Structured Text

MAG(SlaveAxis,MasterAxis, MotionControl,Direction, Ratio,SlaveCounts, MasterCounts, MasterReference,RatioFormat,Cl utch,AccelRate,

The operands are the same as those for the relay ladder MAG instruction.

For the operands that require you to select from available options, enter your selection as:

This operand	Has these options which you...	
	enter as text	or enter as a number
MasterReference	actual	0
	command	1
RatioFormat	real	0
	fraction_slave_master_counts	1
Clutch	enabled	0
	disabled	1
AccelUnits	unitspersec2	0
	%ofmaximum	1

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set when axis gear has been successfully initiated.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.
.IP (In Process) Bit 26	It is set on positive rung transition and cleared if either superseded by another Motion Gear Axes command, or terminated by a stop command, merge, shutdown, or servo fault.

Description: The Motion Axis Gear (MAG) instruction enables electronic gearing between two axes at a specified ratio. Electronic gearing allows any physical axis to be synchronized to the actual or command position of another physical axis at a precise ratio. It provides a direct edge-to-edge lock between the two axes—no maximum velocity, acceleration, or deceleration limits are used. The speed, acceleration, and deceleration of the slave axis is completely determined by the motion of the master axis and the specified gear ratio.

ATTENTION

The maximum velocity, acceleration, or deceleration limits established during axis configuration do not apply to electronic gearing.

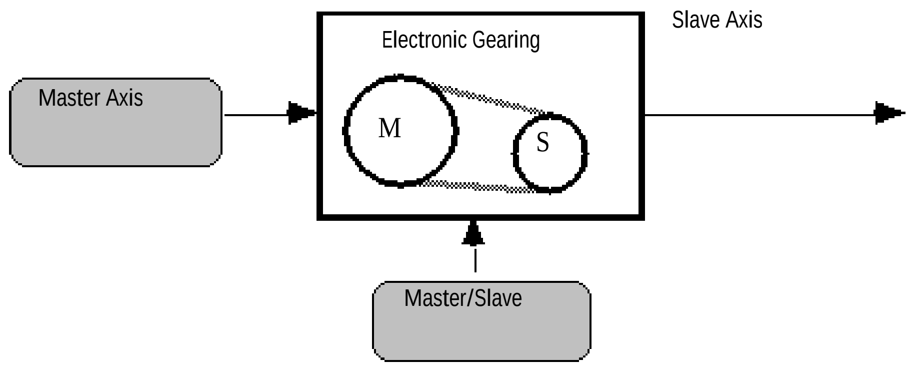
Select or enter the desired Master Axis, Slave Axis, and Direction and enter a value or tag variable for the desired ratio. If an axis is dimmed (gray) or not shown in the Slave Axis pop-up menu, the physical axis is not defined for Servo operation.

If the targeted axis does not appear in the list of available axes, the axis has not been configured for servo operation. Use the Tag Editor to create and configure a new axis.

Electronic gearing remains active through any subsequent execution of jog, or move processes for the slave axis. This allows electronic gearing motions to be superimposed with jog, or move profiles to create complex motion and synchronization.

Slaving to the Actual Position

When Actual Position is entered or selected as the Master Reference source, the slave axis motion is generated from the actual position of the master axis as shown below.

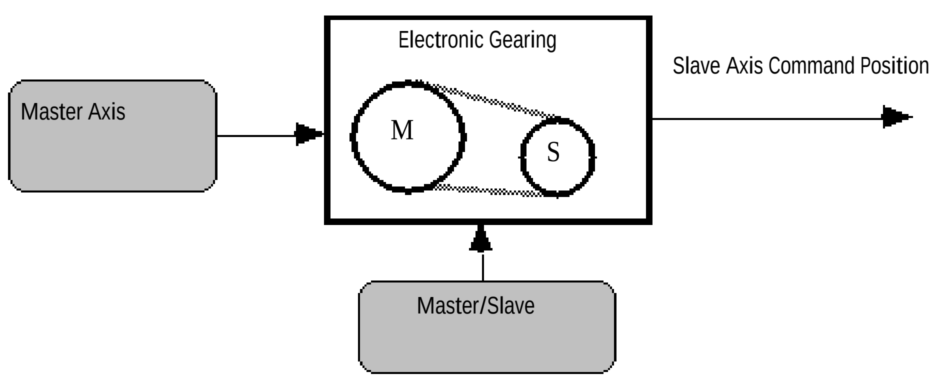


Slaving to Actual Position

Actual position is the current position of a physical axis as measured by the axis encoder. This is the *only* valid selection when the master axis' Axis Type is configured as Feedback Only.

Slaving to the Command Position

When Command Position is entered or selected as the Master Reference source, the slave axis motion is generated from the command position of the master axis as shown below.



Slaving to Command Position

Command position (only valid when the master axis' Axis Type is configured as Servo) is the current desired or commanded position for the master axis.

Since the command position does not incorporate any associated following error, external position disturbances, or quantization noise, it is a more accurate and stable reference for gearing. When gearing to the command position of the master, the master axis must be *commanded* to move to cause any motion on the slave axis. Refer to the Motion Axis Object Specification for more information on Command Position and Actual Position axis parameters.

Gearing in the Same Direction

When Same is selected or entered as the Direction, the slave axis moves in its *positive* direction at the specified gear ratio when the master axis moves in its *positive* direction and vice-versa.

Gearing in the Opposite Direction

When Opposite is selected or entered as the Direction, the slave axis moves in its *negative* direction at the specified gear ratio when the master axis moves in its *positive* direction and vice-versa.

Changing the Gear Ratio

When Unchanged is selected or entered as the Direction, the gear ratio may be changed while preserving the current gearing direction (same or opposite). This is useful when the current direction is not known or not important.

Reversing the Gearing Direction

When Reverse is selected or entered as the Direction, the current direction of the electronic gearing is changed from same to opposite or from opposite to same. This is very useful for winding applications where the gear ratio must be reversed at each end of the wind.

Real Number Gear Ratios

When Ratio Format is selected or entered as Real, the gear ratio is specified as a real number or tag variable with a value between 0.00001 and 9.99999 (inclusive) representing the desired ratio of slave axis position units to master axis position units. A gear ratio expressed this way is easy to interpret since it is defined in the axes' configured position units.

Fraction Gear Ratios

When Ratio Format is selected or entered as Fraction, the gear ratio is specified as a pair of integer numbers or tag variables representing the ratio between the number of slave axis feedback counts and the number of master axis feedback counts. Up to five digits (99999) can be used for the slave counts and up to nine digits (999999999) for the master counts. See The Tag variable Builder earlier in this manual for information on tag variables.

IMPORTANT	The Conversion Constant entered as part of the axis configuration procedure is **not** used when the Ratio Format for the MAG instruction is specified as a Fraction.

If your gear ratio cannot be exactly expressed as a real number with a maximum of five digits to the right of the decimal point, use Fraction as the Ratio Format.

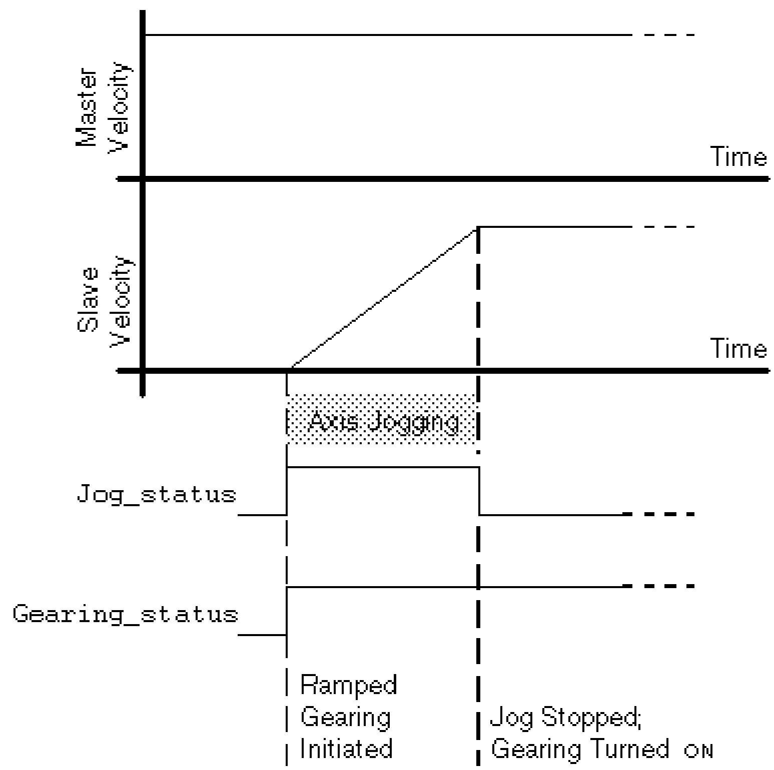
Specifying the gear ratio as a fraction allows the direct implementation of irrational gear ratios (such as $^1/_3$) with no accumulated positioning errors or round off. Since the master and slave count values do not use the axis conversion constants and because they are integers, the actual gear ratio relationship between the slave and master axes exactly match the specified ratio.

For example, the irrational gear ratio of $^1/_3$ can be equivalently specified as 1 slave count to 3 master counts, 10 slave counts to 30 master counts, 3 slave counts to 9 master counts, etc.

Clutch

When the Clutch check box is checked, the slave axis accelerates or decelerates to the speed that it would be moving if it were currently geared to the selected master axis at the specified gear ratio and direction using a trapezoidal velocity profile (linear acceleration or deceleration). Once the slave axis has reached the gearing speed, electronic gearing is automatically activated according to the other selections. Enter the desired Accel Rate as a percentage of the current configured maximum acceleration value or directly in the configured user units for acceleration.

This clutch function works much like the clutch in a car, allowing the slave axis to be smoothly engaged to the master axis as shown below.



Clutch Function

Using the clutch feature avoids the uncontrolled acceleration or deceleration that results when electronic gearing is enabled while the master axis is moving. The clutch feature can also be used to merge gear ratio changes on-the-fly, even changes in direction. The motion controller automatically ramps the slave axis to the speed implied by the master axis at the new ratio and/or direction.

The operation of the clutch ramp generator has no affect on jog or move processes that might be in progress on the slave axis.

Changing Master Axes

The master axis for electronic gearing can be changed at any time, even while gearing is currently enabled. However, since it is possible to have electronic gearing enabled on more than one axis at a time, if a Servo master axis and slave axis are reversed, the axes become cross-coupled and unexpected motion may result.

For example, if you are gearing Axis 0 to Axis 1 (defined as a Servo axis) and then want to change to gearing Axis 1 to Axis 0, you must first disable gearing on Axis 0 (see Disable Gearing later in this section). This is because specifying Axis 1 as the slave axis with Axis 0 as the master axis does *not* automatically disable Axis 0 from being a slave axis with Axis 1 as the master axis.

Moving While Gearing

An incremental MAM instruction may be used for the slave axis (or master axis if the Axis Type is configured as Servo) while the electronic gearing is enabled. This is particularly useful to accomplish phase advance/retard control. The incremental move distance can be used to eliminate any phase error between the master and the slave, or to create an exact non-zero phase relationship. Incremental MAM instruction may also be used in conjunction with electronic gearing to compensate for material slip.

Normally a gear ratio of 1 is used with phase adjustment. A 1:1 ratio ensures that the computed phase error does not change before performing the move to correct it. Electronic gearing is not normally used with absolute moves, since the ultimate endpoint is not predictable.

To successfully execute a Motion Axis Gear instruction, the targeted axis must be configured as a Servo Axis Type and the axis must be in the Servo On state. If any of these conditions are not met than the instruction errs.

IMPORTANT	The MAG instruction execution completes in a single scan, thus the Done (.DN) bit and the In Process (.IP) bit are set and the Process Complete (.PC) bit is cleared immediately. The In Process (.IP) bit remains set until the initiated Gear process is superseded by another MAG instruction, or terminated by a Motion Axis Stop command, Merge operation, or Servo Fault Action.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes: See Error Codes (ERR) for Motion Instructions on page A-383.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions.

Extended Error Codes for Axis Not Configured (11) error code are as follows:

- Extended Error Code 1 signifies that the Slave Axis is not configured.
- Extended Error Code 2 signifies that the Master Axis is not configured.

Extended Error codes for the Parameter Out of Range (13) error code lists a number that refers to the number of the operand as they are listed in the faceplate from top to bottom with the first operand being counted as zero. Therefore for the MAG instruction, an extended error code of 4 would refer to the Ratio operand's value. You would then have to check your value with the accepted range of values for the instruction.

For the Error Code 54 – Maximum Deceleration Value is Zero, if the Extended Error returns a positive number (0-*n*) it is referring to the offending axis in the coordinate system. Go to the Coordinate System Properties General Tab and look under the Brackets ([ ])column of the Axis Grid to determine which axis has a Maximum Deceleration value of 0. Click on the ellipsis button next to the offending axis to access the Axis Properties screen. Go to the Dynamics tab and make the appropriate change to the Maximum Deceleration Value. If the Extended Error number is -1, this means the Coordinate System has a Maximum Deceleration Value of 0. Go to the Coordinate System Properties Dynamics Tab to correct the Maximum Deceleration value.

Status Bits:

MAG Changes to Status Bits

If the Clutch check box is NOT checked, execution of the MAG instruction simply sets the Gear Status bit to True.

Bit Name	State	Meaning
GearingStatus	TRUE	Axis is Gearing

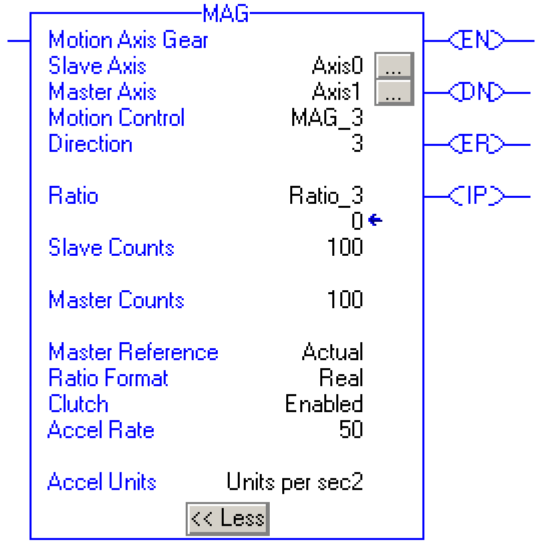
If the Clutch check box is checked, execution of the MAG instruction sets the Gearing Lock Status bit to True when the clutching process completes.

Bit Name	State	Meaning
Gearing Lock Status	TRUE	Axis has finished Clutch and locked in.
GearingStatus	TRUE	Axis is Gearing

Example:

When the input conditions are true, the controller provides electronic gearing between *axis2* and *axis1*.

Relay Ladder



MAG Ladder Example

Structured Text

MAG(Axis0,Axis1,MAG_3,3,Ratio_3,0,100,100,Actual,Real,
Enabled,50,Unitspersec2);

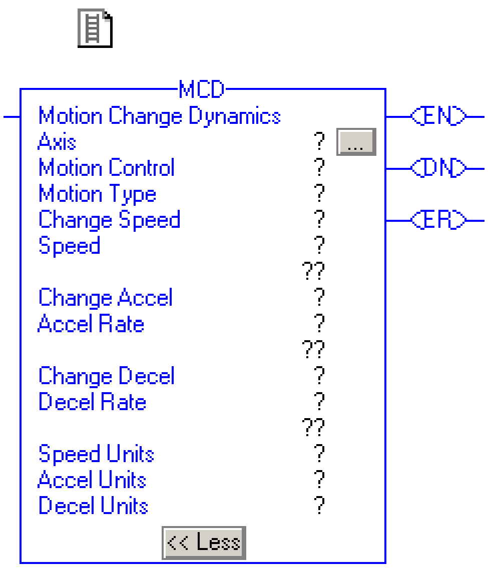
Motion Change Dynamics (MCD)

Use the MCD instruction to selectively change the speed, acceleration rate, or deceleration rate of a move profile or a jog profile in process.

ATTENTION

Use a motion control tag only once. Do not re-use it in another instruction. Otherwise, you can cause unexpected equipment motion and injure people.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Axis	AXIS_VIRTUAL AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.
Motion type	UDINT	immediate	Motion profile (jog or move) to change. Select either: 0 = jog 1 = move
Change speed	BOOLEAN	immediate	Set to enable a change of speed. Select either: 0 = no 1 = yes
Speed	REAL	immediate or tag	The new Speed to move the axis in % or Speed Units.
Change accel	BOOLEAN	immediate	Set to enable an acceleration change. Select either: 0 = no 1 = yes
Accel rate	REAL	immediate or tag	The acceleration rate of the axis in % or Acceleration units.
Change decel	BOOLEAN	immediate	Set to enable a deceleration change. Select either: 0 = no 1 = yes

Operand	Type	Format	Description
Decel rate	REAL	immediate or tag	The deceleration rate of the axis in % or Deceleration units. The axis could overshoot its target position if you reduce the deceleration while a move is in process.
Speed units	BOOLEAN	immediate	Units used to display the Speed value. Select either: 0 = units per sec 1 = % of maximum speed
Accel units	BOOLEAN	immediate	Units used to display the Acceleration value. Select either: 0 = units per sec^2 1 = % of maximum acceleration
Decel units	BOOLEAN	immediate	Units used to display the Deceleration value. Select either: 0 = units per sec^2 1 = % of maximum deceleration

Structured Text

```
MCD(Axis,MotionControl,
MotionType,ChangeSpeed,
Speed,ChangeAccel,AccelRate,C
hangeDecel,DecelRate,
SpeedUnits,AccelUnits,
DecelUnits);
```

The operands are the same as those for the relay ladder MCD instruction.

For the operands that require you to select from available options, enter your selection as:

This operand	Has these options which you...	
	enter as text	or enter as a number
MotionType	jog move	0 1
ChangeSpeed	no yes	0 1
ChangeAccel	no yes	0 1
ChangeDecel	no yes	0 1

This operand	Has these options which you...	
	enter as text	or enter as a number
SpeedUnits	unitspersec %ofmaximum	0 1
AccelUnits	unitspersec2 %ofmaximum	0 1
DecelUnits	unitspersec2 %ofmaximum	0 1

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) BIt 29	It is set when axis change dynamics has been successfully initiated. The instruction execution completes in a single scan, and the DN bit is set immediately.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.

Description: The MCD instruction changes the speed of trapezoidal profile moves on-the-fly and the speed, acceleration, and deceleration of trapezoidal profile jogs on-the-fly. Choose the desired physical axis and type of motion and enter values or tag variables for the Speed, Accel, and Decel. Speed, acceleration, and deceleration values can be entered as percentages of the current maximum configured value or directly in the configured speed or acceleration units of the axis.

If the targeted axis does not appear in the list of available axes, the axis has not been configured for servo operation. Use the Tag Editor to create and configure a new axis.

ATTENTION

If You Use An S-curve Profile

Be careful if you change the speed, acceleration, deceleration, or jerk while an axis is accelerating or decelerating along an S-curve profile. You can cause an axis to **overshoot its speed or reverse direction**.

For more information, see Troubleshoot Axis Motion on page 9-367.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Changing Move Dynamics

When a Motion type of Move is entered or chosen, the speed, acceleration, and/or deceleration of a Move in progress may be changed to the specified value. The speed change occurs at the specified acceleration rate if the new speed is higher than the current speed or at the specified deceleration rate if the new speed is lower than the current speed.

Pausing Moves

The MCD instruction may be used to temporarily pause a move in progress by changing its speed to zero. Use another MCD instruction with a non-zero speed value to complete the move as originally specified.

Changing Jog Dynamics

When a Motion type of Jog is entered or chosen, the speed, acceleration, and/or deceleration of a Jog in progress may be changed to the specified value. The speed change occurs at the specified acceleration rate if the new speed is higher than the current speed or at the specified deceleration rate if the new speed is lower than the current speed.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes: See Error Codes (ERR) for Motion Instructions on page A-383.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions.

Extended Error codes for the Parameter Out of Range (13) error code lists a number that refers to the number of the operand as they are listed in the faceplate from top to bottom with the first operand being counted as zero. Therefore for the MCD instruction, an extended error code of 4 would refer to the Speed operand's value. You would then have to check your value with the accepted range of values for the instruction.

For the Error Code 54 – Maximum Deceleration Value is Zero, if the Extended Error returns a positive number (0-*n*) it is referring to the offending axis in the coordinate system. Go to the Coordinate System Properties General Tab and look under the Brackets ([ ])column of the Axis Grid to determine which axis has a Maximum Deceleration value of 0. Click on the ellipsis button next to the offending axis to access the Axis Properties screen. Go to the Dynamics tab and make the appropriate change to the Maximum Deceleration Value. If the Extended Error number is -1, this means the Coordinate System has a Maximum Deceleration Value of 0. Go to the Coordinate System Properties Dynamics Tab to correct the Maximum Deceleration value.

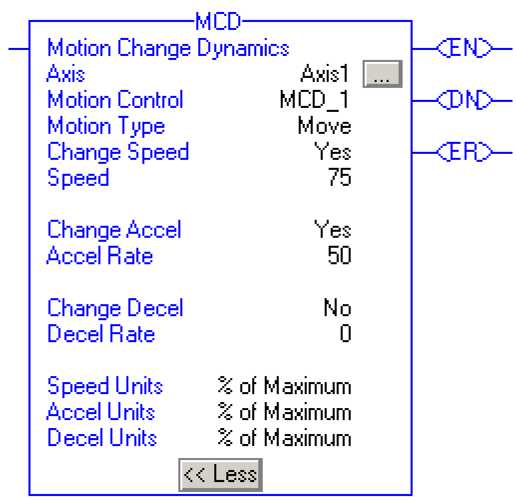
MCD Changes to Status Bits:

None

Example:

When the input conditions are true, the controller changes the speed, acceleration, or deceleration rate of a move profile or jog profile in progress for *axis1*.

Relay Ladder



MCD Ladder Example

Structured Text

MCD(Axis1,MCD_1,Move,Yes,75,Yes,50,No,0,%ofmaximum,
%ofmaximum,%ofmaximum);

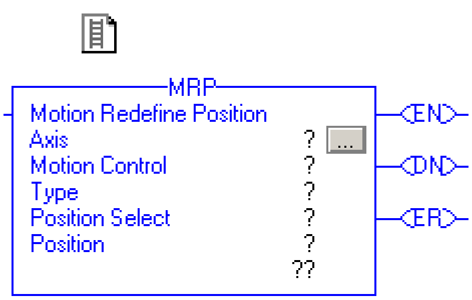
Motion Redefine Position (MRP)

Use the MRP instruction to change the command or actual position of an axis.The value specified by Position is used to update the Actual or Command position of Axis. The position redefinition can be calculated on an Absolute or Relative basis. If Absolute is selected the Position value is assigned to the current Actual or Command position. If Relative is selected the Position value is added as a displacement to the current Actual or Command position. The process of redefining the current axis position has no affect on motion in progress as the instruction preserves the current servo following error during the redefinition process. As a result, axis position can be redefined on-the-fly without disturbing axis motion.

ATTENTION

Use a motion control tag only once. Do not reuse it in another instruction. Otherwise, you can cause unexpected equipment motion and injure people.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Axis	AXIS_FEEDBACK AXIS_VIRTUAL AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction parameters.
Type	BOOLEAN	immediate	The way you want the redefinition operation to work. Select either: 0 = absolute 1 = relative
Position select	BOOLEAN	immediate	Choose what position to perform the redefinition operation on. Select either: 0 = actual position 1 = command position
Position	REAL	immediate or tag	The value to use to change the axis position to or offset to current position.

Structured Text

MRP(Axis,MotionControl,Type,
PositionSelect,Position);

The operands are the same as those for the relay ladder MRP instruction.

For the operands that require you to select from available options, enter your selection as:

This operand	Has these options which you...	
	enter as text	or enter as a number
Type	absolute	0
	relative	1
PositionSelect	actual	0
	command	1

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set when the axis' position action been successfully redefined.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.

Description:

The Motion Redefine Position (MRP) instruction directly sets the actual or command position of the specified axis to the specified absolute or relative position. No motion is caused by this instruction—the current axis position is simply redefined. Select or enter the desired Axis, Type, Position Selection, and enter a value or tag variable for the desired New Position.

If the targeted axis does not appear in the list of available axes, the axis has not been configured for operation. Use the Tag Editor to create and configure a new axis.

The MRP instruction may be used while the axis is moving as well as when it is at rest. MRP is used to redefine position "on-the-fly" for certain registration, slip compensation, and re-calibration applications.

Absolute Mode

When Absolute is selected or entered as the MRP Type, the New Position specifies the new *absolute* position of the axis. No motion occurs—the current axis position (actual or command) is simply redefined to be the specified new position.

If software overtravel limits are used (refer to Motion Axis Object specification for more information on software overtravel configuration), the new position must be between the Max Positive and Max Negative Travel configuration values. Otherwise a software overtravel fault is generated when the instruction is executed.

ATTENTION

If software overtravel limit checking is in effect, execution of an MRP in Absolute Mode may invalidate the current Max Positive and Max Negative Travel limits in the absolute sense. Exercise caution when redefining the absolute position of an axis that has travel limits.

Absolute and relative mode MRP instructions have the same effect when the axis is not moving. When the axis is moving, however, absolute mode introduces a position error equal to the motion of the axis during the time it takes to execute the MRP instruction and assign the new position. Relative mode does not introduce this error and guarantees an exact correction independent of axis speed or position.

Relative Mode

When Relative is selected or entered as the MRP Type, the New Position value is used to *offset* the current position of the axis. No motion occurs—the current axis position (actual or command) is simply redefined to be the current position *plus* the specified new position.

In relative mode, axis position is redefined in such a way that no position errors are introduced if the axis is moving. It is particularly useful for unwinding axis position under program control rather than using the built-in rotary axis feature.

Absolute and relative mode MRP instructions have the same effect when the axis is not moving. When the axis is moving, however, absolute mode introduces a position error equal to the motion of the axis during the time it takes to execute the MRP instruction and assign the new position. Relative mode does not introduce this error and guarantees an exact correction independent of axis speed or position.

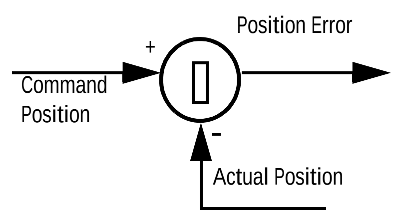
Actual Position

When Actual is selected or entered as the MRP Position Selection, the New Position is directly applied to the actual position of the physical axis. The command position of the axis is also adjusted along with the new actual position to preserve any position error which exists. This ensures that there is no unexpected motion of the axis when the positions are redefined. See the Motion Axis Object Specification for more discussion of command position, actual position, and position error.

Command Position

When Command is selected or entered as the MRP Position Selection, the New Position is directly applied to the command position of the servo or imaginary axis. Since Feedback Only axes do not have a command position, always choose Actual from the Position menu for Master Only axes. The actual position of servo axes is also adjusted along with the new command position to preserve any position error which exists. This ensures that there is no unexpected motion of the axis when the positions are redefined.

Command position is the desired or commanded position of a servo as generated by any previous motion instructions. Actual position is the current position of a physical or virtual axis as measured by the encoder or other feedback device. Position error is the difference between these two and is used to drive the motor to make the actual position equal to the command position. The Figure below shows the relationship of these three positions.



Position Relationship

See the Motion Axis Object specification for a more detailed overview of the Nested Digital Servo Loop used by the ControlLogix motion controllers.

To successfully execute a MRP instruction, the targeted axis must be configured as either a Servo or Feedback Only axis. Otherwise, the instruction errs.

IMPORTANT The MRP instruction execution may take multiple scans to execute due to the fact that it requires transmission of multiple messages to the motion module. Thus, the Done (.DN) bit is not set immediately, but only after these messages have been successfully transmitted.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes: See Error Codes (ERR) for Motion Instructions on page A-383.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions. The following Extended Error codes help to pinpoint the problem when the MRP instruction receives a Servo Message Failure (12) error message.

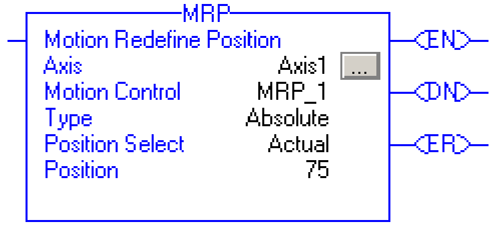
Associated Error Code (decimal)	Extended Error Code (decimal)	Meaning
SERVO_MESSAGE_FAILURE (12)	Device in wrong state (16)	Redefine Position, Home, and Registration 2 are mutually exclusive. (SERCOS).

Extended Error codes for the Parameter Out of Range (13) error code work a little differently. Rather than having a standard enumeration, the number that appears for the Extended Error code refers to the number of the operand as they are listed in the faceplate from top to bottom with the first operand being counted as zero. Therefore for the MRP instruction, an extended error code of 4 would refer to the Position operand's value. You would then have to check your value with the accepted range of values for the instruction.

MRP Changes to Status Bits: None

Example: When the input conditions are true, the controller changes the position of *axis1*.

Relay Ladder



MRP Ladder Example

Structured Text

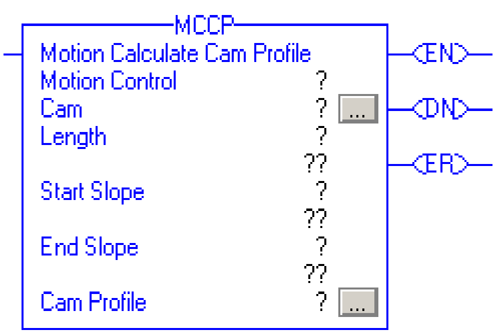
MRP(Axis1,MRP_1,Absolute,Actual,75);

Motion Calculate Cam Profile (MCCP)

The Motion Calculate Cam Profile (MCCP) instruction calculates a cam profile based on an array of cam points. An array of cam points may be established programmatically or by use of the RSLogix 5000 Cam Profile Editor. Each cam point in the cam array consists of a slave position value, a master position (position cam) or time (time cam) value, and an interpolation type (linear or cubic). The resulting cam profile may be used by an Motion Axis Position Cam (MAPC) or Motion Axis Time Cam (MATC) instruction to govern the motion of a slave axis according to master position or time.

ATTENTION	Use a motion control tag only once. Do not re-use it in another instruction. Otherwise, you can cause unexpected equipment motion and injure people.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access block status parameters.
Cam	CAM	array	Tag name of the cam array used to compute the cam profile. The numerical array index indicates the starting cam element in the array used in the cam profile calculation. Ellipsis launches Cam Profile Editor.
Length	UINT	immediate or tag	Determines the number of cam elements in the array used in the cam profile calculation.
Start Slope	REAL	immediate or tag	This is the boundary condition for the initial slope of the profile. It is valid only for a cubic first segment and is used to specify a slope through the first point.
End Slope	REAL	immediate or tag	This is the boundary condition for the ending slope of the profile. It is valid only for a cubic last segment and is used to specify a slope through the last point.
Cam Profile	CAM_PROFILE	array	Tag name of the calculated cam profile array used as input to MAPC and MATC instructions. Only the zero array element ([0]) is allowed for the Cam Profile array. Ellipsis launches Cam Profile Editor.

Structured Text

```
MCCP(MotionControl,Cam,
Length,StartSlope,EndSlope,
CamProfile);
```

The operands are the same as those for the relay ladder MCCP instruction. For the array operands, you do not have to include the array index. If you do not include the index, the instruction starts with the first element in the array ([0]).

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	The enable bit is set when the rung transitions from false-to-true and stays set until the done bit is set and the rung goes false.
.DN (Done) Bit 29	The done bit is set when the calculate cam instruction has been successfully executed and the Cam Profile array calculated.
.ER (Error) Bit 28	The error bit indicates when the instruction detects an error, such as if the cam array is of an illegal length.

Description: The Motion Calculate Cam Profile (MCCP) instruction computes a cam profile based on a given set of points in a specified cam array. The resultant cam profiles generated by this instruction may be used by subsequent MAPC or MATC camming instructions to provide complex motion of a slave axis with respect to either a master axis position or with respect to time.

Since cam profiles can be directly calculated by the RSLogix 5000 Cam Profile Editor, the main purpose of the MCCP instruction is to provide a method for calculating cam profiles in real-time based on programmatic changes to the corresponding cam arrays.

Specifying a Cam Array

In order to execute an MCCP instruction, a Cam array tag must be created using the RSLogix Tag Editor or the Cam Profile Editor. The figure below illustrates how the Cam array tags are established and used as input to the MCCP instruction.

The Cam array elements consist of slave (yp) and master (xp) point pairs as well as an interpolation type. Since there is no association with a specific axis position or time, the x and y point values are unitless. The interpolation type may be specified for each point as either “linear” or “cubic”.

Specifying the Cam Profile Tag

To execute a MAPC instruction, a Cam Profile array tag must also be created. Cam Profile array tags may be created by the RSLogix 5000 tag editor or the MAPC/MATC instructions using the built-in Cam Profile Editor.

The data within the Cam Profile array can be modified at compile time using the Cam Profile Editor, or at run-time with the Motion Calculate Cam Profile (MCCP) instruction. In the case of run-time changes, a Cam array must be created in order to use the MCCP instruction.

The status parameter is used to indicate that the Cam Profile array element has been calculated. If execution of a camming instruction is attempted using any uncalculated elements in a cam profile, the MAPC or MATC instructions error. The type parameter determines the type of interpolation applied between this cam array element and the next cam element.

Cam Profile Array Status Member

The Status member of the first element in the cam profile array is special and used for data integrity checks. For this reason, the MCCP must always specify the cam profile with the starting index set to 0. This first cam profile element Status member can have the following values:

Status Variables	Description
0	Cam profile element has not been calculated
1	Cam profile element is being calculated
2	Cam profile element has been calculated
n	Cam profile element has been calculated and is currently being used by (*n*-2) MAPC or MATC instructions

Linear and Cubic Spline Interpolation

The resultant calculated cam profiles are fully interpolated. This means that if the current master position or time does not correspond exactly with a point in the cam array used to generate the cam profile, the slave axis position is determined by linear or cubic interpolation between adjacent points. In this way, the smoothest possible slave motion is provided. The MCCP instruction accomplishes this by calculating coefficients to a polynomial equation that determines slave position as a function of master position or time.

Calculating the Cam Profile

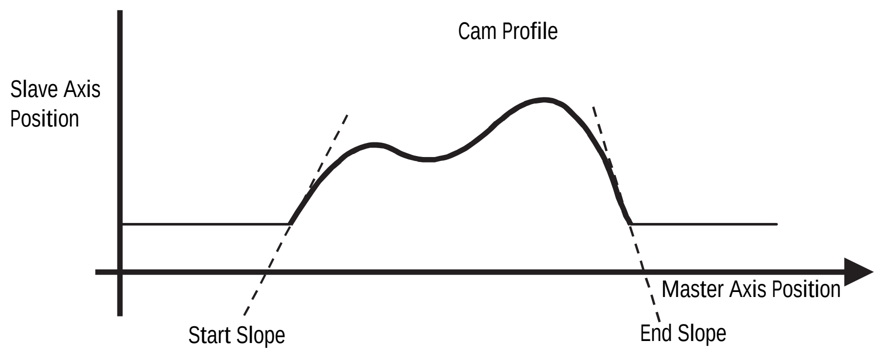
Before calculating a cam profile on a specified axis, the MCCP instructions first checks if the cam profile array has been calculated by checking the value of the first cam profile element's Status member. If the Status value is either 0 or 2, the MCCP proceeds with the calculation of the cam profile. When the cam profile array has been completely calculated, the MCCP instruction sets the first cam profile element's Status value to "being calculated", or 1, and then sets the

Status value of all other cam profile elements to “being calculated”. As the calculation proceeds, individual cam profile members’ Status values are set to “calculated”, or 2. When all elements in the cam profile array have been calculated, the first cam profile element’s Status value is also set to “calculated”.

However, if an MCCP instruction is executed with an initial cam profile Status value of 1, then the cam profile is currently being calculated by another MCCP instruction, and the MCCP instruction errors. If the Status value is >2, then the cam profile is being actively used by an MAPC or MATC instruction process, and the MCCP instruction errs.

Start Slope and End Slope

To facilitate a smooth entry into and exit from a cubic cam profile, slope control is provided. The Start Slope and End Slope parameters determine the initial rate of change of the slave relative to the master. These values are used in the cubic spline calculations performed on the cam array. The diagram below the master slave slope relationship.



Start and End Slope

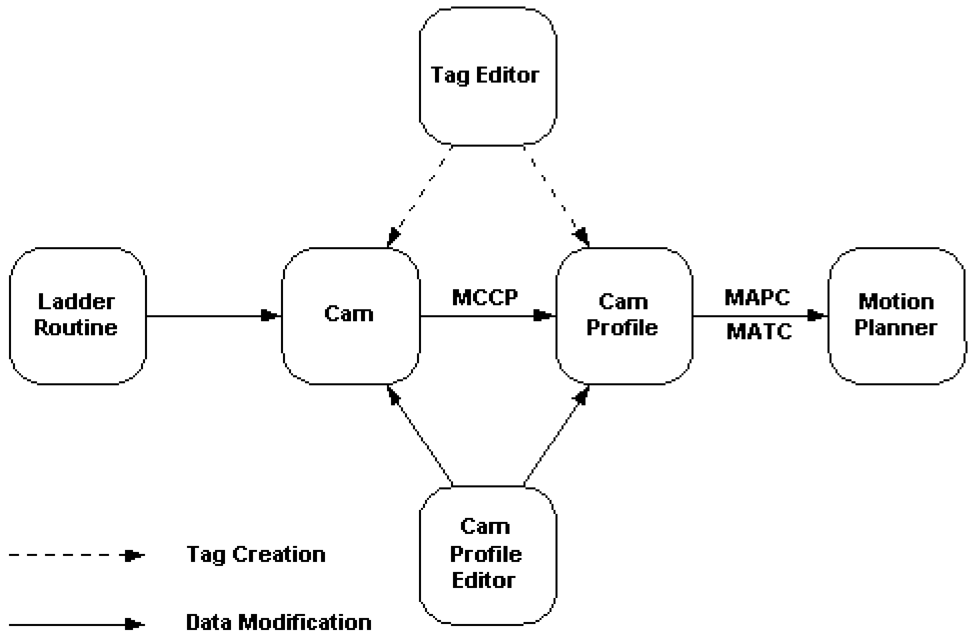
The default values for Start Slope and End Slope are 0 to facilitate a smooth start and end to the cam profile from rest. However, if the axis is already camming, an appropriate non-zero Start Slope can be specified to match the End Slope of the currently executing cam, to seamlessly blend the two cam profiles together.

The Start Slope and End Slope values are not applicable when starting or ending the cam profile with linear interpolation.

IMPORTANT	The MCCP instruction execution completes in a single scan. This instructions should therefore be placed in a separate task to avoid impacting user program scan time.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.



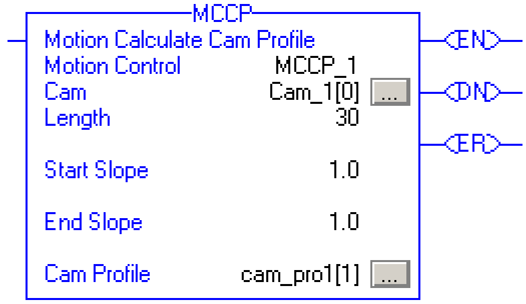
Cam Operation Diagram

Error Codes: See Error Codes (ERR) for Motion Instructions on page A-383.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are not specific enough to help pinpoint the problem. When the MCCP instruction receives an Illegal Cam Length (26) error message to let it know that the length input parameter does not correspond to what the instruction expects, the corresponding Extended Error code provides the number of cams in the Cam Tag provided to the instruction. When the MCCP instruction receives an Illegal Cam Profile Length (27) error message to let it know that the length input parameter does not correspond to what the instruction expects, the corresponding Extended Error code provides the number of cam points the instruction is attempting to generate.

MCCP Changes to Status Bits: None

Example: *Relay Ladder*



MCCP Ladder Example

Structured Text

MCCP(MCCP_1,Cam_1[0],30,1.0,1.0,cam_pro1[1]);

Motion Axis Position Cam (MAPC)

The Motion Axis Position Cam (MAPC) instruction provides electronic camming between any two axes according to the specified Cam Profile.

When executed, the specified Slave Axis is synchronized to the designated Master Axis using a position Cam Profile established by the RSLogix 5000 Cam Profile Editor, or by a previously executed Motion Calculate Cam Profile (MCCP) instruction. The direction of Slave Axis motion relative to the Master Axis is defined by a flexible Direction input parameter. The camming Direction, as applied to the slave, may be explicitly set as the Same or Opposite or set relative to the current camming direction as Reverse or Unchanged.

To accurately synchronize the slave axis position to master axis position, an Execution Schedule setting and an associated Master Lock Position can be specified for the master axis. When the master axis travels past the Master Lock Position in the direction specified by the Execution Schedule parameter, the slave axis is locked to the master axis position according to the specified Cam Profile beginning at the Cam Lock Position.

The cam profile can also be configured via the Execution Schedule parameter to execute Immediately or Pending completion of a currently executing position cam profile. The cam profile can also be executed Once or Continuously by specifying the desired Execution Mode.

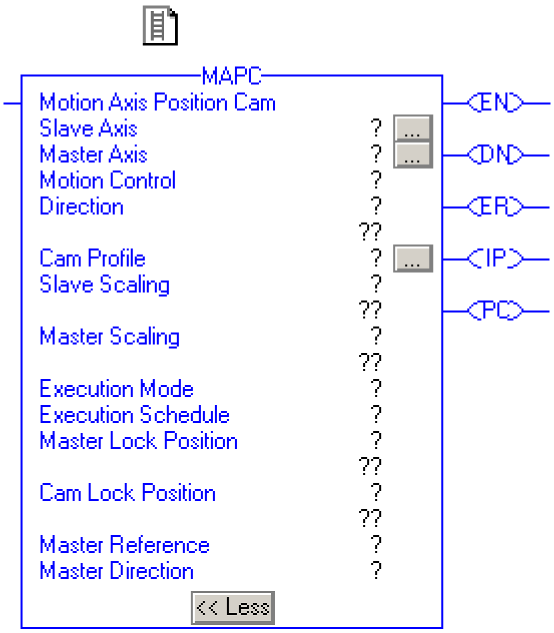
The Master Reference selection allows camming input from the master to be derived from either the Actual or Command position of the Master Axis. To support applications which require unidirectional motion, a “slip clutch” feature is available which prevents the slave from “backing-up” when the master axis reverses direction. This feature is controlled by the Master Direction parameter.

Master and Slave Scaling functionality can be used to scale slave motion based on a standard cam profile without having to create a new cam table and calculate a new cam profile.

ATTENTION

Use a motion control tag only once. Do not re-use it in another instruction. Otherwise, you can cause unexpected equipment motion and injure people.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Slave Axis	AXIS_VIRTUAL AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	The name of the axis that the cam profile is applied to. Ellipsis launches Axis Properties dialog.
Master Axis	AXIS_FEEDBACK AXIS_CONSUMED AXIS_VIRTUAL AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	The axis that the slave axis follows according to the cam profile. Ellipsis launches Axis Properties dialog. If Pending is selected as the Execution Schedule, then Master Axis is ignored.
Motion Control	MOTION_ INSTRUCTION	tag	Structure used to access block status parameters.
Direction	UINT32	immediate or tag	Relative direction of the slave axis to the master axis: • Same – the slave axis position values are in the same sense as the master's. • Opposite – the slave axis position values are in the opposite sense of the master's. Or relative to the current or previous camming direction: • Reverse – the current or previous direction of the position cam is reversed on execution. When executed for the first time with Reverse selected, the control defaults the direction to Opposite. • Unchanged – this allows other cam parameters to be changed without altering the current or previous camming direction. When executed for the first time with Unchanged selected, the control defaults the direction to Same.

Operand	Type	Format	Description
Cam Profile	CAM_PROFILE	array	Tag name of the calculated cam profile array used to establish the master/slave position relationship. Only the zero array element ([0]) is allowed for the Cam Profile array. Ellipsis launches Cam Profile Editor.
Slave Scaling	REAL	immediate or tag	Scales the total distance covered by the slave axis through the cam profile.
Master Scaling	REAL	immediate or tag	Scales the total distance covered by the master axis through the cam profile.
Execution Mode	UINT32	immediate	Determines if the cam profile is executed only one time or repeatedly: 0 = Once – cam motion of slave axis starts only when the master axis moves into the range defined by the start and end points of the cam profile. When the master axis moves beyond the defined range cam motion on the slave axis stops and the Process Complete bit is set. Slave motion does not resume if the master axis moves back into the cam profile range. 1 = Continuous – Once started the cam profile is executed indefinitely. This feature is useful in rotary applications where it is necessary that the cam position run continuously in a rotary or reciprocating fashion. 2 = Persistent – When the Master Axis moves beyond the defined range, cam motion on the Slave Axis stops and the PositionCamLockStatus bit is cleared. Slave motion does resume if the Master Axis moves back into the cam profile range and the PositionCamLockStatus bit is set.

Operand	Type	Format	Description
Execution Schedule	UINT32	immediate	Selects the method used to execute the cam profile. Options are: 0 = Immediate – The slave axis is immediately locked to the master axis and the position camming process begins. 1 = Pending – lets you blend a new position cam execution after an in process position cam is finished. When Pending is selected the following parameters are ignored: Master Axis, Master Lock Position, and Master Reference. 2 = Forward only – the cam profile starts when the master position crosses the Master Lock Position in the forward direction. 3 = Reverse only – the cam profile starts when the master position crosses the Master Lock Position in the reverse direction. 4 = Bi-directional – the cam profile starts when the master position crosses the Master Lock Position in either direction.
Master Lock Position	REAL	immediate or tag	The Master axis absolute position where the slave axis locks to the master axis. If Pending is selected as the Execution Schedule value, then Master Lock Position is ignored.

Operand	Type	Format	Description
Cam Lock Position	REAL	immediate or tag	This determines the starting location in the cam profile.
Master Reference	UINT32	immediate	Sets the master position reference to either Command position or Actual position. If Pending is selected for the Execution Schedule value, then Master Reference is ignored. 0 = Actual – slave axis motion is generated from the current position of the master axis as measured by its encoder or other feedback device. 1 = Command – slave axis motion is generated from the desired or commanded position of the master axis.
Master Direction	UINT32	immediate	This determines the direction of the master axis that generates slave motion according to the cam profile. Options are: 0 = Bi-directional – slave axis can track the master axis in either direction. 1 = Forward only – slave axis tracks the master axis in the forward direction of the master axis. 2 = Reverse only – slave axis tracks the master axis in the opposite direction of the master axis.

Structured Text

MAPC(SlaveAxis,MasterAxis,
MotionControl,Direction,
CamProfile,SlaveScaling,
MasterScaling,ExecutionMode,
ExecutionSchedule,
MasterLockPosition,
CamLockPosition,
MasterReference,

The operands are the same as those for the relay ladder MAPC instruction. For the array operands, you do not have to include the array index. If you do not include the index, the instruction starts with the first element in the array ([0]).

For the operands that require you to select from available options, enter your selection as:

This operand	Has these options which you...	
	enter as text	or enter as a number
ExecutionMode	once	0
	continuous	1
	persistent	2
ExecutionSchedule	immediate	0
	pending	1
	forwardonly	2
	reverseonly	3
	bidirectional	4
MasterReference	actual	0
	command	1
MasterDirection	bidirectional	0
	forwardonly	1
	reverseonly	2

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the rung goes false.
.DN (Done) Bit 29	It is set when axis position cam has been successfully initiated.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.
.IP (In Process) Bit 26	It is set on positive rung transition and cleared if either superseded by another Motion Axis Position Cam command, or terminated by a stop command, merge, shutdown, or servo fault.
.PC (Process Complete) Bit 27	It is cleared on positive rung transition and set, in 'once' Execution Mode, when the position of the master axis leaves the master position range defined by the currently active cam profile.

Description: The Motion Axis Position Cam (MAPC) instruction executes a position cam profile set up by a previous Motion Calculate Cam Profile (MCCP) instruction or, alternatively, by the RSLogix 5000 Cam Profile Editor. Position cams, in effect, provide the capability of implementing non-linear “electronic gearing” relationships between two axes. No maximum velocity, acceleration, or deceleration limits are used. The speed, acceleration, and deceleration of the slave axis are completely determined by the motion of the master axis and the designated cam profile derived from the associated cam table.

ATTENTION

The maximum velocity, acceleration, or deceleration limits established during axis configuration do not apply to electronic camming.

Camming Direction

Cams can be configured to add or subtract their incremental contribution to the slave axis command position. Control over this behavior is via the Direction parameter.

Camming in the Same Direction

When Same is selected or entered as the Direction for the MAPC instruction, the slave axis position values computed from the cam profile are *added* to the command position of the slave axis. This is the most common operation, as the profile position values are used just as entered in the original cam table. That is, consecutive increasing profile values result in axis motion in the *positive* direction and vice-versa.

Camming in the Opposite Direction

When Opposite is selected or entered as the Direction, the slave axis position values computed from the cam profile are *subtracted* from the command position of the slave axis. Thus, axis motion is in the *opposite* direction from that implied by the original cam table. That is, consecutive increasing profile values result in axis motion in the *negative* direction and vice-versa.

Preserving the Current Camming Direction

When Unchanged is selected or entered as the Direction, other position cam parameters may be changed while preserving the current or previous camming direction (same or opposite). This is useful when the current direction is not known or not important. For first time execution of a cam with Unchanged selected, the control defaults the direction to Same.

Reversing the Current Camming Direction

When Reverse is selected the current or previous direction of the position cam is changed from Same to Opposite or from Opposite to Same. For first time execution of a cam with Reverse selected, the control defaults the direction to Opposite.

Specifying the Cam Profile

To execute a MAPC instruction, a calculated Cam Profile data array tag must be specified. Cam Profile array tags may be created by the RSLogix 5000 tag editor or the MAPC instruction using the built-in Cam Profile Editor, or by executing an Motion Calculate Cam Profile (MCCP) instruction on an existing Cam array.

The data within the Cam Profile array can be modified at compile time using the Cam Profile Editor, or at run-time with the Motion Calculate Cam Profile (MCCP) instruction. In the case of run-time changes, a Cam array must be created in order to use the MCCP instruction. Refer to the MCCP instruction specification for more detail on converting Cam arrays.

All but the status element of this Cam Profile array structure element are “hidden” from the RSLogix 5000 tag editor. These elements are of no value to the user. The Status member is used to indicate that the corresponding Cam Profile array element has been calculated. If execution of a camming instruction is attempted with any uncalculated elements in a cam profile, the instruction errors. The type parameter determines the type of interpolation applied between this cam array element and the next cam element, i.e. linear or cubic.

Cam Profile Array Checks

The Status member of the first element in the cam profile array is special and used for data integrity checks. For this reason, the MAPC must always specify the cam profile with the starting index set to 0.

This first cam profile element Status member can have the following values.

Status Value	**Description**
0	Cam profile element has not been calculated
1	Cam profile element is being calculated
2	Cam profile element has been calculated
n	Cam profile element has been calculated and is currently being used by (*n*-2) MAPC or MATC instructions

Before starting a cam on a specified axis, the MAPC instructions checks if the cam profile array has been calculated by checking the value of the first cam profile element's Status member. If Status is 0 or 1 then the cam profile has not been calculated yet and the MAPC instruction errors. If the cam profile array has been completely calculated (Status > 1), the instruction then increments the Status member indicating that it is in use by this axis.

When the cam completes, or terminates, the Status member of the first cam profile array element is decremented to maintain track of the number of cams actively using the associated cam profile.

Linear and Cubic Interpolation

Position cams are fully interpolated. This means that if the current Master Axis position does not correspond exactly with a point in the cam table associated with the cam profile, the slave axis position is determined by linear or cubic interpolation between the adjacent points. In this way, the smoothest possible slave motion is provided.

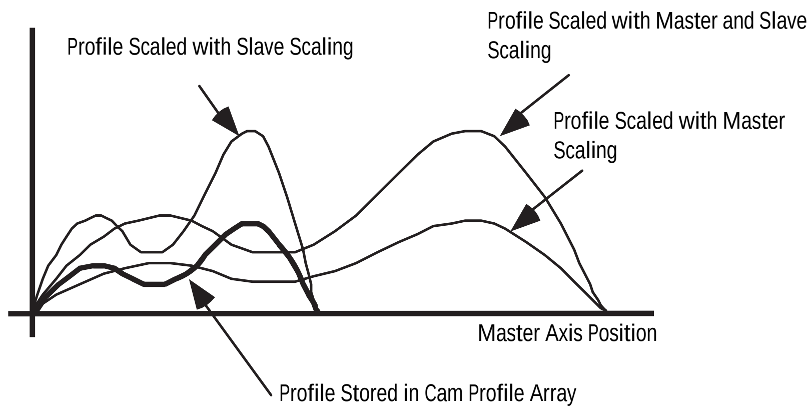
Each point in the Cam array that was used to generate the Cam Profile can be configured for linear or cubic interpolation.

Electronic camming remains active through any subsequent execution of jog, or move processes for the slave axis. This allows electronic camming motions to be superimposed with jog, or move profiles to create complex motion and synchronization.

Scaling Position Cams

A position cam profile can be scaled in both the master dimension and slave dimension when it is executed. This scaling feature is useful to allow the stored cam profile to be used to determine the general *form* of the motion profile. The scaling parameters are then used to define the total master or slave travel over which the profile is executed, as shown in the illustration below. In this way, one standard cam profile can be used to generate a whole family of specific cam profiles.

When a cam profile array is specified by an MAPC instruction, the master and slave values defined by the cam profile array take on the position units of the master and slave axes respectively. By contrast, the Master and Slave Scaling parameters are unitless values that are simply used as multipliers to the cam profile.



Cam Profile Array

By default, both the Master Scaling and Slave Scaling parameters are set to 1. To scale a position cam profile, enter a Master Scaling or Slave Scaling value other than 1.

Note that increasing the master scaling value of a cam profile *decreases* the velocities and accelerations of the profile, while increasing the slave scaling value *increases* the velocities and accelerations of the profile. To maintain the velocities and accelerations of the scaled profile approximately equal to those of the unscaled profile, the master scaling and slave scaling values should be equal. For example, if the slave scaling value of a profile is 2, the master scaling value should also be 2 to maintain approximately equal velocities and accelerations during execution of the scaled position cam.

ATTENTION	Decreasing the Master Scaling value or increasing the Slave Scaling value of a position cam increases the required velocities and accelerations of the profile. This can cause a motion fault if the capabilities of the drive system are exceeded.

Cam Profile Execution Modes

Execution Modes of Once or Continuous can be selected to determine how the cam motion behaves when the master position moves beyond the start and end points of the profile defined by the original cam table.

If Once is selected (default), the cam motion of the slave axis starts only when the master axis moves into the range defined by the start and end points of the cam profile. When the master axis moves outside the range of the profile, cam motion on the slave axis stops and the Process Complete bit of the MAPC instruction is set. Note that, contrary to the current S Class practice, slave motion **does not resume** when and if the master moves back into the profile range specified by the start and end points.

When Continuous mode is selected, the specified cam profile, once started, is executed indefinitely. With continuous operation, the profile's master and slave positions are "unwound" when the position of the master axis moves outside the profile range, causing the cam profile to repeat. This feature is particularly useful in rotary applications where it is necessary that the position cam run continuously in a rotary or reciprocating fashion. To generate smooth continuous motion using this technique, however, care must be taken in designing the cam points of the cam table to ensure that there are no position, velocity, or acceleration discontinuities between the start and end points of the calculated cam profile.

Execution Schedule

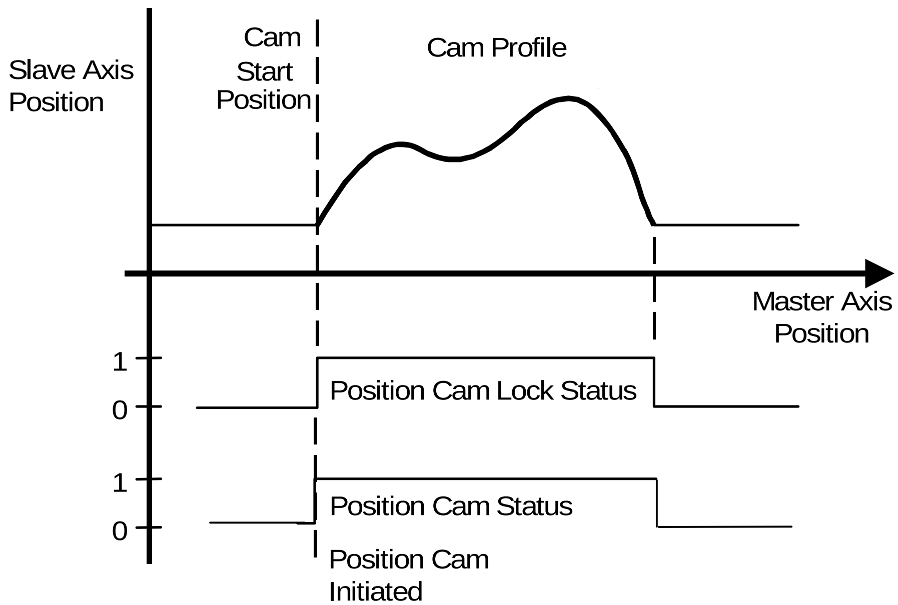
Control over the MAPC instruction's execution is via the Execution Schedule parameter.

Immediate Execution

By default, the MAPC instruction is scheduled to execute Immediately. In this case, there is no delay to the enabling of the position camming process and the Master Lock Position parameter is irrelevant. The slave axis is immediately locked to the master axis beginning at the Cam Lock Position of the specific cam profile.

As illustrated in the diagram below, when the MAPC instruction is executed, the camming process is initiated on the specified slave axis and the Position Cam Status bit in the slave axis' Motion Status word is set. If the Execution Schedule parameter is set to Immediate, the slave axis is immediately locked to the master according to the specified

Cam Profile. This is indicated by the fact that the Position Cam Lock Status bit for the specified slave axis is also set.

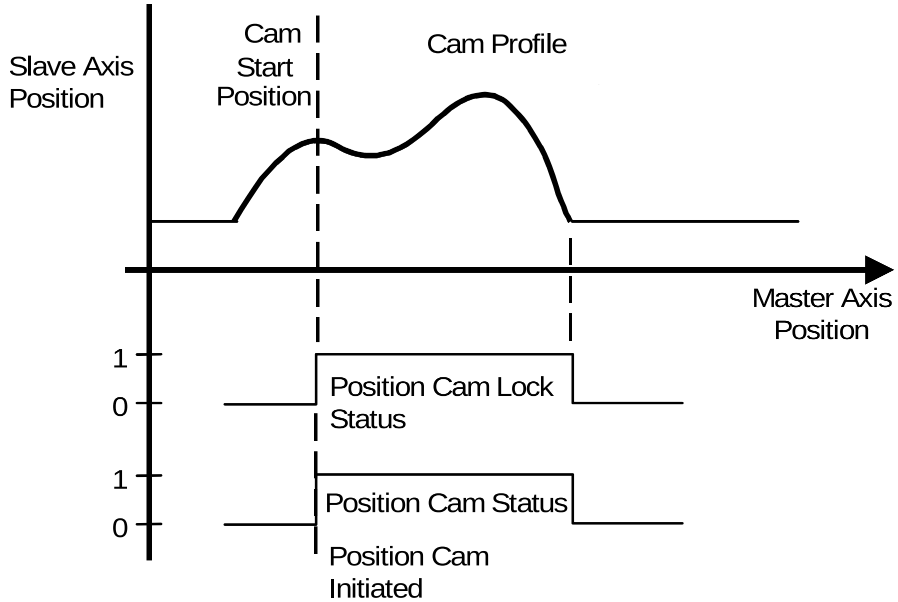


Immediate Execution

Changing the Cam Lock Position

The Cam Lock Position parameter of the MAPC instruction determines the starting location within the cam profile when the slave locks to the master. Typically, the Cam Lock Position is set to the beginning of the cam profile as shown in the above illustration. Since the starting point of most cam tables is 0, the Cam Lock Position is typically set to 0. Alternatively, the Cam Lock Position can be set to any position within the master range of the cam profile. If a Cam Lock Position is specified that is out of this range, the MAPC instruction errors.

The diagram below shows the effect of specifying a Cam Lock Position value other than the starting point of the cam table, in this case, a position within the cam profile itself. Care must be taken not to define a Cam Start Point that results in a velocity or acceleration discontinuity to the slave axis if the master axis is currently moving.

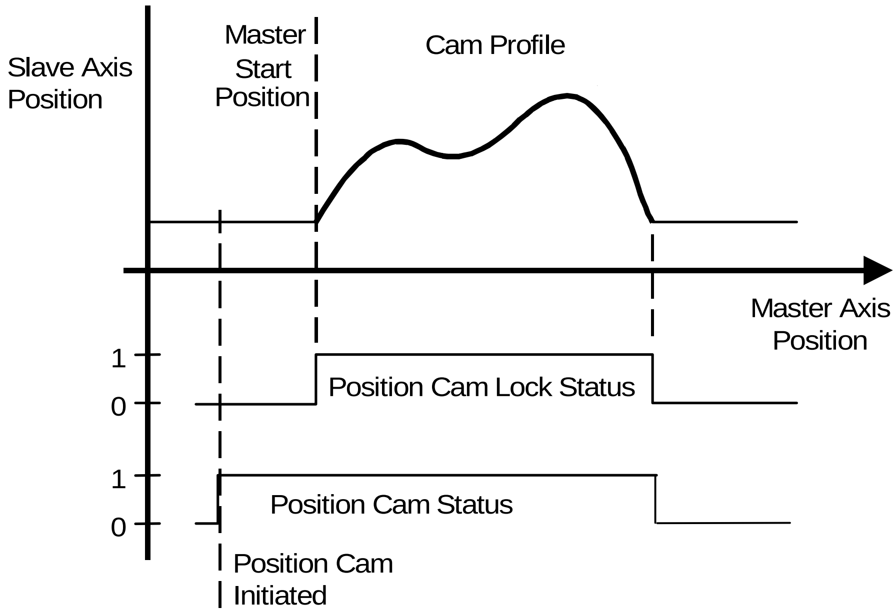


Changing the Cam Lock Position

Forward Only, Reverse Only, or Bi-directional Execution

In the case where the Execution Schedule parameter of the instruction is set to Forward Only, Reverse Only, or Bi-directional, the slave axis is not locked to the master until the master axis satisfies the specified condition. In this case, the master axis is monitored by the camming process to determine when the master axis passes the specified Master Lock Position in the specified direction. In a rotary axis configuration, this lock criterion is still valid, independent of the turns count.

IMPORTANT If the position reference of the master axis is redefined (e.g. an MRP instruction) after the MAPC instruction executes but before the lock condition is satisfied, the cam profile generator monitors the master axis based on the absolute position reference system in effect prior to the redefine position operation.



Forward Only, Reverse Only, or Bi-directional Execution

When the absolute position of the master axis passes the specified Master Lock Position in the specified direction ('Forward Only' direction in the illustration below), the Position Cam Status bit of the Motion Status word for specified slave axis is set. Slave axis motion is then initiated according to the specified cam profile starting at the specified Cam Lock Position of the cam profile. From this point on, only the *incremental change* in the master axis position is used to determine the corresponding slave axis position from the defined cam profile. This is important for applications where the master axis is a rotary axis since the position cam is then unaffected by the position unwind process.

When the master axis moves out of the range defined by the cam profile (assuming Execution Mode configured for Once), both the Position Cam Lock Status and the Position Cam Status bits of the Motion Status word are cleared. This Motion Status bit condition indicates that the cam process has completed. This fact is also reflected in the bit leg behavior of the associated MAPC instruction, PC bit set and IP bit clear.

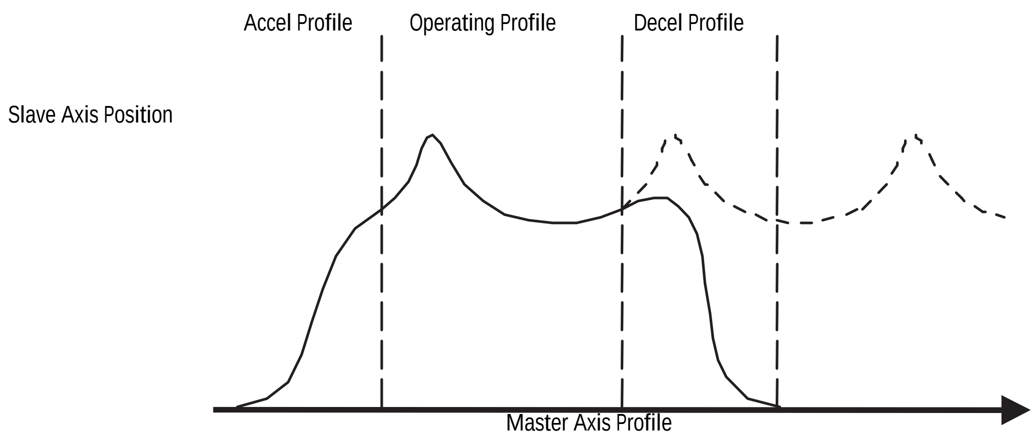
After position cam motion is started when the master axis passes the specified Master Lock Position in either the Forward Only or Reverse Only direction, the master axis can change direction and the slave axis reverses accordingly.

If an MAPC instruction is executed on a slave axis that is already actively position camming, an Illegal Dynamic Change error is generated (error code 23). The only exception for this is if the Execution Schedule is specified as 'pending'.

Pending Cam Execution

Alternatively, the MAPC instruction's execution can be deferred pending completion of a currently executing position cam. An Execution Schedule selection of Pending can thus be used to seamlessly blend two position cam profiles together without stopping motion.

The Pending execution feature is particularly useful in applications like high-speed packaging when a slave axis must be locked onto a moving master axis and accelerate using a specific profile to the proper speed. When this acceleration profile is done, it must be smoothly blended into the operating profile, which is typically executed continuously. To stop the slave axis, the operating profile is smoothly blended into a deceleration profile such that the axis stops at a known location as shown below.



Pending Cam Execution

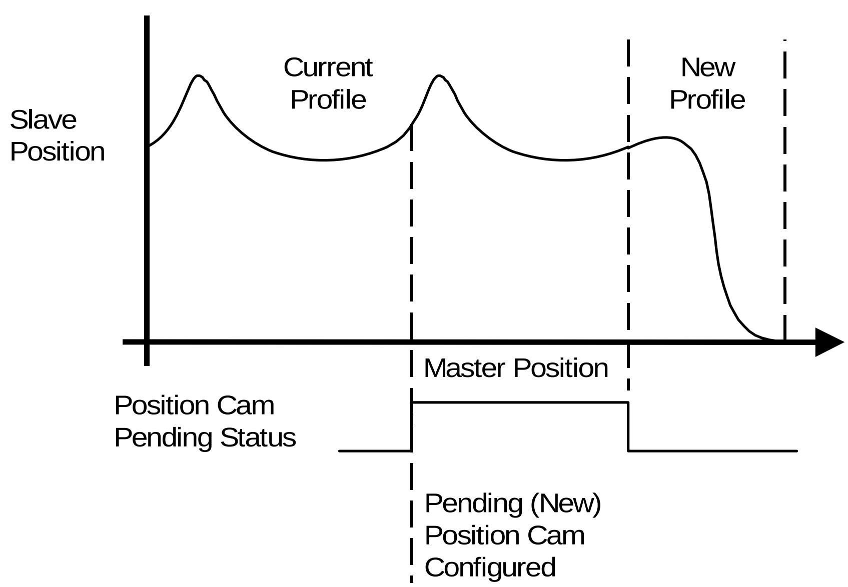
By executing the position cam profile as a Pending cam profile while the current profile is still executing, the appropriate cam profile parameters are set up ahead of time. This makes the transition from the current profile to the pending profile seamless; synchronization between the master and slave axes is maintained. To ensure smooth motion across the transition, however, the profiles must be designed such that no position, velocity, or acceleration discontinuities exist between the end of the current profile and the start of the new one. This is done using the RSLogix 5000 Cam Profile Editor.

Once a pending position cam instruction has been executed, the new cam profile takes effect automatically (and becomes the current profile) when the master axis passes through either the start or end point of the current profile. If the current cam is configured to execute once, the new profile is initiated at the completion of the pass through the current cam profile and the PC bit of the currently active MAPC instruction is set. If the current cam is configured to execute continuously, the new profile is initiated at the completion of the current pass through the current cam profile and the IP bit of the currently active MAPC instruction is cleared. The motion controller keeps track of the master axis and slave axis positions relative to the first profile at the time of the change and uses this information to maintain synchronization between the profiles.

If the Execution Schedule of an MAPC instruction is set to Immediate and a position cam profile is currently in process, the MAPC instruction errs. This is true even when the axis is waiting to lock onto the master axis.

If an Execution Schedule of Pending is selected without a corresponding position cam profile in progress, the MAPC instruction executes but no camming motion occurs until another MAPC instruction with a non-pending Execution Schedule is initiated. This allows pending cam profiles to be preloaded prior to executing the initial cam. This method addresses cases where immediate cams would finish before the pending cam could be reliably loaded.

After a Pending position cam has been configured, the Position Cam Pending Status bit of the Motion Status word for the specified slave axis is set to 1 (true). When the pending (new) profile is initiated and becomes the current profile, Position Cam Pending Status bit is immediately cleared as shown below.



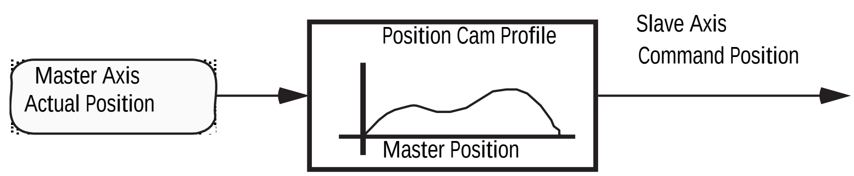
Pending Position Cam

Master Reference

The Master Reference parameter determines the master position source to link to the cam generator. This source can be actual position or command position of the master axis. Smoother motion is derived from command position but in some cases, e.g. when a physical axis is not controlled by a ControlLogix motion module, actual position is the only practical option.

Slaving to the Actual Position

When Actual Position is entered or selected as the Master Reference source, the slave axis motion is generated from the actual position of the master axis as shown below.

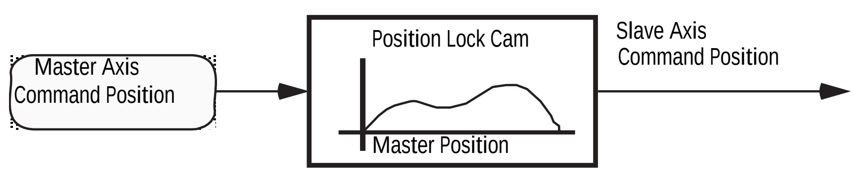


Slaving to the Actual Position

Actual position is the current position of the master axis as measured by its encoder or other feedback device. This is the default selection and the *only* selection when the master Axis Type is configured as Feedback Only since it is often necessary to synchronize the actual positions of two axes.

Slaving to the Command Position

When Command Position is entered or selected as the Master Reference source, the slave axis motion is generated from the command position of the master axis as shown below.



Slaving to the Command Position

Command position (only available when the master axis' Axis Type is a Servo or Virtual axis) is the desired or commanded position of the master axis.

Since the command position does not incorporate any associated following error or external position disturbances, it is a more accurate and stable reference for camming. When camming to the command position of the master, the master axis must be *commanded* to move to cause any motion on the slave axis. Refer to the Motion Axis Object Specification for more information on Command Position and Actual Position axis parameters.

Master Direction

Normally, the Master Direction parameter is set to Bi-directional (default). However, when Forward Only is selected for Master Direction, the slave axis tracks the master axis in the forward direction of the master axis. When Reverse Only is selected, the slave axis tracks the master axis in the reverse direction of the master axis. If the master axis changes direction, the slave axis does *not* reverse direction, but stays where it was when the master reversed. This Uni-directional feature of position cams is used to provide an electronic slip clutch, which prevents the cam motion generator from moving backward through the cam profile if the master reverses direction.

When the master axis again reverses, resuming motion in the desired direction, the slave axis “picks up” again when the master reaches the position where it initially reversed. In this way, the slave axis maintains synchronization with the master while motion in the wrong direction is inhibited. This is especially useful where motion in a certain direction can cause physical damage to the machine or to the product.

Moving While Camming

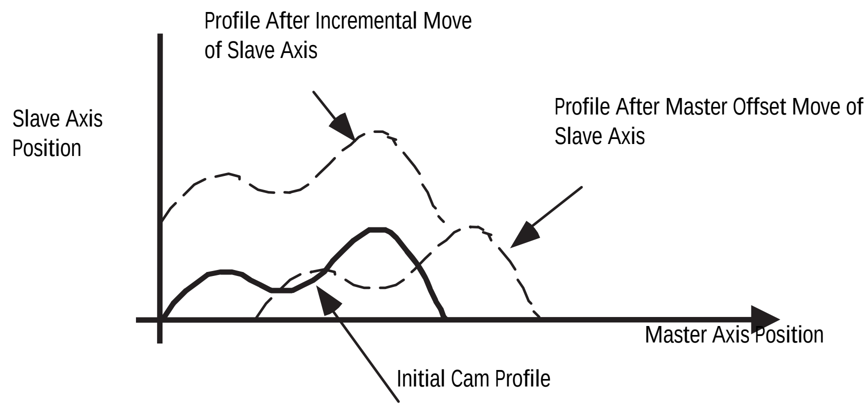
Motion Axis Moves may be performed while camming to provide sophisticated phase and offset control while the slave axis is running.

Incremental Moves

An Incremental Motion Axis Move (MAM) instruction may be used on the slave axis (or master axis if configured for Servo operation) while the position cam is operating. This is particularly useful to accomplish phase advance/retard control. The incremental move distance can be used to eliminate any phase error between the master and the slave, or to create an exact phase relationship.

Master Offset Moves

A MAM instruction can also be used while the position cam is operating to shift the master reference position of the cam on the fly. Unlike an incremental move on the slave axis, a master offset move on the slave axis shifts the cam profile relative to the master axis, as shown below.



Master Offset Move

When the MAPC instruction (except pending) is initiated, the corresponding active Master Offset Move is disabled and the corresponding Master Offset, Strobe Offset, and Start Master Offset are reset to zero. In order to achieve the master reference position shift, the MAM instruction must be initiated after the MAPC is initiated.

See the Motion Axis Move (MAM) instruction for more information on Master Offset moves.

Stopping a Cam

Like other motion generators (jog, move, gear, etc.) active cams must be stopped by the various stop instructions, MAS, or MGS. Cam motion must also stop when the ControlLogix processor changes OS modes. The MAS instruction, in particular, must be able to specifically stop the camming process. This behavior should be identical to the MAS functionality that specifically stops a gearing process.

Merging from a Cam

Like other motion generators (jog, move, gear, etc.) active cams must also be compliant with motion merge functionality. Moves and Jogs, in particular, must be able to merge from active camming. This behavior should be identical to the merge functionality applied to a gearing process.

Fault Recovery

Sometimes it is necessary to respond to an axis fault condition without loosing synchronization between a master and slave axis that are locked in a cam relationship. With an active cam there are a couple ways to handle axis faults.

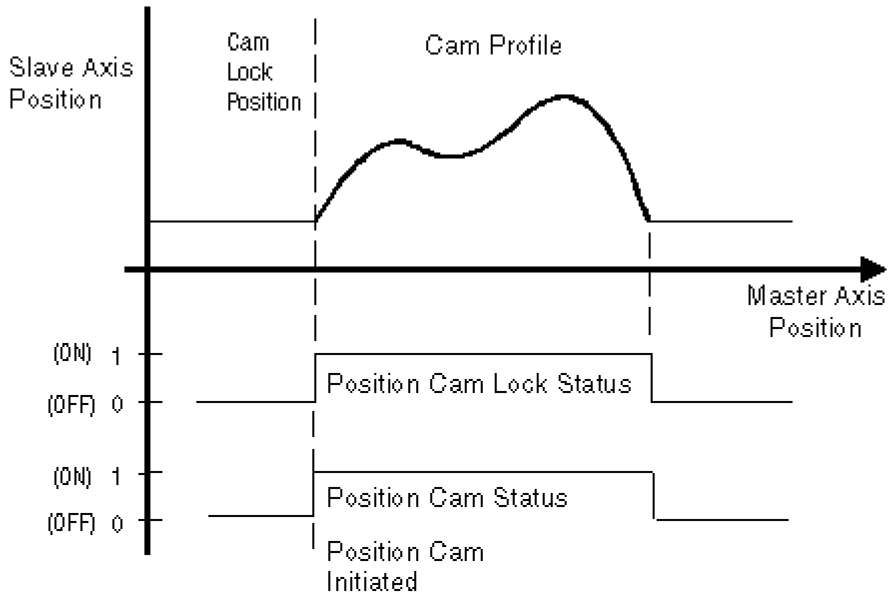
Create a virtual axis and cam everything to it and, if necessary, gear this virtual master axis to actual master axis of the machine. Set the various fault actions for all axes to Status Only. When an axis fault occurs (e.g. a drive fault) an application program monitoring the axes fault status detects the fault and does a controlled stop of all active axes by stopping the virtual master axis. At the profiler level, everything is still fully synchronized. Use the following error on faulted axis to determine how far it is out of position. Reset the fault on the faulted axis, bring into position at a controlled speed using the MAM instruction and the computed following error. Finally, start moving virtual master axis.

