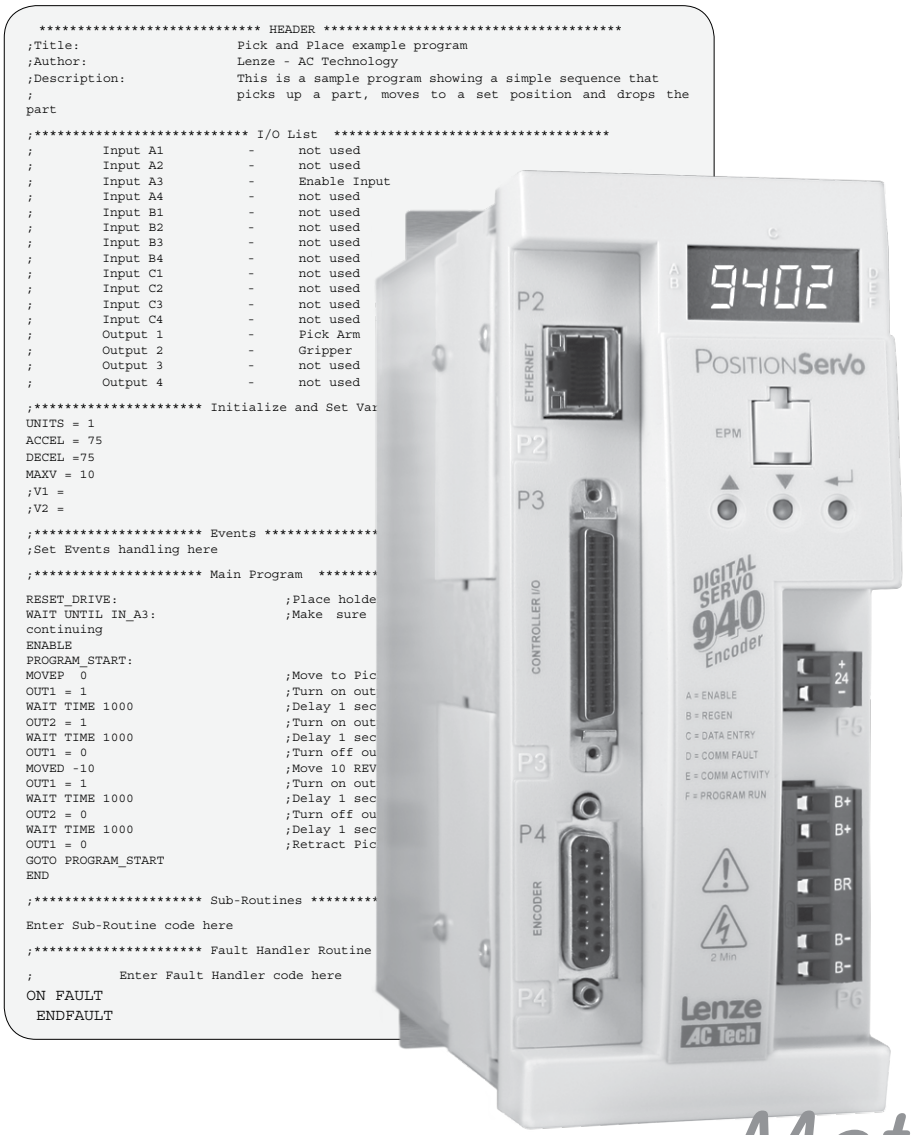
Lenze

AC Tech



MotionView®

OnBoard

PositionServo with MVOB

Programming Manual

Valid for Hardware Version 2

Contents

Contents

About These Instructions

This documentation applies to the programming of the PositionServo drive with model numbers ending in S or M. This documentation should be used in conjunction with the PositionServo User Manual (Document S94H201) that shipped with the drive. These documents should be read in their entirety as they contain important technical data and describe the installation and operation of the drive.

Safety Warnings

Take note of these safety warnings and those in the PositionServo User Manual and related documentation.

WARNING! Hazard of unexpected motor starting!

When using MotionView, or otherwise remotely operating the PositionServo drive, the motor may start unexpectedly, which may result in damage to equipment and/or injury to personnel. Make sure the equipment is free to operate and that all guards and covers are in place to protect personnel.

All safety information contained in these Programming Instructions is formatted with this layout including an icon, signal word and description:

Signal Word! (Characterizes the severity of the danger)

Safety Information (describes the danger and informs on how to proceed)

Table 1: Pictographs used in these Instructions

Icon		Signal Words	
	Warning of hazardous electrical voltage	**DANGER!**	Warns of **impending danger**. Consequences if disregarded: Death or severe injuries.
	Warning of a general danger	**WARNING!**	Warns of **potential, very hazardous situations**. Consequences if disregarded: Death or severe injuries.
	Warning of damage to equipment	**STOP!**	Warns of **potential damage to material and equipment**. Consequences if disregarded: Damage to the controller/ drive or its environment.
	Information	**NOTE**	Designates a general, useful note. If the note is observed then handling the controller/drive system is made easier.

Related Documents

The documentation listed in Table 2 contains information relevant to the operation and programming of the PositionServo drive. To obtain the latest documentation, visit the Technical Library at http://www.lenzeamericas.com.

Table 2: Reference Documentation

Document #	Description
S94H201	PositionServo (with MVOB) User Manual
PM94H201	PositionServo (with MVOB) Programming Manual
P94MOD01	Position Servo ModBus RTU and ModBus TCP/IP
P94CAN01	PositionServo CANopen Communications Reference Guide
P94DVN01	PositionServo DeviceNet Communications Reference Guide
P94ETH01	PositionServo EtherNet/IP Communications Reference Guide
P94PFB01	PositionServo PROFIBUS DP Communications Reference Guide

1. Introduction

1.1 Definitions

Included herein are definitions of several terms used throughout this programming manual and the PositionServo user manual.

PositionServo: The PositionServo is a programmable digital drive/motion controller, that can be configured as a stand alone programmable motion controller, or as a high performance torque, velocity or position amplifier for centralized control systems. The PositionServo family of drives includes the 940 Encoder-based drive and the 941 Resolver-based drive.

MotionView: MotionView is a universal communication and configuration software that is utilized by the PositionServo drive family. Starting with revision 4.xx, drives will have MotionView OnBoard (MVOB) built into the drive. MotionView has an automatic self-configuration mechanism that recognizes what drive it is connected to and configures the tool set accordingly. The MotionView platform is divided up into three sections or windows, the “Parameter Tree Window”, the “Parameter View Window” and the “Message Window”. Refer to Section 1.3 for more detail.

MotionView OnBoard (MVOB): MotionView OnBoard is the embedded version of MotionView software in PositionServo drives with a part number ending in ES, RS, EM or RM.

SimpleMotion Language (SML): SML is the programming language utilized by MotionView. The SML interface within the MotionView software provides a very flexible development environment for creating solutions to motion applications. The SML programming statements allow the programmer to create complex and intelligent motion, process I/O, perform complex logic decision making, execute program branching, utilize timed event processes, as well as a number of other functions common to the majority of motion control and servo applications.

User Program (or Indexer Program): This is the SML program, developed by the user to describe the programmatic behavior of the PositionServo drive. The User Program can be stored in a text file on your PC as well as in the PositionServo’s EPM memory. The User Program needs to be compiled (translated) into binary form with the aid of the MotionView Studio tools before the PositionServo can execute it.

MotionView Studio: MotionView Studio is the front end programming interface of the MotionView platform. It is a tool suite containing all the software tools needed to program and debug the PositionServo. These tools include a full-screen text editor, a program compiler, status and monitoring utilities, an online oscilloscope and a debug function that allows the user to step through the program during program development.

WARNING!

- Hazard of unexpected motor starting! When using the MotionView software, or otherwise remotely operating the PositionServo drive, the motor may start unexpectedly, which may result in damage to equipment and/or injury to personnel. Make sure the equipment is free to operate safely, and that all guards and covers are in place to protect personnel.
- Hazard of electrical shock! Circuit potentials are up to 480 VAC above earth ground. Avoid direct contact with the internal printed circuit boards or with circuit elements to prevent the risk of serious injury or fatality. Disconnect incoming power and wait 60 seconds before servicing drive. Capacitors retain charge after power is removed.

NOTE

To run MotionView OnBoard (MVOB) on a Mac OS, run the PC emulation tool first.

1.2 Programming Flowchart

MotionView utilizes a BASIC-like programming structure referred to as SimpleMotion Programming Language (SML). SML is a quick and easy way to create powerful motion applications.

With SML the programmer describes his system's motion, I/O processing and process flow using the SML structured code. The programming language includes a full set of arithmetic and logical statements that allow the user to perform mathematical calculations and comparisons of variables and apply the results within their application.

Before the PositionServo drive can execute the user's program, the program must first be compiled (translated) into binary machine code, and downloaded to the drive. Compiling the program is done by selecting the [Compile] button from the toolbar located within the indexer program folder. The user can also compile and download the program at the same time by selecting the [Load W Source] button from the toolbar. Once downloaded, the compiled program is stored in both the PositionServo's EPM memory and the internal flash memory. Figure 1 illustrates the flow of the program preparation process.

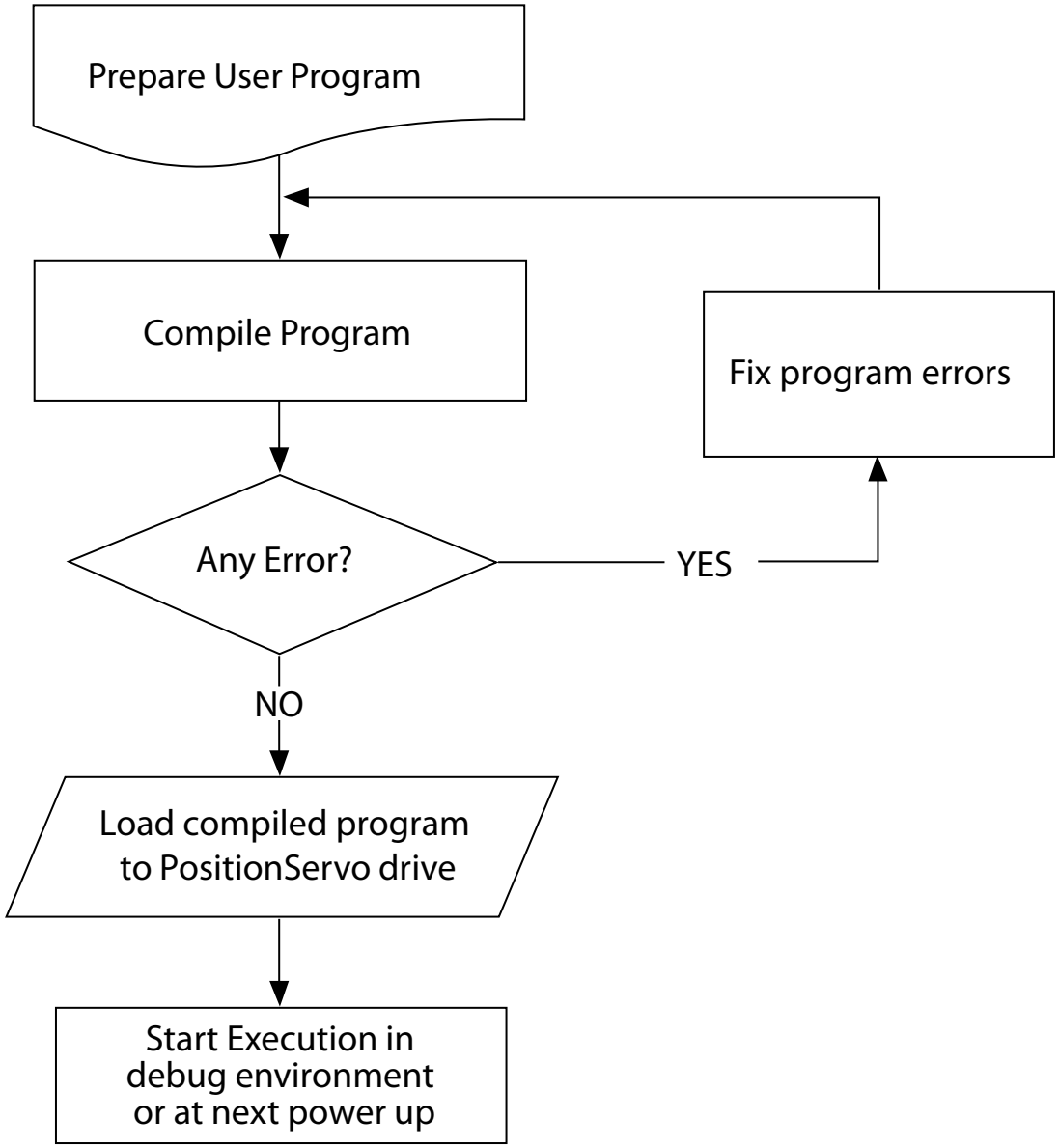


Figure 1: Program Preparation

1.3 MotionView / MotionView Studio

There are two versions of MotionView Software. The current version of MotionView resides inside the drive's memory and is referred to as "MotionView on Board" or MVOB. Previous versions were supplied as a PC-installed software package and were referred to simply as MotionView. This manual refers only to the MotionView OnBoard software. MVOB drives are identified by the model number ending in either an 'S' or an 'M'.

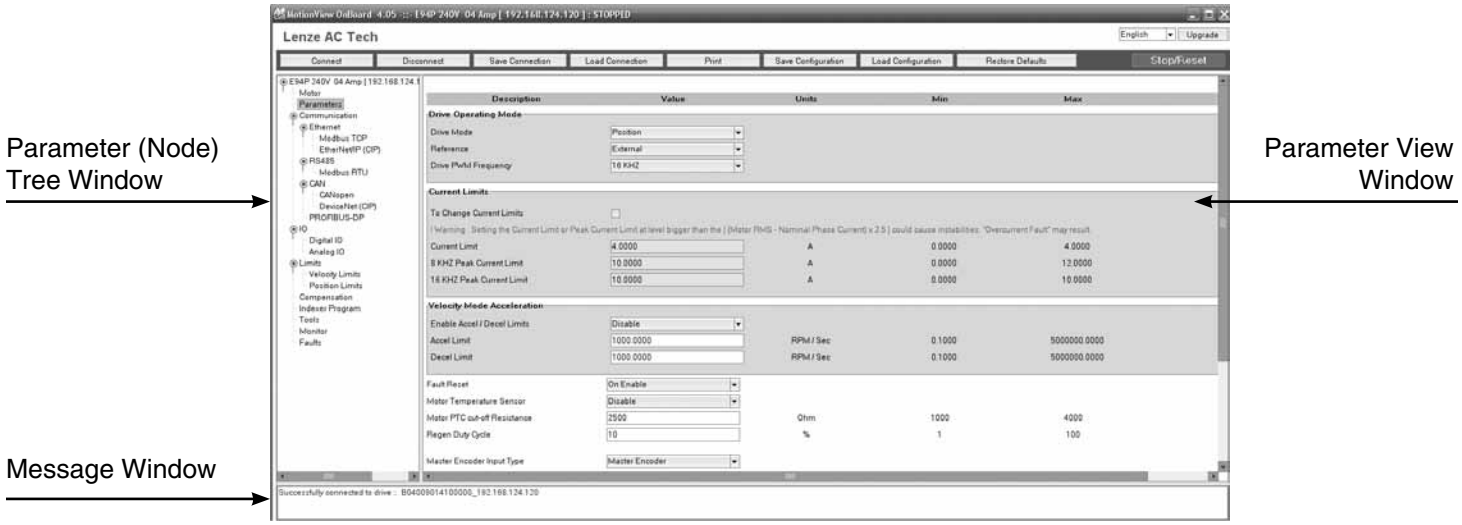


Figure 2: MotionView OnBoard Parameters Display

MotionView is the universal programming software used to communicate with and configure the PositionServo drive. The MotionView platform is segmented into three windows. The first window is the **"Parameter Tree Window"**. This window is used much like Windows Explorer. The various parameter groups for the drive are represented here as folders or files. Once the desired parameter group file is selected, all of the corresponding parameters within that parameter group will appear in the second window, the **"Parameter View Window"**. The user can then enable, disable or edit drive features or parameters from the "Parameter View Window". The third window is the **"Message Window"**. This window is located at the bottom of the screen and will display communication status and errors.

NOTE

To run MotionView OnBoard (MVOB) on a Mac OS, run the PC emulation tool first.

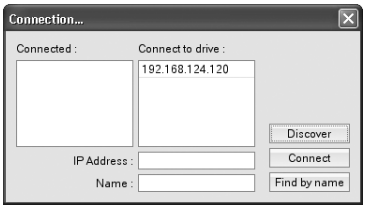
1.3.1 Main Toolbar

The most commonly used functions of MotionView are accessible via the Main Toolbar as illustrated in Figure 3. If a function icon is greyed out that denotes the function is presently unavailable. A function may be unavailable because a drive is not physically connected to the network or the present set-up and operation of the drive prohibits access to that function. Use the pull-down menu in the top right-hand corner to select the language. [English] is the default language.

Figure 3: Main Toolbar

Connect

Build a connection list of the drive(s) to communicate with on the network. Build the connection list by using any one of these three methods:



[Discover] button automatically discovers all drives on the network that are available for connection. Once drives have been discovered they are listed in the 'Connect to drive' list box. To connect one or more drives highlight their IP address in this window and press the [Connect] button. The [Ctrl] key on the keyboard can be used to select multiple drives for connection.

If the IP address of the drive to be connected is known, enter it in the IP Address dialog box and then select [Connect] to access the drive.

If a drive has previously been assigned a name (or text label) within its "Drive Name" parameter then this name can be used to subsequently connect to that drive. Enter the drive name into the "Name" dialog box and select [Find by name]. The IP address for that drive will then appear in the "Connect To Drive" list. The drive can now be connected by highlighting the IP address and pressing the [Connect] button.

Disconnect	Terminate connection to the drive selected (highlighted) in the Parameter (Node) Tree.
Save Connection	Save the connection parameters for all drives currently listed in the Parameter (Node) Tree window. This function saves MVOB communications setup for the project only (for quick reconnect of all project drives at a later date), it does not save the individual parameter and programming configuration of each drive.
Load Connection	Connect (Reconnect) to project. Opens a previously saved connection file and automatically connects to all drives listed within that file (provided they are available).
Print	Print a configuration report for the currently selected drive, containing all parameter set-up and programming information.
Save All	Saves the configuration file of the selected drive. All parameters, indexing program, I/O configuration and compensation gains will be saved within this file.
Load All	Load a saved configuration to the drive.
Default All	Set drive parameters back to factory default values. Note: has no effect on motor data or drive IP address.
Stop/Reset	Stops the drive execution and resets the drive.
Upgrade	Launches firmware upgrade utility.

1.3.2 Program Toolbar

To view the Program Toolbar, click on the [Indexer Program] folder in the Parameter (Node) Tree. This section contains a brief description of the programming tools: Compile, Load with Source, Load Without Source, Reload, Export, Import, Run, Reset, Pause, Step, Step Over and Clear. For detailed descriptions of the program toolbar functions refer to paragraph 1.4.

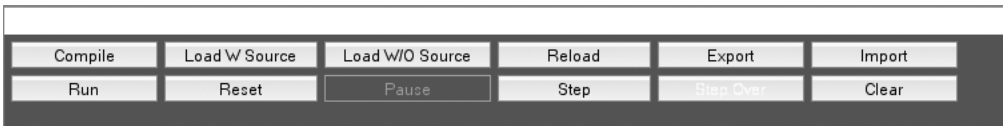


Figure 4: Program Toolbar

Compile	Perform compilation and check for syntax errors for the indexer program currently selected in the List View window.
Load W Source	Compile and Load Binary program and text source file to the PositionServo drive listed in the Parameter (Node) Tree.
Load WO Source	Compile and Load Binary program only (excluding text source file) to the PositionServo drive listed in the Parameter (Node) Tree.

Reload	Reload the text source file presently stored in the selected drive back into the MotionView Indexer program folder.
Export	Export text source file (User program). Saves a copy of the program from the Indexer Program folder as a text file on the PC.
Import	Import text source file (User program). Loads a program from a text file stored on the PC to the Indexer Program folder.
Run	Start/Continue Program execution. Refer to section 1.4 for full description and prior to operation.
Reset	Reset Drive. Disable drive, stop program execution, and return program processing to the beginning. Program will **not** restart program execution automatically.
Pause	Stop program execution on completion of the current statement being executed. **WARNING**: Pause button does not place the drive in a disable state or prevent execution of motion commands waiting on the motion stack.
Step	Execute each line of code in the program sequentially following on each press of the [Step] button. Include step to instructions contained within subroutines.
Step Over	Reserved for future use.
Clear	Clears the Indexer code.

WARNING
"Load W/O Source" will delete the text source file from both the indexer screen and the drive memory. The user must ensure they save a copy of the text source file to their PC before proceeding with this operation.

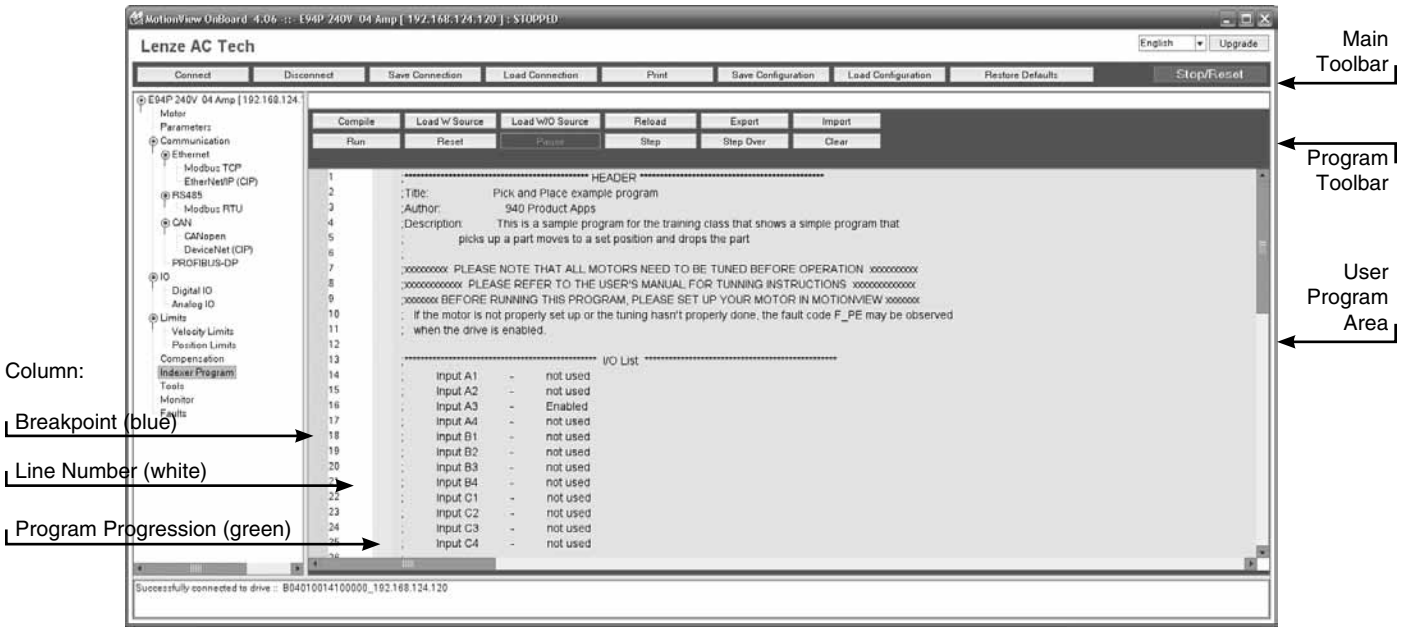


Figure 5: MotionView OnBoard Studio - Indexer Program Display

1.3.3 MotionView Studio - Indexer Program

The MotionView Studio provides a tool suite used by MotionView OnBoard to enter, compile, load and debug the user program. To view and develop the user program, select the [Indexer Program] folder in the Parameter (Node) Tree window. Once selected the program text editor screen and program toolbar are displayed. The program displayed in the text editor window is uploaded from the drive when the indexer folder is selected, any data not compiled to the drive or saved to PC file will be lost once this window is exited. Click anywhere in the Parameter View Window to edit the Indexer program.

Common Programming Actions

Load User program from the PC to the MotionView Indexer Program folder text editor window.

- Select **[Indexer Program]** in the Parameter (Node) Tree.
- Select **[Import]** on the program toolbar.

Select the program to import from the PC folder where it is located. This procedure loads the program from the file to the editor window. It doesn't load the program to the drive's memory.

Compile program and **Load** to the drive

- Select **[Indexer Program]** in the Parameter (Node) Tree.
- Select **[Load WO Source]** on the program toolbar to compile the program and load the compiled binary code to the PositionServo drive. A copy of the original source code is not stored to the drive's memory and therefore cannot be obtained from the drive subsequently. This feature can be used to protect the program from copy but the programmer must ensure that a copy of the program is safely stored to his PC.
- Select **[Load W Source]** on the program toolbar to compile the program and load the source code and the compiled binary file to the PositionServo drive. The original source code contained in the drive can be viewed whenever the drive is accessed through MotionView and the Indexer Program folder is opened.
- Select **[Compile]** to check syntax errors without loading the program to the drive. If the compiler finds any syntax error, further compilation is halted. Errors are reported in the message window at the bottom of the screen.

Save User program from MotionView to PC.

- Select **[Indexer Program]** in the Parameter (Node) Tree.
- Select **[Export]]** on the program toolbar.

Provide a name and folder location for the source file to be stored under. The program will be saved to the Windows "My Documents" folder by default.

Run User program in drive.

- Select **[Indexer Program]** in the Parameter (Node) Tree.
- Select **[Run]** on the program toolbar. Note all warnings contained within product manuals prior to running the user program.

Step Through the User program.

- Select **[Indexer Program]** in the Parameter (Node) Tree.
- Select **[Step]** on the program toolbar.

If [Step] is selected, the drive will execute the program one step at a time including subroutines. For the Step function to be used the drive must be in a 'Indexer program Stopped' condition. If Indexer program is running then Step functions are disabled. If the user program displayed in the Indexer program window does not match the program currently residing within the drive (last compiled and downloaded) then Step functions are also disabled.

Statement execution is tracked by a pointer located in the progression column of the program editor. The pointer indicates the next line of code to be executed. At each Step the pointer will disappear until the statement has been fully executed and will then reappear at the next statement.

Set **Breakpoint(s)** in the program

- Select **[Indexer Program]** in the Parameter (Node) Tree.
- Place the cursor in the 'Breakpoint' Column next to the line number on which a breakpoint is to be added.
- Right-click and select Add Breakpoint (or Clear Breakpoint).

A convenient way to debug a user program is to insert breakpoints at critical junctions throughout the program. These breakpoints are marked by a red plus sign (+) and stop the drive from executing further program statements once a breakpoint is reached, but do not disable the drive and the position variables. Once the program has stopped, the user can continue to run the program, step through the program or reset the program.

Pause program execution

- Select **[Indexer Program]** in the Parameter (Node) Tree.
- Select **[Pause]** on the program toolbar.

The program will stop after completing the current statement. Select **[Run]** or use Step functions to resume the program from the same point.

IMPORTANT!
The [Pause] button only stops the execution of the program code.
It does **not** stop motion or disable the drive.

Reset Program execution

- Select **[Indexer Program]** in the Parameter (Node) Tree.
- Select **[Reset]** on the program toolbar.

The program will be reset and the drive will be disabled. Variables within the drive are not cleared (reset) when program execution is reset. It is important that any variables used by the programmer are set to safe values at the start of the user program.

1.4 Programming Basics

The user program consists of statements which when executed will not only initiate motion but will also process the drives I/O and make decisions based on drive variables, calculations, and comparisons. Before motion can be initiated, certain drive and I/O parameters must be configured. When first getting started with PositionServo programming it is recommended that the following parameters be set within MotionView parameter folders to aid initial program creation.

Parameter setup

Select **[Parameter]** folder in the Parameter (Node) Tree window and set the following parameters.

Set the Drive Operating Mode:

- Select **[Drive mode]** from the Parameter View Window.
- Select **[Position]**, [Velocity], or [Torque] from the drop down menu depending on the mode the drive is to be operated in. In order to execute the examples contained in this section of the manual the drive will need to be in [Position] mode.

Set the [Reference] to [Internal]:

- Select **[Reference]** from the Parameter View Window.
- Select **[Internal]** from the pull down menu to select the user program as the source of the Torque, Velocity, or Position Reference.

Select **[Digital IO]** folder in the Parameter (Node) Tree window and set the following parameter.

Set the [Enable switch function] to [Inhibit]:

- Select **[Enable switch function]** from the Parameter View Window.
- Select **[Inhibit]** from the menu to allow the user program control of the enable / disable status of the drive. Input A3 will now act as a hardware inhibit.

I/O Configuration

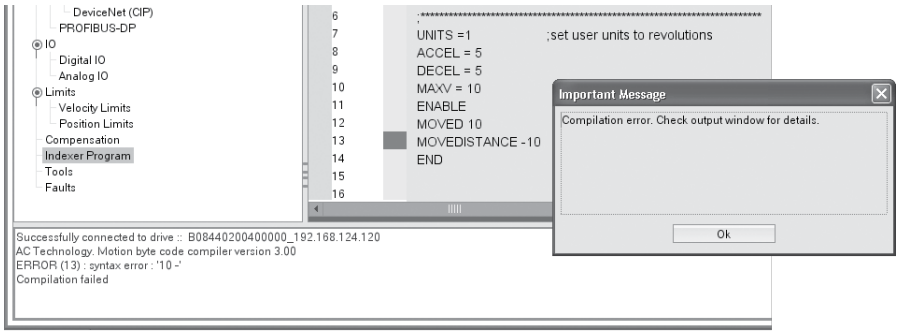
Input A3 is the Inhibit/Enable special purpose input. Refer to the PS User Manual (S94H201) for more information. Before executing any motion related statements, the drive must be enabled by executing "ENABLE" statement. "ENABLE" statement can only be accepted if input A3 is made. If at any time while drive is enabled A3 deactivates then the fault "F36" ("Drive Disabled") will result. This is a hardware safety feature.

Basic Motion Program

Select **[Indexer program]** from the Parameter (Node) Tree. The Parameter View window will display the current User Program stored in the drive. Note that if there is no valid program in the drive's memory the program editing window will be empty.

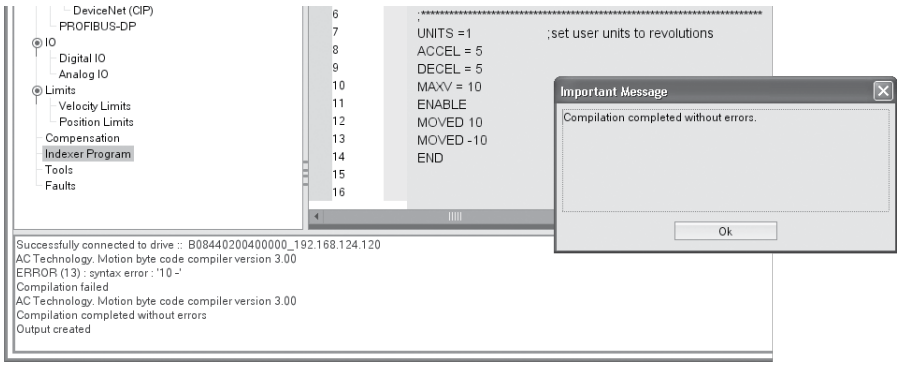
WARNING!
This program will cause motion. The motor should be disconnected from the application (free to rotate) or if a motor is connected, the shaft must be free to spin 10 revs forward and reverse from the location of the shaft at power up. Also, the machine must be capable of 10 RPS and an accel / decel of 5 RPSS.

In the program area, clear any existing program (save if required) and replace it with the following program:

Program	Compile	Resultant MotionView OnBoard Messages
UNITS=1 ACCEL = 5 DECEL = 5 MAXV = 10 ENABLE MOVED 10 MOVEDISTANCE -10 END	Compile Enter the program, then select [Compile] on the toolbar. After compilation is done, a "Compilation Error" message will appear.	

Click [OK] to dismiss the "Compliation error" dialog box. The cause of the compilation error will be displayed in the Message window, located at the bottom of the MotionView OnBoard screen. MotionView will also highlight the program line where the error occurred. In the example program above, in the green 'Program Progression' column there is a red box next to the "MOVEDISTANCE -10" statement.

The problem in this example is that "**MOVEDISTANCE**" is not a valid command. Change the text "**MOVEDISTANCE**" to "**MOVED**".

Program	Compile	Resultant MotionView OnBoard Messages
UNITS=1 ACCEL = 5 DECEL = 5 ENABLE MOVED 10 MOVED -10 END	Compile After editing the program, select [Compile] on the program toolbar. After compilation is done, the "Compilation Complete" message box should appear.	

The program has now been compiled without errors. Select [Load W Source] to load the program to the drive's memory. Click [OK] to dismiss the dialog box.

Run

To **Run** the program, input A3 must be active to remove the hardware inhibit. Select the [**Run**] icon on the program toolbar. The drive will start to execute the User Program. The motor will spin 10 revolutions in the CCW direction and then 10 revolutions in the CW direction. After all the code has been executed, the program will stop and the drive will stay enabled.

Reset

To **Restart** the program, select the [**Reset**] icon on the program toolbar. This will disable the drive and reset the program to execute from the start. The program does not run itself automatically. To run the program again, select the [**Run**] icon on the toolbar.

Program Layout

When developing a program, structure is very important. It is recommended that the program be divided up into the following 7 segments:

Header:	The header defines the title of the program, who wrote the program and description of what the program does. It may also include a date and revision number.
I/O List:	The I/O list describes what the inputs and outputs of the drive are used for. For example input A1 might be used as a Start Switch.
Init & Set Var:	Initialize and Set Variables defines the drives settings and system variables. For example here is where acceleration, deceleration and max speed might be set.
Events:	An Event is a small program that runs independently of the main program. This section is used to define the Events.
Main Program:	The Main Program is the area where the main process of the drive is defined.
Sub-Routines:	This is the area where all sub-routines should reside. These routines will be called out from the Main Program with a GOSUB command.
Fault Handler:	This is the area where the Fault Handler code resides. If the Fault handler is utilized, then this code will be executed when the drive detects a fault condition.

The following is an example of a Pick and Place program divided up into the above segments.

```
***************************** HEADER  ************************************
;Title:          Pick and Place example program
;Author:         Lenze - AC Technology
;Description:    This is a sample program showing a simple sequence that
;                picks up a part, moves to a set position and places the part
;***************************** I/O List  **********************************
;      Input A1    -   not used
;      Input A2    -   not used
;      Input A3    -   Enable Input
;      Input A4    -   not used
;      Input B1    -   not used
;      Input B2    -   not used
;      Input B3    -   not used
;      Input B4    -   not used
;      Input C1    -   not used
;      Input C2    -   not used
;      Input C3    -   not used
;      Input C4    -   not used
;      Output 1    -   Pick Arm
;      Output 2    -   Gripper
;      Output 3    -   not used
;      Output 4    -   not used
```

```
;********************** Initialize and Set Variables **********************
UNITS = 1
ACCEL = 75
DECEL =75
MAXV = 10
;V1 =
;V2 =
;********************** Events ****************************************
;Set Events handling here
;No events are currently defined in this program
;********************** Main Program  *********************************
RESET_DRIVE:         ;Place holder for Fault Handler Routine
WAIT UNTIL IN_A3:    ;Make sure that the Enable input is made before continuing
ENABLE               ;Enable output from drive to motor
PROGRAM_START:       ;Place holder for main program loop
MOVEP  0             ;Move to Pick position
OUT1 = 1             ;Turn on output 1 to extend Pick arm
WAIT TIME 1000       ;Delay 1 sec to extend arm
OUT2 = 1             ;Turn on output 2 to Engage gripper
WAIT TIME 1000       ;Delay 1 sec to Pick part
OUT1 = 0             ;Turn off output 1 to Retract Pick arm
MOVED -10            ;Move 10 REVs to Place position
OUT1 = 1             ;Turn on output 1 to extend Pick arm
WAIT TIME 1000       ;Delay 1 sec to extend arm
OUT2 = 0             ;Turn off output 2 to Disengage gripper
WAIT TIME 1000       ;Delay 1 sec to Place part
OUT1 = 0             ;Retract Pick arm
GOTO PROGRAM_START   ;Loop back and continuously execute main program loop
END
;********************** Sub-Routines **********************************
Enter Sub-Routine code here
;********************** Fault Handler Routine *****************************
;     Enter Fault Handler code here
ON FAULT             ;No Fault Handler is currently defined in this program
ENDFAULT
```

Saving Configuration File to PC

The “Configuration File” consists of all the parameter settings for the drive, as well as the User Program. Once you are done setting up the drive’s parameters and have written your User Program, you can save these setting to your computer. To save the settings, select **[Save All]** from the **Main** toolbar. Then simply assign your configuration file a name, (e.g. Basic Motion), and click [Save] in the dialog box. The configuration file has a “dcf.xml” extension and by default will be saved to the “My Documents” folder.

Loading Configuration File to the Drive

There are times when it is helpful to import a a complete set-up or drive configuration to another drive. To load the configuration file from the PC to the drive, select **[Load All]** from the **Main** toolbar. Select the configuration file to load and click [Open] in the dialog box. MotionView will open the selected configuration file, set all parameters within the drive to the values contained within that file, and then extract, compile and download the saved user program. When the process is complete the [Compilation Complete] dialog box will appear.

Click [OK] to dismiss this dialog box. MotionView will then load the selected file to the drive. When complete, a second dialog box will appear indicating 'indexer program compiled and downloaded successfully'. Click [OK] too clear this message. Load of the configuration file is now complete.

Motion source (Reference)

The PositionServo can be set up to operate in one of three modes: Torque, Velocity, or Position. The drive must be given a command relative to its mode of operation before it can initiate any motion. The source for commanding this motion is referred to as the "Reference". With the PositionServo you have two methods of commanding motion, or two types of References. When the drive's reference signal is from an external source, for example a PLC or Motion Controller, it is referred to as an External Reference. When the drive is being given its reference from the User program or through one of the system variables it is referred to as an Internal Reference.

Table 3: Setting the Reference

	"Reference" Parameter Setting	
Mode	**External**	**Internal**
Torque	Analog input AIN1	System variable "IREF"
Velocity	Analog input AIN1	System variable "IREF"
Position	Step/Direction Inputs Master Encoder Pulse Train Inputs	User Program/Interface (Trajectory generator)

Units

All motion statements in the drive work with User units. The statement on the first line of the test program, UNITS=1, sets the relationship between programming units and motor revolutions. For example, if UNITS=0.5 the motor will turn 1/2 of a revolution when commanded to move 1 Unit. When the UNITS variable is set to zero, programming units for motion will be in motor feedback pulses (User units set to 1 divided into motor feedback pulses).

Time base

Time base for motion is always in seconds i.e. all time-related values are set in USER UNITS/SEC.
Time Base for programming statements (such as wait statements) are always in milliseconds.

Enable/Disable/Inhibit drive

Set "Enable switch function" to "Run".

When the "Enable switch function" parameter is set to Run, and the Input A3 is made, the drive will be enabled. Likewise, toggling input A3 to the off state will disable the drive.

- Select **[IO] then [Digital IO]** from the Parameter Tree Window.
- Select **"Enable switch function"** from the Parameter View Window.
- Select **"Run"** from the drop down menu. This setting is primarily used when operating without any user program in torque or velocity mode or as position follower with Step&Direction/Master Encoder reference.

Set "Enable switch function" to "Inhibit".

In the example of the Enable switch function being set to Run the decision on when to enable and disable the drive is determined by the logic status of input A3 (typically controlled by an external device, PLC or Motion controller). The PositionServo's User Program allows the programmer to define (control) within their program the enable and disable of the drive through execution of program statements. The drive will execute the User Program whether the drive is enabled or disabled, however if a motion statement is executed while the drive is disabled, an F27 fault will occur. If the user program commands the drive to enable and Input A3 (hardware enable) is not present or Input A3 is removed and the drive is enabled through programming then the drive will trip on Fault 36.

When the "**Enable switch function**" parameter is set to **Inhibit**, and Input A3 is on, the drive will be disabled and remain disabled until the ENABLE statement is executed by the User Program.

- Select **[IO] then [Digital IO]** from the Parameter Tree Window.
- Select "**Enable switch function**" from the Parameter View Window.
- Select "**Inhibit**" from the popup menu.

Faults

When a fault condition has been detected by the drive, the following actions will occur:

- Drive will Immediately be placed in a Disabled Condition.
- Motion Stack will be flushed of any Motion Commands
- Execution of the user program will be terminated and program control will be handed over to the Fault Handler section. If no Fault handler is described then program execution will terminate. See fault handler section.
- A fault code defining the nature of the drive trip will be written to the DFAULTS system variable and can be accessed by the fault handler. Refer to section 2.13 for a list of fault codes.
- The fault code will will be displayed on the drive display.
- Dedicated Ready/Enabled output will turn off, provided drive was in enable state prior to fault detection.
- Any Output with assigned special function "Fault" will turn on.
- Any Output with assigned special function "ready/enabled" will turn off, provided drive was in enable state prior to fault detection
- The "enable" status indicator on the drive display will turn off indicating drive in disabled state.

Clearing a fault condition can be done in one of the following ways:

- Select the [**Reset**] button from the toolbar.
- Execute the **RESUME** statement at the end of the Fault Handler routine (see Fault Handler example). This permits the continuation of program execution at the discretion of the programmer and when the fault does not present an issue to the safety or integrity of the system.
- Send "Reset" command over the Host Interface.
- Cycle power (hard reset).

Fault Handler

The Fault Handler is a code segment that will be executed immediately on the drive detecting a fault condition. The fault handler allows the programmer to analyze the type of fault and (when necessary) define a recovery process for the drive Full stop. While the drive is executing the Fault Handler Routine the drive is disabled and therefore will not be able to detect any additional faults that might occur. Fault handler code is the drive's first reaction to a fault condition. While it executes, the drive will not respond to any I/O, interface commands or program events. Therefore the user should use the fault handler to manipulate time critical and safety related I/O and variables and then exit the Fault Handler Routine either by executing a "**RESUME**" statement or by executing the EndFault statement and ending program execution. The Resume statement permits program execution to leave the fault handler and resume back in the main program section of the user code. Use the Resume statement to jump back to a section of the main program that designates the recovery process for the fault. Wait statements within the fault handler for I/O state change or for interface command is not allowed. If a wait statement is required (for example from a fault reset input) then this must be done subsequent to the Resume command when program execution is handed back to the main program.

Without Fault Handler

To simulate a fault, restart the Pick and Place example program. While the program is running, switch the ENABLE input IN_A3 to the off state. This will cause the drive to generate an F_36 fault (Hardware disable while drive enabled in inhibit mode) and put the drive into Fault Mode. While the drive is in Fault Mode, any digital output currently active will remain active and any output deactivated will remain deactivated, excluding the dedicated ready output and any output that has been assigned pre-defined functionality. The program execution will stop and any motion commands will be terminated.

With Fault Handler

Add the following code to the end of your sample program. When the program is running, switch the ENABLE input IN_A3, to the off state. This will cause the drive to generate an F_36 fault ((Hardware disable while drive enabled in inhibit mode) and put the drive into a Fault Mode. From this point the Fault Handler Routine will take over.

```
F_PROCESS:
WAIT UNTIL IN_A4==1   ;Wait until reset switch is made
WAIT UNTIL IN_A4==0   ;and then released before
GOTO RESET_DRIVE      ;returning to the beginning of the program
END
;********************** Sub-Routines ************************************
Enter Sub-Routines here;
;********************** Fault Handler Routine ***************************
ON FAULT              ;Statement starts fault handler routine
                      ;Motion stopped, drive disabled, and events no longer
                      ;scanned while executing the fault handler routine.
OUT2 = 0              ;Output 1 off to Disengage gripper.
                      ;This will drop the part in the gripper
OUT1 = 0              ;Retract Pick arm to make sure it is up and out of the way
RESUME F_PROCESS      ;program restarts from label F_PROCESS
ENDFAULT              ;fault handler MUST end with this statement
```

NOTE

The following statements can not be used inside the Fault Handler Routine:

ENABLE
WAIT
MOVE
MOVED
MOVEP
MOVEDR
MOVEPR
MDV
MOTION SUSPEND
MOTION RESUME
GOTO, GOSUB
JUMP
VELOCITY ON/OFF
WHILE / ENDWHILE
DO / UNTIL
EVENT (ON, OFF)
EVENTS (ON, OFF)
HOME
HALT
STOP MOTION (QUICK)

Refer to section 2.1 for additional details and the Language Reference section for the statement “ON FAULT/ENDFAULT”.

1.5 Using Advanced Debugging Features

To debug a program or view the I/O, open the Diagnostic panel by clicking on the [Tools] in the Parmeter (Node) Tree list then click on the [Parameter & I/O View] button. The Diagnostic panel will open. This panel allows the programmer to monitor and set variables, and to view status of drive digital inputs and outputs.

Λ	Use the up [^] button to move the blue highlighted bar up through the variable list and select a parameter
Add	Use the [Add] button to open the Parameters dialog box. Select the variable(s) to add by clicking on the box adjacent to the variable #. When finished selecting variables, click [Add] in the Parameter dialog box to add these variables to the watch window.
>>	Use the right arrow button to remove highlighted variable from the watch window.
V	Use the down [V] button to move the blue highlighted bar down through the variable list and select a parameter
Clear	Use the [Clear] button to clear all the parameters listed in the watch window.
Load	Use the [Load] button to load a set of previously saved variables to the watch window.
Save	Use the [Save] button to save the configuration of variables listed in the watch window to a file on the PC. Configuration can then easily be restored using the [Load] button.

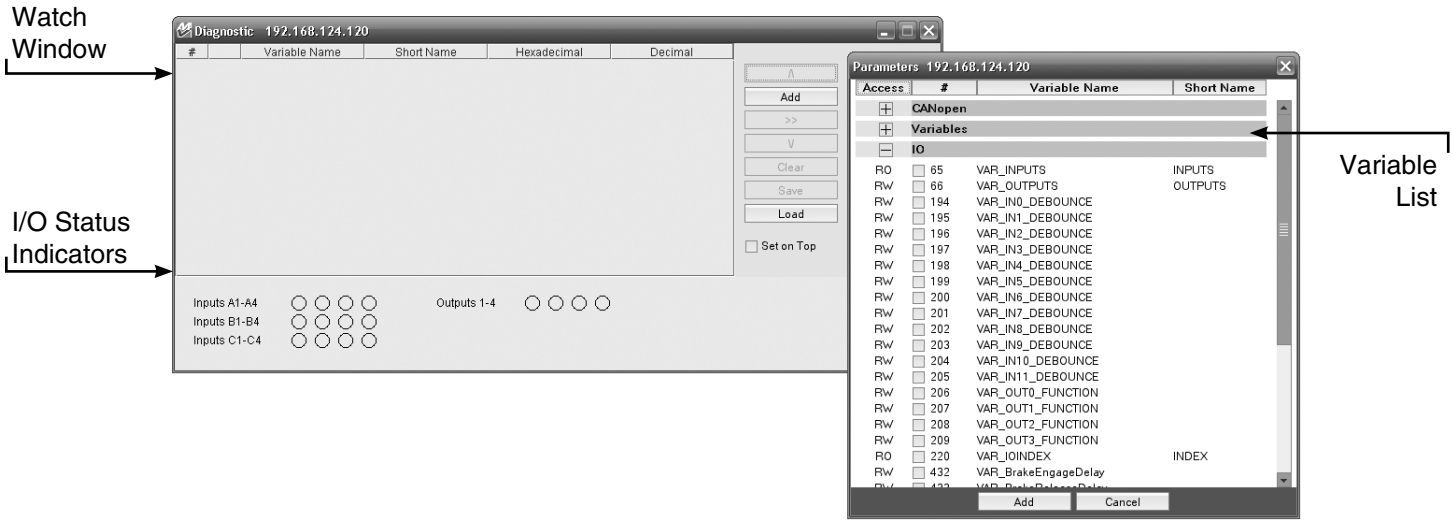


Figure 6: Variable Diagnostic Display

NOTE
Write-only variables cannot be read. Attempts to either display a write-only variable in the diagnostic panel or to read a write-only variable via network communications can show erroneous data.

1.6 Inputs and Outputs

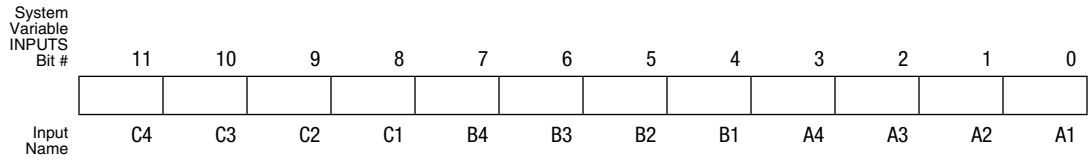
Analog Input and Output

- The PositionServo has two analog inputs. These analog inputs are utilized by the drive as System Variables and are labeled "**AIN1**" and "**AIN2**". Their values can be directly read by the User Program or via a Host Interface. Their value can range from -10 to +10 and correlates to ±10 volts analog input.
- The PositionServo has one analog output. This analog output is utilized by the drive as a System Variable and is labeled "**AOUT**". It can be directly written by the User Program or via a Host Interface. Its value can range from -10 to +10 which correlates to ± 10 volts analog input.

NOTE
If an analog output is assigned to any special function from MotionView, writing to AOUT from the User Program will have no effect. If an analog output is set to "Not assigned" then it can be controlled by writing to the AOUT variable.

Digital Inputs

- The PositionServo has twelve digital inputs that are utilized by the drive for decision making in the User Program. Example uses: travel limit switches, proximity sensors, push buttons and hand shaking with other devices.
- Each input can be assigned an individual debounce time via MotionView. From the **Parameter Tree,** select [**IO**]. Then select the [**Digital Input**] folder. The debounce times will be displayed in the **Parameter View Window**. Debounce times can be set between 0 and 1000 ms (1ms = 0.001 sec). Debounce times can also be set via variables in the user program.
- The twelve inputs are separated into three groups: A, B and C. Each group has four inputs and share one common: Acom, Bcom and Ccom respectfully. The inputs are labeled individually as **IN_A1 - IN_A4, IN_B1 - IN_B4 and IN_C1 - IN_C4**.
- In addition to monitoring each input individually, the status of all twelve inputs can be represented as one binary number. Each input corresponds to 1 bit in the INPUTS system variable. Use the following format:



- Some inputs can be configured for additional predefined functionality such as Travel Limit switch, Enable input, and Registration input. Configuration of these inputs is done from MotionView or through variables in the user program. Input special functionality is summarized in the table below and in the following sections. Table 4 summarizes the special functions for the inputs.

Table 4: Input Functions

Input Name	Special Function
Input A1	Negative limit switch
Input A2	Positive limit switch
Input A3	Inhibit/Enable input
Input A4	N/A
Input B1	N/A
Input B2	N/A
Input B3	N/A
Input B4	N/A
Input C1	N/A
Input C2	N/A
Input C3	Registration sensor input
Input C4	N/A

The current status of the drive's inputs is available to the programmer through dedicated System Flags or as bits of the System Variable INPUTS.

Read Digital Inputs

The Pick and Place example program has been modified below to utilize the “WAIT UNTIL” statement in place of the “WAIT TIME” statement. **IN_A1** and **IN_A4** will be used as proximity sensors to detect when the pick and place arm is extended and when it is retracted. When the arm is extended, **IN_A1** will be in an ON state and will equal “1”. When the arm is retracted, **IN_A4** will be in an ON state and will equal “1”.

```
;********************* Main Program ***************************************
RESET_DRIVE:            ;Place holder for Fault Handler Routine
WAIT UNTIL IN_A3        ;Make sure that the Enable input is made before continuing
ENABLE
OUT1 = 0                ;Initialize Pick Arm - Place in Retracted Position
WAIT UNTIL IN_A4==1     ;Check Pick Arm is in Retracted Position
PROGRAM_START:
MOVEP  0                ;Move to Pick position
OUT1 = 1                ;Turn on output 1 to extend Pick arm
WAIT UNTIL IN_A1==1     ;Arm extends
OUT2 = 1                ;Turn on output 2 to Engage gripper
WAIT TIME 1000          ;Delay 1 sec to Pick part
OUT1 = 0                ;Turn off output 1 to Retract Pick arm
WAIT UNTIL IN_A4==1     ;Make sure Arm is retracted
MOVED -10               ;Move 10 REVs to Place position
OUT1 = 1                ;Turn on output 1 on to extend Pick arm
WAIT UNTIL IN_A1==1     ;Arm is extended
OUT2 = 0                ;Turn off output 2 to Disengage gripper
WAIT TIME 1000          ;Delay 1 sec to Place part
OUT1 = 0                ;Retract Pick arm
WAIT UNTIL IN_A4==1     ;Arm is retracted
GOTO PROGRAM_START
END
```

Once the above modifications have been made, export the program to file and save it as “Pick and Place with I/O”, then compile, download and test the program.

ASSIGN & INDEX - Using inputs to generate predefined indexes

“INDEX” is a variable on the drive that can be configured to represent a specified group of inputs as a binary number. “ASSIGN” is the command that designates which inputs are utilized and how they are configured.

Below the Pick and Place program has been modified to utilize this “INDEX” function. The previous example program simply picked up a part and moved it to a place location. For demonstration purposes we will add seven different place locations. These locations will be referred to as Bins. What Bin the part is placed in will be determined by the state of three inputs, B1, B2 and B3.

Bin 1	-	Input B1 is made
Bin 2	-	Input B2 is made
Bin 3	-	Inputs B1 and B2 are made
Bin 4	-	Input B3 is made
Bin 5	-	Inputs B1 and B3 are made
Bin 6	-	Inputs B2 and B3 are made
Bin 7	-	Inputs B1, B2 and B3 are made

The “ASSIGN” command is used to assign the individual input to a bit in the “INDEX” variable. ASSIGN INPUT <input name> AS BIT <bit #>

```
;********************** Initialize and Set Variables ******************
ASSIGN INPUT IN_B1 AS BIT 0  ;Assign the Variable INDEX to equal 1 when IN_B1 is made
ASSIGN INPUT IN_B2 AS BIT 1  ;Assign the Variable INDEX to equal 2 when IN_B2 is made
ASSIGN INPUT IN_B3 AS BIT 2  ;Assign the Variable INDEX to equal 4 when IN_B4 is made
```

Table 5: Bin Location, Inputs & Index Values

Bin Location	Input state	INDEX Value
Bin 1	Input B1 is made	1
Bin 2	Input B2 is made	2
Bin 3	Inputs B1 and B2 are made	3
Bin 4	Input B3 is made	4
Bin 5	Inputs B1 and B3 are made	5
Bin 6	Inputs B2 and B3 are made	6
Bin 7	Inputs B1, B2 and B3 are made	7

The Main program has been modified to change the end place position based on the value of the "**INDEX**" variable.

```
;************************** Main Program **********************************
ENABLE
OUT1 = 0                    ;Initialize Pick Arm - Place in Retracted Position
WAIT UNTIL IN_A4==1         ;Check Pick Arm is in Retracted Position
PROGRAM_START:
MOVEP  0                    ;Move to (ABS) to Pick position
OUT1 = 1                    ;Turn on output 1 to extend Pick arm
WAIT UNTIL IN_A1==1         ;Arm extends
OUT2 = 1                    ;Turn on output 2 to Engage gripper
WAIT TIME 1000              ;Delay 1 sec to Pick part
OUT1 = 0                    ;Turn off output 1 to Retract Pick arm
WAIT UNTIL IN_A4==0         ;Make sure Arm is retracted

IF INDEX==1                 ;In this area we use the If statement to
GOTO BIN_1                  ;check and see what state inputs B1, B2 & B3
ENDIF                       ;are in.
IF INDEX==2                 ;     INDEX = 1 when input B1 is made
GOTO BIN_2                  ;     INDEX = 2 when input B2 is made
ENDIF                       ;     INDEX = 3 when input B1 & B2 are made.
.                           ;     INDEX = 4 when input B3 is made
.                           ;     INDEX = 5 when input B1 & B3 are made.
.                           ;     INDEX = 6 when input B2 & B3 are made.
IF INDEX==7                 ;     INDEX = 7 when input B1, B2 & B3 are made
GOTO BIN_7                  ;We can now direct the program to one of seven
ENDIF                       ;locations based on three inputs.

BIN_1:                      ;Set up for Bin 1
MOVEP 10                    ;Move to Bin 1 location
GOTO PLACE_PART             ;Jump to place part routine
BIN_2:                      ;Set up for Bin 2
MOVEP 20                    ;Move to Bin 2 location
GOTO PLACE_PART             ;Jump to place part routine
BIN_7:                      ;Set up for Bin 7
MOVEP 70                    ;Move to Bin 7 location
GOTO PLACE_PART             ;Jump to place part routine
PLACE_PART:
OUT1 = 1                    ;Turn on output 1 to extend Pick arm
WAIT UNTIL IN_A4 == 1       ;Arm extends
OUT2 = 0                    ;Turn off output 2 to Disengage gripper
WAIT TIME 1000              ;Delay 1 sec to Place part
OUT1 = 0                    ;Retract Pick arm
WAIT UNTIL IN_A4 == 0       ;Arm is retracted
GOTO PROGRAM_START
END
```

NOTE: with all digital inputs (B1-B3) off, none of the 'If' statements to detect place position are true and the program defaults to placing the part it has picked into bin location 1.

NOTE
Any one of the 12 inputs can be assigned as a bit position within the INDEX variable. Only bits 0 through 7 can be used with the INDEX variable. Bits 8-31 are not used and are always set to 0. Unassigned bits in the INDEX variable are set to 0.

BITS 8-31 (not used)	A1	0	A2	A4	0	0	0	0

Limit Switch Input Functions

Inputs A1 and A2 can be configured as special purpose inputs from the [**Digital IO]** folder in MotionView. They can be set to one of three settings:

- The "**Not assigned**" setting designates the inputs as general purpose inputs which can be utilized by the User Program.
- The "**Fault**" setting will configure A1 and A2 as Hard Limit Switches. When either input is made the drive will be disabled, the motor will come to an uncontrolled stop, and the drive will generate a fault. If the negative limit switch is activated, the drive will display an F-33 fault. If the positive limit switch is activated the drive will display an F32 fault.
- The "**Stop and fault**" setting will configure A1 and A2 as End of Travel limit switches. When either input is made the drive will initiate a rapid stop before disabling the drive and generating an F34 or F35 fault (refer to section 2.15 for details). The speed of the deceleration will be set by the value stored in the "**QDECEL**" System Variable.

NOTE
The "Stop and Fault" function is available in position mode only, ("Drive mode" is set to "Position"). In all other cases, the Stop and Fault function will act the same as the Fault function.

To set this parameter, select the [**IO**] folder from the Parameter Tree. Then select the [**Digital IO**] folder. From the Parameter View Window, use the pull-down menu next to [**Hard Limit Switches Action**] to select the status: Not Assigned, Fault or Stop and Fault.