Same configuration as above but, in this case, when the slave axis faults the axis fault action disables the drive. This, of course, would terminate the active cam process on the slave axis. At this point, the application program should stop all other axes via the virtual master axis. Next, reposition the faulted axis by determining where the master is, and then calculating where the slave axis should be had the fault not occurred. Finally, do an immediate lock MAPC to resynchronize with the Cam Lock Position set to the calculated value.

IMPORTANT The MAPC instruction execution completes in a single scan, thus the Done (.DN) bit and the In Process (.IP) bit are set immediately. The In Process (.IP) bit remains set until the initiated PCAM process completes, is superseded by another MAPC instruction, terminated by a Motion Axis Stop command, Merge operation, or Servo Fault Action. The Process Complete bit is cleared immediately when the MAPC executes and sets when the cam process completes when configured for 'Once' Execution Mode.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.



Position Cam Timing Diagram

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes: See Error Codes (ERR) for Motion Instructions on page A-383.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions.

Extended Error Codes for Axis Not Configured (11) error code are as follows:

- Extended Error Code 1 signifies that the Slave Axis is not configured.
- Extended Error Code 2 signifies that the Master Axis is not configured.

Extended Error codes for the Parameter Out of Range (13) error code lists a number that refers to the number of the operand as they are listed in the faceplate from top to bottom with the first operand being counted as zero. Therefore for the MAPC instruction, an extended error code of 5 would refer to the Slave Scaling operand's value. You would then have to check your value with the accepted range of values for the instruction.

For the Error Code 54 – Maximum Deceleration Value is Zero, if the Extended Error returns a positive number (0-*n*) it is referring to the offending axis in the coordinate system. Go to the Coordinate System Properties General Tab and look under the Brackets ([ ])column of the Axis Grid to determine which axis has a Maximum Deceleration value of 0. Click on the ellipsis button next to the offending axis to access the Axis Properties screen. Go to the Dynamics tab and make the appropriate change to the Maximum Deceleration Value. If the Extended Error number is -1, this means the Coordinate System has a Maximum Deceleration Value of 0. Go to the Coordinate System Properties Dynamics Tab to correct the Maximum Deceleration value.

Status Bits:

MAPC Changes to Status Bits

If the Execution Schedule is set to Immediate, execution of the MAPC instruction simply sets both the Position Cam Status and the Position Cam Lock Status bits to True.

Bit Name	State	Meaning
Position Cam Status	TRUE	Position Camming is Enabled
Position Cam Lock Status	TRUE	Slave Axis is Locked to the Master Axis according to the Cam Profile.
Position Cam Pending Status	FALSE	No pending Position Cam

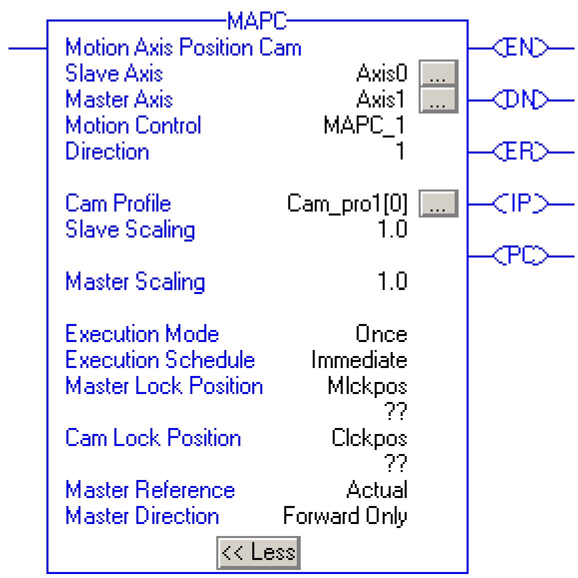
If the Execution Schedule is set to Forward or Reverse, execution of the MAPC instruction initially sets the Position Cam Status bit to True and the Position Cam Lock Status bits to False. Position Cam Lock Status transitions to True when the Execution Schedule condition is satisfied.

Bit Name	State	Meaning
Position Cam Status	TRUE	Position Camming is Enabled
Position Cam Lock Status	FALSE	Slave Axis is waiting for Master Axis to reach Lock Position.
Position Cam Pending Status	FALSE	No pending Position Cam

If the Execution Schedule is set to Pending, execution of the MAPC instruction does not affect the current state of either the Position Cam Status or Position Lock Status bits. Position Cam Pending Status bit is set to True immediately and transitions to False when the pending cam becomes the active cam.

Bit Name	State	Meaning
Position Cam Status	N/A	Position Camming is Enabled
Position Cam Lock Status	N/A	Slave Axis is waiting for Master Axis to reach Lock Position.
Position Cam Pending Status	True	Pending Position Cam

Example: *Relay Ladder*



MAPC Ladder Example

Structured Text

MAPC(Axis0,Axis1,MAPC_1,1,Cam_pro1[0],1.0,1.0,Once,
immediate,Mlckpos,Clckpos,Actual,Forwardonly);

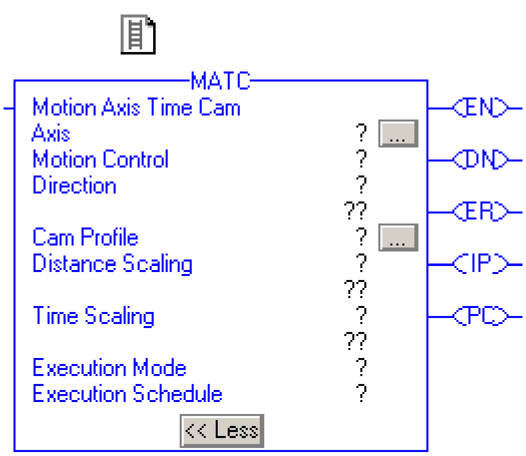
Motion Axis Time Cam (MATC)

The Motion Axis Time Cam (MATC) instruction provides electronic camming of an axis as a function of time, according to the specified Cam Profile. Time cams allow the execution of complex motion profiles other than the built-in trapezoidal, or S-curve move profiles. When executed, the specified Axis is synchronized in time using a time Cam Profile established by the RSLogix 5000 Cam Profile Editor, or by a previously executed Motion Calculate Cam Profile (MCCP) instruction. The direction of axis motion relative to the cam profile is defined by a very flexible Direction input parameter. The camming Direction may be explicitly set as the Same or Opposite or set relative to the current camming direction as Reverse or Unchanged. The cam profile can be configured via the Execution Schedule parameter to execute Immediately or Pending completion of a currently executing time cam profile. The cam profile can also be executed Once or Continuously by specifying the desired Execution Mode. Distance and Time Scaling functionality can be used to scale axis motion based on a standard cam profile without having to create a new cam table and calculate a new cam profile.

ATTENTION

Use a motion control tag only once. Do not re-use it in another instruction. Otherwise, you can cause unexpected equipment motion and injure people.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Axis	AXIS_FEEDBACK AXIS_VIRTUAL AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	The name of the axis to which the cam profile is applied. Ellipsis launches Axis Properties dialog.
Motion Control	MOTION_ INSTRUCTION	tag	Structure used to access block status parameters.

Operand	Type	Format	Description
Direction	UINT32	immediate or tag	Relative direction of the slave axis to the master axis: • Same – the axis position values in the cam profile are added to the command position of the axis. • Opposite – the axis position values in the cam profile are subtracted from the command position of the axis creating axis motion in the other direction from that implied in the original cam table. Or relative to the current or previous camming direction: • Reverse – the current or previous direction of the position cam is changed either from Same to Opposite or vice versa. When executed for the first time with Reverse selected, the control defaults the direction to Opposite. • Unchanged – this allows other cam parameters to be changed without altering the current or previous camming direction. When executed for the first time with Unchanged selected, the control defaults the direction to Same.
Cam Profile	CAM_PROFILE	array	Tag name of the calculated cam profile array. Only the zero array element ([0]) is allowed for the Cam Profile array. Ellipsis launches Cam Profile Editor.
Distance Scaling	REAL	immediate or tag	Scales the total distance covered by the axis through the cam profile.
Time Scaling	REAL	immediate or tag	Scales the time interval covered by the cam profile.

Operand	Type	Format	Description
Execution Mode	UINT32	immediate	Determines how the cam motion behaves when the time moves beyond the end point of the cam profile. The options are: 0 = Once – When the time cam execution time exceeds the time range in the cam profile, the MATC instruction completes, the axis motion stops, and the Time Cam Status bit is cleared. 1 = Continuous – The cam profile motion is executed indefinitely.
Execution Schedule	UINT32	immediate	Selects the method used to execute the cam profile. Options are: 0 = Immediate – instruction is scheduled to execute immediately with no delay enabling the time camming process. 1 = Pending – Defers execution of the time cam until the completion of the currently or next immediate executing time cam. This is useful in blending a new time cam profile with an on going process to achieve a seamless transition.

MATC(Axis,MotionControl,
Direction,CamProfile,
DistanceScaling,TimeScaling,
ExecutionMode,

Structured Text

The operands are the same as those for the relay ladder MATC instruction. For the array operands, you do not have to include the array index. If you do not include the index, the instruction starts with the first element in the array ([0]).

For the operands that require you to select from available options, enter your selection as:

This operand	Has these options which you...	
	enter as text	or enter as a number
ExecutionMode	once	0
	continuous	1
ExecutionSchedule	immediate	0
	pending	1

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	The enable bit is set when the rung transitions from false-to-true and stays set until the rung goes false.
.DN (Done) Bit 29	The done bit is set when the axis time cam instruction is successfully initiated.
.ER (Error) Bit 28	The error bit indicates when the instruction detects an error, such as if the axis is not configured.
.IP (In Process) Bit 26	The in process bit is set on positive rung transition and cleared when terminated by a stop command, merge, shutdown, or servo fault.
.PC (Process Complete) Bit 27	The Process Complete bit is cleared on positive rung transition and set in Once Execution Mode, when the time leaves the time range defined by the currently active cam profile.

Description: The Motion Axis Time Cam (MATC) instruction executes a time cam profile set up by a previous Motion Calculate Cam Profile (MCCP) instruction or, alternatively, by the RSLogix 5000 Cam Profile Editor. Time cams provide the capability of implementing complex motion profiles other than the built-in trapezoidal and S-Curve motion profiles provided. No maximum velocity, acceleration, or deceleration limits are used in this instruction. The speed, acceleration, and deceleration of the slave axis are completely determined by the designated cam profile derived from the associated cam table.

ATTENTION

The maximum velocity, acceleration, or deceleration limits established during axis configuration do not apply to electronic camming.

Camming Direction

Cams can be configured to add or subtract their incremental contribution to the axis command position. Control over this behavior is via the Direction parameter.

Camming in the Same Direction

When Same is selected or entered as the Direction for the MATC instruction, the axis position values computed from the cam profile are *added* to the command position of the axis. This is the most common operation, as the profile position values are used just as entered in the original cam table. That is, consecutive increasing profile values result in axis motion in the *positive* direction and vice-versa.

Camming in the Opposite Direction

When Opposite is selected or entered as the Direction, the axis position values computed from the cam profile are *subtracted* from the command position of the axis. Thus, axis motion is in the *opposite* direction from that implied by the original cam table. That is, consecutive increasing profile values result in axis motion in the *negative* direction and vice-versa.

Changing the Cam Profile

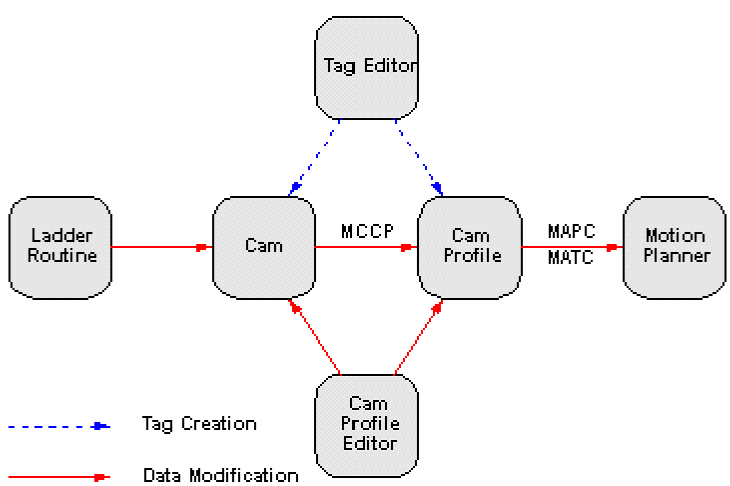
When Unchanged is selected or entered as the Direction, other time cam parameters may be changed while preserving the current or previous camming direction (same or opposite). This is useful when the current direction is not known or not important. For first time execution of a cam with Unchanged selected, the control defaults the direction to Same.

Changing the Camming Direction

When Reverse is selected the current or previous direction of the time cam is changed from Same to Opposite or from Opposite to Same. For first time execution of a cam with Reverse selected, the control defaults the direction to Opposite.

Specifying the Cam Profile

To execute a MATC instruction, a calculated Cam Profile data array tag must be specified. Cam Profile array tags may be created by the RSLogix 5000 tag editor or the MATC instruction using the built-in Cam Profile Editor, or by executing an Motion Calculate Cam Profile (MCCP) instruction on an existing Cam array. See the following figure:



MATC Process

The data within the Cam Profile array can be modified at compile time using the Cam Profile Editor, or at run-time with the Motion Calculate Cam Profile (MCCP) instruction. In the case of run-time changes, a Cam array must be created in order to use the MCCP instruction. Refer to the MCCP instruction specification for more detail on converting Cam arrays.

All but the status and type elements of the Cam Profile array element structure are "hidden" from the RSLogix 5000 tag editor. These hidden elements are of no value. The status parameter is used to indicate that the Cam Profile array element has been calculated. If execution of a camming instruction is attempted with any uncalculated elements in a cam profile, the instruction errors. The type parameter determines the type of interpolation applied between this cam array element and the next cam element.

Cam Profile Array Checks

The Status member of the first element in the cam profile array is special and used for data integrity checks. For this reason, the MATC must always specify the cam profile with the starting index set to 0. This first cam profile element Status member can have the following values:

Status Value	Description
0	Cam profile element has not been calculated
1	Cam profile element is being calculated
2	Cam profile element has been calculated
n	Cam profile element has been calculated and is currently being used by (*n*-2) MAPC or MATC instructions

Before starting a cam on a specified axis, the MATC instructions checks if the cam profile array has been calculated by checking the value of the first cam profile element's Status member. If Status is 0 or 1 then the cam profile has not been calculated yet and the MATC instruction errors. If the cam profile array has been completely calculated (Status > 1), the instruction then increments the Status member indicating that it is in use by this axis.

When the cam completes, or terminates, the Status member of the first cam profile array element is decremented to maintain track of the number of cams actively using the associated cam profile.

Linear and Cubic Interpolation

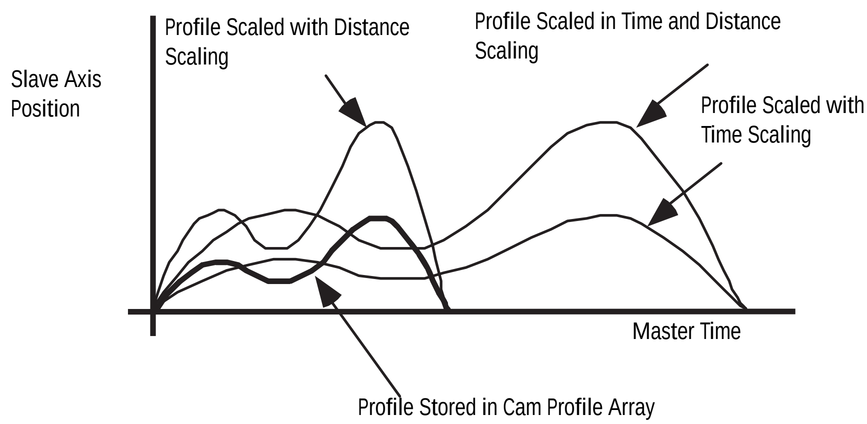
Time cams are fully interpolated. This means that if the current master time value does not correspond exactly with a point in the cam table associated with the cam profile, the slave axis position is determined by linear or cubic interpolation between the adjacent points. In this way, the smoothest possible slave motion is provided.

Each point in the Cam array that was used to generate the Cam Profile can be configured for linear or cubic interpolation.

Electronic camming remains active through any subsequent execution of jog, or move processes for the slave axis. This allows electronic camming motions to be superimposed with jog, or move profiles to create complex motion and synchronization.

Scaling Time Cams

A time cam profile can be scaled in both time and distance when it is executed. This scaling is useful to allow the stored profile to be used only for the *form* of the motion with the scaling used to define the time or distance over which the profile is executed, as shown below.



Scaling Time Cams

When a cam profile array is specified by an MATC instruction, the master coordinate values defined by the cam profile array take on the time units (seconds) and the slave values take on the units of the slave axis. By contrast, the Time and Distance Scaling parameters are “unitless” values that are simply used as multipliers to the cam profile.

By default, both the Time and Distance Scaling parameters are set to 1. To scale a time cam profile, enter a Time Scaling or Distance Scaling value other than 1.

Increasing the Time Scaling value of a cam profile *decreases* the velocities and accelerations of the profile, while increasing the Distance Scaling value *increases* the velocities and accelerations of the profile. To maintain the velocities and accelerations of the scaled profile approximately equal to those of the unscaled profile, the Time Scaling and Distance Scaling values should be equal. For example, if the Distance Scaling value of a profile is 2, the Time Scaling value should also be 2 to maintain approximately equal velocities and accelerations during execution of the scaled time cam.

ATTENTION

Decreasing the Time Scaling value or increasing the Distance Scaling of a time cam increases the required velocities and accelerations of the profile. This can cause a motion fault if the capabilities of the drive system are exceeded.

Cam Profile Execution Modes

Execution Modes of Once or Continuous can be selected to determine how the cam motion behaves when the time moves beyond the end point of the profile defined by the original cam table.

If Once is selected (default), the cam profile motion of the axis starts immediately. When the time cam execution time exceeds the time range defined by the cam profile, the MATC instruction completes, axis motion stops, and the Time Cam Status bit in the slave axis' Motion Status word is cleared.

When Continuous mode is selected, the specified cam profile, starts immediately and is executed indefinitely. With continuous operation, time is "unwound" to the beginning of the cam profile when it moves beyond the end of the cam profile, causing the cam profile to repeat indefinitely. This feature is particularly useful in rotary applications where it is necessary that the time cam run continuously in a rotary or reciprocating fashion. To generate smooth continuous motion using this technique, however, care must be taken in designing the cam points of the cam table to ensure that there are no position, velocity, or acceleration discontinuities between the start and end points of the calculated cam profile.

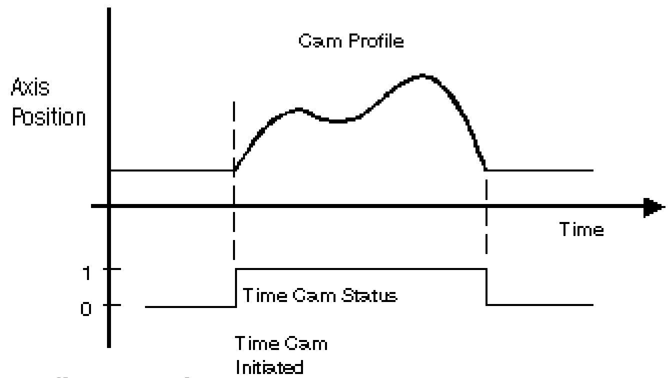
Execution Schedule

Control over the MATC instruction's execution schedule is via the Execution Schedule parameter.

Immediate Execution

By default, the MATC instruction is scheduled to execute immediately by virtue of the fact that the default setting of the Execution Schedule parameter is Immediate. In this case, there is no delay to the enabling of the time camming process.

As illustrated in the diagram below, when the MATC instruction is executed, the camming process is initiated on the specified axis and the Time Cam Status bit in the axis' Motion Status word is set. If the Execution Schedule parameter is set to Immediate, the axis is immediately locked to the time master coordinate according to the specified Cam Profile.



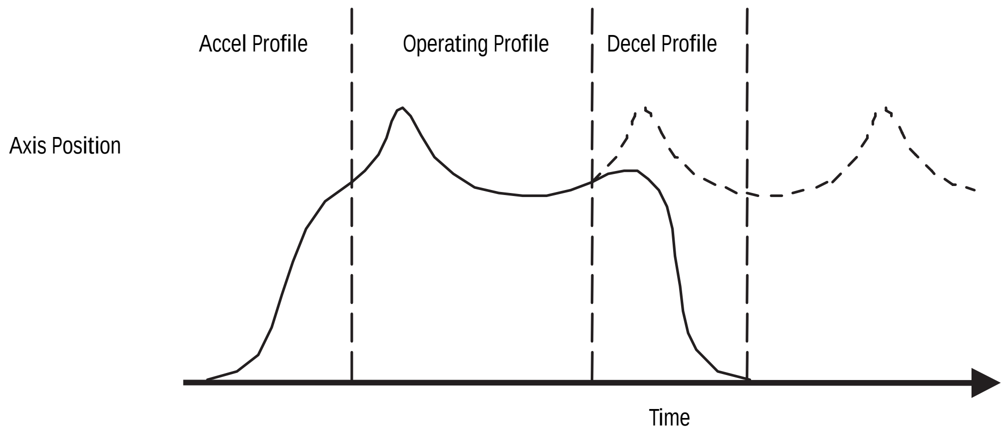
Immediate Execution

If an MATC instruction is executed on an axis that is already actively time camming, an Illegal Dynamic Change error is generated (error code 23). The only exception for this is if the Execution Schedule is specified as 'pending'.

Pending Cam Execution

Alternatively, the MATC instruction's execution can, in effect, be deferred pending completion of a currently executing time cam. An Execution Schedule selection of Pending can thus be used to seamlessly blend two time cam profiles together without stopping motion.

The Pending execution feature is particularly useful in applications when the axis must be accelerated up to speed using a specific velocity profile. When this acceleration profile is done, it must be smoothly blended into a cam profile which is typically executed continuously. To stop the axis, the operating profile can be smoothly blended into a deceleration profile such that the axis stops at a known location as shown below.



Pending Cam Execution

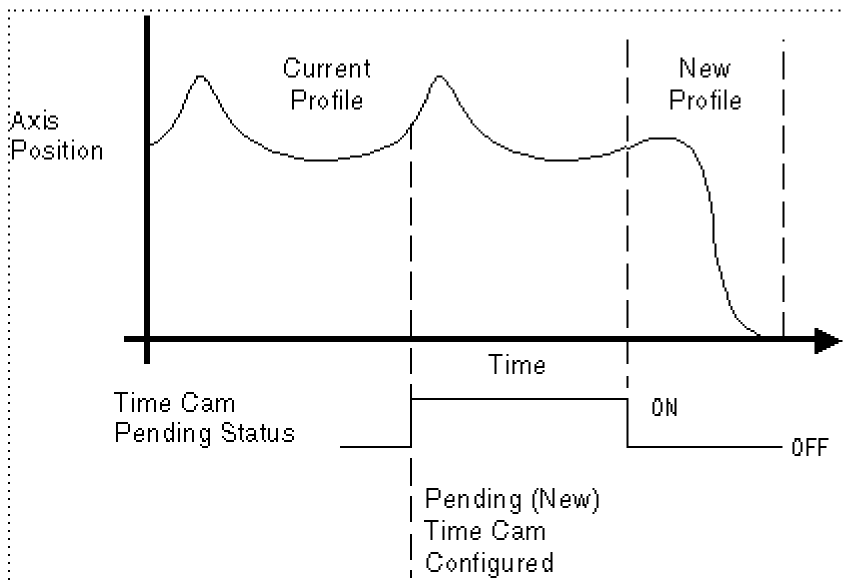
By executing the time cam profile as a Pending cam profile while the current profile is still executing, the appropriate cam profile parameters are set up ahead of time. This makes the transition from the current profile to the pending profile seamless – synchronization between the master time and slave axes position is maintained. To ensure smooth motion across the transition, however, the profiles must be designed such that no position, velocity, or acceleration discontinuities exist between the end of the current profile and the start of the new one. This is done using the RSLogix 5000 Cam Profile Editor.

Once a pending time cam instruction has been executed, the new cam profile takes effect automatically (and becomes the current profile) when cam time passes through the end of the current profile. If the current cam is configured to execute once, the new profile is initiated at the completion of the pass through the current cam profile and the PC bit of the currently active MATC instruction is set. If the current cam is configured to execute continuously, the new profile is initiated at the completion of the current pass through the current cam profile and the IP bit of the currently active MATC instruction is cleared. The motion controller keeps track of time and the axis positions relative to the first profile at the time of the change and uses this information to maintain synchronization between the profiles.

If the Execution Schedule of an MATC instruction is set to Immediate and a time cam profile is currently in process, the MATC instruction generates an Illegal Dynamic Change error.

If an Execution Schedule of Pending is selected without a corresponding time cam profile in progress, the MATC instruction executes but no camming motion occurs until another MATC instruction with a non-pending Execution Schedule is initiated. This allows pending cam profiles to be preloaded prior to executing the initial cam. This method addresses cases where immediate cams would finish before the pending cam could be reliably loaded.

After a Pending time cam has been configured, the Time Cam Pending Status bit of the Motion Status word for the specified axis is set to 1 (true). When the pending (new) profile is initiated and becomes the current profile, Time Cam Pending Status bit is immediately cleared as shown below.



Time Cam Pending

Stopping a Cam

Like other motion generators (jog, move, gear, etc.) active cams must be stopped by the various stop instructions, MAS, or MGS. Cam motion must also stop when the ControlLogix processor changes OS modes. The MAS instruction, in particular, must be able to specifically stop the camming process. This behavior should be identical to the MAS functionality that specifically stops a gearing process.

Merging from a Cam

Like other motion generators (jog, move, gear, etc.) active cams must also be compliant with motion merge functionality. Moves and Jogs, in particular, must be able to merge from active camming. This behavior should be identical to the merge functionality applied to a gearing process.

IMPORTANT The MATC instruction execution completes in a single scan, thus the Done (.DN) bit and the In Process (.IP) bit are set immediately. The In Process (.IP) bit remains set until the initiated Time Camming process is superseded by another MATC instruction, or terminated by a Motion Axis Stop command, Merge operation, or Servo Fault Action.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes: See Error Codes (ERR) for Motion Instructions on page A-383.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions. Extended Error codes for the Parameter Out of Range (13) error code lists a number that refers to the number of the operand as they are listed in the faceplate from top to bottom with the first operand being counted as zero. Therefore for the MATC instruction, an extended error code of 5 would refer to the Time Scaling operand's value. You would then have to check your value with the accepted range of values for the instruction.

For the Error Code 54 – Maximum Deceleration Value is Zero, if the Extended Error returns a positive number (0-*n*) it is referring to the offending axis in the coordinate system. Go to the Coordinate System Properties General Tab and look under the Brackets ([ ])column of the Axis Grid to determine which axis has a Maximum Deceleration value of 0. Click on the ellipsis button next to the offending axis to access the Axis Properties screen. Go to the Dynamics tab and make the appropriate change to the Maximum Deceleration Value. If the Extended Error number is -1, this means the Coordinate System has a Maximum Deceleration Value of 0. Go to the Coordinate System Properties Dynamics Tab to correct the Maximum Deceleration value.

MATC Changes to Status Bits: *Status Bits*

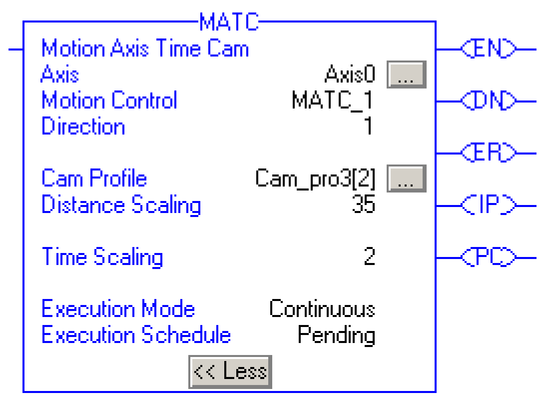
If the Execution Schedule is set to Immediate, execution of the MATC instruction simply sets the Time Cam Status bit to True.

Bit Name	State	Meaning
TimeCamStatus	TRUE	Time Camming is Enabled
TimeCamPendingStatus	FALSE	No pending Time Cam

If the Execution Schedule is set to Pending, execution of the MATC instruction does not affect the current state of the Time Cam Status bits. Time Cam Pending Status bit is set to True immediately and transitions to False when the pending cam becomes the active cam.

Bit Name	State	Meaning
TimeCamStatus	N/A	Time Camming is Enabled
TimeCamPendingStatus	TRUE	Pending Time Cam

Example: *Relay Ladder*



MATC Ladder Example

Structured Text

MATC(Axis0,MATC_1,1,Cam_pro3[2],35,2,Continuous, Pending);

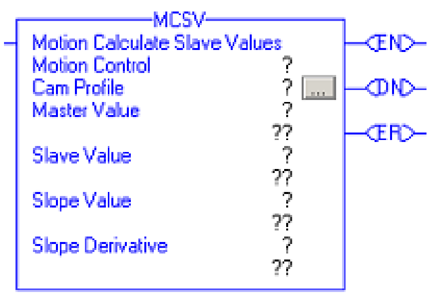
Motion Calculate Slave Values (MCSV)

Use the Motion Calculate Slave Values (MCSV) instruction to calculate the slave value, the slope value, and the derivative of the slope for a given cam profile and master value.

ATTENTION

Use a motion control tag only once. Do not re-use it in another instruction. Otherwise, you can cause unexpected equipment motion and injure people.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Motion Control	MOTION_INSTRUCTION	tag	Structure used to access instruction status parameters.
Cam Profile	CAM_PROFILE	array tag	An array of elements with the array index set to 0. It defines the cam profile used in calculating the slave values.
Master Value	SINT, INT, DINT, or REAL	immediate or tag	The exact value along the master axis of the cam profile that is used in calculating the slave values.
Slave Value	REAL	tag	The value along the slave axis of the cam profile with the master at the specified master value.
Slope Value	REAL	tag	The first derivative of the value along the slave axis of the cam profile with the master at the specified master value.
Slope Derivative	REAL	tag	The second derivative of the value along the slave axis of the cam profile with the master at the specified master value.

Structured Text

```
MCSV(MotionControl,CamProfile,
MasterValue,SlaveValue,
SlopeValue,SlopeDerivative)
```

The operands are the same as those for the relay ladder MCSV instruction.

Description: The Motion Calculate Slave Values (MCSV) instruction determines the slave value, the slope value, and the derivative of the slope for a given cam profile and master value. As an extension to the position and time camming functionality it supplies the values essential for the recovery from faults during camming operations.

Motion Control

The following control bits are affected by the MCSV instruction.

Mnemonic	Description
.EN (Enable) Bit 31	The Enable Bit sets when the rung transitions from false to true. It resets when the rung goes from true to false.
.DN (Done) Bit 29	The Done Bit sets when the slave values have been calculated successfully. It resets when the rung transitions from false to true.
.ER (Error) Bit 28	The Error Bit sets when the slave values have not been calculated successfully. It resets when the rung transitions from false to true.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

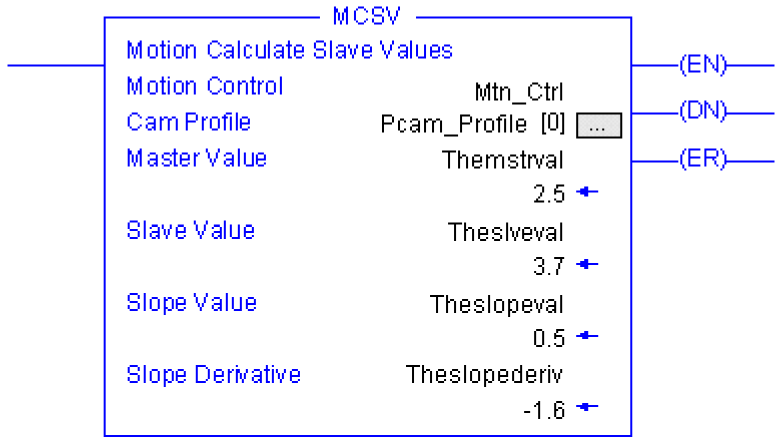
Fault Conditions: none

Error Codes: See Error Codes (ERR) for Motion Instructions on page A-383.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions. Extended Error codes for the Parameter Out of Range (13) error code lists a number that refers to the number of the operand as they are listed in the faceplate from top to bottom with the first operand being counted as zero. Therefore for the MCSV instruction, an extended error code of 2 would refer to the Master Value operand's value. You would then have to check your value with the accepted range of values for the instruction.

MCSV Changes to Status Bits: None

Example: Relay Ladder



MCSV Ladder Instruction

Structured Text

MCSV(Mtn_Ctrl,Pcam_Profile[0],Thestrval,Theslveval,Theslopeval, Theslopederiv)

Notes

Motion Group Instructions
(MGS, MGSD, MGSR, MGSP)

ATTENTION

Tags used for the motion control attribute of instructions should only be used once. Reuse of the motion control tag in other instructions can cause unintended operation. This may result in damage to equipment or personal injury.

Introduction

Group Control Instructions include all motion instructions that operate on all the axes in the specified group. Instructions that can be applied to groups include position strobe, shutdown control, and stopping instructions. Note that at present only one group is supported per Logix controller.

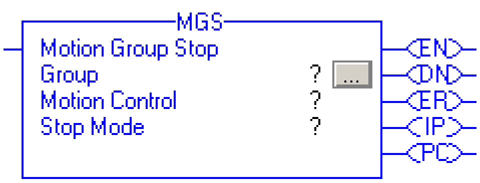
The motion group instructions are:

If you want to	Use this instruction	Available in these languages
Initiate a stop of motion on a group of axes.	MGS	relay ladder structured text
Force all axes in a group into the shutdown operating state.	MGSD	relay ladder structured text
Transition a group of axes from the shutdown operating state to the axis ready operating state.	MGSR	relay ladder structured text
Latch the current command and actual position of all axes in a group.	MGSP	relay ladder structured text

Motion Group Stop (MGS)

The MGS instruction initiates a stop of all motion in progress on all axes in the specified group by a method configured individually for each axis or as a group via the Stop Mode of the MGS instruction. If the MGS Stop Mode is specified as Programmed, each axis in the group is stopped according to the configured Programmed Stop Mode axis attribute. This is the same stopping mechanism that is employed by the Logix Operating System when there is a Logix controller state change. This Programmed Stop Mode attribute currently provides five different methods of stopping an axis: Fast Stop, Fast Disable, Hard Disable, Fast Shutdown, and Hard Shutdown. Alternatively, an explicit Stop Mode may be selected by using the MGS instruction. If a Stop Mode of Fast Disable is selected, all axes in the group stop with Fast Disable behavior. When the motion of all the axes in the group has been brought to a stop, the Process Complete (PC) bit is set in the control structure.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Group	MOTION_ GROUP	tag	Name of the group of axes to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.
Stop Mode	UDINT	immediate	Controls how the axes in the group are stopped. Select one of the following methods: 0 = Programmed - each axis is stopped according to how the individual axis has been configured. 1 = Fast Stop - each axis in the group is decelerated at the Maximum Deceleration rate and the stopped axis is left in the Servo Active state. 2 = Fast Disable - each axis in the group is decelerated at the Maximum Deceleration rate and the stopped axis is placed in the Axis Ready state.

Structured Text

```
MGS(Group,MotionControl,
    StopMode);
```

The operands are the same as those for the relay ladder MGS instruction.

For the operands that require you to select from available options, enter your selection as:

This operand	Has these options which you...	
	enter as text	or enter as a number
StopMode	programmed	0
	faststop	1
	fastdisable	2

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set when the group Programmed Stop has been successfully initiated for all axes in the group.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured group.
.IP (In Process) Bit 26	It is set on positive rung transition and cleared after the Motion Group Programmed Stop is complete.
.PC (Process Complete) Bit 27	It is set after all the axes in group have been successfully brought to a stop according to each axis' Programmed Stop Mode configuration.

Description: With the Stop Mode parameter set for Programmed, the Motion Group Stop (MGS) instruction brings motion for all of the axes in the specified group to a stop according to the configured Programmed Stop Mode for each axis. This instruction initiates the same programmed stopping action that is automatically applied when the processor's operating system changes operating mode (i.e. Run Mode to Program Mode, etc.). This is particularly useful in designing custom motion fault handlers.

If the MGS Stop Mode parameter is set to Fast Stop, each axis in the group is forced to perform a Fast Stop process, regardless of the configured Programmed Stop Mode. Each axis in the group is decelerated at the Maximum Deceleration rate and, once stopped, the axis is left in the Servo Active state.

If the MGS Stop Mode parameter is set to Fast Disable, each axis in the group is forced to perform a Fast Disable process, regardless of the configured Programmed Stop Mode. Each axis in the group is decelerated at the Maximum Deceleration rate and, once stopped, placed into the Axis Ready (servo inactive and drive disabled) state.

There are five Programmed Stop Modes that are currently supported by the MGPS instruction: Fast Stop, Fast Disable, Hard Disable, Fast Shutdown, and Hard Shutdown. Each axis may be configured to use any of these five stop modes. The following is a description of the effect of each these five stopping modes as they apply to an individual axis in the specified group.

Fast Stop

For an axis configured for a Fast Stop the MGPS instruction initiates a controlled stop much like that initiated by an MAS instruction. In this case the Motion Group Programmed Stop (MGS) instruction brings the axis motion to a controlled stop without disabling the axis servo loop. It is useful when a fast decelerated stop the axis is desired with servo control retained.

The MGPS instruction uses the configured Maximum Deceleration for the axis in this stop mode as the basis for the deceleration ramp applied to the axis.

Fast Disable

For an axis configured for a Fast Disable the MGS instruction initiates a controlled stop much like that initiated by an MAS instruction with the exception that the drive is disabled when the axis comes to a stop. Use MGS when a fast decelerated stop the axis is desired before the drive is disabled.

The MGS instruction uses the configured Maximum Deceleration for the axis in this stop mode as the basis for the deceleration ramp applied to the axis.

Hard Disable

For an axis configured for a Hard Disable the MGS instruction initiates the equivalent of an MSF instruction to the axis. This action immediately turns the appropriate axis drive output off, and disables the servo loop. Depending on the drive configuration, this may result in the axis coasting to a stop but offers the quickest disconnect of drive output power.

Fast Shutdown

For an axis configured for a Fast Shutdown, the MGS instruction initiates a Fast Stop and then applies the equivalent of a Motion Axis Shutdown (MASD) instruction to the axis. This action turns the appropriate axis driver output OFF, disables the servo loop, opens any associated motion module's OK contacts, and places the axis into the Shutdown state.

Hard Shutdown

For an axis configured for a Hard Shutdown the MGS instruction initiates the equivalent of an Motion Axis Shutdown (MASD) instruction to the axis. This action turns the appropriate axis drive output OFF, disables the servo loop, opens any associated motion module OK contacts, and places the axis into the Shutdown state. Depending on the drive configuration, this may result in the axis coasting to a stop but offers the quickest disconnect of Drive power via the OK contacts.

To successfully execute a MGS instruction, the targeted group must be configured.

IMPORTANT	The MGS instruction execution may take multiple scans to complete because the messages may require one or more axis motion modules in the group. Thus the Done .DN bit may not be set immediately. However, the In Process .IP bit is set and the Process Complete .PC bit is cleared immediately. The In Process .IP bit remains set until the initiated Programmed Stop process is completed for all axes in the specified group, or the stop instructions superseded by another MGS instruction, or terminated by a Servo Fault Action. The Process Complete .PC bit is only set if the initiated deceleration profile for each of the group's axes has completed prior to any other of the above events terminating the stop process and clearing the In Process .IP bit.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes See Error Codes (ERR) for Motion Instructions.

Status Bits

MGS Changes to Status Bits

Bit Name	State	Definition
StoppingStatus	TRUE	Axis is Stopping (Depending on the Programmed Stop Mode for the axis).
JogStatus	FALSE	Axis is no longer Jogging.
MoveStatus	FALSE	Axis is no longer Moving.
GearingStatus	FALSE	Axis is no longer Gearing.
HomingStatus	FALSE	Axis is no longer Homing.

Example: When the input conditions are true, the controller stops motion on all axes in *group1*. After the controller stops all motion, the axes are inhibited.

Relay Ladder

MGS Ladder Example

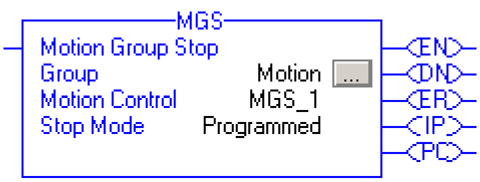
Structured Text

```
MGS(Motion,MSG_1,Programmed);
```

Motion Group Shutdown (MGSD)

Use the MGSD instruction to force all axes in the designated group into a Shutdown state. The Shutdown state of an axis is Servo Off, drive output is deactivated, and the motion module's OK solid-state relay contacts, if applicable, are opened. The group of axes remains in the Shutdown state until either Group Shutdown Reset is executed or each axis is individually reset via the Motion Axis Shutdown (MASD) instruction.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Group	MOTION_ GROUP	tag	Name of the group of axes to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.

Structured Text

```
MGSD(Group,MotionControl);
```

The operands are the same as those for the relay ladder MGSD instruction.

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	The enable bit indicates when the instruction is enabled. It remains set until servo messaging completes and the rung-condition-in goes false.
.DN (Done) Bit 29	The done bit indicates when the instruction sets the group of axes to the shutdown operating state.
.ER Error) Bit 28	The error bit indicates when the instruction detects an error, such as if messaging to the servo module failed.

Description: The Motion Group Shutdown (MGSD) instruction turns drive output off, disables the servo loops of all axes in the specified group, and opens any associated OK contacts for all applicable motion modules in the group. This action places all group axes into the Shutdown state. The MGSD instruction takes only one parameter; simply select or enter the desired group to shutdown.

Another action initiated by the MGSD instruction is the clearing of all motion processes in progress and a clearing of all the motion status bits. Associated with this action, the command also clears all motion instruction .IP bits that may currently be set for each axis in the group.

The MGSD instruction forces the targeted group of axes into the Shutdown state. One of the unique characteristics of the Shutdown state is that the OK solid state relay contact for all of the group's motion modules Open. This feature can be used to open up the E-Stop string(s) that control main power to the various drive systems.

Another characteristic of the Shutdown state is that any instruction that initiates axis motion for an axis within the group is blocked from execution. Attempts to do so results in an execution error. Only by executing one of the Shutdown Reset instructions can motion then be successfully initiated.

To successfully execute a MGSD instruction, the targeted group must be created and configured.

IMPORTANT	The MGSD instruction execution may take multiple scans to execute due to the fact that it requires transmission of a message to one or more motion modules. Thus, the Done .DN bit is not set immediately, but only after this message has been successfully transmitted.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

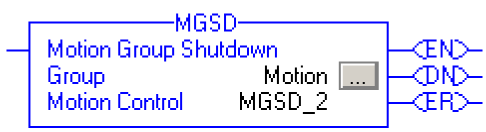
Error Codes See Error Codes (ERR) for Motion Instructions.

Status Bits: *MGSD Changes to Status Bits*

Bit Name	State	Definition
ServoActionStatus	FALSE	Axis is in Axis Ready state with the servo loop inactive.
DriveEnableStatus	FALSE	Axis Drive Enable output is inactive.
ShutdownStatus	TRUE	Axis is in Shutdown state.
AccelStatus	FALSE	Axis is not Accelerating
DecelStatus	FALSE	Axis is not Decelerating
GearingLockStatus	FALSE	Axis is not locked.
JogStatus	FALSE	Axis is not Jogging
MoveStatus	FALSE	Axis is not Moving
GearingStatus	FALSE	Axis is not Gearing
HomingStatus	FALSE	Axis is not Homing

Example: When the input conditions are true, the controller forces all axes in *group1* into a shutdown operating state.

Relay Ladder



MGSD Ladder Example

Structured Text

```
MGSD(Motion,MGSD_2);
```

Motion Group Shutdown Reset (MGSR)

Use the MGSR instruction to transition a group of axes from the shutdown operating state to the axis ready operating state. As a result of this command, all faults associated with the axes in the group are cleared and any OK relay contacts of motion modules associated with the specified group are closed.

Operands:

Relay Ladder

Operand	Type	Format	Description
Group	MOTION_ GROUP	tag	Name of the group of axes to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.

Structured Text

```
MGSR(Group,MotionControl);
```

The operands are the same as those for the relay ladder MGSR instruction.

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	The enable bit indicates when the instruction is enabled. It remains set until servo messaging completes and the rung-condition-in goes false.
.DN (Done) Bit 29	The done bit indicates when the instruction resets the group of axes from the shutdown operating state.
.ER (Error) Bit 28	The error bit indicates when the instruction detects an error, such as if messaging to the servo module failed.

Description:

The Motion Group Shutdown Reset (MGSR) instruction takes all the axes in the specified group out of the Shutdown state by clearing all axis faults and closing any associated OK solid-state relay contacts for the motion modules within the group. This action places all axes within the motion group in the Axis Ready state.

Just as MGSD instruction forces all the axes in the targeted group into the Shutdown state. The MGSR instruction takes all the axis in the specified group out of the Shutdown state and into the Axis Ready state. One of the unique characteristics of the Shutdown state is that, if supported, the OK solid state relay contact for each of the group's motion modules is Open. Hence, the result of an MGSR instruction applied to a group of motion modules is that all motion module OK relay contacts close. This feature can be used to close the E-Stop strings that control main power to the various drive systems and permits the customer to reapply power to the drives.

To successfully execute a MGSR instruction, the targeted group must be configured.

IMPORTANT	The MGSR instruction execution may take multiple scans to execute because it requires transmission of a message to one or more motion modules. The Done .DN bit is not set immediately, but only after this message has been successfully transmitted.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

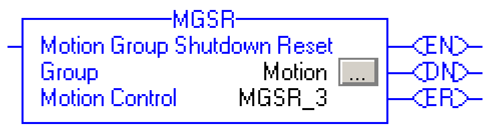
Error Codes See Error Codes (ERR) for Motion Instructions.

Status Bits: *MGSR Changes to Status Bits*

Bit Name	State	Definition
ServoActionStatus	FALSE	Axis is in Axis Ready state with the servo loop inactive.
DriveEnableStatus	FALSE	Axis Drive Enable output is inactive.
ShutdownStatus	FALSE	Axis is NOT in Shutdown state.

Example: When the input conditions are true, the controller transitions all axes in *group1* from the shutdown operating state to the axis ready operating state.

Relay Ladder



MGSR Ladder Example

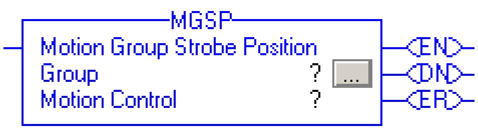
Structured Text

```
MGSR(Motion,MGSR_3);
```

Motion Group Strobe Position (MGSP)

Use the MGSP instruction to latch the current Command and Actual Position of all axes in the specified group at a single point in time. The latched positions are available in the StrobeActualPosition and StrobeCommandPosition parameters in the Motion Axis Object for each axis configured in the group.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Group	MOTION_ GROUP	tag	Name of the group of axes to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.

Structured Text

```
MGSP(Group,MotionControl);
```

The operands are the same as those for the relay ladder MGSP instruction.

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set when the group of axes have been successfully set to Shutdown state.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured group.

Description: The Motion Group Strobe Position (MGSP) instruction synchronously latches all command and actual position values of all axes in the specified group at the time of execution. The MGSP instruction takes only one parameter; simply select or enter the desired axis to strobe.

If the targeted group does not appear in the list of available groups, the group has not been configured for operation. Use the Tag Editor to create and configure a new groups.

The MGSP instruction may be used at any time to capture a complete set of command and actual position information for all axes in the specified group. This operation is often required as a precursor to computations involving position values of different axes within the group.

To successfully execute a MGSP instruction, the targeted group must be configured.

IMPORTANT	The MGSP instruction execution completes in a single scan, setting the Done .DN bit immediately.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

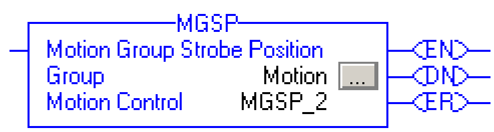
Error Codes See Error Codes (ERR) for Motion Instructions.

Status Bits: *MGSP Changes to Status Bits*

None

Example: When the input conditions are true, the controller latches the current command and the actual position of all axes in *group1*.

Relay Ladder



MGSP Ladder Example

Structured Text

```
MGSP(Motion,MGSP_2);
```

Notes:

Motion Event Instructions
(MAW, MDW, MAR, MDR, MAOC, MDOC)

ATTENTION

Tags used for the motion control attribute of instructions should only be used once. Reuse of the motion control tag in other instructions can cause unintended operation. This may result in damage to equipment or personal injury.

Introduction

Motion event instructions control the arming and disarming of special event checking functions, such as registration and watch position. The motion event instructions are:

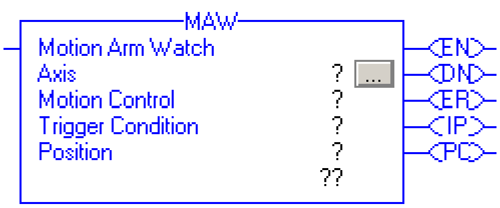
If you want to	Use this instruction	Available in these languages
Arm watch-position event-checking for an axis.	MAW	relay ladder structured text
Disarm watch-position event-checking for an axis.	MDW	relay ladder structured text
Arm servo-module registration-event checking for an axis.	MAR	relay ladder structured text
Disarm servo-module registration-event checking for an axis.	MDR	relay ladder structured text
Arm an Output Cam	MAOC	relay ladder structured text
Disarm an Output Cam	MDOC	relay ladder structured text

Motion Arm Watch (MAW)

Use the MAW instruction to arm motion module watch position event checking for the specified axis. When this instruction is called, a watch position event is enabled using the watch Position for the Axis and specified Forward or Reverse event condition. After the arming is complete the Actual Position for the Axis is monitored against the Watch Position and when the specified watch event condition is met, the Event (PC) bit is set, and the Watch Event Status bit in the Axis data structure is set.

Operands:

Relay Ladder



Operand	Type	Format	Description
Axis	AXIS_FEEDBACK AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.
Trigger condition	BOOLEAN	immediate	Select the watch-event trigger condition: 0 = forward – the servo module looks for the actual position to change from less than the watch position to greater than the watch position. 1 = reverse – the servo module looks for the actual position to change from greater than the watch position to less than the watch position.
Position	REAL	immediate or tag	The new value for the watch position.

Structured Text

```
MAW(Axis,MotionControl,
TriggerCondition,Position);
```

The operands are the same as those for the relay ladder MAW instruction.

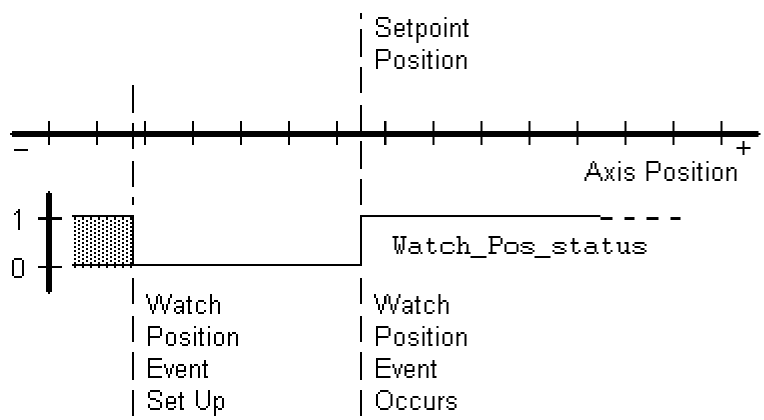
For the operands that require you to select from available options, enter your selection as:

This operand	Has these options which you...	
	enter as text	or enter as a number
TriggerCondition	forward	0
	reverse	1

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set when the axis watch event checking has been successfully armed.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.
.IP (In Process) Bit 26	It is set on positive rung transition and cleared after the watch event has occurred, or has been superseded by another Motion Arm Watch, or terminated by a Motion Disarm Watch command.
.PC (Process Complete) Bit 27	It is set when a watch event occurs.

Description: The Motion Arm Watch (MAW) instruction sets up a Watch Position event to occur when the specified physical axis reaches the specified Set-point position, as shown below.



Set Point Position

Watch Position events are useful for synchronizing an operation to a specified axis position while the axis is moving, such as activating a solenoid to push a carton off a conveyor at a certain axis position. Select or enter the desired physical axis, the desired Trigger Condition, and enter a value or tag variable for the desired Watch Position.

If the targeted axis does not appear in the list of available axes, the axis has not been configured for operation. Use the Tag Editor to create and configure a new axis.

When an Arm Watch Position instruction is executed, the WatchEventStatus bit is set to 0 (FALSE) and the actual position of a physical axis is monitored (at the servo loop update rate) until it reaches the specified Watch Position. After the watch position event occurs, the WatchEventStatus bit for the axis is set to 1 (TRUE).

Multiple watch position events may be active at a given time, however only one may be active at a time for any given physical axis. Each event is monitored independently and may be checked using the appropriate WatchEventStatus bit.

IMPORTANT	In large I/O connections, force values can slow down the rate at which the controller processes repetitive watch positions.

To successfully execute a MAW instruction, the targeted axis must be configured as either a Servo or Feedback Only axis. Otherwise, the instruction errs.

IMPORTANT	The MAW instruction execution may take multiple scans to execute because it requires transmission of a message to the motion module. The Done (.DN) bit is not set immediately, but only after this message has been successfully transmitted.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes See Error Codes (ERR) for Motion Instructions.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions. The following Extended Error codes help to pinpoint the problem when the MAW instruction receives a Servo Message Failure (12) error message.

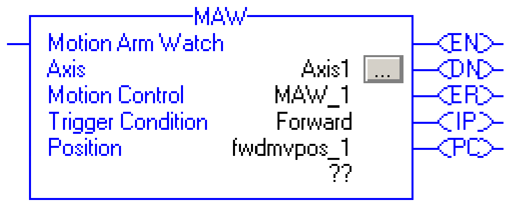
Associated Error Code (decimal)	Extended Error Code (decimal)	Meaning
SERVO_MESSAGE_FAILURE (12)	No Resource (2)	Not enough memory resources to complete request. (SERCOS)

Status Bits: *MAW Changes to Status Bits*

Bit Name	State	Meaning
WatchEventArmedStatus	TRUE	The axis is looking for a watch position event.
WatchEventStatus	FALSE	The previous watch event is cleared.

Example: When the input conditions are true, the controller arms watch-position event-checking for *axis1*.

Relay Ladder



MAW Ladder Example

Structured Text

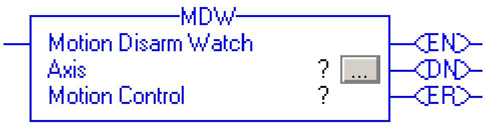
```
MAW(Axis1,MAW_1,Forward,fwdmvpos_1);
```

Motion Disarm Watch (MDW)

Use the MDW instruction to disarm watch-position event-checking for an axis. This instruction has the affect of clearing both the Watch Event Status and Watch Armed Status bits in the axis data structure. Executing this instruction also clears the In Process bit associated with the controlling Motion Arm Watch (MAW) instruction.

Operands:

Relay Ladder



Operand	Type	Format	Description
Axis	AXIS_FEEDBACK AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.

Structured Text

```
MDW(Axis,MotionControl);
```

The operands are the same as those for the relay ladder MDW instruction.

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set when axis watch event checking has been successfully disarmed.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.

Description:

The Motion Disarm Watch (MDW) instruction cancels watch position event checking set up by a previous MAW instruction. The Disarm Watch Position instruction requires no parameters; simply enter or select the desired physical axis.

If the targeted axis does not appear in the list of available axes, the axis has not been configured for operation. Use the Tag Editor to create and configure a new axis.

To successfully execute a MDW instruction, the targeted axis must be configured as either a Servo or Feedback Only axis. Otherwise, the instruction errs.

IMPORTANT The MDW instruction execution may take multiple scans to execute because it requires transmission of a message to the motion module. The Done (.DN) bit is not set immediately, but only after this message has been successfully transmitted.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

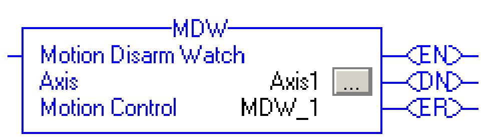
Error Codes See Error Codes (ERR) for Motion Instructions.

Status Bits: *MDW Changes to Status Bits*

Bit Name	State	Meaning
WatchEventArmedStatus	FALSE	The axis is not looking for a watch position event.
WatchEventStatus	FALSE	The previous watch event is cleared.

Example: When the input conditions are true, the controller disarms watch-position event-checking for *axis1*.

Relay Ladder



MDW Ladder Example

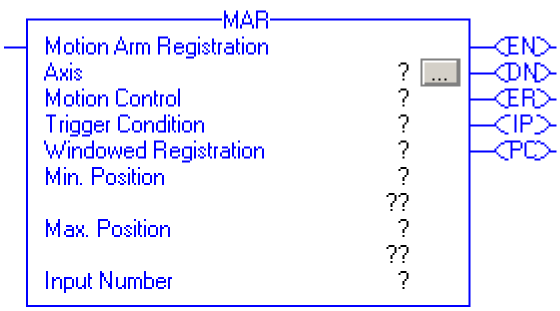
Structured Text

```
MDW(Aaxis1,MDW_1);
```

Motion Arm Registration (MAR)

Use the MAR instruction to arm servo module registration event checking for the specified axis. When the instruction is called, a registration event is armed based on the selected Registration Input and the specified Trigger Condition. When the specified Registration Input transition satisfies the Trigger Condition, the motion module computes the axis position at the moment the event occurred based on hardware latched encoder count data and stores it in the associated Registration Position variable in the axis data structure. Also, the instruction's Event (PC) bit is simultaneously set, as well as the Registration Event Status bit in the axis data structure. If Windowed Registration is selected, only registration events whose computed registration position falls within the Max and Min Position window are accepted. If the Registration Position falls outside this window the registration event checking is automatically rearmed.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Axis	AXIS_FEEDBACK AXIS_VIRTUAL AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.
Trigger condition	BOOLEAN	immediate	Defines the Registration Input transition that defines the registration event. Select either: 0 = trigger on positive edge 1 = trigger on negative edge
Windowed registration	BOOLEAN	immediate	Set (1) if registration is to be Windowed meaning that the computed Registration Position must fall within the specified Min and Max Position limits to be accepted as a valid registration event. Select either: 0 = disabled 1 = enabled

Operand	Type	Format	Description
Minimum position	REAL	immediate or tag	Used when Windowed Registration is enabled. Registration Position must be greater than Min. Position limit before registration event is accepted.
Maximum position	REAL	immediate or tag	Used when Windowed Registration is enabled. Registration Position must be less than Max. Position limit before registration event is accepted.
Input Number	UINT32	1 or 2	Specifies the Registration Input to select. 1 = Registration 1 Position 2 = Registration 2 Position

Structured Text

```
MAR(Axis,MotionControl,
TriggerCondition,
WindowedRegistration,
MinimumPosition,
MaximumPosition,
InputNumber);
```

The operands are the same as those for the relay ladder MAR instruction.

For the operands that require you to select from available options, enter your selection as:

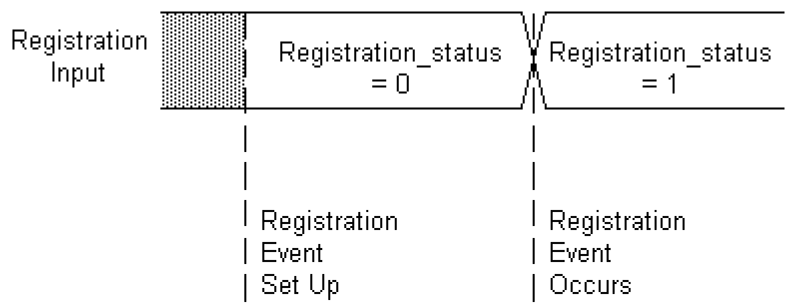
This operand	Has these options which you...	
	enter as text	or enter as a number:
TriggerCondition	positive_edge negative_edge	0 1
WindowedRegistration	disabled enabled	0 1

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set when the axis registration event checking has been successfully armed.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.
.IP (In Process) Bit 26	It is set on positive rung transition and cleared after the registration event has occurred, or has been superseded by another Motion Arm Reg command, or terminated by a Motion Disarm Reg command.
.PC (Process Complete) Bit 27	It is set when a registration event occurs.

Description: The Motion Arm Registration (MAR) instruction sets up a registration event to store the actual positions of the specified physical axis on the specified edge of the selected dedicated high speed Registration input for that axis.

When an MAR instruction is executed, the RegEventStatus bit is set to 0 (FALSE) and the selected Registration input for the specified axis is monitored by the motion module until a Registration input transition of the selected type (the *registration event*) occurs. When the registration event occurs, the RegEventStatus bit for the axis is set to 1 (TRUE) and the Actual Position of the axis is stored in the Registration Position variable corresponding to the registration input (e.g. Registration 1 Position 1 or Registration 2 Position).

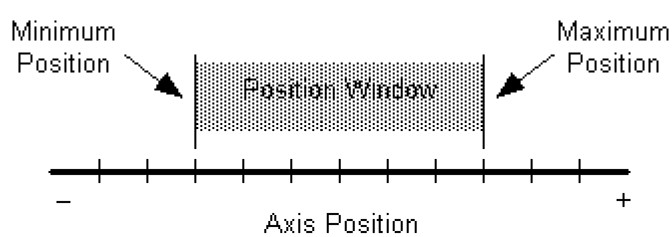


Registration

Multiple registration events may be active at any time for a given axis, but only one may be active per registration input. Each event is monitored independently and may be checked using the appropriate RegEventStatus bit.

Windowed Registration

When the Windowed Reg checkbox is checked, the selected trip state only results in a registration event if it occurs when the axis is within the window defined by the minimum and maximum positions as shown below.

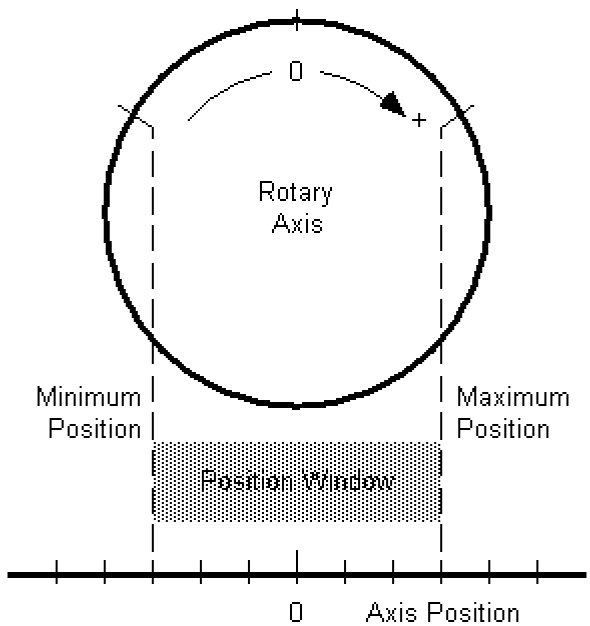


Windowed Registration

Enter values or tag variables for the desired absolute positions that define the position window within which the selected trip state of the

Registration input is valid. Windowed registration is useful in providing a mechanism to ignore spurious or random transitions of the registration sensor, thus improving the noise immunity of high-speed registration inputs.

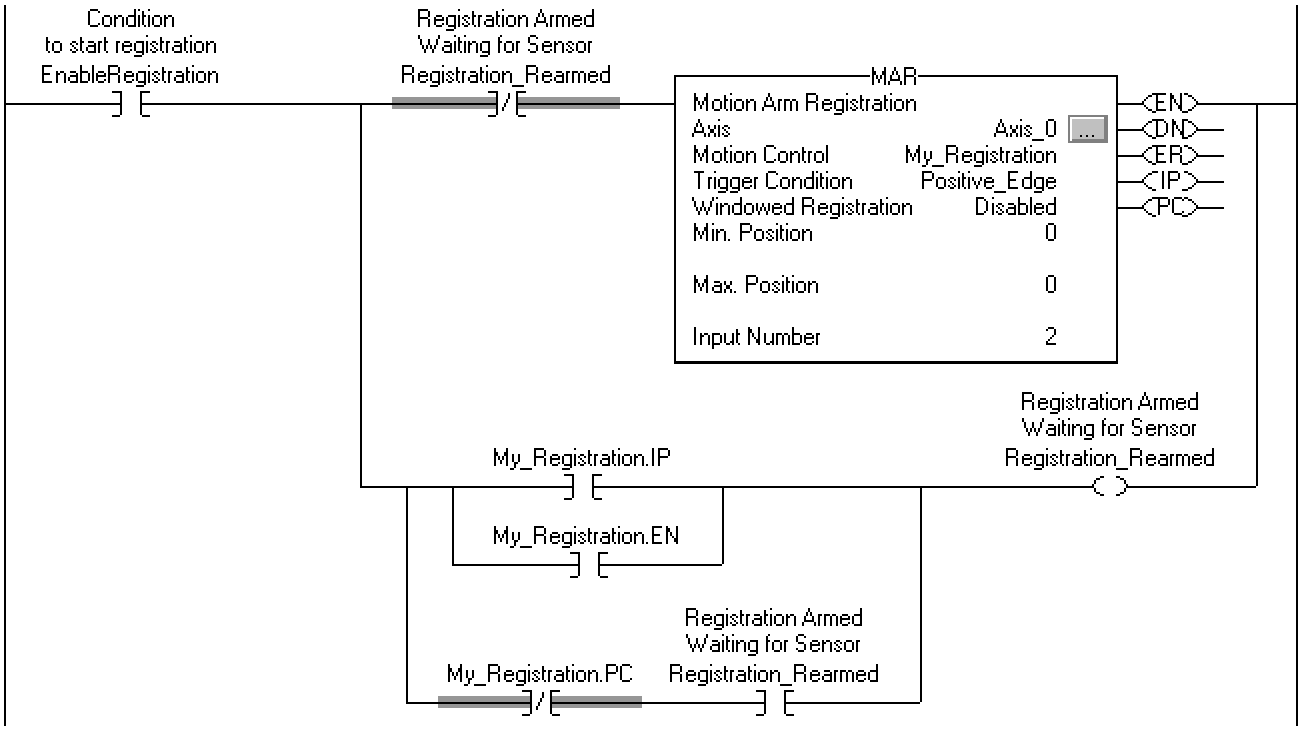
For linear axes, the values can be positive, negative, or a combination. However, the Minimum Position value must be less than the Maximum Position value for the registration event to occur. For rotary axes, both values must be less than the unwind value set in the motion controller's machine setup menu. The Minimum Position value can be greater than the Maximum Position value for registration windows that cross the unwind point of the axis, as shown below.



Position Window for Rotary Axis

Rearming an MAR Instruction

If your application requires rapid and continuous detection of a registration sensor, we recommend that you use the following logic:



Ladder Logic for Continuous Registration Detection

To rearm the MAR instruction, the rung must change from false to true. The rate at which this logic functions depends on the following:

- program scan time
- motion task course update rate

IMPORTANT	In large I/O connections, force values can slow down the rate at which the controller processes repetitive motion registration.

To successfully execute a MAR instruction, the targeted axis must be configured as either a Servo or Feedback Only axis. Otherwise, the instruction errs.

IMPORTANT	The MAR instruction execution may take multiple scans to execute because it requires transmission of a message to the motion module. The Done (.DN) bit is not set immediately, but only after this message has been successfully transmitted.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes See Error Codes (ERR) for Motion Instructions.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions. The following Extended Error codes help to pinpoint the problem when the MAR instruction receives a Servo Message Failure (12) error message.

Associated Error Code (decimal)	Extended Error Code (decimal)	Meaning
SERVO_MESSAGE_FAILURE (12)	No Resource (2)	Not enough memory resources to complete request. (SERCOS)
SERVO_MESSAGE_FAILURE (12)	Invalid value (3)	Registration input provided is out of range.
SERVO_MESSAGE_FAILURE (12)	Device in wrong state (16).	Redefine Position, Home, and Registration 2 are mutually exclusive. (SERCOS)

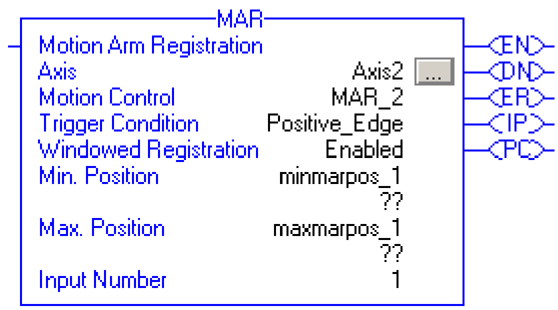
Extended Error codes for the Parameter Out of Range (13) error code work a little differently. Rather than having a standard enumeration, the number that appears for the Extended Error code refers to the number of the operand as they are listed in the faceplate from top to bottom with the first operand being counted as zero. Therefore for the MAR instruction, an extended error code of 4 would refer to the Min Position value. You would then have to check your value with the accepted range of values for the instruction.

Status Bits: *MAR Changes to Status Bits*

Bit Name	State	Meaning
RegEventArmedStatus	True	The axis is looking for a registration event.
RegEventStatus	False	The previous registration event is cleared.

Example: When the input conditions are true, the controller arms servo-module registration-event checking for *axis_0*.

Relay Ladder



MAR Ladder Example

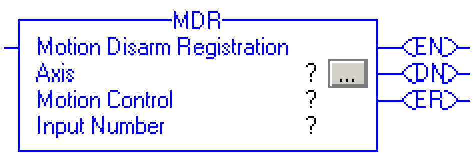
Structured Text

```
MAR(Axis2,MAR_2,positive_edge,enabled,minmarpos_1,
     maxmarpos_1,1;
```

Motion Disarm Registration (MDR)

Use the MDR instruction to disarm the specified motion module registration input event checking for the specified axis. This instruction has the affect of clearing both the RegEventStatus and the RegArmedEventStatus bits. The In Process bit of the controlling Motion Arm Registration instruction, if any, is cleared as a result of executing the MDR instruction.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Axis	AXIS_FEEDBACK AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.
Input Number	UINT32	1 or 2	Specifies the Registration Input to select. 1 = Registration 1 Position 2 = Registration 2 Position

Structured Text

```
MDR(Axis,MotionControl,
InputNumber);
```

The operands are the same as those for the relay ladder MDR instruction.

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set when axis watch event checking has been successfully disarmed.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.

Description: The Motion Disarm Registration (MDR) instruction cancels registration event checking established by a previous Motion Arm Registration instruction. Only the registration checking associated with the specified registration input is disabled.

If the targeted axis does not appear in the list of available axes, the axis has not been configured for operation. Use the Tag Editor to create and configure a new axis.

To successfully execute a MDR instruction, the targeted axis must be configured as either a Servo or Feedback Only axis. Otherwise, the instruction errs.

IMPORTANT The MDR instruction execution may take multiple scans to execute because it requires transmission of a message to the motion module. The Done (.DN) bit is not set immediately, but only after this message has been successfully transmitted.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes See Error Codes (ERR) for Motion Instructions.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions. The following Extended Error codes help to pinpoint the problem when the MDR instruction receives a Servo Message Failure (12) error message.

Associated Error Code (decimal)	Extended Error Code (decimal)	Meaning
SERVO_MESSAGE_FAILURE (12)	Invalid value (3)	Registration input provided is out of range.

Extended Error codes for the Parameter Out of Range (13) error code work a little differently. Rather than having a standard enumeration, the number that appears for the Extended Error code refers to the number of the operand as they are listed in the faceplate from top to bottom with the first operand being counted as zero. Therefore for the MDR instruction, an extended error code of 2 would refer to the Input Number operand's value. You would then have to check your value with the accepted range of values for the instruction.

Status Bits: *MDR Changes to Status Bits*

Bit Name	State	Meaning
RegEventArmedStatus	FALSE	The axis is not looking for a registration event.
RegEventStatus	FALSE	The previous registration event is cleared.

Example: When the input conditions are true, the controller disarms registration-event checking for *axis_0*.

Relay Ladder

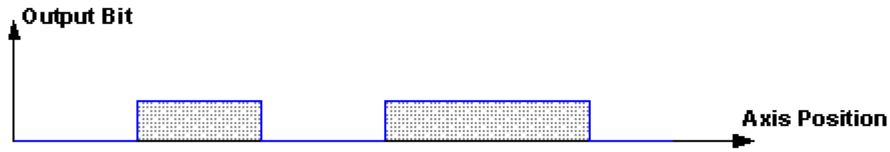
MDR Ladder Example

Structured Text

```
MRD(Axis2,MDR_1,2);
```

Motion Arm Output Cam (MAOC)

The Motion Planner Output Cam functionality provides setting and resetting of output bits based on an axis position.



Motion Planner Functionality

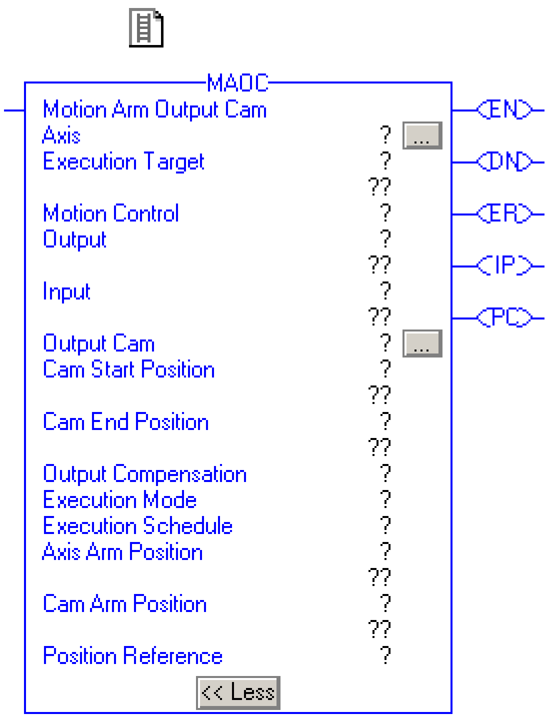
Internally, Output Cam objects handle the Motion Planner Output Cam functionality. Each Output Cam object is responsible for one output, which consists of 32 output bits. Each single output bit can be programmed separately with an Output Cam profile, and compensated for position offset and time delay.

The Motion Arm Output Cam (MAOC) instruction initiates the arming of a specific Output Cam between the designated axis and output. When executed, the specified output cam bits are synchronized to the designated axis using an Output Cam Profile established by the RSLogix 5000 Output Cam Editor. This relationship can be viewed as a master/slave relationship with the axis representing the master and the output bit representing the slave. Hence, the Output Cam functionality is related to the position cam functionality, which provides a relationship between a master axis and a slave axis. To accurately synchronize the output cams to the designated axis, an execution schedule and associated axis and cam arm positions are specified. When the axis travels past the axis arm position in the direction specified by the Execution Schedule parameter, the cam position becomes locked to the axis position starting at the specified Cam Arm Position parameter. At this time the output cam is armed and the Output Cam Armed status is set. The output cam can also be configured via the Execution Schedule parameter to execute Immediately or Pending completion of a currently executing output cam. The output cam can also be executed Once, Continuously or Persistently by specifying the desired Execution Mode. Persistent behavior allows the output cam to become disarmed when the cam position exceeds the output cam range, and rearmed when cam position returns to within range. Output Cam range is defined by input parameters CamStartPosition and CamEndPosition. The Master Reference selection allows axis input to be derived from either the Actual or Commanded position of the designated axis.

ATTENTION

Output cams increase the potential for exceeding coarse update rate. This can cause misbehavior if the motion task execution time exceeds the configured group coarse update period. The only way to check on this condition is to monitor the max execution time from the Motion Group Properties page.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Axis	AXIS_FEEDBACK AXIS_CONSUMED AXIS_VIRTUAL AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis that provides the position input to the Output Cam. Ellipsis launches Axis Properties dialog.
Execution Target	UINT32	immediate or tag	The execution target defines the specific Output Cam from the set connected to the named axis. Behavior is determined by the following: • 0…8 – Output Cams executed in the Logix controller. • 9…31 – Reserved for future use.
Motion Control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.
Output	DINT	tag	A set of 32 output bits that are set or reset based on the specified Output Cam. It can be either a memory location or a physical output. If Pending is selected as the Execution Schedule, then Output is ignored.
Input	DINT	tag	A set of 32 input bits that can be used as enable bits depending on the specified Output Cam. It can be either a memory location or a physical input. If Pending is selected as the Execution Schedule, then Input is ignored.
Output Cam	OUTPUT_CAM	array tag	An array of OUTPUT_CAM elements. The elements do not need to be ordered and the array size is determined by the number of cam elements specified. The array size is limited by the available memory of the Logix controller.

Operand	Type	Format	Description
Cam Start Position	SINT, INT, DINT, or REAL	immediate or tag	Cam Start Position with the Cam End Position define the left and right boundaries of the Output Cam range.
Cam End Position	SINT, INT, DINT or REAL	immediate or tag	Cam End Position with the Cam Start Position define the left and right boundaries of the Output Cam range.
Output Compensation	OUTPUT_ COMPENSATION	array tag	Is an array of 1 to 32 OUTPUT_COMPENSATION elements. The array indices correspond to the output bit numbers. The minimum size of an array is determined by the highest compensated output bit.
Execution Mode	UINT32	immediate	There are three (3) possible execution modes. The behavior is determined by the mode selected. The options are: 0 = Once – Output Cam is disarmed and the Process Complete Bit of the Motion Instruction is set when the cam position moves beyond the cam start or the cam end position. 1 = Continuous – Output Cam continues on the opposite side of the Output Cam range when the cam position moves beyond the cam start or the cam end position. 2 = Persistent – Output Cam disarms when the cam position moves beyond the cam start or the cam end position. The Output Cam is rearmed when the cam position moves back into the Output Cam range.

Operand	Type	Format	Description
Execution Schedule	UINT32	immediate	Selects when to arm the Output Cam. Options are: 0 = Immediate – Output Cam is armed at once. 1 = Pending – Output cam is armed when the cam position of a currently executing Output Cam moves beyond its cam start or cam end position. When Pending is selected the following parameters are ignored Output, Input, Axis Arm Position, and Reference. 2 = Forward only – Output Cam is armed when the axis approaches or passes through the specified axis arm position in the forward direction. 3 = Reverse only – Output Cam is armed when the axis approaches or passes through the specified axis arm position in the reverse direction. 4 = Bi-directional – Output Cam is armed when the axis approaches or passes through the specified axis arm position in either direction.

Operand	Type	Format	Description
Axis Arm Position	SINT, INT, DINT, or REAL	immediate or tag	This defines the axis position where the Output Cam is armed when the Execution Schedule is set to Forward Only, Reverse Only, or Bi-Directional and the axis moves in the specified direction. If Pending is selected as the Execution Schedule, then Axis Arm Position is ignored.
Cam Arm Position	SINT, INT, DINT, or REAL	immediate or tag	This defines the cam position associated with the axis arm position when the Output Cam is armed.
Reference	UINT32	immediate	Sets whether the Output Cam is connected to either Command position or Actual position of the axis. If Pending is selected as the Execution Schedule, then Reference is ignored. 0 = Actual – the current position of the axis as measured by its encoder or other feedback device. 1 = Command – the desired or commanded position of the master axis.

Structured Text

```
MAOC(Axis,ExecutionTarget,
MotionControl,Output,Input,
OutputCam,CamStartPosition,
CamEndPosition,
OutputCompensation,
ExecutionMode,
ExecutionSchedule,
AxisArmPosition,
CamArmPosition,Reference);
```

The operands are the same as those for the relay ladder MAOC instruction. For the array operands, you do not have to include the array index. If you do not include the index, the instruction starts with the first element in the array ([0]).

For the operands that require you to select from available options, enter your selection as:

This operand	Has these options which you...	
	enter as text	or enter as a number
ExecutionMode	once	0
	continuous	1
	persistent	2
ExecutionSchedule	immediate	0
	pending	1
	forwardonly	2
	reverseonly	3
	bidirectional	4
Reference	actual	0
	command	1

MAOC Instruction

A valid Cam Arm position is any position, between and including, the Cam Start and Cam End positions. If the Cam Arm position is set to a value equal to (or very close to) the Cam Start or Cam End position, compensation may put a cam position out of range of the Cam Start and Cam End position. Compensation is affected by Output Compensation values specified for Position Offset, Latch Delay, and Unlatch Delay, as well as internal compensation values applied based on the Reference and Output parameters of the MAOC instruction.

No side affects occur if the MAOC instruction is configured with an Execution mode of “Continuous” or “Persistent”, and a pending MAOC instruction does not exist when the Output Cam is armed and the axis moves.

The following side affects may occur of the MAOC instruction is configured with an Execution Mode of "Once Only", and a pending MAOC exists when the Output Cam is armed and the axis moves.

- One or more outputs may never change state.
- The MAOC instruction may complete immediately.

One possible side affect of a pending MAOC instruction existing when the Output Cam is armed and the axis moves is that one or more outputs could begin executing based on the configuration of the pending MAOC instruction.

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the rung goes false.
.DN (Done) Bit 29	It is set when Output Cam has been successfully initiated.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.
.IP (In Process) Bit 26	It is set when the Output Cam has been initiated successfully and cleared if either superseded by another Motion Arm Output Cam command, terminated by a Motion Disarm Output Cam command, or cam position moves beyond defined Output Cam range while execution mode is set to ‘once’.
.PC (Process Complete) Bit 27	It is cleared on positive rung transition and set in ‘once’ Execution Mode when cam position moves beyond defined Output Cam range.
.SEGMENT	It is set to the array index associated with error 36 (Illegal Output Cam) or error 37 (Illegal Output Compensation). Only the first of multiple errors is stored.

Description: The Motion Arm Output Cam (MAOC) instruction executes an output cam profile set up manually, programmatically, or by the RSLogix 5000 Output Cam Editor. Internally, Output Cam objects handle Motion Planner Cam functionality. Each Output Cam object is responsible for one output, which consists of 32 output bits. Each single output bit can be programmed separately. Currently Output Cam functionality is executed in the Logix controller every course update period (currently configurable between 1 and 32 ms).

Axis

The axis provides the position input to the Output Cam. The axis can be a virtual, physical or consumed one.

Execution Target

The execution target defines a specific Output Cam from the set that is connected to the specified axis. Currently, only eight Output Cams can be specified.

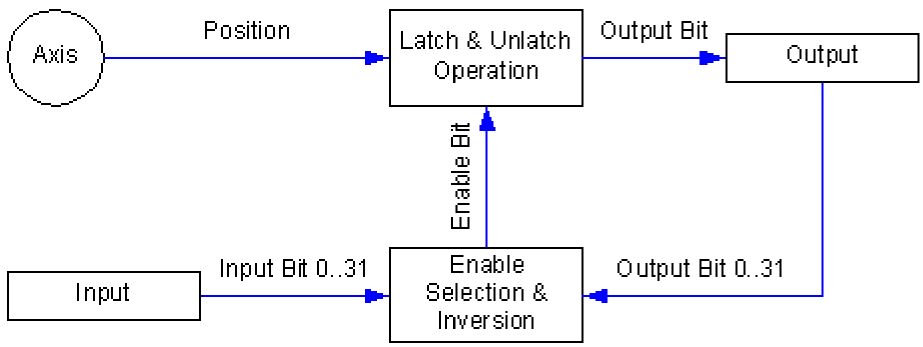
Specifying the Output Cam Profile

To execute a MAOC instruction, a calculated Output Cam data array tag must be specified. Output Cam array tags may be created by the RSLogix 5000 tag editor or the MAOC instruction using the built-in

Output Cam Editor. The data defines the specifics for each Output Cam element. The number of Output Cam elements is limited by the amount of available memory. Zero or more cams can be defined for each output bit. There is no constraint on how these elements are arranged within the Output Cam array.

Refer to the description of the OUTPUT_CAM structure for more information about data types and programming units.

The following diagram shows the relationships between the axis, input, and output that are defined by the Output Cam element.



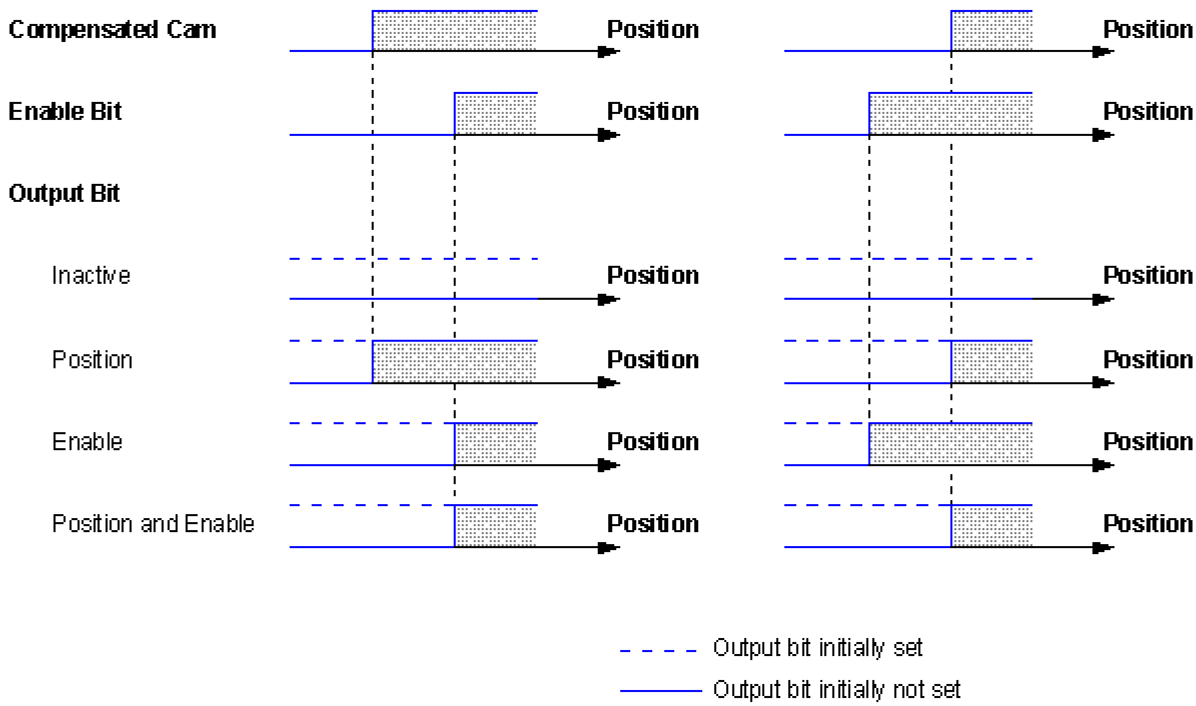
Output Cam Element Relationships

Latch Type

Depending on the selected LatchType, the corresponding output bit is set according to the following table.

Latch Type	Behavior
Inactive	The output bit is not changed.
Position	The output bit is set, when the axis enters the compensated cam range.
Enable	The output bit is set, when the enable bit becomes active.
Position and Enable	The output bit is set, when the axis enters the compensated cam range and the enable bit becomes active.

The following diagram shows the effect of the selected latch type on the output bit for different compensated cam and enable bit combinations as function of position.



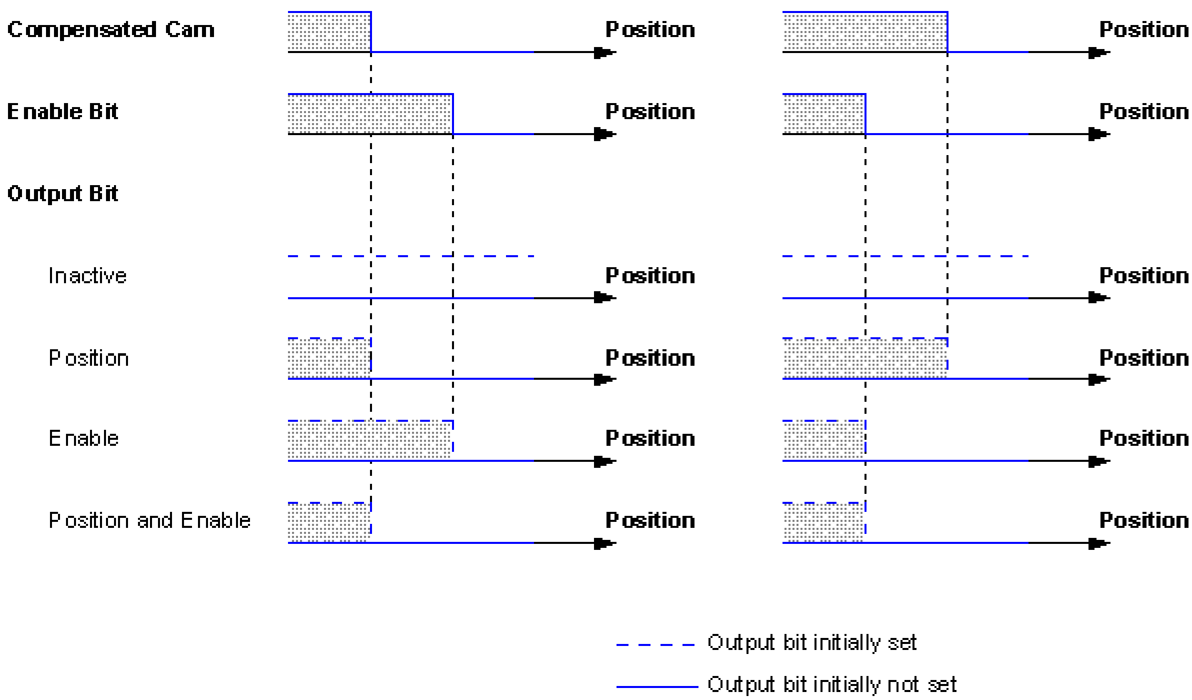
Latched Position

Unlatch Type

Depending on the selected UnlatchType, the corresponding output bit is reset according to the following table.

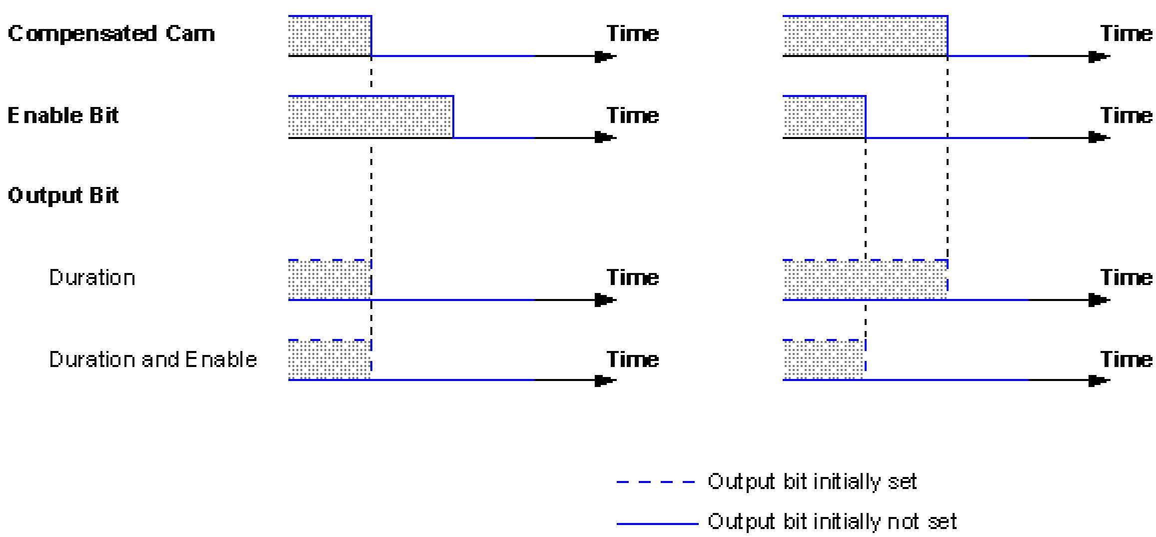
Unlatch Type	Behavior
Inactive	The output bit is not changed.
Position	The output bit is reset, when the axis leaves the compensated cam range.
Duration	The output bit is reset, when the duration expires.
Enable	The output bit is reset, when the enable bit becomes inactive.
Position and Enable	The output bit is reset, when the axis leaves the compensated cam range or the enable bit becomes inactive.
Duration and Enable	The output bit is reset, when the duration expires or the enable bit becomes inactive.

The following diagram shows the effect of the selected unlatch type on the output bit for different compensated cam and enable bit combinations as function of position.



Unlatch as Function of Position

and as function of time.



Unlatch as a Function of Time

Left and Right Cam Positions

The Left and Right cam positions define the range of an Output Cam element. If the latch or unlatch type is set to "Position" or "Position and Enable" with the enable bit active, the left and right cam positions specify the latch or unlatch position of the output bit.

Duration

If the unlatch type is set to "Duration" or "Duration and Enable" with the enable bit active, the cam duration specifies the time between the latching and the unlatching of the output bit.

Enable Type

Depending on the selected enable type, the enable bit is an element of either the input, inverted input, output, or inverted output.

Output Cam Array Checks

If you select an output bit less than 0 or greater than 31, the Output Cam element is not considered and the user is warned with an instruction error "Illegal Output Cam".

If you select a latch type less than 0 or greater than 3, a value of "Inactive" is used and the user is warned with an instruction error "Illegal Output Cam".

If you select an unlatch type less than 0 or greater than 5, a value of "Inactive" is used and the user is warned with an instruction error "Illegal Output Cam".

If you select a left cam position greater than or equal to the right cam position and the latch or unlatch type is set to "Position" or "Position and Enable", the Output Cam element is not considered and the user is warned with an instruction error "Illegal Output Cam".

If you select a left cam position less than the cam start position and the latch type is set to "Position" or "Position and Enable", the cam start position is used and the user is warned with an instruction error "Illegal Output Cam".

If you select a right cam position greater than the cam end position and the unlatch type is set to "Position" or "Position and Enable", the cam end position is used and the user is warned with an instruction error "Illegal Output Cam".

If you select a duration less than or equal to 0 and the unlatch type is set to "Duration" or "Duration and Enable", the Output Cam element is not considered and the user is warned with an instruction error "Illegal Output Cam".

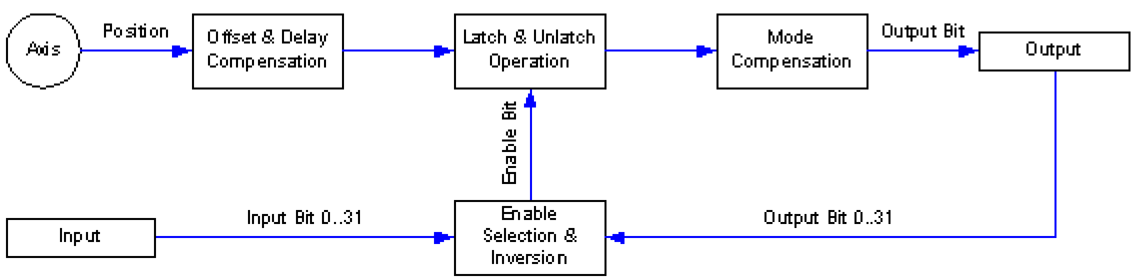
If you select an enable type less than 0 or greater than 3 and the latch or unlatch type is set to "Enable", "Position and Enable", or "Duration and Enable", the Output Cam element is not considered and the user is warned with an instruction error "Illegal Output Cam".

If you select an enable bit less than 0 or greater than 31 and the latch or unlatch type is set to "Enable", "Position and Enable", or "Duration and Enable", the Output Cam element is not considered and the user is warned with an instruction error "Illegal Output Cam".

Specifying Output Compensation

An Output Compensation data array tag may be specified via the RSLogix 5000 tag editor. The data type defines the specifics for each output bit by specifying the characteristics of each actuator. The array indices correspond to the output bit numbers. The number of the highest compensated output bit defines the minimum size of this array. Changes to the output compensation take effect immediately.

The following diagram shows the effect of the output compensation on the relationships between the axis, input, and output.

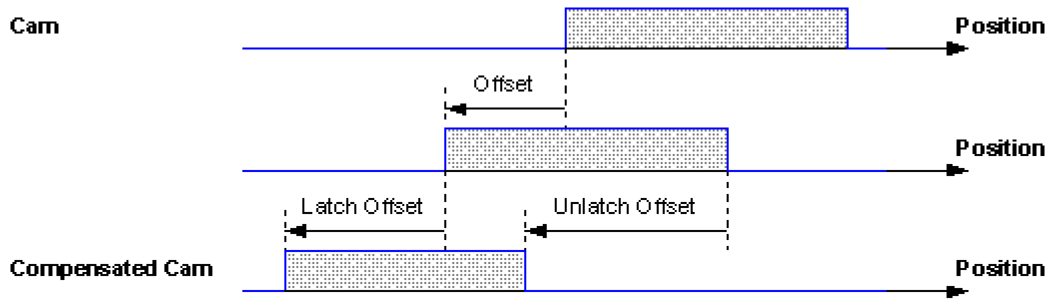


Output Compensation

Refer to the description of the OUTPUT_COMPENSATION structure for more information on data types and programming units.

Offset and Delay Compensation

The offset provides position compensation, while the latch and unlatch delay provides time delay compensation for the latch and unlatch operation. The following diagram shows the effect of the compensation values on an Output Cam element.



Offset and Delay Compensation

The cam range is defined by the left and right cam positions of the Output Cam element. The compensated cam range is defined by the cam range, offset, and latch and unlatch offsets. The latch and unlatch offsets are defined by the current speed *v*:

Latch Offset = *v* * Latch Delay

Unlatch Offset = *v* * Unlatch Delay

The resulting compensation offset can actually be larger than the difference between cam start and cam end position.

The following equation illustrates the effect of the compensation values on the duration of an Output Cam element.

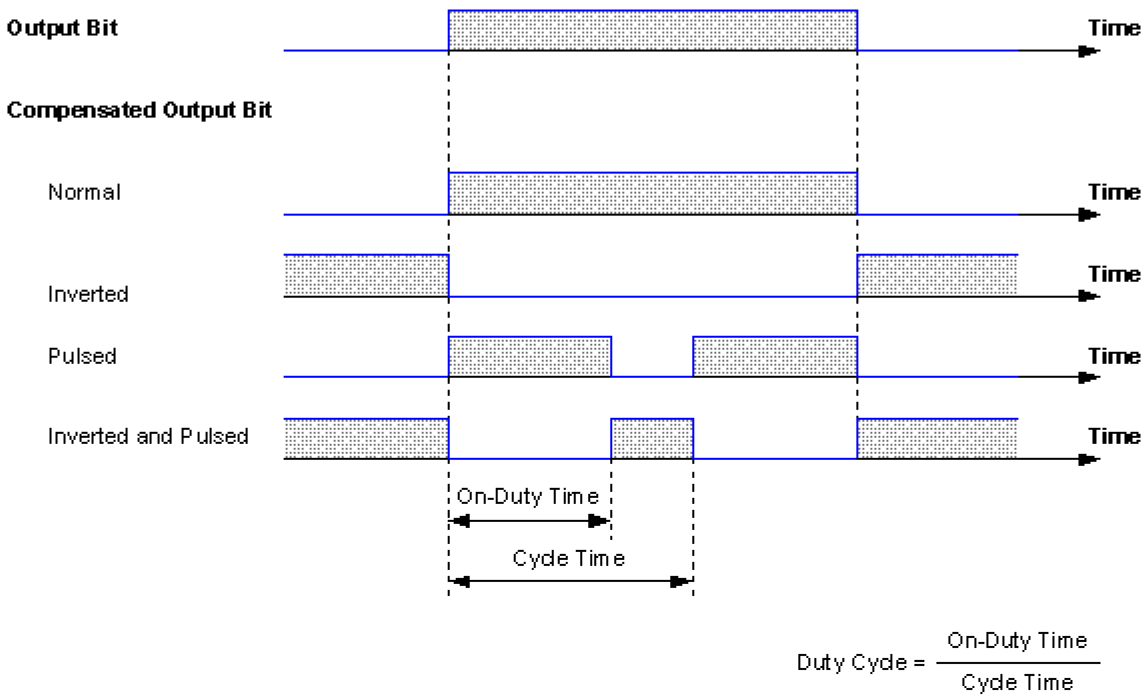
Compensated Duration = Duration + Latch Delay - Unlatch Delay

Mode Compensation

Depending on the selected mode, the compensated output bit is set according to the following table.

Mode	Behavior
Normal	The output bit is set, when the output of the latch and unlatch operation becomes active. The output bit is reset, when the output of the latch and unlatch operation becomes inactive.
Inverted	The output bit is set, when the output of the latch and unlatch operation becomes inactive. The output bit is reset, when the output of the latch and unlatch operation becomes active.
Pulsed	The output bit is pulsed, when the output of the latch and unlatch operation is active. The on-duty state of the pulse corresponds to the active state of the output bit. The output bit is reset, when the output of the latch and unlatch operation becomes inactive.
Inverted and Pulsed	The output bit is pulsed, when the output of the latch and unlatch operation is active. The on-duty state of the pulse corresponds to the inactive state of the output bit. The output bit is set, when the output of the latch and unlatch operation becomes inactive.

The following diagram shows the effect of the mode, cycle time, and duty cycle on an output bit.



Mode Compensation

Output Compensation Array Checks

If you select a latch and unlatch delay combination that results in a compensated cam of less than minimum width, the width of the compensated cam is set to the minimum.

If you select a mode less than 0 or greater than 3, a “Normal” mode is considered and the user is warned with an instruction error “Illegal Output Compensation”.

If you select a duty cycle less than 0 or greater than 100 and the mode is set to “Pulsed” or “Inverted and Pulsed”, a 0 or 100 duty cycle is considered and the user is warned with an instruction error “Illegal Output Compensation”.

If you select a cycle time less than or equal to 0 and the mode is set to “Pulsed” or “Inverted and Pulsed”, the output bit is not pulsed and the user is warned with an instruction error “Illegal Output Compensation”.

Output

The output is the set of 32 output bits that can be set and reset depending on the specified Output Cam. The output can be either a memory location or a physical output (e.g. “Local.0.O.Data”).

Input

The input is the set of 32 input bits that are can be used as enable bits depending on the specified Output Cam. The input can be either a memory location or a physical input (e.g. “Local.0.I.Data”).

Cam Start and Cam End Positions

The cam start and cam end positions define the left and right boundary of the Output Cam range. When the cam position moves beyond the cam start or cam end position, the behavior of the Output Cam is defined by the execution mode and execution schedule. Changes to the cam start or cam end position don’t take effect until the execution of a current MAOC instruction completes.

Execution Mode

Depending on the selected execution mode, the Output Cam behavior may differ, when the cam position moves beyond the cam start or cam end position.

Execution mode	Behavior
Once	When the cam position moves beyond the cam start or cam end position, the Output Cam is disarmed and the Process Complete bit of the Motion Instruction is set.
Persistent	When the cam position moves beyond the cam start or cam end position, the Output Cam is disarmed. However, when the cam position moves back into the Output Cam range the Output Cam is rearmed.
Continuous	When the cam position moves beyond the cam start or cam end position, the Output Cam continues on the opposite side of the Output Cam range.

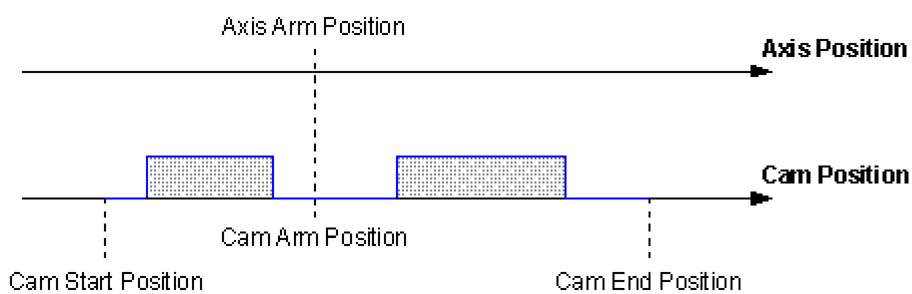
Execution Schedule

Depending on the selected execution schedule, the Output Cam is armed according to the following table.

Execution Schedule	Behavior
Immediate	The Output Cam is armed immediately.
Pending	The Output Cam is armed, when the cam position of an armed Output Cam moves beyond its cam start or cam end position.
Forward Only	The Output Cam is armed, when the axis approaches or passes through the specified axis arm position in the forward direction.
Reverse Only	The Output Cam is armed, when the axis approaches or passes through the specified axis arm position in the reverse direction.
Bi-Directional	The Output Cam is armed, when the axis approaches or passes through the specified axis arm position in the forward or reverse direction.

Axis Arm and Cam Arm Positions

The axis arm position defines the axis position where the Output Cam is armed, if the execution schedule is set to either forward only, reverse only, or bi-directional and the axis moves in the specified direction. The cam arm position defines the cam position that is associated with the axis arm position, when the Output Cam is armed. Changes to the axis arm or cam arm position only take effect after the execution of an MAOC instruction.



Axis Arm and Cam Arm Positions

Reference

Depending on the selected reference, the Output Cam is connected to either the actual or command position of the axis.

IMPORTANT The MAOC instruction execution completes in a single scan, thus the Done (.DN) bit and the In Process .IP bit are set immediately. The In Process .IP bit remains set until the cam position moves beyond the cam start or cam end position in "Once" execution mode, is superseded by another MAOC instruction or is disarmed by the MDOC instruction. The Process Complete bit is cleared immediately when the MAOC executes and set when the cam position moves beyond the cam start or cam end position in "Once" execution mode.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes: *MAOC Error Codes (.ERR)*

Error Message	Code	Description
Execution Collision	3	Attempted execution with another Output Cam currently in process.
Shutdown State Error	7	Attempted execution with the axis in the Shutdown state.
Axis Not Configured	11	Passed axis value references an unconfigured axis meaning the axis has not been assigned to a physical motion module channel.
Value Out of Range	13	Attempted execution with an input parameter that was out of range. 1. Cam start position ≥ cam end position. 2. Cam arm position outside of Output Cam range. See Extended Error section for more information on the cause of the error.
Homing in Process Error	16	Attempted execution with a homing process in progress.
Axis Group Not Synchronized	19	Attempted execution on an axis whose associated axis group is not currently synchronized.
Axis in Faulted State	20	Attempted execution on an axis, which is in the Faulted state.
Group in Faulted State	21	Attempted execution on an axis, which is in a group, which is in the Faulted state.

Error Message	Code	Description
Illegal Dynamic Change	23	Attempted an illegal change of dynamics such as merge on an S-curve, change profile from trap to S-curve on the fly, change S-curve to non-zero speed or changing accel of an S-curve.
Illegal Controller Mode Operation	24	Attempted execution when the processor is in test mode
Illegal Execution Target	35	Attempted execution with a specified Output Cam not supported by the Logix controller.
Illegal Output Cam	36	Attempted execution with an Output Cam array containing at least one member out of range: 1. OutputBit less than 0 or greater than 31. 2. Invalid LatchType value. 3. Invalid UnlatchType value. 4. Left ≥ Right while LatchType is set to ‘position’ or ‘position and enable’. 5. Left < Cam Start Position while LatchType is set to ‘position’ or ‘position and enable’. 6. Right > Cam End Position while UnlatchType is set to ‘position’ or ‘position and enable’. 7. Duration ≤ 0 while UnlatchType is set to ‘duration’ or ‘duration and enable’. 8. Invalid EnableType value while LatchType or UnlatchType is set to ‘enable’ or ‘duration and enable’. 9. Invalid EnableBit value while LatchType or UnlatchType is set to ‘enable’ or ‘position and enable’ or ‘duration and enable’.
Illegal Output Compensation	37	Attempted execution with an Output Cam array containing at least one member out of range: 1. Invalid Mode value. 2. CycleTime ≤ 0 while Mode is set to ‘pulsed’ or ‘inverted and pulsed’. 3. DutyCycle less than 0 or greater than 100 while Mode is set to ‘pulsed’ or ‘inverted and pulsed’.

Extended Error Codes:

Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions. Extended Error codes for the Parameter Out of Range (13) error code lists a number that refers to the number of the operand as they are listed in the faceplate from top to bottom with the first operand being counted as zero. Therefore for the MAOC instruction, an extended error code of 4 would refer to the Output operand’s value. You would then have to check your value with the accepted range of values for the instruction.

Status Bits: *MAOC Effects on Status Bits*

Status bits may be used to determine if an MAOC instruction can be initiated. The MAOC instruction affects the following status words in the Motion Axis Structure:

- OutputCamStatus
- OutputCamPendingStatus
- OutputCamLockStatus
- OutputCamTransitionStatus

If the Execution Schedule is set to Forward Only, Reverse Only or Bi-Directional, an MAOC instruction can be initiated when either of the following two conditions exist:

- OutputCamStatus bit = FALSE

  or

- OutputCamStatus bit = TRUE
  OutputCamLockStatus bit = FALSE
  OutputCamTransitionStatus bit = FALSE

If the Execution Schedule is Pending, the MAOC instruction is initiated if either of the following two conditions exist:

- OutputCamStatus bit = FALSE

  or

- OutputCamStatus bit = TRUE
  OutputCamTransitionStatus bit = FALSE

Axis and Module Fault Conditions Disarm Output Cams

When the controller detects one of the following faults, it disarms output cams:

- For Axis_Servo and Axis_Servo_Drive, axis feedback loss fault
- For Axis_Servo and Axis_Servo_Drive, module fault
- For Axis_Consumed, physical axis fault

Those faults produce unreliable feedback data.

Also, if an axis fault exists when an MAOC instruction is initiated, the instruction errs.

Scheduled Output Module

The 1756-OB16IS Scheduled Output module is designed to work in conjunction with the Motion Axis Output Cam (MAOC) motion instruction to provide position based output control (also know as PLS). The MAOC instruction by itself allows position based output control using the position of any motion axis in ControlLogix as the position reference and any output or boolean as the output. The MAOC updates the outputs based on motion axis position at the motion group coarse update rate (typically 2ms-10ms). While this is adequate for some applications, it is too slow for many high speed applications typically found in converting and packaging segments. The 1756-OB16IS module improves performance by supporting the ability to schedule the output turn-on/turn-off time of 8 of its 16 outputs (outputs 0-7) in 100μs increments. Outputs are scheduled by entering data into one or more of the 16 schedules provided by the output connection data store.

Operation

Scheduled Outputs as defined here should not be confused with the earlier implementation of scheduled outputs. The previous implementation schedules outputs on a per module basis and all output points are controlled by a single timestamp. This implementation schedules outputs on a per point basis and each individual output point is controlled by its own timestamp.

Individual schedules are created in the controller, stored in the output image table for the module, and sent over the backplane to the Scheduled Output module. The schedule specifies a sequence count, the output point to be associated with the schedule, the time at which an output value should be applied to the physical output point, and the value to be applied at the scheduled time. The I/O module receives and stores the schedule. The CST timestamp of each schedule is monitored by the module. When a schedule has expired, that is the current time, matches the scheduled timestamp, the output value is then applied to the corresponding output bit. Timer hardware in the ASIC is used to optimize the scheduling algorithm. This hardware also reduces the latency and jitter performance. Status of each schedule is reported in the output echo connection and reflected in the input image for the module.

The scheduled output functionality relies on CST (Coordinated System Time) timestamp. At least one controller in the chassis must be a CST time master.

Unused outputs may be used as normal outputs and are applied immediately rather than waiting for the CST timestamp to expire. A mask is sent to the module to indicate which outputs are to function as normal outputs.

The scheduled output module supports up to 8 outputs that can be individually scheduled. The scheduled outputs must be between output points 0 and 7. The 1756-OB16IS supports up to 16 schedules with two schedules per output. Outputs that are not “scheduled” are used as normal output points. A mask is used to indicate which points are scheduled and which points are unscheduled. Jitter and latency performance is less than 100 microseconds. All of the scheduling configuration is done through the MAOC instruction.

If a new schedule as indicated by a change in the sequence count is received by the I/O module before the current schedule has expired, the current schedule is overwritten. This mechanism can be used to cancel currently active schedule. Status bits returned in the output echo connection may be used to determine the current state of each schedule and to trigger corresponding event tasks.

If a new schedule is sent by the controller and the CST timestamp has already past, the output is asserted until the CST time has completely wrapped around. The module does not check for an expired CST timestamp.

WARNING

If the time between two schedules is less than the minimum schedule interval (e.g. 100 μs), then jitter occurs. This means that even though two outputs are scheduled at different times (e.g. time 90 and time 110), they both activate at the same time (e.g. time 90). (The minimum schedule interval should not be set to faster than 100 μs.)

Remote Operation

Scheduled outputs using the 1756-OB16IS module do not work with a remote chassis.

Usage with MAOC Instruction

When used with motion and the MAOC instruction values in the output image are controlled by the Motion Planner firmware in the controller. The Motion Planner triggers the data to be sent to the module. Although, the normal program/task scan also triggers data to be sent to the module. Data integrity is maintained by the firmware always setting the sequence count for a given schedule last.

The Output Cam instruction processes cam events for scheduled outputs one coarse update period sooner than unscheduled outputs. When a programmed on or off event is detected, a schedule is sent to the output module to turn the output on/off at the appropriate time within the next coarse update period. The Output Cam instruction divides the coarse update period into sixteen time slots. Each cam on/off event is assigned to a time slot. The accuracy of the scheduled time for the output therefore is 1/16 the coarse update period. A coarse update period of 1 millisecond yields a schedule accuracy of 62.5 microseconds.

IMPORTANT The 1756-OB16IS Scheduled Output Module can be associated with one (1) MAOC axis/execution target only.

The MAOC detects latch and unlatch events one coarse update ahead and schedules the event to occur within the next coarse update. This is accomplished by applying a one coarse update internal delay to each scheduled output latch and unlatch position. When the latch or unlatch event is detected, the delta time from the start of the coarse update to the event is calculated, and the output is scheduled to occur at the Coordinated System Time (CST) corresponding to the next coarse update period. To facilitate this, Output Cam functionality has access to the CST captured when the current coarse update period occurred.

The MAOC is able to process both scheduled and unscheduled output bits for the 1756-OB16IS.

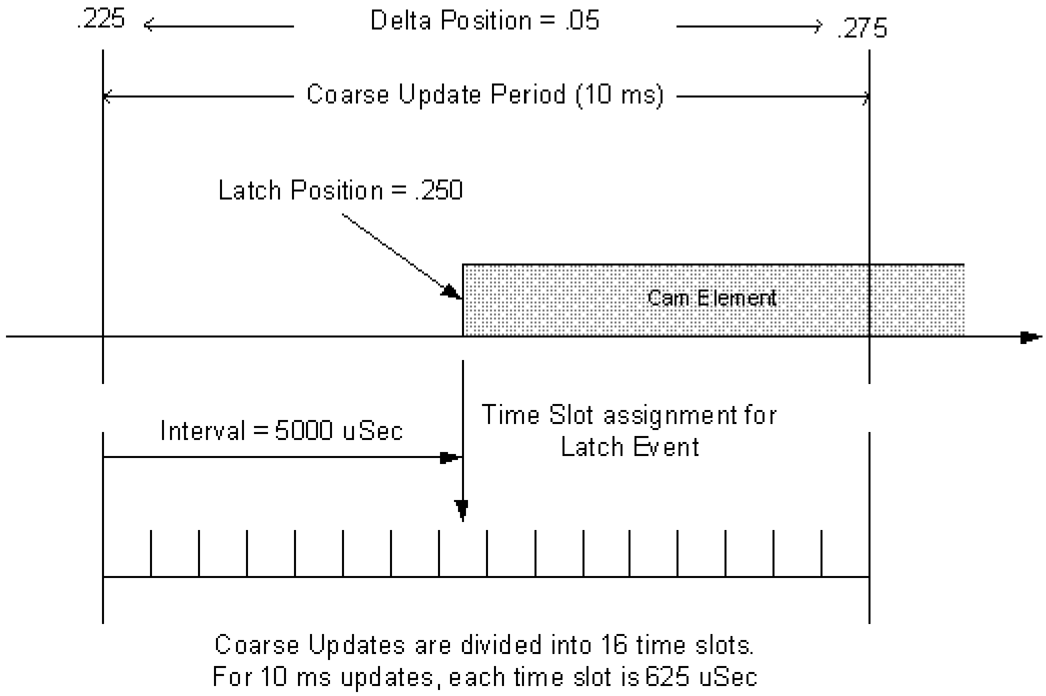
The MAOC allocates all eight scheduled outputs for exclusive use by the motion planner Output Cam. The MAOC sets the Mask field to 0xff every coarse period in case the user attempts to change it. What this implies is that the user cannot directly affect output bits 0-7, but does have the ability to modify output bits 8-15.

IMPORTANT The outputs 0 -7 can be forced by forcing the Data Bit to 0 or 1 and its corresponding bit in the ScheduleMask to 0. For outputs 8 - 15, only the Data Bit needs to be forced.

Due to the limit of sixteen schedules supported by the 1756-OB16IS, some constraints are applied to the number of events that can be processed every coarse update period.

Only eight schedules are available each coarse update. This allows for two consecutive coarse updates in which each update contains eight output events. As a group of eight schedules are currently being processed by the 1756-OB16IS, a second group of eight schedules can concurrently be set up for the next coarse update.

The following diagram illustrates the relationship between the coarse update period, a cam latch event and the time slots.



Inter-relationship of Coarse Update Period, Cam Latch, and Time Slots

Each Time Slot stores the following information:

Latch Event Mask – When a latch event is detected, the time slot in which it belongs is calculated and the bit in the Latch Event Mask corresponding to the output bit of the latch is set.

Unlatch Event Mask – When an unlatch event is detected, the time slot in which it belongs is calculated and the bit in the Unlatch Event Mask corresponding to the output bit of the unlatch is set.

Interval – The time in micro-seconds from the start of the coarse update in which the Latch or Unlatch event occurs.

Pulse On Mask – For pulsed outputs, the time slot in which a pulse on event is calculated and the bit in the Pulse On Mask corresponding to the output bit of the pulse event is set.

Pulse Off Mask – For pulsed outputs, the time slot in which a pulse off event is calculated and the bit in the Pulse Off Mask corresponding to the output bit of the pulse event is set.

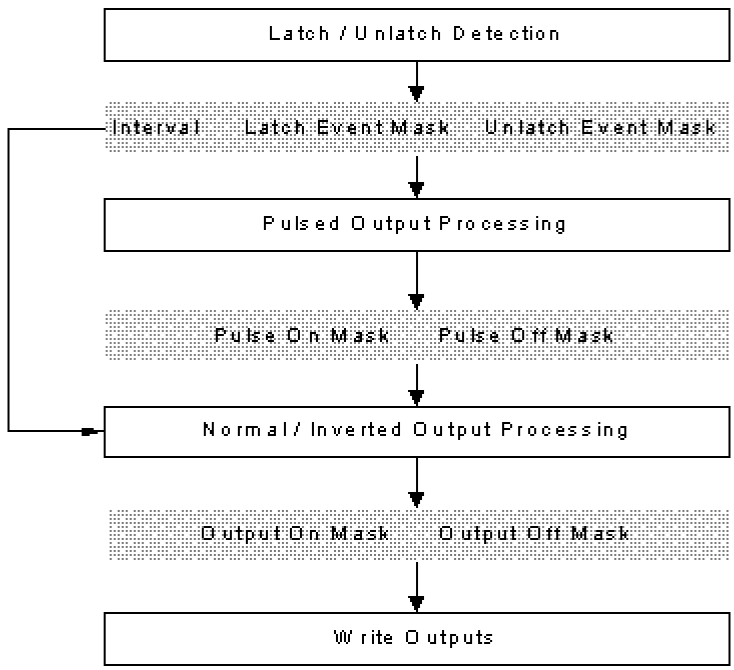
Output On Mask – For normal outputs, the bit corresponding to the output bit of the Latch or Pulse On event is set indicating that the output is to be turned on for these events.

For inverted outputs, the bit corresponding to the output bit of the Unlatch or Pulse Off event is set indicating that the output is to be turned on for these events.

Output Off Mask – For normal outputs, the bit corresponding to the output bit of the Unlatch or Pulse Off event is set indicating that the output is to be turned off for these events.

For inverted outputs, the bit corresponding to the output bit of the Latch or Pulse On event is set indicating that the output is to be turned off for these events.

The following is a simplified overview of how Time Slot data is utilized.



Overview of How Time Slot Data Utilization

Time slots are also used to process overlapping cam elements. A semaphore is maintained to indicate the currently active state of each output bit. In addition, if a programmed cam element Latch and Unlatch event occurs in the same time slot, they cancel each other out.

The minimum width of a cam element corresponds to the width of a time slot, or 1/16 the coarse update period.

I/O Subsystem

The user can specify the Output parameter of an MAOC instruction as either a memory tag or an Output Module's data tag. A pointer to the tag is passed into the MAOC instruction. Also passed into the MAOC is an internal parameter of type IO_MAP. If the Output parameter references controller memory, the IO_MAP parameter is NULL. If the Output parameter references an output module, the IO_MAP parameter points to the map structure for the module. The MAOC instruction can then determine if the Output parameter is associated with a 1756-OB16IS module by checking the module type stored in the driver table.

Output Data Structure

Field	Size	Description
Value	4 bytes	Data values for un-scheduled output bits. 0 = Off 1 = On
Mask	4 bytes	Selects which output bits are to be scheduled. Only the first eight bits (0-7) can be scheduled. 0 = Not scheduled 1 = Scheduled

Array of 16 Schedule Structures

Field	Size	Description
Schedule ID	1 byte	Valid ID's are 1-16. Any other value indicates that the schedule is not to be considered.
Sequence Number	1 byte	The OB16IS will maintain a copy of the schedule. A change in sequence number will tell the OB16IS to process the data in this schedule.
Point ID	1 byte	Indicates the output bit associated with this schedule. Entered as a value 00- 07.
Point Value	1 byte	Next state of output bit specified in Point ID. 0 = Off 1 = On
Time Stamp	4 bytes	The lower 32 bits of CST. Indicates when to change the state of the specified output bit.

Schedule Processing

The Value and Mask fields are processed and all unscheduled data bits are moved to the module output data store. This data is written to the output terminals after all schedules have been processed.

Each schedule is processed. The schedule is not considered if:

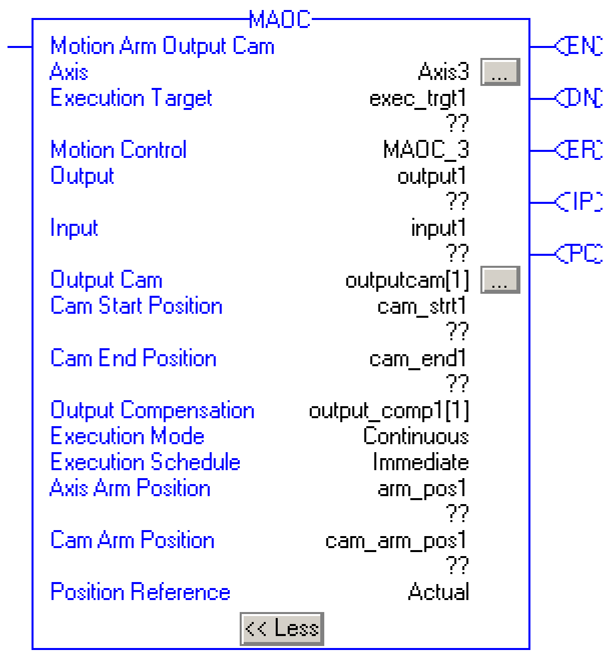
- The Schedule ID is not in the range of 1 – 16
- The Point ID is not in the range of 0 – 7

- The Sequence Number has not changed

If the schedule is to be considered, it is marked “active”.

All “active” schedules are examined every 200 micro-seconds. The schedule Time Stamp is compared to the current CST. If the current CST is greater than or equal to the scheduled Time Stamp, the Point Value in the schedule is moved to the module output data store for the specified output bit.

M Example: *Relay Ladder*



MAOC Ladder Example

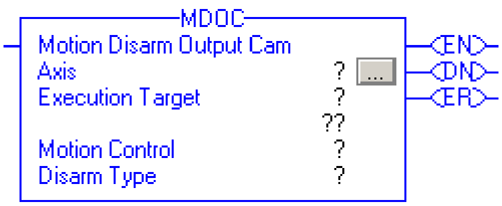
Structured Text

```
MAOC(Axis3,exec_trgt1,MAOC_3,output1,
input1,outputcam1[1],cam_strt1,cam_end1,
output_comp1[1],continuous,immediate,arm_pos1,
cam_arm_pos1,actual);
```

Motion Disarm Output Cam (MDOC)

The Motion Disarm Output Cam (MDOC) instruction initiates the disarming of one or more Output Cams connected to the specified axis. Based on the disarm type, the MDOC disarms either all Output Cams or only a specific Output Cam. The corresponding outputs maintain the last state after the disarming.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Axis	AXIS_FEEDBACK AXIS_CONSUMED AXIS_VIRTUAL AXIS_GENERIC AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis, which provides the position input to the Output Cam. Ellipsis launches Axis Properties dialog.
Execution Target	SINT, INT, or DINT	immediate or tag	The execution target defines the specific Output Cam from the set connected to the named axis. Behavior is determined by the following: • 0...8 – Output Cams executed in the Logix controller. • 9...31 – Reserved for future use.
Motion Control	MOTION_INSTRUCTION	tag	Structure used to access instruction status parameters.
Disarm Type	UINT32	immediate	Selects one or all Output Cams to be disarmed for a specified axis. Select either: 0 = All – Disarms all Output Cams connected to the specified axis. 1 = Specific – Disarms the Output Cam that is connected to the specified axis and defined by the Execution Target.

Structured Text

```
MDOC(Axis,ExecutionTarget,
MotionControl,DisarmType);
```

The operands are the same as those for the relay ladder MDOC instruction.

For the operands that require you to select from available options, enter your selection as:

This operand	Has these options which you...	
	enter as text	or enter as a number
Disarm Type	all	0
	specific	1

MOTION_INSTRUCTION Structure

Mnemonic:	Description:
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the rung goes false.
.DN (Done) Bit 29	It is set when the Output Cam(s) have been successfully disarmed.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error.

Description: The Motion Disarm Output Cam (MDOC) instruction disarms a specific or all output cams for a specified axis depending on the selected disarm type. The axis provides the position input to the Output Cam. The execution target defines a specific Output Cam from the set that is connected to the specified axis.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

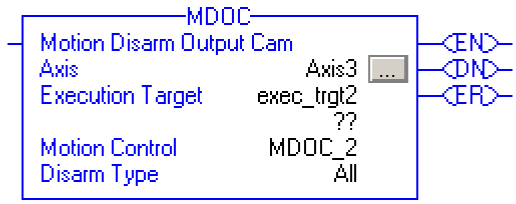
Error Codes See Error Codes (ERR) for Motion Instructions.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions. Extended Error codes for the Parameter Out of Range (13) error code lists a number that refers to the number of the operand as they are listed in the faceplate from top to bottom with the first operand being counted as zero. Therefore for the MDOC instruction, an extended error code of 4 would refer to the Disarm Type operand’s value. You would then have to check your value with the accepted range of values for the instruction.

Status Bits: *MDOC Changes to Status Bits*

None

Example: *Relay Ladder*



MDOC Ladder Example

Structured Text

```
MDOC(Axis3,exec_trgt2,MDOC_2,all);
```

Notes:

Motion Configuration Instructions
(MAAT, MRAT, MAHD, MRHD)

ATTENTION

Tags used for the motion control attribute of instructions should only be used once. Re-use of the motion control tag in other instructions can cause unintended operation. This may result in damage to equipment or personal injury.

Introduction

Configuration instructions include all motion instructions that are used establish and apply servo configuration parameters to an axis. This group of instructions includes hookup test diagnostic instructions and tuning instructions.

Use the motion configuration instructions to tune an axis and to run diagnostic tests for the servo system. These tests include:

- A motor encoder hookup test.
- An encoder hookup test.
- A marker test

The motion configuration instructions are:

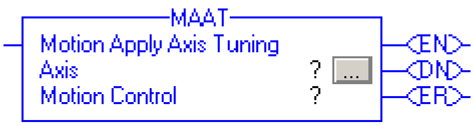
If you want to	Use this instruction	Available in these languages
Compute a complete set of servo gains and dynamic limits based on a previously executed MRAT instruction. The MAAT instruction also updates the servo module with the new gain parameters.	MAAT	Relay Ladder Structured Text
Command the servo module to run a tuning motion profile for an axis.	MRAT	Relay Ladder Structured Text
Apply the results of a previously executed MRHD instruction. The MAHD instruction generates a new set of encoder and servo polarities based on the observed direction of motion during the MRHD instruction.	MAHD	Relay Ladder Structured Text
Command the servo module to run one of three diagnostic tests on an axis.	MRHD	Relay Ladder Structured Text

Motion Apply Axis Tuning (MAAT)

The Motion Apply Axis Tuning (MAAT) instruction is used compute a complete set of servo gains and dynamic limits based on the results of a previously run Motion Run Axis Tuning (MRAT) instruction and update the motion module with these new gain parameters. While this instruction takes no explicit parameters, input is derived from the Axis Tuning Configuration parameters as described in the Motion Axis Object specification. After execution of the MAAT instruction, the corresponding axis should be ready for servo activation.

Operands:

Relay Ladder



Operand	Type	Format	Description
Axis	AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.

Structured Text

```
MAAT(Axis,MotionControl);
```

The operands are the same as those for the relay ladder MAAT instruction.

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	The enable bit indicates when the instruction is enabled. It remains set until servo messaging completes and the rung-condition-in goes false.
.DN (Done) Bit 29	The done bit indicates when the instruction completes an apply axis-tuning process.
.ER (Error) Bit 28	The error bit indicates when the instruction detects an error, such as if the axis is not configured.

Description:

The Motion Apply Axis Tuning (MAAT) instruction is used to execute a series of computations resulting in values for gain and dynamic configuration parameters on the specified axis. As part of the work performed by MAAT, these resultant configuration parameters are applied so that the axis is ready for full servo operation. This instruction is designed to follow execution of the MRAT instruction which generates axis input configuration values for the MAAT instruction. See the MRAT instruction description for more information. MAAT requires no explicit input parameters; simply enter or select the desired physical axis.

If the targeted axis does not appear in the list of available axes, the axis has not been configured for operation. Use the Tag Editor to create and configure a new axis.

The MAAT instruction uses axis configuration parameters as input and output. The input configuration parameters that MRAT uses are shown in the table below. Refer to the Motion Axis Object specification for a detailed description of these parameters.

The axis configuration parameters that MAAT uses as input depends on the External Drive configuration. If the External Vel Servo Drive configuration bit parameter is TRUE, indicating interface to an external velocity servo drive, the following input parameters are required.

Axis Parameter	Data Type	Units	Meaning
Tuning Velocity	Real	pos units/sec	Top Speed of Tuning Profile.
Tune Accel	Real	pos units/sec^2	Calculated Acceleration Time of Tuning Profile.
Tune Decel	Real	pos units/sec^2	Calculated Deceleration Time of Tuning Profile.
Tune Velocity Scaling	Real	mV/KCPS	Measured Velocity Scaling factor of axis Drive/Motor/Encoder system.
Tune Velocity Bandwidth	Real	Hertz	Bandwidth of External Velocity Servo Drive

If the External Vel Servo Drive configuration bit parameter is FALSE, indicating interface to an external torque servo drive, the following input parameters are required.

Axis Parameter	Data Type	Units	Meaning
Damping Factor	Real	-	Damping Factor used to calculate the the gains.
Tuning Velocity	Real	pos units/sec	Top Speed of Tuning Profile.
Tune Accel	Real	pos units/sec^2	Calculated Acceleration Time of Tuning Profile.
Tune Decel	Real	pos units/sec^2	Calculated Deceleration Time of Tuning Profile.
Effective Inertia	Real	mV/ $KCPS^2$	Computed Effective Inertia of Drive/Motor system.
Position Servo Bandwidth	Real	Hertz	Maximum Position Servo Loop Bandwidth.

The axis configuration parameters that MAAT generates as output depend on the External Drive configuration. If the External Vel Servo Drive configuration bit parameter is TRUE, indicating interface to an external velocity servo drive, the following output parameters are generated.

Axis Parameter	Data Type	Units	Meaning
Pos Proportional Gain	Real	1/msec	Position Servo Loop Proportional Gain
Pos Integral Gain	Real	$1/msec^2$	Position Servo Loop Integral Gain -- Set to Zero
Velocity Feedforward	Real	-	Position Servo Loop Proportional Gain
Acceleration Feedforward	Real	-	Velocity Command Feedforward -- Set to Zero
Max Speed	Real	pos units/sec	Maximum Speed for Motion Profiles – Set to Tuning Velocity
Max Acceleration	Real	pos units/sec^2	Maximum Acceleration for Motion Profiles
Max Deceleration	Real	pos units/sec^2	Maximum Acceleration for Motion Profiles
Output Filter Bandwidth	Real	Hertz	Bandwidth of Low Pass Servo Output Filter
Output Scaling	Real	mV/ KCPS	Scale Factor applied to output of the Position Servo Loop to the DAC.
Position Error Tolerance	Real	pos units	Maximum Servo Loop Position Error allowed without Fault.

If the External Vel Servo Drive configuration bit parameter is FALSE, indicating interface to an external torque servo drive, the following output parameters are generated.

Axis Parameter	Data Type	Units	Meaning
Pos Proportional Gain	Real	1/msec	Position Servo Loop Proportional Gain
Pos Integral Gain	Real	$1/msec^2$	Position Servo Loop Integral Gain
Vel Proportional Gain	Real	1/msec	Velocity Servo Loop Proportional Gain
Vel Integral Gain	Real	$1/msec^2$	Velocity Servo Loop Integral Gain
Velocity Feedforward	Real	-	Position Servo Loop Proportional Gain
Acceleration Feedforward	Real	-	Velocity Command Feedforward
Max Speed	Real	pos units/sec	Maximum Speed for Motion Profiles – Set to Tuning Velocity
Max Acceleration	Real	pos units/sec^2	Maximum Acceleration for Motion Profiles

Axis Parameter	Data Type	Units	Meaning
Max Deceleration	Real	pos units/sec^2	Maximum Acceleration for Motion Profiles
Output Filter Bandwidth	Real	Hertz	Bandwidth of Low Pass Servo Output Filter
Output Scaling	Real	mV/ KCPS^2	Scale Factor applied to output of the Velocity Servo Loop to the DAC.
Position Error Tolerance	Real	pos units	Maximum Servo Loop Position Error allowed without Fault.

The above output parameters generated by the MAAT instruction are immediately applied to the specified axis so that subsequent motion can be performed.

For more information about tuning configuration parameters refer to the Motion Axis Object Specification.

To successfully execute a MAAT instruction, the targeted axis must be configured as a Servo axis and be in the Axis Ready state, with servo action off. If these conditions are not met, the instruction errs.

IMPORTANT The MAAT instruction execution may take multiple scans to execute because it requires transmission of a message to the motion module. The Done (.DN) bit not set immediately, but only after this message has been successfully transmitted.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes See Error Codes (ERR) for Motion Instructions.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions. The following Extended Error codes help to pinpoint the problem

when the MAAT instruction receives a Servo Message Failure (12) error message.

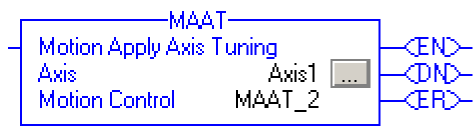
Associated Error Code (decimal)	Extended Error Code (decimal)	Meaning
SERVO_MESSAGE_FAILURE (12)	No Resource (2)	Not enough memory resources to complete request. (SERCOS)
SERVO_MESSAGE_FAILURE (12)	Object Mode conflict (12)	Axis is in shutdown.
SERVO_MESSAGE_FAILURE (12)	Permission denied (15)	Enable input switch error. (SERCOS)
SERVO_MESSAGE_FAILURE (12)	Device in wrong state (16)	Redefine Position, Home, and Registration 2 are mutually exclusive (SERCOS), device state not correct for action. (SERCOS)

Status Bits: *MAAT Changes to Status Bits*

None

Example: When the input conditions are true, the controller computes a complete set of servo gains and dynamic limits for *axis1 based on the results of the previously executed Motion Run Axis Tuning (MRAT) instruction*.

Relay Ladder



MAAT Ladder Example

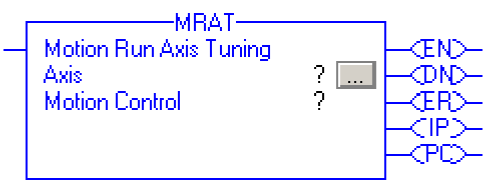
Structured Text

```
MAAT(Axis1,MAAT_2);
```

Motion Run Axis Tuning (MRAT)

Use the MRAT to command the motion module to run a tuning motion profile for the specified axis. The tuning motion profile consists of one or more acceleration and deceleration ramps induced by applying fixed voltages to the servo's drive output. Note that this instruction does not at any time close the servo loop. While this instruction takes no explicit input parameters, it does derive input from the Axis Tuning Configuration parameters. The result of executing the MRAT instruction is a set of measurement data that is stored in the Axis Object for subsequent use with the Motion Apply Axis Tuning (MAAT) instruction.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Axis	AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.

Structured Text

```
MRAT(Axis,MotionControl);
```

The operands are the same as those for the relay ladder MRAT instruction.

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set after the tuning process has been successfully completed.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.
.IP (In Process) Bit 26	It is set on positive rung transition and cleared after the tuning process is complete, or terminated by a stop command, shutdown, or a servo fault.
.PC (Process Complete) Bit 27	It is set after the tuning process has been successfully completed.

Description: The Motion Run Axis Tuning (MRAT) instruction is used to execute a tuning motion profile on the specified axis. During this brief tuning motion profile, the motion module makes timing and velocity measurements that serve as input data for a subsequent MAAT

(Motion Apply Axis Tuning) instruction. MRAT requires no explicit input parameters; simply enter or select the desired physical axis.

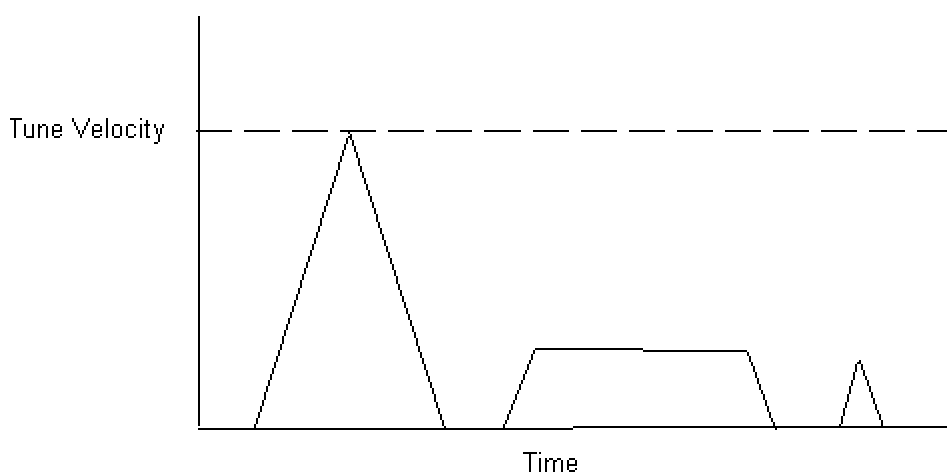
If the targeted axis does not appear in the list of available axes, the axis has not been configured for operation. Use the Tag Editor to create and configure a new axis.

The MRAT instruction uses axis configuration parameters as input and output. The input configuration parameters that MRAT uses are shown in the table below.

Axis Parameter	Data Type	Units	Meaning
Tuning Direction	Boolean	-	Direction of Tuning Motion (0-Fwd, 1-Rev)
Tuning Travel Limit	Real	pos units	Maximum allowed excursion of Axis
Tuning Velocity	Real	pos units/sec	Top Speed of Tuning Profile.
Damping Factor	Real	-	Damping Factor used to calculate the maximum Position Servo Bandwidth.

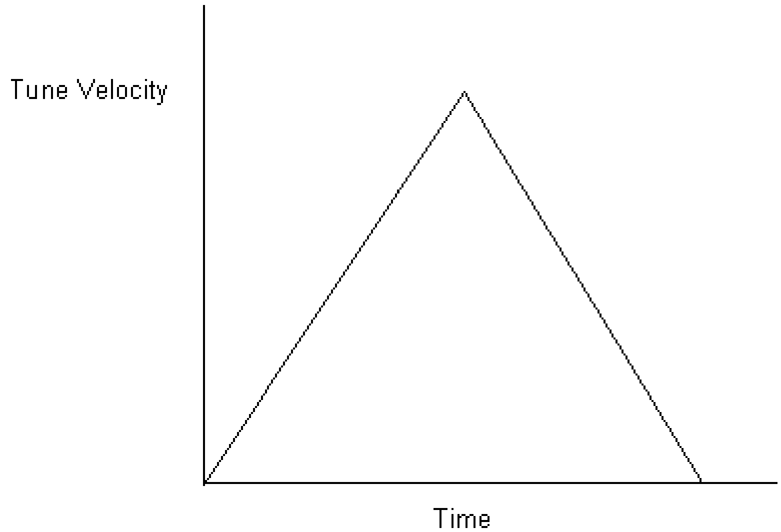
Based on the above configuration parameters, MRAT execution generates a motion event on the specified axis that consists of a single triangular velocity profile or a series of three such profiles. Tune Velocity must be within the maximum speed capability of the drive and motor. The configured value for Tune Velocity should be set to the desired maximum operating speed of the axis so that the resulting tuning parameters are based on the dynamics of the system at that speed.

If the External Vel Servo Drive configuration bit parameter is TRUE, indicating interface to an external velocity servo drive, three pulses are applied to the axis. The tuning velocity profile for this case is shown in the diagram below.



Tuning Velocity Profile when True

If the External Vel Servo Drive configuration bit parameter is FALSE, indicating interface to an external torque servo drive, only one pulse is applied to the axis. The tuning velocity profile is shown below.



Tuning Velocity Profile when False

The axis configuration parameters that MRAT generates as output depend on the External Drive configuration. If the External Vel Servo Drive configuration bit parameter is TRUE, indicating interface to an external velocity servo drive, the following output parameters are generated.

Axis Parameter	Data Type	Units	Meaning
Tune Status	Real	-	Status Report of the Tuning Process
Tune Accel Time	Real	seconds	Measured Acceleration Time of Tuning Profile.
Tune Decel Time	Real	seconds	Measured Deceleration Time of Tuning Profile.
Tune Accel	Real	pos units/sec^2	Calculated Acceleration Time of Tuning Profile.
Tune Decel	Real	pos units/sec^2	Calculated Deceleration Time of Tuning Profile.
Tune Velocity Scaling	Real	mV/KCPS	Measured Velocity Scaling factor of axis Drive/Motor/Encoder system.
Tune Rise Time	Real	mV/KCPS	Measured Rise Time of Tuning Step Response Profile.
Tune Velocity Bandwidth	Real	Hertz	Computed Bandwidth of External Velocity Servo Drive

If the External Vel Servo Drive configuration bit parameter is FALSE, indicating interface to an external torque servo drive, the following output parameters are generated.

Axis Parameter	Data Type	Units	Meaning
Tune Status	Real	-	Status Report of the Tuning Process
Tune Accel Time	Real	seconds	Measured Acceleration Time of Tuning Profile.
Tune Decel Time	Real	seconds	Measured Deceleration Time of Tuning Profile.
Tune Accel	Real	pos units/sec^2	Calculated Acceleration Time of Tuning Profile.
Tune Decel	Real	pos units/sec^2	Calculated Deceleration Time of Tuning Profile.
Effective Inertia	Real	mV/ KCPS^2	Computed Effective Inertia of Drive/Motor system.
Position Servo Bandwidth	Real	Hertz	Calculated Maximum Position Servo Loop Bandwidth.

The above output parameters generated by the MRAT instruction serve as inputs to a subsequent MAAT instruction which performs further tuning calculations and applies the results to various axis' servo and dynamic configuration parameters.

Tune Status Parameter

Conditions may occur that make it impossible for the controller to properly perform the tuning operation. When this is the case, the tuning process is automatically aborted and a tuning fault reported that is stored in the Tune Status output parameter (GSVable). It is also possible to manually abort a tuning process using a MAS instruction which results in a tuning fault reported by the Tune Status parameter. Possible values for Tuning Status are shown in the table below.

Status Code	Code	Meaning
Tune Success	0	Tune process has been successful.
Tune In Process	1	Tuning is in progress.
Tune Aborted	2	Tuning Process was aborted by user.
Tune Time-out	3	Tuning Process has timed out
Tune Servo Fault	4	Tuning Process Failed due to Servo Fault
Tune Travel Fault	5	Axis reached Tuning Travel Limit
Tune Polarity Fault	6	Axis motion heading in wrong direction due to incorrect motor/encoder polarity configuration.
Tune Speed Fault	7	Axis tuning speed too low to achieve minimum measurement accuracy.

IMPORTANT The Tune Status Parameter is not to be mistaken for the .STATUS sub-tag of the MRAT instruction.

To successfully execute a MRAT instruction on an axis, the targeted axis must be configured as a Servo Axis Type and the axis must be in the Axis Ready state. If any of these conditions are not met than the instruction errs.

IMPORTANT When the MRAT instruction is initially executed the In Process (.IP) bit is set and the Process Complete (.PC) bit is cleared. The MRAT instruction execution can take multiple scans to execute because it requires transmission of multiple messages to the motion module. The Done (.DN) bit, is not set immediately, but only after these messages are successfully transmitted. The In Process (.IP) bit is cleared and the Process Complete (.PC) bit is set at the same time that the Done (.DN) bit is set.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes See Error Codes (ERR) for Motion Instructions.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions. The following Extended Error codes help to pinpoint the problem

when the MRAT instruction receives a Servo Message Failure (12) error message.

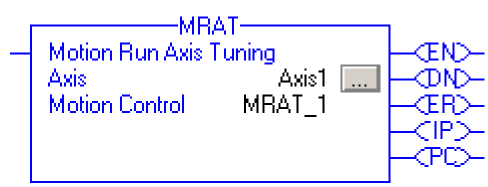
Associated Error Code (decimal)	Extended Error Code (decimal)	Meaning
SERVO_MESSAGE_FAILURE (12)	Process terminated on request (1)	Tune execution followed by an instruction to shutdown/disable drive, or a motion stop instruction or a Processor change requests a cancel of Tune.
SERVO_MESSAGE_FAILURE (12)	Object Mode conflict (12)	Axis is in shutdown.
SERVO_MESSAGE_FAILURE (12)	Device in wrong state (16)	Incorrect Tune Process order. (SERCOS)

Status Bits: *MRAT Changes to Status Bits*

Bit Name:	State:	Meaning:
DriveEnableStatus	TRUE	Axis is in Drive Control state with the Drive Enable output active while the Tuning Profile is running.
TuneStatus	TRUE	The axis is running a tuning process.

Example: When the input conditions are true, the controller commands the servo module to run a tuning motion profile for *axis1*.

Relay Ladder



MRAT Ladder Example

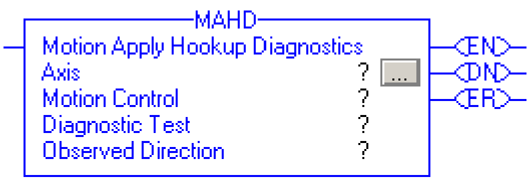
Structured Text

```
MAR(Axis1,MRAT_1);
```

Motion Apply Hookup Diagnostics (MAHD)

The Motion Apply Hookup Diagnostics (MAHD) instruction is used to apply the results of a previously run Motion Run Hookup Diagnostic (MRHD) instruction to generate a new set of encoder and servo polarities based on the Observed Direction of motion during the test. As part of the application process the instruction updates the motion module with these new polarity settings. After execution of the MAHD instruction, and assuming that a stable set of gains has been established, the corresponding axis should be ready for servo activation.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Axis	AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.
Diagnostic test	UDINT	immediate	Selects the specific test for the motion module to run: 0 = motor/encoder hookup test 1 = encoder hookup test 2 = encoder marker test
Observed direction	BOOLEAN	immediate	Sets the direction of the test motion. Select either: 0 = forward 1 = reverse

Structured Text

```
MAHD(Axis,MotionControl,
DiagnosticTest,
ObservedDirection);
```

The operands are the same as those for the relay ladder MAHD instruction.

For the operands that require you to select from available options, enter your selection as:

This operand	Has these options which you...	
	enter as text	or enter as a number
DiagnosticTest	motor_encoder	0
	encoder	1
	marker	2
ObservedDirection	forward	0
	reverse	1

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	is set after the hookup test apply process has been successfully executed.
.ER (Error) Bit 28	is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.

Description: The Motion Apply Hookup Diagnostics (MAHD) instruction is used to execute a series of computations resulting in values for the Encoder Polarity and Servo Polarity configuration bit parameters of the specified axis. As part of work performed by MAHD, these resultant configuration bit parameters are applied to the motion module so that the axis is ready for full servo operation. This instruction is designed to follow execution of the MRHD instruction which generates axis input configuration values for the MAHD instruction. See the MRHD instruction description for more information. MAHD requires specification of the Diagnostic Test to apply and the Observed Direction of motion during the previous MRHD test process. Enter or select the Diagnostic Test and the Observed Direction and the desired physical axis.

If the targeted axis does not appear in the list of available axes, the axis has not been configured for operation. Use the Tag Editor to create and configure a new axis.

The MAHD instruction uses axis configuration parameters as input and output. The input configuration parameters that MAHD uses are shown in the table below. The Test Direction Forward bit is automatically established as output from the MRHD instruction. Refer to the Motion Axis Object specification for a detailed description of these and other parameters.

Axis Parameter	Data Type	Units	Definition
Test Direction Forward	Boolean	-	Direction of axis travel during hookup test as seen by the motion module.

Motor Encoder Hookup Test

If the Motor Encoder Test is selected, the controller computes the proper setting for both the Encoder Polarity and the Drive Polarity based on the Observed Direction instruction parameter and the state of Test Direction Forward bit which was established by the output of the MRHD instruction. Once the Encoder Polarity and Drive Polarity settings are computed the MAHD applies these values to the corresponding axis configuration parameter bits as shown in the following table:

Axis Parameter	Data Type	Units	Definition
Encoder Polarity Negative	Boolean	-	Inverts the sense of the encoder feedback input to the motion module.
Drive Polarity Negative	Boolean	-	Inverts the sense of the DAC analog output from the motion module.

Encoder Hookup Test

If the Encoder Test is selected, the controller computes the proper setting for just the Encoder Polarity based on the Observed Direction instruction parameter and the state of Test Direction Forward bit which was established by the output of the MRHD instruction. Once the Encoder Polarity and Drive Polarity settings are computed, the MAHD applies these values to the corresponding axis configuration parameter bits as shown in the following table:

Axis Parameter	Data Type	Units	Definition
Encoder Polarity Negative	Boolean	-	Inverts the sense of the encoder feedback input to the motion module.

To successfully execute a MAHD instruction running the Motor Encoder Test, the targeted axis must be configured as either a Servo or

Feedback Only axis type. If any of these conditions are not met than the instruction errs.

IMPORTANT The MAHD instruction execution can take multiple scans to execute because it requires transmission of a message to the motion module. The Done (.DN) bit is not set immediately, but only after this message is successfully transmitted.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes See Error Codes (ERR) for Motion Instructions.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions. The following Extended Error codes help to pinpoint the problem when the MAHD instruction receives a Servo Message Failure (12) error message.

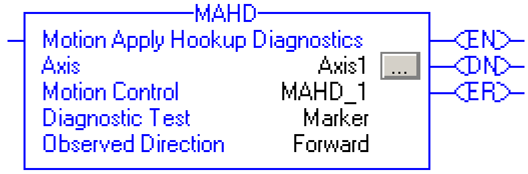
Associated Error Code (decimal)	Extended Error Code (decimal)	Meaning
SERVO_MESSAGE_FAILURE (12)	No Resource (2)	Not enough memory resources to complete request. (SERCOS)
SERVO_MESSAGE_FAILURE (12)	Object Mode conflict (12)	Axis is in shutdown.
SERVO_MESSAGE_FAILURE (12)	Permission denied (15)	Enable input switch error. (SERCOS)
SERVO_MESSAGE_FAILURE (12)	Device in wrong state (16)	Redefine Position, Home, and Registration 2 are mutually exclusive (SERCOS), device state not correct for action. (SERCOS)

Status Bits: *MAHD Changes to Status Bits*

None

Example: When the input conditions are true, the controller applies the results of a previously executed Motion Run Hookup Diagnostics (MRHD) instruction to *axis1*.

Relay Ladder



MAHD Ladder Example

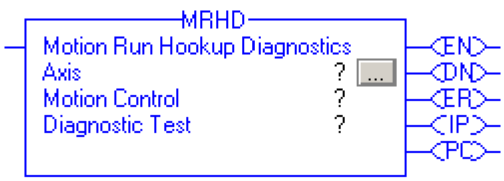
Structured Text

```
MAHD(axis1,axis1_MAHD,marker,forward);
```

Motion Run Hookup Diagnostics (MRHD)

Use the MRHD instruction to command the motion module to run any one of three different diagnostics on the specified axis as selected by the Test ID. Currently diagnostics are available to test the motor/encoder hookup for a servo axis, the encoder hookup only, and the encoder marker hookup. Only the motor/encoder diagnostic initiates motion on the axis. This action consists of a short move of a user Motor Encoder Test Increment. The move is initiated by roughly 1 Volt per second ramping level of the servo's drive output. The result of executing the MRHD instruction is that the parameters, Test Status and Test Direction Forward are updated.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Axis	AXIS_SERVO AXIS_SERVO_DRIVE	tag	Name of the axis to perform operation on.
Motion control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.
Diagnostic test	DINT	immediate	Selects the specific test for the motion module to run: 0 = motor/encoder hookup test 1 = encoder hookup test 2 = encoder marker hookup test 3 = Watchdog OK test

Structured Text

```
MRHD(Axis,MotionControl,
DiagnosticTest);
```

The operands are the same as those for the relay ladder MRHD instruction.