Digital Outputs Control

- The PositionServo has 5 digital outputs. The "**RDY**" or READY output is dedicated and will only come on when the drive is enabled, i.e. in **RUN** mode. The other outputs are labeled **OUT1 - OUT4**.
- Outputs can be configured as Special Purpose Outputs. If an output is configured as a **Special Purpose Output** it will activate when the state assigned to it becomes true. For example, if an output is assigned the function "**Zero speed**", the assigned output will come on when the motor is not in motion. To configure an output as a Special Purpose Output, select the [**IO**] folder from the Parameter Tree. Then select the [**Digital IO**] folder. From the Parameter View Window, select the "**Output function**" parameter you wish to set (1, 2, 3 or 4).
- Outputs that are configured as "Not assigned" can be activated either via the User Program or from a host interface. If an output is assigned as a Special Purpose Output, neither the user program nor the host interface can overwrite its status.
- The Systems Variable "**OUTPUTS**" is a read/write variable that allows the User Program, or host interface, to monitor and set the status of all four outputs. Each output allocates 1 bit in the OUTPUTS variable. For example, if you set this variable equal to 15 in the User Program,i.e. 1111 in binary format, then all 4 outputs will be turned on.
- The example below summarizes the output functions and corresponding System Flags. To set the output, write any non-0 value (TRUE) to its flag. To clear the output, write a 0 value (FALSE) to its flag. You can also use flags in an expression. If an expression is evaluated as TRUE then the output will be turned ON. Otherwise, it will be turned OFF.

```
OUT1 = 1                      ;turn OUT1 ON
OUT2 = 10                     ;any value but 0 turns output ON
OUT3 = 0                      ;turn OUT3 OFF
OUT2 = APOS>3 && APOS<10      ;ON when position within window, otherwise OFF
```

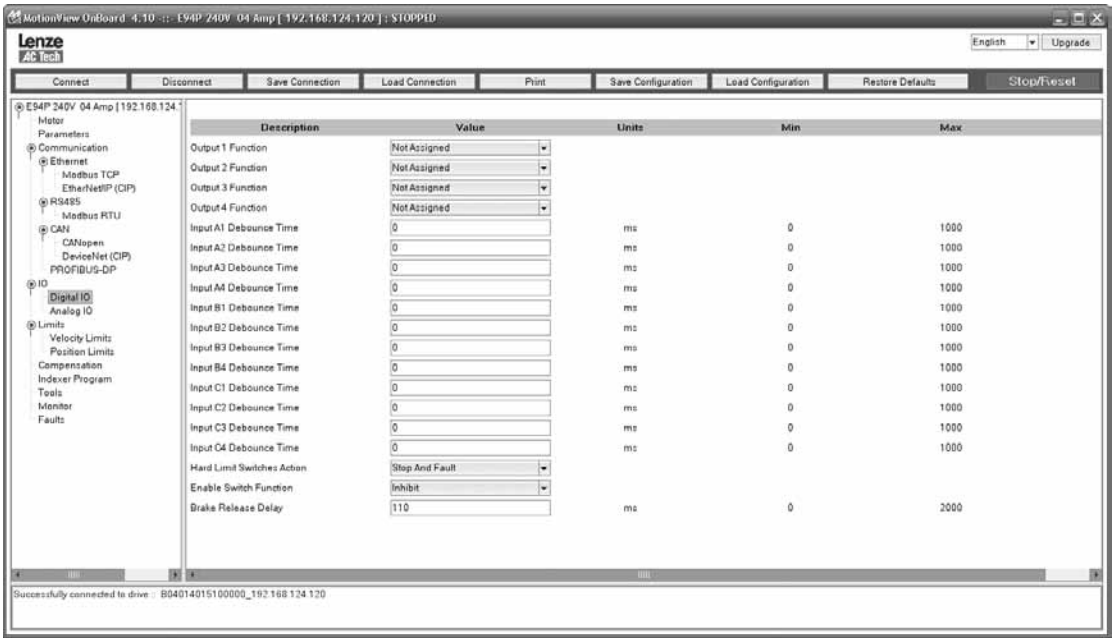


Figure 7: Digital IO Folder

1.7 Events

A Scanned Event is a small program that runs independently of the main program. An event statement establishes a condition that is scanned on a regular basis. Once established, the scanned event can be enabled and disabled in the main program. If condition becomes true and EVENT is enabled, the code placed between EVENT and ENDEVENT executes. Scanned events are used to trigger the actions independently of the main program.

In the following example the Event "**SPRAY_GUNS_ON**" will be setup to turn Output 3 on when the drive's position becomes greater than 25. Note: the event will be triggered only at the instant when the drive position becomes greater than 25. It will not continue to execute while the position remains greater than 25. (i.e the event is triggered by the transition in logic from false to true). Note also that the main program does not need to be interrupted to perform this action.

```
;********************** EVENT SETUP ************************************
EVENT SPRAY_GUNS_ON       APOS>25
OUT3=1
ENDEVENT
;*****************************************************************************
```

Enter the Event code in the EVENT SETUP section of the program. To Setup an Event, the **"EVENT"** command must be entered. This is followed by the Event Name **"SPRAY_GUNS_ON"** and the triggering mechanism, **"APOS>25"**. After that a sequence of programming statements can be entered once the event is triggered. In our case, we will turn on output 3. To end the Event, the **"ENDEVENT"** command must be used. Events can be activated (turned on) and deactivated (turned off) throughout the program. To turn on an Event, the "**EVENT**" command is entered, followed by the Event Name "**SPRAY_GUNS_ON**". This is completed by the desired state of the Event, "**ON**" or "**OFF**". Refer to Section 2.10 for more on Scanned Events.

```
;*****************************************************************************
EVENT SPRAY_GUNS_ON       ON
;*****************************************************************************
```

Two Scanned Events have been added to the Pick and Place program below to trigger a spray gun on and off. The Event will be triggered after the part has been picked up and is passing in front of the spray guns (position greater than 25). Once the part is in position, output 3 is turned on to activate the spray guns. When the part has passed by the spray guns, (position greater than 75), output 3 is turned off, deactivating the spray guns.

```
;************************** Events ************************************************
EVENT   SPRAY_GUNS_ON     APOS>25   ;Event will trigger as position passes 25 in pos dir.
OUT3=1                              ;Turn on the spray guns (out 3 on)
ENDEVENT                            ;End event
EVENT   SPRAY_GUNS_OFF    APOS>75   ;Event will trigger as position passes 75 in pos dir.
OUT3=0                              ;Turn off the spray guns (out 3 off)
ENDEVENT                            ;End event
;************************** Main Program ******************************************
WAIT UNTIL IN_A3                    ;Make sure the Enable input is made before continuing
ENABLE
OUT1 = 0                            ;Initialize Pick Arm - Place in Retracted Position
WAIT UNTIL IN_A4==1                 ;Check Pick Arm is in Retracted Position
EVENT   SPRAY_GUNS_ON     ON
EVENT   SPRAY_GUNS_OFF    ON
PROGRAM_START:
MOVEP 0                             ;Move to Pick position
OUT1 = 1                            ;Turn on output 1 to extend Pick arm
WAIT UNTIL IN_A1==1                 ;Arm extends
OUT2 = 1                            ;Turn on output 2 to Engage gripper
WAIT TIME 1000                      ;Delay 1 sec to Pick part
OUT1 = 0                            ;Turn off output 1 to Retract Pick arm
WAIT UNTIL IN_A4==1                 ;Make sure Arm is retracted
MOVEP 100                           ;Move to Place position
OUT1 = 1                            ;Turn on output 1 to extend Pick arm
WAIT UNTIL IN_A1==1                 ;Arm extends
OUT2 = 0                            ;Turn off output 2 to Disengage gripper
WAIT TIME 1000                      ;Delay 1 sec to Place part
OUT1 = 0                            ;Retract Pick arm
WAIT UNTIL IN_A4==1                 ;Arm is retracted
GOTO PROGRAM_START
END
```

1.8 User Variables and the Define Statement

In the previous program for the pick and place machine constant values were used for position limits to trigger the events and turn the spray gun ON and OFF. If limits must be calculated based on some parameters unknown before the program runs (like home origin, material width, etc.), then this system data can be stored in user variables. The PositionServo provides 32 User Variables V0-V31 and 32 User Network Variables NV0-NV31. Network variables have an additional function associated to them (refer to 'Send' Command) but can, for most purposes, be considered as user variables in the same way as the standard user variables (V0-31). Hence 64 user variables or data storage locations are available to the programmer. In the program following the example DEFINE statements, the limit APOS (actual position) is compared to V1 for an ON event and V2 for an OFF event. The necessary limit values could be calculated earlier in the program or supplied by an HMI or host PC. The DEFINE statement can be used to assign a name to a constant, variable, or drive Input/Output. In the program below, constants 1 and 0 are defined as Output_On and Output_Off. DEFINE is a pseudo statement, i.e it is not executed by the program interpreter, but rather substitutes expressions in the subsequent program at the time of compilation. Examples of the DEFINE statement:

```
; Definition of Constant Values
DEFINE Move_1 100
DEFINE BallScrewPitch 0.357

; Definition of Inputs/Outputs
DEFINE System_Run_IP In_B1
DEFINE Process_Run_OP Out1

; Definition User Variables
DEFINE Distance_Travelled V2
DEFINE Network_Healthy NV10

Programming the following statement: Distance_Travelled = Move_1 * BallScrewPitch
Is now the equivalent of writing: V2 = 100 * 0.357
```

```
;************************** Initialize and Set Variables ****************************
UNITS = 1                           ;Define units for program, 1=revolution of motor shaft
ACCEL = 5                           ;Set Acceleration rate for Motion command
DECEL = 5                           ;Set Deceleration rate for Motion command
MAXV = 10                           ;Maximum Velocity for Motion commands
V1 = 25                             ;Set Variable V1 equal to 25
V2 = 75                             ;Set Variable V2 equal to 75
DEFINE       Output_On    1         ;Define Name for output On
DEFINE       Output_Off   0         ;Define Name for output Off
;**************************  EVENTS  ************************************************
EVENT SPRAY_GUNS_ON   APOS > V1     ;Event will trigger as position passes 25 in pos dir.
OUT3= Output_On                     ;Turn on the spray guns (out 3 on)
ENDEVENT                            ;End event

EVENT SPRAY_GUNS_OFF  APOS > V2     ;Event will trigger as position passes 75 in pos dir.
OUT3= Output_Off                    ;Turn off the spray guns (out 3 off)
ENDEVENT                            ;End event
;**************************  Main Program  ******************************************
WAIT UNTIL IN_A3                    ;Make sure the Enable input is made before continuing
ENABLE
OUT1 = 0                            ;Initialize Pick Arm - Place in Retracted Position
WAIT UNTIL IN_A4==1                 ;Check Pick Arm is in Retracted Position
EVENT        SPRAY_GUNS_ON ON       ;Enable the Event
EVENT        SPRAY_GUNS_OFF ON      ;Enable the Event
PROGRAM_START:
MOVEP 0                             ;Move to position 0 to pick part
OUT1 = Output_On                    ;Turn on output 1 to extend Pick arm
WAIT UNTIL IN_A1==1                 ;Check input to make sure Arm is extended
OUT2 = Output_On                    ;Turn on output 2 to Engage gripper
WAIT TIME 1000                      ;Delay 1 sec to Pick part
OUT1 = Output_Off                   ;Turn off output 1 to Retract Pick arm
WAIT UNTIL IN_A4==1                 ;Check input to make sure Arm is retracted
MOVED 100                           ;Move to Place position
OUT1 = Output_On                    ;Turn on output 1 to extend Pick arm
WAIT UNTIL IN_A1==1                 ;Check input to make sure Arm is extended
OUT2 = Output_Off                   ;Turn off output 2 to Disengage gripper
WAIT TIME 1000                      ;Delay 1 sec to Place part
OUT1 = Output_Off                   ;Retract Pick arm
WAIT UNTIL IN_A4==1                 ;Check input to make sure Arm is retracted
GOTO PROGRAM_START
END
```

1.9 IF/ELSE Statements

An IF/ELSE statement allows the user to execute one or more statements conditionally. The programmer can use an IF or IF/ELSE construct:

Single IF example:

This example increments a counter, Variable “V1”, until the Variable, “V1”, is greater than 10.

```
Again:
      V1=V1+1
      IF V1>10
            V1=0
      ENDIF
      GOTO Again
END
```

IF/ELSE example:

This example checks the value of Variable V1. If V1 is greater than 3, then V2 is set to 1. If V1 is not greater than 3, then V2 is set to 0.

```
IF V1>3
      V2=1
ELSE
      V2=0
ENDIF
```

Whether you are using an IF or IF/ELSE statement the construct must end with ENDIF keyword.

1.10 Motion

Figure 8 illustrates the Position and Velocity regulator of the PositionServo drive.

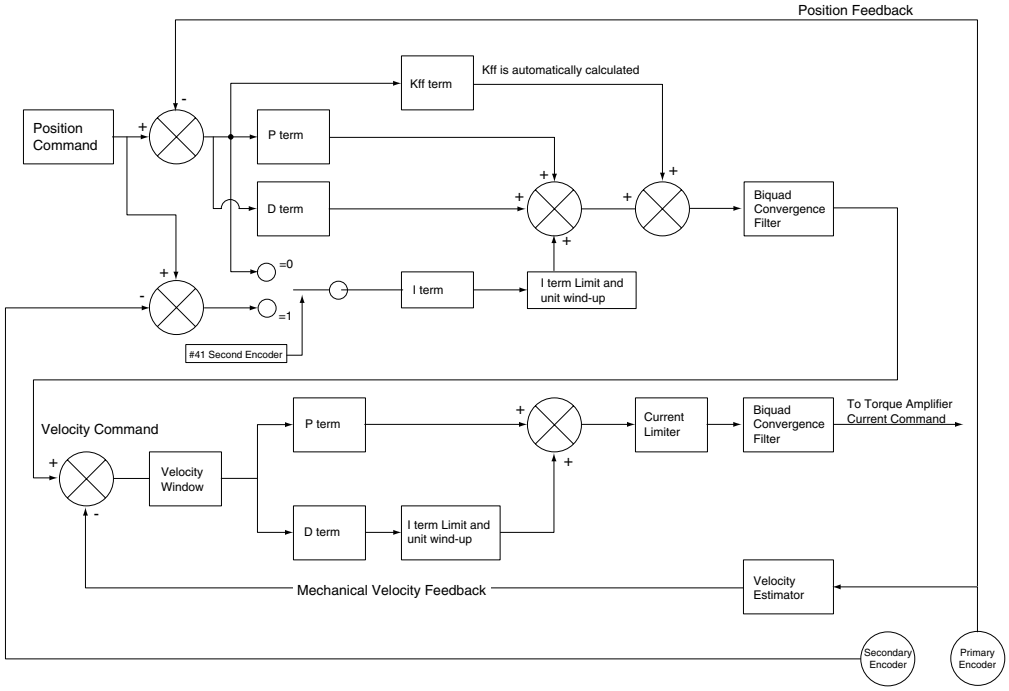


Figure 8: PositionServo Position and Velocity Regulator's Diagram

The "**Position Command**", as shown in the regulator's diagram (Figure 9), is produced by a **Trajectory Generator**. The Trajectory Generator processes the motion commands produced by the User's program to calculate the position increment or decrement, also referred to as the "index" value, for every servo loop. This calculated target (or theoretical) position is then supplied to the **Regulator** input.

The main purpose of the **Regulator** is to set the motors position to match the target position created by the Trajectory Generator. This is done by comparing the input from the Trajectory Generator with the position feedback from the primary motor feedback (resolver or encoder) to control the torque and velocity of the motor. There will always be some error in the position following. Such error is referred to as "Position Error" and is expressed as follows:

Position Error = Target Position - Actual Position

When the actual Position Error exceeds a certain threshold value for greater than the predefined time limit a "Position Error limit", fault (F_PE) will be generated. The Position Error limit and Position Error time can be set under the Parameter (Node) Tree "Limits"/ "Position Limits" in MotionView. The Position Error time specifies how long the actual position error can exceed the Position Error limit before the fault is generated.

1.10.1 Drive Operating Modes

There are three modes of operation for the PositionServo: Torque, Velocity and Position. Torque and Velocity modes are generally used when the command reference is from an external device (via analog input 1), however mechanisms also exist for operation in these modes from within the internal user program. Position mode is used when the command comes from the drives User Program, or from an external device (drive fed from encoder or step/direction signal). Setting the drive's mode is done from the [**Parameter**] folder in MotionView. To command motion from the user program the drive must be configured to internal reference mode. When the drive is in position mode, it can be placed into a simulated velocity mode without the need to change operating mode to 'Velocity'. Velocity profiling from Positioning mode can be turned on and off from the User Program. Executing the VELOCITY ON statement is used to activate this mode while VELOCITY OFF will deactivate this mode. This mode is used for special case indexing moves. When in Velocity simulation mode the target position is constantly advanced with a rate set in the VEL system variable. The Reference arrangements for the different modes of operation are illustrated in Figure 9.

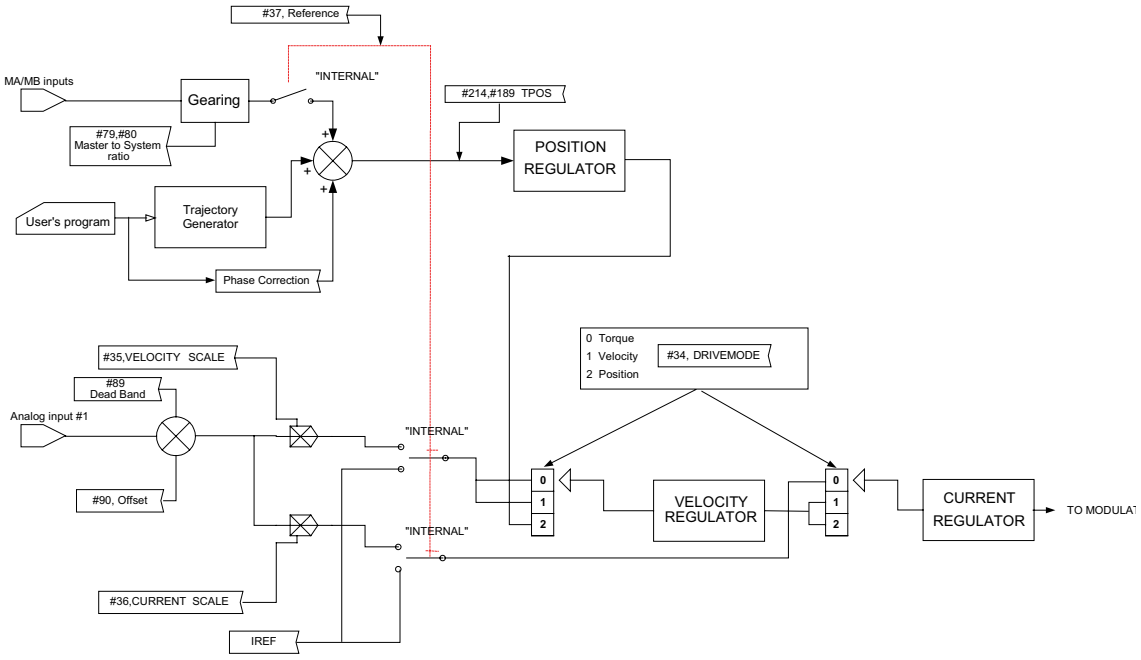


Figure 9: Reference Arrangement Diagram

1.10.2 Point To Point Moves

The PositionServo supports two types of moves: absolute and incremental. The statement MOVEP (Move to Position) is used to make an absolute move. When executing an absolute move, the motor is instructed to move to a known position. The move to this known position is always referenced from the motor's "home" or "zero" location. For example, the statement (MOVEP 0) will cause the motor to move to its zero or home position, regardless of where the motor is located at the beginning of the move. The statement MOVED (Move Distance) makes incremental, (or relative), moves from its current position. For example, MOVED 10, will cause the motor to move forward 10 user units from it current location.

MOVEP and MOVED statements generate what is called a trapezoidal point to point motion profile. A trapezoidal move is when the motor accelerates, using the current acceleration setting, (ACCEL), to a pre-defined top speed, (MAXV), it then maintains that speed for a period of time before decelerating to the end position using the deceleration setting, (DECEL). If the distance to be moved is fairly small, a triangular move profile will be used. A triangular move is a move that starts to accelerate toward the Max Velocity setting but has to decelerate before ever achieving the max velocity in order to reach the desired end point.

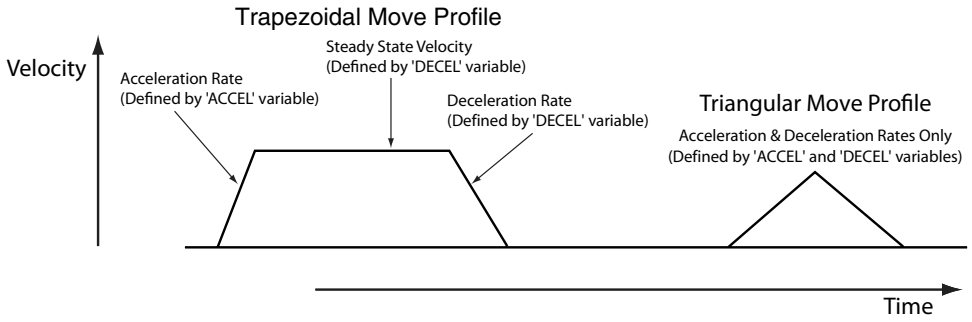


Figure 10: Trapezoidal Move

1.10.3 Segment Moves

MOVED and MOVEP commands facilitate simple motion to be commanded, but if the required move profile is more complex than a simple trapezoidal will allow, then the segment MDV move can be used.

The profile shown in Figure 11 is divided into 8 segments or 8 MDV moves. An MDV move (Move Distance Velocity) has two arguments. The first argument is the distance moved in that segment. This distance is referenced from the motor's current position in User Units. The second argument is the desired target velocity for the end of the segment move. That is the velocity at which the motor will run at the moment when the specified distance in this segment is completed.

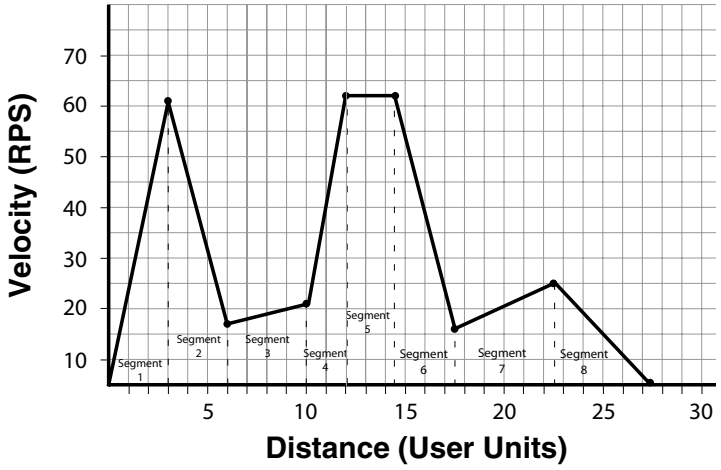


Figure 11: Segment Move

Table 6: Segment Move

Segment Number	Distance moved during segment	Velocity at the end of segment
1	3	56
2	3	12
3	4	16
4	2	57
5	2.5	57
6	3	11
7	5	20
8	5	0
-	-	-

Here is the user program for the segment move example. The last segment move must have a “0” for the end velocity, (MDV 5 , 0). Otherwise, fault F_24 (Motion Queue Underflow), will occur.

```
;Segment moves
LOOP:
WAIT UNTIL IN_A4==0   ;Wait until input A4 is off before starting the move
MDV 3 , 56            ;Move 3 units accelerating to 56 User Units per sec
MDV 3 , 12            ;Move 3 units decelerating to 12 User Units per sec
MDV 4 , 16            ;Move 4 units accelerating to 16 User Units per sec
MDV 2 , 57            ;Move 2 units accelerating to 57 User Units per sec
MDV 2.5 , 57          ;Move 2.5 units maintaining 57 User Units per sec
MDV 3 , 11            ;Move 3 units decelerating to 11 User Units per sec
MDV 5 , 20            ;Move 5 units accelerating to 20 User Units per sec
MDV 5 , 0             ;Move 5 units decelerating to 0 User Units per sec
WAIT UNTIL IN_A4==1   ;Wait until input A4 is on before looping
GOTO LOOP
END
```

NOTE

When an MDV move is executed, the segment moves are stored to a Motion Queue. A maximum of 32 moves (MDV segments) can be held on the Motion Queue at any one time. When a move or segment is completed it is cleared from the Motion Queue. If the program attempts to place more than 32 moves in the Motion Queue (because motion is complex or the program continuously loops on itself) then a fault 23 (F_23) will occur to indicate motion queue overflow.

Since a series of MDV segments need to be loaded quickly to the Motion Queue, the **[Step]** debugging feature can not be used.

1.10.4 Registration

Both absolute and incremental motion can be used for registration moves. The statements associated with these moves are MOVEPR and MOVEDR. These statements have two arguments. The first argument specifies the commanded move distance or position. The second argument specifies the move made after the registration input is detected. If the registration move is an absolute move, for MovePR, the first argument is absolute (referenced to the 0 position), the second argument is relative to the registration position. For MoveDR, both arguments are relative. The first is relative to the shaft position when motion is started and the second is relative to the registration position.

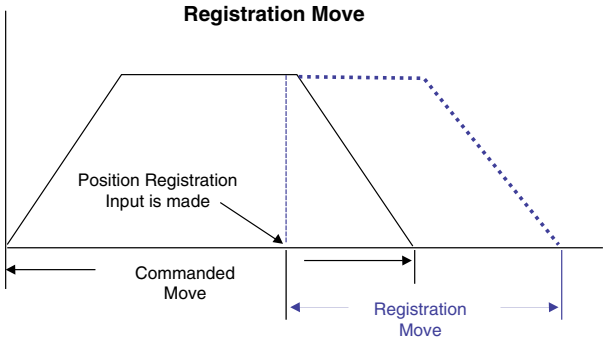


Figure 12: Registration Move

1.10.5 S-Curve Acceleration/Deceleration

It is often necessary, particularly for very dynamic applications, to smooth transition between periods of acceleration / deceleration and steady state velocity. A smoothing of this transition could improve the results of tuning and hence improve overall performance of the system. Additionally smoothing the ramp rates can have the effect of minimizing wear and tear on the system's mechanical components.

With normal straight line ramp rates, the axis is accelerated or decelerated to the target velocity in a linear fashion. With S-curve acceleration/deceleration, the motor ramp rate changes slowly at the first and then slowly stops accelerating/ decelerating as it reaches the target velocity. In order for the overall or average ramp rate to remain the same (as specified in the ACCEL/DECEL variables) the slow rates of change at the beginning and the end of the S-curve are compensated by a faster ramp rate in the middle section of the ramp. Maximum ramp rate (occurring in the mid-point of the S-curve) is twice that of using straight line ramps and of the values entered in the ramp rate variables. With straight line ramp rates, the acceleration/deceleration changes can be abrupt at the beginning of the ramp period and again once the motor reaches the target velocity. With S-curve ramp rates, the ramp rate gradually builds to the peak value then gradually decreases to no acceleration/deceleration. The disadvantage with S-curve acceleration/deceleration is that for the same accel/decel distance the peak acceleration/deceleration is twice that of straight line acceleration/ deceleration, which often requires twice the peak torque. Note that the axis will arrive at the target position at the same time regardless of which acceleration/deceleration method is used.

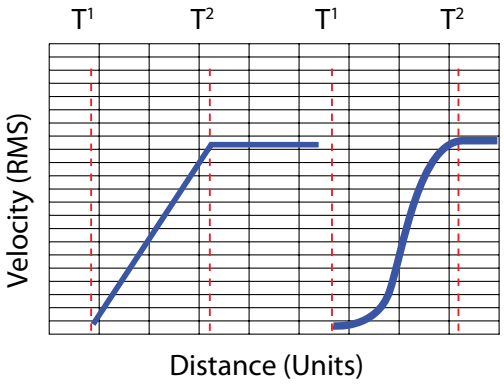


Figure 13: Sequential Move

To use S-curve acceleration/deceleration in a MOVED, MOVEP or MDV statement requires only the additional ",S" at the end of the statement.

Examples:

```
MOVED   10 , S
MOVEP   10 , S
MDV     10,20,S
MDV     10,0,S
```

1.10.6 Motion Queue

The PositionServo drive executes the User Program one statement at a time. When a move statement (MOVED or MOVEP) is executed, the move profile is stored to the Motion Queue. The program will, by default, wait on that statement until the Motion Queue has executed the move. Once the move is completed, the next statement in the program will be executed. By default motion commands (other than MDV statements) effectively suspend the program until the motion is complete.

In order for subsequent program statements to be executed during a motion command (Move, MoveD, MoveP) an additional line argument can be used. ',C' placed on the end of the move statement, for example MoveP 0,C or MoveD 100,C will continue user program execution while those motion commands are executed.

Continuous program execution during a move allows for additional move commands or motion profiles to be stored to the Motion Queue. The Motion Queue has a limit of 32 profiles and exceeding this will result in a 'Motion Stack Overflow'. The Continue "C" argument is used when it is necessary to trigger an action (e.g. handle I/O) while the motor is in motion. The following Pick and Place Example Program has been modified to utilize the Continue, "C", argument.

```
;*************************** Main Program ********************************
WAIT UNTIL IN_A3           ;Make sure the Enable input is made before continuing
ENABLE
OUT1 = 0                   ;Initialize Pick Arm - Place in Retracted Position
WAIT UNTIL IN_A4==1        ;Check Pick Arm is in Retracted Position
PROGRAM_START:
MOVEP 0                    ;Move to position 0 to pick part
OUT1 = 1                   ;Turn on output 1 to extend Pick arm
WAIT UNTIL IN_A1==1        ;Check input to make sure Arm is extended
OUT2 = 1                   ;Turn on output 2 to Engage gripper
WAIT TIME 1000             ;Delay 1 sec to Pick part
OUT1 = 0                   ;Turn off output 1 to Retract Pick arm
WAIT UNTIL IN_A4==1        ;Check input to make sure Arm is retracted
MOVED 100,C                ;Move to Place position and continue code execution
WAIT UNTIL APOS >25        ;Wait until pos is greater than 25
OUT3 = 1                   ;Turn on output 3 to spray part
WAIT UNTIL APOS >=75       ;Wait until pos is greater than or equal to 75
OUT3 = 0                   ;Turn off output 3 to shut off spray guns
WAIT UNTIL APOS >=95       ;Wait until move is almost done before extending arm
OUT1 = 1                   ;Turn on output 1 to extend Pick arm
WAIT UNTIL IN_A1==1        ;Check input to make sure Arm is extended
OUT2 =0                    ;Turn off output 2 to Disengage gripper
WAIT TIME 1000             ;Delay 1 sec to Place part
OUT1 = 0                   ;Retract Pick arm
WAIT UNTIL IN_A4==1        ;Check input to make sure Arm is retracted
GOTO PROGRAM_START
END
```

When the “C” argument is added to the standard MOVED and MOVEP statements, program execution is not interrupted by the execution of the motion command. Note: with an MDV move the execution of the program is never suspended.

Generated motion profiles are stored directly to the Motion Queue and are then executed in sequence. If the MOVED and MOVEP statements don’t have the “C” modifier, then the motion profiles generated by these statements go to the motion stack and the program is suspended until each profile has been executed.

1.11 Subroutines and Loops

1.11.1 Subroutines

Often it is necessary to repeat a series of program statements in several places in a program. Subroutines are typically used where code is used multiple times and within various sections of the main program. Subroutines are placed after the main program, i.e. after the END statement, and must start with the subname: label (where subname is the name of subroutine), and must end with a statement RETURN.

Note that there can be more than one RETURN statement in a subroutine. Subroutines are called using the GOSUB statement.

1.11.2 Loops

SML language supports WHILE/ENDWHILE block statement which can be used to create conditional loops. Note that IF-GOTO and DO/UNTIL statements can also be used to create loops.