For the operands that require you to select from available options, enter your selection as:

This operand:	Has these options which you...	
	enter as text:	or enter as a number:
DiagnosticTest	motor_encoder	0
	encoder	1
	marker	2

MOTION_INSTRUCTION Structure

Mnemonic	Description
.EN (Enable) Bit 31	It is set when the rung makes a false-to-true transition and remains set until the servo message transaction is completed and the rung goes false.
.DN (Done) Bit 29	It is set after the hookup test process has been successfully executed.
.ER (Error) Bit 28	It is set to indicate that the instruction detected an error, such as if you specified an unconfigured axis.
.IP (In Process) Bit 26	It is set on positive rung transition and cleared after the diagnostic test process is complete, or terminated by a stop command, shutdown, or a servo fault.
.PC (Process Complete) Bit 27	It is set after the diagnostic test process has been successfully completed.

Description: The Motion Run Hookup Diagnostics (MRHD) instruction is used to execute various test diagnostics on the specified axis to test the integrity and, in some cases, the polarity of servo field connections. There are currently test diagnostics supporting drive hookup, encoder hookup, marker hookup and motion module OK contact hookup. During some of these test processes the motion module generates output to the external drive to produce a small amount of motion. Measurements made during some of these hookup diagnostic tests are saved as output configuration parameters that also serve as input data for a subsequent MAHD (Motion Apply Hookup Diagnostic) instruction. MRHD requires only one explicit input parameter, Diagnostic Test. Enter or select the Diagnostic Test to run and the axis to test.

If the targeted axis does not appear in the list of available axes, the axis has not been configured for operation. Use the Tag Editor to create and configure a new axis.

The MRHD instruction uses axis configuration parameters as input and output. The input configuration parameters that MRHD uses are shown in the table below.

Axis Parameter	Data Type	Units	Definition
Motor Encoder Test Increment	Real	-	Distance that the Axis must travel to satisfy the Hookup Diagnostic Test

The axis configuration parameters that MRHD generates as output depend on the specified Hookup Diagnostic.

Motor Encoder Hookup Test

If the Motor Encoder Test is selected, the motion module enables the external drive and generates a 1 Volt per second output ramp to the drive while monitoring the encoder feedback. When the axis has moved a distance greater than or equal to the configured Motor Encoder Test Increment, the test voltage is set back to zero and the drive disabled. The motion module than reports the direction of travel which is stored as one of the following output parameters:

Axis Parameter	Data Type	Units	Definition
Test Status	Integer	-	Status Report of the Hookup Diagnostic Test Process
Test Direction Forward	Boolean	-	Direction of axis travel during hookup test as seen by the motion module.

If due to improper hookup, or some other problem with the system, the axis feedback fails to detect that axis reaching the configured Motor Encoder Test Increment within 2 seconds, the servo sets the test voltage back to zero and disables the drive. The control reflects this condition through the Test Status axis output parameter. This usually indicates that either the cabling to the drive or the cabling to the encoder is incorrect. Running MRHD with the Encoder Hookup Test selected is an effective method of isolating the problem to the encoder or drive.

Encoder Hookup Test

If the Encoder Test is selected, the motion module does not generate any axis motion, but simply monitors axis encoder feedback. The axis can then be moved by hand or by some other independent drive actuator to generate motion. When the motion module detects that the axis has moved a distance greater than or equal to the configured Motor Encoder Test Increment, the test is complete. The motion module then reports the direction of travel as one of the following MRHD output parameters.

Axis Parameter	Data Type	Units	Definition
Test Status	Integer	-	Status Report of the Hookup Diagnostic Test Process
Test Direction Forward	Boolean	-	Direction of axis travel during hookup test as seen by the motion module.

If due to improper hookup, or some other problem with the system, the axis feedback fails to detect the axis reaching the configured Motor Encoder Test Increment after moving the axis at least that distance, then abort the test using the MAS instruction and check the encoder wiring.

Marker Hookup Test

If the Marker Test is selected, the motion module does not generate any axis motion, but simply monitors axis encoder feedback. The axis can then be moved by hand or by some other independent drive actuator to generate motion. When the motion module detects a marker (Channel Z) pulse, the test is then complete. The motion module then reports success via the Test Status

Axis Parameter	Data Type	Units	Definition
Test Status	Integer	-	Status Report of the Hookup Diagnostic Test Process
Test Direction Forward	Boolean	-	Direction of axis travel during hookup test as seen by the motion module.

If due to improper hookup, or some other problem with the system, the axis feedback fails to detect that axis reaching the configured Motor Encoder Test Increment after moving the axis at least that distance, then abort the test using the MAS instruction and check the encoder wiring.

Watchdog OK Test

If the Watchdog OK Test is selected, the motion module does not generate any axis motion, but simply simulates a CPU Watchdog failure which opens the OK contacts. The OK contacts should remain closed for 2 seconds. This test is used to check the OK contact wiring into the E-Stop string of the Drive system. In the event of a motion module DSP failure this mechanism is used to shut off the power supply to the drive(s). When the two second Watchdog OK Test is complete, the motion module then reports success via the Test Status as shown below:

Axis Parameter	Data Type	Units	Definition
Test Status	Integer	-	Status Report of the Hookup Diagnostic Test Process

Test Status

Conditions may occur that make it impossible for the control to properly perform the test operation. When this is the case, the test process is automatically aborted and a test fault is reported and stored in the Test Status output parameter. It is also possible to manually abort a test process using a MAS instruction which results in a test fault reported by the Test Status parameter. Possible values for Test Status are shown in the table below:

Error Message	Code	Definition
Test Success	0	Test process has been successful.
Test In Process	1	Test is in progress.
Test Aborted	2	Test Process was aborted by user.
Test Time-out	3	Test Process has exceeded timed out (2 Seconds)
Test Servo Fault	4	Test Process Failed due to Servo Fault
Test Increment Fault	5	Test Process Failed due to insufficient test increment distance to make a reliable measurement.

To successfully execute a MRHD instruction running the Motor Encoder Test, the targeted axis must be configured as a Servo Axis Type and the axis must be in the Axis Ready state. For other tests this instruction executes properly on either a Servo or Feedback Only axis type. If any of these conditions are not met than the instruction errs.

IMPORTANT When the MRHD instruction is initially executed the In process (.IP) bit is set and the Process Complete (.PC) bit is cleared. The MRHD instruction execution can take multiple scans to execute because it requires transmission of multiple messages to the motion module. The Done (.DN) bit, is not set immediately, but after these messages are successfully transmitted. The In process (.IP) bit is cleared and the Process Complete (.PC) bit is set at the same time that the Done (.DN) bit is set.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes See Error Codes (ERR) for Motion Instructions.

Extended Error Codes: Extended Error Codes provide additional instruction specific information for the Error Codes that are generic to many instructions. The following Extended Error codes help to pinpoint the problem

when the MRHD instruction receives a Servo Message Failure (12) error message.

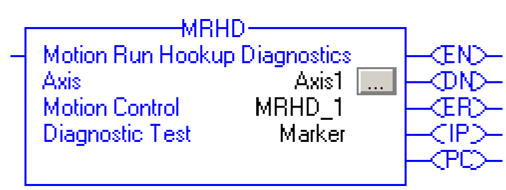
Associated Error Code (decimal)	Extended Error Code (decimal)	Meaning
SERVO_MESSAGE_FAILURE (12)	Process terminated on request (1)	Test execution followed by an instruction to shutdown/disable drive, or a motion stop instruction or a Processor change requests a cancel of the Test.
SERVO_MESSAGE_FAILURE (12)	Object Mode conflict (12)	Axis is in shutdown.
SERVO_MESSAGE_FAILURE (12)	Device in wrong state (16)	Incorrect Tune Process order. (SERCOS)

Status Bits: *MRHD Changes to Status Bits*

Bit Name	State	Meaning
DriveEnableStatus	TRUE	• The axis is in the drive control state. • The drive enable output is active while the tuning profile is running.
TestStatus	TRUE	The axis is running a testing process.

Example: When the input conditions are true, the controller runs the *encoder* diagnostic test on *axis1*.

Relay Ladder



MRHD Ladder Example

Structured Text

```
MRHD(Axis1,MRHD_1,Marker);
```

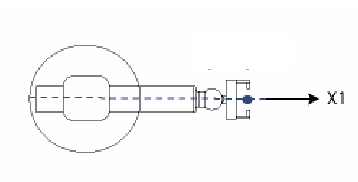
Notes:

Motion Coordinated Instructions
(MCLM, MCCM, MCCD, MCS, MCSD, MCT, MCTP, MCSR)

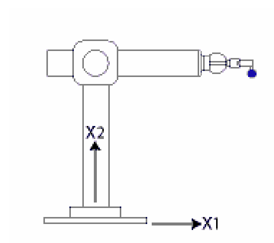
Introduction

Use the motion coordinated instructions to move up to three axes in a coordinate system.

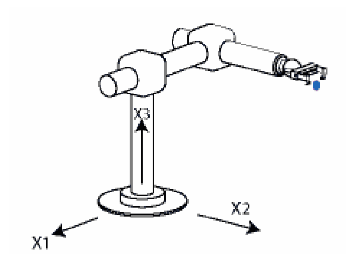
Coordinate Systems with Orthogonal Axes



One Dimension Cartesian Coordinate System

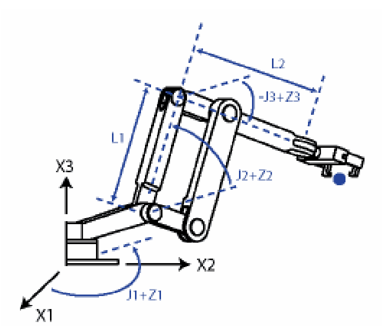


Two Dimension Cartesian Coordinate System



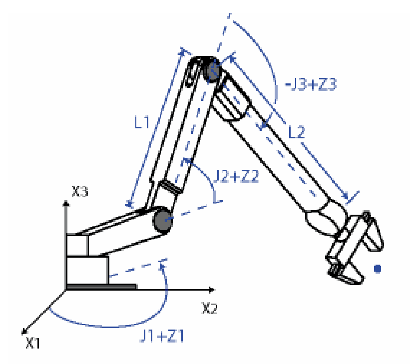
Three Dimension Cartesian Coordinate System

Coordinate Systems with Non-orthogonal Axes



Articulated Dependent Coordinate System

or



Articulated Independent Coordinate System

Use this table to choose a motion coordinated instruction.

If you want to	Use this instruction	Available in these languages
Initiate a single or multi-dimensional linear coordinated move for the specified axes within a Cartesian coordinate system.	Motion Coordinated Linear Move (MCLM)	Relay ladder Structured text
Initiate a two- or three-dimensional circular coordinated move for the specified axes within a Cartesian coordinate system.	Motion Coordinated Circular Move (MCCM)	Relay ladder Structured text
Initiate a change in path dynamics for coordinate motion active on the specified coordinate system.	Motion Coordinated Change Dynamics (MCCD)	Relay ladder Structured text
Stop the axes of a coordinate system or cancel a transform.	Motion Coordinated Stop (MCS)	Relay ladder Structured text
Initiate a controlled shutdown of all of the axes of the specified coordinate system.	Motion Coordinated Shutdown (MCSD)	Relay ladder Structured text
Start a transform that links two coordinate systems together.	Motion Coordinated Transform (MCT)[1]	Relay ladder Structured text
Calculate the position of one coordinate system with respect to another coordinate system.	Motion Calculate Transform Position (MCTP)[1]	Relay ladder Structured text
Initiate a reset of all of the axes of the specified coordinate system from the shutdown state to the axis ready state and clear the axis faults.	Motion Coordinated Shutdown Reset (MCSR)	Relay ladder Structured text

(1) You can use this instruction only with 1756-L6x controllers.

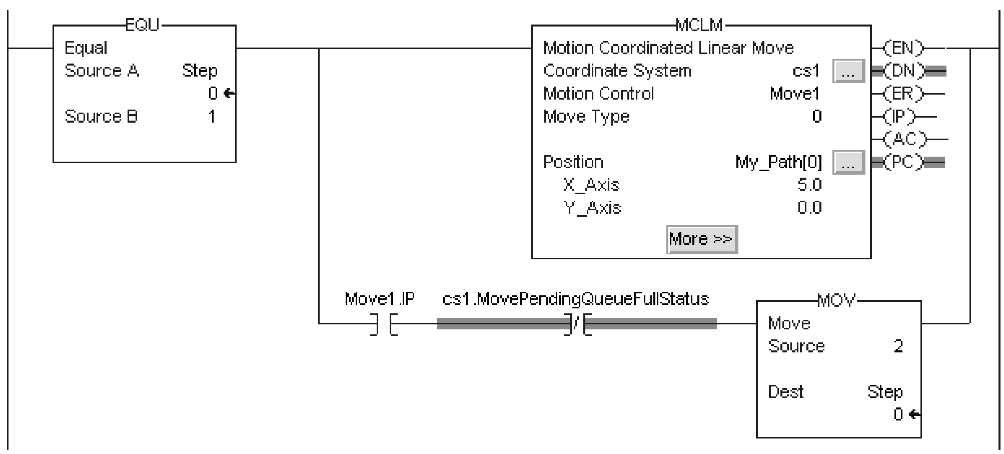
Using Different Termination Types When Blending Instructions

To blend 2 MCLM or MCCM instructions, start the first one and queue the second one. The tag for the coordinate system gives you 2 bits for queueing instructions. Both bits always have the same value because you can queue only one instruction at a time.

When an instruction	Then
enters the queue	• MovePendingStatus bit = 1 • MovePendingQueueFullStatus bit = 1 • You can't queue another instruction.
leaves the queue and starts	• MovePendingStatus bit = 0 • MovePendingQueueFullStatus bit = 0 • You can queue another instruction.

For example, the following ladder diagram uses Coordinate System cs1 to blend Move1 into Move2.

If Step = 1 then
Move1 starts and moves the axes to a position of 5, 0
And once Move1 is in process
And there is room to queue another move
Step = 2



Continued on next page

If Step = 2 then

Move1 is already happening.

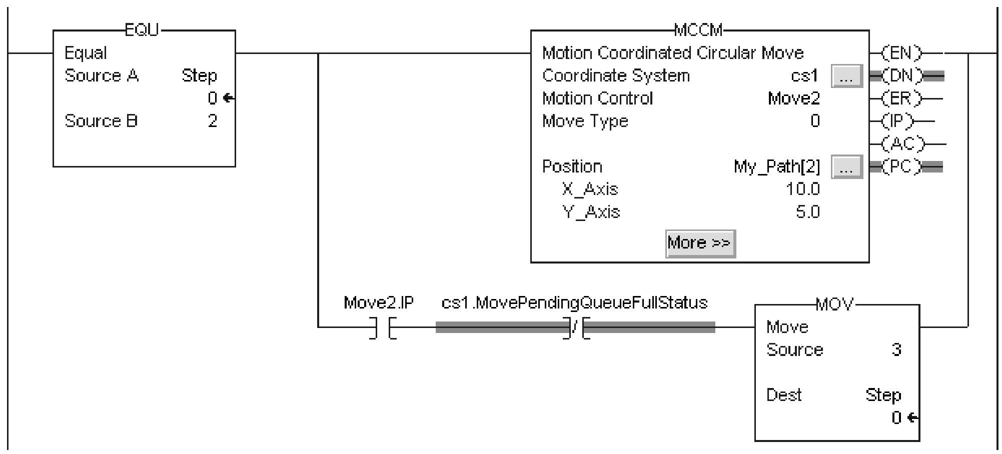
Move2 goes into the queue and waits for Move1 to complete.

When Move1 is complete, Move2 moves the axes to a position of 10, 5.

And once Move2 is in process

And Move2 comes off the queue and starts

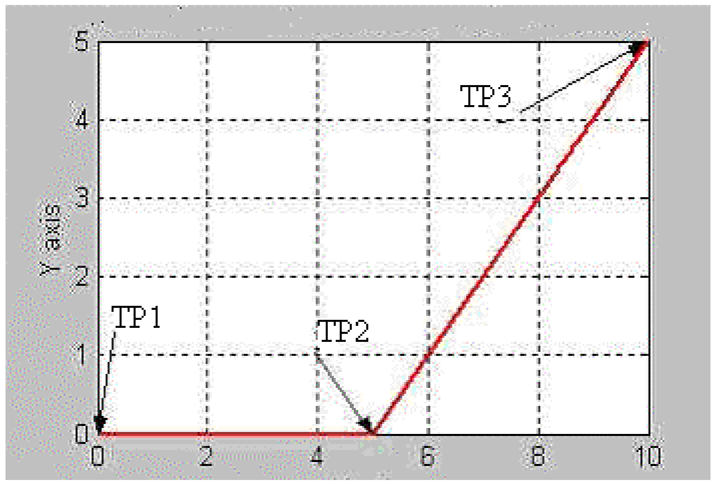
Step = 3



The Termination Type operand for the MCLM or MCCM instruction specifies how the currently executing move gets terminated. The following illustrations show the states of instruction bits and coordinate system bits that get affected at various transition points.

Bit States at Transition Points of Blended Move Using Actual Tolerance or No Settle

linear → linear move

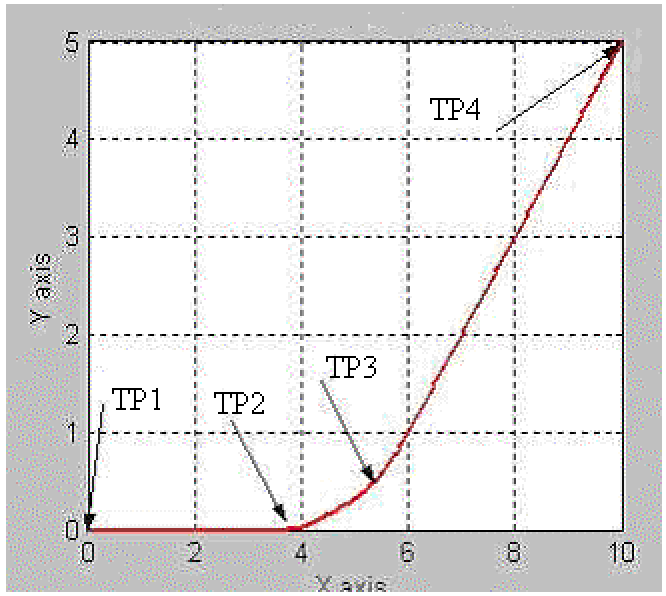


The following table shows the Bit Status at the various transition points shown in the preceding graph with Termination Type of either Actual Tolerance or No Settle.

Bit	TP1	TP2	TP3
Move1.DN	T	T	T
Move1.IP	T	F	F
Move1.AC	T	F	F
Move1.PC	F	T	T
Move2.DN	T	T	T
Move2.IP	T	T	F
Move2.AC	F	T	F
Move2.PC	F	F	T
cs1.MoveTransitionStatus	F	F	F
cs1.MovePendingStatus	T	F	F
cs1.MovePendingQueueFullStatus	T	F	F

Bit States at Transition Points of Blended Move Using No Decel

linear → linear move

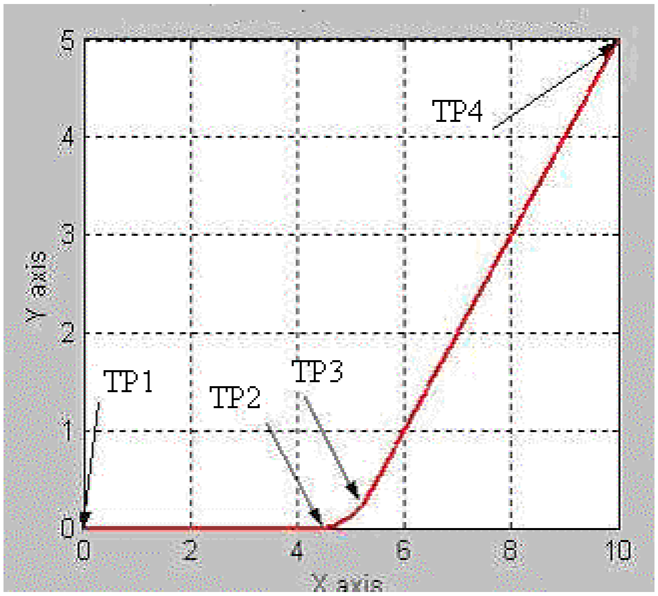


The following table shows the Bit Status at the various transition points shown in the preceding graph with Termination Type of No Decel. For No Decel Termination Type distance-to-go for transition point TP2 is equal to deceleration distance for the Move1 instruction.

Bit	TP1	TP2	TP3	TP4
Move1.DN	T	T	T	T
Move1.IP	T	F	F	F
Move1.AC	T	F	F	F
Move1.PC	F	T	T	T
Move2.DN	T	T	T	T
Move2.IP	T	T	T	F
Move2.AC	F	T	T	F
Move2.PC	F	F	F	T
cs1.MoveTransitionStatus	F	T	F	F
cs1.MovePendingStatus	T	F	F	F
cs1.MovePendingQueueFullStatus	T	F	F	F

Bit States at Transition Points of Blended Move Using Command Tolerance

linear → linear move

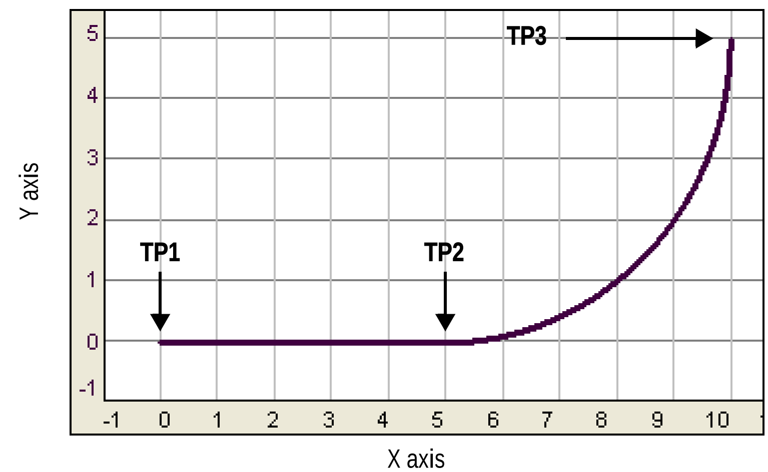


The following table shows the Bit Status at the various transition points shown in the preceding graph with Termination Type of Command Tolerance. For Command Tolerance Termination Type distance-to-go for transition point TP2 is equal to command tolerance for the coordinate system cs1.

Bit	TP1	TP2	TP3	TP4
Move1.DN	T	T	T	T
Move1.IP	T	F	F	F
Move1.AC	T	F	F	F
Move1.PC	F	T	T	T
Move2.DN	T	T	T	T
Move2.IP	T	T	T	F
Move2.AC	F	T	T	F
Move2.PC	F	F	F	T
cs1.MoveTransitionStatus	F	T	F	F
cs1.MovePendingStatus	T	F	F	F
cs1.MovePendingQueueFullStatus	T	F	F	F

Bit States at Transition Points of Blended Move Using Follow Contour Velocity Constrained or Unconstrained

linear → circular move



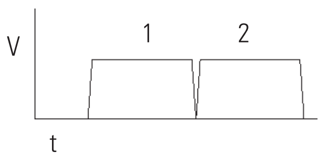
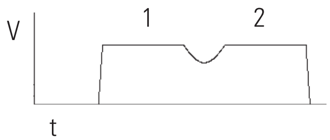
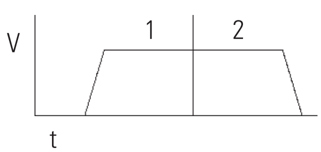
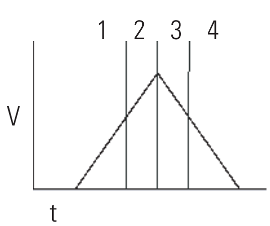
The following table shows the bits at the transition points.

Bit	TP1	TP2	TP3
Move1.DN	T	T	T
Move1.IP	T	F	F
Move1.AC	T	F	F
Move1.PC	F	T	T
Move2.DN	T	T	T
Move2.IP	T	T	F
Move2.AC	F	T	F
Move2.PC	F	F	T
cs1.MoveTransitionStatus	F	F	F
cs1.MovePendingStatus	T	F	F
cs1.MovePendingQueueFullStatus	T	F	F

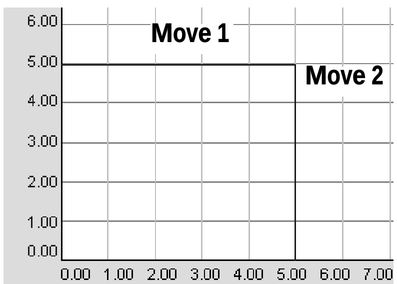
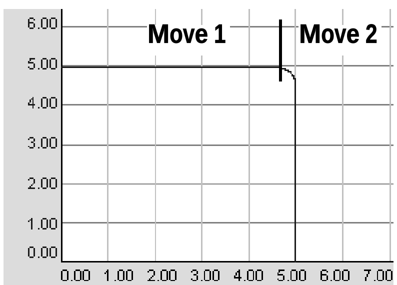
Choose a termination type

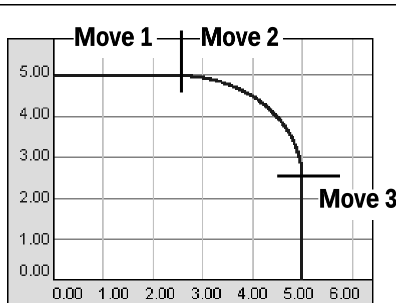
The termination type determines when the instruction is complete. It also determines how the instruction blends its path into the queued MCLM or MCCM instruction, if there is one.

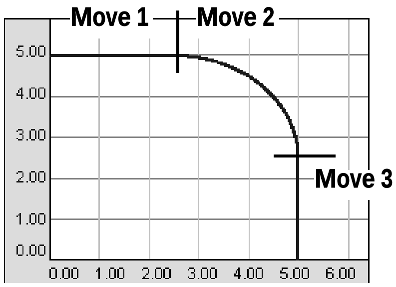
1. Choose a termination type.

<table>
<tr><th>If you want the axes to (vector speeds)</th><th>And you want the instruction to complete when the</th><th>Then use this termination type</th></tr>
<tr><td rowspan="2">stop between moves

</td><td>both of these happen:
• Command position equals target position.
• The vector distance between the target and actual positions is less than or equal to the Actual Position Tolerance of the coordinate system.</td><td>0 - Actual Tolerance</td></tr>
<tr><td>command position equals the target position</td><td>1 - No Settle</td></tr>
<tr><td rowspan="2">keep the speed constant except between moves

</td><td>command position gets within the Command Position Tolerance of the coordinate system</td><td>2 - Command Tolerance</td></tr>
<tr><td>axes get to the point at which they must decelerate at the deceleration rate</td><td>3 - No Decel</td></tr>
<tr><td>transition into or out of a circle without stopping

</td><td>⇒ ⇒ ⇒</td><td>4 - Follow Contour Velocity Constrained</td></tr>
<tr><td>accelerate or decelerate across multiple moves

</td><td>⇒ ⇒ ⇒</td><td>5 - Follow Contour Velocity Unconstrained</td></tr>
</table>

2. Make sure this is the right choice for you.

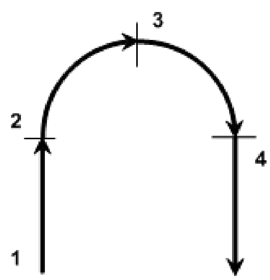
<table>
<tr><th>Termination type</th><th>Example path</th><th>Description</th></tr>
<tr><td>0 - Actual Tolerance</td><td>
</td><td>The instruction stays active until both of these happen:
• Command position equals target position.
• The vector distance between the target and actual positions is less than or equal to the Actual Position Tolerance of the coordinate system.
At that point, the instruction is complete and a queued MCLM or MCCM instruction can start.
Important: Make sure that you set the actual tolerance to a value that your axes can reach. Otherwise the instruction stays in process.</td></tr>
<tr><td>1 - No Settle</td><td>
</td><td>The instruction stays active until the command position equals the target position. At that point, the instruction is complete and a queued MCLM or MCCM instruction can start.</td></tr>
<tr><td>2 - Command Tolerance</td><td>
</td><td>The instruction stays active until the command position gets within the command position tolerance of the coordinate system. At that point, the instruction is complete and a queued MCLM or MCCM instruction can start.

If you don’t have a queued MCLM or MCCM instruction, the axes stop at the target position.</td></tr>
<tr><td>3 - No Decel</td><td>
</td><td>The instruction stays active until the axes get to the deceleration point. At that point, the instruction is complete and a queued MCLM or MCCM instruction can start.
• The deceleration point depends on whether you use a trapezoidal or S-curve profile.
• If you don’t have a queued MCLM or MCCM instruction, the axes stop.</td></tr>
<tr><td>4 - Follow Contour Velocity Constrained</td><td>
</td><td>The instruction stays active until the axes get to the target position. At that point, the instruction is complete and a queued MCLM or MCCM instruction can start.
• This termination type works best with tangential transitions. For example, use it to go from a line to a circle, a circle to a line, or a circle to a circle.
• The axes follow the path.
• If the moves are long enough, the axes won’t decelerate between moves. If the moves are too short, the axes decelerate between moves.</td></tr>
</table>

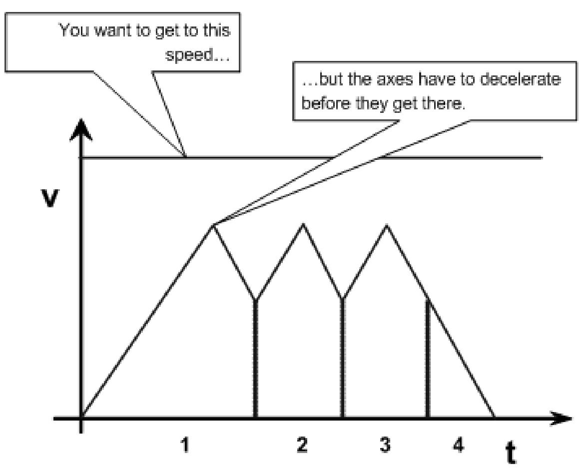
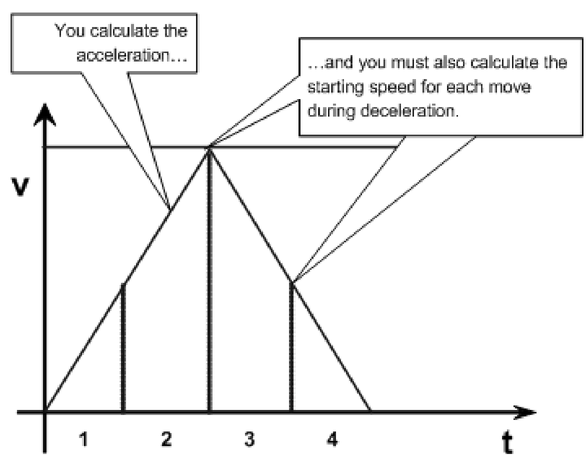
Termination type	Example path	Description
5 - Follow Contour Velocity Unconstrained		This termination type is similar to the contour velocity constrained. It has these differences: • Use this termination type to get a triangular velocity profile across several moves. This reduces jerk. • You must calculate the acceleration for the triangular velocity profile. • You must also calculate the starting speed for each move in the deceleration half of the profile.

How do I get a triangular velocity profile?

Suppose you want to program a pick and place action in 4 moves. And to keep jerk low you want to use a triangular velocity profile.

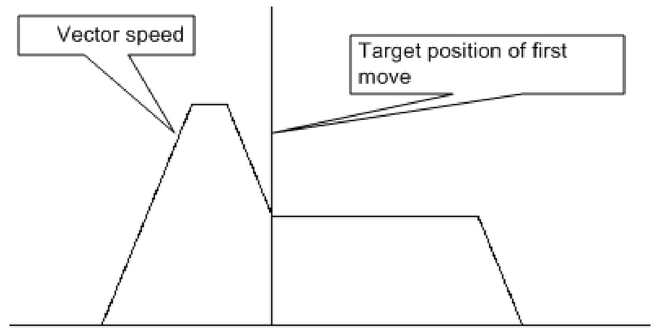
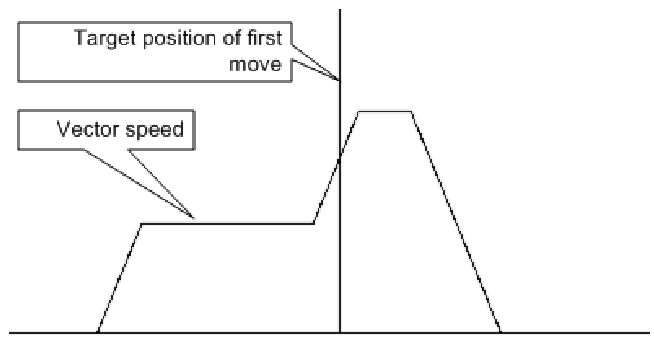
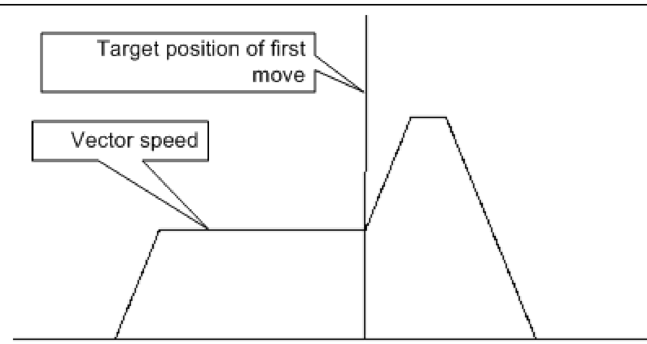


For this situation, use termination type 5. The other termination types may not let you get to the speed you want.

Termination types 2, 3, or 4	Termination type 5
	
The axes won't go any faster than a speed that lets them decelerate to 0 without overshooting the target position. The shorter the move, the lower the maximum speed.	The axes accelerate to the speed that you want. You must calculate the starting speed for each move in the deceleration half of the profile.

Blending Moves at Different Speeds

You can blend MCLM and MCCM instructions where the vector speed of the second instruction is different from the vector speed of the first instruction.

If the next move is	And the termination type of the first move is	Then
slower	2 - Command Tolerance 3 - No Decel 4 - Contour Velocity Constrained 5 - Contour Velocity Unconstrained	
faster	2 - Command Tolerance 3 - No Decel	
	4 - Contour Velocity Constrained 5 - Contour Velocity Unconstrained	

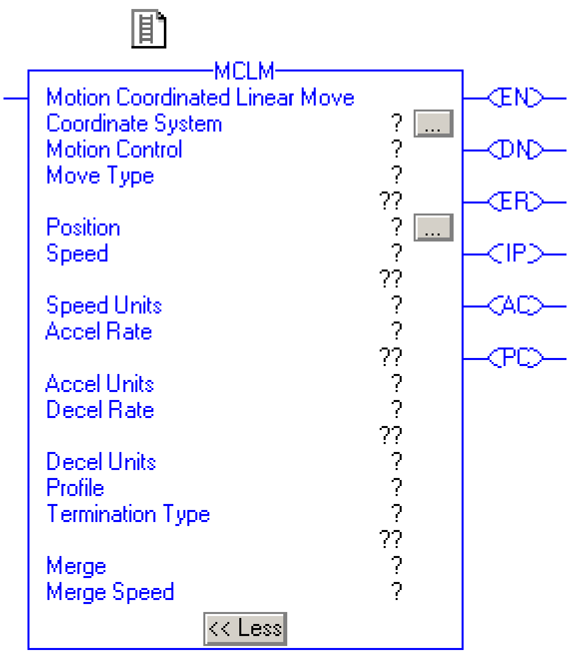
Motion Coordinated Linear Move (MCLM)

Use the MCLM instruction to start a single or multi-dimensional linear coordinated move for the specified axes within a Cartesian coordinate system. You can define the new position as either absolute or incremental.

ATTENTION

Use a motion control tag only once. Do not re-use it in another instruction. Otherwise, you can cause unexpected equipment motion and injure people.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Coordinate System	COORDINATE_SYSTEM	tag	Coordinated group of axes.
Motion Control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.
Move Type	SINT, INT, or DINT	immediate or tag	Select the Move Type: 0 = Absolute 1 = Incremental
Position	REAL	array tag [ ]	[coordination units]
Speed	SINT, INT, DINT, or REAL	immediate or tag	[coordination units]
Speed Units	SINT, INT, or DINT	immediate	0 = Units per Sec 1 = % of Maximum
Accel Rate	SINT, INT, DINT, or REAL	immediate or tag	[coordination units]
Accel Units	SINT, INT, or DINT	immediate	0 = Units per Sec^2 1 = % of Maximum
Decel Rate	SINT, INT, DINT, or REAL	immediate or tag	[coordination units]
Decel Units	SINT, INT, or DINT	immediate	0 = Units per Sec^2 1 = % of Maximum
Profile	SINT, INT, or DINT	immediate	0 = Trapezoidal 1 = S-Curve

Operand	Type	Format	Description
Termination Type	SINT, INT, or DINT	immediate or tag	0 = Actual Tolerance 1 = No Settle 2 = Command Tolerance 3 = No Decel 4 = Follow Contour Velocity Constrained 5 = Follow Contour Velocity Unconstrained See Choose a termination type on page 259.
Merge	SINT, INT, or DINT	immediate	0 = Disabled 1 = Coordinated Motion 2 = All Motion
Merge Speed	SINT, INT, or DINT	immediate	0 = Programmed 1 = Current

```
MCLM(CoordinateSystem,Motion
Control,MoveType,Position,
Speed,SpeedUnits,AccelRate,A
ccelUnits,DecelRate,
DecelUnits,VelocityProfile,
TerminationType,Merge,
MergeSpeed);
```

Structured Text

The operands are the same as those for the relay ladder MCLM instruction.

> When entering enumerations for the operand value in Structured Text, multiple word enumerations must be entered without spaces. For example: when entering Decel Units the value should be entered as unitspersec^2 rather than Units per Sec^2 as displayed in the ladder logic.

For the operands that have enumerated values, enter your selection as:

This operand	Has these options which you...	
	enter as text	**or enter as a number**
Move Type	no enumeration	0 (Absolute) 1 (Incremental)
Speed Units	unitspersec %ofmaximum	0 1
Accel Units	unitspersec^2 %ofmaximum	0 1
Decel Units	unitspersec^2 %ofmaximum	0 1
Profile	trapezoidal scurve	0 1

This operand	Has these options which you...	
	enter as text	or enter as a number
Termination Type	no enumeration	0 (Actual Tolerance) 1 (No Settle) 2 (Command Tolerance) 3 (No Decel) 4 (Follow Contour Velocity Constrained) 5 (Follow Contour Velocity Unconstrained) See Choose a termination type on page 259.
Merge	Disabled Coordinatedmotion Allmotion	0 1 2
Merge Speed	Programmed Current	0 1

Description

The Motion Coordinated Linear Move (MCLM) instruction performs a linear move using up to three (3) axes statically coupled to the coordinate system as primary axes in a Cartesian coordinate system. You specify whether to use absolute or incremental target position and the desired speed. The actual speed is a function of both the mode of the move (commanded speed or percent of maximum speed) and the combination of primary axes that are commanded to move. The vector speed of the move is based on the time it takes to complete a vector move using the programmed axes. Each axis is commanded to move at a speed that allows all axes to reach the endpoint (target position) at the same time.

ATTENTION

If You Use An S-curve Profile

Be careful if you change the speed, acceleration, deceleration, or jerk while an axis is accelerating or decelerating along an S-curve profile. You can cause an axis to **overshoot its speed or reverse direction**.

For more information, see Troubleshoot Axis Motion on page 367.

Coordinate System

The Coordinate System operand specifies the set of motion axes that define the dimensions of a Cartesian coordinate system. For this release the coordinate system supports up to three (3) primary axes. Only those axes configured as primary axes are included in the coordinate velocity calculations.

Motion Control

The following control bits are affected by the MCLM instruction.

Mnemonic	Description
.EN (Enable) Bit 31	The Enable Bit is set when the rung transitions from false to true and resets when the rung goes from true to false.
.DN (Done) Bit 29	The Done Bit sets when the coordinated instruction has been verified and queued successfully. Because it is set at the time it is queued it may appear as set when a runtime error is encountered during the verify operation after it comes out of the queue. It resets when the rung transitions from false to true.
.ER (Error) Bit 28	The Error Bit is reset when the rung transitions from false to true. It is set when the coordinated move has not successfully initiated. It is also set with the Done Bit when a queued instruction encounters a runtime error.
.IP (In Process) Bit 26	The In Process Bit is set when the coordinated move is successfully initiated. It is reset when there is no succeeding move and the coordinated move reaches the new position, or when there is a succeeding move and the coordinated move reaches the specifications of the termination type, or when the coordinated move is superseded by another MCLM or MCCM instruction with a merge type of Coordinated Move, or when terminated by an MCS instruction.
.AC (Active) Bit 23	When you have a coordinated move instruction queued, the Active Bit lets you know which instruction is controlling the motion. It sets when the coordinated move becomes active. It is reset when the Process Complete bit is set or when the instruction is stopped.
.PC (Process Complete) Bit 27	The Process Complete Bit is reset when the rung transitions from false to true. It is set when there is no succeeding move and the coordinated move reaches the new position, or when there is a succeeding move and the coordinated move reaches the specified Termination Type.
.ACCEL (Acceleration Bit) Bit 01	The Acceleration Bit sets while the coordinated move is in the acceleration phase. It resets while the coordinated move is in the constant velocity or deceleration phase, or when coordinated motion concludes.
.DECEL (Deceleration Bit) Bit 02	The Deceleration bit sets while the coordinated move is in the deceleration phase. It resets while the coordinated move is in the constant velocity or acceleration phase, or when coordinated motion concludes.

Move Type

The Move Type operand specifies the method used to indicate the coordinated move path. The Move Type can be either Absolute or Incremental.

Absolute

When the Move Type is Absolute, the axes move via a linear path to the position defined by the position array at the Speed, Accel Rate and Decel Rate as specified by the operands.

When the axis is configured for rotary operation, an Absolute Move type behaves in the same manner as for a linear axis. When the axis position exceeds the Unwind Parameter, it is unwound. In this way, axis position is never greater than the Unwind value nor less than zero.

The sign of the specified position is interpreted by the interpolator and can be either positive or negative. Negative position values instruct the interpolator to move the rotary axis in a negative direction to obtain the desired absolute position, while positive values indicate that positive motion is desired to reach the target position. When the position value is greater than the unwind value, an error is generated. The axis never moves through more than one unwind cycle before stopping at an absolute position.

Incremental

When the Move Type is Incremental, the coordinate system moves the distance as defined by the position array at the specified Speed, using the Accel and Decel rates determined by the respective operands, via a linear path.

The specified distance is interpreted by the interpolator and can be positive or negative. Negative position values instruct the interpolator to move the axis in a negative direction, while positive values indicate positive motion is desired to reach the target position. Motion greater than one unwind cycle is allowed in Incremental mode.

MCLM Move Type Examples

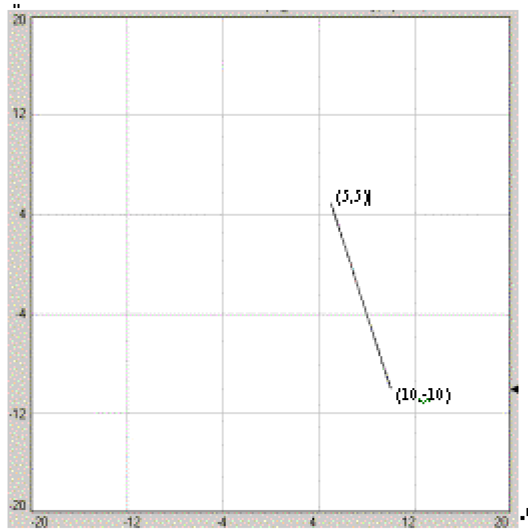
The following examples show the use of the MCLM with Move Type of Absolute (first example) and Incremental (second example) to arrive at the same result. The basic assumptions are:

- The 2 axes, Axis0 and Axis1, are both members of the coordinate system, Coordinated_sys.
- Axis0 and Axis1 are orthogonal to each other.

- Coordinated_sys is initially at (5,5) units.

Move the Coordinated_sys linearly to (10,-10) units at the vector speed of 10.0 units per second with the acceleration and deceleration values of 5.0 units per second^2.

The following graph is the path generated by the above assumptions.



Resulting Plot of Path

The total distance travelled along the path of the vector is:

DAxis0 = 10 - 5 = 5

DAxis1 = -10 - 5 = -15

$$TotalDist = \sqrt{(DAxis0)^2 \oplus (DAxis1)^2} = 15.811388$$

The vector speed of the selected axes is equal to the specified speed in the position units per second. The speed of each axis is proportional to the distance traveled by the axis divided by the square root of the sum of the squares of the distance moved by all axes. The actual speed of Axis0 is the following percent of the vector speed of the move.

%Axis0 Speed = |Daxis0 / TotalDist| = |5 / 15.811388| = .3162 = 31.62%

%Axis1 Speed = |Daxis1 / TotalDist| = |-15 / 15.811388| = .9487 = 94.87%

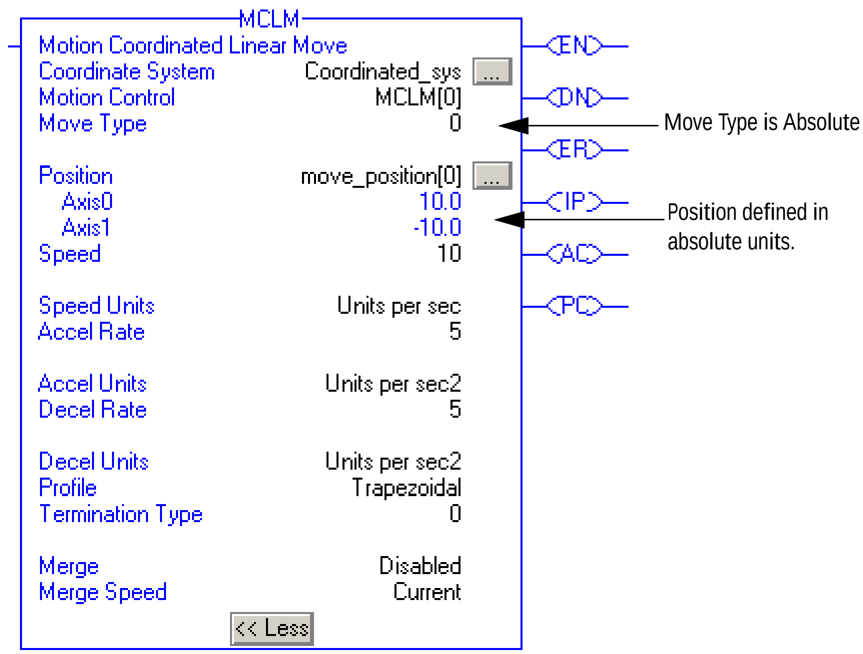
For the example,

Axis0 Speed = .3162 * 10.0 = 3.162 units/sec.

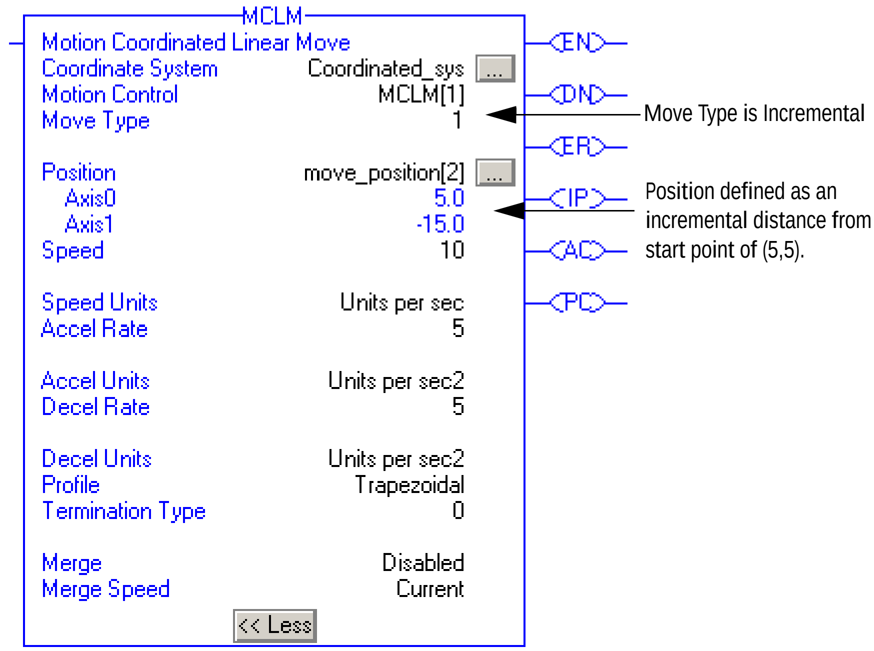
Axis1 Speed = .9487 * 10.0 = 9.487 units/sec.

The acceleration and deceleration for each axis is the same percentage as speed.

The following ladder instructions show the ladder logic necessary to achieve this path using Move Type = Absolute and Move Type = Incremental, respectively.



MCLM Ladder Instruction with Move Type of Absolute



MCLM Ladder Instruction with Move Type of Incremental

MCLM Instruction With Rotary Axes Examples

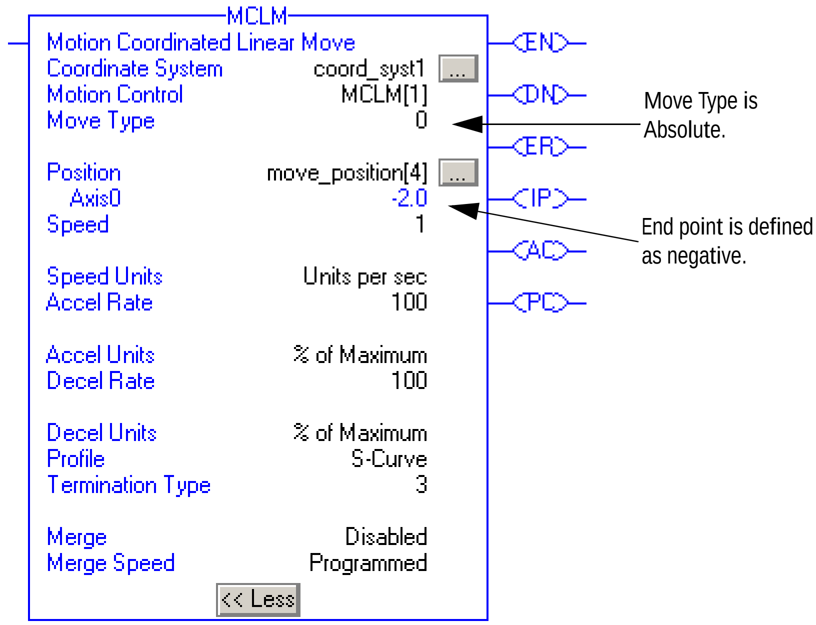
The following examples show the plot of the paths for MCLM instructions that have axes defined as Rotary.

MCLM with One Rotary Axis and Move Type of Absolute

The first example uses a coordinate system of one axis and a Move type of Absolute. The plot of the path is based on the following assumptions:

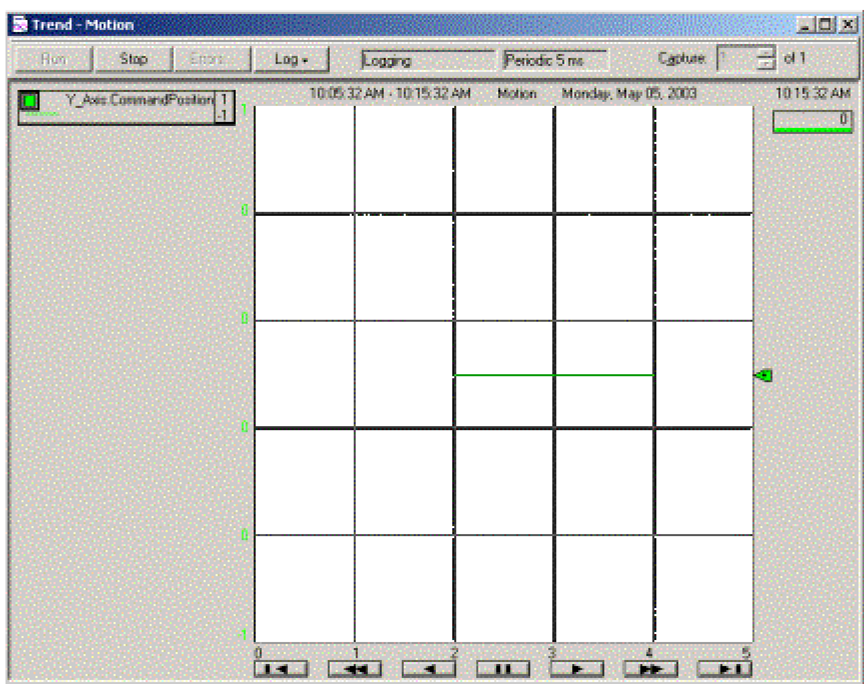
- 1 axis Coordinate System named coord_syst1
- Axis0 is Rotary with an unwind of 5 revs.
- Start position is 4.

- End position is -2.



MCLM Ladder Instruction with Move Type of Absolute

The resultant plot of the move's path is shown in the following illustration.



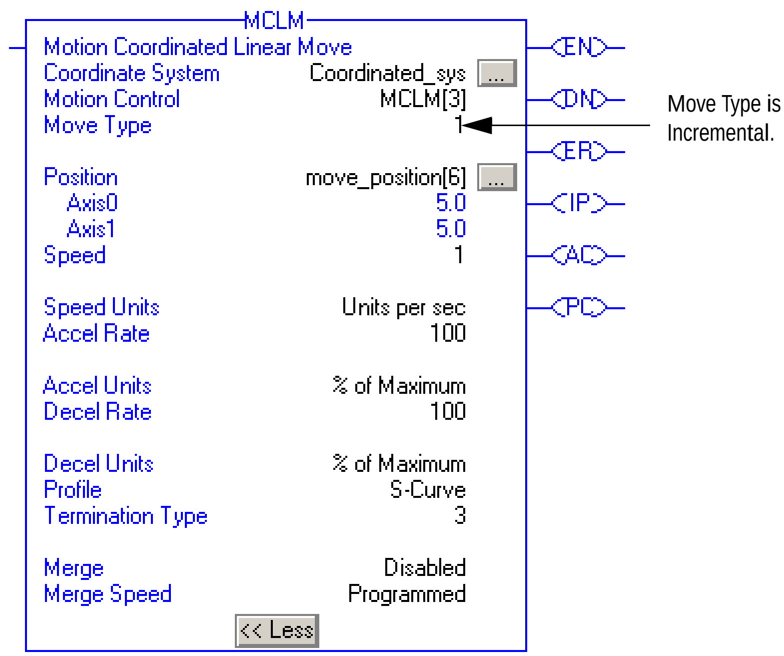
Plot of MCLM with One Rotary Axis and Move Type of Absolute

The endpoint was a negative value therefore the axis travelled in a negative direction moving from 4 to 2. It did not travel through the unwind. For this move the endpoint is required to fit within the absolute position defined by the rotary unwind of the axis. Therefore an unwind value of 6 or -6 would not be valid.

MCLM with Two Rotary Axes and Move Type of Incremental

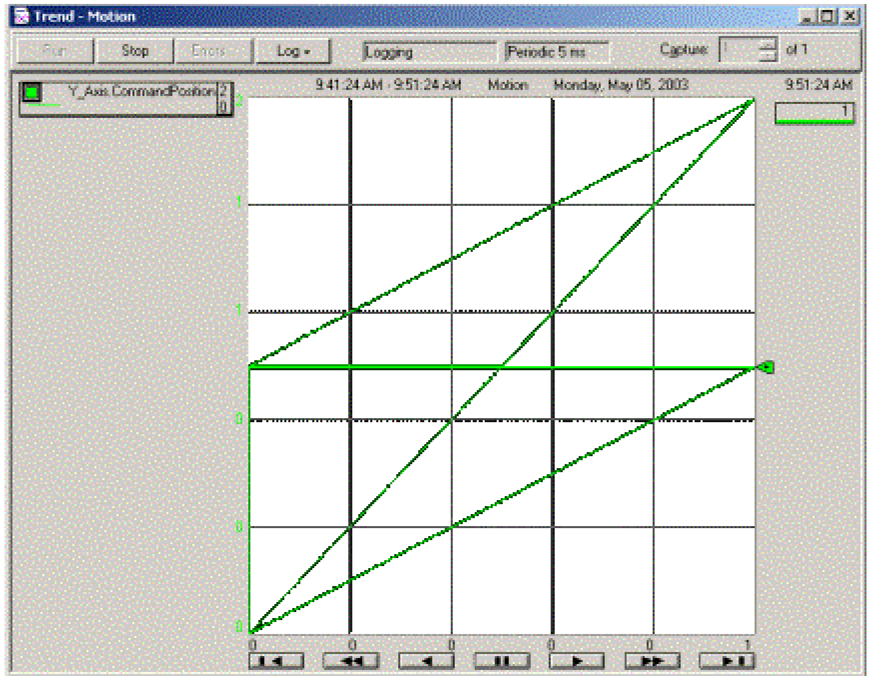
The second MCLM example with rotary axes has two rotary axes and a Move Type of Incremental. The plot of the path has the following assumptions:

- 2 axis Coordinate System named Coordinated_sys
- Axis0 is Rotary with an unwind of 1 revs.
- Axis1 is Rotary with an unwind of 2 revs.
- Start position is 0,0.
- Increment to end position is 5,5.



MCLM Ladder Instruction with Move Type of Incremental

The previous MCLM ladder program produces the following plot of the moves' path.



Plot of MCLM with Two Rotary Axes and Move Type of Incremental

In the plot above the axes travel a reverse "z" pattern two and one half times, stopping at an actual position of 0,1. This equates to 5 revolutions/unwinds for Axis0 and 2.5 revolutions/unwinds for Axis1. The position increments for this move are positive therefore, the axes move in a positive direction with Axis0 moving from 0 to 1 and Axis1 moving from 0 to 2. In this example the endpoint is not required to fit within the absolute position defined by the rotary unwind of the axes. The path of the coordinated motion is determined in linear space but position of the axes is limited by the rotary configuration.

Position

A one dimensional array, whose dimension is defined to be at least the equivalent of the number of axes specified in the coordinate system. The Position array defines either the new absolute or incremental position.

Speed

The Speed operand defines the maximum vector speed along the path of the coordinated move.

Speed Units

The Speed Units operand defines the units applied to the Speed operand either directly in coordination units of the specified coordinate system or as a percentage of the maximum values defined in the coordinate system.

Accel Rate

The Accel Rate operand defines the maximum acceleration along the path of the coordinated move.

Accel Units

The Accel Units operand defines the units applied to the Accel Rate operand either directly in coordination units of the specified coordinate system or as a percentage of the maximum values defined in the coordinate system.

Decel Rate

The Decel Rate operand defines the maximum deceleration along the path of the coordinated move.

Decel Units

The Decel Units operand defines the units applied to the Decel Rate operand either directly in coordination units of the specified coordinate system or as a percentage of the maximum values defined in the coordinate system.

Profile

The Profile operand determines whether the coordinated move uses a trapezoidal or S-Curve velocity profile.

The ControlLogix motion controller provides trapezoidal (linear acceleration and deceleration), and S-Curve (controlled jerk) velocity profiles. A guide to the effects of these motion profiles on various application requirements is given below.

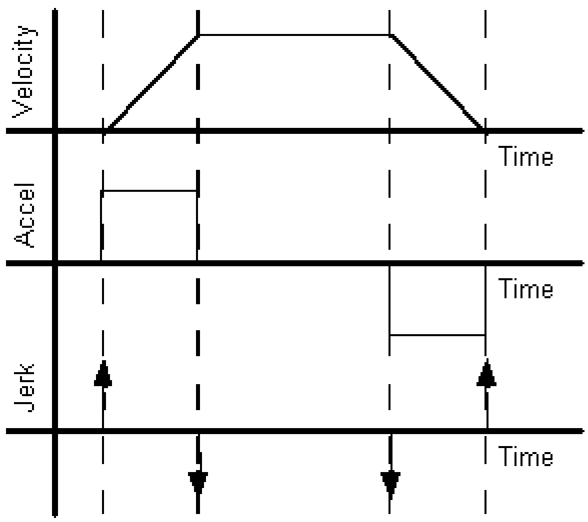
Velocity Profile Effects

Profile	**ACC/DEC**	**Motor**	**Priority of Control**
Type	Time	Stress	**Highest to Lowest**

Profile	ACC/DEC	Motor	Priority of Control			
Trapezoidal	Fastest	Worst	Acc/Dec	Velocity	Position	
S-Curve	2X Slower	Best	Jerk	Acc/Dec	Velocity	Position

Trapezoidal

The trapezoidal velocity profile is the most commonly used profile since it provides the most flexibility in programming subsequent motion and the fastest acceleration and deceleration times. The maximum change in velocity is specified by acceleration and deceleration. Since jerk is not a factor for trapezoidal profiles, it is considered infinite and is shown as series of vertical lines in the following graph.



Trapezoidal Accel/Decel Time

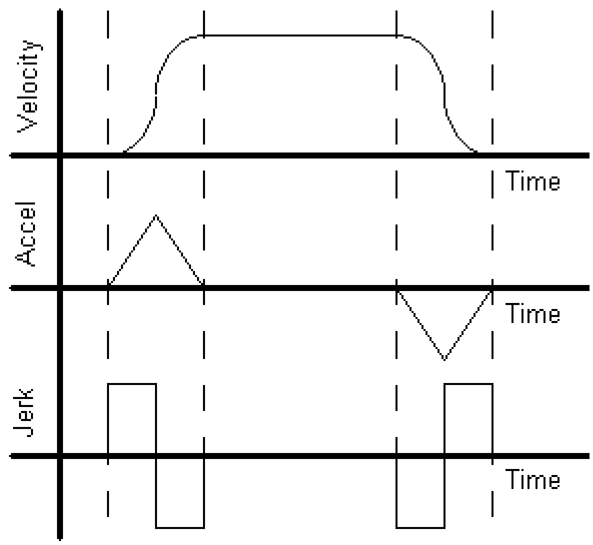
S-Curve

S-Curve velocity profiles are most often used when the stress on the mechanical system and load needs to be minimized. The S-Curve profile, however, sacrifices acceleration and deceleration time compared to the trapezoidal. The maximum rate at which velocity can accelerate or decelerate is further limited by jerk. The Jerk rate is calculated as follows:

$$\text{Accel Jerk} = (\text{Max Accel})^2 / \text{Max Velocity}$$

$$\text{Decel Jerk} = (\text{Max Decel})^2 / \text{Max Velocity}$$

Coordinate motion acceleration and deceleration jerk rate calculations are performed when an MCLM, MCCM, MCCD, or MCS instruction is started. The calculated Jerk Rate produces triangular acceleration and deceleration profiles, as shown in the following diagram.



S-Curve Accel/Decel Time

See the MCCD instruction for more details about the impact changes made by an MCCD instruction.

Merge

The Merge operand determines whether or not to turn the motion of all specified axes into a pure coordinated move. The Merge options include: Merge Disabled, Coordinated Motion, or All Motion.

Merge Disabled

Any currently executing single axis motion instructions involving any axes defined in the specified coordinate system are not affected by the activation of this instruction, and results in superimposed motion on the affected axes. Also, any coordinated motion instructions involving the same specified coordinate system runs to completion based on its termination type.

Coordinated Motion

Any currently executing coordinated motion instructions involving the same specified coordinate system are terminated. The active motion is blended into the current move at the speed defined in the merge speed parameter. Any pending coordinated motion instructions are cancelled. Any currently executing system single axis motion instructions involving any axes defined in the specified coordinate system will not be affected by the activation of this instruction, and will result in superimposed motion on the affected axes.

All Motion

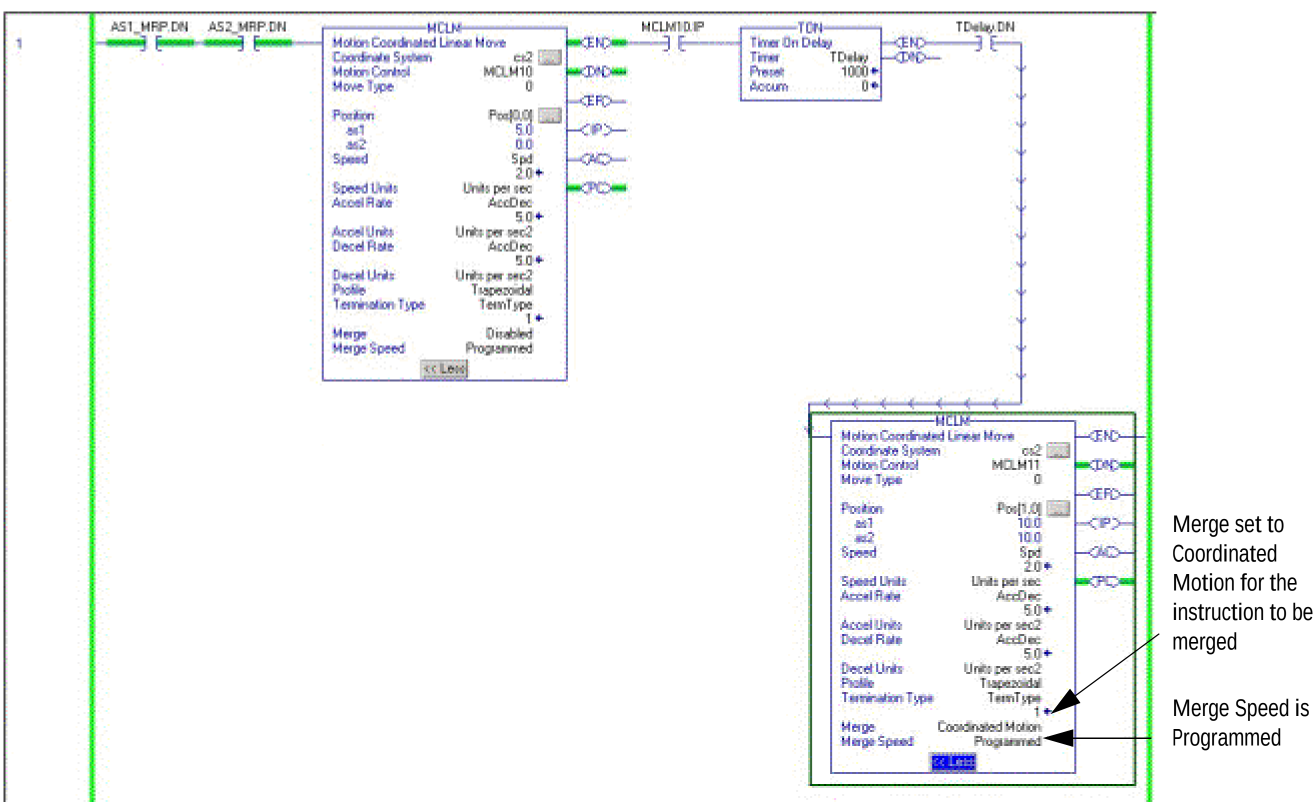
Any currently executing single axis motion instructions involving any axes defined in the specified coordinate system and any currently executing coordinated motion instructions are terminated. The prior motion is merged into the current move at the speed defined in Merge Speed parameter. Any pending coordinated move instructions are cancelled.

Merge Speed

The Merge Speed operand is always set to Programmed in current version. Programmed speed is used as the maximum speed along the path of the coordinated move.

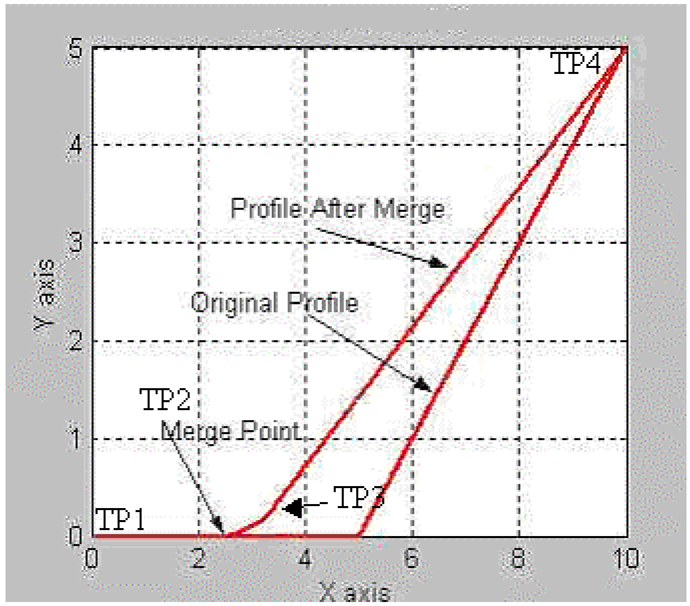
Merge Example

The MCLM ladder diagram uses Coordinate System cs2 to merge an mclm10 instruction with a target absolute position of (5,0) into an mclm11 instruction with the target position of (10,5).



Ladder Diagram Showing Merge

If the axes are orthogonal to each other, and the coordinate system cs2 is initially at (0,0) units, then the motion caused by this diagram depends on the time at which the second instruction is executed. The blending begins as soon as the second move is initiated and the first move is terminated immediately. In the ladder diagram for this example, transition begins when the timer Tdelay expires.



Graph Showing Result of Merge

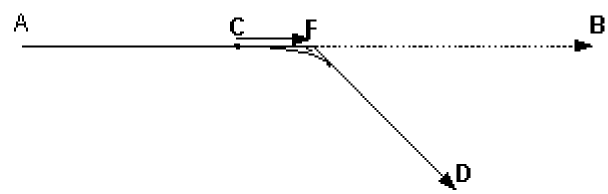
The bit states at various transition points for the merge move.

Bit	TP1	TP2	TP3	TP4
Move1.DN	T	T	T	T
Move1.IP	T	F	F	F
Move1.AC	T	F	F	F
mcclm10.PC	F	T	T	T
Move2.DN	T	T	T	T
Move2.IP	T	T	T	F
Move2.AC	F	T	T	F
Move2.PC	F	F	F	T
cs2.MoveTransitionStatus	F	T	F	F
cs2.MovePendingStatus	T	F	F	F
cs2.MovePendingQueueFullStatus	T	F	F	F

Currently, Coordinated Motion only supports the queueing of one coordinated motion instruction. Therefore the MovePendingStatus bit and the MovePendingQueueFullStatus bit are always the same.

Additional Note On Merging Instructions

A move from point A to point B is initiated as shown in the figure below. When the axis is at point C, a merge to point D is initiated. As a result the current instruction is terminated at point C. The control computes the deceleration distance needed at point C along the vector AB from the current velocity to zero velocity. This distance is shown as vector CF. The imaginary point F is then computed by adding the vector CF to point C. The resultant merged motion from C to D is as shown below. It follows the curved line starting from C then joins the straight line from F to D. This path is identical as if the original programmed move was made from point A to F then from F to D with a Termination Type of No Decel.

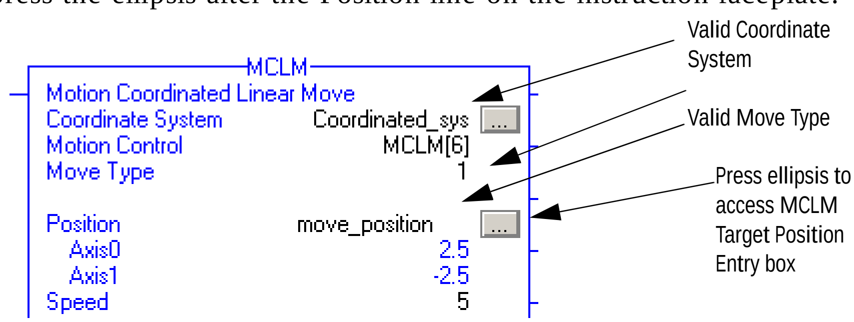


Merge Example

MCLM Target Position Entry Dialog Box

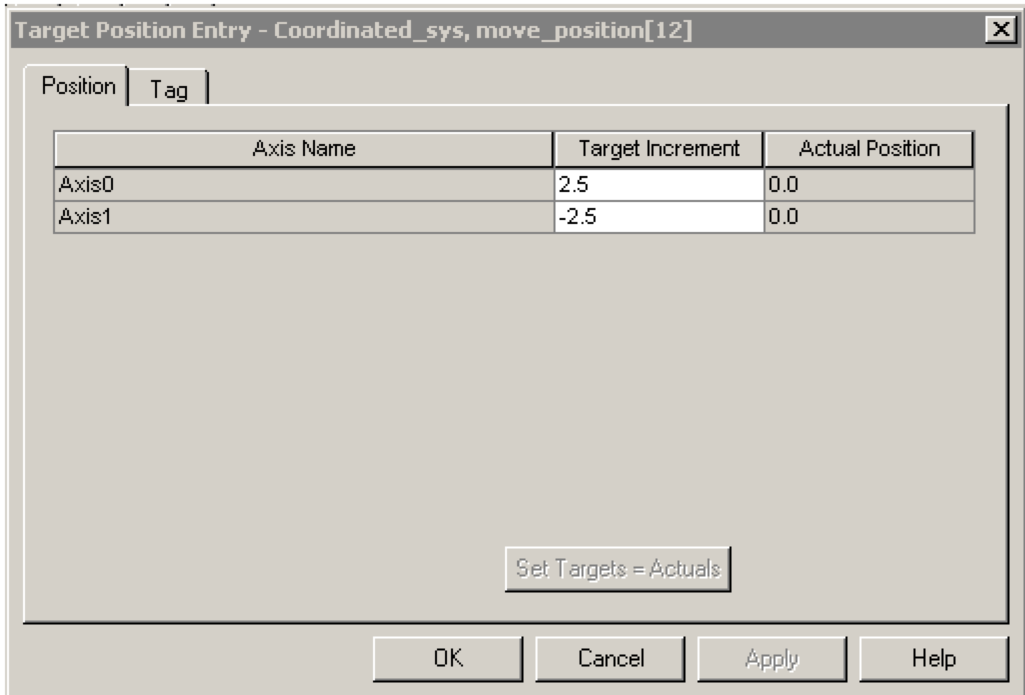
The Target Position Entry Dialog box for the MCLM instruction provides an easy format for editing Position. To gain access to the Target Position Entry dialog box you must have inserted the name of the coordinated system into the instruction, you must have a valid tag name entered in the position field with sufficient elements to handle the number of axes, and you must have selected a valid Move Type.

To access the MCLM Instruction Target Position Entry Dialog box, press the ellipsis after the Position line on the instruction faceplate.



MCLM Ladder Valid Values for Accessing Target Position Entry Box

Pressing the ellipsis button at the Position line of the ladder instruction faceplate calls the following Target Position Entry box for editing the position values.



MCLM Instruction Target Position Entry Dialog Box - Position Tab

The dialog title indicates the Coordinate System and Tag Names for the instruction.

Feature	Description
Axis Name	These fields list the names of each axis contained in the Coordinate System. You cannot alter the axis names in this dialog.
Target Position/Target Increment	This field contains the endpoint or increment of the coordinated move as specified in the instruction faceplate. It is numeric.
Actual Position	These are the current actual positions of the axes in the coordinate system. These positions are updated dynamically when on-line and Coordinate System Auto Tag Update is enabled.
Set Targets = Actuals Button	This button automatically copies the actual position values to the Target Position Column.

The selected Move type governs the appearance and the availability of the Set Targets = Actuals button.

When the Move Type is Absolute, the target column is entitled Target Position and when the Move Type is Incremental, the target column is entitled Target Increment and the Set Targets = Actuals button is disabled (Grayed out).

MCLM is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes: See Error Codes (ERR) for Motion Instructions.

Extended Error Codes: Extended Error codes help to further define the error message given for this particular instruction. Their behavior is dependent upon the Error Code with which they are associated.

The Extended Error Codes for Servo Off State (5), Shutdown State (7), Axis Type Not Servo (8), Axis Not Configured (11), Homing In Process Error (16), and Illegal Axis Data type (38) errors all function in the same fashion. A number between 0 and *n* is displayed for the Extended Error Code. This number is the index to the Coordinate System indicating the axis that is in the error condition.

For Error Code Axis Not Configured (11) there is an additional value of -1 which indicates that Coordinate System was unable to setup the axis for coordinate motion.

For the MCLM instruction, Error Code 13 - Parameter Out of Range, Extended Errors return a number that indicates the offending parameter as listed on the faceplate in numerical order from top to bottom beginning with zero. For example, 2 indicates the parameter value for Move Type is in error.

Referenced Error Code and Number	Extended Error Numeric Indicator	Instruction Parameter	Description
Parameter Out Of Range (13)	2	Move Type	Move Type is either less than 0 or greater than 1.
Parameter Out Of Range (13)	3	Position	The position array is not large enough to provide positions for all the axes in the coordinate system.
Parameter Out Of Range (13)	4	Speed	Speed is less than 0.

Referenced Error Code and Number	Extended Error Numeric Indicator	Instruction Parameter	Description
Parameter Out Of Range (13)	6	Accel Rate	Accel Rate is less than or equal to 0.
Parameter Out Of Range (13)	8	Decel Rate	Decel Rate is less than or equal to 0.
Parameter Out Of Range (13)	11	Termination Type	Termination Type is less than 0 or greater than 3.

For the Error Code 54 – Maximum Deceleration Value is Zero, if the Extended Error returns a positive number (0-*n*) it is referring to the offending axis in the coordinate system. Go to the Coordinate System Properties General Tab and look under the Brackets ([ ])column of the Axis Grid to determine which axis has a Maximum Deceleration value of 0. Click on the ellipsis button next to the offending axis to access the Axis Properties screen. Go to the Dynamics tab and make the appropriate change to the Maximum Deceleration Value. If the Extended Error number is -1, this means the Coordinate System has a Maximum Deceleration Value of 0. Go to the Coordinate System Properties Dynamics Tab to correct the Maximum Deceleration value.

MCLM Changes to Status Bits:

Status Bits provide a means for monitoring the progress of the motion instruction. There are three types of Status Bits that provide pertinent information. They are: Axis Status Bits, Coordinate System Status Bits, and Coordinate Motion Status Bits. When the MCLM instruction initiates, the status bits undergo the following changes.

Axis Status Bits

Bit Name	Meaning
CoordinatedMotionStatus	Sets when the instruction starts. Clears when the instruction ends.

Coordinate System Status Bits

Bit Name	Meaning
MotionStatus	Sets when the MCLM instruction is active and the Coordinate System is connected to its associated axes.

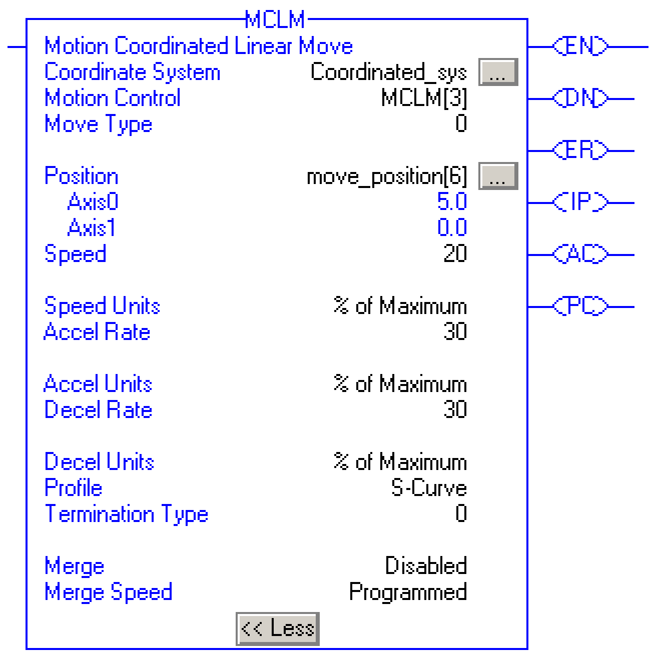
Coordinate Motion Status Bits

Bit Name	Meaning
AccelStatus	Sets when vector is accelerating. Clears when a blend is in process or when vector move is decelerating.
DecelStatus	Sets when vector is decelerating. Clears when a blend is in process or when vector move is accelerating.
ActualPosToleranceStatus	Sets for Actual Tolerance Termination Type only. It sets after the following two conditions are met. 1) Interpolation is complete. 2) The actual distance to the programmed endpoint is less than the configured coordinate system Actual Tolerance value. The bit remains set after an instruction completes. The bit is reset when a new instruction is started.
CommandPosToleranceStatus	Sets for all Termination Types whenever the distance to the programmed endpoint is less than the configured coordinate system Command Tolerance value. The bit remains set after an instruction completes. It resets when a new instruction is started.
StoppingStatus	The Stopping Status bit is cleared when the MCLM instruction initiates.
MoveStatus	Sets when MCLM begins axis motion. Clears on .PC bit of the last motion instruction or when a motion instruction executes which causes a stop.
MoveTransitionStatus	Sets when No Decel or Command Tolerance Termination Type is satisfied. When blending collinear moves the bit is not set because the machine is always on path. It clears when a blend completes, the motion of a pending instruction starts, or a motion instruction executes which causes a stop. Indicates not on path.

Bit Name	Meaning
MovePendingStatus	Sets when one pending coordinated motion instruction is in the instruction queue. Clears when the instruction queue is empty.
MovePendingQueueFullStatus	Sets when the instruction queue is full. It clears when the queue has room for a new coordinated motion instruction.

Currently, Coordinated Motion only supports the queueing of one coordinated motion instruction. Therefore the MovePendingStatus bit and the MovePendingQueueFullStatus bit are always the same.

Example: Relay Ladder



MCLM Ladder Instruction

Structured Text

```
MCLM(Coordinated_sys,MCLM[3],0,move_position[6],5.0,0.0,20
%ofmaximum,30,%ofmaximum,30,%ofmaximum,scurve,0,
disabled,programmed);
```

Motion Coordinated Circular Move (MCCM)

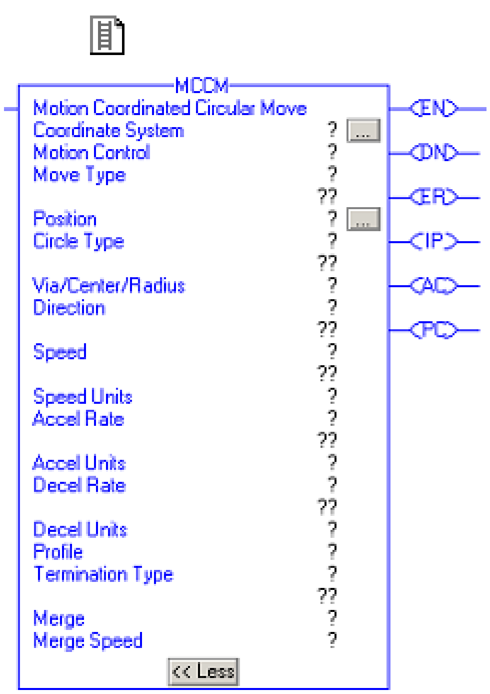
Use the MCCM instruction to initiate a two or three dimensional circular coordinated move for the specified axes within the Cartesian coordinate system. New position is defined as either an absolute or incremental position.

The dimension of the circle is defined by the number of axes contained within the coordinate system. For example, if you have a coordinate system that contained three axes with an MCCM instruction that has motion in only two dimensions, the resultant move is still considered a three dimensional arc or circle.

ATTENTION

Use a motion control tag only once. Do not re-use it in another instruction. Otherwise, you can cause unexpected equipment motion and injure people.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Coordinate System	COORDINATE_SYSTEM	tag	Coordinate group of axes.
Motion Control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.
Move Type	SINT, INT, or DINT	immediate or tag	0 = Absolute 1 = Incremental
Position	REAL	array tag[]	[coordination units]
Circle Type	SINT, INT, or DINT	immediate or tag	0 = Via 1 = Center 2 = Radius 3 = Center Incremental
Via/Center/ Radius	REAL	array tag[] (via/center) Immediate or tag (radius)	[coordination units]
Direction	SINT, INT, or DINT	immediate or tag	**2D** **3D** 0 = CW Shortest 1 = CCW Longest 2 = CW Full Shortest Full 3 = CCW Full Longest Full

Operand	Type	Format	Description
Speed	SINT, INT, DINT, or REAL	immediate or tag	[coordination units]
Speed Units	SINT, INT, or DINT	immediate	0 = Units per Sec 1 = % of Maximum
Accel Rate	SINT, INT, DINT, or REAL	immediate or tag	[coordination units]
Accel Units	SINT, INT, or DINT	immediate	0 = Units per Sec^2 1 = % of Maximum
Decel Rate	SINT, INT, DINT, or REAL	immediate or tag	[coordination units]
Decel Units	SINT, INT, or DINT	immediate	0 = Units per Sec^2 1 = % of Maximum
Profile	SINT, INT, or DINT	immediate	0 = Trapezoidal 1 = S-Curve
Termination Type	SINT, INT, or DINT	immediate or tag	0 = Actual Tolerance 1 = No Settle 2 = Command Tolerance 3 = No Decel 4 = Follow Contour Velocity Constrained 5 = Follow Contour Velocity Unconstrained See Choose a termination type on page 259.
Merge	SINT, INT, or DINT	immediate	0 = Disabled 1 = Coordinated Motion 2 = All Motion
Merge Speed	SINT, INT, or DINT	immediate	0 = Programmed 1 = Current

```
MCCM(CoordinateSystem,
MotionControl,MoveType,
Position,Speed,Speedunits,
AccelRate,AccelUnits,
DecelRate,DecelUnits,
VelocityProfile,
TerminationType,Merge,
MergeSpeed);
```

Structured Text

The operands are the same as those for the relay ladder MCCM instruction.

When entering enumerations for the operand value in Structured Text, multiple word enumerations must be entered without spaces. For example: when entering Decel Units the value should be entered as unitspersec^2 rather than Units per Sec^2 as displayed in the ladder logic.

For the operands that have enumerated options, enter your selection as:

This operand	Has these options which you...		
	enter as text		or enter as a number
MoveType	no enumeration		0 (Absolute) 1 (Incremental)
Circle Type	no enumeration		0 (Via) 1 (Center) 2 (Radius) 3 (Center Incremental)
Direction	**2D**	**3D**	
	clockwise	shortest	0
	counterclockwise	longest	1
	clockwisefull	shortestfull	2
	counterclockwisefull	longestfull	3
Speed Units	unitspersec %ofmaximum		0 1
Accel Units	unitspersec^2 %ofmaximum		0 1
Decel Units	unitspersec^2 %ofmaximum		0 1
Profile	trapezoidal s-curve		0 1

This operand	Has these options which you...	
	enter as text	or enter as a number
Termination Type	no enumeration	0 (Actual Tolerance) 1 (No Settle) 2 (Command Tolerance) 3 (No Decel) 4 (Follow Contour Velocity Constrained) 5 (Follow Contour Velocity Unconstrained) See Choose a termination type on page 259.
Merge	Disabled Coordinatedmotion Allmotion	0 1 2
Merge Speed	Programmed Current	0 1

Description: The Motion Coordinated Circular Move (MCCM) performs a circular move using up to three (3) axes statically coupled to the coordinate system as primary axes in a Cartesian coordinate system. The circular move is specified as either absolute or incremental and done at a desired speed. The actual speed of the MCCM is a function of the mode of the move (commanded speed or percent of maximum speed). The speed of the move is based on the time it takes to complete the circular move using the programmed axes. Each axis is commanded to move at a speed that allows for all axes to reach the endpoint (target position) at the same time.

ATTENTION

If You Use An S-curve Profile

Be careful if you change the speed, acceleration, deceleration, or jerk while an axis is accelerating or decelerating along an S-curve profile. You can cause an axis to **overshoot its speed or reverse direction**.

For more information, see Troubleshoot Axis Motion on page 367.

Coordinate System

The Coordinate System operand specifies the system of motion axes that define the dimensions of a Cartesian coordinate system. For this release the coordinate system supports up to three (3) primary axes.

Only the axes configured as primary axes (up to 3) are included in speed calculations. Only primary axes participate in the actual circular move.

Motion Control

The following control bits are affected by the MCCM instruction.

Mnemonic	Description
.EN (Enable) Bit 31	The Enable Bit is set when the rung transitions from false to true. It resets the rung transitions from true to false.
.DN (Done) Bit 29	The Done Bit sets when the coordinated instruction has been verified and queued successfully. Because it is set at the time it is queued it may appear as set when a runtime error is encountered during the verify operation after it comes out of the queue. It resets when the rung transitions from false to true.
.ER (Error) Bit 28	The Error Bit resets when the rung transitions from false to true. It sets when the coordinated move fails to initiate successfully. It can also be set with the Done bit when a queued instruction encounters a runtime error.
.IP (In Process) Bit 26	The In Process Bit sets when the coordinated move is successfully initiated. It resets when there is a succeeding move and the coordinated move reaches the new position, or when there is no succeeding move and the coordinated move reaches the termination type specifications, or when the coordinated move is superseded by another MCCM or MCLM instruction with a Merge Type of Coordinated Move or when terminated by an MCS or an MCSD instruction.
.AC (Active) Bit 23	When you have a coordinated move instruction queued, the Active Bit lets you know which instruction is controlling the motion. It sets when the coordinated move becomes active. It is reset when the Process Complete bit is set or when the instruction is stopped.
.PC (Process Complete) Bit 27	The Process Complete Bit resets when the rung transitions from false to true. It sets when there is no succeeding move and the coordinated move reaches the new position, or when there is a succeeding move and the coordinated move reaches the Termination Type specification.
.ACCEL (Acceleration) Bit 01	The Acceleration Bit sets while the coordinated move is in acceleration phase. It resets while the coordinated move is in the constant velocity or deceleration phase, or when coordinated motion concludes.
.DECEL (Deceleration) Bit 02	The Deceleration Bit sets while the coordinated move is in deceleration phase. It resets while the coordinated move is in the constant velocity or acceleration phase, or when coordinated motion concludes.

Move Type

The Move Type operand determines the method used by the position array to indicate the path of the coordinated move and the method the via/center/radius parameter uses to indicate the via and center circle positions. The options are: Absolute or Incremental.

Absolute

When the Move Type is Absolute, the coordinate system moves to the specified Position at the defined Speed, using the Accel and Decel Rates as designated by their respective operands, along a circular path.

When an axis is configured for rotary operation, absolute moves are handled in the same manner as with linear axes. When the axis position exceeds the Unwind parameter, an error is generated.

The sign of the specified position array is interpreted by the controller as the direction for the move. Negative position values instruct the interpolator to move the rotary axis in a negative direction to obtain the desired absolute position. A positive value indicates that positive motion is desired to reach the target position. To move to the unwind position in the negative direction a negative unwind position value must be used as 0 and -0 are treated as 0. When the position is greater than the unwind value, an error is generated. The axis can move through the unwind position but never incrementally more than one unwind value.

Incremental

When the Move Type is Incremental, the coordinate system moves the distance as defined by the position array at the specified Speed, using the Accel and Decel rates determined by the respective operands, along a circular path.

The specified distance is interpreted by the interpolator and can be positive or negative. Negative position values instruct the interpolator to move the rotary axis in a negative direction, while positive values indicate positive motion is desired to reach the target position.

Position

The Position operand is a one dimensional array whose dimension is at least equivalent to the number of axes specified in the coordinate system. It is the position array that defines the new absolute or incremental position.

Circle Type

The Circle Type operand specifies how the array labeled via/center/radius is interpreted. The options are: Via, Circle, Radius, Center Incremental.

Via

Via indicates that the via/center/radius array members specify a via point between the start and end points.

Center

Center indicates that the via/center/radius array members contain the circle center.

Radius

Radius indicates that the first via/center/radius array member contains the radius. Other members are ignored.

Center Incremental

Center Incremental indicates that the via/center/radius array members define a position that always incrementally defines the center of the circle regardless of Move Type operand. Sign of the incremental value is measured from the start point to the center.

Two Dimensional Arc Examples

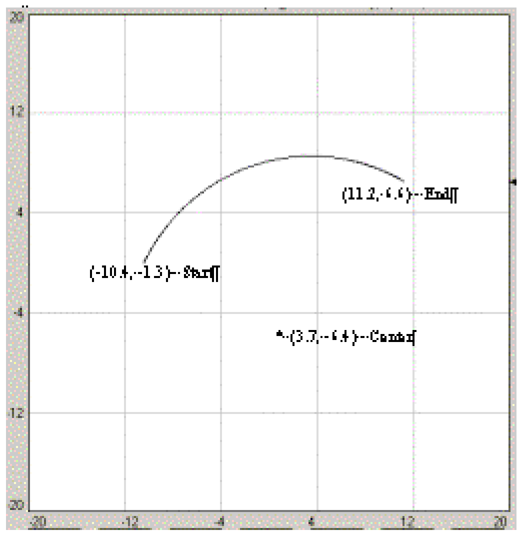
The following examples show the use of Absolute and Incremental Move Types with the various Circle Types.

MCCM Using Center Circle Type

The following examples show the use of the MCCM with a Circle Type of Center and a Move Type of Absolute (first example) and Incremental (second example) to arrive at the same result. The basic assumptions are:

- The 2 axes, Axis0 and Axis1, are both members of the coordinate system, Coordinated_sys.
- Axis0 and Axis1 are orthogonal to each other.
- Coordinated_sys is initially at (-10.4,-1.3) units.

Move Coordinated_sys along an arc to (11.2,6.6) units with a center of (3.7,-6.4) units at the vector speed of 10.0 units per second with the acceleration and deceleration values of 5.0 units per second^2. The following graph shows the path generated by the preceding information.

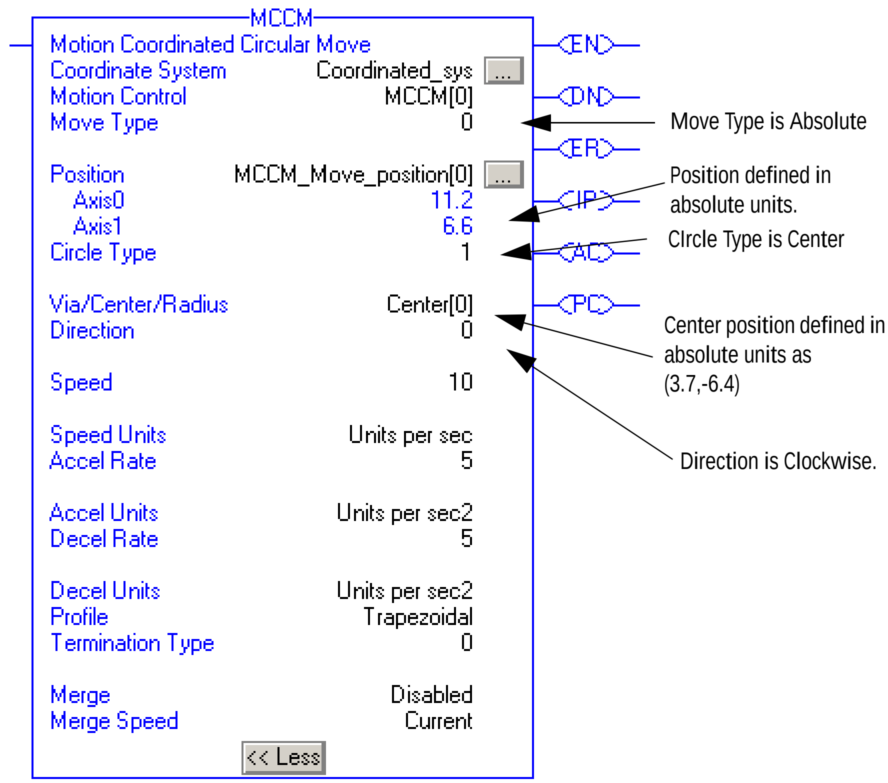


Plot of MCCM Instruction with Circle Type of Center

The vector speed of the selected axes is equal to the specified speed in the units per second or percent of the maximum speed of the coordinate system. Likewise the vector acceleration and deceleration is equal to the specified acceleration/deceleration in the units per second^2 or percent of maximum acceleration of the coordinate system.

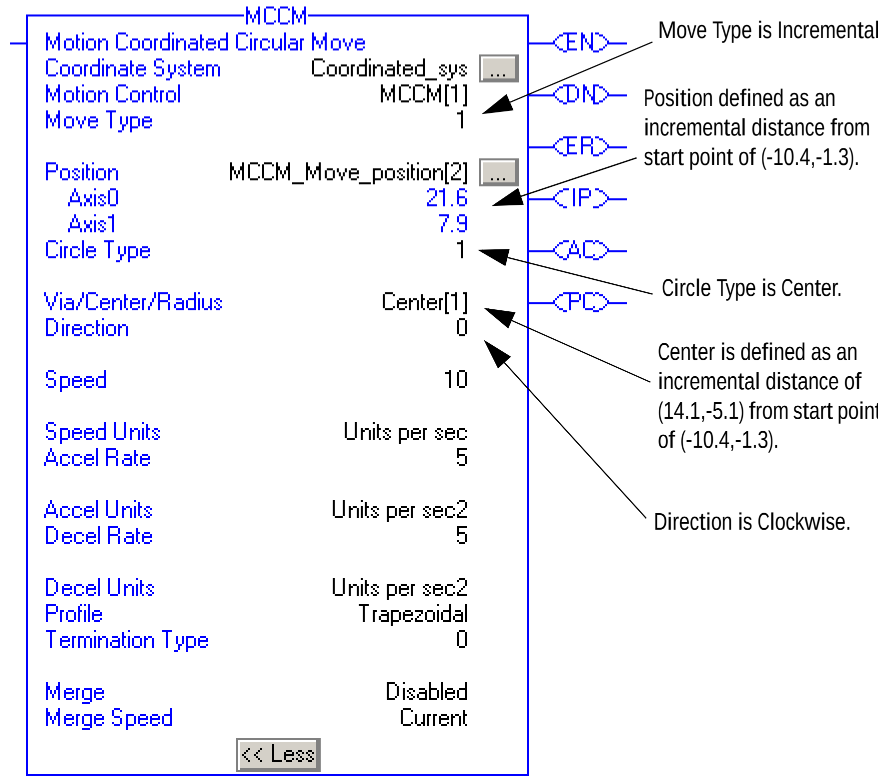
This path can be achieved by using an MCCM instruction in the Clockwise direction with a Move Type = Absolute or with a Move Type = Incremental. When a Circle Type of Center is chosen, the Via/Center/Radius position defines the center of the arc.

MCCM Instruction Move Type Absolute



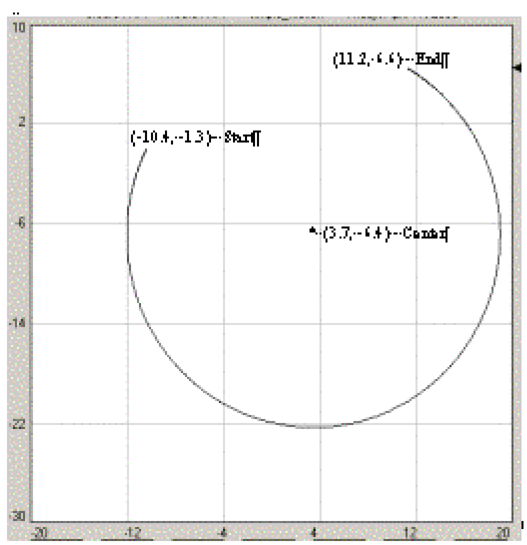
MCCM Ladder Instruction with Move Type of Absolute

MCCM Instruction Move Type Incremental



MCCM Ladder Instruction with Move Type of Incremental

Had a Direction of Counterclockwise been selected (Direction = 1), the axes move along the curve shown in the following graph.



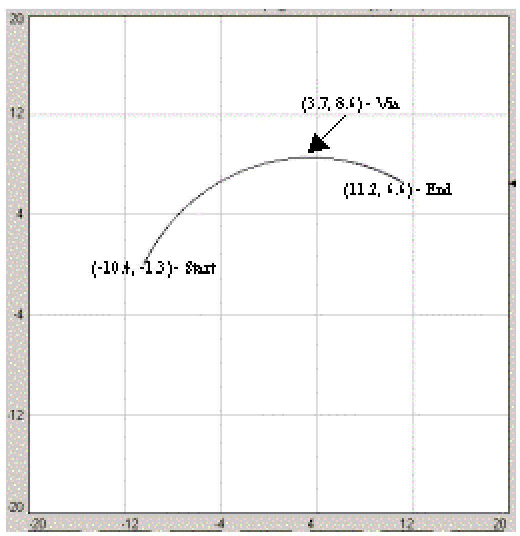
Plot of Path with Direction as Counterclockwise

MCCM Using Via Circle Type

The following examples show the use of the MCCM with a Circle Type of Via and a Move Type of Absolute (first example) and Incremental (second example) to arrive at the same result. The basic assumptions are:

- The 2 axes, Axis0 and Axis1, are both members of the coordinate system, Coordinated_sys.
- Axis0 and Axis1 are orthogonal to each other.
- Coordinated_sys is initially at (-10.4,-1.3) units.

Move Coordinated_sys along an arc to (11.2,6.6) units passing through the point (3.7,8.6) units at the vector speed of 10.0 units per second with the acceleration and deceleration values of 5.0 units per second^2. The following graph shows the path generated by the preceding information.

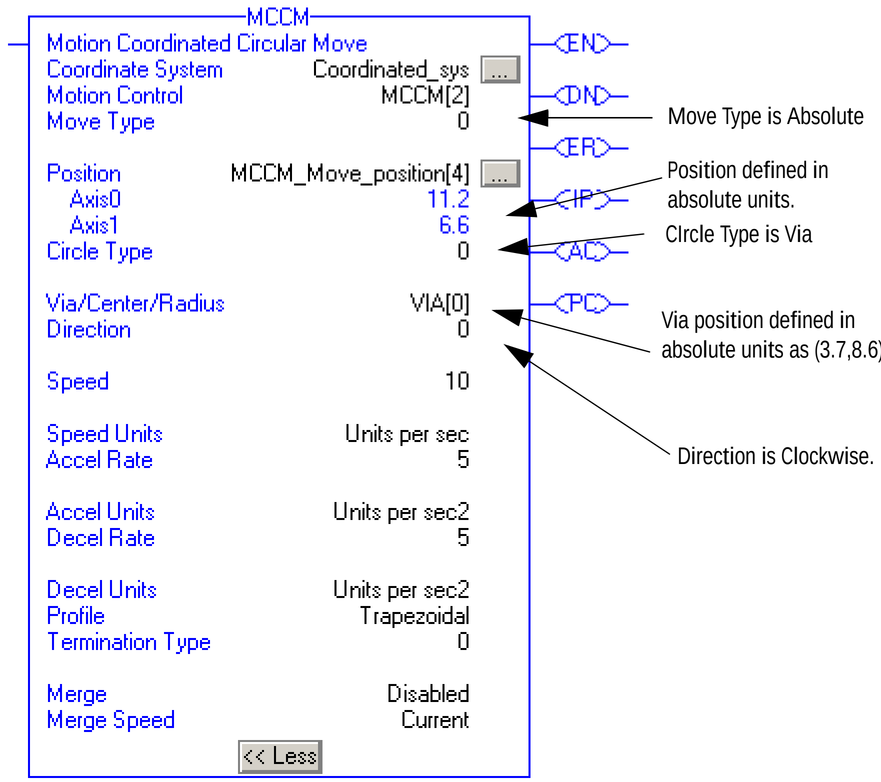


Plot of Path of MCCM with Operands of Via and Absolute

The vector speed of the selected axes is equal to the specified speed in the units per second or percent of the maximum speed of the coordinate system. Likewise the vector acceleration and deceleration is equal to the specified acceleration/deceleration in the units per second^2 or percent of maximum acceleration of the coordinate system.

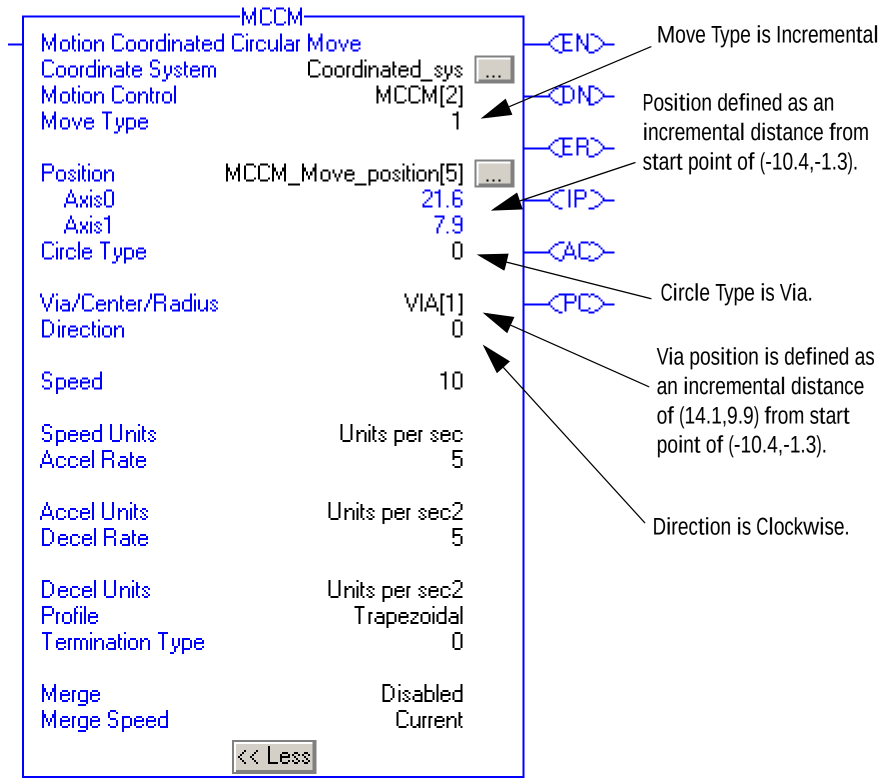
This path can be achieved by using an MCCM instruction in the Clockwise direction with a Move Type = Absolute or with a Move Type = Incremental. When a Circle Type of Via is chosen, the Via/Center/Radius position defines a point through which the arc must pass.

MCCM Instruction Move Type Absolute; Circle Type Via



MCCM Ladder Instruction with Operand Values of Via and Absolute

MCCM Instruction Move Type Incremental



MCCM Ladder Instruction with Operand Values of Via and Incremental

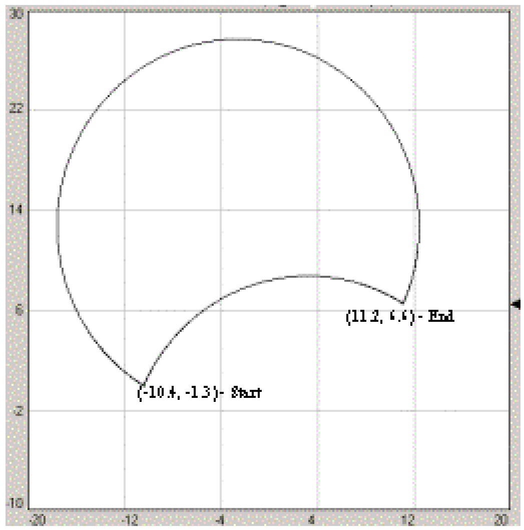
Since there are three points (the current position of the axes, the specified end point, and the specified via point) it is difficult to program a bad arc. While it is still certainly possible to program an arc which is not the intended one, a Circular Programming Error runtime fault occurs with an arc if the three points are co-linear (all three on the same line) or not unique (two or more points are the same). In addition, the via point implies that the direction of the arc and thus it is not necessary (and is ignored) to specify the direction.

MCCM Using Radius Circle Type

The following examples show the use of the MCCM with a Circle Type of Radius and a Move Type of Absolute (first example) and Incremental (second example) to arrive at the same result. The basic assumptions are:

- The 2 axes, Axis0 and Axis1, are both members of the coordinate system, Coordinated_sys.
- Axis0 and Axis1 are orthogonal to each other.
- Coordinated_sys is initially at (-10.4,-1.3) units.

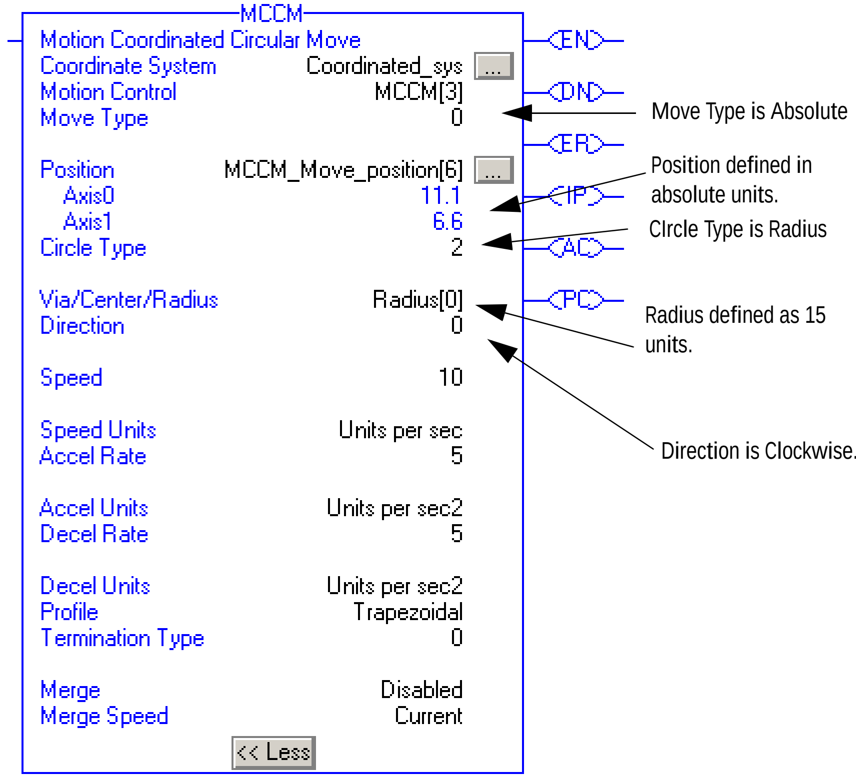
Move Coordinated_sys along an arc to (11.2,6.6) units with a radius of 15 units at the vector speed of 10.0 units per second with the acceleration and deceleration values of 5.0 units per second^2. The following graph shows the paths generated by the preceding information using a Radius value of 15 units and -15 units.



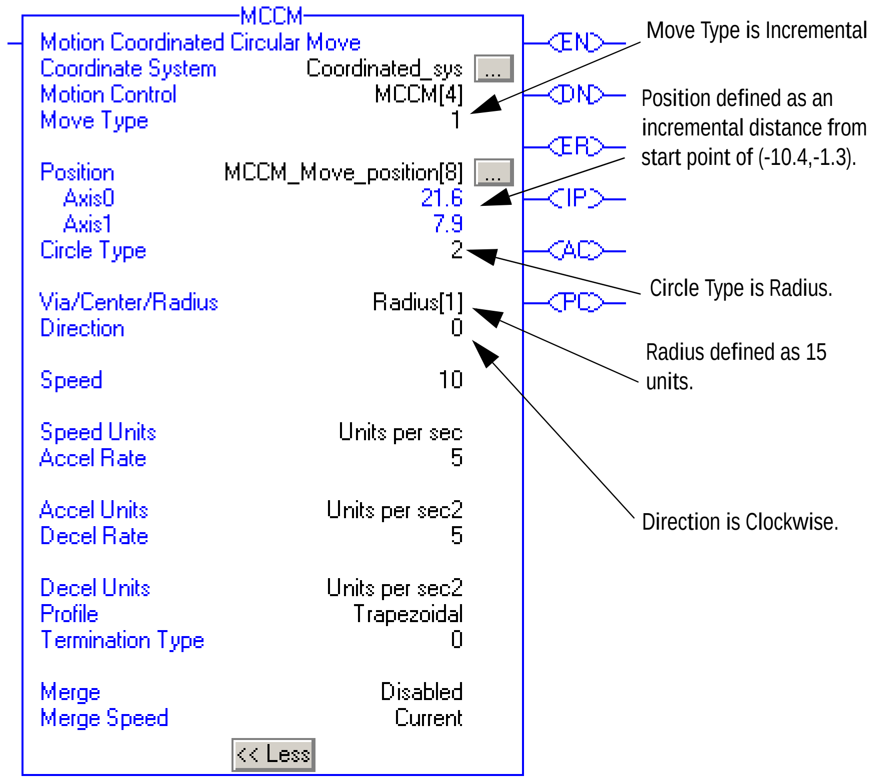
Plot of Path with Circle Type of Radius

This path can be achieved by using an MCCM instruction in the Clockwise direction with a Move Type = Absolute or with a Move

Type = Incremental. When a Circle Type of Radius is chosen, the Via/Center/Radius position defines the radius of the arc.



MCCM Instruction Move Type Absolute; Circle Type Radius



MCCM Instruction Move Type Incremental; Circle Type Radius

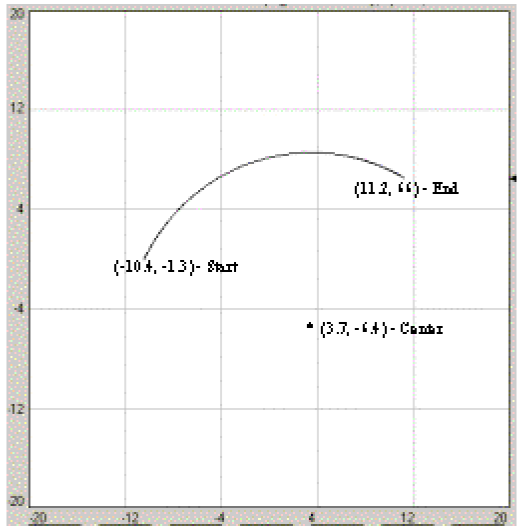
The Move Type has no effect on the Radius value specification. A Positive radius always creates a shorter (<180°) arc and a negative radius creates a longer (>180°) arc (see path graph). If it is 180°, the sign of the radius is irrelevant.A Circle Type of Radius is valid in two-dimensional coordinate systems only.

MCCM Using Center Incremental Circle Type

The following examples show the use of the MCCM with a Circle Type of Center Incremental and a Move Type of Absolute (first example) and Incremental (second example) to arrive at the same result. The basic assumptions are:

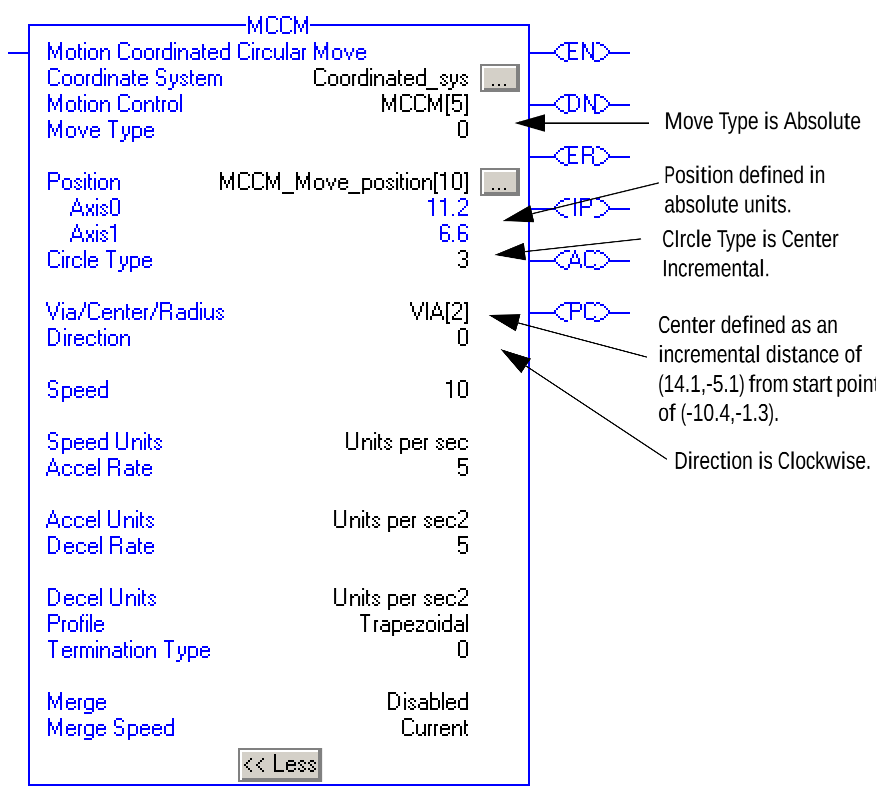
- The 2 axes, Axis0 and Axis1, are both members of the coordinate system, Coordinated_sys.
- Axis0 and Axis1 are orthogonal to each other.
- Coordinated_sys is initially at (-10.4,-1.3) units.

Move Coordinated_sys along an arc to (11.2,6.6) units with a center at an increment of (14.1,-5.1) units from the start point at the vector speed of 10.0 units per second with the acceleration and deceleration values of 5.0 units per second^2. The following graph shows the path generated by the preceding information.



Plot of Path with Circle Type of Center Incremental

This path can be achieved by using an MCCM instruction in the Clockwise direction with a Move Type = Absolute or with a Move Type = Incremental. When a Circle Type of Center Incremental is chosen, the Via/Center/Radius position defines the center of the arc.



MCCM Instruction Move Type Absolute; Circle Type Center Incremental

The MCCM Instruction with Move Type of Incremental and Center Type of Center Incremental is the same as an MCCM instruction with Move Type Incremental and Circle Type of Center.

Two Dimensional Full Circle Example

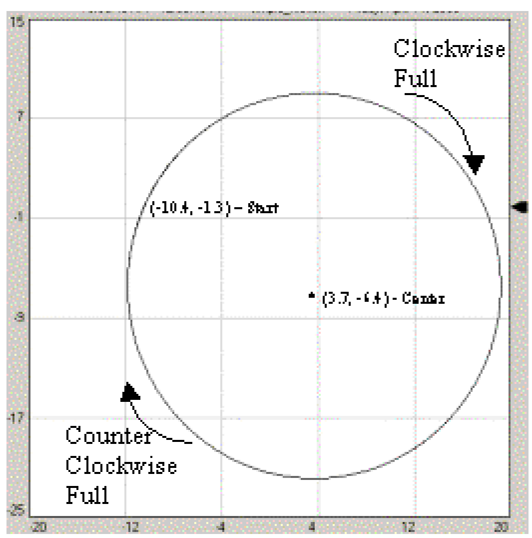
Creating a full circle is a special case of a circular arc. The following is an example of a two dimensional full circle.

MCCM Full Circle

The following examples show the use of the MCCM with a Circle Type of Center and a Move Type of Absolute (first example) and Incremental (second example) to create a full circle. The basic assumptions are:

- The 2 axes, Axis0 and Axis1, are both members of the coordinate system, Coordinated_sys.
- Axis0 and Axis1 are orthogonal to each other.
- Coordinated_sys is initially at (-10.4,-1.3) units.

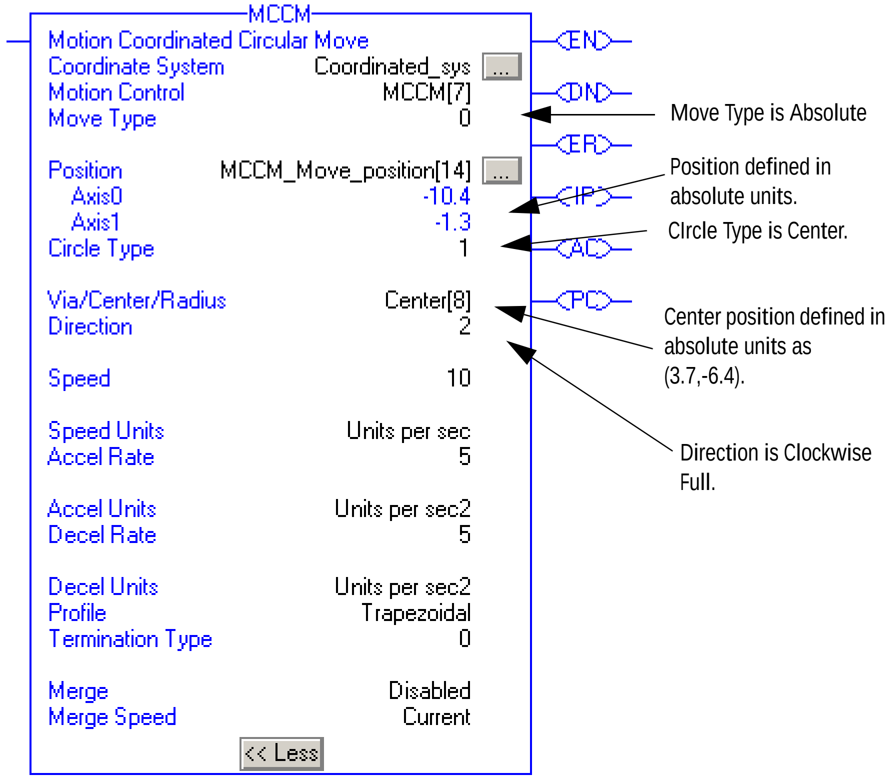
Move Coordinated_sys along an arc to (-10.4,-1.3) units with a center at (3.7,-6.4) units from the start point at the vector speed of 10.0 units per second with the acceleration and deceleration values of 5.0 units per second^2. The following graph shows the circle generated by the preceding information.



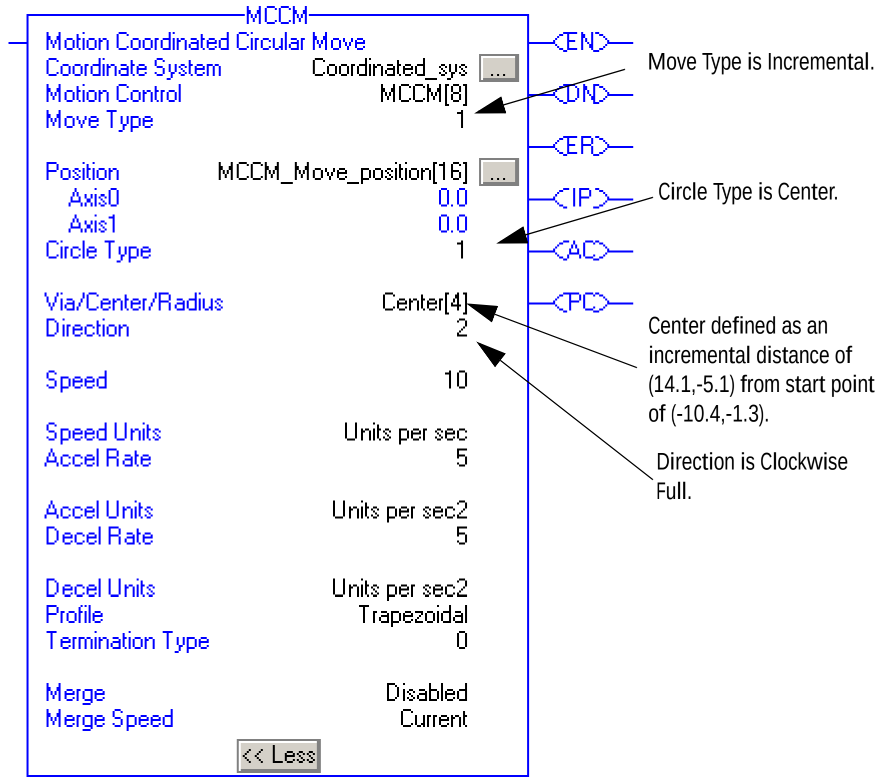
Plot of Path of MCCM Full CIrcle

This path can be achieved by using an MCCM instruction in the Clockwise direction with a Move Type = Absolute or with a Move

Type = Incremental. When a Circle Type of Center is chosen, the Via/Center/Radius position defines the center of the arc.



MCCM Instruction Move Type Absolute; Circle Type Center.



MCCM with Move Type as Incremental and Center Type as Center.

To draw a full circle using Radius as the Circle Type:

- Start point must not equal the end point.
- Direction must be either Clockwise Full or Counter Clockwise Full.
- Sign of Radius is irrelevant.

MCCM with Rotary Axes Examples

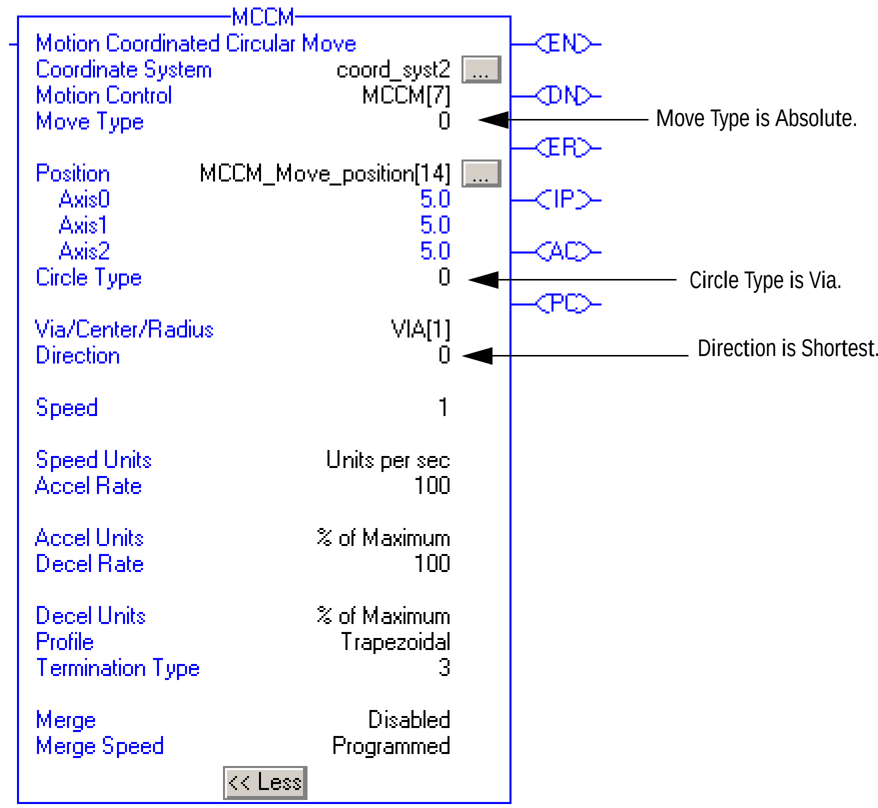
The following examples show the use of the MCCM instruction with Rotary axes and Move Types of Absolute and Incremental.

MCCM with Three Axes, One Rotary Axis, and Move Type of Absolute

The first example uses a coordinate system of three axes with one Rotary axis and a Move type of Absolute. The plot of the path is based on the following assumptions:

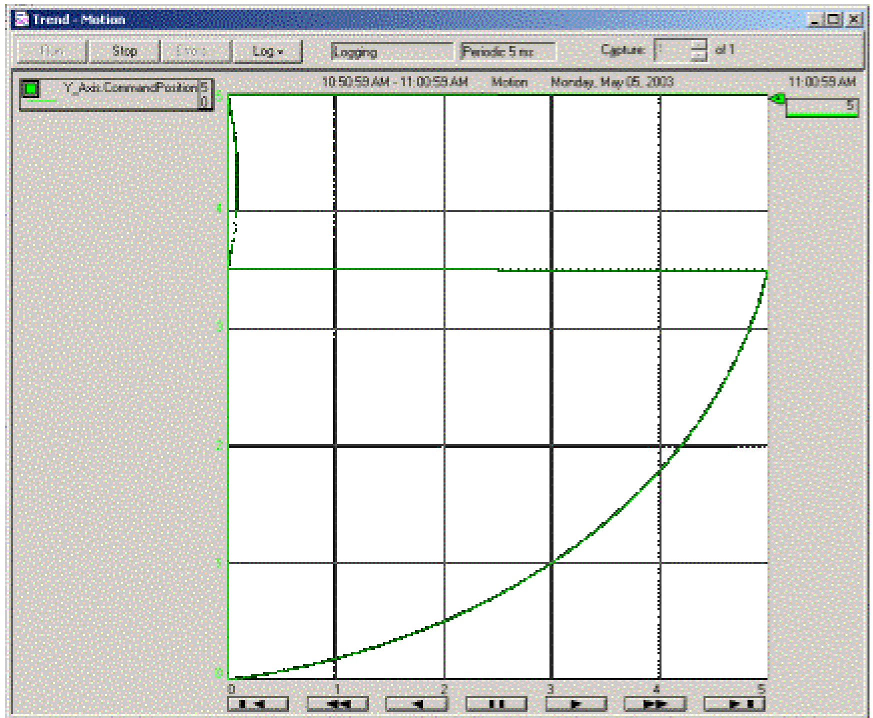
- 3 axis Coordinate System named coord_syst2 (Axis2, the Z axis, is ignored in plots to reduce the confusion and to better illustrate the actions of the rotary axis (Axis0).
- Axis0 is Rotary with an unwind of 5 revs.
- Start position is 0,0,0.

- End position is 5,5,5.
- Via position is 5,3.5,3.5



MCCM Ladder Instruction with Move Type of Absolute

The preceding MCCM instruction produces the following plot.



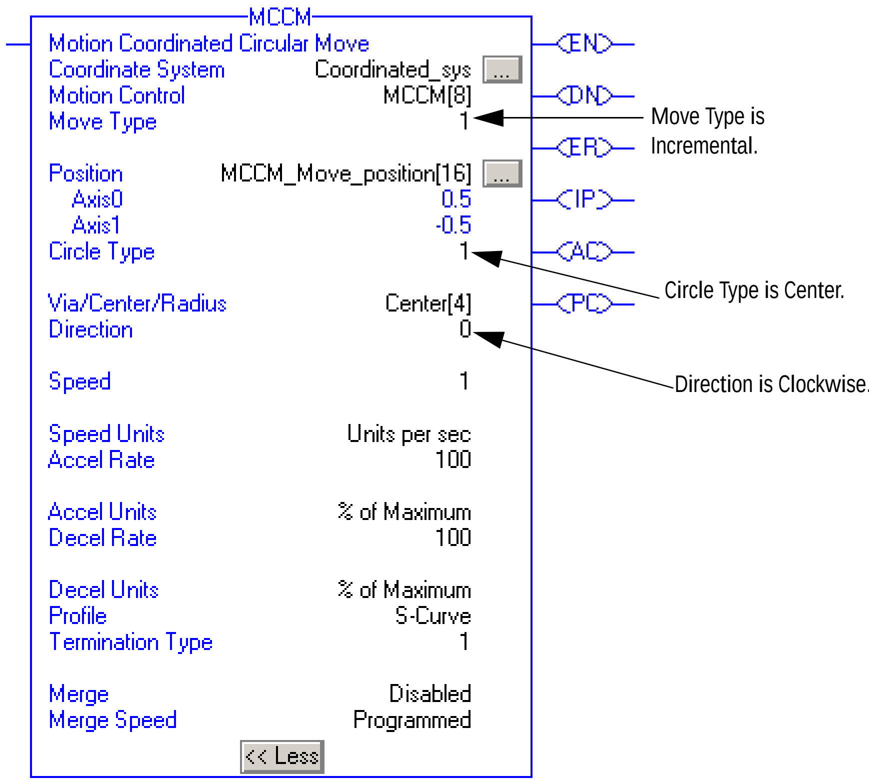
Plot of MCCM with Three Axes, One Rotary Axis & Move Type of Absolute

The axis actually travels counter clockwise in an arc from (0,0,0) to (5,5,5) via the (5,3.5,3.5) position. The Direction was specified as clockwise but with Via specified for the Circle Type the Direction operand is ignored. The move stops after generating a 90 degree arc. There was one travel through the unwind for Axis0 even though it was in Move Type of Absolute. It should be noted that the path of the coordinated motion is determined in linear space but the position of the axes is limited by the rotary configuration. The End and Via points are required to fit within the absolute position defined by the rotary unwind of Axis0. However, the resulting motion from these choices can travel through the unwind of the rotary axis.

MCCM with Two Rotary Axis and Move Type of Incremental

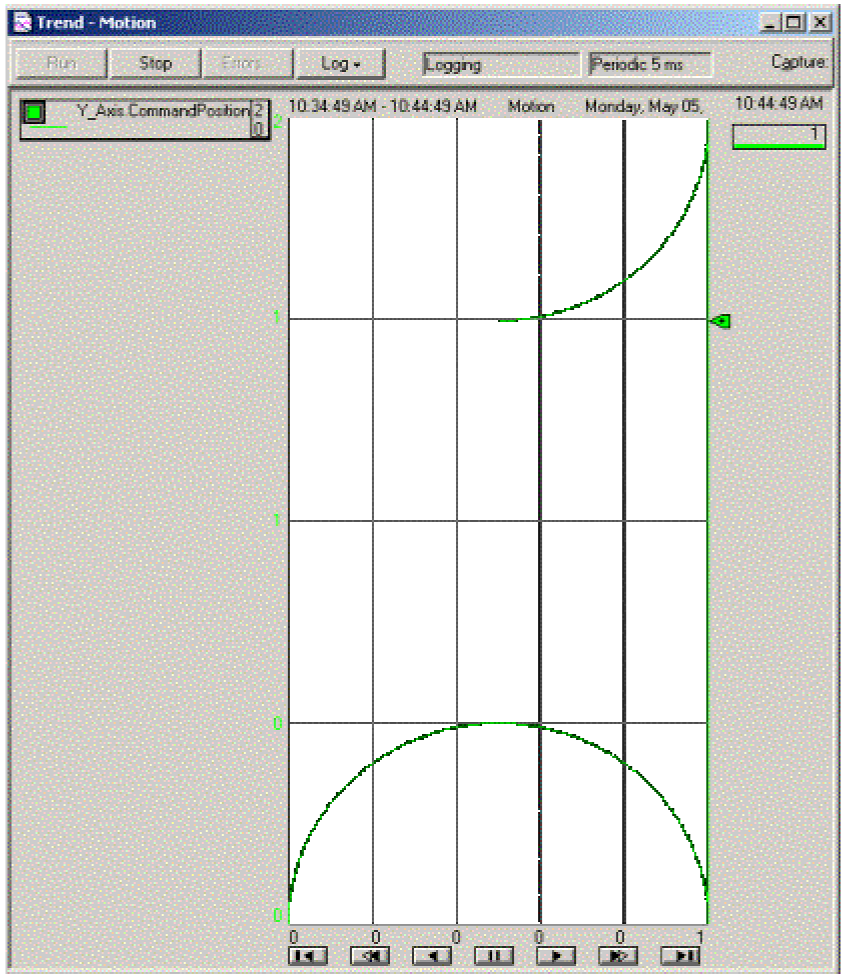
The second example uses a coordinate system of two Rotary axes and a Move type of Incremental. The plot of the path is based on the following assumptions:

- 2 axis Coordinate System named Coordinated_sys.
- Axis0 is Rotary with an unwind of 1 rev.
- Axis1 is Rotary with an unwind of 2 revs.
- Start position is 0,0.
- Increment to end position is 0.5,-0.5.
- Increment to Center position is 0.5,0.



MCCM Ladder Instruction with Move Type of Absolute

The preceding MCCM instruction produces the following plot.



Plot of MCCM with Two Rotary Axes and Move Type of Incremental

The axis travels clockwise in a circle from (0,0) to (0.5,1.5). The move stops after generating a 270 degree arc. There was one travel through the unwind for Axis1. It should be noted that the path of the coordinated motion is determined in linear space but the position of the axes is limited by the rotary configuration. The endpoint was (0.5,-0.5) for the circle calculations but the actual endpoint for the move was (0.5,1.5). The instruction specified and we obtained a clockwise move even though one axis had a negative incremental target position. The endpoint is not required to fit within the absolute position defined by the rotary unwind of the axes.

Three Dimensional Arcs

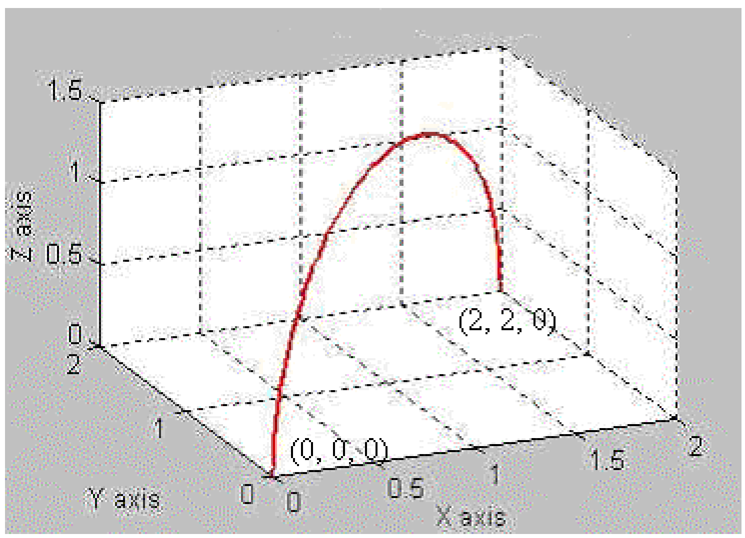
For Coordinate Systems that have three primary axes associated to them, it is possible to create three dimensional arcs.

3D Arc Using MCCM with Circle Type Via

The following example shows the use of the MCCM with a Circle Type of Via and a Move Type of Absolute to create a three dimensional arc. The basic assumptions are:

- The 3 axes, Axis0 and Axis1, Axis2 are all members of the coordinate system, Coordinated_sys1.
- Coordinated_sys1 is a three dimensional coordinate system.
- Axis0, Axis1, and Axis2 are orthogonal to each other.
- Coordinated_sys1 is initially at (0.0, 0.0, 0.0) units.

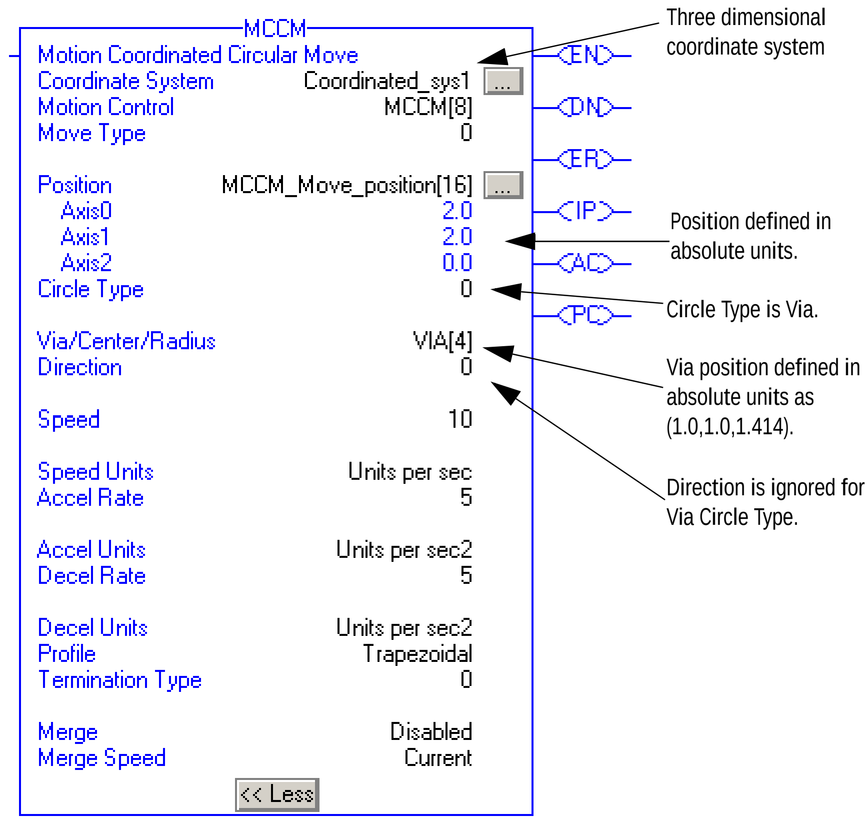
Move Coordinated_sys1 along an arc to (2.0, 2.0, 0.0) units passing through (1.0, 1.0, 1.414) units at the vector speed of 10.0 units per second with the acceleration and deceleration values of 5.0 units per second^2. The following graph shows the 3D arc generated by the preceding information.



3D Arc Using Circle Type of Via

This path is achieved by using an MCCM instruction with a Move Type of Absolute and a Circle Type of Via. When Via is selected, the

Via/Center/Radius position defines a point through which the arc must pass.



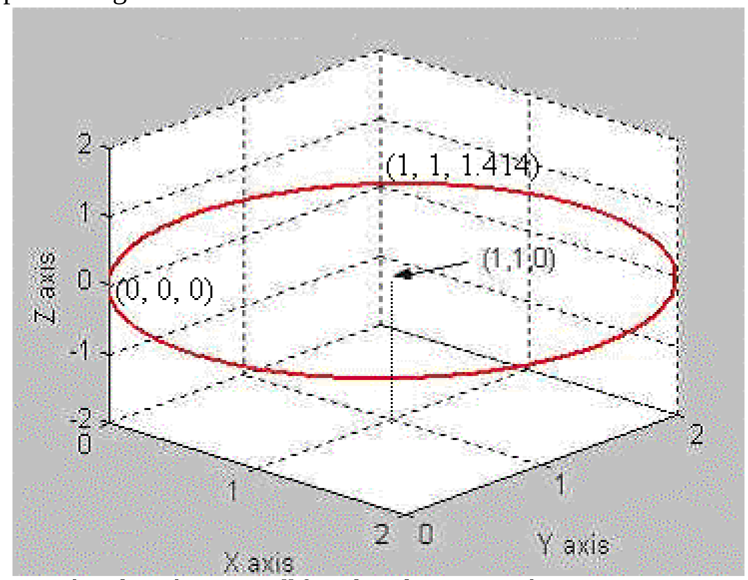
MCCM Ladder Instruction for 3D Arc Using Circle Type of Via

3D Arc Using MCCM with Circle Type Center

The following example shows the use of the MCCM with a Circle Type of Center and a Move Type of Absolute to create a three dimensional arc. The basic assumptions are:

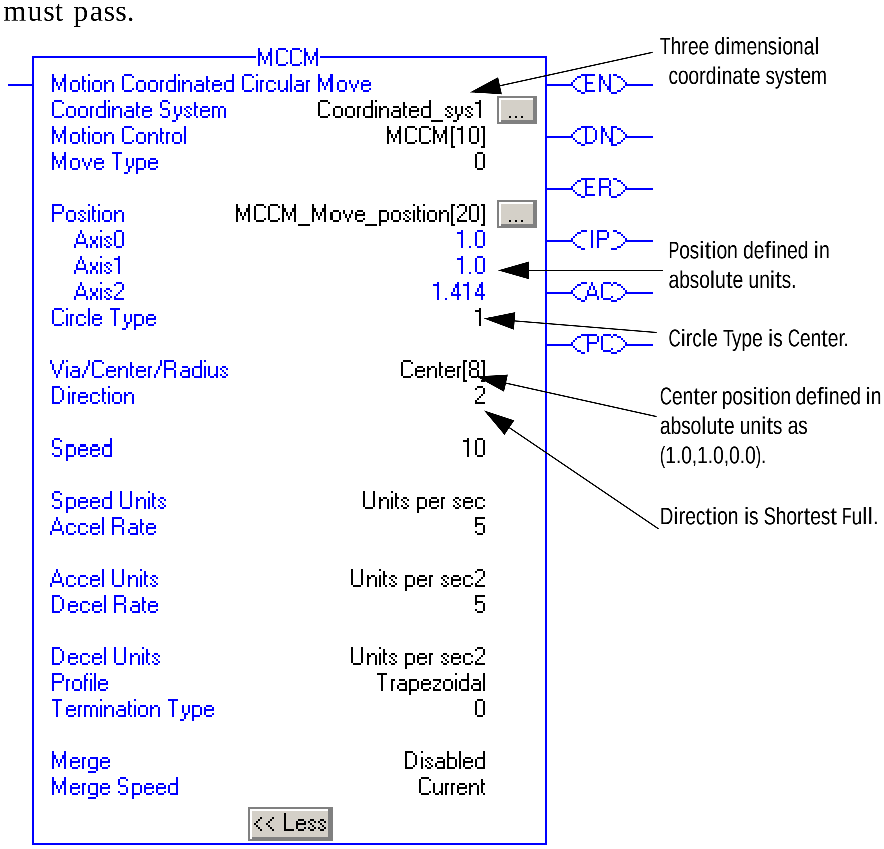
- The 3 axes, Axis0 and Axis1, Axis2 are all members of the coordinate system, Coordinated_sys1.
- Coordinated_sys1 is a three dimensional coordinate system.
- Axis0, Axis1, and Axis2 are orthogonal to each other.
- Coordinated_sys1 is initially set at (0.0, 0.0, 0.0) units.

Move Coordinated_sys1 along an arc to (1.0, 1.0, 1.414 units with center at (1.0, 1.0, 1.0) units at the vector speed of 10.0 units per second with the acceleration and deceleration values of 5.0 units per second². The following graph shows the 3D arc generated by the preceding information.



3D Path Using Shortest Full for Direction Operand

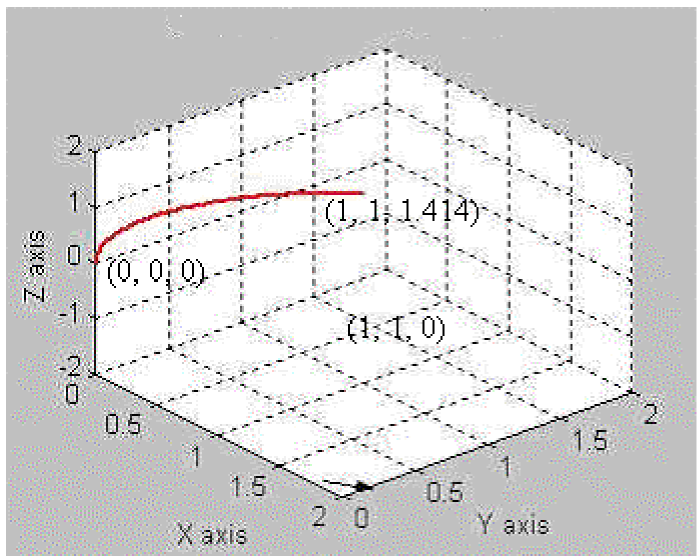
This path is achieved by using an MCCM instruction with a Move Type of Absolute and a Circle Type of Center. When Via is selected, the Via/Center/Radius position defines a point through which the arc must pass.



MCCM Ladder Instruction for 3D Arc Using Circle Type of Center

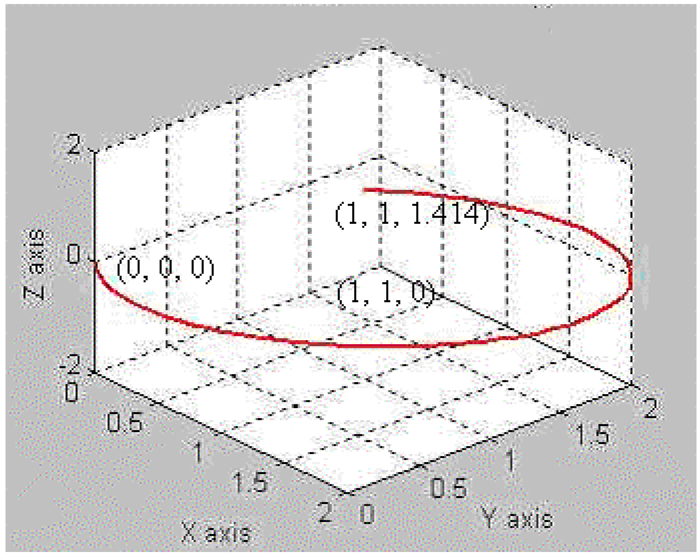
For full circles set Position operand to any point except the start point and use one of the "Full" Direction types. The endpoint is assumed to be the start point. This is because in the three dimensional space you need three points to specify a plane for the circle.

By changing the Direction operand to Shortest in the preceding MCCM instruction, the following path is generated. The Shortest option of the Direction operand takes the shortest route from the start point to the point defined by the Position operand of the MCCM instruction.



3D Path Using Shortest for Direction Operand

Change the Direction operand to Longest in the preceding MCCM instruction and the path followed is the longest from the start point to the point defined by the Position operand in the MCCM instruction. See the following diagram for an example of the longest path.



3D Path Using Longest for Direction Operand

Via/Center/Radius

Depending on the selected Move Type and Circle Type, the via/center/radius position parameter defines the absolute or incremental value of a position along the circle, the center of the circle, or the radius of the circle as defined in the following table. If the Circle Type is via or center, the via/center/radius position parameter is a one-dimensional array, whose dimension is defined to be at least the equivalent of the number of axes specified in the coordinate system. If the Circle Type is radius, the via/center/radius position parameter is a single value.

Move Type	Circle Type	Behavior
Absolute	Via	The via/center/radius position array defines a position along the circle. For a non-full circle case, the Position parameter array defines the endpoint of the arc. For a full circle case, the Position parameter array defines any second point along the circle except the endpoint.
Incremental	Via	The sum of the via/center/radius position array and the old position defines the position along the circle. For a non-full circle case, the sum of the Position parameter array and the old position defines the endpoint of the arc. For a full circle case, the sum of the Position parameter array and the old position defines any second point along the circle except the endpoint.
Absolute	Center	The via/center/radius position array defines the center of the circle. For a non-full circle case, the Position parameter array defines the endpoint of the arc. For a full circle case, the Position parameter array defines any second point along the circle except the endpoint.
Incremental	Center	The sum of the via/center/radius position array and the old position defines the center of the circle. For a non-full circle case, the sum of the Position parameter array and the old position defines the endpoint of the arc. For a full circle case, the sum of the Position parameter array and the old position defines any second point along the circle except the endpoint.
Absolute or Incremental	Radius	The via/center/radius position single value defines the arc radius. The sign of the value is used to determine the center point to distinguish between the two possible arcs. A positive value indicates a center point that generates an arc less than 180 degrees. A negative value indicates a center point that generates an arc greater than 180 degrees. This Circle Type is only valid for two-dimensional circles. The position parameter array follows the Move Type to define the endpoint of the arc.
Absolute	Center Incremental	The sum of the via/center/radius position array and the old position defines the center position of the circle. For a non-full circle case, the Position parameter array defines the endpoint of the arc. For a full circle case, the Position parameter array defines any second point along the circle except the endpoint.
Incremental	Center Incremental	The sum of the via/center/radius position array and the old position defines the center position of the circle. For a non-full circle case, the sum of the Position parameter array and the old position defines the endpoint of the arc. For a full circle case, the sum of the Position parameter array and the old position defines any second point along the circle except the endpoint.

Direction

The Direction operand defines the rotational direction of a 2D circular move as either clockwise or counterclockwise according to the right-hand screw rule. For a 3D circular move the direction is either Shortest or Longest. In both 2D and 3D it can also indicate if the circular move is to be a full circle.

Speed

The Speed operand defines the maximum vector speed along the path of the coordinated move.

Speed Units

The Speed Units operand defines the units applied to the Speed operand either directly in coordination units or as a percentage of the maximum values defined in the coordinate system.

Accel Rate

The Accel Rate operand defines the maximum acceleration along the path of the coordinated move.

Accel Units

The Accel Units operand defines the units applied to the Accel Rate operand either directly in coordination units of the specified coordinate system or as a percentage of the maximum values defined in the coordinate system.

Decel Rate

The Decel Rate operand defines the maximum deceleration along the path of the coordinated move.

Decel Units

The Decel Units operand defines the units applied to the Decel Rate operand either directly in coordination units of the specified coordinate system or as a percentage of the maximum values defined in the coordinate system.

Profile

The Profile operand determines whether the coordinated move uses a trapezoidal or an S-Curve velocity profile. See the Profile section of the MCLM instruction earlier in this chapter for more information about Trapezoidal and S-Curve profiles.

Merge

The merge defines whether or not to turn the motion of all specified axes into a pure coordinated move. The options are: Merge Disabled, Coordinated Motion, or All Motion.

Merge Disabled

Any currently executing single axis motion instructions involving any axes defined in the specified coordinate system are not affected by the activation of this instruction, and result in superimposed motion on the affected axes. An error is flagged if a second instruction is initiated in the same coordinate system or in another coordinate system containing any axes in common with the coordinate system that is active.

Coordinated Motion

Any currently executing coordinated motion instructions involving the same specified coordinate system are terminated, and the active motion is blended into the current move at the speed defined in the merge speed parameter. Any pending coordinated motion instructions in the specified coordinate system are cancelled. Any currently executing system single axis motion instructions involving any axes defined in the specified coordinate system are not affected by the activation of this instruction, and result in superimposed motion on the affected axes.

All Motion

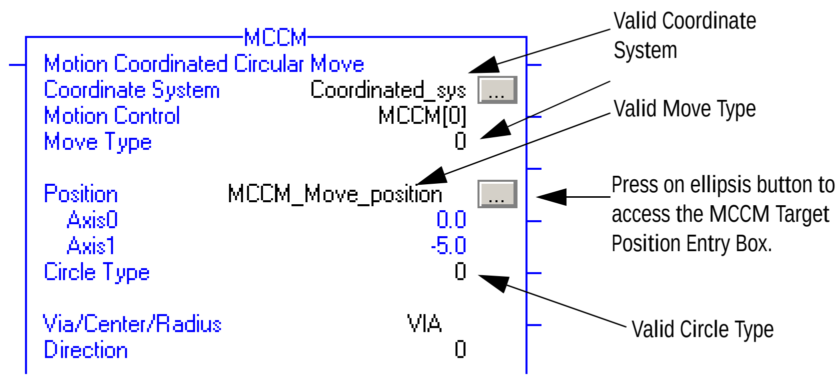
Any currently executing single axis motion instructions involving any axes defined in the specified coordinate system and any currently executing coordinated motion instructions are terminated. The prior motion is merged into the current move at the speed defined in Merge Speed parameter. Any pending coordinated move instructions are cancelled.

Merge Speed

The Merge Speed operand defines whether the current speed or the programmed speed is used as the maximum speed along the path of the coordinated move when Merge is enabled. Current speed is the vector sum of all motion (jogs, MAM's, geared motion, etc.) for all axes defined in the current coordinate system.