The following example illustrates calling subroutines as well as how to implement looping by utilizing WHILE / ENDWHILE statements.

```
;****************************** Initialize and Set Variables *********************
UNITS = 1                          ;Units in Revolutions (R)
ACCEL = 15                         ;15 Rev per second per second (RPSS)
DECEL = 15                         ;15 Rev per second per second (RPSS)
MAXV = 100                         ;100 Rev per second (RPS)/6000RPM
APOS = 0                           ;Set current position to 0 (absolute zero position)
DEFINE   LOOPCOUNT      V1
DEFINE   LOOPS          10
DEFINE   DIST           V2
DEFINE   REPETITIONS    V3
REPETITIONS = 0
;******************************** Main Program *********************************
WAIT UNTIL IN_A3                   ;Make sure the Enable input is made before continuing
ENABLE
PROGRAM_START:
MAINLOOP:
        LOOPCOUNT=LOOPS            ;Set up the loopcount to loop 10 times
        DIST=10                    ;Set distance to 10
        WHILE   LOOPCOUNT          ;Loop while loopcount is greater than zero
                GOSUB   MDS        ;Call to subroutine
                WAIT TIME  100     ;Delay executes after returned from the subroutine
        LOOPCOUNT=LOOPCOUNT-1      ;decrement loop counter
        ENDWHILE
        REPETITIONS=REPETITIONS+1  ;outer loop
        IF      REPETITIONS < 5
GOTO MAINLOOP
Wait Motioncomplete                ;Wait for MDV segments to be completed
        ENDIF
END

;****************************** Sub-Routines *********************************
MDS:
      V4=dist/3
      MDV      V4,10
      MDV      V4,10
      MDV      V4,0
RETURN
```

Note: Execution of this code will most likely result in F_23. There are 3 MDV statements that are executed 10 times totaling 30 moves. Then the condition set on the repetitions variable makes the program execute the above another 4 times. 4 x 30 = 120. The 120 moves, with no waits anywhere in the program will most likely produce an F_23 fault (Motion Queue overflow). Where the possibility exists to overflow the Motion Queue additional code should be used to detect ‘Motion Queue Full’ condition and to wait for space on the Motion Queue to become available.

2. Programming

2.1 Program Structure

One of the most important aspects of programming is developing the program's structure. Before writing a program, first develop a plan for that program. What tasks must be performed? And in what order? What things can be done to make the program easy to understand and allow it to be maintained by others? Are there any repetitive procedures?

Most programs are not a simple linear list of instructions where every instruction is executed in exactly the same order each time the program runs. Programs need to perform different functions in response to external events and operator input. SML contains program control structures and scanned event functions that may be used to control the flow of execution in an application program. Control structure statements are the instructions that cause the program to change the path of execution. Scanned events are instructions that execute at the same time as the main body of the application program.

Header - *Enter in program description and title information*

```
;********************************* HEADER *********************************
;Title:           Pick and Place example program
;Author:          Lenze
;Description:     This is a sample program showing a simple sequence that
;                 picks up a part, moves to a set position and drops the part
```

I/O List - *Define what I/O will be used*

```
;******************************** I/O List  ******************************
;      Input A1    -     not used
;      Input A2    -     not used
;      Input A3    -     Enable Input
;      Input A4    -     not used
;      Input B1    -     not used
;      Input B2    -     not used
;      Input B3    -     not used
;      Input B4    -     not used
;      Input C1    -     not used
;      Input C2    -     not used
;      Input C3    -     not used
;      Input C4    -     not used
;
;      Output 1    -     Pick Arm
;      Output 2    -     Gripper
;      Output 3    -     not used
;      Output 4    -     not used
```

Initialize and Set Variables - *Define and assign Variables values*

```
;*************************** Initialize and Set Variables ****************
UNITS = 1
ACCEL = 75
DECEL =75
MAXV = 10
;V1 =
;V2 =
DEFINE Output_on 1
DEFINE Output_off 0
```

Events - *Define Event name, Trigger and Program Statements*

```
;***************************** Events ************************************
EVENT SPRAY_GUNS_ON   APOS > 25 ;Event will trigger as position passes 25 in pos dir.
   OUT3= Output_On              ;Turn on the spray guns (out 3 on)
ENDEVENT                        ;End event

EVENT SPRAY_GUNS_OFF  APOS > 75 ;Event will trigger as position passes 75 in pos dir.
   OUT3= Output_Off             ;Turn off the spray guns (out 3 off)
ENDEVENT                        ;End even
```

Main Program *- Define the motion and I/O handling of the machine*

```
;***************************** Main Program ******************************
RESET_DRIVE:                  ;Place holder for Fault Handler Routine
WAIT UNTIL IN_A3              ;Make sure the ENABLE input is made before continuing
ENABLE
OUT1 = 0                      ;Initialize Pick Arm - Place in Retracted Position
WAIT UNTIL IN_A4==1           ;Check Pick Arm is in Retracted Position
EVENT    SPRAY_GUNS_ON  ON    ;Enable the Event
EVENT    SPRAY_GUNS_OFF ON    ;Enable the Event
PROGRAM_START:
MOVEP 0                       ;Move to position 0 to pick part
OUT1 = Output_On              ;Turn on output 1 to extend Pick arm
WAIT UNTIL IN_A1==1           ;Check input to make sure Arm is extended
OUT2 = Output_On              ;Turn on output 2 to Engage gripper
WAIT TIME 1000                ;Delay 1 sec to Pick part
OUT1 = Output_Off             ;Turn off output 1 to Retract Pick arm
WAIT UNTIL IN_A4==1           ;Check input to make sure Arm is retracted
MOVED 100                     ;Move to Place position
OUT1 = Output_On              ;Turn on output 1 to extend Pick arm
WAIT UNTIL IN_A1==1           ;Check input to make sure Arm is extended
OUT2 = Output_Off             ;Turn off output 2 to Disengage gripper
WAIT TIME 1000                ;Delay 1 sec to Place part
OUT1 = Output_Off             ;Retract Pick arm
WAIT UNTIL IN_A4==1           ;Check input to make sure Arm is retracted
GOTO PROGRAM_START
END
```

Sub-Routine - *All Sub-Routine code should reside here*

```
;************************* Sub-Routines ********************************
;     Enter Sub-Routine code here
```

Fault Handler - *Define what the program should do when a fault is detected*

```
;************************ Fault Handler Routine *************************
;     Enter Fault Handler code here
ON FAULT
ENDFAULT
```

The **header section** of the program contains description information, program name, version number, description of process and programmers name. The **I/O List section** of the program contains a listing of all the I/O used within the application. The **Initialize and Set Variables section** of the program defines the names for the user variables and constants used in the program and provides initial setting of these and other variables.

The **Events section** contains all scanned events. Remember to execute the **EVENT <eventname> ON** statement in the main program to enable the events. Please note that not all of the SML statements are executable from within the EVENT body. For more detail, reference “EVENT” and “ENDEVENT” in Section 3 of the manual. The GOTO statement can not be executed from within the Event body. However, the JUMP statement can be used to jump to code in the main program body. This technique allows the program flow to change based on the execution of an event. For more detail, reference “JUMP”, in Section 3.1 (Program Statement Glossary) of this manual.

The **main program** body of the program contains the main part of the program, which can include all motion and math statements, labels, I/O commands and subroutine calls. The main body should be finished with an END statement, however, if the program loops indefinitely then the END statement can be omitted.

Subroutines are routines that are called from the main body of the program. When a subroutine is called, (GOSUB), the program’s execution is transferred from the main program to the called subroutine. It will then process the subroutine until a RETURN statement occurs. Once a RETURN statement is executed, the program’s execution will return back to the main program at the line of code immediately following the GOSUB statement.

Fault handler is the section of code that is executed when the drive detects a fault. This section of code begins with the “ON FAULT” statement and ends with an “ENDFAULT” statement. When a fault occurs, the normal program flow is interrupted, motion is stopped, the drive is disabled, Event scanning is stopped and the statements in the Fault Handler are executed. The Fault handler can be exited in two ways:

- The “RESUME” statement will cause the program to end the Fault Handler routine and return the execution to the main program. The location (label) called out in the “RESUME” command will determine where the program will commence.
- The “ENDFAULT” statement will cause the user program to be terminated.

While the Fault Handler is being executed, Events are not being processed and detection of additional faults is not possible. Because of this, the Fault Handler code should be kept as short as possible.

If extensive code must be written to process the fault, then this code should be placed in the main program and the “RESUME” statement should be utilized. Not all SML statements can be utilized by the Fault Handler. For more details reference “ON FAULT/ENDFAULT”, in Section 3.1 (Program Statement Glossary) of this manual.

Comments are allowed in any section of the program and are preceded by a semicolon. They may occur on the same line as an instruction or on a line by themselves. Any text following a semicolon in a line will be ignored by the compiler.

2.2 Variables

Variables can be System or User. User variables do not have a predefined meaning and are available to the programmer to store any valid numeric value. System variables have a predefined meaning and are used to configure, control or monitor the operations of the PositionServo. (Refer to paragraph 2.6 for more information on System Variables).

All variables can be used in valid arithmetic expressions. All variables have their own corresponding index or identification number. Any variable can be accessed by their identification number from the User’s program or from a Host Interface. In addition to identification numbers all of the variables have predefined names and can be accessed by that name from the user program.

The following syntax is used when accessing variables by their identification number:

```
@102 = 20    ; set variable #102 to 20
@88=@100     ; copy value of variable #100 to variable #88
```

Variable @102 has the variable name ‘V2’; Variable @88 has the variable name ‘VAR_AOUT’ and Variable @100 has the variable name ‘V0’. Hence the program statements above could be written as:

```
V2 = 20
VAR_AOUT = V0
```

Using variable names rather than identification numbers creates code that is more easily read and understood.

There are two types of variables in the PositionServo drive - **User Variables** and **System Variables**.

User Variables are a fixed set of variables that the programmer can use to store data and perform arithmetic calculations. All variables are of a single type. Single type variables, i.e. typeless variables, relieve the programmer of the task of remembering to apply conversion rules between types, thus greatly simplifying programming.

User Variables

V0-V31 User defined variables. Variables can hold any numeric value including logic (Boolean 0 - FALSE and non 0 - TRUE) values. They can be used in any valid arithmetic or logical expressions.

NV0-NV31 User defined network variables. Variables can hold any numeric value including logic (Boolean 0 - FALSE and non 0 - TRUE) values. They can be used in any valid arithmetic or logical expressions. Variables can be shared across Ethernet network with use of statements SEND and SENDTO.

Since SML is a typeless language, there is no special type for Boolean type variables (variables that can be only 0 or 1). Regular variables are used to facilitate Boolean variables. Assigning a variable a “FALSE” state is done by setting it equal to “0”. Assigning a variable a “TRUE” state is done by assigning it any value other than “0”.

Scope

SML variables are accessible from several sources. Each of the variables can be read and set from the user program or Host communications interface at any time. There is no provision to protect a variable from change. This is referred to as global scope.

Volatility

User variables are volatile i.e. they don’t maintain their values after the drive is powered down. After power up the values of the user variables are set to 0. Loading or resetting the user program doesn’t reset variables values. Two programming statements are provided should the programmer wish to implement some non-volatile memory storage within their application (the LoadVars and StoreVars Statements - refer to section 3.1).

In addition to the user variables, system variables are also provided. System variables are dedicated variables that contain specific information relative to the set-up and operation of the drive. For example, **APOS** variable holds actual position of the motor shaft. For more details refer to Section 2.9.

Resolution and Accuracy

Any variable can be used as a condition in a conditional expression. Variables are often used to indicate that some event has occurred, logic state of an input has changed or that the program has executed to a particular point. Variables with non ‘0’ values are evaluated as “TRUE” and variables with a “0” value are evaluated as “FALSE”.

Variables are stored internally as 4 bytes (double word) for integer portion and 4 bytes (double word) for fractional portion. Every variable in the system is stored as 64 bit in 32.32 fixed point format. Maximum number can be represented by this format is +/- 2,147,483,648. Variable resolution in this format is 2.3E-10.

2.3 Arithmetic Expressions

Table 7 lists the four arithmetic functions supported by the Indexer program. Constants as well as User and System variables can be part of the arithmetic expressions.

Examples.

```
V1 = V1+V2              ;Add two user variables
V1 = V1-1               ;Subtract constant from variable
V2 = V1+APOS            ;Add User and System (actual position) variables
APOS = 20               ;Set System variable
V5 = V1*(V2+V3*5/3)     ;Complicated expression
```

Table 7: Supported Arithmetic Expressions

Operator	Symbol
Addition	+
Subtraction	-
Multiplication	*
Division	/

Register (variable) overflow for "*" and "/" operations will cause arithmetic overflow fault F_19. Register (variable) overflow/underflow for "+" and "-" operations does not cause an arithmetic fault.

2.4 Logical Expressions and Operators

Bitwise, Boolean, and comparison operators are referred to as Logical Operators. Bitwise operators are used to change individual bits within an operand (variable). Bitwise operation works at the binary level of the variables, changing specified bits or bit patterns within those variables.

Boolean operators are used to combine simple or complex expressions within a single logic statement. They are used to define a condition that ultimately equates to either True or False.

Comparison operators are used to perform a test between two values and to return a result indicating whether or not the test (Comparison) evaluates to true or false.

2.4.1 Bitwise Operators

Table 8 lists the bitwise operators supported by the Indexer program.

Table 8: Supported Bitwise Operators

Operator	Symbol
AND	&
OR	|
XOR	^
NOT	!

Both User or System variables can be used with these operators. In order to perform a bitwise (Boolean) operation, the value often easier in entered in hexadecimal format. To enter a number in hexadecimal use the characters '0x' immediately prior to the hexadecimal number. Example: bit 22 alone would be referenced as 0x400000.

Examples:

```
V1 = V2 & 0xF             ;clear all bits but lowest 4
IF  (INPUTS & 0x3)        ;check inputs 0 and 1
V1 = V1 | 0xff            ;set lowest 8 bits
V1 = INPUTS ^ 0xF         ;invert inputs 0-3
V1 = !IN_A1               ;invert input A1
```

2.4.2 Boolean Operators

Table 9 lists the boolean operators supported by the Indexer program. Boolean operators are used in logical expressions.

Table 9: Supported Boolean Operators

Operator	Symbol
AND	&&
OR	||
NOT	!

Examples:

```
IF (APOS >2 && APOS <6) || (APOS >10 && APOS <20)
            {statements if true}
      ENDIF
```

The above example checks if APOS (actual position) is within one of two windows; 2 to 6 units or 10 to 20 units. In other words:

If (APOS is more than 2 AND less than 6)
OR
If (APOS is more than 10 AND less then 20)
THEN the logical expression is evaluated to TRUE. Otherwise it is FALSE

2.5 Comparison Operators

Table 10 lists the comparison operators supported by the Indexer program.

Table 10: Supported Comparison Operators

Operator	Symbol
More	>
Less	<
Equal or more	>=
Equal or less	=<
Not Equal	<>
Equal	==

Examples:

```
IF APOS <=10          ;If Actual Position equal or less than 10
IF APOS > 20          ;If Actual Position greater than 20
IF V0==5              ;If V0 equal to 5
IF  V1<2  && V2 <>4   ;If V1 less than 2 And V2 doesn't equal 4
```

2.6 System Variables and Flags

System variables are variables that have a predefined meaning. They give the programmer/user access to drive parameters and functions. Some of these variables can also be set via the parameters in MotionView. In most cases the value of these variables can be read and set in the user program or via a Host Interface. Variables are either read only, write only or read and write. Read only variables can only be read and can't be set. For example, INPUTS = 5, is an illegal action because you can not set an input. Conversely, write-only variables cannot be read. Reading a write-only variable by either the variable watch window or network communications can result in erroneous data.

System Flags are predefined bits that are used by a program either to remember something or to signal some condition. Flags are binary values so contain only values 1 or 0 (True or False). For example, IN_A1 is the system flag that reflects the state of digital input A1. Since inputs can only be ON or OFF, then the value of IN_A1 can only be 0 or 1.

2.7 System Variables Storage Organization

The PositionServo drive contains dual variable storage locations, the operational memory (RAM), that is the volatile operating memory, and the EPM memory, that is the non-volatile configuration memory. When the PositionServo is turned on it copies the retained settings from the EPM non-volatile memory into the RAM memory for use during program execution.

When a system variable is changed during normal program execution its value is changed only in the RAM memory and subsequently these values are lost following power down. System variables that are changed through the MotionView parameter set are stored in both EPM and RAM memory so changes have both immediate effect and are retained after power down. The StoreVars command (Refer to section 3.1) can be used to store the user variables (V0-V31) from RAM memory into the EPM memory during program execution so the programmer has the oppertunity to retain these after power down.

Host Interfaces have the capability of changing all of the system variable values through any one of the adopted communications protocols available for PositionServo. Communications protocols contain mechanisms to write to RAM memory only, or to RAM memory and EPM memory.

NOTE:

EPM memory is specified for a limited number of write cycles (approximately 1 million). Care must be taken not to excessively write to the EPM memory or not to exceed the maximum write cycle count.

2.7.1 RAM File for User's Data Storage

In addition to the standard user variables (V0-V31 & NV0-NV31) MotionView OnBoard drives have a section of RAM memory (256k) allocated as data storage space and available to the programmer for storage of program data.

The RAM file data storage is often required in systems where it is desirable to store large amounts of data prepared by a host controller (PLC, HMI, PC, etc). This data might represent more complex Pick and Place coordinates, complicated trajectory coordinates, or sets of gains/limits specific for given motion segments.

RAM memory is also utilized in applications that require data collection during system operation. At the end of a period of time the collected data can be acquired by the host controller for analysis. For example, position errors and phase currents collected during the move are then analyzed by the host PLC/PC to qualify system tolerance to error free operation.

Implementation

There are 256K (262,144) bytes provided as RAM file for data storage. Since the basic data type in the drive is 64 bit (8 bytes) 32,768 data elements can be stored in the RAM file. The file is accessible from within the User's program or through any external communications interface (Ethernet, ModBus, CAN etc.). Two statements and three system variables are provided for accessing the RAM file memory. The RAM file is volatile storage and is intended for "per session" usage. The data saved in the RAM file will be lost when the drive is powered off.

The three system variables provided to support file access are:

VAR_MEM_VALUE	(PID = 4)
VAR_MEM_INDEX	(PID = 5)
VAR_MEM_INDEX_INCREMENT	(PID = 6)

In addition, two statements are provided to allow access and storage to the RAM file direct from the user program. The statements MEMSET, MEMGET are described in paragraph 2.7.3 and Tables 44 & 45.

2.7.2 Memory Access Through Special System Variables

VAR_MEM_VALUE holds the value that will be read or written to the RAM file. VAR_MEM_INDEX points to the position in the RAM file (0 to 32767) that data will be read from or written to, and VAR_MEM_INDEX_INCREMENT holds the value that will be modified after the read or write operation is completed.

The RAM memory access is illustrated with the example program herein.

```
;-------------------------------------------------------------------------
;User's program to read/write to RAM file.
;Advance index after writing/reading by 1
;Record position error to RAM file every 100 ms for 10 seconds. 10/0.1 = 100
;locations are needed
;-------------------------------------------------------------------------

DEFINE        IndexStart       0
DEFINE        MemIncrement     1
DEFINE        RecordLength     100
DEFINE        PElimit          0.1                          ;0.1 user unit

VAR_MEM_INDEX = IndexStart                                  ;set start position
VAR_MEM_INDEX_INCREMENT=MemIncrement                        ;set increment

;-------------------------------------------------------------------------
EVENT  StorePE TIME 100

              VAR_MEM_VALUE = VAR_POSERROR                  ;store in RAM file.

ENDEVENT

PROGRAMSTART:

              EVENT StorePE ON

                                {
                                          Start some motion activity….

                                }
;wait until data collection is over

WHILE    VAR_MEM_INDEX <    (IndexStart+RecordLength)
ENDWHILE
EVENT StorePE Off                                           ;turn off storage

;Analyze data collected. If PE > PElimit then signal system has low performance…
VAR_MEM_INDEX= IndexStart
WHILE   VAR_MEM_INDEX <   (IndexStart+RecordLength)
   IF (VAR_MEM_VALUE > PElimit)
      GOTO Label_SignalBad
   ENDIF
ENDWHILE

LabelSignalBad:

                                {
                                          Signal that PE out of limits
                                                    …
                                }

END
```

In the RAM memory access program example, the values of PE (position error) are stored sequentially in the RAM file every 100ms for 10 seconds. (100 samples). After collection is done the data is read from the file one by one and compared with limit value set.

Variable VAR_MEM_INDEX is incremented every read or write by the value stored in VAR_MEM_INDEX_INCREMENT. This could be any value from -32767 to 32767. This allows for decrement through storage locations in the RAM file in addition to Increment. If the value is 0 (zero) no increment/decrement is produced. Var_Mem_Index is a modular variable (it wraps around it maximum or minimum values). I.e. if the next increment or decrement of Var_Mem_Index results in a value beyond the modulus (32767 or -32767) then the variable will wrap around to the opposite end of the variable range. This allows for the creation of circular arrays. This feature can be used for diagnostics when certain parameter(s) are stored in the memory continuously and then, if the system fails, the data array can be examined to simplify diagnostics.

2.7.3 Memory Access Through MEMSET, MEMGET Statements

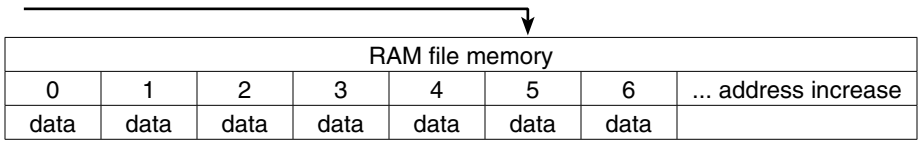
The memory access statements MEMSET and MEMGET are provided for simplified transfer of data between the RAM memory and the user variables V0-V31. Using these statements, any combinations of variables V0-V31 can be stored/ retrieved with a single statement. This allows for efficient access to the RAM memory area. For example, reading 10 values from RAM memory and storing them in 10 user variables using the system variables would normally require 10 separate program statements (Vx = Var_Mem_Value). With the MEMGET statement all 10 user variables can be read in one program statement. The format of MEMSET/MEMGET is as follows:

MEMSET	<offset> [<varlist>]
MEMGET	<offset> [<varlist>]
<offset>	any valid expression that evaluates to a number between -32767 to 32767 This specifies the offset in the RAM file where data will be stored or retrieved.
<varlist>	any combinations of variables V0-V31

Examples for <offset> expression

5	constant	
10+23+1	constant expression	
V0	variable	Must hold values in -32767 to 32767 range
V0+V1+3	expression	Must evaluate to -32767 to 32767 range

Example: <offset> =5



Examples for <varlist> instruction

[V0]	single variable will be stored/retreived
[V0,V3,V2]	variables V0,V3,V2 will be stored/retrieved
[V3-V7]	variables V3 to V7 inclusively will be stored/retrieved
[V0,V2,V4-V8]	variables V0,V2, V4 through V8 will be stored/retrieved

Storage/Retrieval order with MEMSET/MEMGET

Variables in the list are always stored in order: the variable with lowest index first and the variables with highest index last regardless of the order they appear in the <varlist> argument.

Example: [V0,V3, V5-V7] will be stored in memory in the order of increasing memory index as follows:

RAM file memory							
V0	V3	V5	V6	V7	...	...	... index increase

For comparison: [V5-V7, V0, V3] will have the same storage order as the above list regardless of the order in which the variables are listed.

When retrieving data with MEMGET statements memory locations will be sequentially copied to variables starting from the one with lowest index in the list to the last with highest index. Consider the list for the MEMGET statement:
[V2, V5-V7, V3]

RAM file memory							
Data1	Data2	Data3	Data4	Data5	Data6	...	... index increase

Here is how the data will be assigned to variables:
V2 <- Data1
V3 <- Data2
V5 <- Data3
V6 <- Data4
V7 <- Data5

2.7.4 Store and Retrieve Variables from the EPM

The EPM access statements LOADVARS and STOREVARS are provided to store/retrieve the values of the user variables, V0-V31, to/from the EPM. The LOADVARS statement loads the stored values of the users variables V0-V31 from the EPM. Variable values V0-V31 can be previously stored via the interface or the STOREVARS statement. The STOREVARS statement stores the values of the user variables V0-V31 to the EPM. Variable values V0-V31 can be later retrieved via the interface or the LOADVARS statement. Refer to the Program Statement Glossary in section 3.1 for syntax and example details.

NOTE

At Bootup, variables V0-V31 are automatically retrieved from the EPM.

NOTE

EPM memory is specified for a limited number of write cycles (approximately 1 million). Care must be taken not to excessively write to the EPM memory or not to exceed the maximum write cycle count.

2.8 System Variables and Flags Summary

2.8.1 System Variables

Section 3.2 provides a complete list of the system variables. Every aspect of the PositionServo can be controlled by the manipulation of the values stored in the System Variables. All System Variables start with a “VAR_” followed by the variable name. Alternatively, System Variables can be addressed as an @NUMBER where the number is the variable Index. The most frequently used variables also have alternate names as listed in Table 11.

Table 11: System Variables

Index	Variable	Access	Variable Description	Units
181	ACCEL	R/W	Acceleration for motion commands	User Units/Sec^2
71	AIN1	R	Analog input. Scaled in volts. Range from -10 to +10 volts	V(olt)
72	AIN2	R	Analog input 2. Scaled in Volts. Range from -10 to +10 volts	V(olt)
88	AOUT	R/W	Analog output. Value in Volts. Valid range from -10 to +10 (V)(2)	V(olt)
215	APOS	R/W	Actual motor position	User Units
190	APOS_PLS	R/W	Actual Motor Position	Encoder Counts
182	DECEL	R/W	Deceleration for motion commands	User Units/Sec^2
83	DEXSTATUS	R	Drive Extended Status Word	-
54	DSTATUS	R	Status flags register	-
	DFAULTS	R	Fault code register	-
245	HOME	W	Start Homing (pre-defined homing)	-
	INDEX	R	Lower 8 bits are used. See ASSIGN statement for details.	-
184	INPOSLIM	R/W	Maximum deviation of position for INPOSITION Flag to remain set	User Units
65	INPUTS	R	Digital Inputs states. The first 12 bits correspond to the 12 drive inputs	-
139	IREF	W	Internal Reference: Velocity / Torque	RPS/A
187	MECOUNTER	R	Master Encoder Counts (Master Encoder Input)	Encoder Counts
180	MAXV	R/W	Maximum velocity for motion commands	User Units/Sec
140-171	NV0 - NV31	R/W	User Network Variables	-
66	OUTPUTS	R/W	Digital outputs. Bits #0 to #4 represent outputs 1 through 5	-
216	PERROR	R	Position Error	Feedback Pls
191	PERROR_PLS	R	Position Error	User Units
48	PGAIN_D	R/W	Position loop D-gain	-
47	PGAIN_I	R/W	Position loop I-gain	-
49	PGAIN_ILIM	R/W	Position loop I gain limit	-
46	PGAIN_P	R/W	Position loop P-gain	-
188	PHCUR	R	Motor phase current	A(mpere)
183	QDECEL	R/W	Quick Deceleration for STOP MOTION QUICK statement	User Units/Sec^2
213	RPOS	R	Registration position. Valid when system flag F_REGISTRATION set	User Units
212	RPOS_PLS	R	Registration position	Feedback Pls
218	TA	R	Commanded acceleration	User units/Sec^2
214	TPOS	R/W	Theoretical/commanded position	User Units
219	TPOS_ADV	W	Theoretical/commanded position advance	Feedback Pls
189	TPOS_PLS	R/W	Theoretical/commanded position	Feedback Pls
217	TV	R	Commanded velocity in	User Units/Sec
186	UNITS	R/W	User Units scale.(1)	UserUnits/Rev
185	VEL	R/W	Set Velocity when in velocity mode	User Units/Sec
44	VGAIN_P	R/W	Velocity loop P-gain	-
45	VGAIN_I	R/W	Velocity loop I-gain	-
100-131	V0 - V31	R/W	User Variables	

(1) When a “0”, (zero), value is assigned to the variable “UNITS”, then “USER UNITS” is set to QUAD ENCODER COUNTS.

(2) Any value outside +/- 10 range assigned to AOUT will be automatically trimmed to that range.

Example:

```
AOUT=100 , AOUT will be assigned value of 10.
V0=236
VOUT=V0, VOUT will be assigned 10 and V0 will be unchanged.
```

2.8.2 System Flags

Flags don't have an Index number assigned to them. They are the product of a BIT mask applied to a particular system variable within the drive and are available to the programmer only from the User's program. Table 12 lists the System Flags with access rights and description.

Table 12: System Flags

Name	Access	Description
IN_A1-4, IN_B1-4, IN_C1-4	R	Digital inputs . TRUE if input active, FALSE otherwise
OUT1, OUT2, OUT3, OUT4, OUT5	W	Digital outputs OUTPUT1- OUTPUT5
F_ICONTROLOFF	R	Interface Control Status (ON/OFF) #27 in DSTATUS register
F_IN_POSITION	R	TRUE when Actual Position (APOS) is within limits set by INPOSLIM variable and motion completed
F_ENABLED	R	Set when drive is enabled
F_EVENTSOFF	R	Events Disabled Status (ON/OFF) #30 in DSTATUS register
F_MCOMPLETE	R	Set when motion is completed and there are no motion commands waiting in the Motion Queue
F_MQUEUE_FULL	R	Motion Queue full
F_MQUEUE_EMPTY	R	Motion Queue empty
F_FAULT	R	Set if any fault detected
F_ARITHMETIC_FLT	R	Arithmetic fault
F_REGISTRATION	R	Set when registration mark is detected. Contents of the RPOS variable valid when this flag is active. Flag reset by any registration moves MOVEPR, MOVEDR or by command REGISTRATION ON
F_MSUSPENDED	R	Set if motion suspended by statement MOTION SUSPEND

Flag logic is shown herein.

```
IF  (TPOS-INPOSLIM < APOS) && (APOS < TPOS+INPOSLIM) && F_MCOMPLETE && F_MQUEUE_EMPTY
      Out1 = 1
ELSE
      Out1 = 0
ENDIF
```

For VELOCITY mode F_MCOMPLETE and F_MQUEUE_EMPTY flags are ignored and assumed TRUE.

2.9 Control Structures

Control structures allow the user to control the flow of the program's execution. Most of the control and flexibility of any programming language comes from its ability to change statement order with structure and loops.