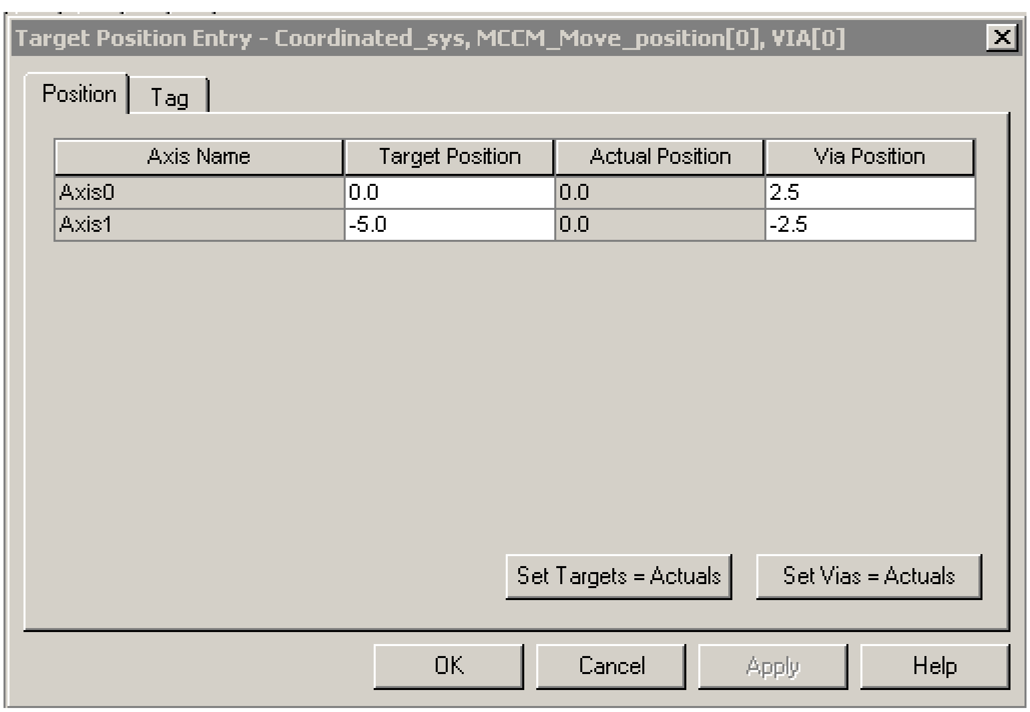
MCCM Target Position Entry Dialog Box

The MCCM Target Position Entry Dialog box is accessed by pressing the ellipsis button to the right of the position operand of the ladder instruction faceplate. The Target Position Entry box can only be accessed if the coordinate system for the instruction has been named, has a valid tag name for the Position operand that contains enough elements to accommodate the number of axes, selected a valid Move Type and a valid Circle Type. If these criteria have not been satisfied, an error message is displayed on the status bar.



MCCM Ladder Valid Values for Accessing Target Position Entry Box

Press the ellipsis and the following dialog box displays.



MCCM Instruction Target Position Entry Dialog Box - Position Tab

Feature	Description
Axis Name	This column has the names of each axis in the coordinate system named in the ladder faceplate. These names are not editable.
Target Position/Target Increment	The values in this column are numeric. They show the endpoint or incremental departure of the move depending on the active Move Type. The column heading indicates which is displayed.
Actual Position	This column contains the current actual position of the axes in the coordinate system. These values update dynamically when on-line and the Coordinate System Auto Tag Update is enabled.
Via Position/Via Increment Center Position/Center Increment Radius	Depending on the Circle Type selected, this column contains the Via point position or increment, the Center Position or increment.
Set Targets = Actuals	This button is enabled when the Move Type is Absolute and is used to copy the value from the Actual Position fields to the Target Position fields.
Set Vias = Actuals	This button is only active if the Move Type is Absolute. It is used to copy the values from the Actual Position fields to the Vias Fields.

The Move Type and Circle Type selected govern the appearance of this dialog box. The following table illustrates how the screen is affected by the combinations of Move Type and Circle Type selected.

Move Type	Circle Type	Behavior
Absolute	Via	Target column is entitled Target Position. Via column is entitled Via Position. Set Targets = Actuals button is active. Set Vias = Actuals button is active.
Incremental	Via	Target column is entitled Target Increment. Via Column is entitled Via Increment. Set Targets = Actuals button is inactive (Grayed Out). Set Vias = Actuals button is inactive (Grayed Out).
Absolute	Center	Target column is entitled Target Position. Center column is entitled Center Position. Set Targets = Actuals button is active. Set Vias = Actuals button is active.

Move Type	Circle Type	Behavior
Incremental	Center	Target column is entitled Target Increment. Center Column is entitled Center Increment. Set Targets = Actuals button is inactive (Grayed Out). Set Vias = Actuals button is inactive (Grayed Out).
Absolute	Radius	Target column is entitled Target Position. Radius column is entitled Radius. Set Targets = Actuals button is active. Set Vias = Actuals button is inactive (Grayed Out).
Incremental	Radius	Target column is entitled Target Increment. Radius Column is entitled Radius. Set Targets = Actuals button is inactive (Grayed Out). Set Vias = Actuals button is inactive (Grayed Out).
Absolute	Center Incremental	Target column is entitled Target Position. Center Incremental column is entitled Center Incremental. Set Targets = Actuals button is active. Set Vias = Actuals button is inactive (Grayed Out).
Incremental	Center Incremental	Target column is entitled Target Increment. Center Incremental column is entitled Center Incremental. Set Targets = Actuals button is inactive (Grayed Out). Set Vias = Actuals button is inactive (Grayed Out).

MCCM is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes: See Error Codes (ERR) for Motion Instructions.

Extended Error Codes:

Extended Error codes help to further define the error message given for this particular instruction. Their behavior is dependent upon the Error Code with which they are associated.

The Extended Error Codes for Servo Off State (5), Shutdown State (7), Axis Type Not Servo (8), Axis Not Configured (11), Homing In Process Error (16), and Illegal Axis Data type (38) errors all function in the same fashion. A number between 0 and *n* is displayed for the Extended Error Code. This number is the index to the Coordinate System indicating the axis that is in the error condition.

For Error Code Axis Not Configured (11) there is an additional value of -1 which indicates that Coordinate System was unable to setup the axis for coordinate motion.

For the MCCM instruction, Error Code 13 - Parameter Out of Range, Extended Errors return a number that indicates the offending parameter as listed on the faceplate in numerical order from top to bottom beginning with zero. For example, 2 indicates the parameter value for Move Type is in error.

Error Code and (Number)	Extended Error Numeric Indicator	Instruction Parameter	Description
Parameter Out Of Range (13)	0	Coordinate System	Number of primary axes is not 2 or 3.
Parameter Out Of Range (13)	2	Move Type	Move Type is either less than 0 or greater than 1.
Parameter Out Of Range (13)	3	Position	The position array is not large enough to provide positions for all the axes in the coordinate system.
Parameter Out Of Range (13)	4	Circle Type	Circle Type is either less than 0 or greater than 4.
Parameter Out Of Range (13)	5	Via/Center/Radius	The size of the Via/Center array is not large enough to provide positions for all of the axes in the defining via/center point.
Parameter Out Of Range (13)	6	Direction	Direction is either less than 0 or greater than 3.
Parameter Out Of Range (13)	7	Speed	Speed is less than 0.
Parameter Out Of Range (13)	9	Accel Rate	Accel Rate is less than or equal to 0.
Parameter Out Of Range (13)	11	Decel Rate	Decel Rate is less than or equal to 0.
Parameter Out Of Range (13)	14	Termination Type	Termination Type is less than 0 or greater than 3.

For the Error Code 54 – Maximum Deceleration Value is Zero, if the Extended Error returns a positive number (0-*n*) it is referring to the offending axis in the coordinate system. Go to the Coordinate System Properties General Tab and look under the Brackets ([ ])column of the

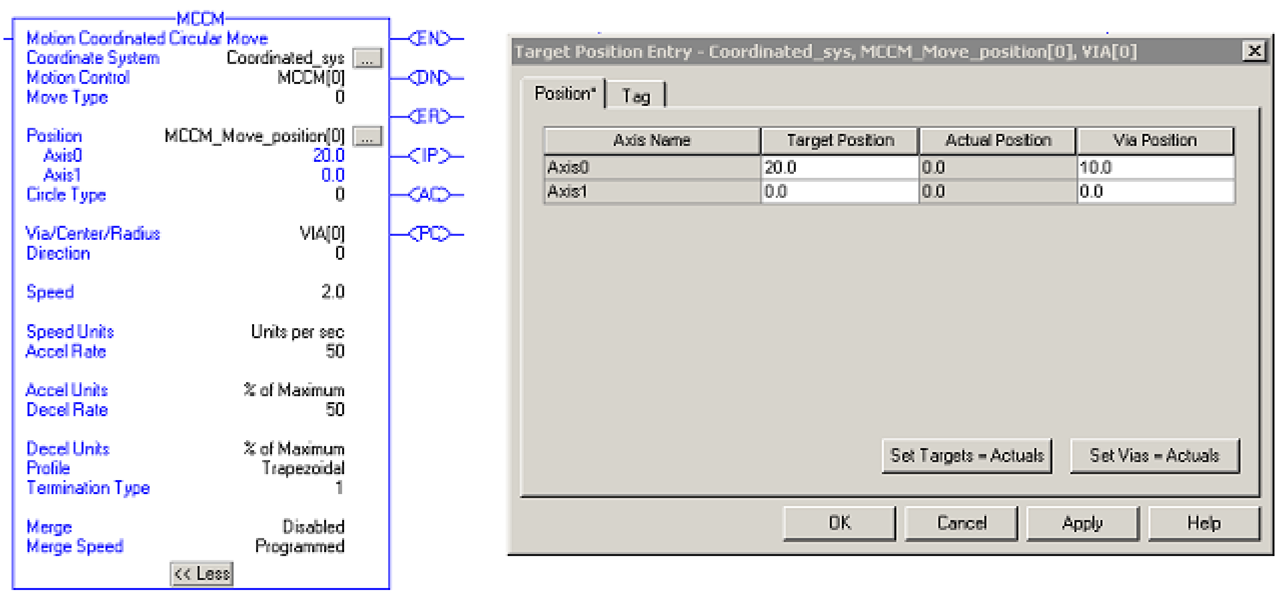
Axis Grid to determine which axis has a Maximum Deceleration value of 0. Click on the ellipsis button next to the offending axis to access the Axis Properties screen. Go to the Dynamics tab and make the appropriate change to the Maximum Deceleration Value. If the Extended Error number is -1, this means the Coordinate System has a Maximum Deceleration Value of 0. Go to the Coordinate System Properties Dynamics Tab to correct the Maximum Deceleration value.

Circular Error Examples

Due to the complexity of the MCCM instruction and the error codes it can generate, the following simple examples are given to aide in the understanding of the MCCM instruction.

CIRCULAR_COLLINEARITY_ERROR (44) Example

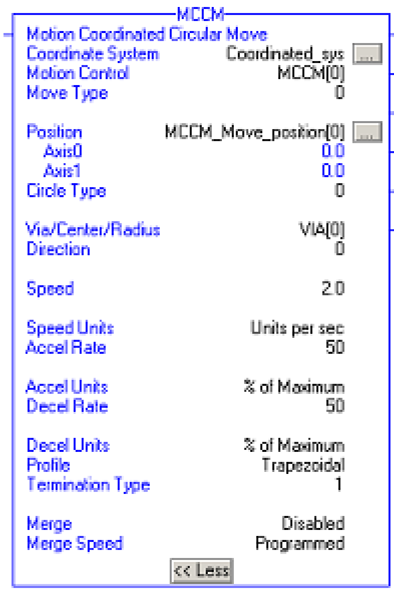
The following example for error #44 shows a situation where the startpoint, via-point, and endpoint all lie on a straight line. The program is trying to generate a two dimensional arc going from 0,0 (current position) to 20,0 through the location 10,0. Because these points all lie on a straight line no circular centerpoint can be computed for the circle. This error would also be generated if the program was for a three dimensional center type circle using a startpoint, centerpoint, and endpoint all lying on a straight line. Here, an infinite number of circles could be fit through the programmed points in an infinite number of planes.

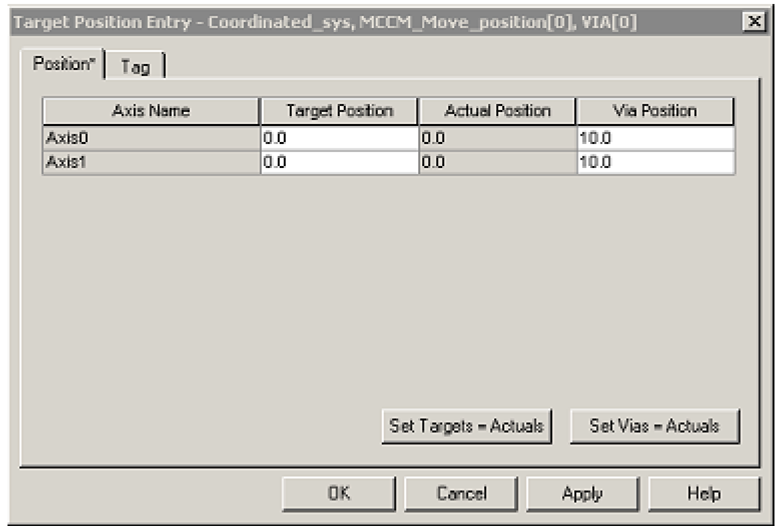


Ladder Program and Target Entry Screen that Generate Error #44

CIRCULAR_START_END_ERROR (45) Example

The following example for error #45 depicts a situation where the startpoint and via-point are the same. The program is trying to generate a two dimensional full circle from 0,0 (current position) back to 0,0 through the location 10,10. Because the startpoint and the via-point are the same, no circular centerpoint can be found for this circle.

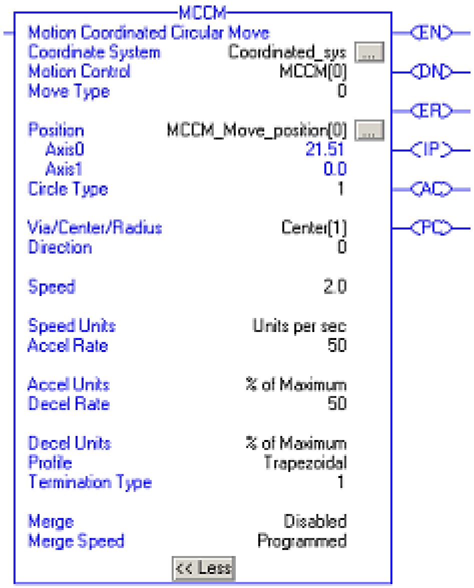


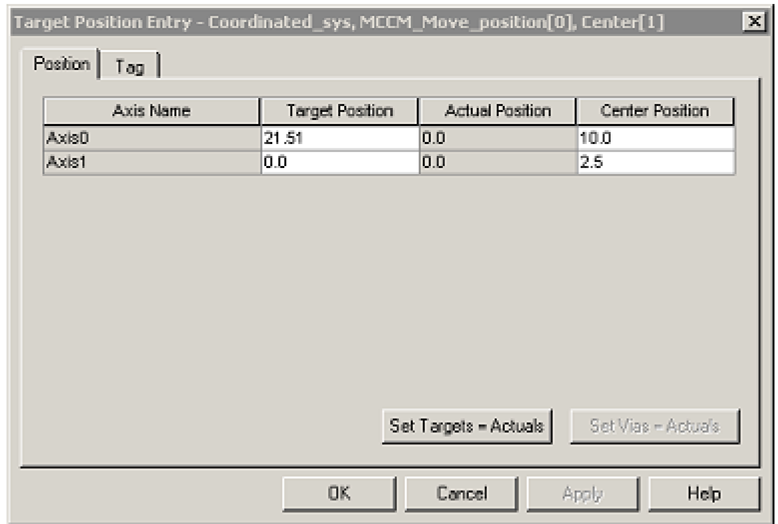


Ladder Program and Target Entry Screen that Generate Error #45

CIRCULAR_R1_R2_MISMATCH_ERROR (46) Example

The following example for error #46 shows a situation where the difference in radial start/end lengths exceeds 15% of the radial start length. The program is trying to generate a two dimensional arc from 0,0 (current position) to 21.51,0 using a centerpoint at 10,10. Because the difference of the radial start/end lengths is 21.51 - 10 = 1.51 it exceeds 15% of the radial start length .15 * 10 = 1.5. Had the endpoint been 21.5 this example would have worked, and the centerpoint would have been recomputed to lie exactly halfway between start and end points.

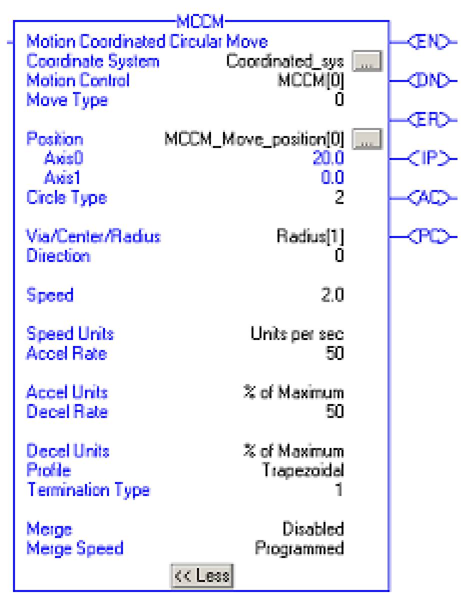


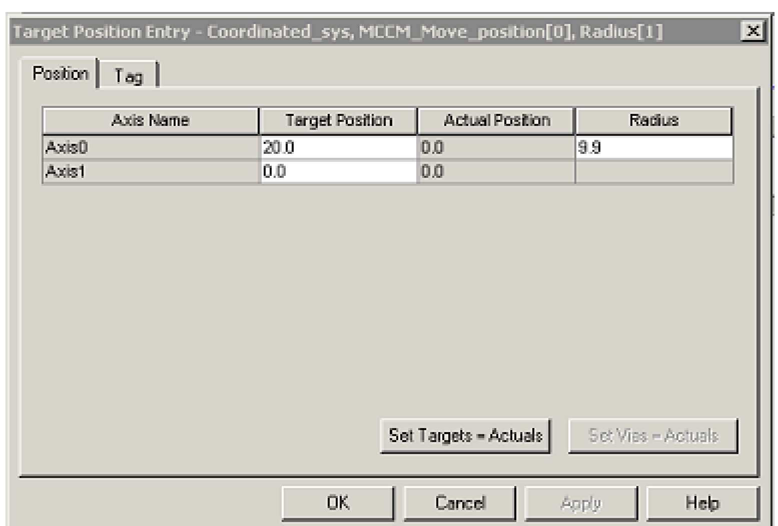


Ladder Program and Target Entry Screen that Generate Error #46

CIRCULAR_SMALL_R_ERROR (49) Example

This first example of error #49 depicts a situation where the radius type circle uses a radius that is too short to span the distance between the start point and the end point. The program is trying to generate a two dimensional arc going from 0,0 (current position) to 20,0. However, the user tried to program a radius type circle with a radius that is too short to span the distance between the startpoint and endpoint.

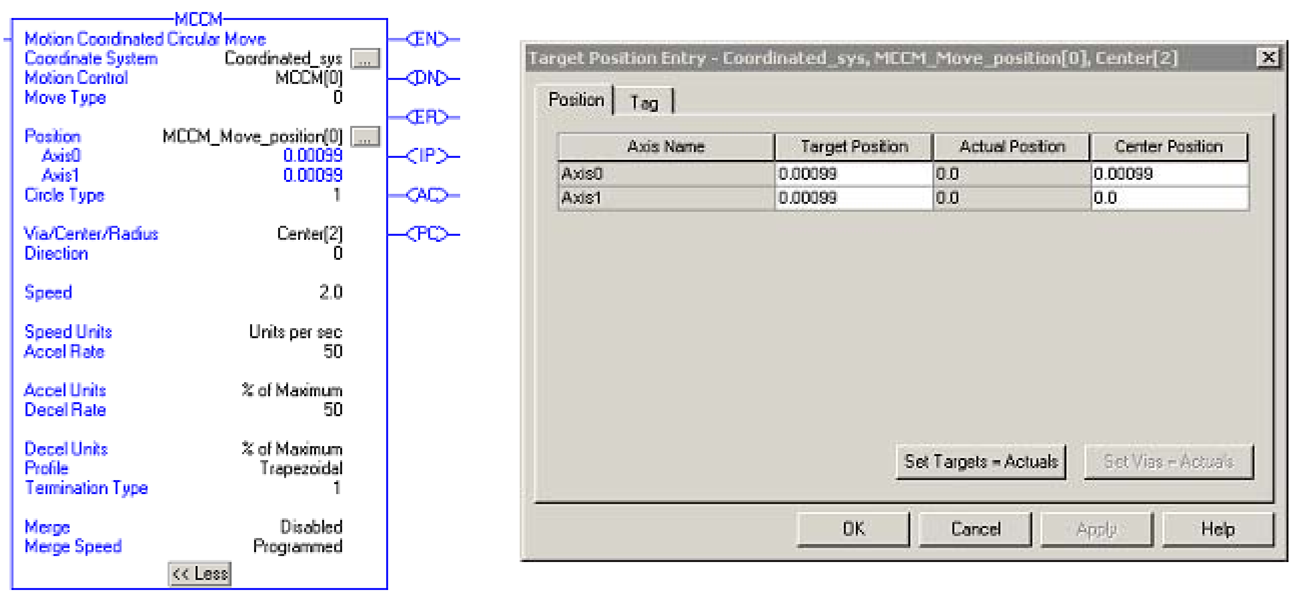




Ladder Program and Target Entry Screen that Generate Error #49

CIRCULAR_SMALL_R_ERROR (49) Example

This second example of error #49 shows a situation where the radius type circle uses a radius of magnitude of less than 0.001. The program is trying to generate a two dimensional arc going from 0,0 (current position) to 0.00099,0.00099. This error occurs because the user tried to program a radius type circle with a radius of a magnitude less than 0.001 units.



Ladder Program and Target Entry Screen that Generate Error #49

MCCM Changes to Status Bits:

Status Bits provide a means for monitoring the progress of the motion instruction. There are three types of Status Bits that provide pertinent information. They are: Axis Status Bits, Coordinate System Status Bits, and Coordinate Motion Status Bits. When the MCCM instruction initiates, the status bits undergo the following changes.

Axis Status Bits

Bit Name	Meaning
CoordinatedMotionStatus	Sets when the MCCM instruction executes and is cleared when the instruction completes.

Coordinate System Status Bits

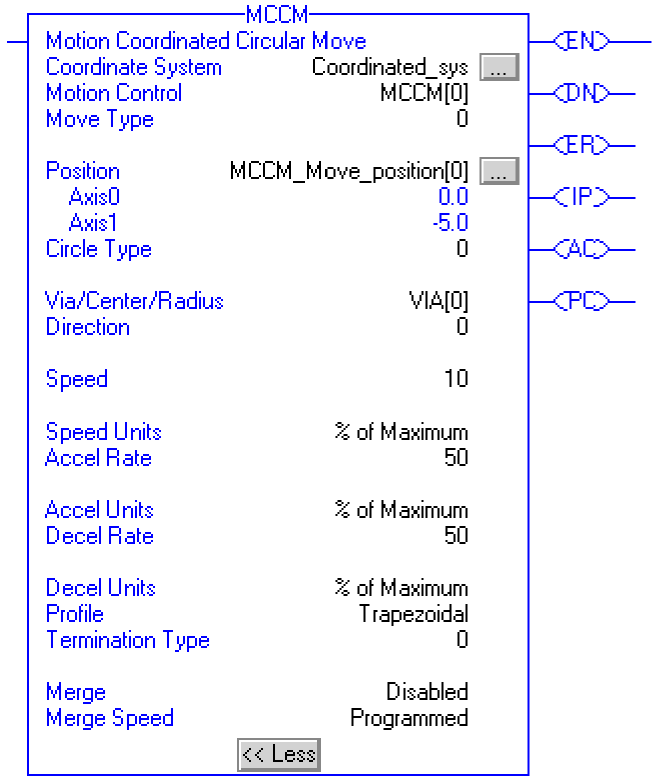
Bit Name	Meaning
MotionStatus	Sets when the MCCM instruction is active and the Coordinate System is connected to its associated axes.

Coordinate Motion Status Bits

Bit Name	Meaning
AccelStatus	Sets when vector is accelerating. Clears when a blend is in process or when vector move is at speed or decelerating.
DecelStatus	Sets when vector is decelerating. Clears when a blend is in process or when vector move is accelerating or when move completes.
ActualPosToleranceStatus	Sets for Actual Tolerance Termination Type only. The bit is set after the following two conditions have been met. 1) Interpolation is complete. 2) The actual distance to the programmed endpoint is less than the configured coordinate system's Actual Tolerance value. It remains set after the instruction completes. It is reset when a new instruction is started.
CommandPosToleranceStatus	Sets for all Termination Types whenever the distance to the programmed endpoint is less than the configured coordinate system's Command Tolerance value and remains set after the instruction completes. It is reset when a new instruction is started.
StoppingStatus	The Stopping Status bit is cleared when the MCCM instruction executes.
MoveStatus	Sets when MCCM begins axis motion. Clears on the .PC bit of the last motion instruction or a motion instruction executes which causes a stop.
MoveTransitionStatus	Sets when No Decel or Command Tolerance Termination Type is satisfied. When blending collinear moves the bit is not set because the machine is always on path. It clears when a blend completes, the motion of a pending instruction starts, or a motion instruction executes which causes a stop. Indicates not on path.
MovePendingStatus	Sets when one pending coordinated motion instruction is in the instruction queue. Clears when the instruction queue is empty.
MovePendingQueueFullStatus	Sets when the instruction queue is full. It clears when the queue has room to hold another new coordinated move instruction.

Currently, Coordinated Motion only supports the queueing of one coordinated motion instruction. Therefore the MovePendingStatus bit and the MovePendingQueueFullStatus bit are always the same.

Example: Relay Ladder



MCCM Ladder Instruction

Structured Text

```
MCCM(Coordinated_sys,MCCM[0],0,MCCM_Move_position,0.0,-5.0,
via,0,0,10,%ofmaximum,50,%ofmaximum,50,%ofmaximum,
Trapezoidal,0,Disabled,programmed);
```

Circular Programming Reference Guide

<table>
<tr><th>Circle Type</th><th>Used in 2D/3D/Both</th><th>Validation Errors</th><th>Direction – 2D</th><th>Direction – 3D</th><th>Comments</th></tr>
<tr><td>Radius</td><td>2D</td><td>Error 25; Illegal Instruction

Error 45 Endpoint = Startpoint

Error 49; R too small (|R| < .001) or R too short to span programmed points.</td><td>CW/CCW as viewed from the ’+’ perpendicular to the circular plane.</td><td>N/A</td><td>A ’+” radius forces arc length to be <= 180° (Shortest arc).

A “-” radius forces arc length to be => 180° (Longest arc).

Full Circles can be programmed.

For full circles: set “Position” to be any point on circle except Startpoint and use one of the “Full” direction types.</td></tr>
<tr><td>Center Point</td><td>Both</td><td>Error 44; Collinearity (3D only)

Error 45; Endpoint = Startpoint (3D only)

Error 46; Start/End radius mismatch (|R1 - R2| > .15 * R1).</td><td>CW/CCW as viewed from the ’+’ perpendicular to the circular plane.</td><td>Shortest/Longest arc. In Full circles, placement of endpoint defines shortest/longest paths referred to by direction parameter.</td><td>3. Full Circles can be programmed.

4. In 2D only, Endpoint = Startpoint is legal. Therefore, full circles may be generated:

a. By setting Endpoint = Startpoint, in which case, all direction types produce full circles.

b. By setting Endpoint not = Startpoint and using “Full” direction type.

5. For 3D Full Circles: set Position to be any point on the circle except Startpoint, and use one of the “Full” direction types. Position defines both arc and “Shortest” direction types.</td></tr>
<tr><td>Via Point</td><td>Both</td><td>Error 44; Collinearity

Error 45; Endpoint = Startpoint</td><td>Via point always determines direction.</td><td>Via point always determines direction. Direction operand is only used to determine if circle is partial or full.</td><td>1. Full Circles can be programmed.

2. For full circles: set “Position” to be any point on circle except Startpoint and use one of the “Full” direction types.</td></tr>
</table>

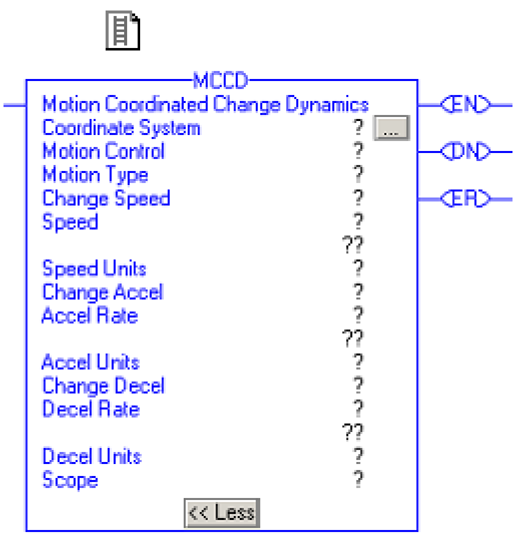
Motion Coordinated Change Dynamics (MCCD)

Use the MCCD instruction to initiate a change in the path dynamics for the motion active on the specified coordinate system.

ATTENTION

Use a motion control tag only once. Do not re-use it in another instruction. Otherwise, you can cause unexpected equipment motion and injure people.

Operands: *Relay Ladder*



Operand	Type	Format	Description
Coordinate System	COORDINATE_SYSTEM	tag	Coordinated group of axes.
Motion Control	MOTION_ INSTRUCTION	tag	Structure used to access instruction status parameters.
Motion Type	SINT, INT, or DINT	immediate	1 = Coordinated Move
Change Speed	SINT, INT, or DINT	immediate	0 = No 1 = Yes
Speed	SINT, INT, DINT, or REAL	immediate or tag	[Coordination Units]
Speed Units	SINT, INT, or DINT	immediate	0 = Units per Sec 1 = % of Maximum
Change Accel	SINT, INT, or DINT	immediate	0 = No 1 = Yes
Accel Rate	SINT, INT, DINT, or REAL	immediate or tag	[Coordination Units]
Accel Units	SINT, INT, or DINT	immediate	0 = Units per Sec^2 1 = % of Maximum
Change Decel	SINT, INT, or DINT	immediate	0 = No 1 = Yes
Decel Rate	SINT, INT, DINT, or REAL	immediate or tag	[Coordination Units]
Decel Units	SINT, INT, or DINT	immediate	0 = Units per Sec^2 1 = % of Maximum
Scope	SINT, INT, or DINT	immediate	0 = Active Motion

Structured Text

```
MCCD(CoordinateSystem,
MotionControl,MotionType
ChangeSpeed,Speed,SpeedUnits,
ChangeAccel,AccelRate,
AccelUnits,ChangeDecel,
DecelRate,DecelUnits,Scope);
```

The operands are the same as those for the relay ladder MCCD instruction.

When entering enumerations for the operand value in Structured Text, multiple word enumerations must be entered without spaces. For example: when entering Decel Units the value should be entered as unitspersec^2 rather than Units per Sec^2 as displayed in the ladder logic.

For the operands that have enumerated values, enter your selection as:

This operand	Has these options which you...	
	enter as text	**or enter as a number**
Motiontype	Coordinatedmove	1
ChangeSpeed	No Yes	0 1
SpeedUnits	Unitspersec %ofmaximum	0 1
ChangeAccel	No Yes	0 1
AccelUnits	Unitspersec^2 %ofmaximum	0 1
ChangeDecel	No Yes	0 1
DecelUnits	Unitspersec^2 %ofmaximum	0 1
Scope	activemotion	0

Description: The Motion Coordinated Change Dynamics (MCCD) instruction starts a change in the path dynamics of the specified coordinate system. Based upon the Motion Type, the MCCD changes the coordinated motion profile that is currently active on the system.

ATTENTION

If You Use An S-curve Profile

Be careful if you change the speed, acceleration, deceleration, or jerk while an axis is accelerating or decelerating along an S-curve profile. You can cause an axis to **overshoot its speed or reverse direction**.

For more information, see Troubleshoot Axis Motion on page 367.

Coordinate System

The Coordinate System operand specifies the set of motion axes that define the dimensions of a coordinate system. For this release the coordinate system supports up to three (3) primary axes.

Motion Control

The following control bits are affected by the MCCD instruction.

Mnemonic	Description
.EN (Enable) Bit 31	The Enable Bit is set when the rung transitions from false to true. It resets when the rung transitions from true to false.
.DN (Done) Bit 29	The Done Bit resets when the rung transitions from false to true. It sets when target position is calculated successfully.
.ER (Error) Bit 28	The Error Bit resets when the rung transitions from false to true. It sets when target position fails to calculate successfully.

Motion Type

The motion type operand determines which motion profile to change. Currently Coordinated Move is the only available option.

Coordinated Move

When selected, the Coordinated Move option changes the motion of the currently active move in the coordinate system.

Change Speed

The Change Speed operand determines whether or not to change the speed of the coordinated motion profile.

No

No change is made to the Speed of the coordinated motion.

Yes

The speed of the coordinated motion is changed by the value defined in the Speed and Speed Units operands.

Speed

The Speed operand defines the maximum speed along the path of the coordinated move.

Speed Units

The Speed Units operand defines the units applied to the Speed operand either directly in coordination units of the specified coordinate system or as a percentage of the maximum values defined in the coordinate system.

Change Accel

The Change Accel operand determines whether or not to change the acceleration of the coordinated motion profile.

No

No change is made to the acceleration of the coordinated motion.

Yes

The acceleration of the coordinated motion is changed by the value defined in the Accel Rate and Accel Units operands.

Accel Rate

The Accel Rate operand defines the maximum acceleration along the path of the coordinated move.

Accel Units

The Accel Units operand defines the units applied to the Accel Rate operand either directly in coordination units of the specified coordinate system or as a percentage of the maximum values defined in the coordinate system.

Change Decel

The Change Decel operand determines whether or not to change the deceleration of the coordinated motion profile.

No

No change is made to the deceleration of the coordinated motion.

Yes

The deceleration of the coordinated motion is changed by the value defined in the Decel Rate and Decel Units operands.

Decel Rate

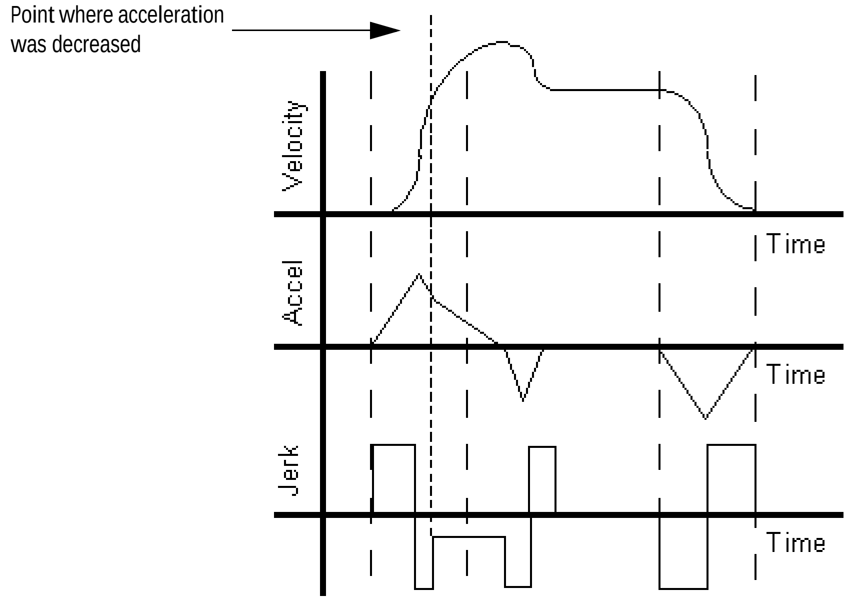
The Decel Rate operand defines the maximum deceleration along the path of the coordinated move.

Decel Units

The Decel Units operand defines the units applied to the Decel Rate operand either directly in coordination units of the specified coordinate system or as a percentage of the maximum values defined in the coordinate system.

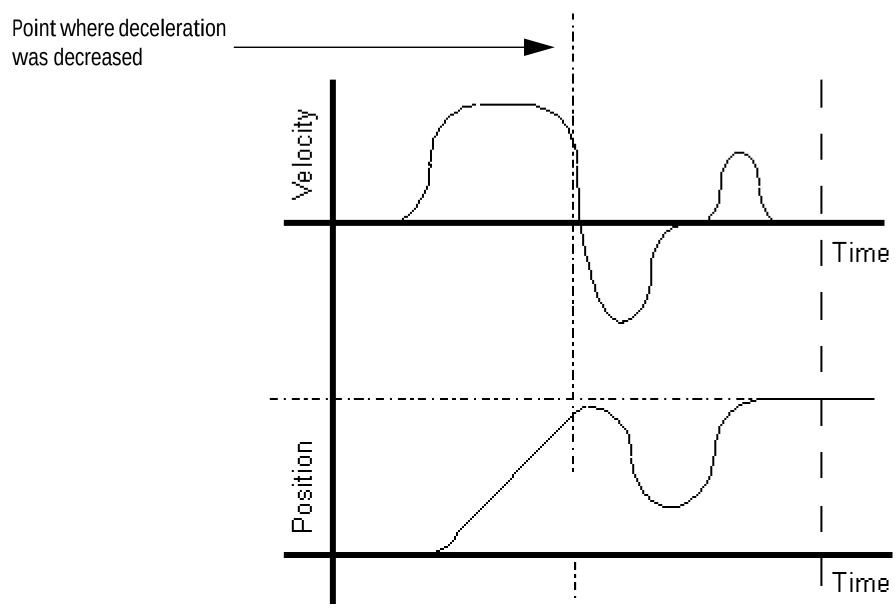
Impact of Changes to Acceleration and Deceleration Values on Motion Profile

The following graph illustrates what could happen when a MCCD instruction is used to reduce the acceleration as velocity approaches maximum. The new acceleration Jerk Rate becomes smaller, further limiting the maximum change in acceleration. Velocity overshoot occurs due to the additional time required for acceleration to reach zero. Another profile is generated to bring velocity back to the programmed maximum.



Effect of Change to Acceleration

The following graph illustrates what could happen when an MCCD instruction is used to reduce the deceleration as velocity and position approach their target endpoints. The new deceleration Jerk Rate becomes smaller. The time required to decelerate to zero causes velocity to undershoot, passing through zero and becoming negative. Axis motion also reverses direction until velocity returns to zero. An additional profile is generated to bring position back to the programmed target.



Affect of Change to Deceleration

Scope

The Scope operand lets you determine whether the changes are to affect the current active instruction.

MCCD is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes: See Error Codes (ERR) for Motion Instructions.

Extended Error Codes: Extended Error codes help to further define the error message given for this particular instruction.Their behavior is dependent upon the Error Code with which they are associated.

The Extended Error Codes for Servo Off State (5), Shutdown State (7), Axis Type Not Servo (8), Axis Not Configured (11), Homing In Process Error (16), and Illegal Axis Data type (38) errors all function in the same fashion. A number between 0 and *n* is displayed for the Extended Error Code. This number is the index to the Coordinate System indicating the axis that is in the error condition.

For the MCCD instruction, Error Code 13 - Parameter Out of Range, Extended Errors return a number that indicates the offending parameter as listed on the faceplate in numerical order from top to bottom beginning with zero. For example, 2 indicates the parameter value for Move Type is in error.

Referenced Error Code and Number	Extended Error Numeric Indicator	Instruction Parameter	Description
Parameter Out Of Range (13)	2	Move Type	Move Type is either less than 0 or greater than 1.
Parameter Out Of Range (13)	4	Speed	Speed is less than 0.
Parameter Out Of Range (13)	7	Accel Rate	Accel Rate is less than or equal to 0.
Parameter Out Of Range (13)	10	Decel Rate	Decel Rate is less than or equal to 0.

For the Error Code 54 – Maximum Deceleration Value is Zero, if the Extended Error returns a positive number (0-*n*) it is referring to the offending axis in the coordinate system. Go to the Coordinate System

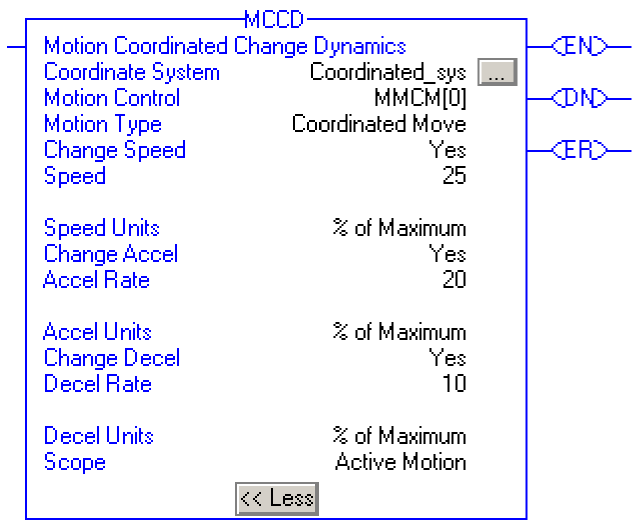
Properties General Tab and look under the Brackets ([ ])column of the Axis Grid to determine which axis has a Maximum Deceleration value of 0. Click on the ellipsis button next to the offending axis to access the Axis Properties screen. Go to the Dynamics tab and make the appropriate change to the Maximum Deceleration Value. If the Extended Error number is -1, this means the Coordinate System has a Maximum Deceleration Value of 0. Go to the Coordinate System Properties Dynamics Tab to correct the Maximum Deceleration value.

MCCD Changes to Status Bits:

No effect.

Example:

Relay Ladder



MCCD Ladder Instruction

Structured Text

```
MCCD(Coordinated_sys,MCCM[0],CoordinatedMove,Yes,25,
%ofmaximum,Yes,20,%ofmaximum,Yes,10,%ofmaximum,0)
```

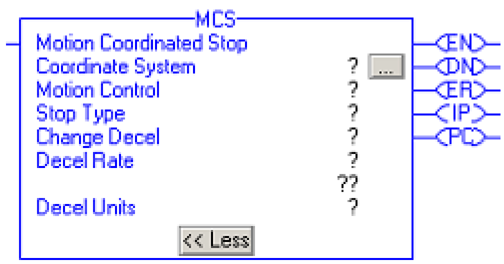
Motion Coordinated Stop (MCS)

Use the MCS instruction to stop the axes of a coordinate system or cancel a transform.

ATTENTION

Use a motion control tag only once. Do not reuse it in another instruction. Otherwise, you can cause unexpected equipment motion and injure people.

Operands: *Ladder Diagram*



<table>
<tr><th>Operand</th><th>Type</th><th>Format</th><th colspan="2">Description</th></tr>
<tr><td>Coordinate System</td><td>COORDINATE_SYSTEM</td><td>Tag</td><td colspan="2">Name of the coordinate system</td></tr>
<tr><td>Motion control</td><td>MOTION_ INSTRUCTION</td><td>Tag</td><td colspan="2">Control tag for the instruction</td></tr>
<tr><td rowspan="4">Stop Type</td><td rowspan="4">DINT</td><td rowspan="4">Immediate</td><td>If you want to</td><td>Choose this Stop Type</td></tr>
<tr><td>Stop all motion for the axes of the coordinate system and stop any transform that the coordinate system is a part of</td><td>All (0)</td></tr>
<tr><td>Stop only coordinated moves</td><td>Coordinated Move (2)</td></tr>
<tr><td>Cancel any transform that the coordinate system is a part of</td><td>Coordinated Transform (3)</td></tr>
</table>

<table>
<tr><th>Operand</th><th>Type</th><th>Format</th><th colspan="2">Description</th></tr>
<tr><td rowspan="3">Change Decel</td><td rowspan="3">DINT</td><td rowspan="3">Immediate</td><td>If you want to</td><td>Then choose</td></tr>
<tr><td>Use the maximum deceleration rate of the coordinate system</td><td>No (0)</td></tr>
<tr><td>Specify the deceleration rate</td><td>Yes (1)</td></tr>
<tr><td>Decel Rate</td><td>REAL</td><td>Immediate or tag</td><td colspan="2">Important: An axis could overshoot its target position if you reduce the deceleration while a move is in process.

Deceleration along the path of the coordinated move. The instruction uses this value:
• Only if Change Decel is Yes.
• Only for coordinated moves.
Enter a value greater than 0.</td></tr>
<tr><td>Decel Units</td><td>DINT</td><td>Immediate</td><td colspan="2">Which units do you want to use for the Decel Rate?
• Units per sec^2 (0)
• % of Maximum (1)</td></tr>
</table>

Structured Text

```
MCS(CoordinateSystem,
MotionControl,StopType,
ChangeDecel,
DecelRate,DecelUnits);
```

The structured text operands are the same as the ladder diagram operands. Enter the stop type and decel units without spaces.

Example: Enter a stop type of Coordinated Move as CoordinatedMove.

Description: The Motion Coordinated Stop (MCS) instruction initiates a controlled stop of coordinated motion. Any pending motion profiles are cancelled.

ATTENTION

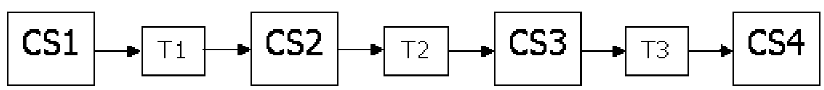
If You Use An S-curve Profile

Be careful if you change the speed, acceleration, deceleration, or jerk while an axis is accelerating or decelerating along an S-curve profile. You can cause an axis to **overshoot its speed or reverse direction**.

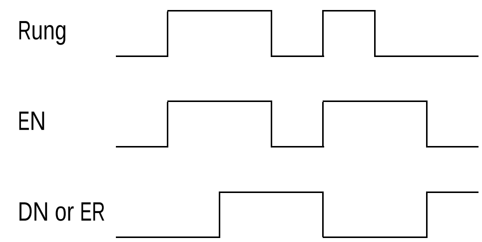
For more information, see Troubleshoot Axis Motion.

How Stop Types Affect Transforms

The following table describes how the stop types affect coordinate systems that are a part of a transform.

Stop type	Description
All	This stop type: • Stops the axes in the specified coordinate system. It also stops the axes of any coordinate system that shares axes with this coordinate system. • Cancels any transforms that the coordinate system is a part of.
Coordinated Move	This stop type stops only the coordinated moves. Any transforms stay active.
Coordinated Transform	This stop type cancels the transforms associated with the specified coordinate system. All transform-related motion stops on all associated target coordinate systems. However, source coordinate axes will continue to move as instructed. **Example** Suppose four coordinate systems are linked via three transforms. And suppose with the first coordinate system (CS1) is the source and is processing commanded motion.  If you execute an MCS instruction on CS2 and use a stop type of coordinated transform, then. • Transforms T1 and T2 are canceled. • Transform T3 stays active. • The axes in CS1 stay in motion. • The axes in Coordinate Systems CS2 and CS3 stop via the deceleration rate selected in the MCS instruction or the maximum coordinate deceleration rate. • The axes in CS4 follow the respective CS3 axes. In an Motion Axis Stop (MAS) instruction, a stop type of all also cancels transforms.

MOTION_INSTRUCTION Data Type

To see if	Check if this bit is on	Data type	Notes
The rung is true.	EN	BOOL	Sometimes the EN bit stays on even if the rung goes false. This happens if the rung goes false before the instruction is done or errored. 
The stop was successfully initiated.	DN	BOOL	
An error happened.	ER	BOOL	
The axis is stopping.	IP	BOOL	Any of these actions ends the MCS instruction and turns off the IP bit: • The coordinate system is stopped. • Another MCS instruction supersedes this MCS instruction. • Shutdown instruction. • Fault Action.
The axis is stopped.	PC	BOOL	The PC bit stays on until the rung makes a false-to-true transition.

Arithmetic Status Flags: not affected

Fault Conditions: none

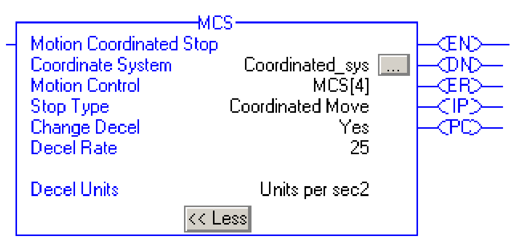
Error Codes: See Error Codes (ERR) for Motion Instructions.

Extended Error Codes: See Error Codes (ERR) for Motion Instructions. It has information about how to use the extended error codes.

Changes to Status Bits: The instruction changes these status bits when it executes.

In the tag for the	This bit	When the stop type is	Turns
Axis	CoordinatedMotionStatus	→	Off when the coordinated move stops
	TransformStateStatus	Coordinated Move	Unchanged
		• All • Coordinated Transform	Off
	ControlledByTransformStatus	Coordinated Move	Off when the axes stop and the PC bit of the MCS instruction turns on
		• All • Coordinated Transform	Off
Coordinate system	MotionStatus	→	Off when the coordinated move stops
	AccelStatus	→	Off
	DecelStatus	→	On during the stop and then off when the stop completes
	StoppingStatus	→	On during the stop and then off when the PC bit turns on
	MoveStatus	→	Off
	MoveTransitionStatus	→	Off
	MovePendingStatus	→	Off
	TransformSourceStatus	Coordinated Move	Unchanged
		• All • Coordinated Transform	Off
	TransformTargetStatus	Coordinated Move	Unchanged
		• All • Coordinated Transform	Off

Example 1: *Ladder Diagram*

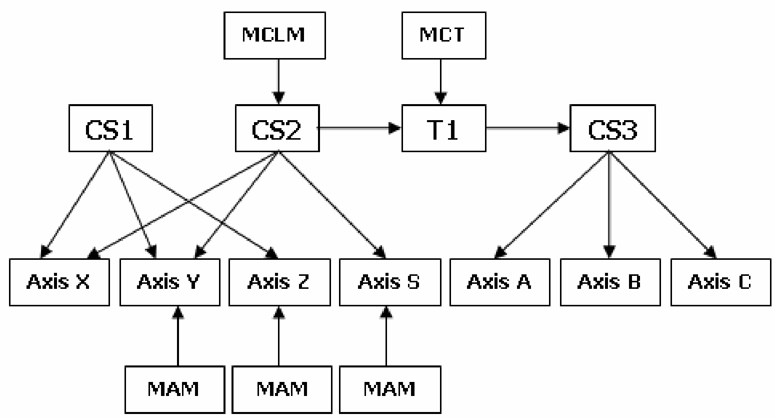


Structured Text

```
MCS(Coordinated_sys,MCS[4],CoordinatedMove,Yes,25,
Unitspersec2);
```

Example 2: *How Stop Types Affect Transforms and Axis Motion*

Suppose you have this situation.



Where:

- Coordinate system 1 (CS1) contains the X, Y, and Z axes.
- Coordinate system 2 (CS2) contains the Y, Z, and S axes.
- Coordinate system 3 (CS3) contains the A, B, and C axes.
- Transform (T1) links source coordinate CS2 to target CS3.
- CS2 (XYS) axes are mapped to CS3 (ABC) axes.
- MAM instructions executed on the Y, Z, and S axes.
- MCLM instruction executed on CS2.
- MCT instruction executed with CS2 as the source and CS3 as the target.
- No coordinate instructions were executed on CS2 or CS3.

This table shows the results of executing various MCS and MAS instructions with different stop types.

Instruction	Stop Type	Result
MCS on CS1	All	The MCLM instruction on CS2 will stop.
		The MAM on Y will stop.
		The MAM on Z will stop.
		The MAM on S will continue.
		T1 is canceled.
		Axes ABC will stop due to canceling the transform.

Instruction	Stop Type	Result
MCS on CS2	All	The MCLM instruction on CS2 will stop.
		The MAM on Y will stop.
		The MAM on S will stop.
		The MAM on Z will continue.
		T1 is canceled.
		Axes ABC will stop due to canceling the transform.
MCS on CS3	All	The MCLM instruction on CS2 will continue.
		The MAM on Y will continue.
		The MAM on S will continue.
		The MAM on Z will continue.
		T1 is canceled.
		Axes ABC will stop due to canceling the transform.
MCS on CS1	Coordinated Move	The MCLM instruction on CS2 will continue.
		The MAM on Y will continue.
		The MAM on S will continue.
		The MAM on Z will continue.
		T1 stays active.
		Axes ABC will follow the respective CS2 axes.
MCS on CS2	Coordinated Move	The MCLM instruction on CS2 will stop.
		The MAM on Y will continue.
		The MAM on S will continue.
		The MAM on Z will continue.
		T1 stays active.
		Axes ABC will follow the respective CS2 axes.
MCS on CS3	Coordinated Move	The MCLM instruction on CS2 will continue.
		The MAM on Y will continue.
		The MAM on S will continue.
		The MAM on Z will continue.
		T1 stays active.
		Axes ABC will follow the respective CS2 axes.
MAS on Y	All	The MCLM instruction on CS2 will stop.
		The MAM on Y will stop.
		The MAM on S will continue.
		The MAM on Z will continue.
		T1 is canceled.
		Axes ABC will stop due to canceling the transform.

Instruction	Stop Type	Result
MAS on Y	Move	The MCLM instruction on CS2 will continue.
		The MAM on Y will stop.
		The MAM on S will continue.
		The MAM on Z will continue.
		T1 stays active.
		Axes ABC will follow the respective CS2 axes.
MAS on Z	All	The MCLM instruction on CS2 will continue.
		The MAM on Y will continue.
		The MAM on S will continue.
		The MAM on Z will stop.
		T1 stays active.
		Axes ABC will follow the respective CS2 axes.
MAS on Z	Move	The MCLM instruction on CS2 will continue.
		The MAM on Y will continue.
		The MAM on S will continue.
		The MAM on Z will stop.
		T1 stays active.
		Axes ABC will follow the respective CS2 axes.
MCS on CS1	Coordinated Transform	MCLM instruction on CS2 continues.
		The MAM on Y will continue.
		The MAM on S will continue.
		The MAM on Z will continue.
		T1 stays active.
		Axes ABC will follow the respective CS2 axes.
MCS on CS2	Coordinated Transform	T1 is canceled.
		MCLM instruction on CS2 continues.
		The MAM on Y will continue.
		The MAM on S will continue.
		The MAM on Z will continue.
		Axes ABC will stop due to canceling the transform.
MCS on CS3	Coordinated Transform	T1 is canceled.
		MCLM instruction on CS2 continues.
		The MAM on Y will continue.
		The MAM on S will continue.
		The MAM on Z will continue.
		Axes ABC will stop due to canceling the transform.

Motion Coordinated Shutdown (MCSD)

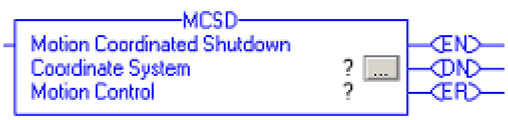
Use the Motion Coordinated Shutdown (MCSD) instruction to perform a controlled shutdown of all the axes in the named coordinate system.

ATTENTION

Use a motion control tag only once. Do not re-use it in another instruction. Otherwise, you can cause unexpected equipment motion and injure people.

Operands:

Relay Ladder



Operand	Type	Format	Description
Coordinate System	COORDINATE_SYSTEM	tag	Coordinated group of axes.
Motion Control	MOTION_INSTRUCTION	tag	Structure used to access instruction status parameters.

Structured Text

```
MCSD(CoordinateSystem,
MotionControl);
```

The operands are the same as those for the relay ladder MCSD instruction.

Description:

The Motion Coordinated Shutdown (MCSD) instruction shuts down all of the axes in the associated coordinate system.

Coordinate System

The Coordinate System operand specifies the set of motion axes that define the dimensions of a Cartesian coordinate system. For this release the coordinate system supports up to three (3) primary axes. Only the axes configured as primary axes (up to 3) are included in the coordinate velocity calculations.

Motion Control

The following control bits are affected by the MCSD instruction.

Mnemonic	Description
.EN (Enable) Bit 31	The Enable Bit sets when the rung transitions from false to true. It resets when the rung goes from true to false.
.DN (Done) Bit 29	The Done Bit sets when the coordinated shutdown is successfully initiated. It resets when the rung transitions from false to true.
.ER (Error) Bit 28	The Error Bit sets when the coordinated shutdown fails to initiate successfully. It resets when the rung transitions from false to true.

MCSD is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes: See Error Codes (ERR) for Motion Instructions.

MCSD Changes to Status Bits: Status Bits provide a means for monitoring the progress of the motion instruction. There are three types of Status Bits that provide pertinent information. They are: Axis Status Bits, Coordinate System Status Bits, and Coordinate Motion Status Bits. When the MCS instruction initiates, the status bits undergo the following changes.

Axis Status Bits

Bit Name	Effect
CoordinatedMoveStatus	Cleared

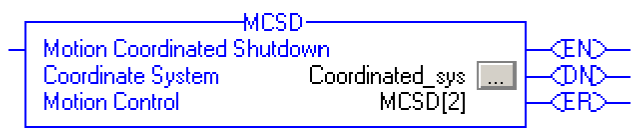
Coordinate System Status Bits

Bit Name	Effect
ShutdownStatus	Sets when MCSD is executed and all associated axes are shutdown.
ReadyStatus	Cleared after MCSD executes.

Coordinate Motion Status Bits

Bit Name	Effect
AccelStatus	Cleared after MCSD executes.
DecelStatus	Cleared after MCSD executes.
ActualPosToleranceStatus	Cleared after MCSD executes.
CommandPosToleranceStatus	Cleared after MCSD executes.
StoppingStatus	Cleared after MCSD executes.
MoveStatus	Cleared after MCSD executes.
MoveTransitionStatus	Cleared after MCSD executes.
MovePendingStatus	Cleared after MCSD executes.
MovePendingQueueFullStatus	Cleared after MCSD executes.

Example: Relay Ladder



MCSD Ladder Instruction

Structured Text

```
MCSD(Coordinated_sys,MCSD[2]);
```

Motion Coordinated Transform (MCT)

Use the MCT instruction to start a transform that links two coordinate systems together.

ATTENTION

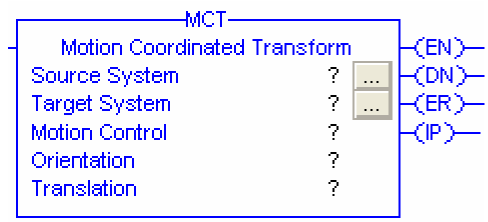
Use a motion control tag only once. Do not reuse it in another instruction. Otherwise, you can cause unexpected equipment motion and injury to people.

IMPORTANT

You can use this instruction only with 1756-L6x controllers.

Operands

Ladder Diagram



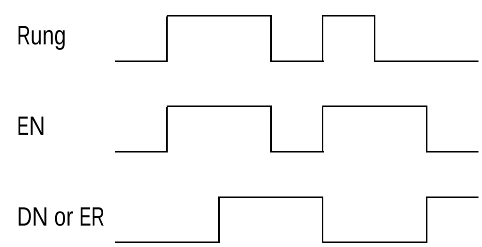
<table>
<tr><th>Operand</th><th>Type</th><th>Format</th><th colspan="2">Description</th></tr>
<tr><td>Source System</td><td>COORDINATE_SYSTEM</td><td>Tag</td><td colspan="2">Coordinate system that you use to program the moves. Typically, this is the Cartesian coordinate system.</td></tr>
<tr><td>Target System</td><td>COORDINATE_SYSTEM</td><td>Tag</td><td colspan="2">Non-Cartesian coordinate system that controls the actual equipment</td></tr>
<tr><td>Motion Control</td><td>MOTION_INSTRUCTION</td><td>Tag</td><td colspan="2">Control tag for the instruction</td></tr>
<tr><td rowspan="5">Orientation</td><td rowspan="5">REAL[3]</td><td rowspan="5">Array</td><td colspan="2">Do you want to rotate the target position around the X1, X2, or X3 axis?</td></tr>
<tr><td>If</td><td>Then</td></tr>
<tr><td>No</td><td>Leave the array values at zero.</td></tr>
<tr><td>Yes</td><td>Enter the degrees of rotation into the array. Put the degrees of rotation around X1 in the first element of the array, and so on.</td></tr>
<tr><td colspan="2">Use an array of three REALs even if a coordinate system has only one or two axes.</td></tr>
<tr><td rowspan="5">Translation</td><td rowspan="5">REAL[3]</td><td rowspan="5">Array</td><td colspan="2">Do you want to offset the target position along the X1, X2, or X3 axis?</td></tr>
<tr><td>If</td><td>Then</td></tr>
<tr><td>No</td><td>Leave the array values at zero.</td></tr>
<tr><td>Yes</td><td>Enter the offset distances into the array. Enter the offset distances in coordination units. Put the offset distance for X1 in the first element of the array, and so on.</td></tr>
<tr><td colspan="2">Use an array of three REALs even if a coordinate system has only one or two axes.</td></tr>
</table>

```
MCT(Source System, Target
System, Motion Control,
Orientation,
Translation);
```

Structured Text

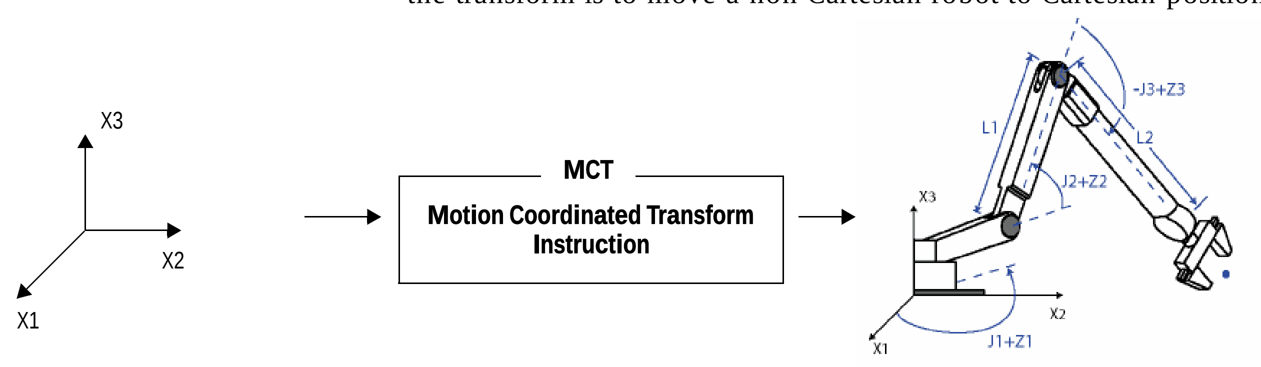
The structured text operands are the same as the ladder diagram operands.

MOTION_INSTRUCTION Data Type

To see if	Check if this bit is on	Data type	Notes
The rung is true.	EN	BOOL	Sometimes the EN bit stays on even if the rung goes false. This happens if the rung goes false before the instruction is done or errored. 
The instruction is done.	DN	BOOL	The transform process keeps running after the instruction is done.
An error happened.	ER	BOOL	Identify the error number listed in the error code field of the Motion control tag then, refer to Error Codes (ERR) for Motion Instructions on page 383 of this manual.
The transform process is running.	IP	BOOL	Any of these actions cancels the transform and turns off the IP bit: • Applicable stop instruction • Shutdown instruction • Fault action

Description

Use the MCT instruction to start a transform that links two coordinate systems together. This is like bi-directional gearing. One way to use the transform is to move a non-Cartesian robot to Cartesian positions.



You move a system of virtual axes to Cartesian positions (X1, X2, X3).

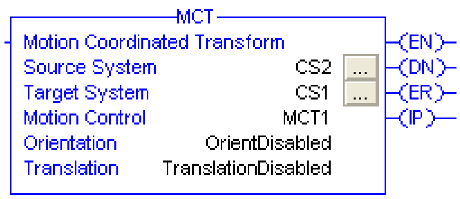
The transform converts the motion to joint angles and moves the robot.

The transform controls up to three joints of the robot: J1, J2, and J3.

Data Flow of MCT Instruction Between Two Coordinate Systems

The following illustrations show the flow of data when an MCT Instruction is active. CS1 is a Cartesian Coordinate system containing X1, X2 and X3 axes as the source of the MCT instruction. CS2 is the joint coordinate system containing J1, J2 and J3 axes as the target of the MCT instruction

Data Flow When a Move is Executed with an MCT Instruction - Forward Transform



Input Data

CS2: DATA	SOURCE
Joint Positions (J1, J2, J3)	Machine Real Coordinate System
Link Lengths (L1, L2)	Coordinate System dialog
Base Offsets (X1b, X2b, X3b)	Coordinate System dialog
End Effector Offsets (X1e, X2e, X3e)	Coordinate System dialog
Zero Angle Orientations (Z1, Z2, Z3)	Coordinate System dialog
Orientations (Array [3])	Instruction Faceplate
Translations (Array [3])	Instruction Faceplate

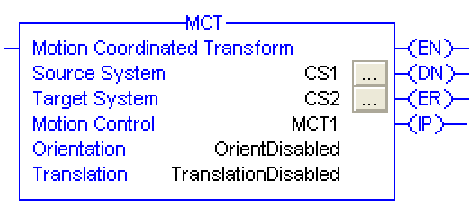
Active Instruction

MCT

Computed Output

CS1: Data	
Cartesian Positions (X1, X2, X3)	Destination
	Machine Virtual
	Coordinate System

Data Flow When a Move is Executed with an MCT Instruction - Inverse Transform



Input Data

CS1: DATA	SOURCE
Cartesian Positions (X1, X2, X3)	Machine Virtual Coordinate System
Link Lengths (L1, L2)	Coordinate System dialog
Base Offsets (X1b, X2b, X3b)	Coordinate System dialog
End Effector Offsets (X1e, X2e, X3e)	Coordinate System dialog
Zero Angle Orientations (Z1, Z2, Z3)	Coordinate System dialog
Orientations (Array [3])	Instruction Faceplate
Translations (Array [3])	Instruction Faceplate

Active Instruction

MCT

Computed Output

CS2: Data	Destination
Joint Positions (J1, J2, J3)	Machine Real
	Coordinate System

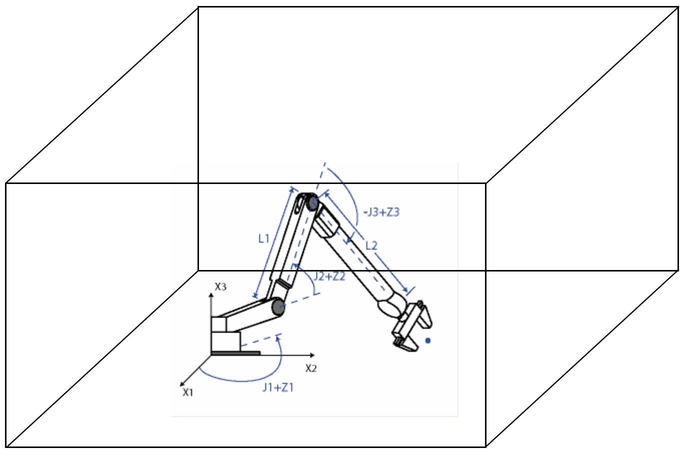
Programming Guidelines

Follow these guidelines to use an MCT instruction.

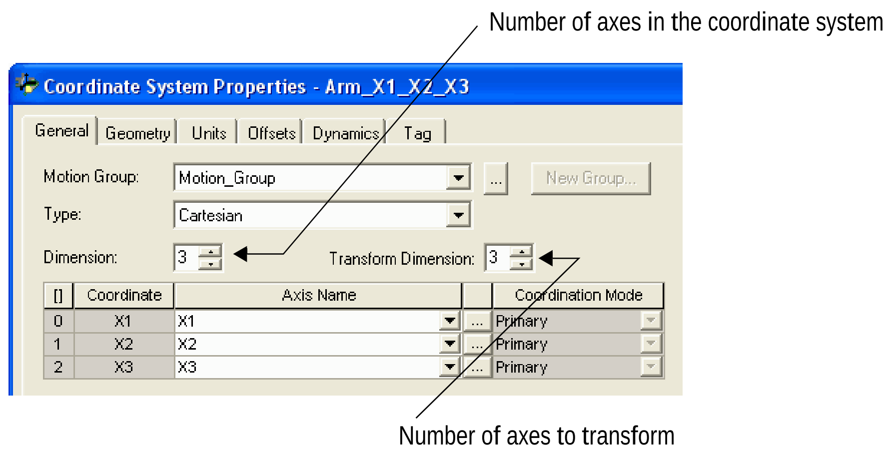
ATTENTION

Don't let the robot get fully stretched or fold back on itself. Otherwise it can start to move at a very high speed. In those positions, it loses its configuration as a left or right arm. When that happens, it can start to move at a very high speed.

Determine the working limits of the robot and keep it within those limits.



<table>
<tr><th>Guideline</th><th>Examples and notes</th></tr>
<tr><td>Set up a coordinate system of axes for the Cartesian positions of the robot. These axes are typically virtual.</td><td>



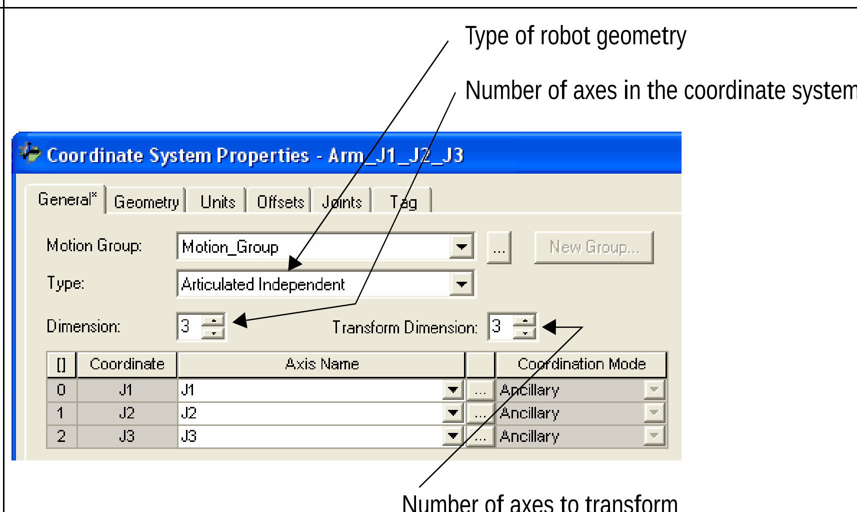
Important: You may see truncation error in the precision of computations. This happens when both of these conditions are true:

- The conversion constants of the virtual Cartesian axes in a transformation are small, such as 8000 counts/position unit.
- The link lengths of the non-Cartesian coordinate system are small, such as 0.5 inches.

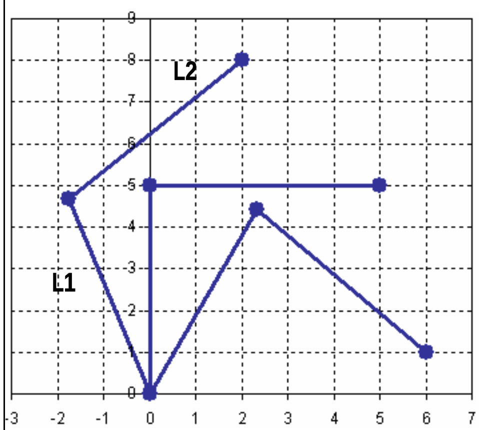
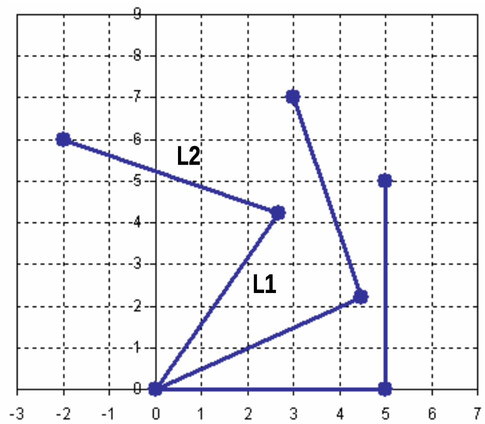
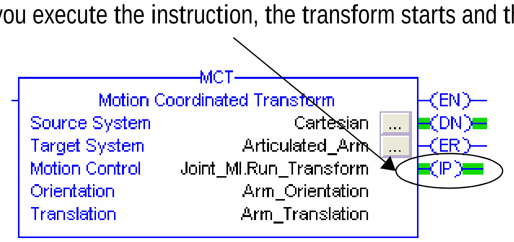
It's best to give large conversion constants to the virtual Cartesian axes in a transform, such as 100,000 or 1,000,000 counts/position unit. The maximum travel limit of the robot is

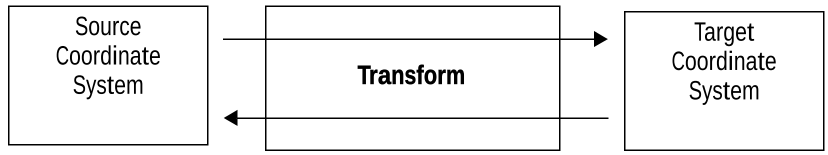
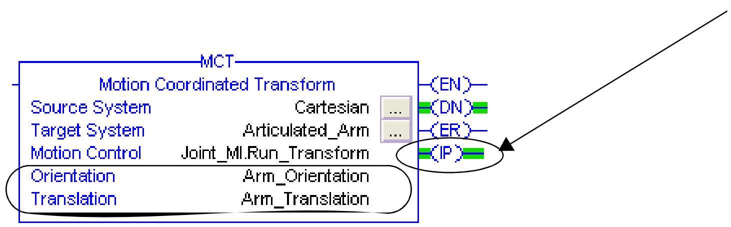
$$\frac{\pm 2^{31}}{\text{Conversion Constant}} \text{ Coordination Units}$$

</td></tr>
<tr><td>Set up another coordinate system for the actual joints of the robot.</td><td>



</td></tr>
</table>

Guideline	Examples and notes
Move the robot to a left- or right-arm starting position.	Do you want the robot to move like a left arm or a right arm?  Left arms  Right arms Before you start the transform, move the robot to a resting position that gives it the arm side that you want (left or right). Once you start the transform and initiate a Cartesian move in the Source coordinate system, the robot stays as a left arm or a right arm. If it starts as a left arm, it moves as a left arm. If it starts as a right arm, it moves as a right arm. You can always flip it from a left arm to a right arm or vice versa. To do that, move the joints directly.
Toggle the rung from false to true to execute the instruction.	This is a transitional instruction. In a ladder diagram, toggle the rung-condition-in from false to true each time you want to execute the instruction. When you execute the instruction, the transform starts and the IP bit turns on.  You can let the rung go false once you execute the instruction. The transform stays active.
In structured text, condition the instruction so that it only executes on a transition.	In structured text, instructions execute each time they are scanned. Condition the instruction so that it only executes on a transition. Use either of these methods: • Qualifier of an SFC action • Structured text construct
Start the transform before you start any motion.	You can't start a transform if any motion process is controlling an axis of the source or target coordinate systems. **Example:** Start the transform before you start gearing or camming.

Guideline	Examples and notes
Expect bi-directional motion between the source and target coordinate systems. Use an MCS instruction to cancel the transform.	A transform is bi-directional.  When you start the transform, the position of the source coordinate system changes to match the corresponding position of the target coordinate system. After that, if you move either system, the other system moves in response. The controller continues to control the axes even if you stop scanning the MCT instruction or its rung goes false. Use a Motion Coordinated Stop (MCS) instruction to stop the motion in the coordinate system, cancel the transform, or both.
Execute the MCT instruction again if you change the orientation or translation.	Suppose you change orientation or translation values after the transform is running.  In that case, execute the instruction again. To execute the instruction, toggle the rung-condition-in from false to true. Also execute the instruction again if you change the geometry of the equipment.

Arithmetic Status Flags not affected

Fault Conditions none

Error Codes See Error Codes (ERR) for Motion Instructions.

Extended Error Codes

Use Extended Error Codes (EXERR) for more information about an error.

ERR	EXERR	Corrective Action	Notes
61	1	Assign both coordinate systems to the motion group.	
	2	Check that you're using the correct source and target systems.	You can't use the same coordinate system as source and target.
	3	Set the transform dimension of the source system to the number of axes in the system, up to three.	
	4	Set the transform dimension of the target system to the number of axes in the system to be transformed, up to three.	
	5	Use a different source system.	You can only use one coordinate system as the source for one active transform.
	6	Use a different target system.	You can only use one coordinate system as the target for one active transform.
	7	Look for source or target axes that you're already using in another transform. Use different axes in the coordinate system.	You can only use an axis in one source system and one target system.
	8	Use a target system that isn't the source for this chain of transforms.	You can't create a circular chain of transforms that leads back to the original source.
	9	Check that you've assigned the correct axes to each coordinate system.	You can't use the same axes in the source and target systems.
	10	Stop all motion processes for all the axes in both systems (jog, move, gear, etc.).	You can't start the transform if any motion process is controlling a source or target axis.
	11	Insufficient resources available to initiate the transform connection.	
	12	Set the link lengths.	You can't use a link length of zero.
	13	Look for source or target axes that are in the shutdown state. Use a Motion Axis Shutdown Reset (MASR) instruction or direct command to reset the axes.	
	14	Uninhibit all the source or target axes.	