2.9.1 IF Structure

The flowchart and code segment in Figure 17 illustrate the use of the IF statement. The "IF" statement is used to execute an instruction or block of instructions one time if a condition is true. The simplified syntax for the IF statement is:

```
IF condition
      …statement(s)
ENDIF
```

```
…statements

IF IN_A2
      OUT2 = 1
      MOVED 3
ENDIF

..statements
```

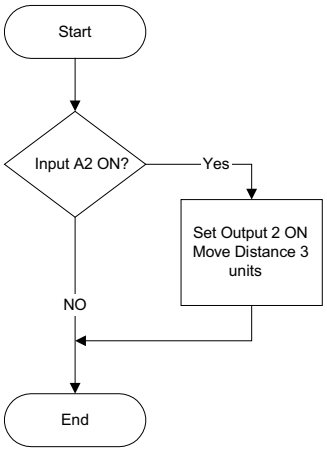


Figure 17: IF Code and Flowchart

IF/ELSE

The flowchart and code segment in Figure 18 illustrate the use of the IF/ELSE instruction. The IF/ELSE statement is used to execute a statement or a block of statements one time if a condition is true and a different statement or block of statements if condition is false. The simplified syntax for the IF/ELSE statement is:

```
IF <condition>
      …statement(s)
ELSE
      …statement(s)
ENDIF
```

```
…statements

IF IN_A2
      OUT2=1
      MOVED 3
ELSE
      OUT2=0
      MOVED 5
ENDIF

..statements
```

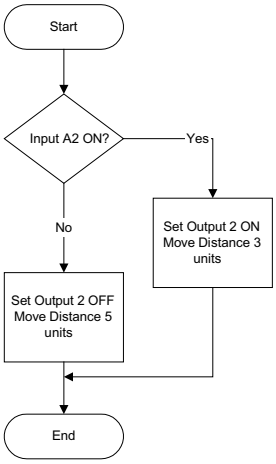


Figure 18: IF/ELSE Code and Flowchart

2.9.2 DO/UNTIL Structure

The flowchart and code segment in Figure 14 illustrate the use of the DO/UNTIL statement. This statement is used to execute a block of code one time and then continue executing that block until a condition becomes true (satisfied). The difference between DO/UNTIL and WHILE statements is that the DO/UNTIL instruction tests the condition after the block is executed so the conditional statements are always executed at least one time. The syntax for DO/UNTIL statement is:

```
DO
      …statements
UNTIL  <condition>
```

```
… statements

DO
        MOVED 3
        WAIT TIME 2000
UNTIL   IN_A3
…statements
```

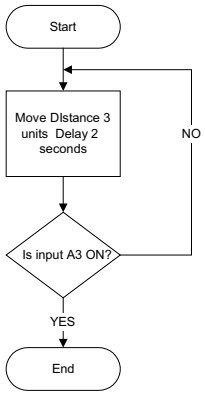


Figure 14: DO/UNTIL Code and Flowchart

2.9.3 WHILE Structure

The flowchart and code segment in Figure 15 illustrate the syntax for the WHILE instruction. This statement is used if you want a block of code to execute while a condition is true.

```
WHILE  <condition>

      …statements

ENDWHILE
```

```
…statements

WHILE    IN_A3
      MOVED 3
      WAIT TIME   2000
ENDWHILE

…statements
```

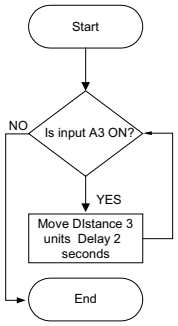


Figure 15: WHILE Code and Flowchart

2.9.4 WAIT Statement

The WAIT statement is used to suspend program execution until or while a condition is true, for a specified time period (delay) or until motion has been completed. The simplified syntax for this statement is:

```
WAIT  UNTIL <condition>
WAIT  WHILE <condition>
WAIT  TIME  <time>
WAIT  MOTION COMPLETE
```

2.9.5 GOTO Statement and Labels

The GOTO statement can be used to transfer program execution to a section of the Main Program identified by a label. This statement is often executed conditionally based on the logical result of an If Statement. The destination label may be above or below the GOTO statement in the application program.

Labels must be an alphanumeric string of up to 64 characters in length, ending with a colon ":" and containing no spaces.

```
GOTO TestInputs
      …statements
TestInputs:
      …statements
IF (IN_A1) GOTO TestInputs
```

Table 13 provides a short description of the instructions used for program branching.

Table 13: Program Branching Instructions

Name	Description
GOTO	Transfer code execution to a new line marked by a label
DO/UNTIL	Do once and keep doing until conditions becomes true
IF and IF/ELSE	Execute once if condition is true
RETURN	Return from subroutine
WAIT	Wait fixed time or until condition is met
WHILE	Execute while a condition is true
GOSUB	Go to Subroutine

2.9.6 Subroutines

A subroutine is a group of SML statements that is located at the end of the main body of the program. It starts with a label which is used by the GOSUB statement to call the subroutine and ends with a RETURN statement. The subroutine is executed by using the GOSUB statement in the main body of the program. Subroutines can not be called from an EVENT or from the FAULT handler.

When a GOSUB statement is executed, program execution is transferred to the first line of the subroutine. The subroutine is then executed until a RETURN statement is met. When the RETURN statement is executed, the program's execution returns to the program line (in the main program) following the GOSUB statement. A subroutine may have more than one RETURN statement in its body.

Subroutines may be nested up to 32 times. Only the main body of the program and subroutines may contain a GOSUB statement. Refer to Section 3.1 for more detailed information on the GOSUB and RETURN statements. The flowchart and code segment in Figure 16 illustrate the use of subroutines.

```
…statements
GOSUB CalcMotionParam
MOVED V1
OUT2=1
…statements
END
;
CalcMotionParam:
V1 = (V3*2)/V4
RETURN
```

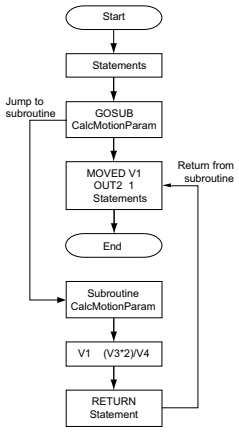


Figure 16: GOSUB Code and Flowchart

2.10 Scanned Event Statements

A Scanned Event is a small program that runs independently of the main program. SCANNED EVENTS are very useful when it is necessary to trigger an action (i.e. handle I/O) while the motor is in motion or other tasks within the Main Program are executing. When setting up Events, the first step is to define both the action that will trigger the event as well as the sequence of statements to be executed once the event has been triggered. Events are scanned every 512µs. Before an Event can be scanned however it must be enabled. Events can be enabled or disabled from the user program or from another event (see explanations below). Once the Event is defined and enabled, the Event will be constantly scanned until the trigger condition is met, this scan rate is independent of the main program's timing. Once the trigger condition is met, the Event statements will be executed independently of the user program.

Scanned events are used to record events and perform actions independent of the main body of the program. For example, if the programmer wants output 3 to come ON when the position is greater then 4 units, or if he needs to turn output 4 ON whenever inputs A4 and B1 are ON, he could use the following scanned event statements.

```
EVENT       PositionIndicator APOS > 4
            OUT3=1
ENDEVENT

EVENT       InputsLogic       IN_A4 & IN_B1
            OUT4=1
ENDEVENT
```

Scanned events may also be used with a timer to perform an action on a periodic time basis.

The program statements contained in the scanned event code cannot include any that are related to the command of Motion from the motor or that result in a delay to program execution. A full list of illegal event code statements is given in section 3.1. Syntax for defining Events is as follows.

EVENT <name> INPUT <inputname> RISE

This scanned event statement is used to execute a block of code each time a specified input <inputname> changes its state from low to high.

EVENT <name> INPUT <inputname> FALL

This scanned event statement is used to execute a block of code each time a specified input <inputname> changes its state from high to low.

EVENT <name> TIME <timeout>

This scanned event statement is used to execute a block of code with a repetition rate specified by the <timeout> argument. The range for "timeout" is 0 - 50,000ms (milliseconds). Specifying a timeout period of 0 ms will result in the event running every event cycle (512µs).

EVENT <name> expression

This scanned event statement is used to execute a block of code when the expression evaluates as true.

EVENT <name> ON/OFF

This statement is used to enable/disable a scanned event.

Table 14 contains a summary of instructions that relate to scanned events. Refer to Section 3 "Language Reference" for more detailed information.

Table 14: Scanned Events Instructions

Name	Description
EVENT <name> ON/OFF	enable / disable event
EVENT <name> INPUT <inputname> RISE	Scanned event when <input name> goes low to high
EVENT <name> INPUT <inputname> FALL	Scanned event when <input name> goes high to low
EVENT <name> TIME <value>	Periodic event with <value> repetition rate.
EVENT <name> expression	Scanned event on expression = true

2.11 Motion

2.11.1 How Moves Work

The position command that causes motion to be generated comes from the profile generator or profiler for short. The profile generator is used by the MOVE, MOVED, MOVEP, MOVEPR, MOVEDR and MDV statements. MOVE commands generate motion in a positive or negative direction, while or until certain conditions are met. For example you can specify a motion while a specific input remains ON (or OFF). MOVEP generates a move to specific absolute position. MOVED generates incremental distance moves, i.e. move some distance from its current position. MOVEPR and MOVEDR are registration moves. MDV commands are used to generate complicated profiles. Profiles generated by these commands are put into the motion stack which is 32 level. By default when one of these statements (except for MDV) is executed, the execution of the main User Program is suspended until the generated motion is completed. Motion requests generated by an MDV statement, or by MOVE statement with the “C” modifier do not suspend the program. All motion statements are put into the motion stack and executed by the profiler in the order in which they where loaded. The Motion Stack can hold up to 32 moves. The SML language allows the programmer to load moves into the stack and continue on with the program. It is the responsibility of the programmer to check the motion stack to make sure there is room available before loading new moves. This is done by checking the appropriate bits in the System status register or the appropriate system flag.

2.11.2 Incremental (MOVED) and Absolute (MOVEP) Motion

MOVED and MOVEP statements are used to create incremental and absolute moves respectively. The motion that results from these commands is by default a trapezoidal velocity move or an S-curved velocity move if the “,S” modifier is used within the statement.

For example:

```
MOVEP 10      ;will result in a trapezoidal move
```

But

```
MOVEP 10,S    ;will result in an S-curved move
```

In the above example, (MOVEP 10), the length of the move is determined by the argument following the MOVEP command, (10). This argument can be a number, a variable or any valid arithmetic expression. The maximum velocity of the move is determined by setting the system variable MAXV. The acceleration and deceleration are determined by setting the system variables ACCEL and DECEL respectively.

If values for velocity, acceleration and deceleration, for a specified distance, are such that there is not enough time to accelerate to the specified velocity, the motion profile will result in triangular or double S profile Full Stop. The following code extract generates the motion profiles shown in Figure 19.

```
ACCEL =  200
DECEL =  200
MAXV  =  20
MOVED 4           ;Move 1
MOVED 1.5         ;Move 2
MOVED 4 , S       ;Move 3
MOVED 1.5 , S     ;Move 4
```

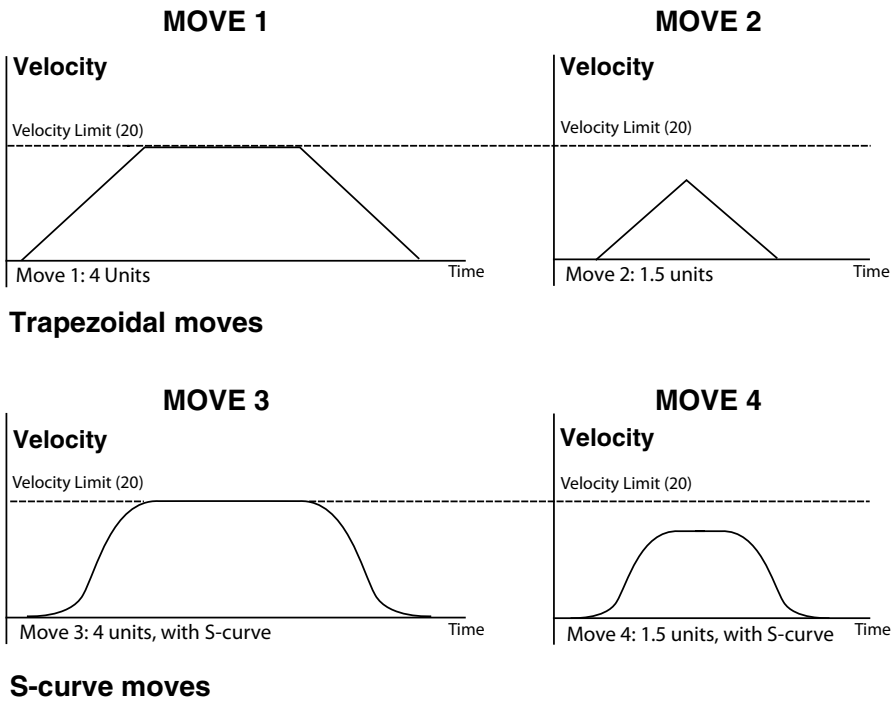


Figure 19: Move Illustration

All four of the moves shown in Figure 19 have the same Acceleration, Deceleration and Max Velocity values. Moves 1 and 3 have a larger value for the move distance than Moves 2 and 4. In Moves 1 and 3 the distance is long enough to allow the motor to accelerate to the profiled max velocity and maintain that velocity before decelerating down to a stop. In Moves 2 and 4 the commanded distance is so small that the calculated point of deceleration occurs before the motor has reached the profiled Maximum velocity. On reaching the calculated deceleration point the drive will start decelerating the motor in order to arrive at the commanded target position.

2.11.3 Incremental (MOVED) Motion

Incremental motion is defined as a move of some distance from the current position. 'Move four revolutions from the current position' is an example of an incremental move.

MOVED is the statement used to create incremental moves. The simplified syntax is:

MOVED <+/-distance>

+/- sign will tell the drive in which direction to move the motor shaft.

2.11.4 Absolute (MOVEP) Move

Absolute motion is defined as motion that is always specified relative to the same 'known' location. The location that each move is specified relative to is termed the zero (0) position. For example an absolute move of 20 will result in a move to a position that is 20 user units from the zero position regardless of whether the current shaft location is less than or greater than this commanded position (required motion is forward or reverse). The Zero position is normally established during a homing cycle performed after power up where the programmer specifies (using a switch or other device) a known point within the system mechanics from where they will reference all further motion.

If an incremental move is repeated (e.g. MoveD 10) then a subsequent move will result as motion is relative to the position of the shaft at the point the motion is initiated. If an absolute move is repeated (e.g. MoveP 10) then only one motion is executed as the subsequent target position commanded is already equal to the motor shaft's current position.

2.11.5 Registration (MOVEDR MOVEPR) Moves

MovePR and MoveDR are move commands subject to (modified by) the drive registration input (C3) activating. They are defined as registration moves as their function is to capture a position based on a sensor input and then move to a subsequent position determined by the captured position plus an offset. Registered move commands contain two motion arguments, the first defining the initial move to attempt detection of registration, and the second defining the modified motion to complete subject to registration being detected.

The difference between MoveDR and MovePR is that MoveDR is incremental and performs the initial move subject to its current position while checking for registration. MovePR is absolute so initial target position (motion) is referenced to the absolute zero position.

If registration is not detected during a MoveDR or MovePR command then the initial move commanded by the first motion argument will be completed and the registration flag will not be set. If registration is detected then both MoveDR or MovePR will modify target position to the captured registration position (stored in the RPOS variable) plus the second motion argument. If registration is detected then the registration flag will be set to true (1).

MOVEPR and MOVEDR are used to move to position or distance respectively just like MOVEP and MOVED. The difference is that while the statements are being executed they are looking for a registration signal or registration input (C3). If during the motion a registration signal is detected, then a new end position is generated. With both the MoveDR and MovePR statements the drive will increment the distance called out in the registration argument. This increment will be referenced from the position where the registration input has detected.

Example:

```
MOVEDR  5, 1  ;Statement moves a distance of 5 user units or registration position +
              ;1 user units if registration input is activated during motion.
```

There are two exceptions to the behavior of registration moves.

Exception one:

The move will not be modified to “Registration position +displacement” if the registration was detected while system was decelerating to complete the initial motion command.

Exception two:

Once the registration input is detected, there must be enough distance set by the second argument to allow for the motor to decelerate to a stop using the profiled Decel Value. If the modified registration move is smaller than the distance necessary to come to a stop, then the motor will overshoot the programmed registration position. Over-shoot of the target position is not rectified automatically, either realistic arguments must be entered for the registered move command and deceleration rate or a comparison statement used to detect and rectify over-shoot.

2.11.6 Segment Moves

In addition to the simple moves that can be generated by MOVED and MOVEP statements, complex profiles can be generated using segment moves. A segment move represents one portion of a complete move. A complete move is constructed out of two or more segments, starting and ending at zero velocity.

2.11.7 MDV Segments

Profiles are created using a sequence of MDV statements. The simplified syntax for the **MDV** (**M**ove **D**istance with **V**elocity) statement is:

MDV <**d**istance>,<**v**elocity>

The <distance> is the total distance completed during the segment move. The <velocity> is the target velocity for the end of the segment move. The starting velocity is either zero or the final velocity of the previous segment. The final segment in a complete profile must have a velocity of zero. If the final segment has a velocity other than zero, a motion stack under flow fault will occur (F_24).

The profile shown in Figure 20 can be broken up into 8 MDV moves. The first segment defines the distance between point 1 and point 2 and the velocity at point 2. So, if the distance between point 1 and 2 was 3 units and the velocity at point 2 was 56 Units/S, the command would be: MDV 3 , 56. The second segment gives the distance between point 2 and 3 and the velocity at point 3, and so on.

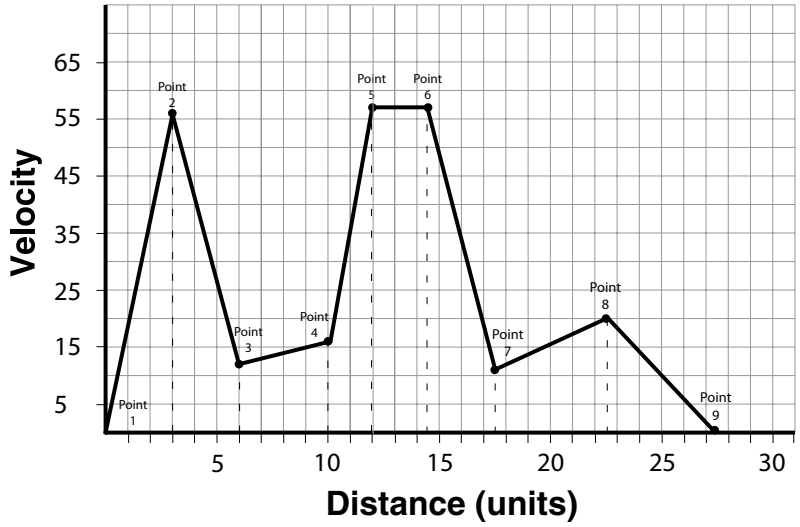


Figure 20: MDV Segment Example

Table 15 lists the supporting data for the graph in Figure 20.

Table 15: MDV Segment Example

Segment Number	Distance moved during segment	Velocity at the end of segment
1	3	56
2	3	12
3	4	16
4	2	57
5	2.5	57
6	3	11
7	5	20
8	5	0
-	-	-

```
;Segment moves
MDV   3 , 56
MDV   3 , 12
MDV   4 , 16
MDV   2 , 57
MDV   2.5 , 57
MDV   3 , 11
MDV   5 , 20
MDV   5 , 0
END
```

The following equation can be used to calculate the acceleration / deceleration that results from a segment move.

Accel = $(V_f^2 - V_0^2) / [2*D]$
V_f = Final velocity
V_0 = Starting velocity
D = Distance

2.11.8 S-curve Acceleration/Deceleration

Instead of using a linear acceleration/deceleration, the motion created using segment moves (MDV statements) can use S-curve acceleration/deceleration. The syntax for MDV move with S-curve acceleration/deceleration is:

MDV <distance>,<velocity>,S

Segment moves using S-curve acceleration/deceleration will take the same amount of time as linear acceleration/deceleration segment moves. S-curve acceleration/deceleration is useful because it is much smoother at the beginning and end of the segment, however, the peak acceleration/deceleration of the segment will be twice as high as the acceleration/deceleration used in the linear acceleration/deceleration segment.

2.11.9 Motion SUSPEND/RESUME

At times it is necessary to control motion by preloading the motion stack with motion profiles and then executing them consecutively, based on the user program and/or some logical condition being detected. The statement “MOTION SUSPEND” will suspend motion until the statement “MOTION RESUME” is executed. While motion is suspended, any motion statement executed by the User Program will be loaded into the motion stack. When the “MOTION RESUME” statement is executed, the preloaded motion profiles will be executed in the order that they were loaded.

Example:

```
MOTION SUSPEND
MDV   10,2        ;placed in stack
MDV   20,2        ;placed in stack
MDV   2,0         ;placed in stack
MOVED   3,C       ;must use ",C "modifier. Otherwise program will hang.
MOTION RESUME
```

Caution should be taken when using MOVED, MOVEP and MOVE statements. If any of the MOVE instructions are written without the “C” modifier, the program will hang or lock up. The “MOTION SUSPEND” command effectively halts all execution of motion. In the example, as the program executes the “MDV” and “MOVED” statements, those move profiles are loaded into the motion stack. If the final “MOVED” is missing the “C” modifier then the User Program will wait until that move profile is complete before continuing on. Because motion has been suspended, the move will never be complete and the program will hang on this instruction.

2.11.10 Conditional Moves (MOVE WHILE/UNTIL)

The statements “MOVE UNTIL <expression>” and “MOVE WHILE <expression>” will both start their motion profiles based on their acceleration and max velocity profile settings. The “MOVE UNTIL <expression> statement will continue the move until the <expression> becomes true. The “MOVE WHILE <expression>” will also continue its move while it’s <expression> is true. Expression can be any valid arithmetic or logical expressions or their combination.

Examples:

```
MOVE WHILE APOS<20          ;Move while the position is less then 20, then
                            ;stop with current deceleration rate.
MOVE UNTIL APOS>V1          ;Move positive until the position is greater than
                            ;the value in variable V1
MOVE BACK UNTIL APOS<V1     ;Move negative until the position is less than the
                            ;value in variable V1
MOVE WHILE  IN_A1           ;Move positive while input A1 is activated.
MOVE WHILE !IN_A1           ;Move positive while input A1 is not activated.
                            ;The exclamation mark (!) in front of IN_A1 inverts
                            ;(or negates) the value of IN_A1.
```

This last example is a convenient way to find a sensor or switch.

2.11.11 Motion Queue and Statement Execution while in Motion

By default when the program executes a MOVE, MOVED or MOVEP statement, it waits until the motion is complete before going on to the next statement. This effectively will suspend the program until the requested motion is complete. Note that “EVENTS” are not suspended however and continue executing in parallel with the User Program. The Continue “C” argument is very useful when it is necessary to trigger an action (handle I/O) while the motor is in motion. Below is an example of the Continue “C” argument.

```
;This program monitors I/O in parallel with motion:
START:
      MOVED 100,C             ;start moving max 100 revs
WHILE F_MCOMPLETE=0           ;while moving
      IF IN_A2 == 1           ;if sensor detected
            OUT1=1            ;turn ON output
            WAIT TIME 500     ;500 mS
            OUT1=0            ;turn output OFF
            WAIT TIME 500     ;wait 500 ms
      ENDIF
ENDWHILE
MOVED -100                    ;Return back
WAIT TIME 1000                ;wait time
GOTO START                    ;and start all over
END
```

This program starts a motion of 100 revolutions. While the motor is in motion, input A2 is monitored. If Input A2 is made during the move, then output 1 is turned on for 500ms and then turned off. The program will continue to loop in the WHILE statement, monitoring input A2, until the move is completed. If input 2 remains ON, or made, during the move, then Output 1 will continue to toggle On and Off every 500ms until the move is complete. If input A2 is only made while the motion passes by a sensor wired to the input, then output 1 will stay on for 500ms only. By adding the “Continue” argument “C” to the MOVE statement, the program is able to monitor the input while executing the motion profile. Without this modifier the program would be suspended until all motion is complete. After the motor has traveled the full distance it then returns back to its initial position and the process repeats.

Figure 21 illustrates the structure and operation of the Motion Queue. All moves are loaded into the Motion Queue before they are executed. If the move is a standard move, “MOVEP 10” or “MOVED 10”, then the move will be loaded into the queue and the execution of the User Program will be suspended until the move is completed. If the move has the continue argument, e.g. “MOVEP 10,C” or “MOVED 10,C”, or if it is an “MDV” move, then the moves will be loaded into Motion Queue and executed simultaneously with the User Program.

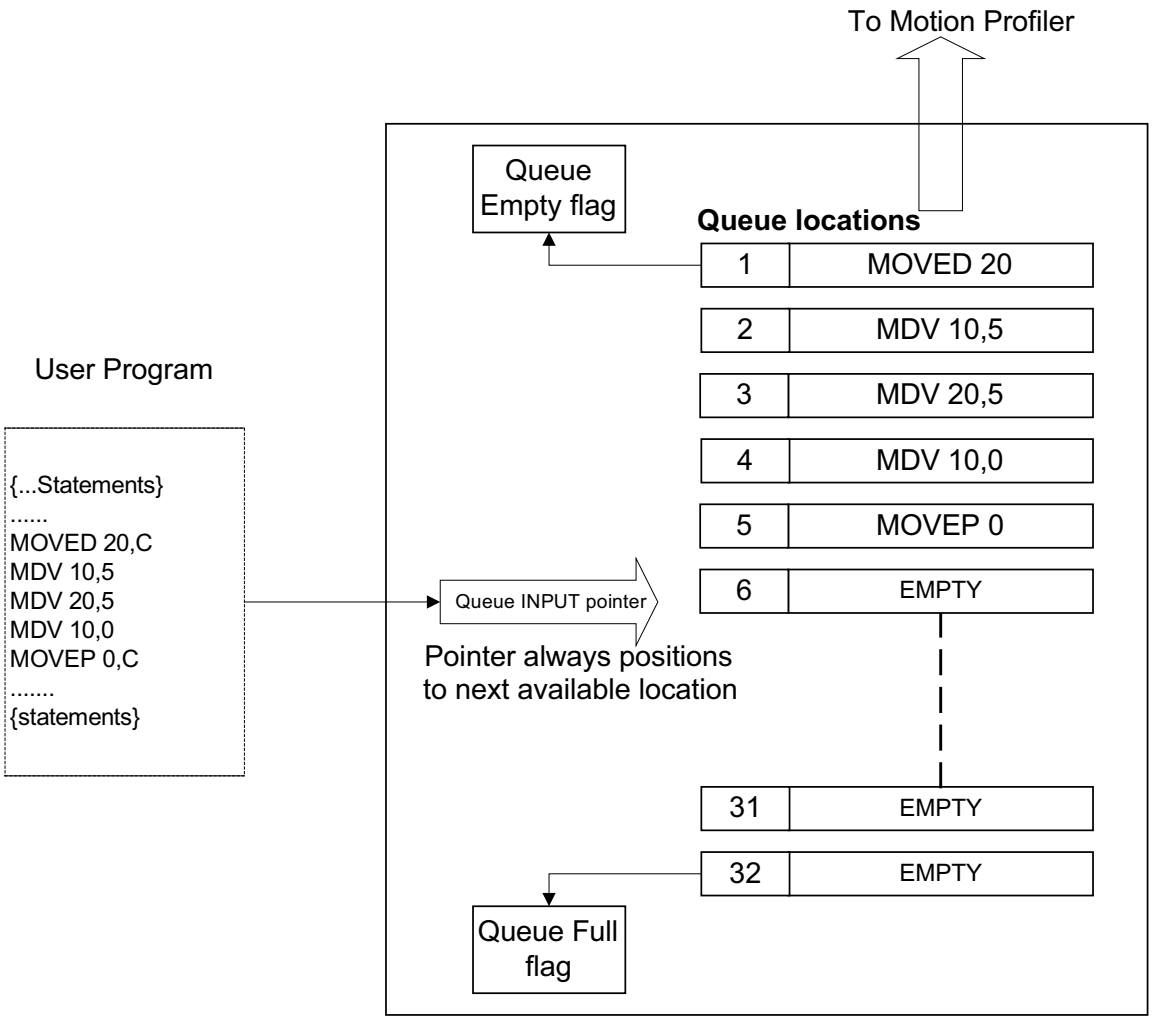


Figure 21: Motion Queue

The Motion Queue can hold a maximum of 32 motion statements. The System Status Register contains bit values that indicate the state of the Motion Queue. Additionally, system flags (representing individual bits of the status register) are available for ease of programming. If the possibility of motion queue overflow exists, the programmer should check the Motion Queue full flag before executing any MOVE statements, especially in programs where MOVE statements are executed in a continuous cycle. Attempts to execute a motion statement while the Motion Queue is full will result in fault #23. MDV statements don't have the "C" option because the program is never suspended by these statements. If the last MDV statement in the Queue doesn't specify a return to 0 velocity then a Stack Underflow (Fault #24) will occur.

The "MOTION SUSPEND" and "MOTION RESUME" statements can be utilized to help manage the User Program and the Motion Queue. If the motion profiles loaded into the queue are not managed correctly, the Motion Queue can become overloaded which will cause the drive to fault.

2.12 System Status Register (DSTATUS register)

System Status Register, (DSTATUS), is a Read Only register. Its bits indicate the various states of the PositionServo's subsystems. Some of the bits are available as System Flag Variables and previously summarized in Table 12.

Table 16: DSTATUS Register

Bit in register	Description
0	Set when drive enabled
1	Set if DSP subsystem at any fault
2	Set if drive has a valid program
3	Set if byte-code or system or DSP at any fault
4	Set if drive has a valid source code
5	Set if motion completed and target position is within specified limits
6	Set when scope is triggered and data collected
7	Set if motion stack is full
8	Set if motion stack is empty
9	Set if byte-code halted
10	Set if byte-code is running
11	Set if byte-code is set to run in step mode
12	Set if byte-code has reached the end of program
13	Set if current limit is reached
14	Set if byte-code at fault
15	Set if no valid motor selected
16	Set if byte-code at arithmetic fault
17	Set if byte-code at user fault
18	Set if DSP initialization completed
19	Set if registration has been triggered
20	Set if registration variable was updated from DSP after last trigger
21	Set if motion module at fault
22	Set if motion suspended
23	Set if program requested to suspend motion
24	Set if system waits completion of motion
25	Set if motion command completed and motion Queue is empty
26	Set if byte-code task requested reset
27	If set interface control is disabled. This flag is set/clear by ICONTROL ON/OFF statement.
28	Set if positive limit switch active
29	Set if negative limit switch active
30	Events disabled. All events disabled when this flag is set. After executing EVENTS ON all events previously enabled by EVENT EventName ON statements become enabled again

PositionServo variable #83 provides Extended Status Bits, the encoding of which is listed in Table 17.

Table 17: Extended Status Bits (Variable #83 EXSTATUS)

Bit #	Function	Comment
0	Reserved	
1	Velocity in specified window	Velocity in limits as per parameter #59: VAR_VLIMIT_SPEEDWND
2-4	Reserved	
5	Velocity at 0 (zero)	Velocity 0: Zero defined by parameter #58: VAR_VLIMIT_ZEROSPEED
6,7	Reserved	
8	Bus voltage below under-voltage limit	Utilized to indicate drive is operating from +24V keep alive and a valid DC bus voltage level is not present.
9,10	Reserved	
11	Regen circuit is on	Drive regeneration circuit is active. Drive will be dissipating power through the braking resistor (if fitted).
12-20	Reserved	
21	Set if homing operation in progress	Drive executing Pre-defined homing function (see section 2.15).
22	Set if system homed	Drive completed Pre-defined homing function (see section 2.15).
23	If set then last fault will remain on the display until re-enabled.	User can set this bit to retain fault code on the display until re-enabled. It is useful if there is a fault handler routine. When the fault handler is exited, the fault number on the display will be replaced by current status (usually DiS if bit #23 is not set). Setting bit #23 retains diagnostics on the display.
24	Set if EIP IO exclusive owner connection is established. Cleared if closed.	Checks if drive is controlled by EthernetIP master. Use bit #24 and bit #25 to process “loss of connection” condition (if needed) in the user’s program
25	Set if EIP IO exclusive owner connection times out. Cleared if exc. owner conn exsists.	Checks if connection with Ethernet/IP master is lost. Use bit #24 and bit #25 to process “loss of connection” condition (if needed) in the user’s program
26-31	Reserved	

2.13 Fault Codes (DFAULTS register)

Whenever a fault occurs in the drive, a record of that fault is recorded in the Fault Register (DFAULTS). In addition, specific flags in the System Status Register will be set helping to indicate what class of fault the current fault belongs to. Table 18 summarizes the fault codes. Codes from 1 to 16 are used for DSP subsystem errors. Codes above that range are generated by various subsystems of the PositionServo.

Table 18: DFAULTS Register

Fault ID	Associated flags in status register	Description
1	1, 3	Over voltage
2	1, 3	Invalid Hall sensors code
3	1, 3	Over current
4	1, 3	Over temperature
5	1, 3	The drive is disabled by the ISO 13849-1 Safety Function
6	1, 3	Over speed. (Over speed limit set by motor capability in motor file)
7	1, 3	Position error excess.
8	1, 3	Attempt to enable while motor data array invalid or motor was not selected.
9	1,3	Motor over temperature switch activated
10	1,3	Sub processor error
11-13	-	Reserved
14	1,3	Under voltage (hardware revision 1)
15	1,3	Hardware current trip protection
16	-	Reserved
18	16	Division by zero
19	16	Arithmetic overflow
20	3	Subroutine stack overflow. Exceeded 32 levels subroutines stack depth.

Fault ID	Associated flags in status register	Description
21	3	Subroutine stack underflow. Executing RETURN statement without preceding call to subroutine.
22	3	Variable evaluation stack overflow. Expression too complicated for compiler to process.
23	21	Motion Queue overflow. 32 levels depth exceeded
24	21	Motion Queue underflow. Last queued MDV statement has non 0 target velocity
25	3	Unknown opcode. Byte code interpreter error; Occurs when program is missing END statement
26	3	Unknown byte code. Byte code interpreter error; Occurs when RETURN statement missing from subroutine; or when EPM data is corrupted at run-time
27	21	Drive disabled. Attempt to execute motion while drive is disabled.
28	16, 21	Accel/Decel too high. Motion statement parameters calculate Accel /Decel value above system capability
29	16, 21	Accel/Decel too low. Motion statement parameters calculate Accel/Decel value below system capability.
30	16, 21	Velocity too high. Motion statement parameters calculate a velocity above the system capability.
31	16, 21	Velocity too low. Motion statement parameters calculate a velocity below the system capability.
32	3,21	Positive limit switch engaged
33	3,21	Negative limit switch engaged
34	3,21	Attempt at positive motion with engaged positive limit switch
35	3,21	Attempt at negative motion with engaged negative limit switch
36	3	Hardware disable (enable input not active when attempting to enable drive from program or interface)
37	3	Under voltage (hardware revision 2)
38	3	EPM loss
39	3,21	Positive soft limit reached
40	3,21	Negative soft limit reached
41	3	Attempt to use variable with unknown ID from user program
45	1,3	Second encoder position error excess
49	1,3	Illegal manipulation of APOS variable

2.14 Limitations and Restrictions

Communication Interfaces Usage Restrictions

Simultaneous connection to the RS485 port is allowed for retransmitting (conversion) between interfaces.

WARNING!
Usage of the RS485 simultaneously with Ethernet may lead to unpredictable behavior since the drive will attempt to perform commands from both interfaces concurrently.

Motion Parameters Limitation

Due to a finite precision in the calculations there are some restrictions for acceleration/deceleration and max velocity for a move. If the programmer receives arithmetic faults during his program's execution, it is likely due to these limitations. Min/Max values are expressed in counts or counts/sample, where the sample is a position loop sample interval (512μsec).

Table 19: Motion Parameter Limits

Parameter	MIN	MAX	Units
Accel / Decel	65/(2^32)	512	counts/sample^2
MaxV (maximum velocity)	0	2048	counts/sample
Max move distance	0	+/- 2^31	counts

Stacks and Queues Depth Limitations

Table 20: Stack Depth Limit

Stack/Queue	Motion Queue	Subroutines Stack	Number of Events
Depth	32	32	32

2.15 Homing

2.15.1 What is Homing?

Homing is the method by which a drive seeks the home position (also called the datum, reference point, or zero point). There are various methods of achieving this using:

- limit switches at the ends of travel, or
- a dedicated home switch, or
- an Index Pulse or zero reference from the motor feedback device, or
- a combination of the above.

Predefined (firmware based) homing functionality is available on PositionServo drives with firmware 3.03 or later. In addition custom homing functionality can be created by the programmer within the user program by utilizing the programming command set available.

Examples of custom homing routine creation as well as user program code to replicate each of the predefined homing routines is available from technical support.

2.15.2 The Homing Function

The homing function provides a set of trajectory parameters to the position loop, as shown in Figure 22. They are calculated based on user supplied variable values as listed below:

VAR_HOME_OFFSET
VAR_HOME_METHOD
VAR_HOME_SWITCH_INPUT
VAR_HOME_FAST_VEL
VAR_HOME_SLOW_VEL
VAR_HOME_ACCEL
VAR_START_HOMING

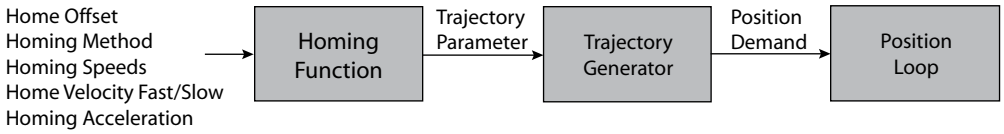


Figure: 22: Homing Function

Homing Function Monitoring:

The extended drive status variable (#83 EXSTATUS variable) contains bit values for monitoring the homing function over a communications interface.

Bit 21 of EXSTATUS indicates homing procedure in progress and is set to logic 1 while homing is being executed.
Bit 22 of EXSTATUS indicates homing complete. It is set to 1 upon the successful completion of the homing routine.

2.15.3 Home Offset

The home offset is the difference between the zero position for the application and the machine home position (found during homing). During homing the home position is found and once the homing is completed the zero position is offset from the home position by adding the home offset to the home position. All subsequent absolute moves are made relative to this new zero position. This is illustrated in Figure 23. Offset can either be set in User Units (UU) by writing to variable #240, or in encoder counts by writing to variable #241. Setting a value for either variable #240 or #241 will result in a value automatically being calculated and stored in the respective variable.

VAR_HOME_OFFSET (#240)
VAR_HOME_OFFSET_PULSES (#241)

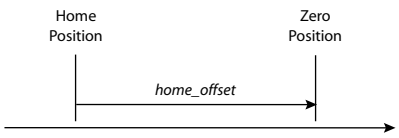


Figure 23: Home Offset

2.15.4 Homing Velocity

There are two homing velocities: fast and slow. These velocity variables are used to find the home switch and to find the index pulse. How the two velocities are implemented within the homing routines depends on the homing routine selected. Refer to section 2.5.9.

VAR_HOME_FAST_VEL (#242)
VAR_HOME_SLOW_VEL (#243)

2.15.5 Homing Acceleration

Homing acceleration establishes the velocity ramp rate to be used for all accelerations and decelerations within the standard homing modes. Note that in the pre-defined homing methods, it is not possible to program a separate deceleration rate.

VAR_HOME_ACCEL (#239)

2.15.6 Homing Switch

The homing switch variable enables the user to select the PositionServo input used for the Home Switch connection. The Homing Switch Input Assignment range is 0 - 11. Inputs A1-A4 are assigned 0 to 3, respectively; inputs B1-B4 are assigned 4 to 7, respectively; and inputs C1-C4 are assigned 8 to 11, respectively.

VAR_HOME_SWITCH_INPUT (#246)

WARNING!

- Setting inputs A1 and A2 as the home switch, even in methods that do NOT use limit switches can cause the drive to behave in an unexpected manner.
- Input A3 is a dedicated hardware enable input and should never be assigned as the homing switch input.
- Input C3 can be used as the homing switch input only in methods that do not home to an index pulse.

2.15.7 Homing Start

There are two methods of starting pre-defined homing operation, the ‘HOME’ command and the Var_Start_Homing variable. When Homing is initiated from the user program the ‘HOME’ command should always be used. The HOME command is a blocking instruction that prevents further execution of the Main Program until homing operation is completed. Any events that are enabled whilst homing is carried out will continue to process.

WARNING!

If using firmware prior to 4.50 then execution of homing functionality does not prevent simultaneous execution of subsequent programming statements and it is required to immediately follow the HOME command with the following code line:

WAIT UNTIL VAR_EXSTATUS & 0x400000 == 0x400000.

Doing this ensures no further lines of code will be executed until homing is complete.

The home start variable (Var_Start_Homing) is used to initiate pre-defined homing functionality from a host interface. It should not be used if the drive contains or is executing a user program. Var_Start_Homing range is: 0 or 1. When set to 0, no action occurs. When set to 1, the homing operation is started.

VAR_START_HOMING (#245)

2.15.8 Homing Method

VAR_HOME_METHOD (#244)

The Home Method variable establishes the method that will be used for homing. All supported methods are summarized in Table 21 and described in sections 2.15.9.1 through 2.15.9.25. These homing methods define the required operation of the drive in location of the home position. The zero position is always the home position adjusted by the homing offset.

Table 21: Homing Methods

Method	Home Position
0	No operation/reserved. An attempt to execute 0 will result in execution of method 1.
1	Location of first index pulse is on the positive side of the negative limit switch.
2	Location of first index pulse is on the negative side of the positive limit switch.
3	Location of first index pulse is on the negative side of a positive home switch.[1]
4	Location of first index pulse is on the positive side of a positive home switch.[1]
5	Location of first index pulse is on the positive side of a negative home switch.[2]
6	Location of first index pulse is on the negative side of a negative home switch.[2]
7	Location of first index pulse is on the negative side of the negative edge of an intermittent home switch.[3]
8	Location of first index pulse is on the positive side of the negative edge of an intermittent home switch.[3]
9	Location of first index pulse is on the negative side of the positive edge of an intermittent home switch.[3]
10	Location of first index pulse is on the positive side of the positive edge of an intermittent home switch.[3]
11	Location of first index pulse is on the positive side of the positive edge of an intermittent home switch.[3]
12	Location of first index pulse is on the negative side of the positive edge of an intermittent home switch.[3]
13	Location of first index pulse is on the positive side of the negative edge of an intermittent home switch.[3]
14	Location of first index pulse is on the negative side of the negative edge of an intermittent home switch.[3]
15	Reserved for future use.
16	Reserved for future use
17	The edge of a negative limit switch.
18	The edge of a positive limit switch.
19	The edge of a positive home switch.
20	Reserved for future use.
21	The edge of a negative home switch.
22	Reserved for future use.
23	Positive edge of an intermittent home switch.
24	Reserved for future use.
25	The negative edge of an intermittent home switch.
26	Reserved for future use.
27	Negative edge of an intermittent home switch.
28	Reserved for future use.
29	The positive edge of an intermittent home switch.
30	Reserved for future use.
31	Reserved for future use.
32	Reserved for future use.
33	The first index pulse on the negative side of the current position.
34	The first index pulse on the positive side of the current position.
35	Current position becomes home position. Home offset is also active and will be added to current position to set the zero position.

1 - A positive home switch is one that goes active at a set position, and remains active for all positions greater than the set position.

2 - A negative home switch is one that goes active at a set position, and remains active for all positions less than the set position.

3 - An intermittent home switch is one that is only active for a limited range of travel.

2.15.9 Homing Methods

There are several types of homing methods but each method establishes the:

- Homing signal (positive limit switch, negative limit switch, home switch ,or index pulse)
- Direction of actuation and, where appropriate, the direction of the index pulse.

The homing method descriptions and diagrams in this manual are based on those in the CANopen Profile for Drives and Motion Control (DSP 402). As illustrated in Figure 24, each homing method diagram shows the motor in the starting position on a mechanical stage. The arrow line indicates direction of motion and the circled number indicates the homing method (the mode selected by the Homing Method variable).

The location of the circled method number indicates the home position reached with that method. The text designators (A, B) indicate the logical transition required for the homing function to complete it's current phase of motion. Dashed lines overlay these transitions and reference them to the relevant transitions of limit switches, homing sensors, or index pulses.

Definitions

Positive home switch: goes active at a set position, and remains active for all positions greater than the set position.

Negative home switch: goes active at a set position, and remains active for all positions less than the set position.

Intermittent home switch: is one that is only active for a limited range of travel.

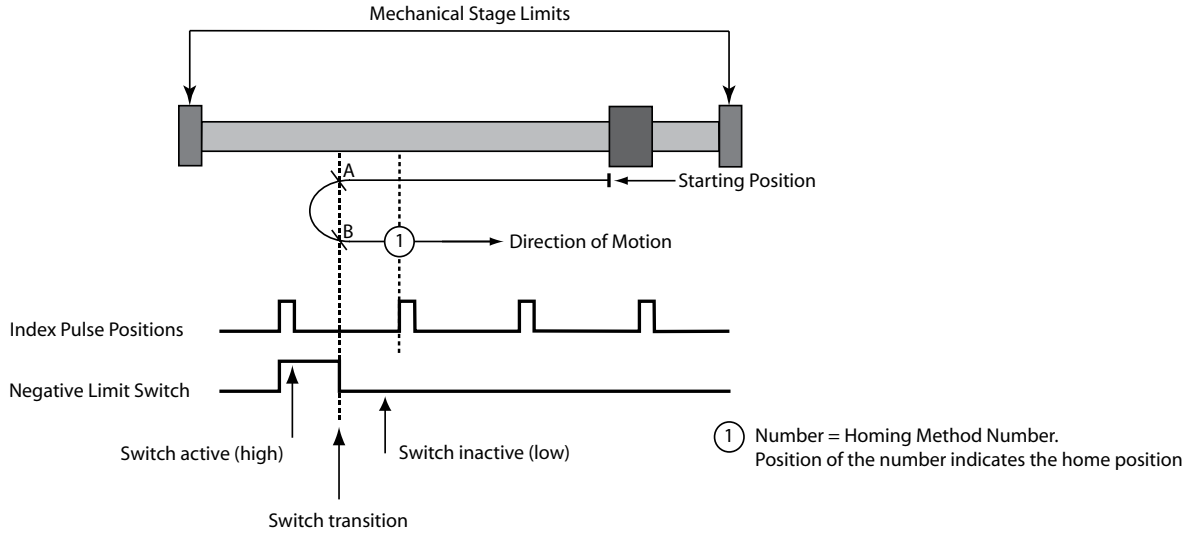


Figure 24: Homing Terms

NOTE

In the homing method descriptions, negative motion is leftward and positive motion is rightward

BLUE lines indicate fast velocity moves

GREEN lines indicate slow velocity moves

RED lines indicate slow velocity/100 moves

2.15.9.1 Homing Method 1: Homing on the Negative Limit Switch & Index Pulse

Using this method, the initial direction of movement is negative if the negative limit switch is inactive (here shown as low). The home position is at the first index pulse to the positive side of the position where the negative limit switch becomes active.

Axis will accelerate to ***fast*** homing velocity in the negative direction and continue until Negative Limit Switch (A1) is activated (rising edge) shown at position A. Axis then decelerates to zero velocity. If the negative limit switch is already active when the homing routine commences then this initial move is not executed. Axis will then accelerate to ***slow*** homing velocity in the positive direction. Motion will continue until first the falling edge of the negative limit switch is detected (position B) and then the rising edge of the first index pulse (position 1) is detected.

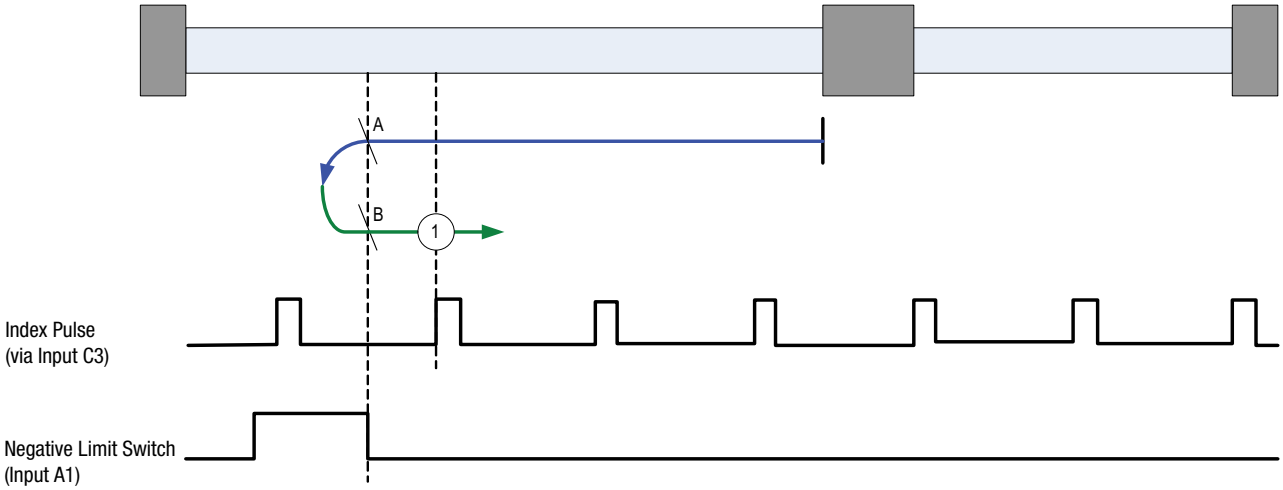


Figure 25: Homing Method 1

2.15.9.2 Homing Method 2: Homing on the Positive Limit Switch & Index Pulse

Using this method the initial direction of movement is positive if the positive limit switch is inactive (here shown as low). The position of home is at the first index pulse to the negative side of the position where the positive limit switch becomes active.

Axis will accelerate to ***fast*** homing velocity in the positive direction and continue until Positive Limit Switch (A2) is activated (rising edge) shown at position A. Axis then decelerates to zero velocity. If the positive limit switch is already active when the homing routine commences then this initial move is not executed. Axis will then accelerate to ***slow*** homing velocity in the negative direction. Motion will continue until first the falling edge of the positive limit switch is detected (position B) and then the rising edge of the first index pulse (position 2) is detected.

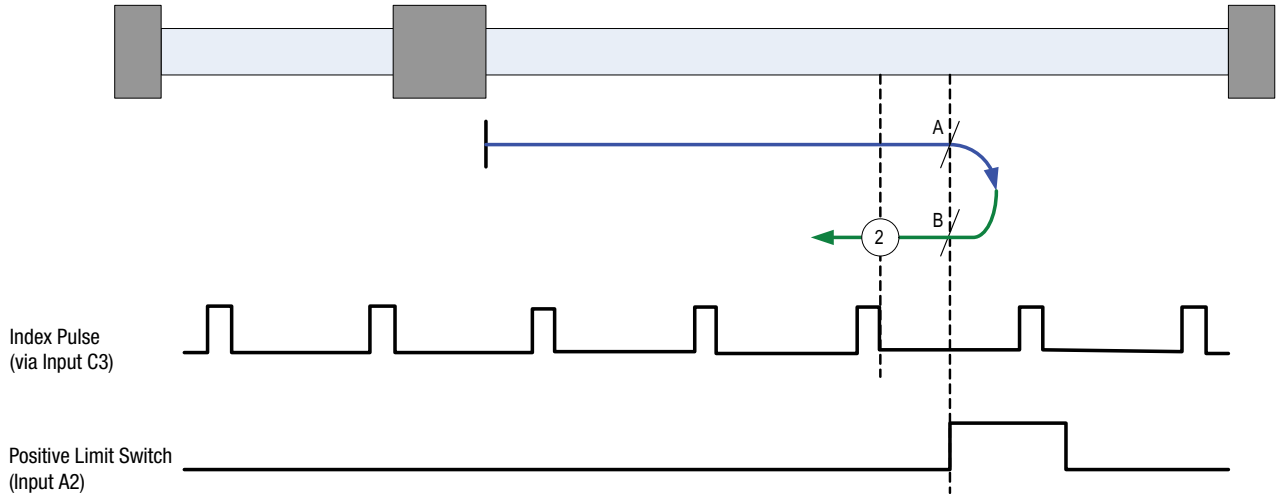


Figure 26: Homing Method 2

2.15.9.3 Homing Method 3: Homing on the Positive Home Switch & Index Pulse

Using this method the initial direction of movement is positive (if the homing switch is inactive). The home position is the first index pulse to the negative side of the position where the homing switch becomes active.

Axis will accelerate to ***fast*** homing velocity in the positive direction and continue until Homing Switch (selectable via Var_Home_Switch_Input Variable) is activated (rising edge) shown at position A. Axis then decelerates to zero velocity. If the homing switch is already active when the homing routine commences then this initial move is not executed. Axis will then accelerate to ***fast*** homing velocity in negative direction. Motion will continue until first the falling edge of the Homing switch is detected (position B) and then the rising edge of the first index pulse (position 3) is detected.

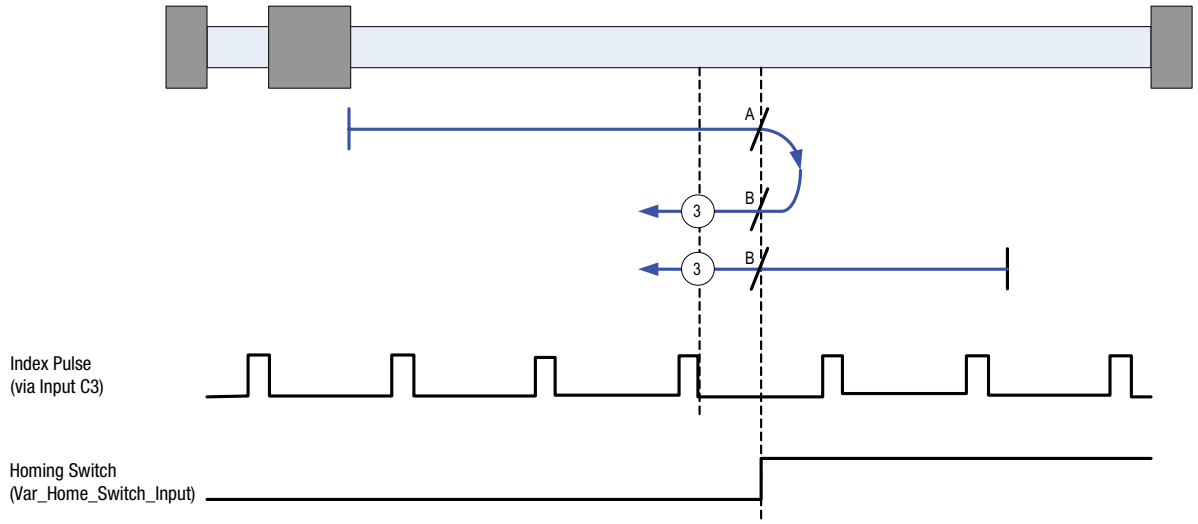


Figure 27: Homing Method 3

2.15.9.4 Homing Method 4: Homing on the Positive Home Switch & Index Pulse

Using this method the initial direction of movement is negative (if the homing switch is active). The home position is the first index pulse to the positive side of the position where the homing switch becomes inactive.

Axis will accelerate to ***fast*** homing velocity in the negative direction and continue until Homing Switch (selectable via Var_Home_Switch_Input Variable) is deactivated (falling edge) shown at position A. Axis then decelerates to zero velocity. If the homing switch is already inactive when the homing routine commences then this initial move is not executed. Axis will then accelerate to ***fast*** homing velocity in positive direction. Motion will continue until first the rising edge of the Homing switch is detected (position B) and then the rising edge of the first index pulse (position 4) is detected.

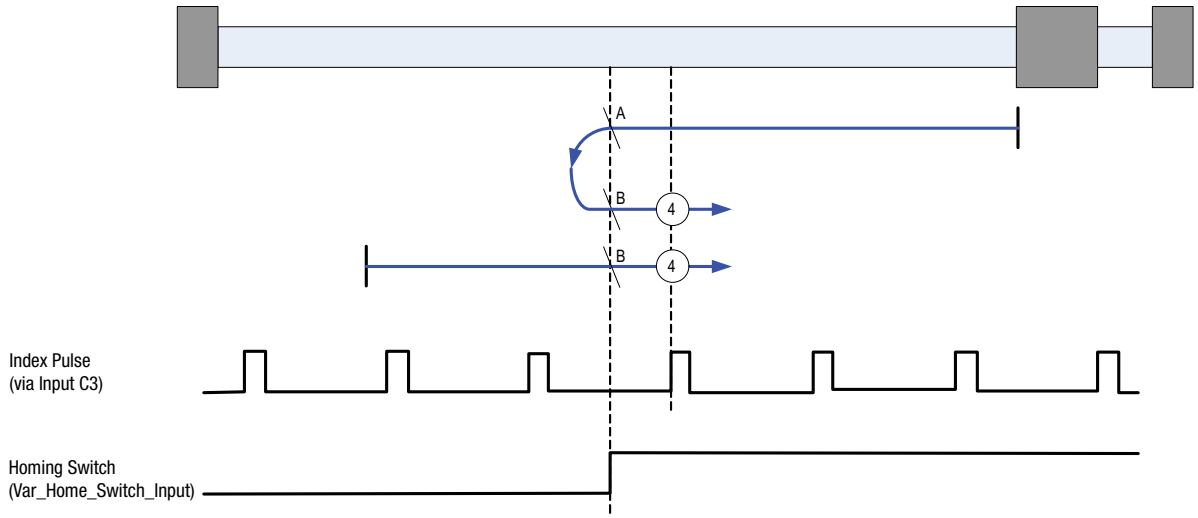


Figure 28: Homing Method 4

2.15.9.5 Homing Method 5: Homing on the Negative Home Switch & Index Pulse

Using this method the initial direction of movement is negative (if the homing switch is inactive). The home position is the first index pulse to the positive side of the position where the homing switch becomes active.

Axis will accelerate to ***fast*** homing velocity in the negative direction and continue until Homing Switch (selectable via Var_Home_Switch_Input Variable) is activated (rising edge) shown at position A. Axis then decelerates to zero velocity. If the homing switch is already active when the homing routine commences then this initial move is not executed. Axis will then accelerate to ***fast*** homing velocity in positive direction. Motion will continue until first the falling edge of the Homing switch is detected (position B) and then the rising edge of the first index pulse (position 5) is detected.

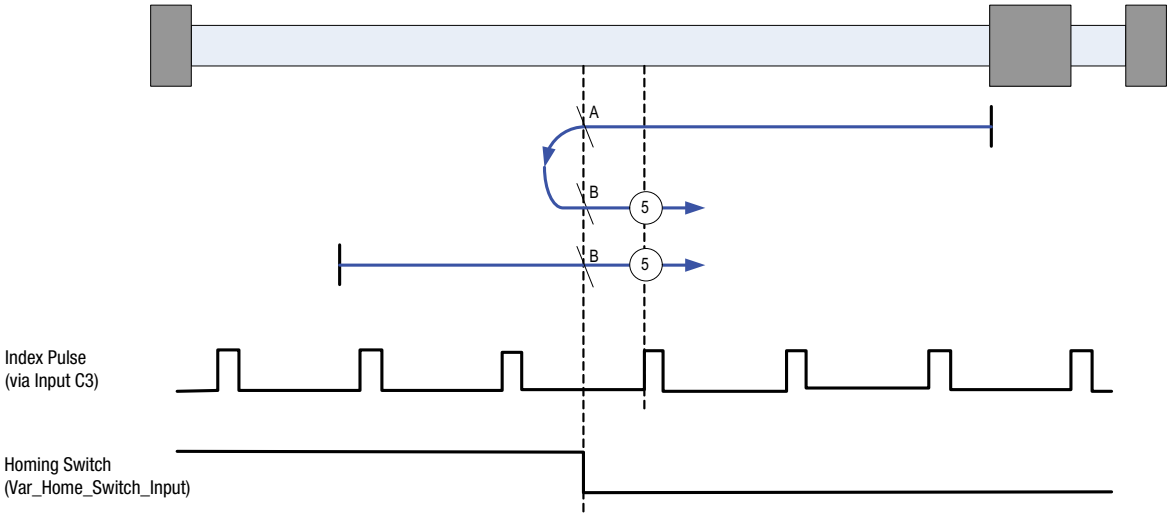


Figure 29: Homing Method 5

2.15.9.6 Homing Method 6: Homing on the Negative Home Switch & Index Pulse

Using this method the initial direction of movement is positive (if the homing switch is active). The home position is the first index pulse to the negative side of the position where the homing switch becomes inactive.

Axis will accelerate to ***fast*** homing velocity in the positive direction and continue until Homing Switch (selectable via Var_Home_Switch_Input Variable) is deactivated (falling edge) shown at position A. Axis then decelerates to zero velocity. If the homing switch is already inactive when the homing routine commences then this initial move is not executed. Axis will then accelerate to ***fast*** homing velocity in negative direction. Motion will continue until first the rising edge of the Homing switch is detected (position B) and then the rising edge of the first index pulse (position 6) is detected.