Changes to Status Bits

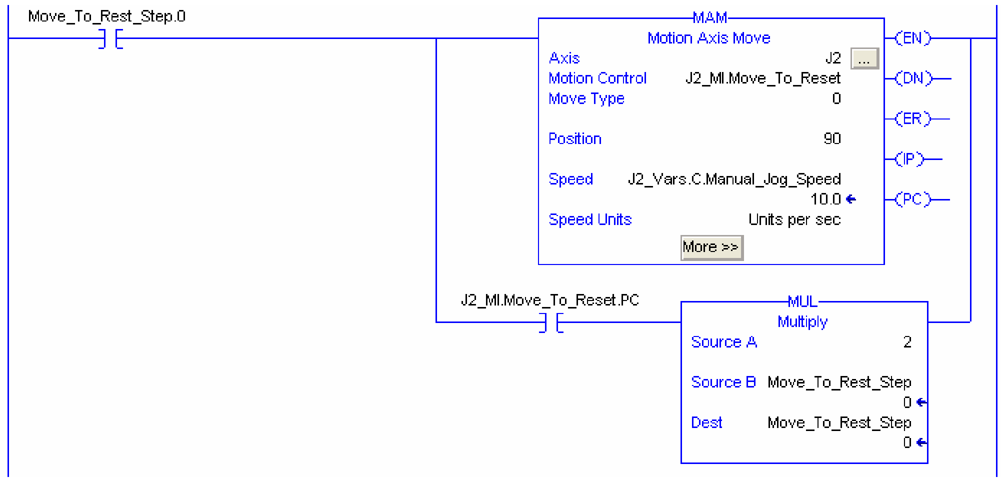
To see if	Check the tag for the	And this bit	For
A coordinate system is the source of an active transform.	Coordinate system	TransformSourceStatus	On
A coordinate system is the target of an active transform.	Coordinate system	TransformTargetStatus	On
An axis is part of an active transform.	Axis	TransformStateStatus	On
An axis is moving because of a transform.	Axis	ControlledByTransformStatus	On

Example 1 *Pick and Place—Ladder Diagram*

1. Move to rest routine

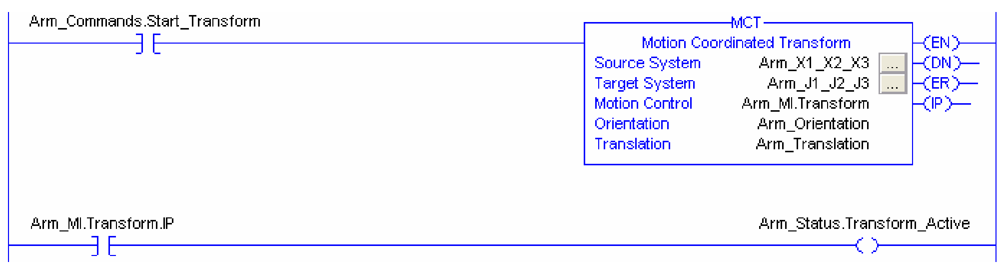
This routine is a sequence of moves that put an articulated independent robot in an at-rest position at the desired left or right arm angles.

When Move_To_Rest_Step.0 turns on, axis J2 moves to 90°. Then the sequence goes to the next step.



2. Start transform routine

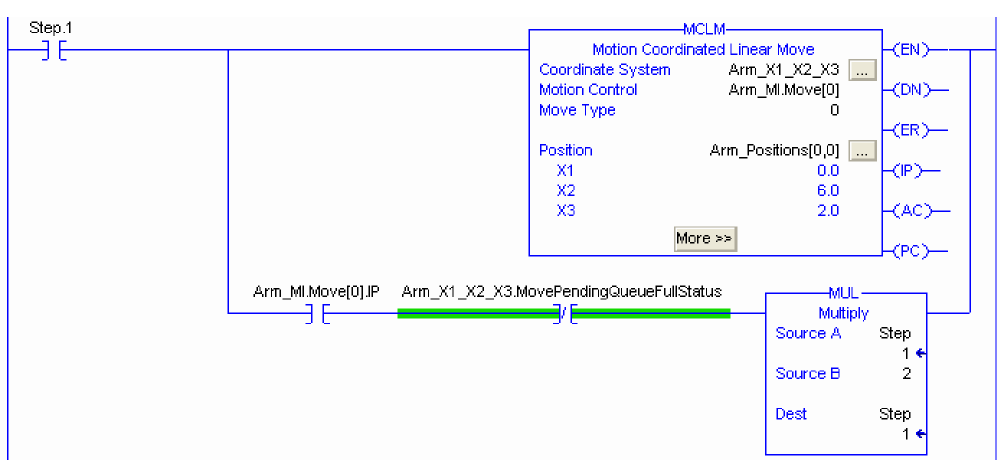
When Arm_Commands.Start_Transform turns on, the transform starts. The IP bit signals that the transform is running.



3. Pick and place routine

This routine is one in a sequence of MCLM instructions that move the Cartesian system. The joints of the robot follow the moves.

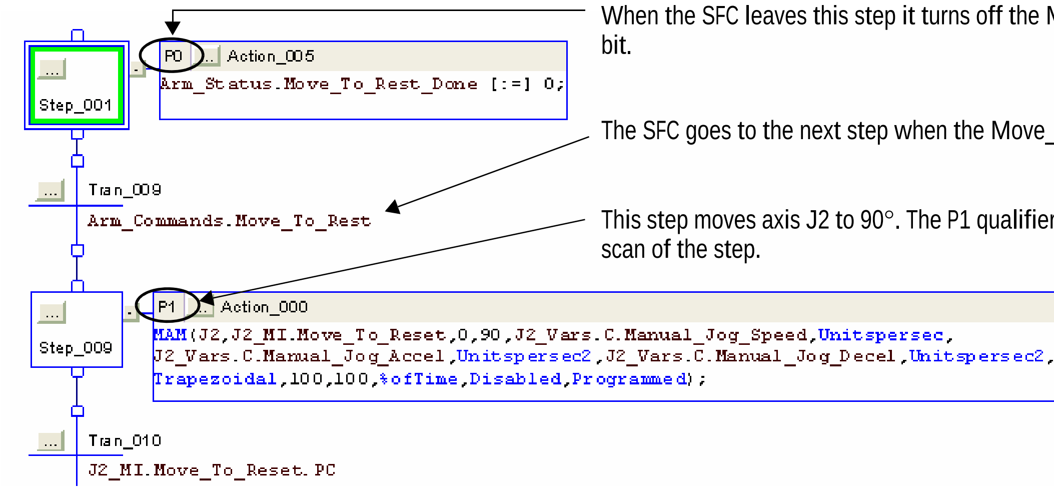
When Step.1 turns on, the coordinate system moves to 0, 6, 2. When the move is in process (IP), the sequence queues the next move.



Pick and Place—Structured Text

1. Move to rest routine

This routine is a sequence of moves that put the robot in an at-rest position at the desired left or right arm angles.

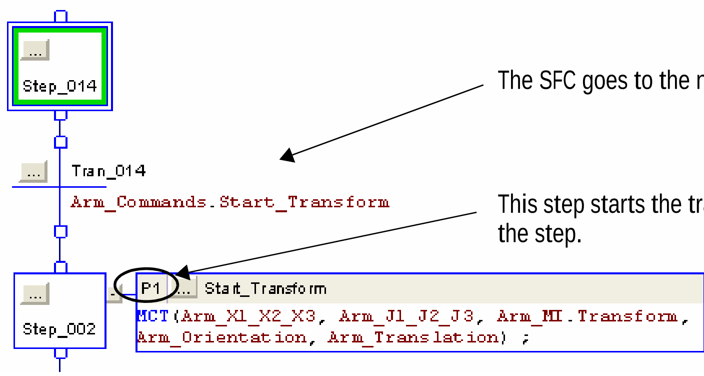


When the SFC leaves this step it turns off the Move_To_Reset_Done bit.

The SFC goes to the next step when the Move_To_Rest bit turns on.

This step moves axis J2 to 90°. The P1 qualifier limits this to the first scan of the step.

2. Start transform routine

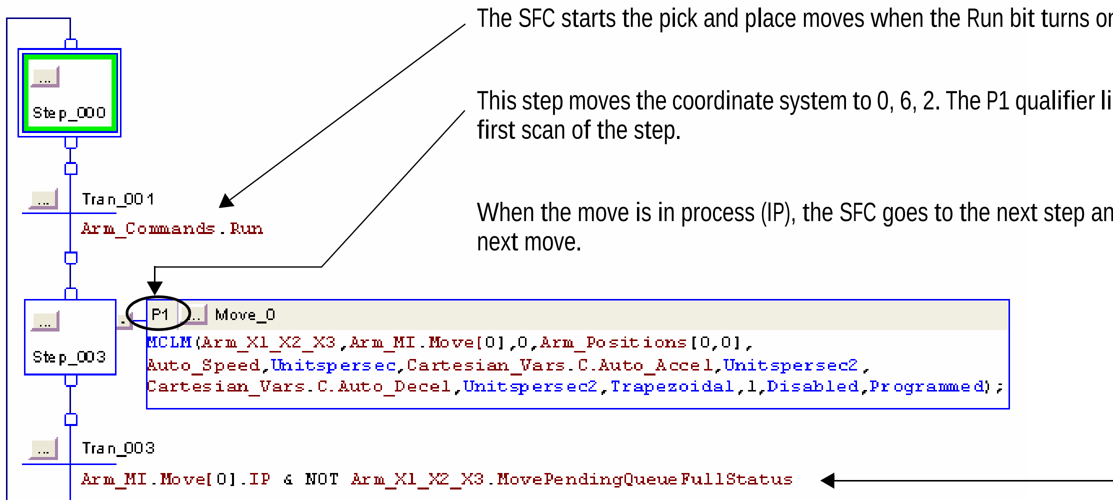


The SFC goes to the next step when the Move_To_Rest bit turns on.

This step starts the transform. The P1 qualifier limits this to the first scan of the step.

3. Pick and place routine

This routine is one in a sequence of MCLM instructions that move the Cartesian system. The joints of the robot follow the moves.



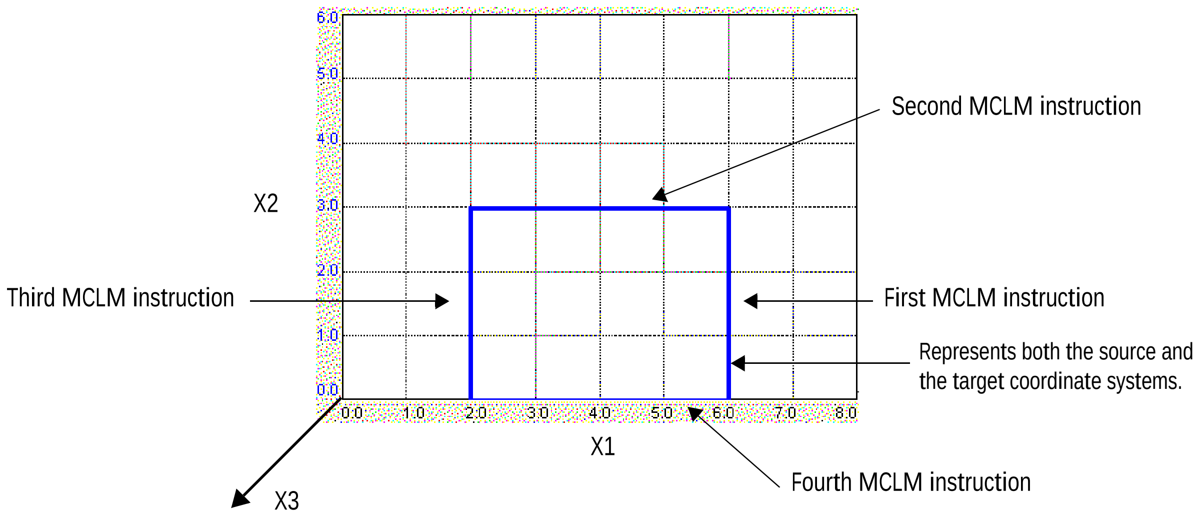
The SFC starts the pick and place moves when the Run bit turns on.

This step moves the coordinate system to 0, 6, 2. The P1 qualifier limits this to the first scan of the step.

When the move is in process (IP), the SFC goes to the next step and queues the next move.

Example 2 *Change Orientation*

Suppose you want to move the target coordinate system in a rectangular path. In that case, execute the MCT instruction to start the transform. Then, execute four Motion Coordinated Linear Move (MCLM) instructions to produce the rectangular path.

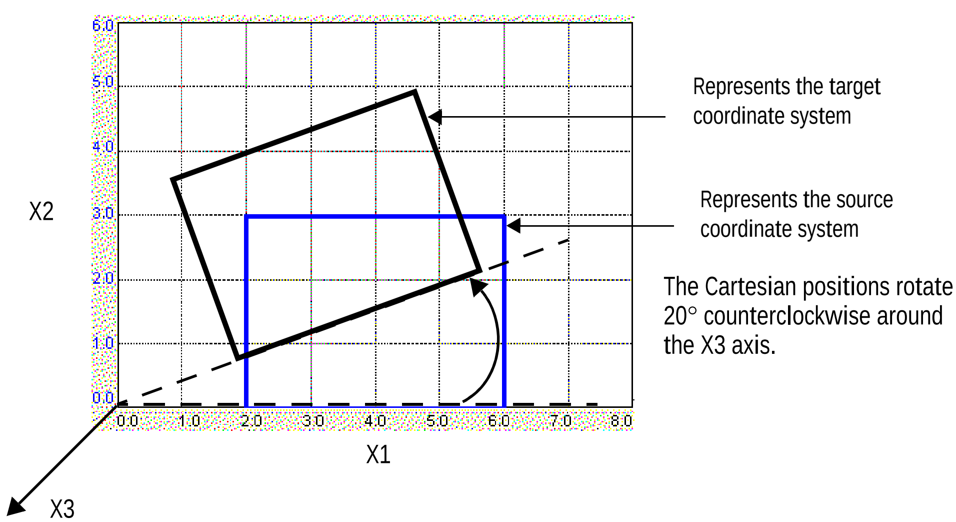


Now suppose you want to rotate the Cartesian positions of the target coordinate system by 20° counterclockwise around the X3 axis. In that case:

1. Enter orientation values of 0°, 0°, 20° into the MCT instruction.

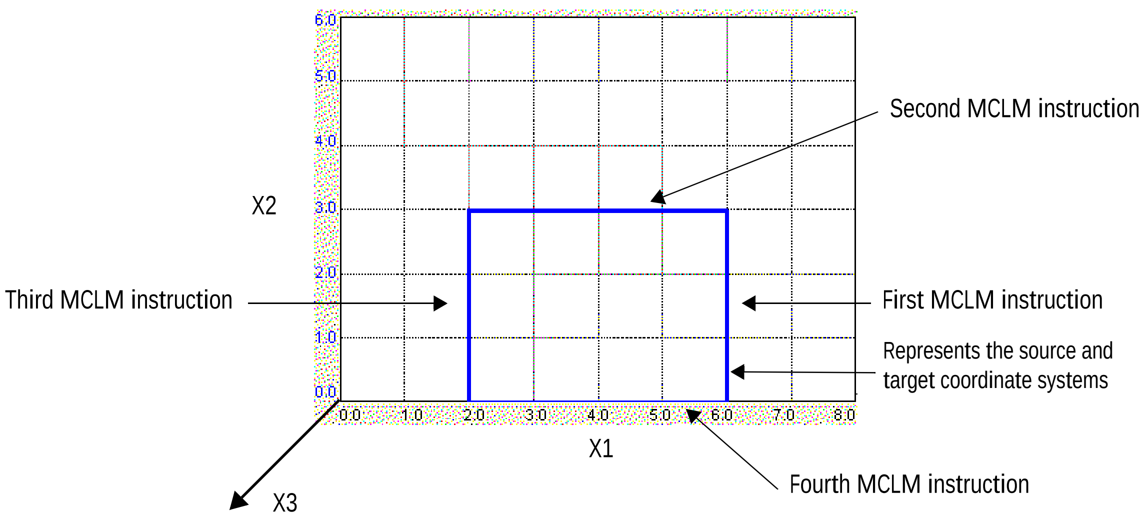
2. Execute the MCT instruction again to apply the orientation to the transform.

3. Execute the same four MCLM instructions again.



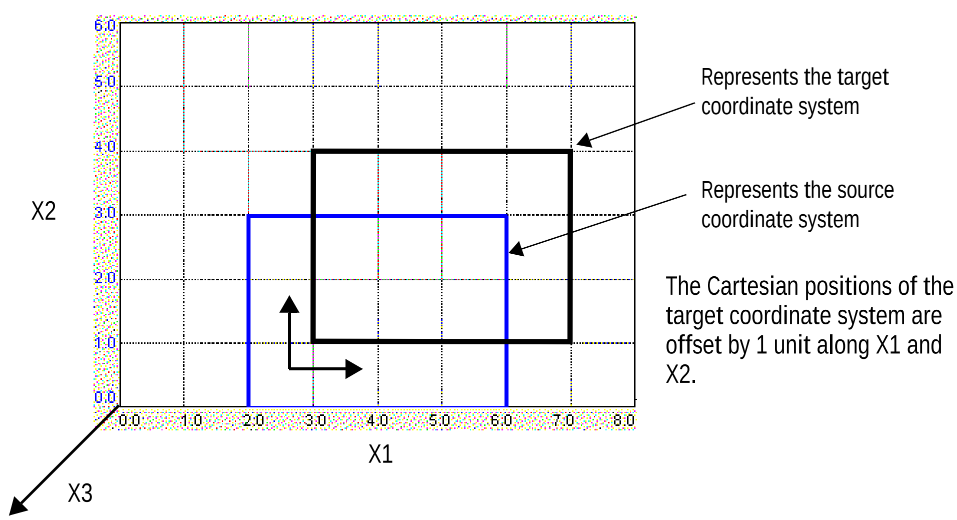
Example 3 *Change Translation*

Suppose you want to move the target coordinate system in a rectangular path. In that case, execute the MCT instruction to start the transform. Then, execute four Motion Coordinated Linear Move (MCLM) instructions to produce the rectangular path.



Now suppose you want to offset the Cartesian positions of the target coordinate system by 1 unit along both the X1 and X2 axes. In that case:

1. Enter translation values of 1, 1, 0 into the MCT instruction.
2. Execute the MCT instruction again to apply the translation to the transform.
3. Execute the same four MCLM instructions again.



Motion Calculate Transform Position (MCTP)

Use the MCTP instruction to calculate the position of a point in one coordinate system to the equivalent point in a second coordinate system.

ATTENTION

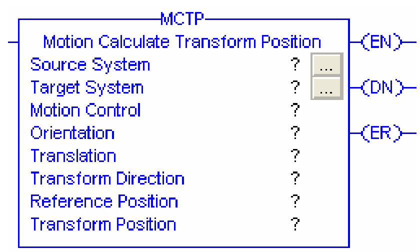
Use a motion control tag only once. Do not reuse it in another instruction. Otherwise, you can cause unexpected equipment motion and injury to people.

IMPORTANT

You can use this instruction only with 1756-L6x controllers.

Operands

Ladder Diagram



<table>
<tr><th>Operand</th><th>Type</th><th>Format</th><th colspan="2">Description</th></tr>
<tr><td>Source System</td><td>COORDINATE_SYSTEM</td><td>Tag</td><td colspan="2">Cartesian coordinate system for Cartesian positions of the robot</td></tr>
<tr><td>Target System</td><td>COORDINATE_SYSTEM</td><td>Tag</td><td colspan="2">Non-Cartesian coordinate system that controls the actual equipment</td></tr>
<tr><td>Motion Control</td><td>MOTION_INSTRUCTION</td><td>Tag</td><td colspan="2">Control tag for the instruction</td></tr>
<tr><td rowspan="5">Orientation</td><td rowspan="5">REAL[3]</td><td rowspan="5">Array</td><td colspan="2">Do you want to rotate the target position around the X1, X2, or X3 axis?</td></tr>
<tr><td>If</td><td>Then</td></tr>
<tr><td>No</td><td>Leave the array values at zero.</td></tr>
<tr><td>Yes</td><td>Enter the degrees of rotation into the array. Put the degrees of rotation around X1 in the first element of the array, and so on.</td></tr>
<tr><td colspan="2">Use an array of three REALs even if a coordinate system has only one or two axes.</td></tr>
<tr><td rowspan="5">Translation</td><td rowspan="5">REAL[3]</td><td rowspan="5">Array</td><td colspan="2">Do you want to offset the target position along the X1, X2, or X3 axis?</td></tr>
<tr><td>If</td><td>Then</td></tr>
<tr><td>No</td><td>Leave the array values at zero.</td></tr>
<tr><td>Yes</td><td>Enter the offset distances into the array. Enter the offset distances in coordination units. Put the offset distance for X1 in the first element of the array, and so on.</td></tr>
<tr><td colspan="2">Use an array of three REALs even if a coordinate system has only one or two axes.</td></tr>
</table>

Operand	Type	Format	Description			
Transform Direction	DINT	Immediate	**To calculate**	**With the base turned to the**	**And the robot as a**	**Choose**
			Cartesian position	→	→	Forward
			Joint angles	Same X1-X2 quadrant as the point	Right arm	Inverse Right Arm
					Left arm	Inverse Left Arm
				Opposite X1-X2 quadrant from the point	Right arm	Inverse Right Arm Mirror
					Left arm	Inverse Left Arm Mirror
Reference Position	REAL[3]	Array	**If the transform direction is**		**Then enter an array that has the**	
			Forward		Joint angles	
			Inverse		Cartesian positions	
Transform Position	REAL[3]	Array	Array that stores the calculated position			

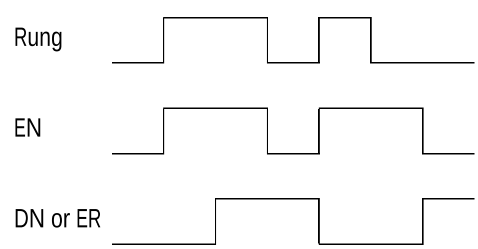
Structured Text

```
MCTP(Source System, Target
System, Motion Control,
Orientation, Translation,
Transform Direction,
Reference Position,
Transform Position);
```

The structured text operands are the same as the ladder diagram operands. Enter the transform direction without spaces.

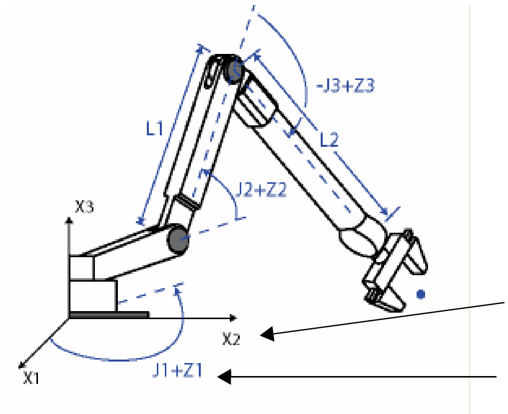
Example: Enter a transform direction of Inverse Left Arm as InverseLeftArm.

MOTION_INSTRUCTION Data Type

To see if	Check if this bit is on	Data type	Notes
The rung is true.	EN	BOOL	Sometimes the EN bit stays on even if the rung goes false. This happens if the rung goes false before the instruction is done or errored. 
The instruction is done.	DN	BOOL	
An error happened.	ER	BOOL	Identify the error number listed in the error code field of the Motion control tag then, refer to Error Codes (ERR) for Motion Instructions on page 383 of this manual.

Description

Use the MCTP instruction to calculate the position of a point in one coordinate system to the equivalent point in a second coordinate system.



You can give the instruction the X1, X2, and X3 positions and get the corresponding J1, J2, and J3 angles.

Or you can give the instruction the J1, J2, and J3 angles and get the corresponding X1, X2, and X3 positions.

The MCTP instruction is similar to the MCT instruction except the MCTP instruction doesn't start a transform. It calculates a position once each time you execute it.

Programming Guidelines

Follow these guidelines to use an MCTP instruction.

Guideline	Examples and notes
Toggle the rung from false to true to execute the instruction.	This is a transitional instruction. In a ladder diagram, toggle the rung-condition-in from false to true each time you want to execute the instruction.
In structured text, condition the instruction so that it only executes on a transition.	In structured text, instructions execute each time they are scanned. Condition the instruction so that it only executes on a transition. Use either of these methods: • Qualifier of an SFC action • Structured text construct

Arithmetic Status Flags

not affected

Fault Conditions

none

Error Codes

See Error Codes (ERR) for Motion Instructions.

Extended Error Codes

none

Changes to Status Bits

none

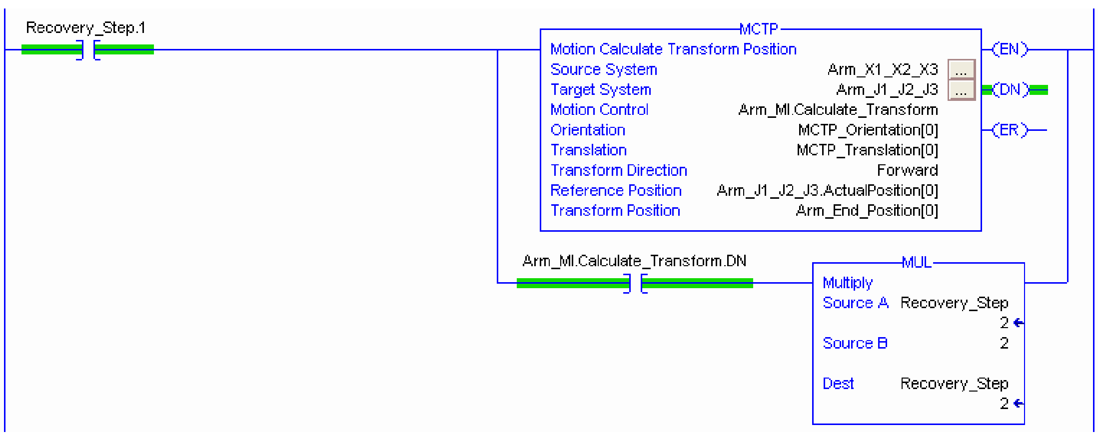
Example 1 Suppose you want to write a recovery sequence for faults. As one of your steps, you want to get the current position of an articulated independent robot. In that case, you can use an MCTP instruction to calculate the robot's Cartesian position when you know its joint angles.

Calculate Position—Ladder Diagram

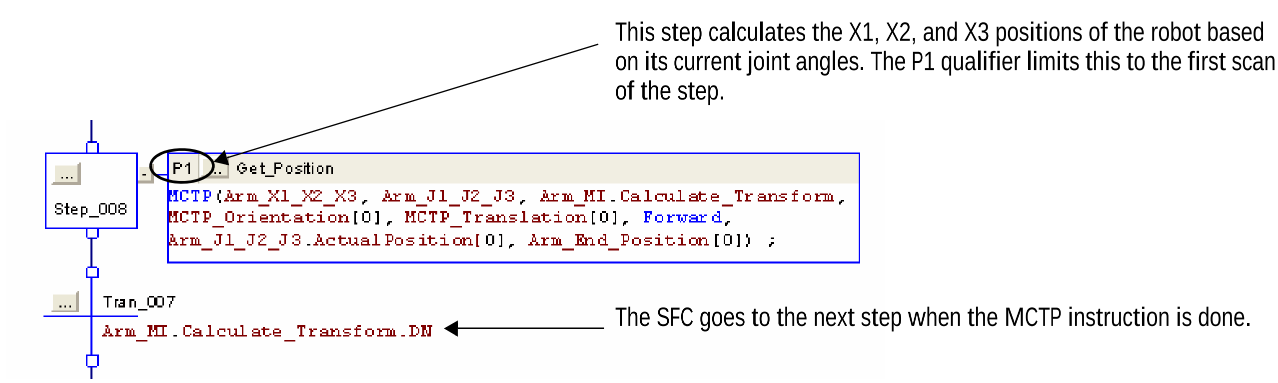
If Recovery_Step.1 turns on, then

Calculate the X1, X2, and X3 positions of the robot based on its current joint angles

When the instruction is done, the MUL instruction takes the sequence to the next step.

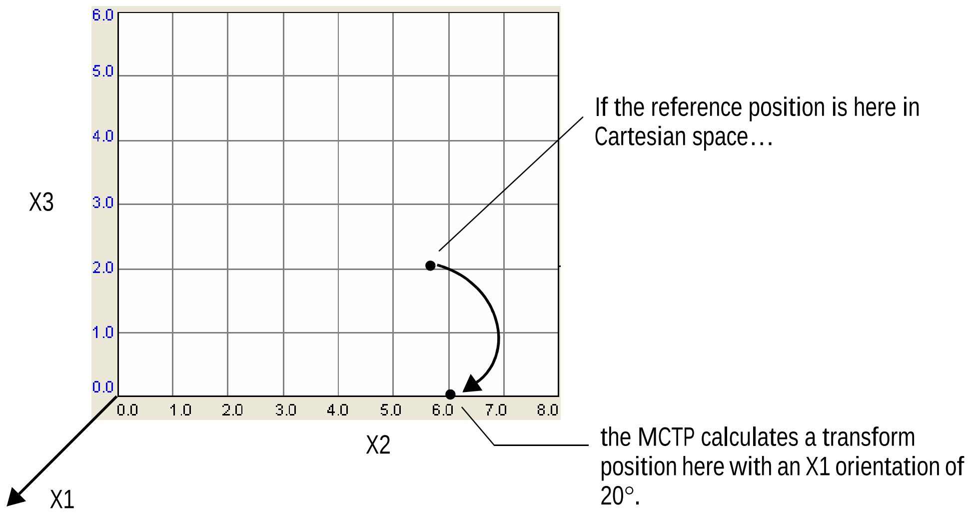


Calculate Position—Structured Text



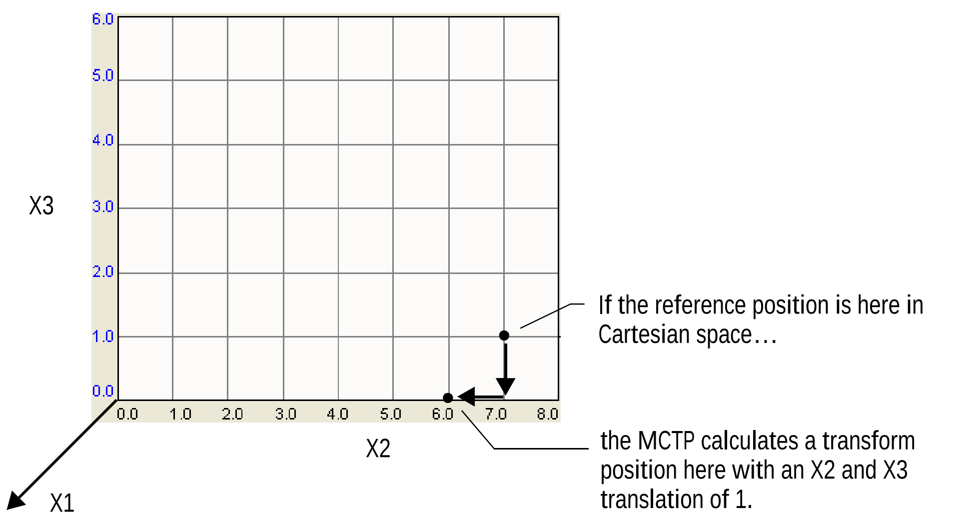
Example 2 *Change Orientation*

Suppose you enter orientation values of 20°, 0°, 0° into example 1. In that example, the MCTP instruction does a forward transform.



Example 3 *Change Translation*

Suppose you enter translation values of 0, 1, 1 into example 1. In that example, the MCTP instruction does a forward transform.



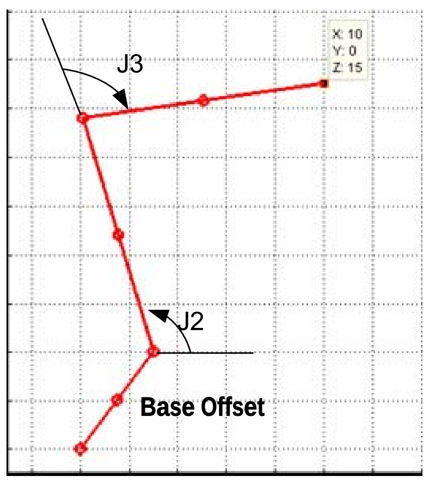
Example 4 *Transform Direction*

If your robot has base offsets, there can be up to four different ways to get to a given point. Suppose your robot has this geometry:

- L1 = 10
- L2 = 10
- X1b = 3.0
- X3b = 4.0

This example shows the ways to get a position of X1 = 10, X2 = 0, and X3 = 15.

Inverse left arm

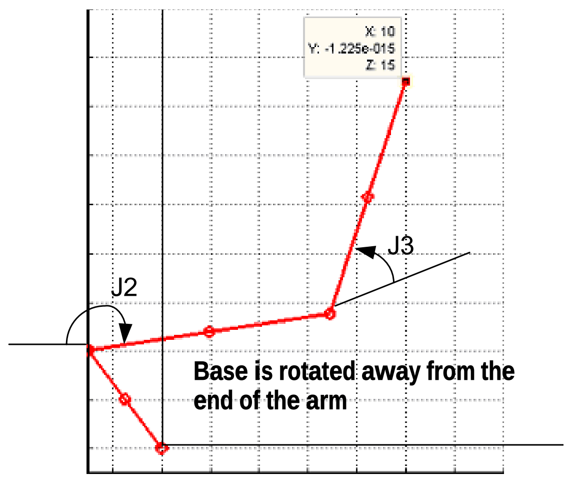


J1 = 0

J2 = 106.84

J3 = -98.63

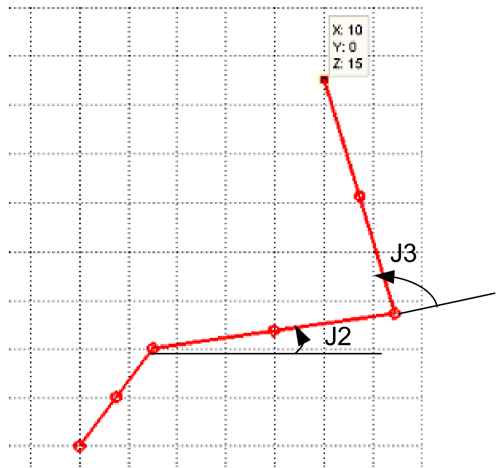
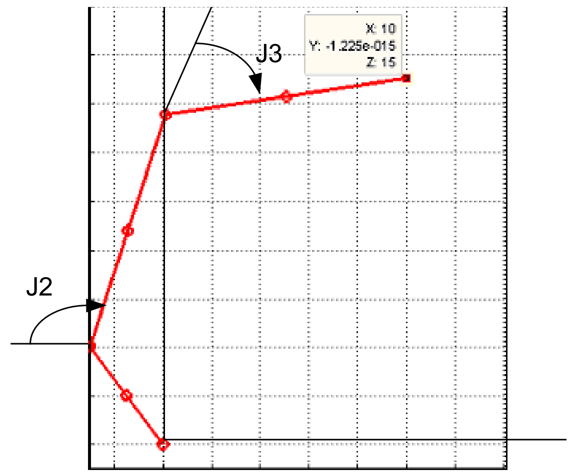
Inverse left arm mirror



J1 = 180

J2 = 171.39

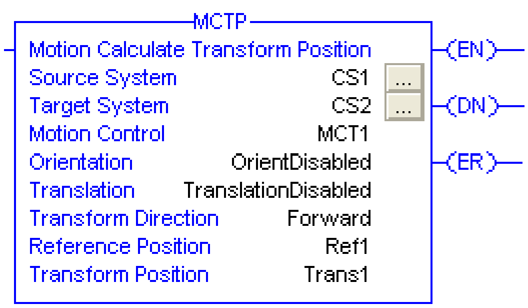
J3 = -63.26

Inverse right arm	Inverse right arm mirror
 J1 = 0 J2 = 8.22 J3 = 98.63	 J1 = 180 J2 = 108.14 J3 = 63.26

Data Flow of MCTP Instruction Between Two Coordinate Systems

The following illustrations show the flow of data when an MCTP Instruction is executed to perform a forward transformation and an inverse transformation. The CS1 indicator represents a Cartesian Coordinate system containing X1, X2 and X3 axes as the source of the MCTP instruction. The CS2 indicator represents the joint coordinate system containing J1, J2 and J3 axes as the target of the MCTP instruction.

Data Flow When a Move is Executed with an MCTP Instruction - Forward Transform



Input Data

CS2: DATA	SOURCE
Link Lengths (L1, L2)	Coordinate System dialog
Base Offsets (X1b, X2b, X3b)	Coordinate System dialog
End Effector Offsets (X1e, X2e, X3e)	Coordinate System dialog
Zero Angle Orientations (Z1, Z2, Z3)	Coordinate System dialog
Orientation (Array [3])	Instruction Faceplate
Translations (Array [3])	Instruction Faceplate
Transform Direction	Instruction Faceplate
Reference Position Typically Joint - Target	Instruction Faceplate

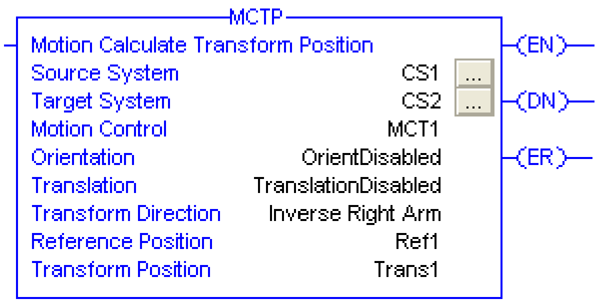
Executed Instruction

MCTP

Computed Output

CS1: Data	Destination
Cartesian Positions (X1, X2, X3) Typically Cartesian	Instruction Faceplate Transform Position

Data Flow When a Move is Executed with an MCTP Instruction - Inverse Transform



Input Data

CS1: DATA	SOURCE
Link Lengths (L1, L2)	Coordinate System dialog
Base Offsets (X1b, X2b, X3b)	Coordinate System dialog
End Effector Offsets (X1e, X2e, X3e)	Coordinate System dialog
Zero Angle Orientations (Z1, Z2, Z3)	Coordinate System dialog
Orientation (Array [3])	Instruction Faceplate
Translations (Array [3])	Instruction Faceplate
Transform Direction	Instruction Faceplate
Reference Position Typically Cartesian - Source	Instruction Faceplate

Executed Instruction

MCTP

Computed Output

CS2: Data	Destination
Joint Positions (J1, J2, J3) Typically Joint	Instruction Faceplate Transform Position

Motion Coordinated Shutdown Reset (MCSR)

Use the Motion Coordinated Shutdown Reset (MCSR) instruction to reset all axes in a coordinate system. The MCSR instruction resets the axes from a shutdown state to an axis ready state.

ATTENTION

Use a motion control tag only once. Do not re-use it in another instruction. Otherwise, you can cause unexpected equipment motion and injure people.

Operands: *Relay Ladder*

Operand	Type	Format	Description
Coordinate System	COORDINATE_SYSTEM	tag	Name of the axis, which provides the position input to the Output Cam. Ellipsis launches Axis Properties dialog.
Motion Control	MOTION_INSTRUCTION	tag	Structure used to access instruction status parameters.

Structured Text

```
MCSR(CoordinateSystem,
MotionControl);
```

The operands are the same as those for the relay ladder MCSR instruction.

Description: The Motion Coordinated Shutdown Reset (MCSR) instruction initiates a reset of all axes within a specified coordinate system from a shutdown state to an axis ready state. MCSR also clears any axis faults.

Coordinate System

The Coordinate System operand specifies the set of motion axes that define the dimensions of a Cartesian coordinate system. For this release the coordinate system supports up to three (3) primary axes. Only the axes configured as primary axes (up to 3) are included in the coordinate velocity calculations.

Motion Control

The following control bits are affected by the MCSR instruction.

Mnemonic	Description
.EN (Enable) Bit 31	The Enable Bit sets when the rung transitions from false to true. It resets when the rung goes from true to false.
.DN (Done) Bit 29	The Done Bit sets when the coordinated shutdown reset is successfully initiated. It resets when the rung transitions from true to false.
.ER (Error) Bit 28	The Error Bit sets when the reset of the coordinated shutdown fails to initiate. It resets when the rung transitions from false to true.

This is a transitional instruction:

- In relay ladder, toggle the rung-condition-in from cleared to set each time the instruction should execute.
- In structured text, condition the instruction so that it only executes on a transition. See Appendix C.

Arithmetic Status Flags: not affected

Fault Conditions: none

Error Codes: See Error Codes (ERR) for Motion Instructions.

MCSR Changes to Status Bits: Status Bits provide a means for monitoring the progress of the motion instruction. There are three types of Status Bits that provide pertinent information. They are: Axis Status Bits, Coordinate System Status Bits, and Coordinate Motion Status Bits. When the MCS instruction initiates, the status bits undergo the following changes.

Axis Status Bits

Bit Name	Effect
CoordinatedMoveStatus	No effect.

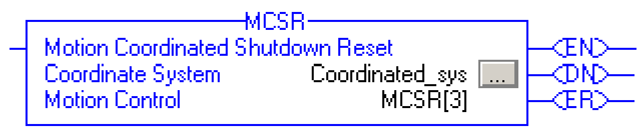
Coordinate System Status Bits

Bit Name	Effect
ShutdownStatus	Clears the Shutdown status bit.

Coordinate Motion Status Bits

Bit Name	Effect
MovePendingStatus	Flushes instruction queue and clears status bit.
MovePendingQueueFullStatus	Flushes instruction queue and clears status bit.

Example: Relay Ladder



MCSR Ladder Instruction

Structured Text

```
MCSR(Coordinated_sys,MCSR[3]);
```

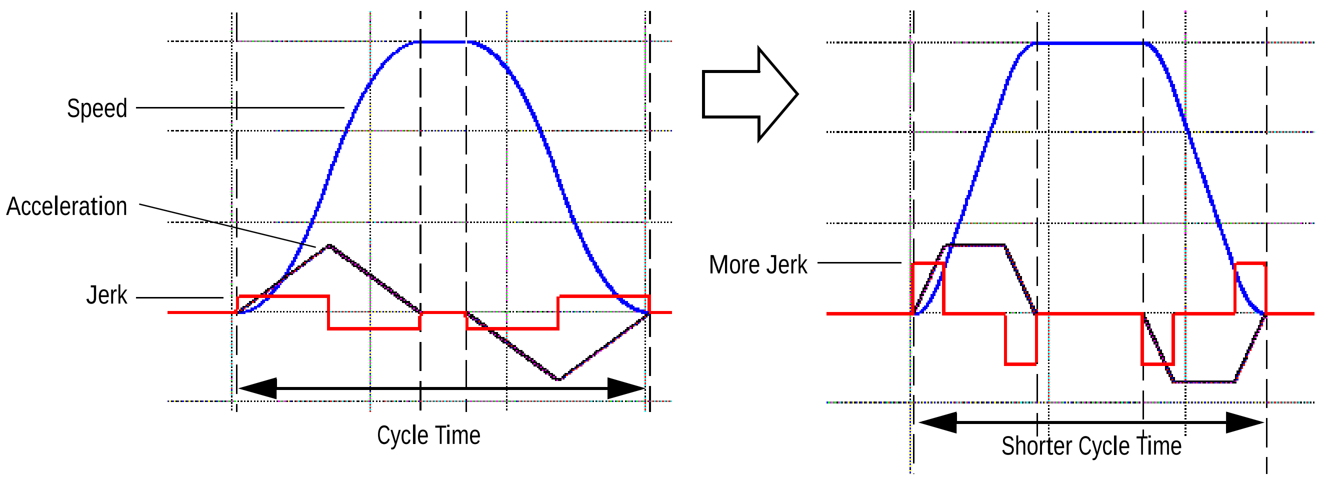
Tune an S-curve Profile

Introduction

Use this procedure to balance the smoothness and cycle time of motion that uses an S-curve profile.

Do This When

Do this procedure when you want to decrease the cycle time of an S-curve motion profile but keep some of the profile's smoothness.



To use this procedure, your application must meet these requirements:

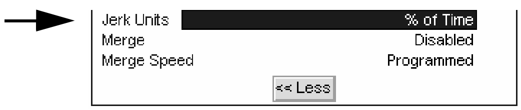
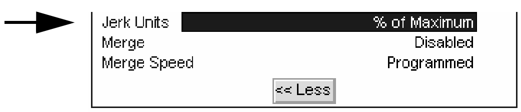
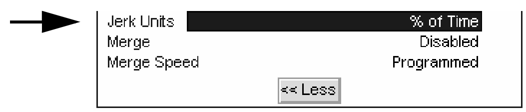
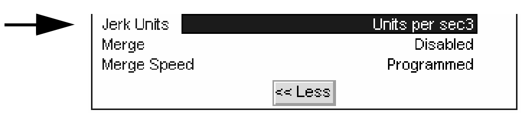
- The controller is at revision 16 or later.
- One of these instructions produce the motion:
  - Motion Axis Move (MAM)
  - Motion Axis Jog (MAJ)
  - Motion Axis Stop (MAS)
- The instruction uses an S-curve profile

Before You Begin

IMPORTANT	In this procedure, you increase the jerk. This increases the stress on the equipment and load. Make sure you can identify when the equipment or load has reached its jerk limit.

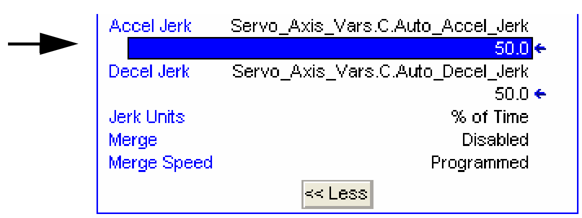
Procedure

1. Are the Jerk Units set to % of Time?

If the Jerk Units are	Then
% of Time 	Continue with step 2.
% of Maximum 	A. Change the Jerk Units to % of Time. 
Units per sec3 	B. Continue with step 2.

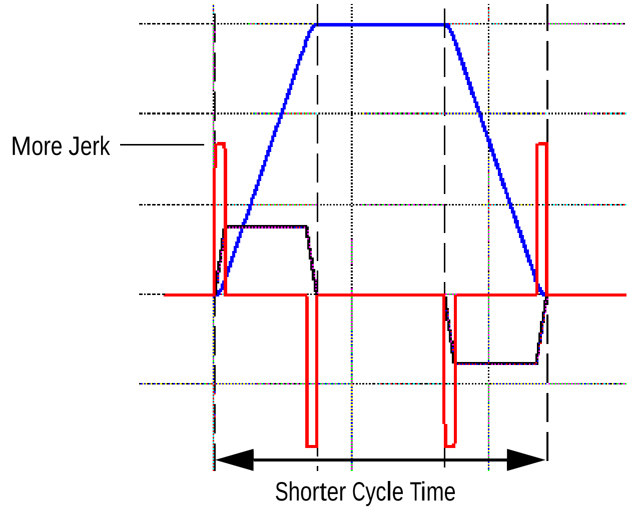
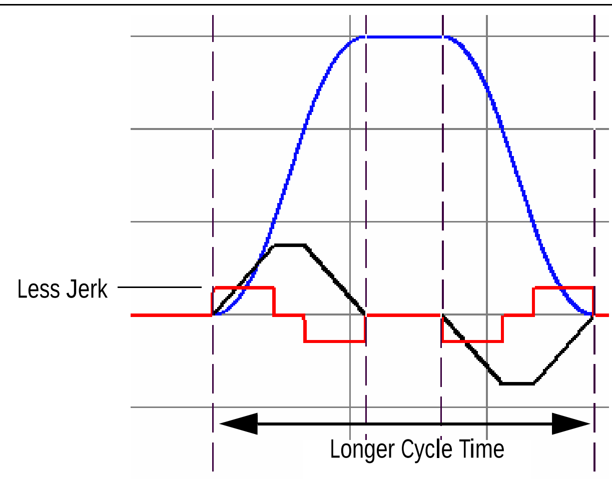
2. Set the Jerk values to 50% of Time.

   Example



3. Test your equipment and observe its jerk.

4. Adjust the Jerk values.

If there Is	Then	Which results In
NOT too much jerk	Reduce the % of Time.	
Too much jerk	Increase the % of Time.	

5. Repeat steps 3 and 4 until you have the desired balance between smoothness and cycle time.

Additional Resources

- Program a Velocity Profile on page 22
- Troubleshoot Axis Motion on page 367

Notes:

Troubleshoot Axis Motion

Introduction

Use this information to troubleshoot some situations that could happen while you are running an axis.

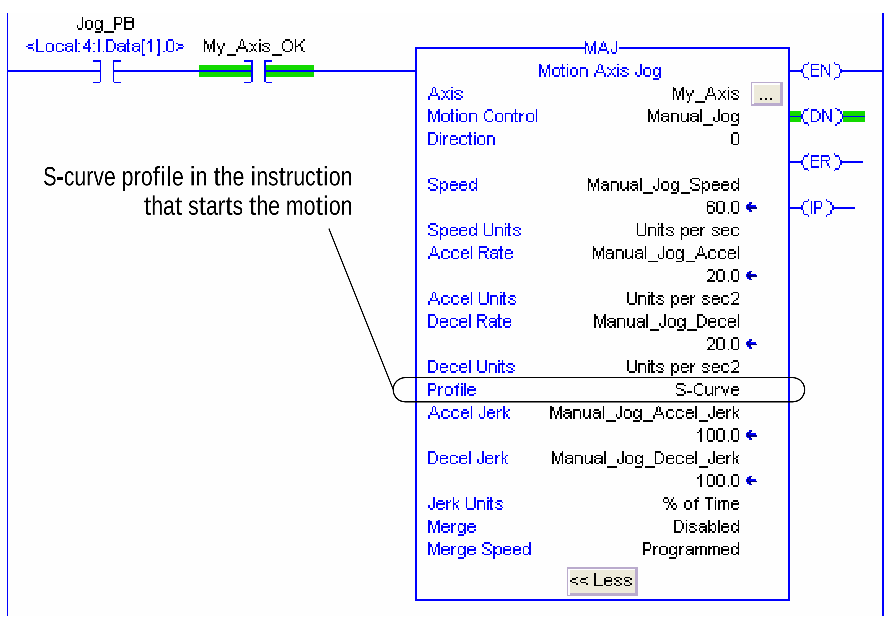
Why does my axis accelerate when I stop it?

While an axis is accelerating, you try to stop it. The axis keeps accelerating for a short time before it starts to decelerate.

Example

You start a Motion Axis Jog (MAJ) instruction. Before the axis gets to its target speed, you start a Motion Axis Stop (MAS) instruction. The axis continues to speed up and then eventually slows to a stop.

Look for



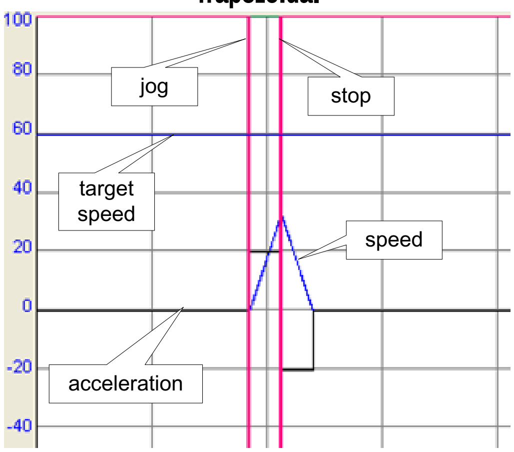
Cause When you use an S-curve profile, jerk determines how fast an axis can change its acceleration and deceleration.

- An S-curve profile has to get acceleration to zero before the axis can slow down.
- The time it takes depends on the jerk, acceleration, and speed.
- In the meantime, the axis continues to speed up.

The following trends show how the axis stops with a trapezoidal profile and an S-curve profile.

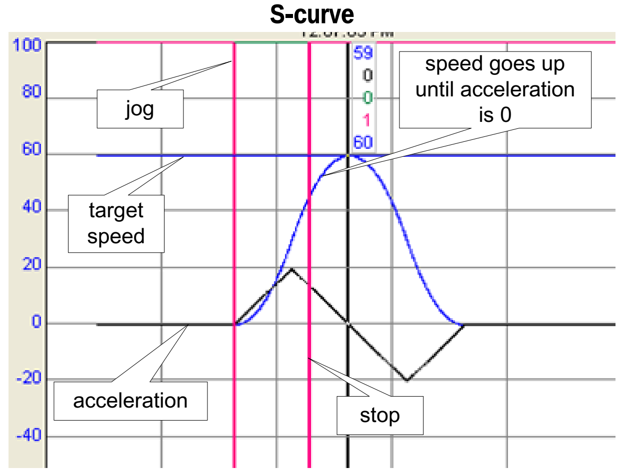
Stop while accelerating

Trapezoidal



The axis slows down as soon as you start the stopping instruction.

S-curve



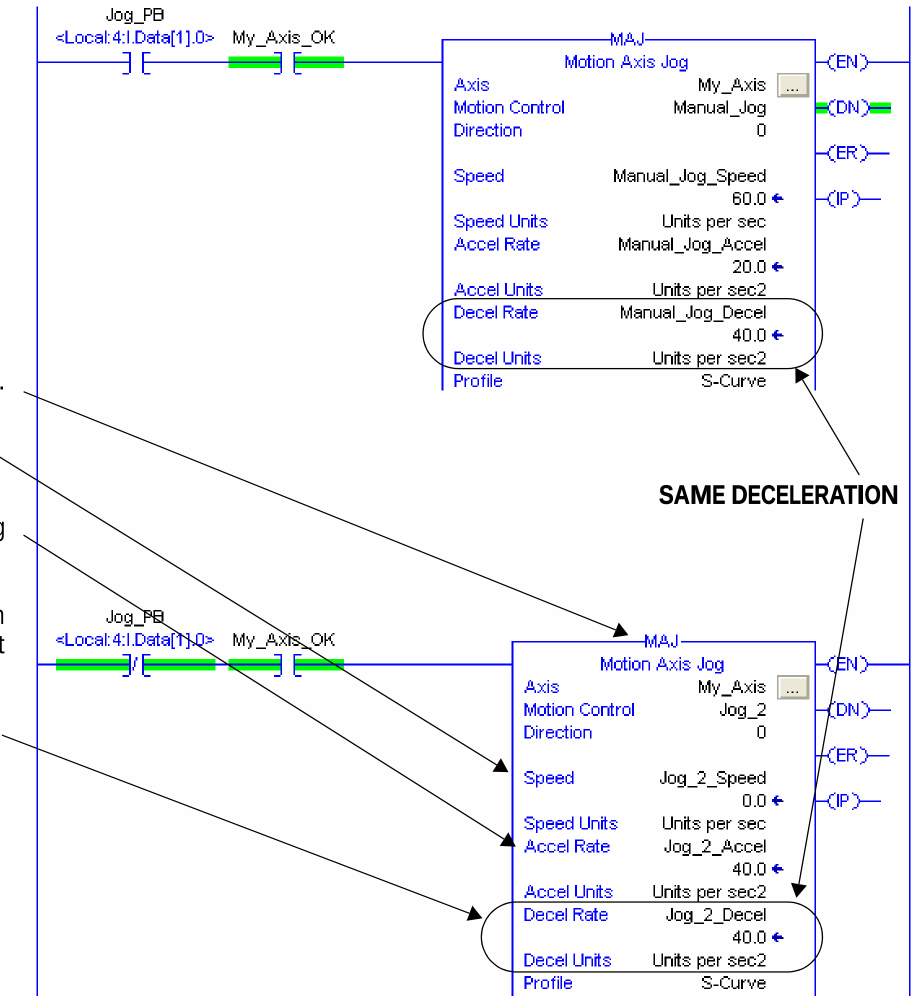
The axis continues to speed up until the S-curve profile brings the acceleration to zero.

Corrective action *Revision 15 or Earlier*

1. Use an MAJ instruction to stop the axis.
2. Set the speed of the stopping MAJ to zero.
3. Use a higher acceleration in the stopping MAJ.

   Reason: This increases the acceleration jerk. The axis can begin to stop sooner at the higher acceleration jerk.
4. Use a deceleration that gives you the response you want without too much jerk.

   Important: Use the **same** deceleration in both instructions. Otherwise the axis could reverse directions when you go from stopping to starting again.



Revision 16 or Later

IMPORTANT Leave bit 0 of the DynamicsConfigurationBits attribute for the axis turned ON. Otherwise this corrective action won't work.

See the RSLogix 5000 online help for more information.

Help > Contents > GSV/SSV Objects > Axis > Dynamics Configuration Bits

Revision 16 and later lets you increase the deceleration jerk of an MAS instruction to get a quicker stop.

If the Jerk Units are	Then make this change to the Decel Jerk
% of Time	Reduce the % of Time
% of Maximum	Increase % of Maximum
Units per sec3	Increase Units per sec3

Why does my axis overshoot its target speed?

While an axis is accelerating, you try to stop the axis or change its speed. The axis keeps accelerating and goes past its initial target speed. Eventually it starts to decelerate.

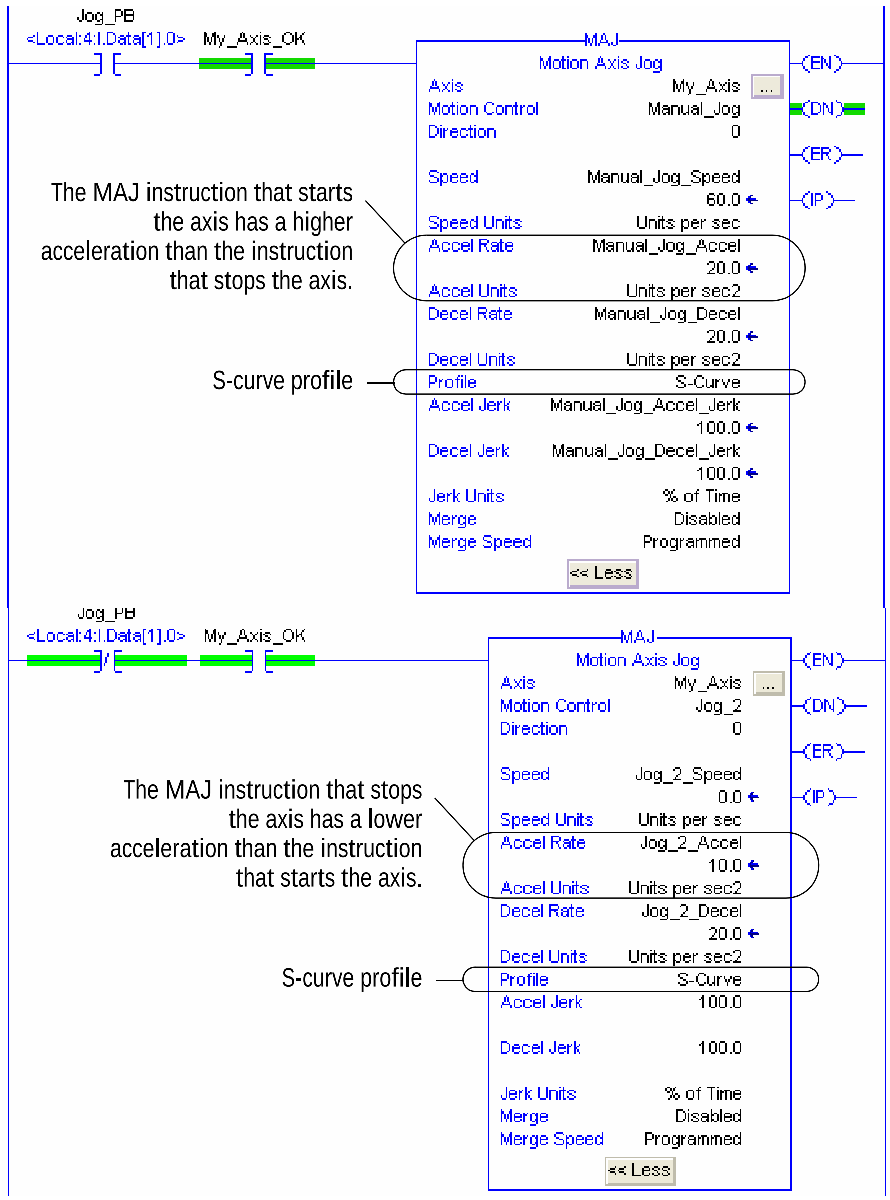
IMPORTANT Revision 16 improved how the controller handles changes to an S-curve profile.

See the RSLogix 5000 online help for more information.

Help > Contents > GSV/SSV Objects > Axis > Dynamics Configuration Bits

Example You start a Motion Axis Jog (MAJ) instruction. Before the axis gets to its target speed, you try to stop it with another MAJ instruction. The speed of the second instruction is set to zero. The axis continues to speed up and overshoots its initial target speed. Eventually it slows to a stop.

Look for



Jog_PB
<Local:4:I.Data[1].0>
My_Axis_OK
MAJ
Motion Axis Jog
Axis My_Axis
Motion Control Manual_Jog
Direction 0
Speed Manual_Jog_Speed
60.0
Speed Units Units per sec
Accel Rate Manual_Jog_Accel
20.0
Accel Units Units per sec2
Decel Rate Manual_Jog_Decel
20.0
Decel Units Units per sec2
Profile S-Curve
Accel Jerk Manual_Jog_Accel_Jerk
100.0
Decel Jerk Manual_Jog_Decel_Jerk
100.0
Jerk Units % of Time
Merge Disabled
Merge Speed Programmed
<< Less
EN
DN
ER
IP
The MAJ instruction that starts the axis has a higher acceleration than the instruction that stops the axis.
S-curve profile
Jog_PB
<Local:4:I.Data[1].0>
My_Axis_OK
MAJ
Motion Axis Jog
Axis My_Axis
Motion Control Jog_2
Direction 0
Speed Jog_2_Speed
0.0
Speed Units Units per sec
Accel Rate Jog_2_Accel
10.0
Accel Units Units per sec2
Decel Rate Jog_2_Decel
20.0
Decel Units Units per sec2
Profile S-Curve
Accel Jerk 100.0
Decel Jerk 100.0
Jerk Units % of Time
Merge Disabled
Merge Speed Programmed
<< Less
EN
DN
ER
IP
The MAJ instruction that stops the axis has a lower acceleration than the instruction that starts the axis.
S-curve profile

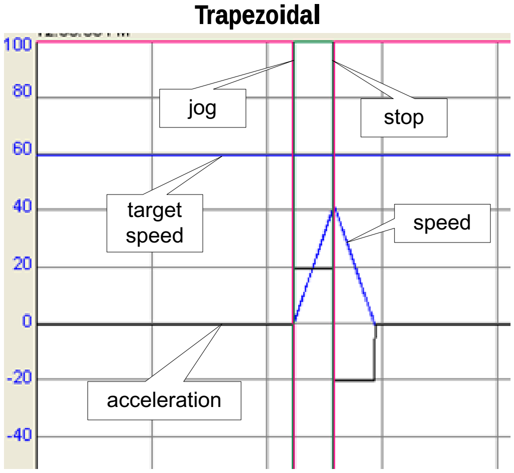
Cause When you use an S-curve profile, jerk determines how fast an axis can change its acceleration and deceleration.

- When the stopping instruction starts, the controller recalculates jerk and builds a new S-curve profile.
- If the stopping instruction uses a lower acceleration, the jerk is lower. It takes longer at the lower jerk to get acceleration to zero.
- In the meantime, the axis continues past its initial target speed.

The following trends show how the axis stops with a trapezoidal profile and an S-curve profile.

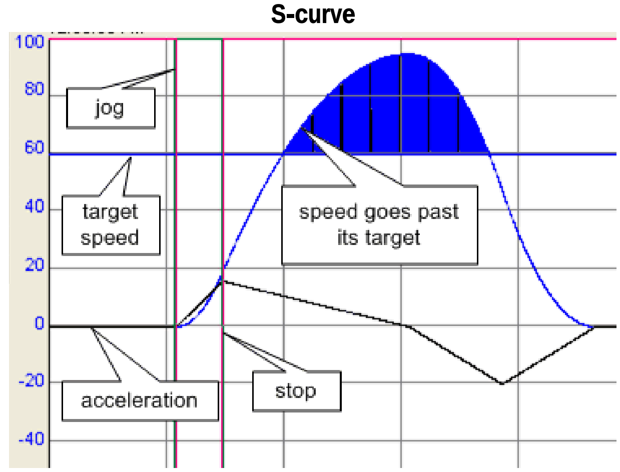
Stop while accelerating and reduce the acceleration rate

Trapezoidal



The axis slows down as soon as you start the stopping instruction. The lower acceleration doesn't change the response of the axis.

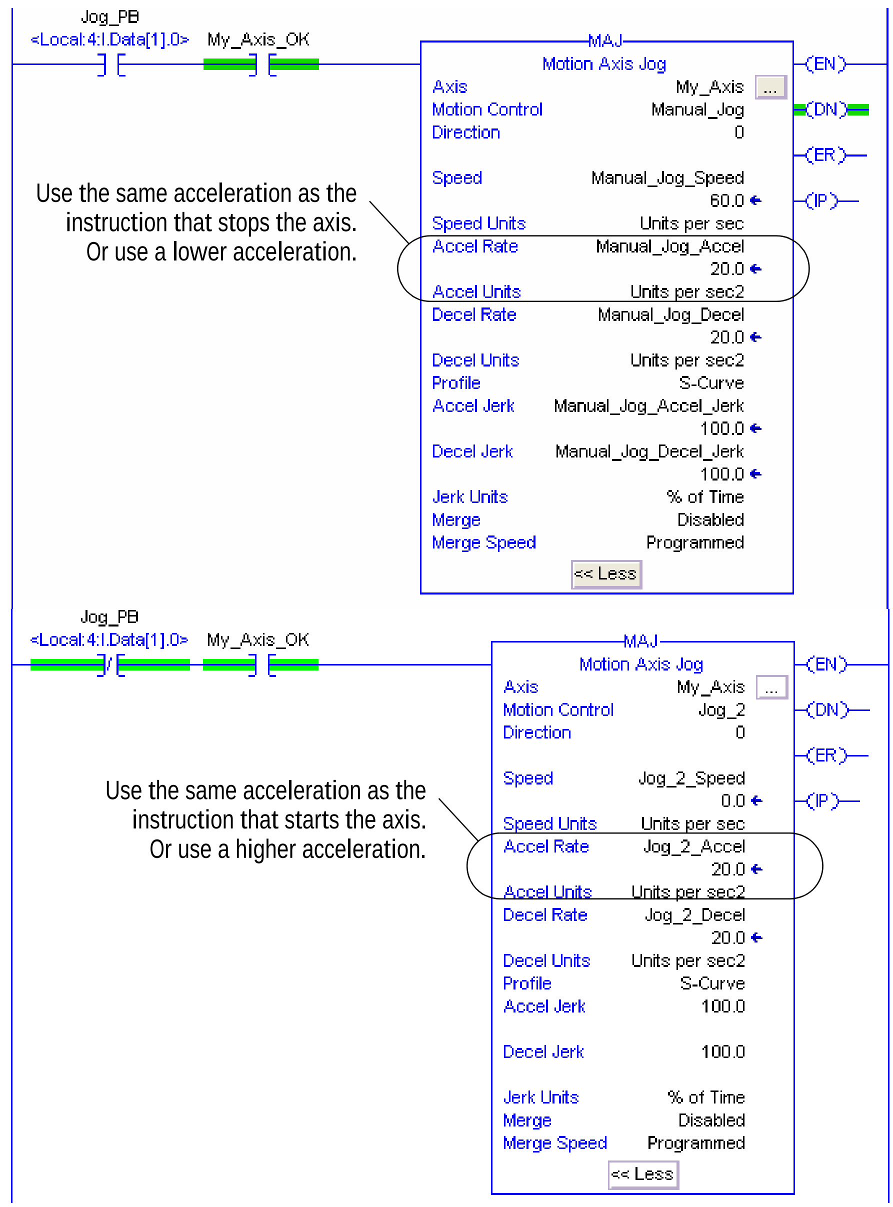
S-curve



The stopping instruction reduces the acceleration of the axis. It now takes longer to bring the acceleration to zero. The axis continues past its target speed until acceleration equals zero.

Corrective action Use a Motion Axis Stop (MAS) instruction to stop the axis.

Or set up your instructions like this:



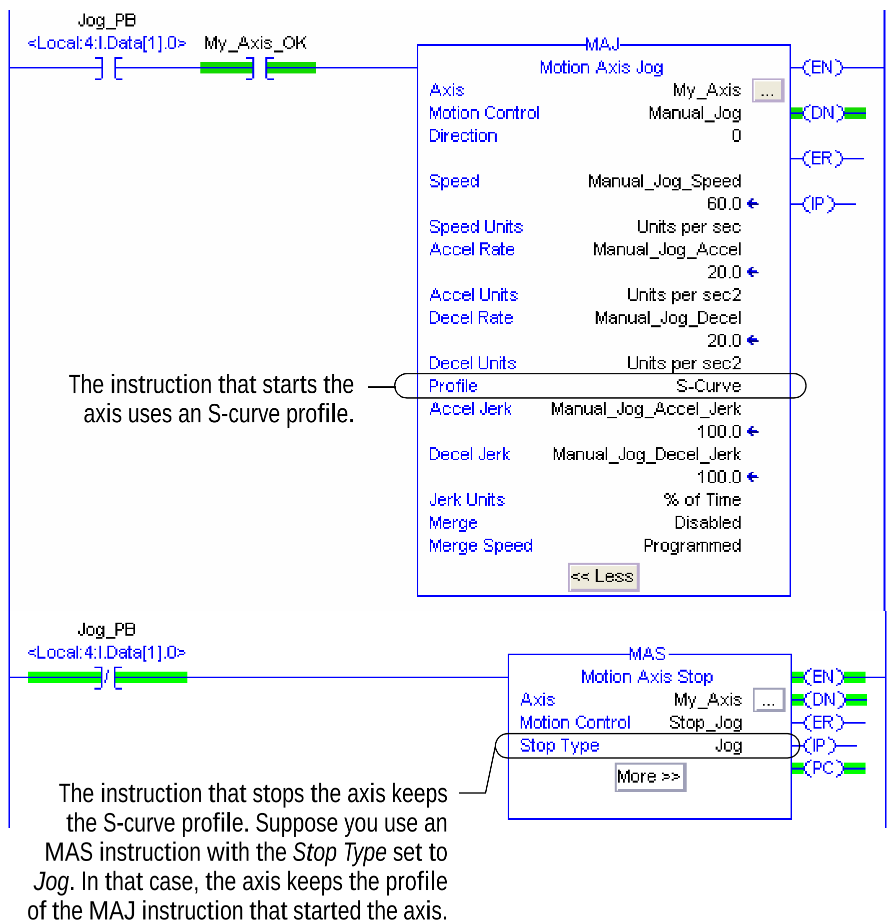
Why is there a delay when I stop and then restart a jog?

While an axis is jogging at its target speed, you stop the axis. Before the axis stops completely, you restart the jog. The axis continues to slow down before it speeds up.

Example

You use a Motion Axis Stop (MAS) instruction to stop a jog. While the axis is slowing down, you use a Motion Axis Jog (MAJ) instruction to start the axis again. The axis doesn't respond right away. It continues to slow down. Eventually it speeds back up to the target speed.

Look for



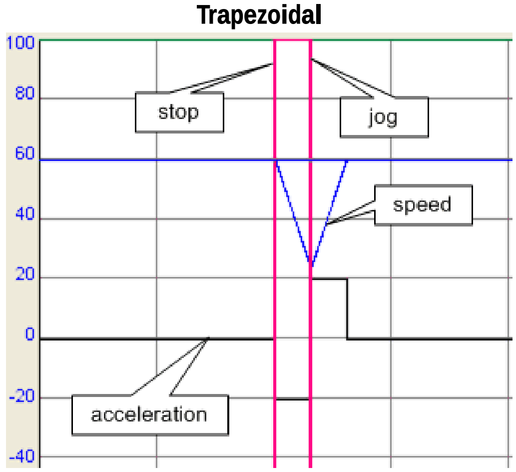
Cause When you use an S-curve profile, jerk determines how fast an axis can change its acceleration and deceleration.

- An S-curve profile has to get acceleration to zero before the axis can speed up again.
- The axis continues to slow down until the S-curve profile brings the acceleration to zero.

The following trends show how the axis stops and starts with a trapezoidal profile and an S-curve profile.

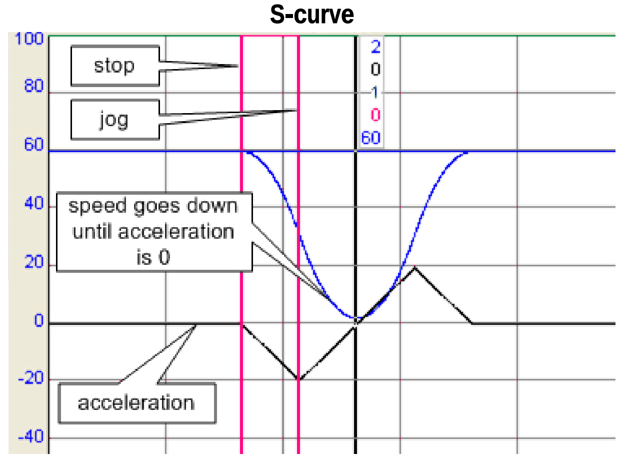
Start while decelerating

Trapezoidal



The axis speeds back up as soon as you start the jog again.

S-curve



The axis continues to slow down until the S-curve profile brings the acceleration rate to zero.

Corrective action

If your controller is this revision	Then	Result
15 or earlier	Increase the deceleration rate of the MAJ instruction that starts the jog.	This increases the deceleration jerk. The axis stops the deceleration sooner at the higher deceleration jerk.
16 or later	Increase the deceleration jerk of the MAJ instruction that starts the jog.	The axis stops the deceleration sooner at the higher deceleration jerk.

Why does my axis reverse direction when I stop and start it?

While an axis is jogging at its target speed, you stop the axis. Before the axis stops completely, you restart the jog. The axis continues to slow down and then reverse direction. Eventually the axis changes direction again and moves in the programmed direction.

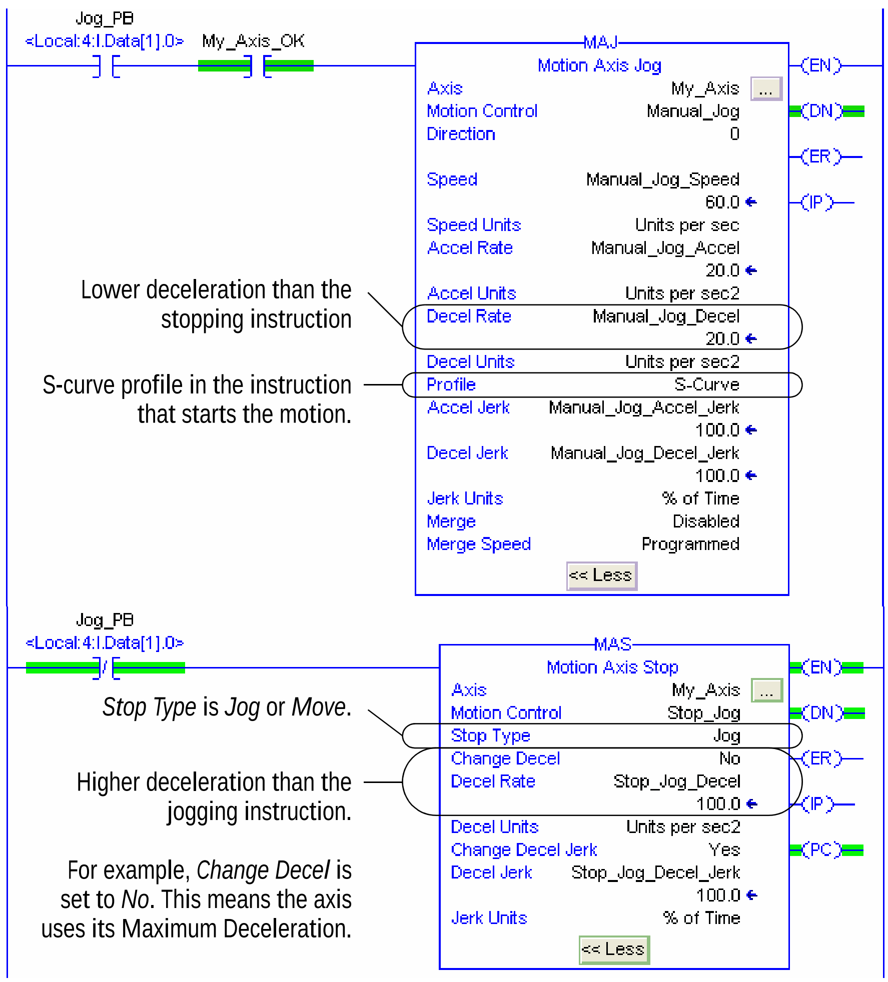
IMPORTANT You shouldn't see this situation in revision 16 and later.

See Corrective action for Revision 16 or Later on page 380.

Example

You use a Motion Axis Stop (MAS) instruction to stop a jog. While the axis is slowing down, you use a Motion Axis Jog (MAJ) instruction to start the axis again. The axis continues to slow down and then moves in the opposite direction. Eventually goes back to its programmed direction.

Look for



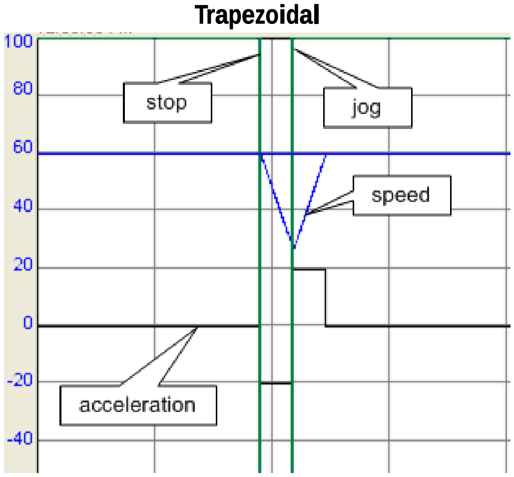
Cause When you use an S-curve profile, jerk determines how fast an axis can change its acceleration and deceleration.

- When the MAJ instruction starts again, the controller recalculates jerk and builds a new S-curve profile.
- If the MAJ instruction uses a lower deceleration, the jerk is lower. It takes longer at the lower jerk to get deceleration to zero.
- In the meantime, the axis continues past zero speed and moves in the opposite direction.

The following trends show how the axis stops and starts with a trapezoidal profile and an S-curve profile.