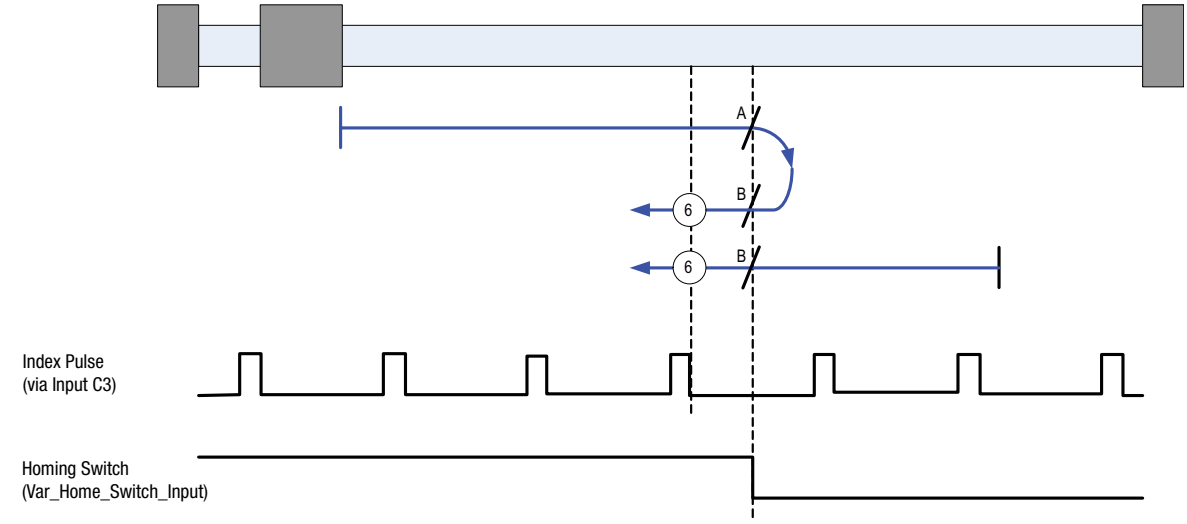


Figure 30: Homing Method 6

2.15.9.7 Homing Method 7: Homing on the Home Switch & Index Pulse

Using this method the initial direction of movement is positive (if the homing switch is inactive). The home position is the first index pulse to the negative side of the position where the homing switch becomes active.

Axis will accelerate to ***fast*** homing velocity in the positive direction and continue until Homing Switch (selectable via Var_Home_Switch_Input Variable) is activated (rising edge) shown at position A. Axis then decelerates to zero velocity.

If the homing switch is already active when the homing routine commences then this initial move is not executed.

Axis will then accelerate to ***fast*** homing velocity in the negative direction. Motion will continue until first the falling edge of the Homing switch is detected (position B) and then the rising edge of the first index pulse (position 7) is detected.

NOTE: if the axis is on the wrong side of the homing switch when homing is started then the axis will move positive until it contacts the positive limit switch (A2). Upon activating the positive limit switch the axis will change direction (negative) following the procedure as detailed above, but moving negative instead of positive and without stopping on detection of the homing switch rising edge.

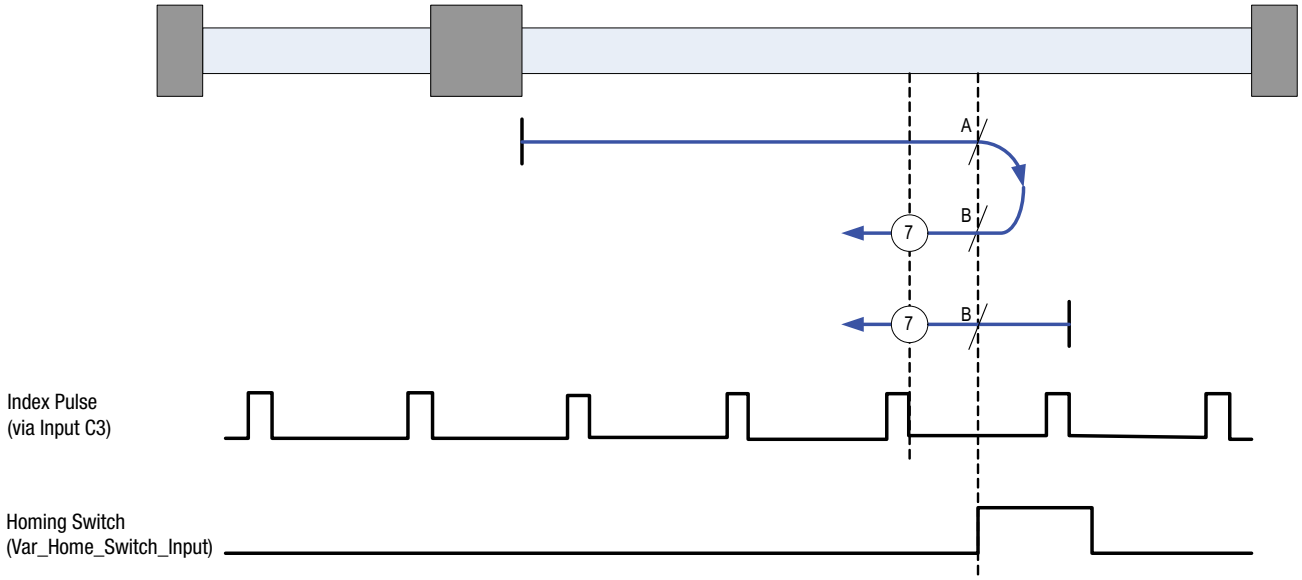


Figure 31: Homing Method 7

2.15.9.8 Homing Method 8: Homing on the Home Switch & Index Pulse

Using this method the initial direction of movement is negative (if the homing switch is active). The home position is the first index pulse to the positive side of the position where the homing switch becomes inactive.

Axis will accelerate to ***fast*** homing velocity in the negative direction and continue until Homing Switch (selectable via Var_Home_Switch_Input Variable) is deactivated (falling edge) shown at position A. Axis then decelerates to zero velocity.

If the homing switch is already inactive when the homing routine commences then this initial move is not executed.

Axis will then accelerate to ***fast*** homing velocity in the positive direction. Motion will continue until first the rising edge of the Homing switch is detected (position B) and then the rising edge of the first index pulse (position 8) is detected.

NOTE: if the axis is on the wrong side of the homing switch when homing is started then the axis will move positive until it contacts the positive limit switch (A2). Upon activating the positive limit switch the axis will change direction (negative) following the procedure as detailed above.

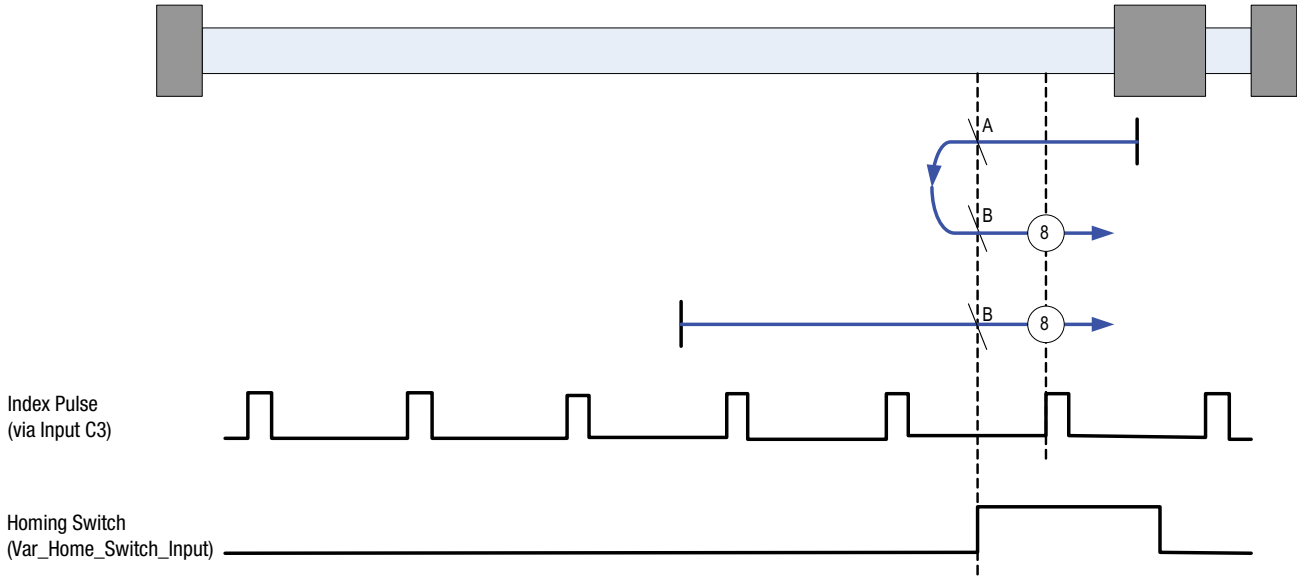


Figure 32: Homing Method 8

2.15.9.9 Homing Method 9: Homing on the Home Switch & Index Pulse

Using this method the initial direction of movement is positive. The home position is the first index pulse to the negative side of the position where the homing switch becomes inactive on its negative edge.

Axis will accelerate to ***fast*** homing velocity in the positive direction and continue until Homing Switch (selectable via Var_Home_Switch_Input Variable) is deactivated (falling edge) shown at position A. Axis then decelerates to zero velocity.

If the homing switch is already active when the homing routine commences then this does not effect this mode of homing as the procedure is searching for falling edge of homing switch in both cases.

Axis will then accelerate to ***fast*** homing velocity in the negative direction. Motion will continue until first the rising edge of the Homing switch is detected (position B) and then the rising edge of the first index pulse (position 9) is detected.

NOTE: if the axis is on the wrong side of the homing switch when homing is started then the axis will move positive until it contacts the positive limit switch (A2). Upon activating the positive limit switch the axis will change direction (negative) following the procedure as detailed above but ignoring the initial move in the positive direction.

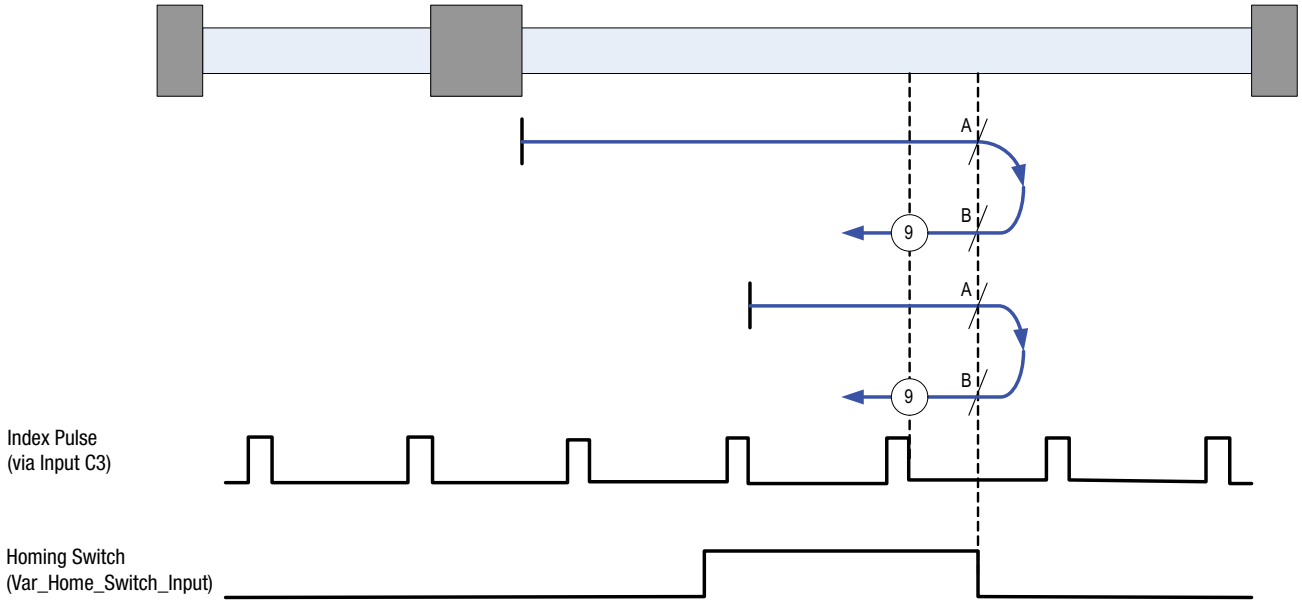


Figure 33: Homing Method 9

2.15.9.10 Homing Method 10: Homing on the Home Switch & Index Pulse

Using this method the initial direction of movement is positive. The home position is the first index pulse to the positive side of the position where the homing switch becomes inactive.

Axis will accelerate to ***fast*** homing velocity in the positive direction and continue until Homing Switch (selectable via Var_Home_Switch_Input Variable) is deactivated (falling edge) shown at position A.

If the homing switch is already active when the homing routine commences then this does not effect this mode of homing as the procedure is searching for falling edge of homing switch in both cases.

Axis will continue running at ***fast*** homing velocity in the positive direction until the rising edge of the first index pulse (position 10) is detected.

NOTE: if the axis is on the wrong side of the homing switch when homing is started then the axis will move positive until it contacts the positive limit switch (A2). Upon activating the positive limit switch the axis will change direction (negative) continuing motion until it sees the rising edge of the homing switch. The axis will then stop and follow the procedure as detailed above.

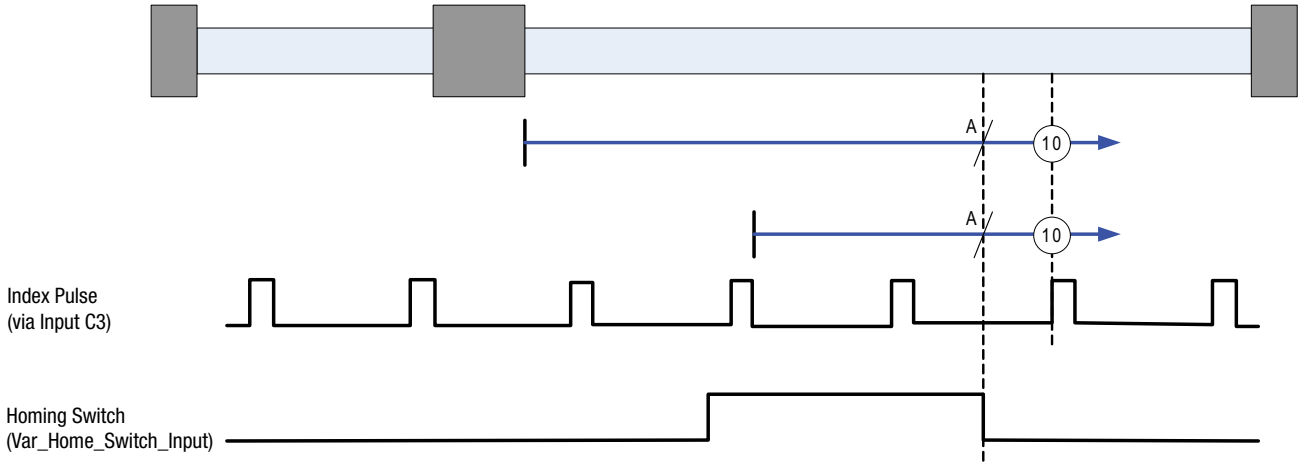


Figure 34: Homing Method 10

2.15.9.11 Homing Method 11: Homing on the Home Switch & Index Pulse

Using this method the initial direction of movement is negative (if the homing switch is inactive). The home position is the first index pulse to the positive side of the position where the homing switch becomes active.

Axis will accelerate to ***fast*** homing velocity in the negative direction and continue until Homing Switch (selectable via Var_Home_Switch_Input Variable) is activated (rising edge) shown at position A. Axis then decelerates to zero velocity.

If the homing switch is already active when the homing routine commences then this initial move is not executed.

Axis will then accelerate to ***fast*** homing velocity in the positive direction. Motion will continue until first the falling edge of the Homing switch is detected (position B) and then the rising edge of the first index pulse (position 11) is detected.

NOTE: if the axis is on the wrong side of the homing switch when homing is started then the axis will move negative until it contacts the negative limit switch (A1). Upon activating the negative limit switch the axis will change direction (positive) following the procedure as detailed above, but moving positive instead of negative and without stopping on detection of the homing switch rising edge.

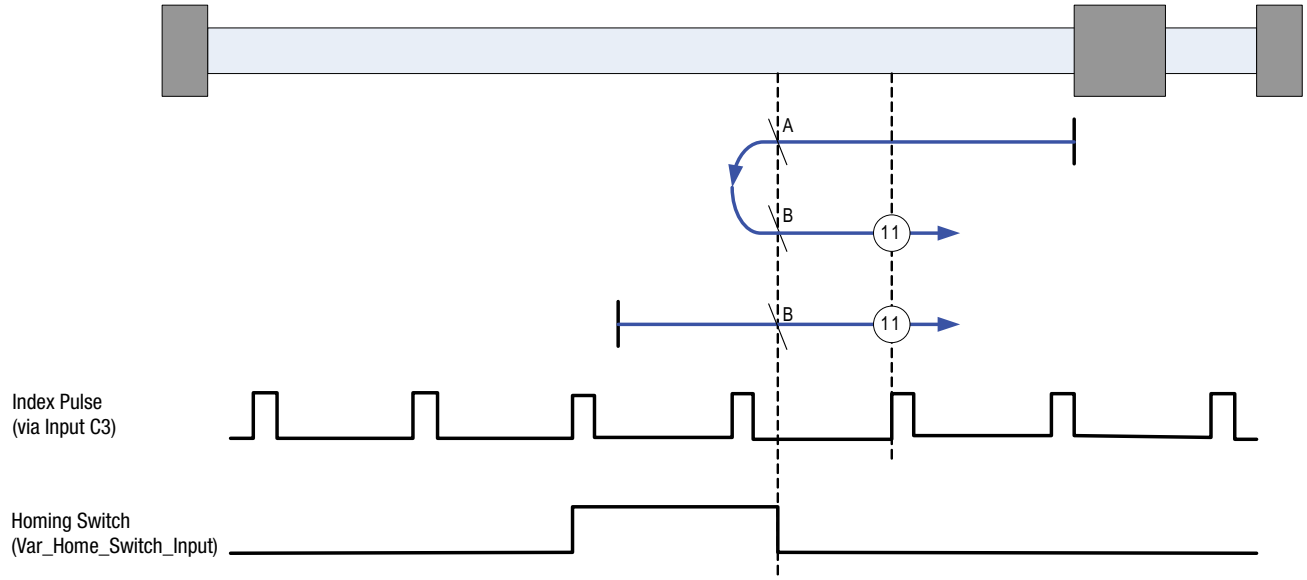


Figure 35: Homing Method 11

2.15.9.12 Homing Method 12: Homing on the Home Switch & Index Pulse

Using this method the initial direction of movement is positive (if the homing switch is active). The home position is the first index pulse to the negative side of the position where the homing switch becomes inactive.

Axis will accelerate to ***fast*** homing velocity in the positive direction and continue until Homing Switch (selectable via Var_Home_Switch_Input Variable) is deactivated (falling edge) shown at position A. Axis then decelerates to zero velocity.

If the homing switch is already inactive when the homing routine commences then this initial move is not executed.

Axis will then accelerate to ***fast*** homing velocity in the negative direction. Motion will continue until first the rising edge of the Homing switch is detected (position B) and then the rising edge of the first index pulse (position 12) is detected.

NOTE: if it the axis is on the wrong side of the homing switch when homing is started then the axis will move negative until it contacts the negative limit switch (A1). Upon activating the negative limit switch the axis will change direction (positive) following the procedure as detailed above.

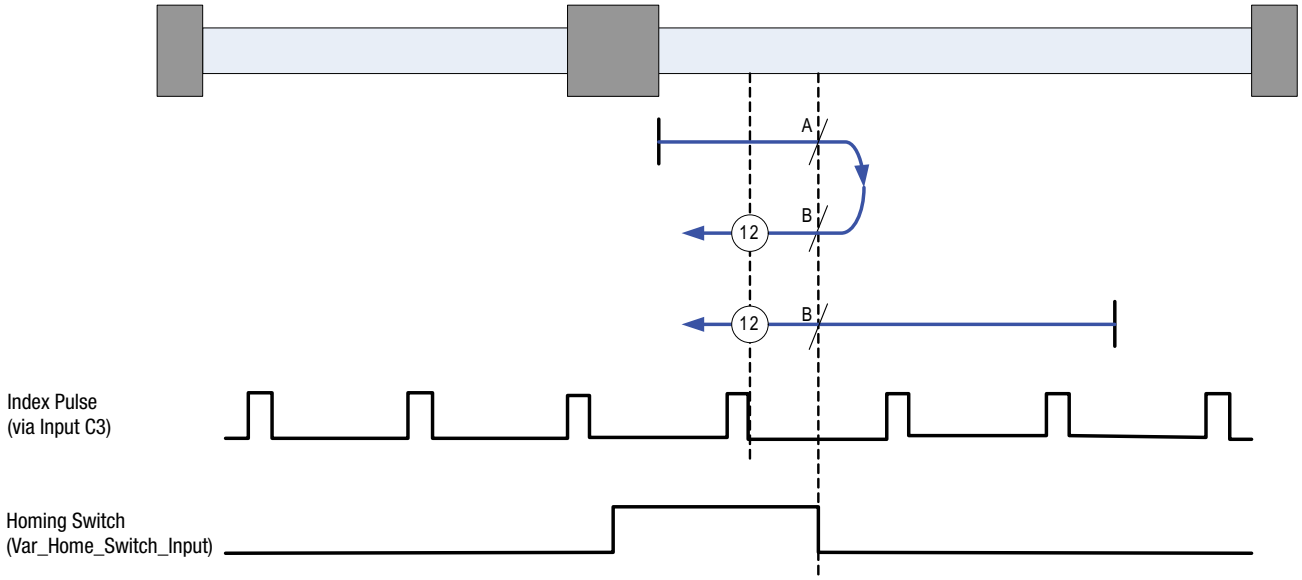


Figure 36: Homing Method 12

2.15.9.13 Homing Method 13: Homing on the Home Switch & Index Pulse

Using this method the initial direction of movement is negative. The home position is the first index pulse to the positive side of the position where the homing switch becomes inactive on its positive edge.

Axis will accelerate to ***fast*** homing velocity in the negative direction and continue until Homing Switch (selectable via Var_Home_Switch_Input Variable) is deactivated (falling edge) shown at position A. Axis then decelerates to zero velocity.

If the homing switch is already active when the homing routine commences then this does not effect this mode of homing as the procedure is searching for falling edge of homing switch in both cases.

Axis will then accelerate to ***fast*** homing velocity in the positive direction. Motion will continue until first the rising edge of the Homing switch is detected (position B) and then the rising edge of the first index pulse (position 13) is detected.

NOTE: if the axis is on the wrong side of the homing switch when homing is started then the axis will move negative until it contacts the negative limit switch (A1). Upon activating the negative limit switch the axis will change direction (positive) following the procedure as detailed above but ignoring the initial move in the negative direction.

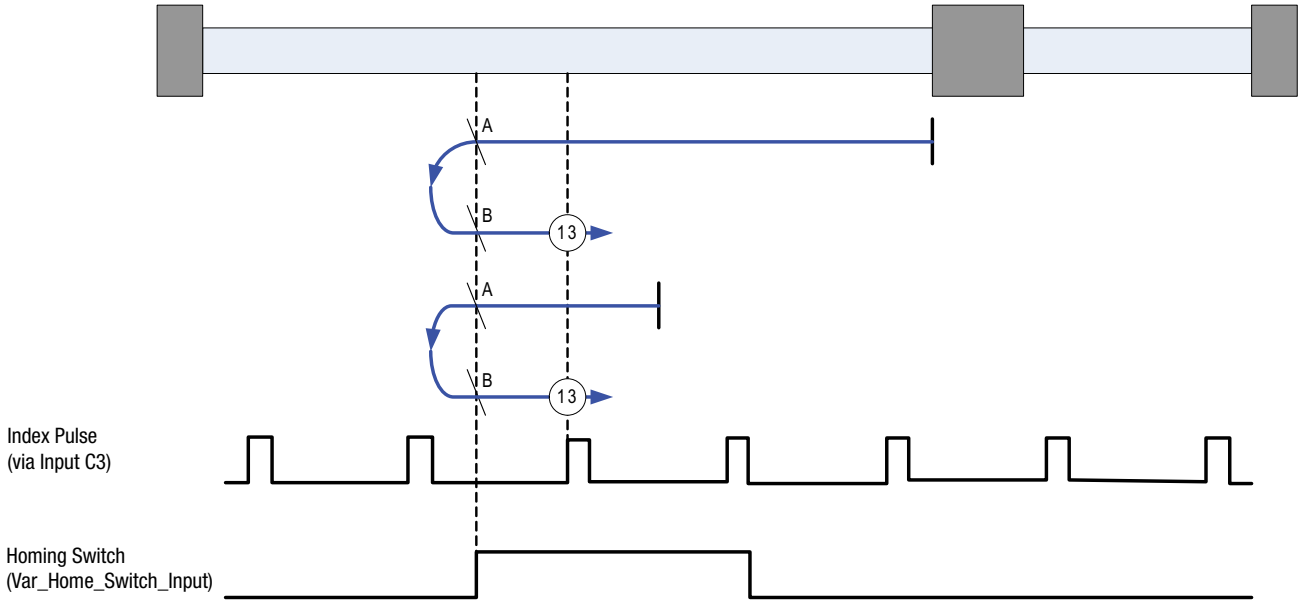


Figure 37: Homing Method 13

2.15.9.14 Homing Method 14: Homing on the Home Switch & Index Pulse

Using this method the initial direction of movement is negative. The home position is the first index pulse to the negative side of the position where the homing switch becomes inactive.

Axis will accelerate to ***fast*** homing velocity in the negative direction and continue until Homing Switch (selectable via Var_Home_Switch_Input Variable) is deactivated (falling edge) shown at position A.

If the homing switch is already active when the homing routine commences then this does not effect this mode of homing as the procedure is searching for falling edge of homing switch in both cases.

Axis will continue running at ***fast*** homing velocity in the negative direction until the rising edge of the first index pulse (position 14) is detected.

NOTE: if the axis is on the wrong side of the homing switch when homing is started then the axis will move negative until it contacts the negative limit switch (A1). Upon activating the negative limit switch the axis will change direction (positive) continuing motion until it sees the rising edge of the homing switch. The axis will then stop and follow the procedure as detailed above.

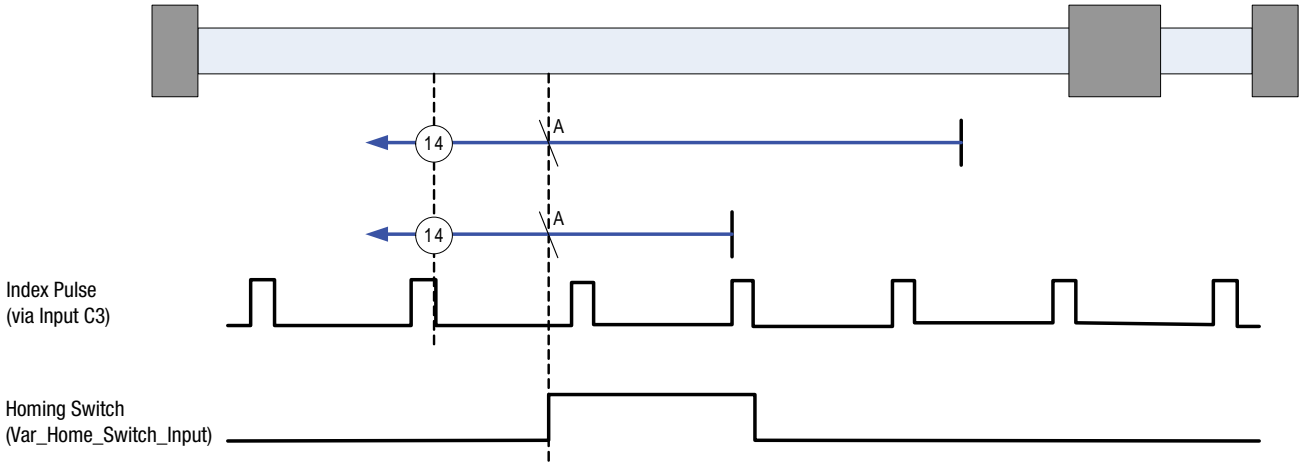


Figure 38: Homing Method 14

2.15.9.15 Homing Method 17: Homing to Negative Limit Switch (without index pulse)

Method 17 is similar to method 1, except that the home position is not dependent on the index pulse but only on the negative limit switch translation.

Using this method the initial direction of movement is negative. The home position is the leading edge of the Negative limit switch.

Axis will accelerate to ***fast*** homing velocity in the negative direction and continue until Negative Limit Switch (A1) is activated (rising edge) shown at position A. Axis then decelerates to zero velocity.

If the negative limit switch is already active when the homing routine commences then this initial move is not executed.

Axis will then accelerate to ***fast*** homing velocity in the positive direction. Motion will continue until the falling edge of the negative limit switch is detected (position B), where the axis will decelerate to 0 velocity.

Axis will then accelerate to ***slow*** homing velocity in the negative direction. Motion will continue until the rising edge of the negative limit switch is detected (position C), where the axis will decelerate to 0 velocity.

Axis will then accelerate to ***slow*** homing velocity ***divided by 100*** in the positive direction. Motion will continue until the falling edge of the negative limit switch is detected (position 17). This is the home position (excluding offset).