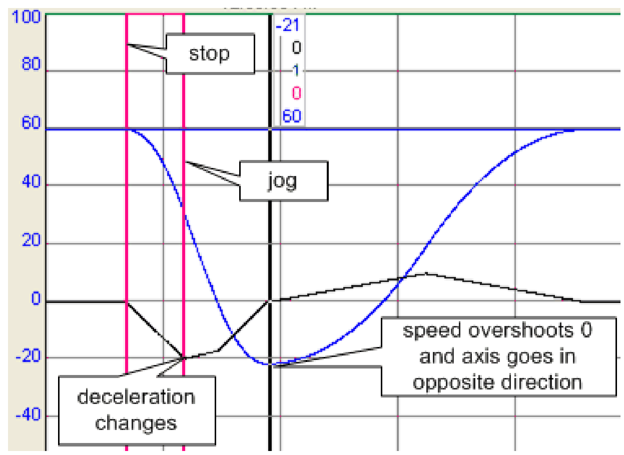
Start while decelerating and reduce the deceleration rate

Trapezoidal



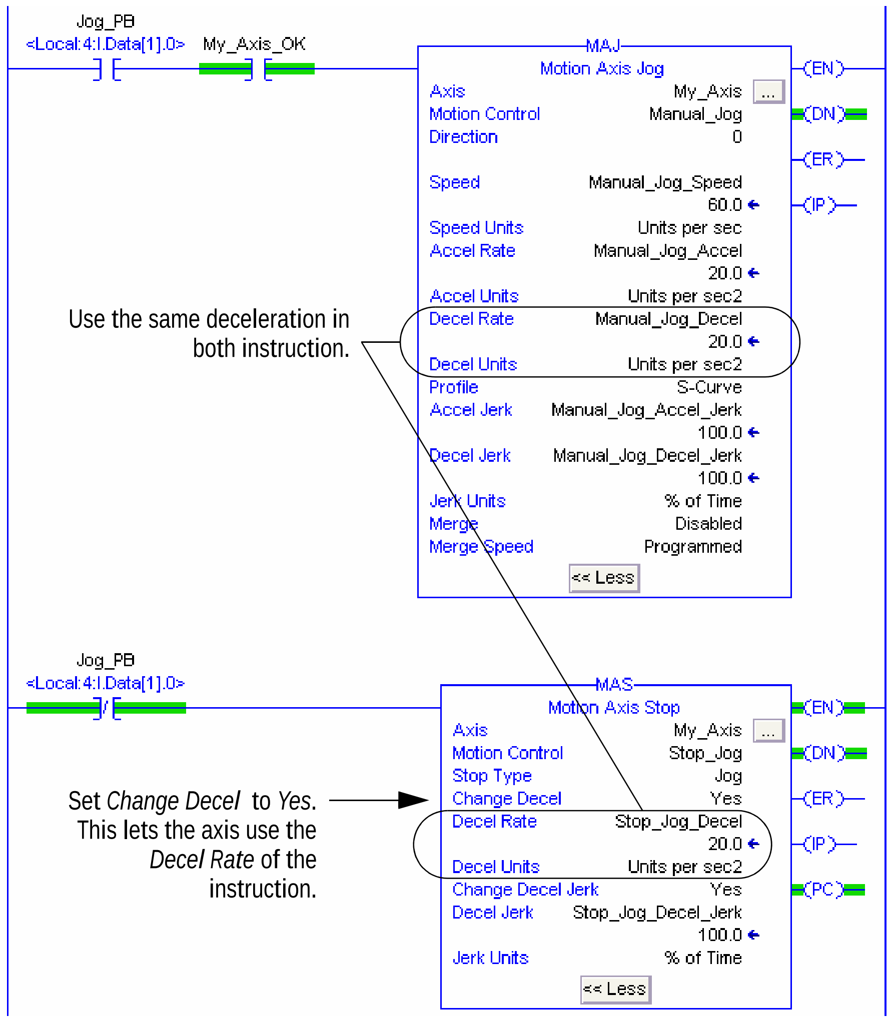
The axis speeds back up as soon as you start the jog again. The lower deceleration doesn't change the response of the axis.

S-curve



The jog instruction reduces the deceleration of the axis. It now takes longer to bring the deceleration to zero. The speed overshoots zero and the axis moves in the opposite direction.

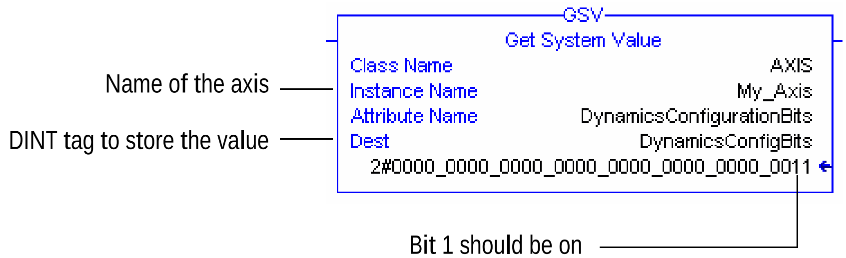
Corrective action *Revision 15 or Earlier*



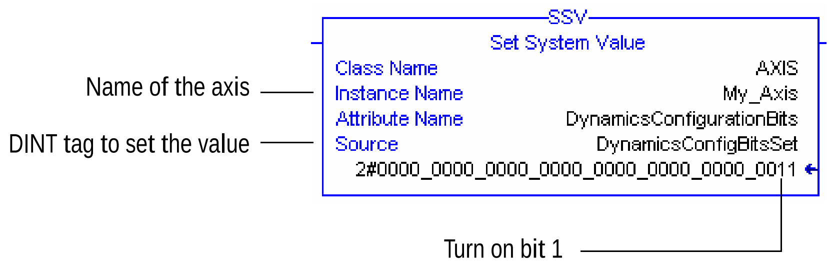
Revision 16 or Later

Revision 16 improved how the controller handles changes to an S-curve profile. If you're still seeing an axis reversal, make sure bit 1 of the DynamicsConfigurationBits for the axis is on:

1. Use a Get System Value (GSV) instruction to see if the algorithm is on.



2. If bit 1 is off, turn it on.



See the RSLogix 5000 online help for more information:

Help > Contents > GSV/SSV Objects > Axis > Dynamics Configuration Bits

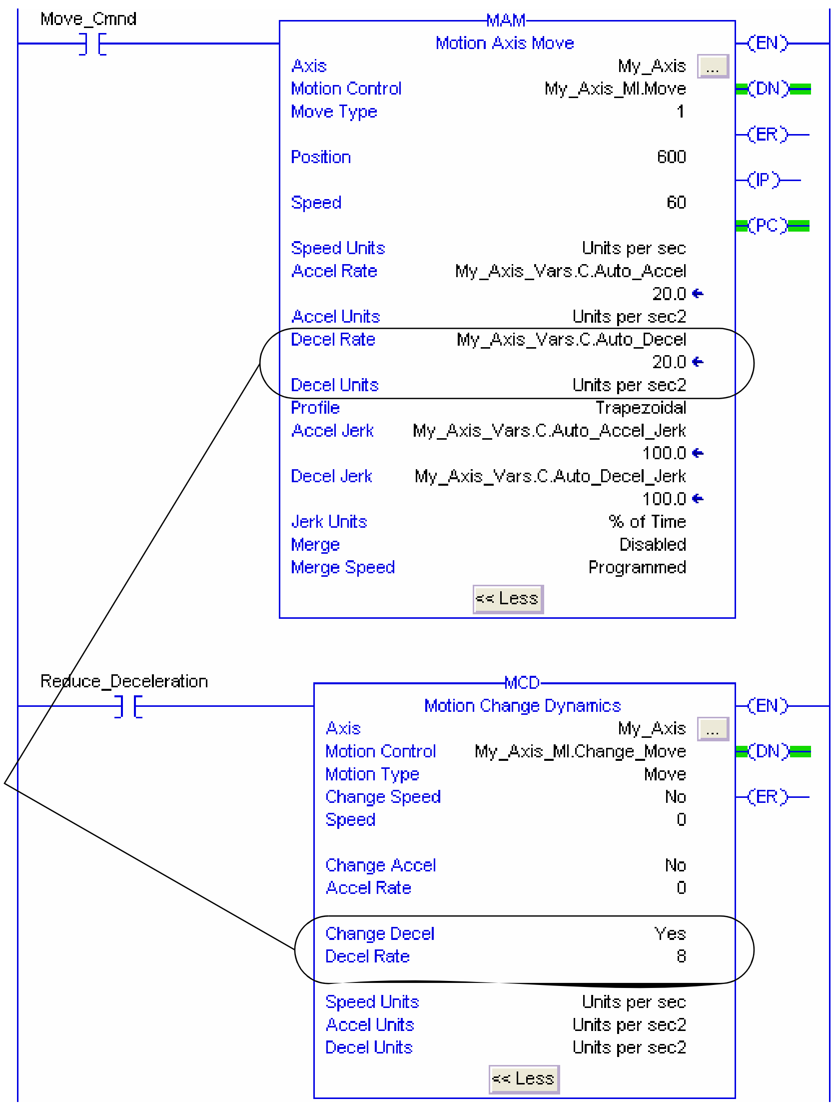
Why does my axis overshoot its position and reverse direction?

While an axis is moving to a target position, you change a parameter of the move. The axis overshoots its target position. Eventually the axis stops and moves back to its target position.

Example

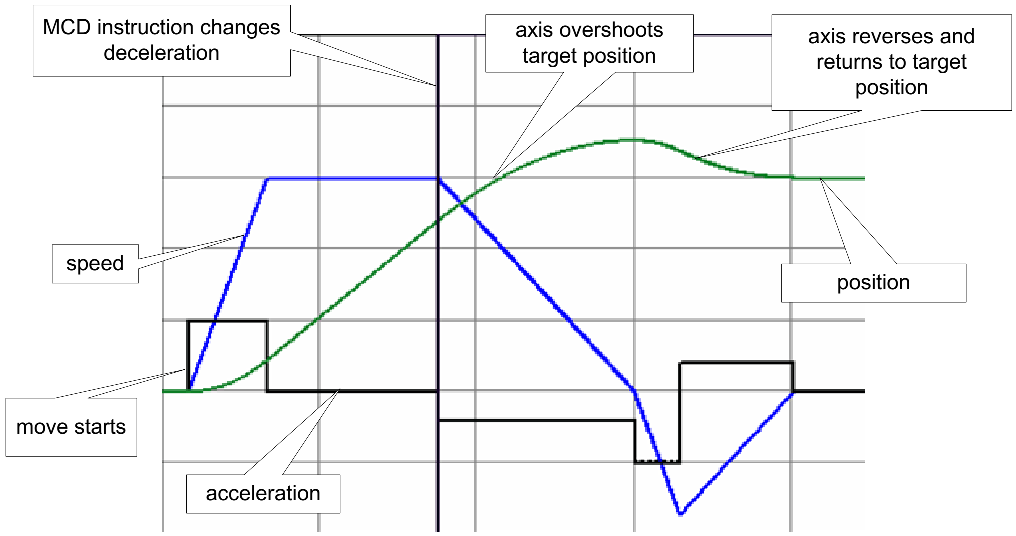
You use a Motion Change Dynamics (MCD) instruction to reduce the deceleration while a Motion Axis Move (MAM) instruction is in process. The axis continues past the target position of the move, stops, and returns to the target position.

Look for



MCD instruction that reduces the deceleration of the move

Cause The axis doesn't have enough time at the new lower deceleration to stop at the target position. It stops past the target position. Then it corrects to get back to the target position.

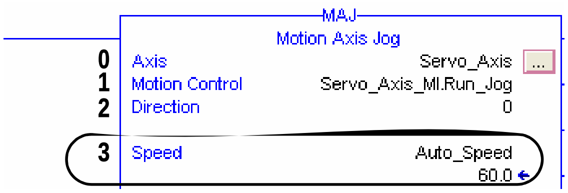


Corrective action To avoid overshooting position, either:

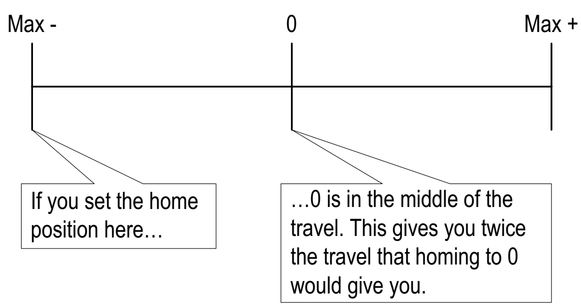
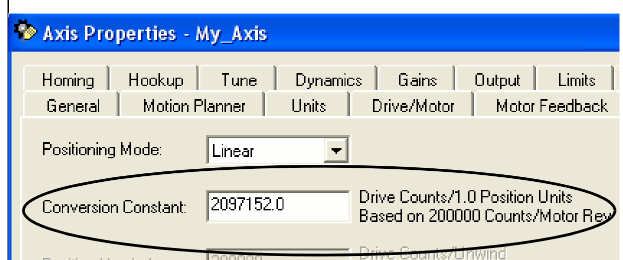
- Avoid decreasing the deceleration or deceleration jerk while an axis is decelerating along an S-curve profile.
- Avoid increasing the programmed speed while an axis is decelerating along an S-curve profile. This has the same effect as decreasing the deceleration jerk.
- Test any changes in small increments to make sure a change doesn't cause an overshoot during normal operation.

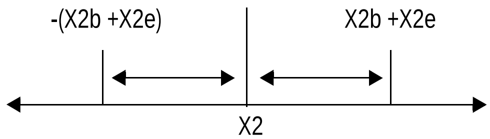
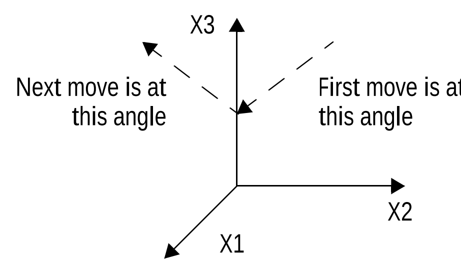
Error Codes (ERR) for Motion Instructions

Error	Corrective Action or Cause	Notes
3	Look for another instance of this type of instruction. See if its EN bit is on but its DN and ER bits are off (enabled but not done or errored). Wait until its DN or ER bit turns on.	Execution Collision You can't execute an instruction if the same type of instruction is enable but not done or errored.
4	Open the servo loop before you execute this instruction.	Servo On State Error
5	Close the servo loop before you execute this instruction.	Servo Off State Error For a motion coordinated instruction, look at the extended error code (EXERR). It identifies which axis caused the error. **Example**: If EXERR is zero, check the axis for dimension zero.
6	Disable the axis drive.	Drive On State Error
7	Execute a Motion Axis Shutdown Reset (MASR) instruction or direct command to reset the axis.	Shutdown State Error For a motion coordinated instruction, look at the extended error code (EXERR). It identifies which axis caused the error. **Example**: If EXERR is zero, check the axis for dimension zero.
8	The configured axis type is not correct.	Wrong Axis Type For a motion coordinated instruction, look at the extended error code (EXERR). It identifies which axis caused the error. **Example**: If EXERR is zero, check the axis for dimension zero.
9	The instruction tried to execute in a direction that aggravates the current overtravel condition.	Overtravel Condition
10	The master axis reference is the same as the slave axis reference.	Master Axis Conflict
11	At least one axis is not configured to a physical motion module or has not been assigned to a Motion Group.	Axis Not Configured For a motion coordinated instruction, look at the extended error code (EXERR). It identifies which axis caused the error. **Example**: If EXERR is zero, check the axis for dimension zero.
12	Messaging to the servo module failed.	Servo Message Failure

Error	Corrective Action or Cause	Notes
13	Look at the extended error code (EXERR) for the instruction. It identifies an operand that is outside its range. **Example**: Suppose an MAJ instruction has an ERR = 13 and an EXERR = 3. In that case, change the speed so that it's in range. 	Parameter Out Of Range An EXERR = 0 means the first operand of the instruction is outside its range.
14	The instruction cannot apply the tuning parameters because of an error in the run tuning instruction.	Tune Process Error
15	The instruction cannot apply the diagnostic parameters because of an error in the run diagnostic test instruction.	Test Process Error
16	Wait until the homing process is done.	Home In Process Error
17	The instruction tried to execute a rotary move on an axis that is not configured for rotary operation.	Axis Mode Not Rotary
18	The axis type is configured as unused.	Axis Type Unused
19	The motion group is not in the synchronized state. This could be caused by a missing or mis-configured servo module.	Group Not Synchronized
20	The axis is in the faulted state.	Axis In Faulted State
21	The group is in the faulted state.	Group In Faulted State
22	Stop the axis before you execute this instruction.	Axis In Motion
23	An instruction attempted an illegal change of dynamics.	Illegal Dynamic Change
24	Take the controller out of test mode.	Illegal AC Mode Op
25	You attempted to execute an instruction that is not correct.	Illegal Instruction
26	The cam array is of an illegal length.	Illegal Cam Length
27	The cam profile array is of an illegal length.	Illegal Cam Profile Length
28	You have an illegal segment type in the cam element.	Illegal Cam Type
29	You have an illegal order of cam elements.	Illegal Cam Order
30	You tried to execute a cam profile while it is being calculated.	Cam Profile Being Calculated
31	The cam profile array you tried to execute is in use.	Cam Profile Being Used
32	The cam profile array you tried to execute has not been calculated.	Cam Profile Not Calculated
33	It attempted to execute an MAH instruction without a position cam in process.	Position Cam Not Enabled
34	A MAH instruction is trying to start while a registration is already running.	Registration in Progress
35	Either the controller or the Output Cam module does not support the specified Output Cam, axis, input or output.	Illegal Execution Target
36	Either the size of the Output Cam array is not supported or the value of one of its members is out of range.	Illegal Output Cam

Error	Corrective Action or Cause	Notes
37	Either the size of the Output Compensation array is not supported or the value of one of its members is out of range.	Illegal Output Compensation
38	The axis data type is illegal. It is incorrect for the operation.	Illegal Axis Data Type For a motion coordinated instruction, look at the extended error code (EXERR). It identifies which axis caused the error. **Example**: If EXERR is zero, check the axis for dimension zero.
39	You have a conflict in your process. Test and Tune cannot be run at the same time.	Process Conflict
40	You are trying to run a MSO or MAH instruction when the drive is locally disabled.	Drive Locally Disabled
41	The Homing configuration is illegal. You have an absolute homing instruction when the Homing sequence is not immediate.	Illegal Homing Config
42	The MASD or MGSD instruction has timed out because it did not receive the shutdown status bit. Usually a programmatic problem caused when either MASD or MGSD is followed by a reset instruction which is initiated before the shutdown bit has been received by the shutdown instruction.	Shutdown Status Timeout
43	You have tried to activate more motion instructions than the instruction queue can hold.	Coordinate System Queue Full
44	You have drawn a line with three 3 points and no centerpoint viapoint or plane centerpoint can be determined.	Circular Collinearity Error
45	You have specified one 1 point radius or “drawn a line” centerpoint, viapoint and no centerpoint radius or plane centerpoint, viapoint can be determined.	Circular Start End Error
46	The programmed centerpoint is not equidistant from start and end point.	Circular R1 R2 Mismatch Error
47	Contact Rockwell Automation Support.	Circular Infinite Solution Error
48	Contact Rockwell Automation Support.	Circular No Solutions Error
49	|R| < 0.01. R is basically too small to be used in computations.	Circular Small R Error
50	The coordinate system tag is not associated with a motion group.	Coordinate System Not in Group
51	You have set your Termination Type to Actual Position with a value of 0. This value is not supported.	Invalid Actual Tolerance
52	At least one axis is currently undergoing coordinated motion in another coordinate system.	Coordination Motion In Process Error
53	Uninhibit the axis.	Axis Is Inhibited
54	1. Open the properties for the axis. 2. On the Dynamics tab, enter a value for the Maximum Deceleration.	Zero Max Decel You can’t start motion if the maximum deceleration for the axis is zero.
61	See the extended error code (EXERR) for the instruction.	Connection Conflict

Error	Corrective Action or Cause	Notes
62	Cancel the transform that controls this axis or don't use this instruction while the transform is active.	Transform In Progress You can't execute this instruction if the axis is part of a active transform.
63	Cancel the transform that controls this axis or wait until the transform is done moving the axis.	Axis In Transform Motion You can't execute this instruction if a transform is moving the axis.
64	Use a Cartesian coordinate system.	Ancillary Not Supported You can't use a non-Cartesian coordinate system with this instruction.
65	The axis moved too far and the controller can't store the position. To prevent this error, Set up soft travel limits that keep the axis within the position range. One way to get more travel is to use the max negative or max positive position as your home position. **Example**  	Axis Position Overflow The range for position depends on the conversion constant of the axis.  • Maximum positive position = 2,147,483,647 / conversion constant of the axis • Maximum negative position = -2,147,483,648 / conversion constant of the axis Suppose you have a conversion constant of 2,097,152 counts/inch. In that case: • Maximum positive position = 2,147,483,647 / 2,097,152 counts/inch = 1023 inches • Maximum negative position = -2,147,483,648 / 2,097,152 counts/inch = -1023 inches For a motion coordinated instruction, look at the extended error code (EXERR). It identifies which axis caused the error. **Example**: If EXERR is zero, check the axis for dimension zero.
66	Be sure to keep the robot in the arm solution that you configured it in. You can configure the robot in either a left arm or right arm solution.	You are attempting to fold back an articulated independent or dependent two axis robot on itself at the quadrant boundaries.

Error	Corrective Action or Cause	Notes
67	• Change the target positions to values that are within the reach of the robot. • If X2b +X2e isn't zero, stay out of this region: 	Invalid Transform Position • You're trying to move to a place the robot can't reach. • MCT attempted while at origin.
68	Move the joints so that the end of the robot isn't at the origin of the coordinate system.	Transform At Origin You can't start the transform if the joint angles result in X1 = 0 and X2 = 0.
69	• Check the maximum speed configuration of the joints. • Use target positions that keep the robot from getting fully stretched or folding back on itself at the origin of the coordinate system. • Move in a relatively straight line through positions where X1 = 0 and X2 = 0.	Max Joint Velocity Exceeded The calculated speed is very high. This happens when the robot either: • gets fully stretched. • folds back on itself. • moves away from X1 = 0 and X2 = 0 in a different angle than it approached that position. • is configured with the wrong velocity limit. **Example**: These moves produce this error. 
70	Look for source or target axes that are configured as rotary positioning mode. Change them to linear positioning mode.	Axes In Transform Must Be Linear A transform works only with linear axes.
71	Wait until the transform that you are canceling is completely canceled.	Transform Is Canceling
72	Check the target positions. A calculated joint angle is beyond +/- 360°.	Max Joint Angle Exceeded
73	Check that each MCT instruction in this chain is producing valid positions.	Coord System Chaining Error This MCT instruction is part of a chain of MCT instructions. There is a problem with one of the instructions in the chain.
74	Change the orientation to angles that are within +/- 360°.	Invalid Orientation Angle

<table>
<tr><th>Error</th><th>Corrective Action or Cause</th><th>Notes</th></tr>
<tr><td>75</td><td>Use this instruction only with a 1756-L6x controller.</td><td>Instruction Not Supported

You can use an MCT or MCTP instruction only with a 1756-L6x controller.</td></tr>
<tr><td>76</td><td>1. Open the properties for the axis.

2. On the Dynamics tab, enter a value for the maximum deceleration jerk.</td><td>Zero Max Decel Jerk

You can't start motion that uses an S-curve profile if the maximum deceleration jerk for the axis is zero.</td></tr>
<tr><td>77</td><td>How many axes are in your coordinate system?

• 2 — Use a non-mirror transform direction.
• 3 — Use a non-inverse transform direction.</td><td>Transform Direction Not Supported

1. You're trying to use the mirror directions with a 3-axis coordinate system and a non-zero base offset (X2b) or effector offset (X2e).

2. Mirror directions are not supported for 2-axis Coordinate Systems.

3. You are attempting to use either a 2 or 3-axis Cartesian target coordinate system with transform directions other than forward and inverse.

You can use inverse mirror directions only when both these conditions are true:

• You have a 3-axis coordinate system.
• The base offset (X2b) and end effector offset (X2e) of the X2 dimension are zero.</td></tr>
</table>

Appendix B

Motion-related Data Types (Structures)

Introduction

Use this appendix for information about the following motion-related data types:

Additional Resources

For other motion-related data types, see this publication:

Motion Modules in Logix5000 Control Systems User Manual, publication LOGIX-UM002A-EN-P.

CAM Structure

The Cam data type consists of slave and master point pairs as well as an interpolation type. Since there is no association with a specific axis position or time, the point values are unit-less. The interpolation type can be specified for each segment as either linear or cubic. The format of the cam array element is shown in the following table.

<table>
<tr><th>Mnemonic</th><th>Data Type</th><th colspan="2">Description</th></tr>
<tr><td>MASTER</td><td>REAL</td><td colspan="2">The x value of the point.</td></tr>
<tr><td>SLAVE</td><td>REAL</td><td colspan="2">The y value of the point.</td></tr>
<tr><td rowspan="4">Segment Type</td><td rowspan="4">DINT</td><td colspan="2">The type of interpolation.</td></tr>
<tr><td>Value</td><td>Description</td></tr>
<tr><td>0</td><td>linear</td></tr>
<tr><td>1</td><td>cubic</td></tr>
</table>

CAM_PROFILE Structure

The CAM_PROFILE data type is an array of coefficients representing a calculated cam profile that can be used as input to a time cam or position cam instruction. The only element available to the user is Status which is defined in the following table.

<table>
<tr><th>Mnemonic</th><th>Data Type</th><th colspan="2">Description</th></tr>
<tr><td rowspan="6">Status</td><td rowspan="6">DINT</td><td colspan="2">The status parameter is used to indicate that the Cam Profile array element has been calculated. If execution of a camming instruction is attempted using an uncalculated element in a Cam Profile, the instruction produces an error.</td></tr>
<tr><td>Value</td><td>Description</td></tr>
<tr><td>0</td><td>Cam profile element has not been calculated.</td></tr>
<tr><td>1</td><td>Cam profile element is being calculated.</td></tr>
<tr><td>2</td><td>Cam profile element has been calculated.</td></tr>
<tr><td>n</td><td>Cam profile element has been calculated and is currently being used by (n-2) MAPC and MATC instructions.</td></tr>
</table>

MOTION_GROUP Structure

There is one MOTION_GROUP structure per controller. This structure contains status and configuration information about the motion group.

<table>
<tr><th>Mnemonic</th><th>Data Type</th><th colspan="4">Description</th></tr>
<tr><td rowspan="6">GroupStatus</td><td rowspan="6">DINT</td><td colspan="4">The status bits for the group.</td></tr>
<tr><td>Bit</td><td>Number</td><td>Data Type</td><td>Description</td></tr>
<tr><td>InhibStatus</td><td>00</td><td>DINT</td><td>inhibit status</td></tr>
<tr><td>GroupSynced</td><td>01</td><td>DINT</td><td>synchronization status</td></tr>
<tr><td>-no-tag</td><td>02</td><td>DINT</td><td>Timer Event started</td></tr>
<tr><td>Reserved</td><td>03-31</td><td></td><td></td></tr>
<tr><td rowspan="5">MotionFault</td><td rowspan="5">DINT</td><td colspan="4">The motion fault bits for the group.</td></tr>
<tr><td>Bit</td><td>Number</td><td>Data Type</td><td>Description</td></tr>
<tr><td>ACAsyncConnFault</td><td>00</td><td>DINT</td><td>asynchronous connection fault</td></tr>
<tr><td>ACSyncConnFault</td><td>01</td><td>DINT</td><td>synchronous connection fault</td></tr>
<tr><td>Reserved</td><td>02-31</td><td></td><td></td></tr>
<tr><td rowspan="15">ServoFault</td><td rowspan="15">DINT</td><td colspan="4">The servo-module fault bits for the group.</td></tr>
<tr><td>Bit</td><td>Number</td><td>Data Type</td><td>Description</td></tr>
<tr><td>POtrvlFault</td><td>00</td><td>DINT</td><td>positive overtravel fault</td></tr>
<tr><td>NOtrvlFault</td><td>01</td><td>DINT</td><td>negative overtravel fault</td></tr>
<tr><td>PosErrorFault</td><td>02</td><td>DINT</td><td>position error fault</td></tr>
<tr><td>EncCHALossFault</td><td>03</td><td>DINT</td><td>encoder channel A loss fault</td></tr>
<tr><td>EncCHBLossFault</td><td>04</td><td>DINT</td><td>encoder channel B loss fault</td></tr>
<tr><td>EncCHZLossFault</td><td>05</td><td>DINT</td><td>encoder channel Z loss fault</td></tr>
<tr><td>EncNsFault</td><td>06</td><td>DINT</td><td>encoder noise fault</td></tr>
<tr><td>DriveFault</td><td>07</td><td>DINT</td><td>drive fault</td></tr>
<tr><td>Reserved</td><td>08-31</td><td></td><td></td></tr>
<tr><td>Bit</td><td>Number</td><td>Data Type</td><td>Description</td></tr>
<tr><td>SyncConnFault</td><td>00</td><td>DINT</td><td>synchronous connection fault</td></tr>
<tr><td>HardFault</td><td>01</td><td>DINT</td><td>servo hardware fault</td></tr>
<tr><td>Reserved</td><td>02-31</td><td></td><td></td></tr>
<tr><td rowspan="6">GroupFault</td><td rowspan="6">DINT</td><td colspan="4">The fault bits for the group.</td></tr>
<tr><td>Bit</td><td>Number</td><td>Data Type</td><td>Description</td></tr>
<tr><td>GroupOverlapFault</td><td>00</td><td>DINT</td><td>group task overlap fault</td></tr>
<tr><td>CSTLossFault</td><td>01</td><td>DINT</td><td>The controller has lost synchronization with the CST master</td></tr>
<tr><td>GroupTaskLoadingFault</td><td>02</td><td>DINT</td><td>The group coarse update period is too low, user application tasks are not getting enough time to execute.</td></tr>
<tr><td>Reserved</td><td>03-31</td><td></td><td></td></tr>
</table>

MOTION_INSTRUCTION Data Type

You must define a motion control tag for each motion instruction that you use. The tag uses the MOTION_INSTRUCTION data type and stores status information about the instruction.

Mnemonic	Data Type	Description	
FLAGS	DINT	Use this DINT to access all the status bits for the instruction in one 32-bit value.	
		For this status bit	**Use this bit number**
		EN	31
		DN	29
		ER	28
		PC	27
		IP	26
		AC	23
		DECEL	1
		ACCEL	0
EN	BOOL	The enable bit indicates that the instruction is enabled (the rung-in and rung-out condition is true).	
DN	BOOL	The done bit indicates that all calculations and messaging (if any) are complete.	
ER	BOOL	The error bit indicates when the instruction is used illegally.	
PC	BOOL	The process complete bit indicates that the operation is complete. The .DN bit sets after an instruction has completed execution. The .PC bit sets when the initiated process has completed.	
IP	BOOL	The in process bit indicates that a process is being executed.	
AC	BOOL	The Active Bit lets you know which instruction is controlling the motion when you have instructions queued. It sets when the instruction becomes active. It is reset when the Process Complete bit is set or when the instruction is stopped.	
ACCEL	BOOL	The .ACCEL bit indicates that the velocity has increased for the individual instruction that it is tied to i.e jog, move, gearing	
DECEL	BOOL	The .DECEL bit indicates that the velocity has decreased for the individual instruction that it is tied to i.e jog, move, gearing.	
ERR	INT	The error value contains the error code associated with a motion function. See Error Codes (ERR) for Motion Instructions on page 383.	
STATUS	SINT	The status of any message associated with the motion function.	
		Message Status	**Description**
		0x0	The message was successful.
		0x1	The module is processing another message.
		0x2	The module is waiting for a response to a previous message.
		0x3	The response to a message failed.
		0x4	The module is not ready for messaging.

Mnemonic	Data Type	Description
STATE	SINT	The execution status value keeps track of the execution state of a function. Many motion functions have several steps and this value tracks these steps. The execution status is always set to 0 when the controller sets the EN bit for a motion instruction. Other execution states depend on the motion instruction.
SEGMENT	DINT	A segment is the distance from one point up to but, not including the next point. A SEGMENT gives the relative position by segment number as the Cam is executing.
EXERR	SINT	Extended error code —use it for more information about an error.

OUTPUT_CAM Structure

The OUTPUT_CAM data type is an array that defines the specifics for each Output Cam element. The OUTPUT_CAM contains the following members.

<table>
<tr><th>Mnemonic</th><th>Data Type</th><th colspan="2">Description</th></tr>
<tr><td>OutputBit</td><td>DINT</td><td colspan="2">You must select an output bit within the range of 0 to 31. A selection of less than 0 or greater than 31 results in an Illegal Output Cam error and the cam element is not considered.</td></tr>
<tr><td rowspan="6">LatchType</td><td rowspan="6">DINT</td><td colspan="2">The Latch Type determines how the corresponding output bit is set. A value of less than 0 or greater than 3 results in an Illegal Output Cam error and a latch type of Inactive is used.</td></tr>
<tr><td>Value</td><td>Description</td></tr>
<tr><td>0 = Inactive</td><td>The output bit is not changed.</td></tr>
<tr><td>1 = Position</td><td>The output bit is set when the axis enters the compensated cam range.</td></tr>
<tr><td>2 = Enable</td><td>The output bit is set when the enable bit becomes active.</td></tr>
<tr><td>3 = Position and Enable</td><td>The output bit is set when the axis enters the compensated cam range and the enable bit becomes active.</td></tr>
<tr><td rowspan="8">UnlatchType</td><td rowspan="8">DINT</td><td colspan="2">The Unlatch Type determines how the output bit is reset. Selecting a value less than 0 or greater than 5 results in an Illegal Output Cam error and an unlatch type of Inactive is used.</td></tr>
<tr><td>Value</td><td>Description</td></tr>
<tr><td>0 = Inactive</td><td>The output bit is not changed.</td></tr>
<tr><td>1 = Position</td><td>The output bit is reset when the axis leaves the compensated cam range.</td></tr>
<tr><td>2 = Duration</td><td>The output bit is reset when the duration expires.</td></tr>
<tr><td>3 = Enable</td><td>The output bit is reset when the enable bit becomes inactive.</td></tr>
<tr><td>4 = Position and Enable</td><td>The output bit is reset when the axis leaves the compensated cam range or the enable bit becomes inactive.</td></tr>
<tr><td>5 = Duration and Enable</td><td>The output bit is reset when the duration expires or the enable bit becomes inactive.</td></tr>
<tr><td>Left</td><td>REAL</td><td colspan="2">The left cam position along with the right cam position define the cam range of the Output Cam element. The left and right cam positions specify the latch or unlatch positions of the output bit when the latch or unlatch type is set to Position or Position and Enable with the enable bit active. If the left position is less than the Cam Start position or greater than the Cam End position, an Illegal Output Cam error is returned and the cam element is not considered.</td></tr>
</table>

<table>
<tr><th>Mnemonic</th><th>Data Type</th><th colspan="2">Description</th></tr>
<tr><td>Right</td><td>REAL</td><td colspan="2">The right cam position along with the left cam position define the cam range of the Output Cam element. The right and left cam positions specify the latch or unlatch positions of the output bit when the latch or unlatch type is set to Position or Position and Enable with the enable bit active. If the right position is less than the Cam Start position or greater than the Cam End position, an Illegal Output Cam error is returned and the cam element is not considered.</td></tr>
<tr><td>Duration</td><td>REAL</td><td colspan="2">Duration specifies the time in seconds between latching and unlatching when the Unlatch Type is Duration or Duration and Enable with the enable bit active. A value less than or equal to 0 results in an Illegal Output Cam error and the cam element is not considered.</td></tr>
<tr><td rowspan="6">EnableType</td><td rowspan="6">DINT</td><td colspan="2">This defines the source and polarity of the specified EnableBit when LatchType or UnlatchType is Enable, Position and Enable or Duration and Enable. A value of less than 0 or greater than 31 results in an Illegal Output Cam error and the cam element is not considered.</td></tr>
<tr><td>Value</td><td>Description</td></tr>
<tr><td>0 = Input</td><td>The enable bit is in the Input parameter.</td></tr>
<tr><td>1 = Inverted Input</td><td>The enable bit is in the input parameter and is active low.</td></tr>
<tr><td>2 = Output</td><td>The enable bit is in the Output parameter.</td></tr>
<tr><td>3 = Inverted Output</td><td>The enable bit is in the Output parameter and is active low.</td></tr>
<tr><td>EnableBit</td><td>DINT</td><td colspan="2">The value of the Enable Bit selected must be between 0 and 31 when LatchType or UnlatchType is Enable, Position and Enable or Duration and Enable. A value of less than 0 or greater than 31 results in an Illegal Output Cam error and the cam element is not considered.</td></tr>
</table>

OUTPUT_COMPENSATION Structure

The OUTPUT_COMPENSATION data type defines the details for each output bit by setting the characteristics of each actuator. OUTPUT_COMPENSATION contains the following members:

<table>
<tr><th>Mnemonic</th><th>Data Type</th><th colspan="2">Description</th></tr>
<tr><td>Offset</td><td>REAL</td><td colspan="2">Offset provides position compensation for both the latch and unlatch operations.</td></tr>
<tr><td>LatchDelay</td><td>REAL</td><td colspan="2">Latch delay, programmed in seconds, provides time compensation for the latch operation.</td></tr>
<tr><td>UnlatchDelay</td><td>REAL</td><td colspan="2">Unlatch delay, programmed in seconds, provides time compensation for the unlatch operation.</td></tr>
<tr><td rowspan="6">Mode</td><td rowspan="6">DINT</td><td colspan="2">The Mode determines the behavior of the output bit. The following four mode options are available. A value of less than 0 or greater than 3 results in an Illegal Output Compensation error.</td></tr>
<tr><td>Value</td><td>Description</td></tr>
<tr><td>0 = Normal</td><td>The output bit is set for the latch operation and is reset for the unlatch operation.</td></tr>
<tr><td>1 = Inverted</td><td>The output bit is reset for the latch operation and is set for the unlatch the operation.</td></tr>
<tr><td>2 = Pulsed</td><td>The output bit is set for the latch operation and for the on-duty state of pulse and is reset for the unlatch operation and for the off-duty state of the pulse.</td></tr>
<tr><td>3 = Inverted and Pulsed</td><td>The output bit is reset for the latch operation and for the on-duty state of the pulse and is set for the unlatch operation and for the off-duty state of the pulse.</td></tr>
<tr><td>CycleTime</td><td>REAL</td><td colspan="2">Pulse time in seconds. If mode is Pulsed or Inverted and Pulsed, and CycleTime is less than or equal to 0, an Illegal Output Compensation error results.</td></tr>
<tr><td>DutyCycle</td><td>REAL</td><td colspan="2">The percent of CycleTime in which the pulse is to be turned on (on-duty). A value of 50 represents 50% on-duty. A value of less than 0 or greater than 100 returns an Illegal Output Compensation error.</td></tr>
</table>

Notes:

Structured Text Programming

Introduction

This appendix describes issues that are unique with structured text programming. Review the information in this appendix to make sure you understand how your structured text programming will execute.

Structured Text Syntax

Structured text is a textual programming language that uses statements to define what to execute. Structured text is not case sensitive. Structured text can contain these components:

Term	Definition	Examples
assignment (see page 399)	Use an assignment statement to assign values to tags. The := operator is the assignment operator. Terminate the assignment with a semi colon ";".	`tag := expression;`

Term	Definition		Examples
expression (see page 401)	An expression is part of a complete assignment or construct statement. An expression evaluates to a number (numerical expression) or to a true or false state (BOOL expression). An expression contains:		
	tags	A named area of the memory where data is stored (BOOL, SINT,INT,DINT, REAL, string).	`value1`
	immediates	A constant value.	`4`

	operators	A symbol or mnemonic that specifies an operation within an expression.	`tag1 + tag2`
			
			`tag1 >= value1`

	functions	When executed, a function yields one value. Use parentheses to contain the operand of a function. Even though their syntax is similar, functions differ from instructions in that functions can only be used in expressions. Instructions cannot be used in expressions.	`function(tag1)`
instruction (see page 408)	An instruction is a standalone statement. An instruction uses parenthesis to contain its operands. Depending on the instruction, there can be zero, one, or multiple operands. When executed, an instruction yields one or more values that are part of a data structure. Terminate the instruction with a semi colon “;”. Even though their syntax is similar, instructions differ from functions in that instructions cannot be used in expressions. Functions can only be used in expressions.		*`instruction();`* *`instruction(operand);`* *`instruction(operand1, operand2,operand3);`*

construct
 	
(see page 409)
	A conditional statement used to trigger structured text code (i.e, other statements).
	
	Terminate the construct with a semi colon “;”.
			`IF...THEN`
			
			`CASE`
			
			`FOR...DO`
			
			`WHILE...DO`
			
			`REPEAT...UNTIL`
			
			`EXIT`

comment (see page 425)	Text that explains or clarifies what a section of structured text does. • Use comments to make it easier to interpret the structured text. • Comments do not affect the execution of the structured text. • Comments can appear anywhere in structured text.		*`//comment`* `(`*`*start of comment . . . end of comment*`*`)` *`/*start of comment . . . end of comment*/`*

Assignments

Use an assignment to change the value stored within a tag. An assignment has this syntax:

tag := *expression* ;

where:

Component	Description	
tag	represents the tag that is getting the new value the tag must be a BOOL, SINT, INT, DINT, or REAL	
:=	is the assignment symbol	
expression	represents the new value to assign to the tag	
	If *tag* is this data type	**Use this type of expression**
	BOOL	BOOL expression
	SINT INT DINT REAL	numeric expression
;	ends the assignment	

The *tag* retains the assigned value until another assignment changes the value.

The expression can be simple, such as an immediate value or another tag name, or the expression can be complex and include several operators and/or functions. See the Expressions on page 401 for details.

TIP

I/O module data updates asynchronously to the execution of logic. If you reference an input multiple times in your logic, the input could change state between separate references. If you need the input to have the same state for each reference, buffer the input value and reference that buffer tag.

For more information, see Logix5000 Controllers Common Procedures, publication 1756-PM001.

Specify a non-retentive assignment

The non-retentive assignment is different from the regular assignment described above in that the tag in a non-retentive assignment is reset to zero each time the controller:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for *Automatic reset* (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

A non-retentive assignment has this syntax:

`tag` [:=] `expression` ;

where:

<table>
<tr><th>Component</th><th colspan="2">Description</th></tr>
<tr><td><code>tag</code></td><td colspan="2">represents the tag that is getting the new value

the tag must be a BOOL, SINT, INT, DINT, or REAL</td></tr>
<tr><td>[:=]</td><td colspan="2">is the non-retentive assignment symbol</td></tr>
<tr><td rowspan="4"><code>expression</code></td><td colspan="2">represents the new value to assign to the tag</td></tr>
<tr><td>If <code>tag</code> is this data type</td><td>Use this type of expression</td></tr>
<tr><td>BOOL</td><td>BOOL expression</td></tr>
<tr><td>SINT
INT
DINT
REAL</td><td>numeric expression</td></tr>
<tr><td>;</td><td colspan="2">ends the assignment</td></tr>
</table>

Assign an ASCII character to a string

Use the assignment operator to assign an ASCII character to an element of the DATA member of a string tag. To assign a character, specify the value of the character or specify the tag name, DATA member, and element of the character. For example:

This is OK	This is *not* OK.
string1.DATA[0]:= 65;	string1.DATA[0] := A;
string1.DATA[0]:= string2.DATA[0];	string1 := string2;

To add or insert a string of characters to a string tag, use either of these ASCII string instructions:

To	Use this instruction
add characters to the end of a string	CONCAT
insert characters into a string	INSERT

Expressions

An expression is a tag name, equation, or comparison. To write an expression, use any of these elements:

- tag name that stores the value (variable)
- number that you enter directly into the expression (immediate value)
- functions, such as: ABS, TRUNC
- operators, such as: +, -, <, >, And, Or

As you write expressions, follow these general rules:

- Use any combination of upper-case and lower-case letter. For example, these three variations of “AND” are acceptable: AND, And, and.
- For more complex requirements, use parentheses to group expressions within expressions. This makes the whole expression easier to read and ensures that the expression executes in the desired sequence. See “Determine the order of execution“on page 407.

In structured text, you use two types of expressions:

BOOL expression: An expression that produces either the BOOL value of 1 (true) or 0 (false).

- A bool expression uses bool tags, relational operators, and logical operators to compare values or check if conditions are true or false. For example, `tag1>65`.
- A simple bool expression can be a single BOOL tag.
- Typically, you use bool expressions to condition the execution of other logic.

Numeric expression: An expression that calculates an integer or floating-point value.

- A numeric expression uses arithmetic operators, arithmetic functions, and bitwise operators. For example, `tag1+5`.
- Often, you nest a numeric expression within a bool expression. For example, `(tag1+5)>65`.

Use the following table to choose operators for your expressions:

If you want to	Then
Calculate an arithmetic value	“Use arithmetic operators and functions“on page 403.
Compare two values or strings	“Use relational operators“on page 404.
Check if conditions are true or false	“Use logical operators“on page 406.
Compare the bits within values	“Use bitwise operators“on page 407.

Use arithmetic operators and functions

You can combine multiple operators and functions in arithmetic expressions.

Arithmetic operators calculate new values.

To	Use this operator	Optimal data type
add	+	DINT, REAL
subtract/negate	-	DINT, REAL
multiply	*	DINT, REAL
exponent (x to the power of y)	**	DINT, REAL
divide	/	DINT, REAL
modulo-divide	MOD	DINT, REAL

Arithmetic functions perform math operations. Specify a constant, a non-boolean tag, or an expression for the function.

For	Use this function	Optimal data type
absolute value	ABS (*numeric_expression*)	DINT, REAL
arc cosine	ACOS (*numeric_expression*)	REAL
arc sine	ASIN (*numeric_expression*)	REAL
arc tangent	ATAN (*numeric_expression*)	REAL
cosine	COS (*numeric_expression*)	REAL
radians to degrees	DEG (*numeric_expression)*	DINT, REAL
natural log	LN (*numeric_expression*)	REAL
log base 10	LOG (*numeric_expression*)	REAL
degrees to radians	RAD (*numeric_expression*)	DINT, REAL
sine	SIN (*numeric_expression*)	REAL
square root	SQRT (*numeric_expression*)	DINT, REAL
tangent	TAN (*numeric_expression*)	REAL
truncate	TRUNC (*numeric_expression*)	DINT, REAL

For example:

Use this format	Example	
	For this situation	You'd write
value1 operator value2	If gain_4 and gain_4_adj are DINT tags and your specification says: "Add 15 to gain_4 and store the result in gain_4_adj."	`gain_4_adj := gain_4+15;`
operator value1	If alarm and high_alarm are DINT tags and your specification says: "Negate high_alarm and store the result in alarm."	`alarm:= -high_alarm;`
function(numeric_expression)	If overtravel and overtravel_POS are DINT tags and your specification says: "Calculate the absolute value of overtravel and store the result in overtravel_POS."	`overtravel_POS := ABS(overtravel);`
value1 operator (function((value2+value3)/2)	If adjustment and position are DINT tags and sensor1 and sensor2 are REAL tags and your specification says: "Find the absolute value of the average of sensor1 and sensor2, add the adjustment, and store the result in position."	`position := adjustment + ABS((sensor1 + sensor2)/2);`

Use relational operators

Relational operators compare two values or strings to provide a true or false result. The result of a relational operation is a BOOL value:

If the comparison is	The result is
true	1
false	0

Use these relational operators:

For this comparison	Use this operator	Optimal Data Type
equal	=	DINT, REAL, string
less than	<	DINT, REAL, string
less than or equal	<=	DINT, REAL, string
greater than	>	DINT, REAL, string
greater than or equal	>=	DINT, REAL, string
not equal	<>	DINT, REAL, string

For example:

Use this format	Example	
	For this situation	**You'd write**
value1 operator value2	If temp is a DINT tag and your specification says: "If temp is less than 100° then…"	`IF temp<100 THEN...`
stringtag1 operator stringtag2	If bar_code and dest are string tags and your specification says: "If bar_code equals dest then…"	`IF bar_code=dest THEN...`
char1 operator char2 To enter an ASCII character directly into the expression, enter the decimal value of the character.	If bar_code is a string tag and your specification says: "If bar_code.DATA[0] equals 'A' then…"	`IF bar_code.DATA[0]=65 THEN...`
bool_tag := bool_expressions	If count and length are DINT tags, done is a BOOL tag, and your specification says "If count is greater than or equal to length, you are done counting."	`done := (count >= length);`

How Strings Are Evaluated

The hexadecimal values of the ASCII characters determine if one string is less than or greater than another string.

- When the two strings are sorted as in a telephone directory, the order of the strings determines which one is greater.

ASCII Characters	Hex Codes
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

lesser

greater

AB < B

a > B

- Strings are equal if their characters match.
- Characters are case sensitive. Upper case "A" ($41) is *not* equal to lower case "a" ($61).

Use logical operators

Logical operators let you check if multiple conditions are true or false. The result of a logical operation is a BOOL value:

If the comparison is	The result is
true	1
false	0

Use these logical operators:

For	Use this operator	Data Type
logical AND	&, AND	BOOL
logical OR	OR	BOOL
logical exclusive OR	XOR	BOOL
logical complement	NOT	BOOL

For example:

Use this format	Example	
	For this situation	**You'd write**
BOOLtag	If photoeye is a BOOL tag and your specification says: "If photoeye_1 is on then…"	`IF photoeye THEN...`
NOT *BOOLtag*	If photoeye is a BOOL tag and your specification says: "If photoeye is off then…"	`IF NOT photoeye THEN...`
expression1 & *expression2*	If photoeye is a BOOL tag, temp is a DINT tag, and your specification says: "If photoeye is on and temp is less than 100° then…".	`IF photoeye & (temp<100) THEN...`
expression1 OR *expression2*	If photoeye is a BOOL tag, temp is a DINT tag, and your specification says: "If photoeye is on or temp is less than 100° then…".	`IF photoeye OR (temp<100) THEN...`
expression1 XOR *expression2*	If photoeye1 and photoeye2 are BOOL tags and your specification says: "If: • photoeye1 is on while photoeye2 is off or • photoeye1 is off while photoeye2 is on then…"	`IF photoeye1 XOR photoeye2 THEN...`
BOOLtag := *expression1* & *expression2*	If photoeye1 and photoeye2 are BOOL tags, open is a BOOL tag, and your specification says: "If photoeye1 and photoeye2 are both on, set open to true".	`open := photoeye1 & photoeye2;`

Use bitwise operators

Bitwise operators manipulate the bits within a value based on two values.

For	Use this operator	Optimal Data Type
bitwise AND	&, AND	DINT
bitwise OR	OR	DINT
bitwise exclusive OR	XOR	DINT
bitwise complement	NOT	DINT

For example:

Use this format	Example	
	For this situation	You'd write
value1 operator value2	If input1, input2, and result1 are DINT tags and your specification says: "Calculate the bitwise result of input1 and input2. Store the result in result1."	`result1 := input1 AND input2;`

Determine the order of execution

The operations you write into an expression are performed in a prescribed order, not necessarily from left to right.

- Operations of equal order are performed from left to right.
- If an expression contains multiple operators or functions, group the conditions in parenthesis "( )". This ensures the correct order of execution and makes it easier to read the expression.

Order	Operation
1.	( )
2.	function (…)
3.	**
4.	– (negate)
5.	NOT
6.	*, /, MOD
7.	+, - (subtract)
8.	<, <=, >, >=
9.	=, <>
10.	&, AND
11.	XOR
12.	OR

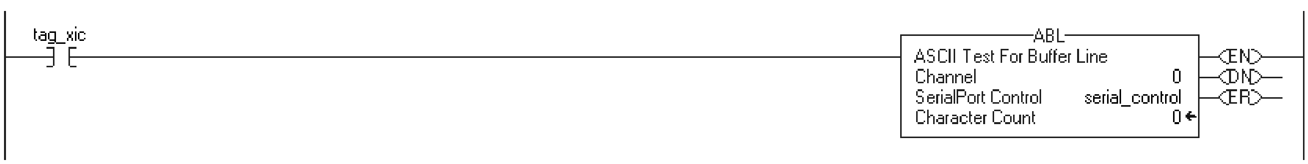
Instructions

Structured text statements can also be instructions. See the Locator Table at the beginning of this manual for a list of the instructions available in structured text. A structured text instruction executes each time it is scanned. A structured text instruction within a construct executes every time the conditions of the construct are true. If the conditions of the construct are false, the statements within the construct are not scanned. There is no rung-condition or state transition that triggers execution.

This differs from function block instructions that use EnableIn to trigger execution. Structured text instructions execute as if EnableIn is always set.

This also differs from relay ladder instructions that use rung-condition-in to trigger execution. Some relay ladder instructions only execute when rung-condition-in toggles from false to true. These are transitional relay ladder instructions. In structured text, instructions will execute each time they are scanned unless you pre-condition the execution of the structured text instruction.

For example, the ABL instruction is a transitional instruction in relay ladder. In this example, the ABL instruction only executes on a scan when tag_xic transitions from cleared to set. The ABL instruction does not execute when tag_xic stays set or when tag_xic is cleared.



In structured text, if you write this example as:

```
IF tag_xic THEN ABL(0,serial_control);

END_IF;
```

the ABL instruction will execute every scan that tag_xic is set, not just when tag_xic transitions from cleared to set.

If you want the ABL instruction to execute only when tag_xic transitions from cleared to set, you have to condition the structured text instruction. Use a one shot to trigger execution.

```
osri_1.InputBit := tag_xic;
OSRI(osri_1);

IF (osri_1.OutputBit) THEN
     ABL(0,serial_control);
END_IF;
```

Constructs

Constructs can be programmed singly or nested within other constructs.

If you want to	Use this construct	Available in these languages	See page
do something if or when specific conditions occur	IF...THEN	structured text	410
select what to do based on a numerical value	CASE...OF	structured text	413
do something a specific number of times before doing anything else	FOR...DO	structured text	416
keep doing something as long as certain conditions are true	WHILE...DO	structured text	419
keep doing something until a condition is true	REPEAT...UNTIL	structured text	422

Some key words are reserved for future use

These constructs are not available:

- GOTO
- REPEAT

RSLogix 5000 software will not let you use them.

IF...THEN

Use IF…THEN to do something if or when specific conditions occur.

Operands:

```
IF bool_expression THEN
    <statement>;
END_IF;
```

Structured Text

Operand	Type	Format	Enter
bool_ expression	BOOL	tag expression	BOOL tag or expression that evaluates to a BOOL value (BOOL expression)

Description:

The syntax is:

```
IF bool_expression1 THEN
     <statement >;          statements to execute when
                            bool_expression1 is true
        .
        .
        .
ELSIF bool_expression2 THEN                         optional
     <statement>;           statements to execute when
                            bool_expression2 is true
        .
        .
        .
ELSE                                                optional
     <statement>;           statements to execute when
                            both expressions are false
        .
        .
        .
END_IF;
```

To use ELSIF or ELSE, follow these guidelines:

1. To select from several possible groups of statements, add one or more ELSIF statements.
   - Each ELSIF represents an alternative path.
   - Specify as many ELSIF paths as you need.
   - The controller executes the first true IF or ELSIF and skips the rest of the ELSIFs and the ELSE.
2. To do something when all of the IF or ELSIF conditions are false, add an ELSE statement.

This table summarizes combinations of IF, THEN, ELSIF, and ELSE.

If you want to	And	Then use this construct
do something if or when conditions are true	do nothing if conditions are false	IF…THEN
	do something else if conditions are false	IF…THEN…ESLE
choose from alternative statements (or groups of statements) based on input conditions	do nothing if conditions are false	IF…THEN…ELSIF
	assign default statements if all conditions are false	IF…THEN…ELSIF…ELSE

Example 1: IF…THEN

If you want this	Enter this structured text
IF rejects > 3 then	`IF rejects > 3 THEN`
conveyor = off (0)	`conveyor := 0;`
alarm = on (1)	`alarm := 1;`
	`END_IF;`

Example 2: IF…THEN…ELSE

If you want this	Enter this structured text
If conveyor direction contact = forward (1) then	`IF conveyor_direction THEN`
light = off	`light := 0;`
Otherwise light = on	`ELSE`
	`light [:=] 1;`
	`END_IF;`

The [:=] tells the controller to clear light whenever the controller:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for Automatic reset (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

Example 3: IF…THEN…ELSIF

If you want this	Enter this structured text
If sugar low limit switch = low (on) and sugar high limit switch = not high (on) then	`IF Sugar.Low & Sugar.High THEN`
inlet valve = open (on)	`Sugar.Inlet [:=] 1;`
Until sugar high limit switch = high (off)	`ELSIF NOT(Sugar.High) THEN`
	`Sugar.Inlet := 0;`
	`END_IF;`

The [:=] tells the controller to clear Sugar.Inlet whenever the controller:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for Automatic reset (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

Example 4: IF…THEN…ELSIF…ELSE

If you want this	Enter this structured text
If tank temperature > 100	`IF tank.temp > 200 THEN`
then pump = slow	`pump.fast :=1; pump.slow :=0; pump.off :=0;`
If tank temperature > 200	`ELSIF tank.temp > 100 THEN`
then pump = fast	`pump.fast :=0; pump.slow :=1; pump.off :=0;`
otherwise pump = off	`ELSE`
	`pump.fast :=0; pump.slow :=0; pump.off :=1;`
	`END_IF;`

CASE...OF

Use CASE to select what to do based on a numerical value.

Operands:

```
CASE numeric_expression OF
    selector1: statement;
    selectorN: statement;
ELSE
    statement;
END_CASE;
```

Structured Text

Operand	Type	Format	Enter
numeric_expression	SINT INT DINT REAL	tag expression	tag or expression that evaluates to a number (numeric expression)
selector	SINT INT DINT REAL	immediate	same type as *numeric_expression*

IMPORTANT If you use REAL values, use a range of values for a selector because a REAL value is more likely to be within a range of values than an exact match of one, specific value.

Description:

The syntax is:

```
CASE numeric_expression OF
    selector1 :  <statement>;
                 .
                 .
                 .
    selector2 :  <statement>;
                 .
                 .
                 .
    selector3 :  <statement>;
                 .
                 .
                 .
ELSE
                 <statement>;
                 .
                 .
                 .
END_CASE;
```

specify as many alternative selector values (paths) as you need

statements to execute when numeric_expression = selector1

statements to execute when numeric_expression = selector2

statements to execute when numeric_expression = selector3

optional

statements to execute when numeric_expression ≠ any selector

See the table on the next page for valid selector values.

The syntax for entering the selector values is:

When selector is	Enter
one value	value: statement
multiple, distinct values	value1, value2, valueN : <statement> Use a comma (,) to separate each value.
a range of values	value1...valueN : <statement> Use two periods (..) to identify the range.
distinct values plus a range of values	valuea, valueb, value1...valueN : <statement>

The CASE construct is similar to a switch statement in the C or C++ programming languages. However, with the CASE construct the controller executes only the statements that are associated with the first matching selector value. Execution always breaks after the statements of that selector and goes to the END_CASE statement.

Example

If you want this	Enter this structured text
If recipe number = 1 then	`CASE recipe_number OF`
Ingredient A outlet 1 = open (1)	`1: Ingredient_A.Outlet_1 :=1;`
Ingredient B outlet 4 = open (1)	`Ingredient_B.Outlet_4 :=1;`
If recipe number = 2 or 3 then	`2,3: Ingredient_A.Outlet_4 :=1;`
Ingredient A outlet 4 = open (1)	`Ingredient_B.Outlet_2 :=1;`
Ingredient B outlet 2 = open (1)	
If recipe number = 4, 5, 6, or 7 then	`4..7: Ingredient_A.Outlet_4 :=1;`
Ingredient A outlet 4 = open (1)	`Ingredient_B.Outlet_2 :=1;`
Ingredient B outlet 2 = open (1)	
If recipe number = 8, 11, 12, or 13 then	`8,11..13 Ingredient_A.Outlet_1 :=1;`
Ingredient A outlet 1 = open (1)	`Ingredient_B.Outlet_4 :=1;`
Ingredient B outlet 4 = open (1)	
Otherwise all outlets = closed (0)	`ELSE`
	`Ingredient_A.Outlet_1 [:=]0;`
	`Ingredient_A.Outlet_4 [:=]0;`
	`Ingredient_B.Outlet_2 [:=]0;`
	`Ingredient_B.Outlet_4 [:=]0;`
	`END_CASE;`

The [:=] tells the controller to also clear the outlet tags whenever the controller:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for Automatic reset (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

FOR…DO

Use the FOR…DO loop to do something a specific number of times before doing anything else.

Operands:

```
FOR count:= initial_value TO
final_value BY increment DO
    <statement>;
END_FOR;
```

Structured Text

Operand	Type	Format	Description
count	SINT INT DINT	tag	tag to store count position as the FOR…DO executes
initial_ value	SINT INT DINT	tag expression immediate	must evaluate to a number specifies initial value for count
final_ value	SINT INT DINT	tag expression immediate	specifies final value for count, which determines when to exit the loop
increment	SINT INT DINT	tag expression immediate	(*optional*) amount to increment count each time through the loop If you don't specify an increment, the count increments by 1.

IMPORTANT Make sure that you do not iterate within the loop too many times in a single scan.

- The controller does not execute any other statements in the routine until it completes the loop.
- If the time that it takes to complete the loop is greater than the watchdog timer for the task, a major fault occurs.
- Consider using a different construct, such as IF...THEN.

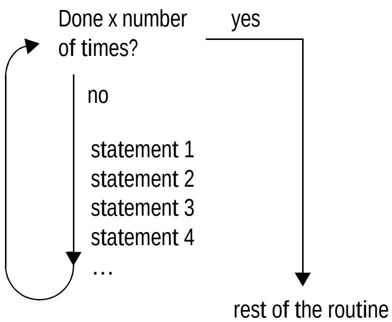
Description: The syntax is:

```
FOR count := initial_value
   TO final_value
optional {   BY increment
   DO
   <statement>;
optional {   IF bool_expression THEN
      EXIT;
   END_IF;
END_FOR;
```

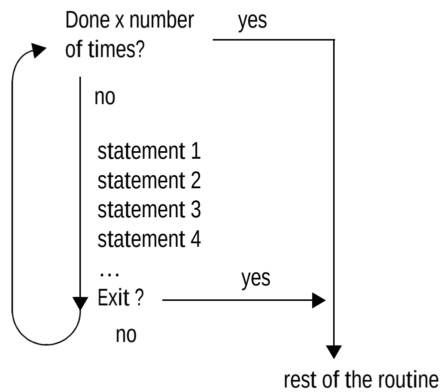
If you don't specify an increment, the loop increments by 1.

If there are conditions when you want to exit the loop early, use other statements, such as an IF...THEN construct, to condition an EXIT statement.

These diagrams show how a FOR...DO loop executes and how an EXIT statement leaves the loop early.



The FOR…DO loop executes a specific number of times.



To stop the loop before the count reaches the last value, use an EXIT statement.

Example 1:

If you want this	Enter this structured text

Clear bits 0 - 31 in an array of BOOLs:
1. Initialize the subscript tag to 0.
2. Clear array[subscript] . For example, when subscript = 5, clear array[5].
3. Add 1 to subscript.
4. If subscript is £ to 31, repeat 2 and 3.
Otherwise, stop.
	`For subscript:=0 to 31 by 1 do`
	`    array[subscript] := 0;`
	`End_for;`

Example 2:

<table>
<tr><th>If you want this</th><th>Enter this structured text</th></tr>
<tr><td>
A user-defined data type (structure) stores this information about an item in your inventory:

Barcode ID of the item (string data type)
Quantity in stock of the item (DINT data type)

An array of the above structure contains an element for each different item in your inventory. You want to search the array for a specific product (use its bar code) and determine the quantity that is in stock.

Get the size (number of items) of the Inventory array and store the result in Inventory_Items (DINT tag).
Initialize the position tag to 0.
If Barcode matches the ID of an item in the array, then:
a. Set the Quantity tag = Inventory[position].Qty. This produces the quantity in stock of the item.
b. Stop.
Barcode is a string tag that stores the bar code of the item for which you are searching. For example, when position = 5, compare Barcode to Inventory[5].ID.
Add 1 to position.
If position is £ to (Inventory_Items -1), repeat 3 and 4. Since element numbers start at 0, the last element is 1 less than the number of elements in the array.
Otherwise, stop.

</td><td>
<pre>
SIZE(Inventory,0,Inventory_Items);
For position:=0 to Inventory_Items - 1 do
    If Barcode = Inventory[position].ID then
        Quantity := Inventory[position].Qty;
        Exit;
    End_if;
End_for;
</pre>
</td></tr>
</table>

WHILE…DO

Use the WHILE…DO loop to keep doing something as long as certain conditions are true.

```
WHILE bool_expression DO
    <statement>;
END_WHILE;
```

Operands:

Structured Text

Operand	Type	Format	Enter
bool_ expression	BOOL	tag expression	BOOL tag or expression that evaluates to a BOOL value

IMPORTANT Make sure that you do not iterate within the loop too many times in a single scan.

- The controller does not execute any other statements in the routine until it completes the loop.
- If the time that it takes to complete the loop is greater than the watchdog timer for the task, a major fault occurs.
- Consider using a different construct, such as IF...THEN.

Description:

The syntax is:

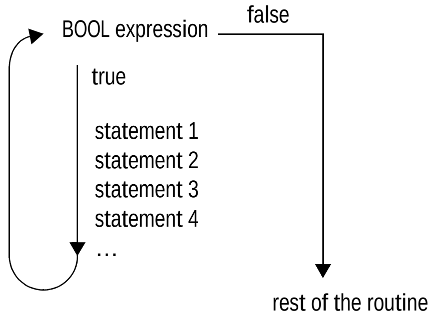
```
WHILE bool_expression1 DO
   <statement>;
   IF bool_expression2 THEN
     EXIT;
   END_IF;
END_WHILE;
```

statements to execute while bool_expression1 is true

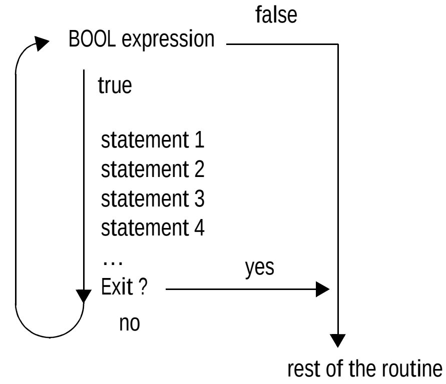
optional

If there are conditions when you want to exit the loop early, use other statements, such as an IF...THEN construct, to condition an EXIT statement.

These diagrams show how a WHILE...DO loop executes and how an EXIT statement leaves the loop early.



While the *`bool_expression`* is true, the controller executes only the statements within the WHILE…DO loop.



To stop the loop before the conditions are true, use an EXIT statement.

Example 1:

If you want this	Enter this structured text

The WHILE...DO loop evaluates its conditions first. If the conditions are true, the controller then executes the statements within the loop.
 	
This differs from the REPEAT...UNTIL loop because the REPEAT...UNTIL loop executes the statements in the construct and then determines if the conditions are true before executing the statements again. The statements in a REPEAT...UNTIL loop are always executed at least once. The statements in a WHILE...DO loop might never be executed.
	`pos := 0;`
	
	`While ((pos <= 100) & structarray[pos].value <> targetvalue)) do`
	
	`    pos := pos + 2;`
	
	`    String_tag.DATA[pos] := SINT_array[pos];`
	
	`end_while;`

Example 2:

If you want this	Enter this structured text

Move ASCII characters from a SINT array into a string tag. (In a SINT array, each element holds one character.) Stop when you reach the carriage return.
 	
1. Initialize Element_number to 0.
2. Count the number of elements in SINT_array (array that contains the ASCII characters) and store the result in SINT_array_size (DINT tag).
3. If the character at SINT_array[element_number] = 13 (decimal value of the carriage return), then stop.
4. Set String_tag[element_number] = the character at SINT_array[element_number].
5. Add 1 to element_number. This lets the controller check the next character in SINT_array.
6. Set the Length member of String_tag = element_number. (This records the number of characters in String_tag so far.)
7. If element_number = SINT_array_size, then stop. (You are at the end of the array and it does not contain a carriage return.)
8. Go to 3.
	`element_number := 0;`
	`SIZE(SINT_array, 0, SINT_array_size);`
	`While SINT_array[element_number] <> 13 do`
	    `String_tag.DATA[element_number] := SINT_array[element_number];`
	    `element_number := element_number + 1;`
	    `String_tag.LEN := element_number;`
	    `If element_number = SINT_array_size then`
	        `exit;`
	    `end_if;`
	`end_while;`

REPEAT…UNTIL

Use the REPEAT…UNTIL loop to keep doing something until conditions are true.

Operands:

```
REPEAT
    <statement>;
UNTIL bool_expression
END_REPEAT;
```

Structured Text

Operand	Type	Format	Enter
bool_ expression	BOOL	tag expression	BOOL tag or expression that evaluates to a BOOL value (BOOL expression)

IMPORTANT Make sure that you do not iterate within the loop too many times in a single scan.

- The controller does not execute any other statements in the routine until it completes the loop.
- If the time that it takes to complete the loop is greater than the watchdog timer for the task, a major fault occurs.
- Consider using a different construct, such as IF...THEN.

Description:

The syntax is:

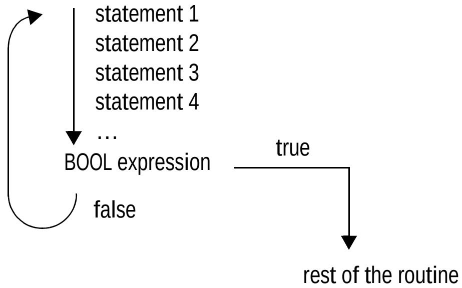
```
REPEAT
   <statement>;

   IF bool_expression2 THEN
     EXIT;
   END_IF;

UNTIL bool_expression1
END_REPEAT;
```

statements to execute while bool_expression1 is false

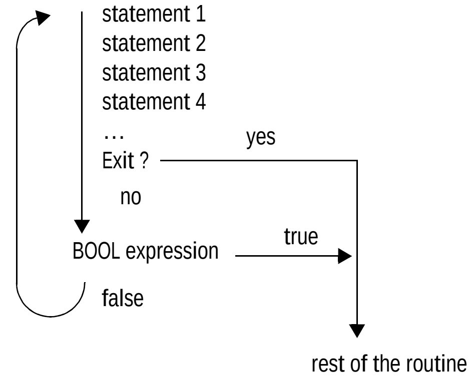
optional

If there are conditions when you want to exit the loop early, use other statements, such as an IF...THEN construct, to condition an EXIT statement.

These diagrams show how a REPEAT...UNTIL loop executes and how an EXIT statement leaves the loop early.



While the *bool_expression* is false, the controller executes only the statements within the REPEAT…UNTIL loop.



To stop the loop before the conditions are false, use an EXIT statement.

Example 1:

If you want this	Enter this structured text

The REPEAT...UNTIL loop executes the statements in the construct and then determines if the conditions are true before executing the statements again.
 	
This differs from the WHILE...DO loop because the WHILE...DO The WHILE...DO loop evaluates its conditions first. If the conditions are true, the controller then executes the statements within the loop. The statements in a REPEAT...UNTIL loop are always executed at least once. The statements in a WHILE...DO loop might never be executed.
	`pos := -1;`
	`REPEAT`
	`    pos := pos + 2;`
	`UNTIL ((pos = 101) OR (structarray[pos].value = targetvalue))`
	`end_repeat;`

Example 2:

If you want this	Enter this structured text

Move ASCII characters from a SINT array into a string tag. (In a SINT array, each element holds one character.) Stop when you reach the carriage return.
1. Initialize Element_number to 0.
2. Count the number of elements in SINT_array (array that contains the ASCII characters) and store the result in SINT_array_size (DINT tag).
3. Set String_tag[element_number] = the character at SINT_array[element_number].
4. Add 1 to element_number. This lets the controller check the next character in SINT_array.
5. Set the Length member of String_tag = element_number. (This records the number of characters in String_tag so far.)
6. If element_number = SINT_array_size, then stop. (You are at the end of the array and it does not contain a carriage return.)
7. If the character at SINT_array[element_number] = 13 (decimal value of the carriage return), then stop.
Otherwise, go to 3.
	`element_number := 0;`
	`SIZE(SINT_array, 0, SINT_array_size);`
	`Repeat`
	`String_tag.DATA[element_number] := SINT_array[element_number];`
	`element_number := element_number + 1;`
	`String_tag.LEN := element_number;`
	`If element_number = SINT_array_size then`
	`exit;`
	`end_if;`
	`Until SINT_array[element_number] = 13`
	`end_repeat;`

Comments

To make your structured text easier to interpret, add comments to it.

- Comments let you use plain language to describe how your structured text works.
- Comments do not affect the execution of the structured text.

Structured text comments are downloaded into controller memory and are available for upload. To add comments to your structured text:

To add a comment	Use one of these formats
on a single line	//comment (*comment*) /*comment*/
at the end of a line of structured text	
within a line of structured text	(*comment*) /*comment*/
that spans more than one line	(*start of comment . . . end of comment*) /*start of comment . . . end of comment*/

For example:

Format	Example

//comment	**At the beginning of a line**
	//Check conveyor belt direction
	IF conveyor_direction THEN...
	
	At the end of a line
	ELSE //If conveyor isn’t moving, set alarm light
	light := 1;
	END_IF;

(*comment*)	Sugar.Inlet[:=]1;(*open the inlet*) IF Sugar.Low (*low level LS*)& Sugar.High (*high level LS*)THEN... (*Controls the speed of the recirculation pump. The speed depends on the temperature in the tank.*) IF tank.temp > 200 THEN...
/*comment*/	Sugar.Inlet:=0;/*close the inlet*/ IF bar_code=65 /*A*/ THEN... /*Gets the number of elements in the Inventory array and stores the value in the Inventory_Items tag*/ SIZE(Inventory,0,Inventory_Items);

Numerics

A

B

C

D

E

F

I

J

K

L

M

N

O

How Are We Doing?

Your comments on our technical publications will help us serve you better in the future.
Thank you for taking the time to provide us feedback.

You can complete this form and mail (or fax) it back to us or email us at
RADocumentComments@ra.rockwell.com

Pub. Title/Type Logix5000 Controllers Motion Instructions

Cat. No. ______ Pub. No. 1756-RM007H-EN-P Pub. Date December 2006 Part No. 953030-45

Please complete the sections below. Where applicable, rank the feature (1=needs improvement, 2=satisfactory, and 3=outstanding).

Overall Usefulness	1	2	3	How can we make this publication more useful for you?
Completeness (all necessary information is provided)	1	2	3	Can we add more information to help you? procedure/step · illustration · feature example · guideline · other explanation · definition
Technical Accuracy (all provided information is correct)	1	2	3	Can we be more accurate? text · illustration
Clarity (all provided information is easy to understand)	1	2	3	How can we make things clearer?
Other Comments				You can add additional comments on the back of this form.

Your Name ______

Your Title/Function ______

Location/Phone ______

Would you like us to contact you regarding your comments?

___No, there is no need to contact me

___Yes, please call me

___Yes, please email me at ______

___Yes, please contact me via ______

Return this form to: Rockwell Automation Technical Communications, 1 Allen-Bradley Dr., Mayfield Hts., OH 44124-9705

Fax: 440-646-3525 Email: RADocumentComments@ra.rockwell.com

PLEASE FASTEN HERE (DO NOT STAPLE)

Other Comments

PLEASE FOLD HERE

PLEASE REMOVE

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 18235 CLEVELAND OH

POSTAGE WILL BE PAID BY THE ADDRESSEE

Allen-Bradley
RELIANCE ELECTRIC
DODGE
ROCKWELL SOFTWARE

Rockwell Automation

1 ALLEN-BRADLEY DR
MAYFIELD HEIGHTS OH 44124-9705

Rockwell Automation Support

Rockwell Automation provides technical information on the Web to assist you in using its products. At http://support.rockwellautomation.com, you can find technical manuals, a knowledge base of FAQs, technical and application notes, sample code and links to software service packs, and a MySupport feature that you can customize to make the best use of these tools.

For an additional level of technical phone support for installation, configuration, and troubleshooting, we offer TechConnect Support programs. For more information, contact your local distributor or Rockwell Automation representative, or visit http://support.rockwellautomation.com.

Installation Assistance

If you experience a problem with a hardware module within the first 24 hours of installation, please review the information that's contained in this manual. You can also contact a special Customer Support number for initial help in getting your module up and running.

United States	1.440.646.3223 Monday – Friday, 8am – 5pm EST
Outside United States	Please contact your local Rockwell Automation representative for any technical support issues.

New Product Satisfaction Return

Rockwell tests all of its products to ensure that they are fully operational when shipped from the manufacturing facility. However, if your product is not functioning, it may need to be returned.

United States	Contact your distributor. You must provide a Customer Support case number (see phone number above to obtain one) to your distributor in order to complete the return process.
Outside United States	Please contact your local Rockwell Automation representative for return procedure.

www.rockwellautomation.com

Power, Control and Information Solutions Headquarters

Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe/Middle East/Africa: Rockwell Automation, Vorstlaan/Boulevard du Souverain 36, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, Level 14, Core F, Cyberport 3, 100 Cyberport Road, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publication 1756-RM007H-EN-P - December 2006 PN 953030-45
Supersedes Publication 1756-RM007G-EN-P - May 2005

Allen-Bradley

Logix5000 Controllers Motion Instructions

Reference Manual