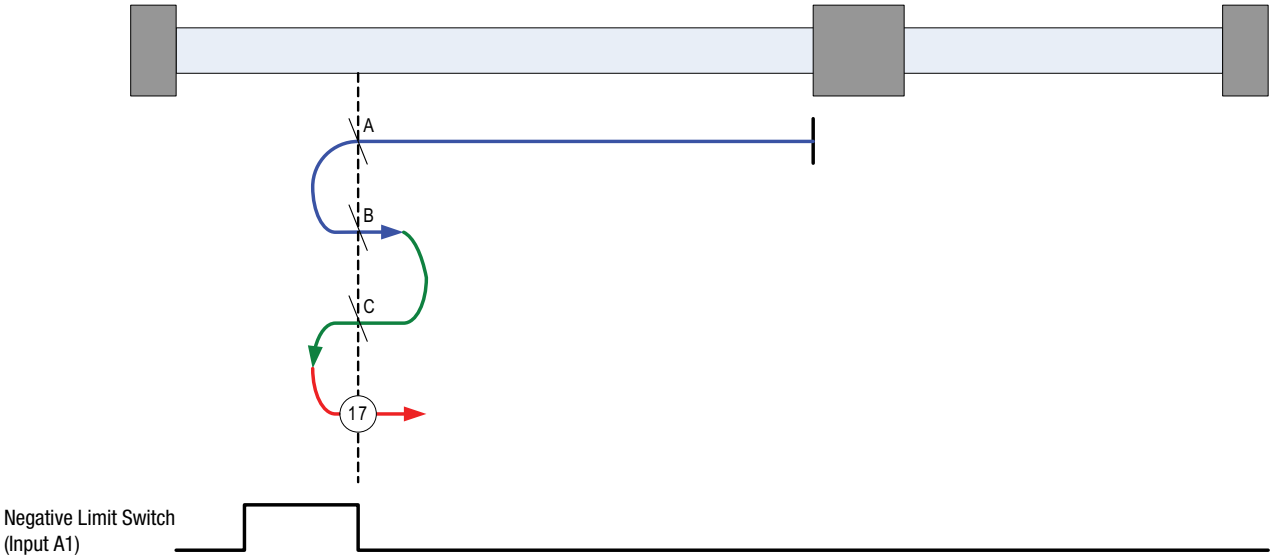


Figure 39: Homing Method 17

2.15.9.16 Homing Method 18: Homing to Positive Limit Switch (without index pulse)

Method 18 is similar to method 2, except that the home position is not dependent on the index pulse but only on the Positive limit switch translation.

Using this method the initial direction of movement is positive. The home position is the leading edge of the Positive limit switch.

Axis will accelerate to ***fast*** homing velocity in the positive direction and continue until Positive Limit Switch (A2) is activated (rising edge) shown at position A. Axis then decelerates to zero velocity.

If the positive limit switch is already active when the homing routine commences then this initial move is not executed.

Axis will then accelerate to ***fast*** homing velocity in the negative direction. Motion will continue until the falling edge of the positive limit switch is detected (position B), where the axis will decelerate to 0 velocity.

Axis will then accelerate to ***slow*** homing velocity in the positive direction. Motion will continue until the rising edge of the positive limit switch is detected (position C), where the axis will decelerate to 0 velocity.

Axis will then accelerate to ***slow*** homing velocity ***divided by 100*** in the negative direction. Motion will continue until the falling edge of the positive limit switch is detected (position 18). This is the home position (excluding offset).

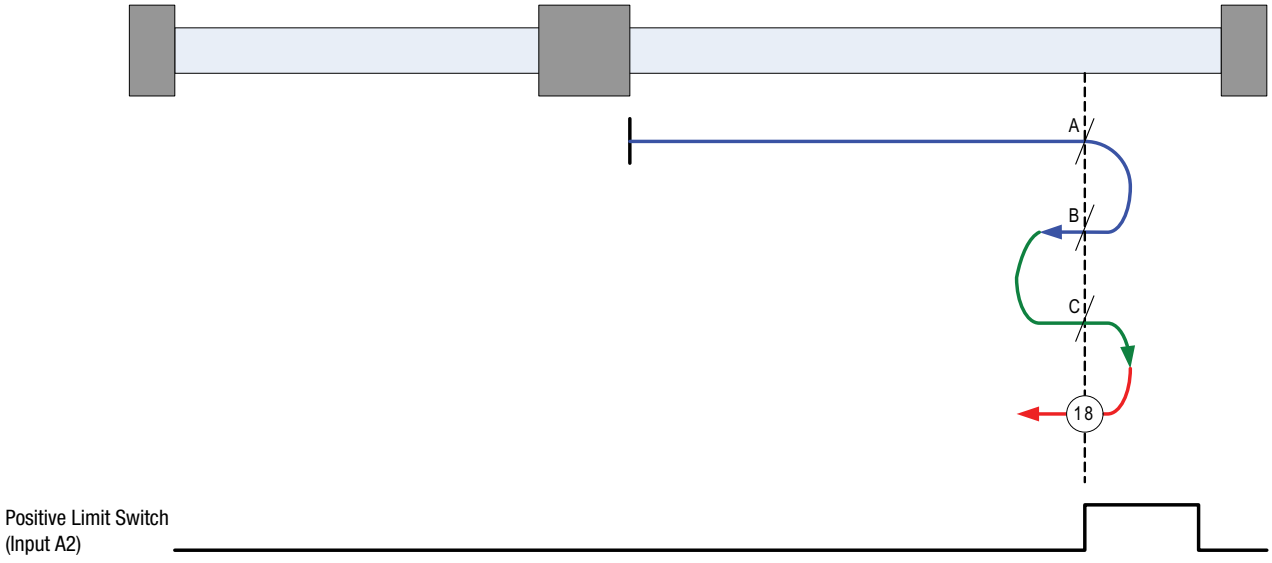


Figure 40: Homing Method 18

2.15.9.17 Homing Method 19: Homing to Homing Switch (without index pulse)

Using this method the initial direction of movement is positive (if the homing switch is inactive). The home position is the leading edge of the homing switch.

Axis will accelerate to ***fast*** homing velocity in the positive direction and continue until the homing switch is activated (rising edge) shown at position A. Axis then decelerates to zero velocity.

If the homing switch is already active when the homing routine commences then this initial move is not executed.

Axis will then accelerate to ***fast*** homing velocity in the negative direction. Motion will continue until the falling edge of the homing switch is detected (position B), where the axis will decelerate to 0 velocity.

Axis will then accelerate to ***slow*** homing velocity in the positive direction. Motion will continue until the rising edge of the homing switch is detected (position C), where the axis will decelerate to 0 velocity.

Axis will then accelerate to ***slow*** homing velocity in the negative direction. Motion will continue until the falling edge of the homing switch is detected (position 19). This is the home position (excluding offset).

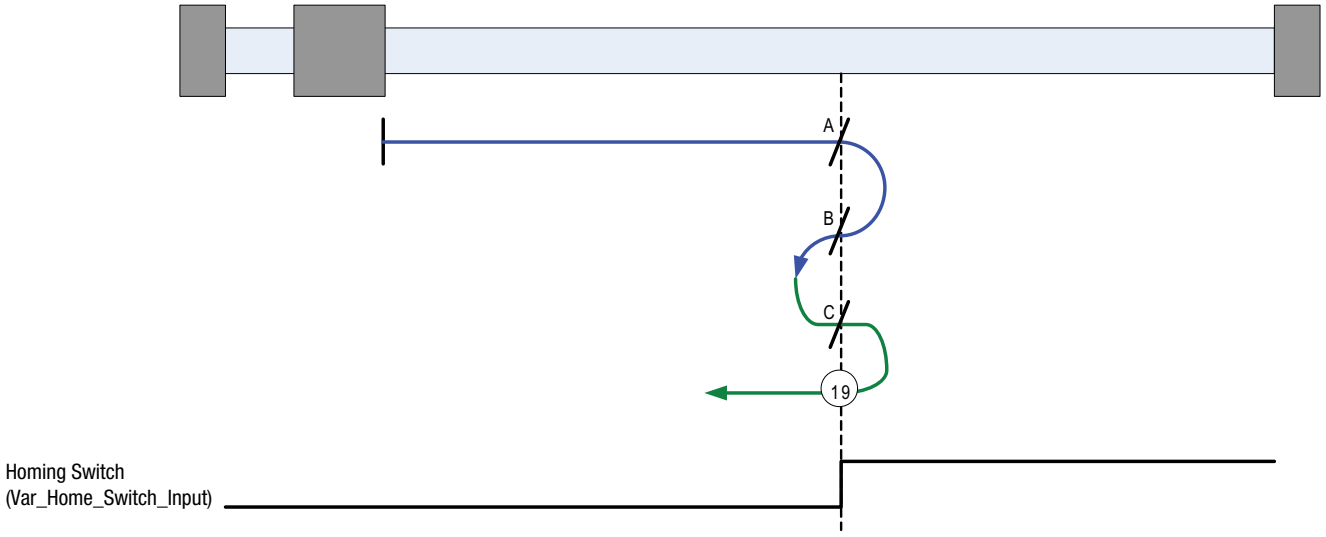


Figure 41: Homing Method 19

2.15.9.18 Homing Method 21: Homing to Homing Switch (without index pulse)

Using this method the initial direction of movement is negative (if the homing switch is inactive). The home position is the leading edge of the homing switch.

Axis will accelerate to ***fast*** homing velocity in the negative direction and continue until the homing switch is activated (rising edge) shown at position A. Axis then decelerates to zero velocity.

If the homing switch is already active when the homing routine commences then this initial move is not executed.

Axis will then accelerate to ***fast*** homing velocity in the positive direction. Motion will continue until the falling edge of the homing switch is detected (position B), where the axis will decelerate to 0 velocity.

Axis will then accelerate to ***slow*** homing velocity in the negative direction. Motion will continue until the rising edge of the homing switch is detected (position C), where the axis will decelerate to 0 velocity.

Axis will then accelerate to ***slow*** homing velocity in the positive direction. Motion will continue until the falling edge of the homing switch is detected (position 21). This is the home position (excluding offset).

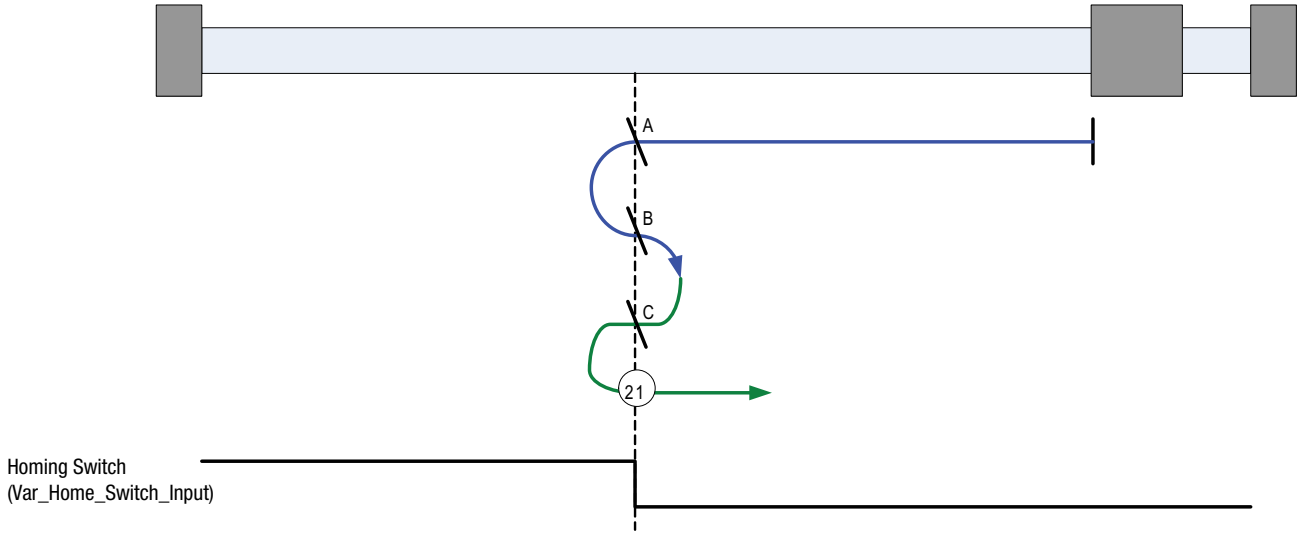


Figure 42: Homing Method 21

2.15.9.19 Homing Method 23: Homing to Homing Switch (without index pulse)

Using this method the initial direction of movement is positive (if the homing switch is inactive). The home position is the leading edge of the homing switch.

Axis will accelerate to ***fast*** homing velocity in the positive direction and continue until the homing switch (selectable via Var_Home_Switch_Input Variable) is activated (rising edge) shown at position A. Axis then decelerates to zero velocity.

If the homing switch is already active when the homing routine commences then this initial move is not executed.

Axis will then accelerate to ***fast*** homing velocity in the negative direction. Motion will continue until the falling edge of the homing switch is detected (position B), where the axis will decelerate to 0 velocity.

Axis will then accelerate to ***slow*** homing velocity in the positive direction. Motion will continue until the rising edge of the homing switch is detected (position C), where the axis will decelerate to 0 velocity.

Axis will then accelerate to ***slow*** homing velocity in the negative direction. Motion will continue until the falling edge of the homing switch is detected (position 23). This is the home position (excluding offset).

NOTE: if the axis is on the wrong side of the homing switch when homing is started then the axis will move positive until it contacts the positive limit switch (A2). Upon activating the positive limit switch the axis will change direction (negative) following the procedure as detailed above but ignoring the initial move in the positive direction.

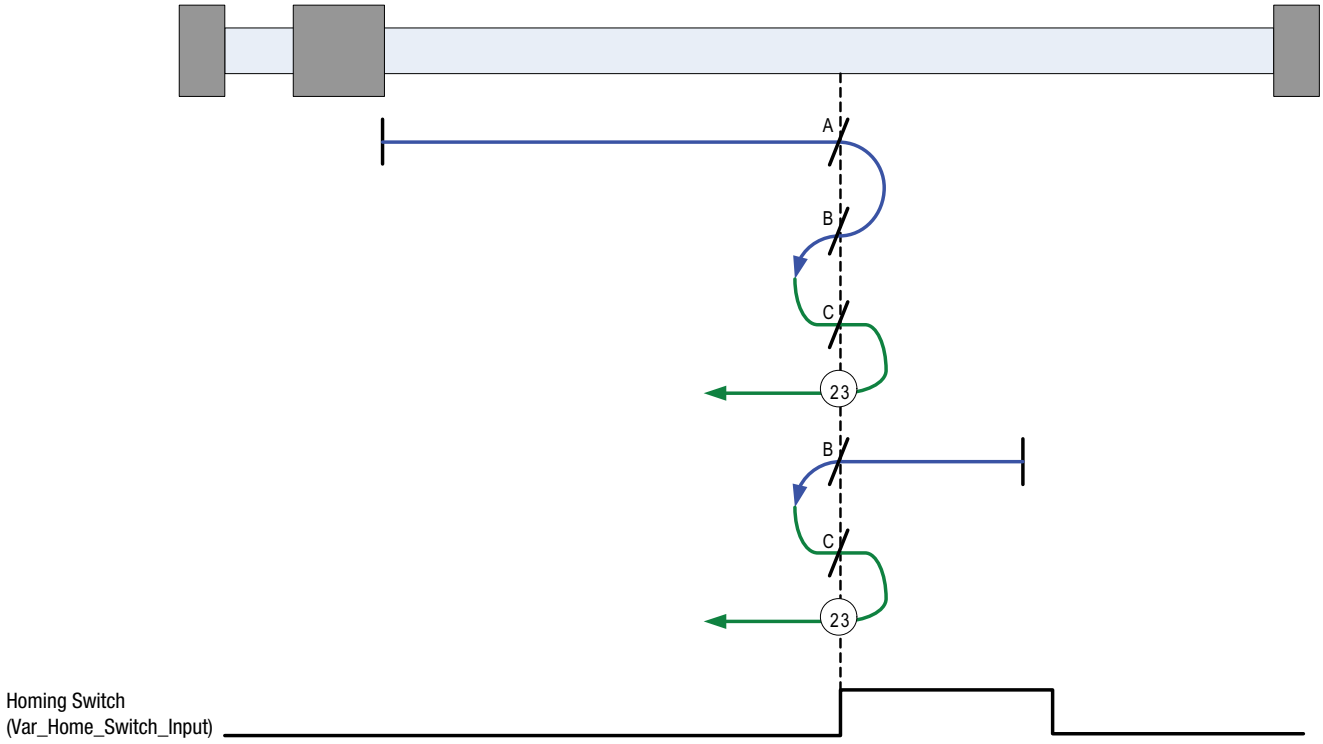


Figure 43: Homing Method 23

2.15.9.20 Homing Method 25: Homing to Homing Switch (without index pulse)

Using this method the initial direction of movement is positive. The home position is the negative edge of the homing switch.

Axis will accelerate to ***fast*** homing velocity in the positive direction and continue until Homing Switch (selectable via Var_Home_Switch_Input Variable) is deactivated (falling edge) shown at position A. Axis then decelerates to zero velocity.

If the homing switch is already active when the homing routine commences then this does not effect this mode of homing as the procedure is searching for falling edge of homing switch in both cases.

Axis will then accelerate to ***slow*** homing velocity in the negative direction. Motion will continue until the rising edge of the homing switch is detected (position B), where the axis will decelerate to 0 velocity.

Axis will then accelerate to ***slow*** homing velocity in the positive direction. Motion will continue until the falling edge of the homing switch is detected (position 25). This is the home position (excluding offset).

NOTE: if the axis is on the wrong side of the homing switch when homing is started then the axis will move positive until it contacts the positive limit switch (A2). Upon activating the positive limit switch the axis will change direction (negative) continuing motion until it sees the rising edge of the homing switch. The axis will then stop and follow the procedure as detailed above.

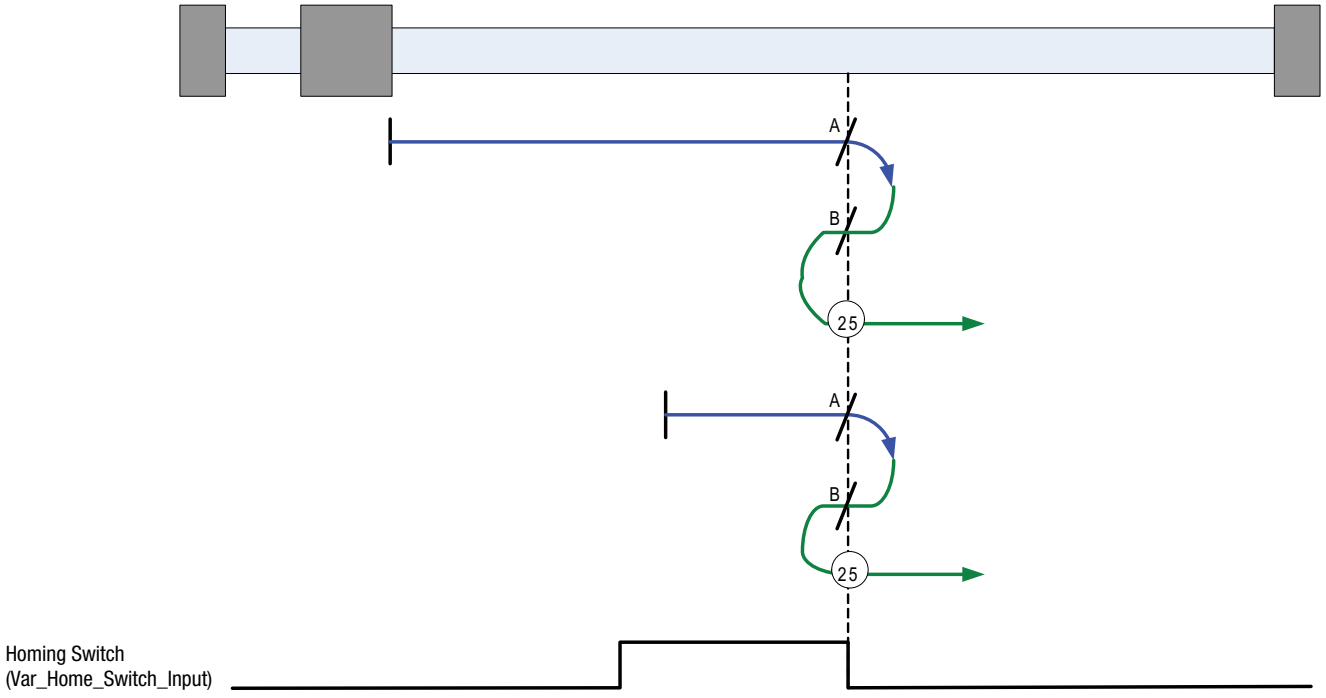


Figure 44: Homing Method 25

2.15.9.21 Homing Method 27: Homing to Homing Switch (without index pulse)

Using this method the initial direction of movement is negative. The home position is the negative edge of the homing switch.

Axis will accelerate to ***fast*** homing velocity in the negative direction and continue until Homing Switch (selectable via Var_Home_Switch_Input Variable) is deactivated (falling edge) shown at position A. Axis then decelerates to zero velocity.

If the homing switch is already active when the homing routine commences then this does not effect this mode of homing as the procedure is searching for falling edge of homing switch in both cases.

Axis will then accelerate to ***slow*** homing velocity in the positive direction. Motion will continue until the rising edge of the homing switch is detected (position B), where the axis will decelerate to 0 velocity.

Axis will then accelerate to ***slow*** homing velocity in the negative direction. Motion will continue until the falling edge of the homing switch is detected (position 27). This is the home position (excluding offset).

NOTE: if the axis is on the wrong side of the homing switch when homing is started then the axis will move negative until it contacts the negative limit switch (A1). Upon activating the negative limit switch the axis will change direction (positive) continuing motion until it sees the rising edge of the homing switch. The axis will then stop and follow the procedure as detailed above.

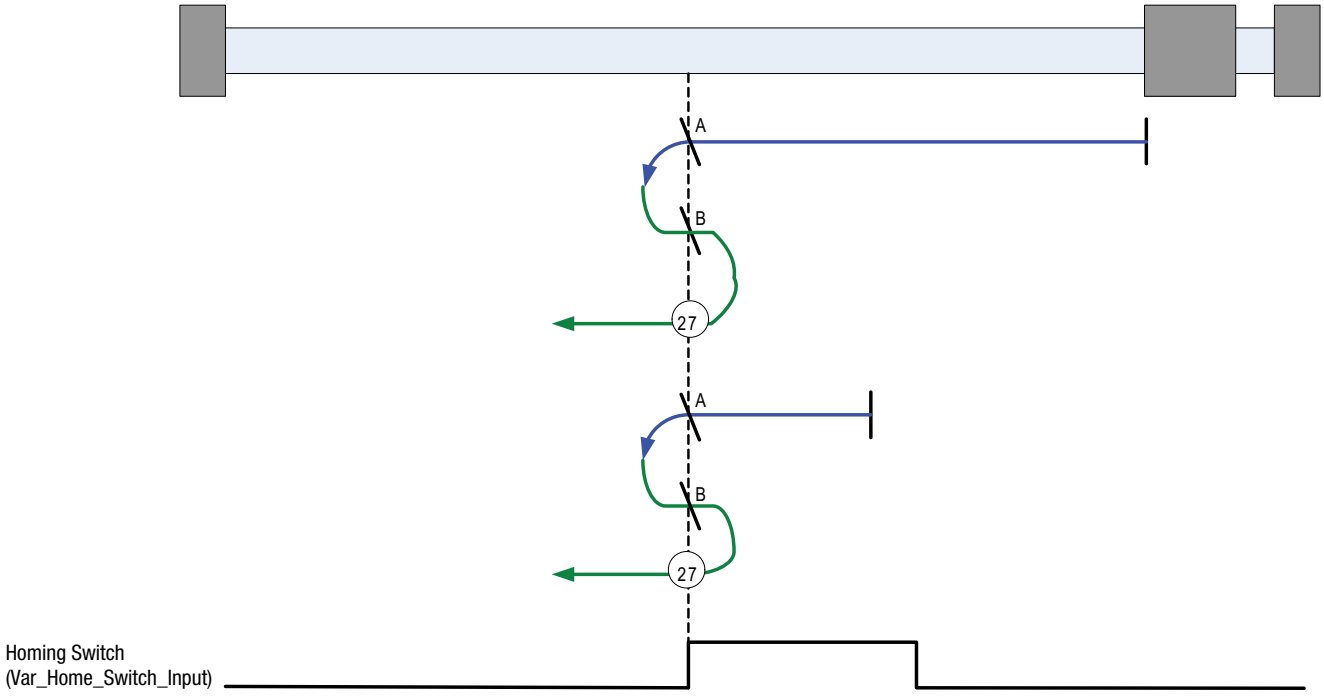


Figure 45: Homing Method 27

2.15.9.22 Homing Method 29: Homing to Homing Switch (without index pulse)

Using this method the initial direction of movement is negative (if the homing switch is inactive). The home position is the leading edge of the homing switch.

Axis will accelerate to ***fast*** homing velocity in the negative direction and continue until the homing switch (selectable via Var_Home_Switch_Input Variable) is activated (rising edge) shown at position A. Axis then decelerates to zero velocity.

If the homing switch is already active when the homing routine commences then this initial move is not executed.

Axis will then accelerate to ***fast*** homing velocity in the positive direction. Motion will continue until the falling edge of the homing switch is detected (position B), where the axis will decelerate to 0 velocity.

Axis will then accelerate to ***slow*** homing velocity in the negative direction. Motion will continue until the rising edge of the homing switch is detected (position C), where the axis will decelerate to 0 velocity.

Axis will then accelerate to ***slow*** homing velocity in the positive direction. Motion will continue until the falling edge of the homing switch is detected (position 29). This is the home position (excluding offset).

NOTE: if the axis is on the wrong side of the homing switch when homing is started then the axis will move negative until it contacts the negative limit switch (A1). Upon activating the negative limit switch the axis will change direction (positive) following the procedure as detailed above but ignoring the initial move in the negative direction.

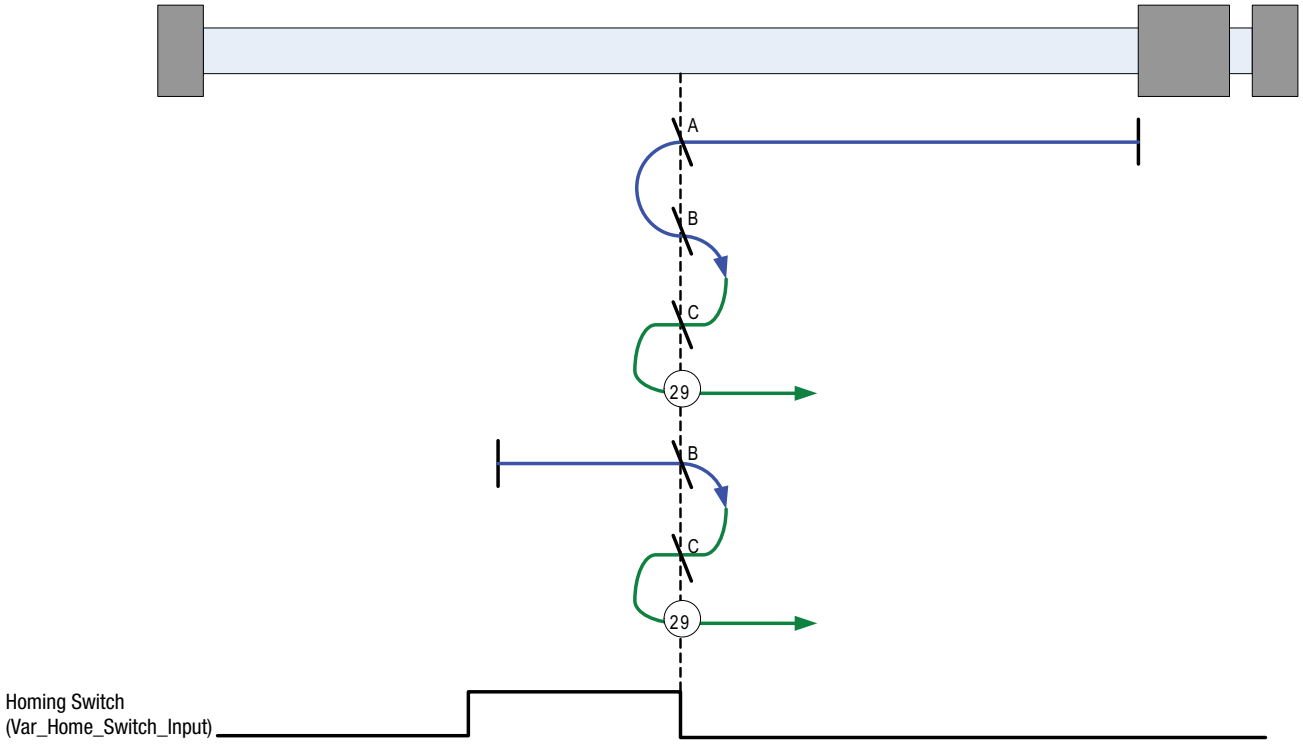


Figure 46: Homing Method 29

2.15.9.23 Homing Method 33: Homing to an Index Pulse

Using this method the initial direction of movement is negative. The home position is the first index pulse to the negative side of the shaft starting Position. Axis will accelerate to ***fast*** homing velocity in the negative direction and continue until the rising edge of the first index pulse (position 33) is detected.

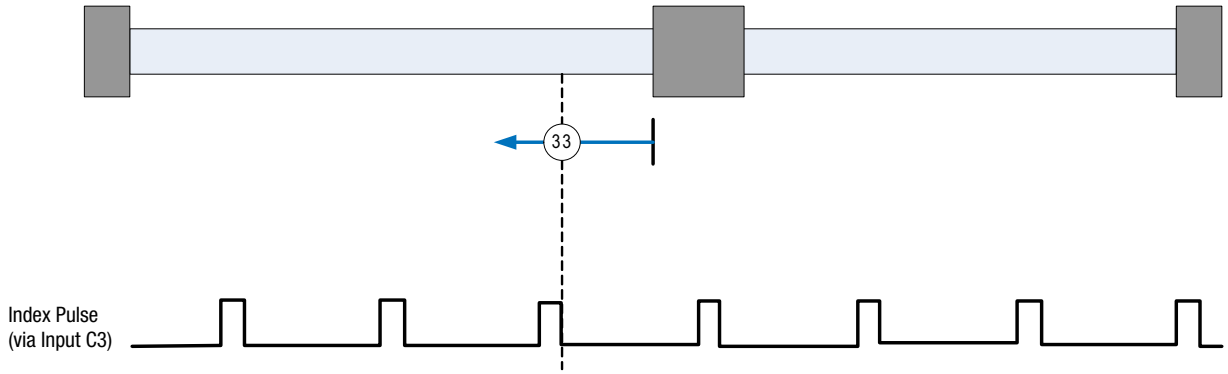


Figure 47: Homing Method 33

2.15.9.24 Homing Method 34: Homing to an Index Pulse

Using this method the initial direction of movement is positive. The home position is the first index pulse to the positive side of the shaft starting Position. Axis will accelerate to ***fast*** homing velocity in the positive direction and continue until the rising edge of the first index pulse (position 34) is detected.

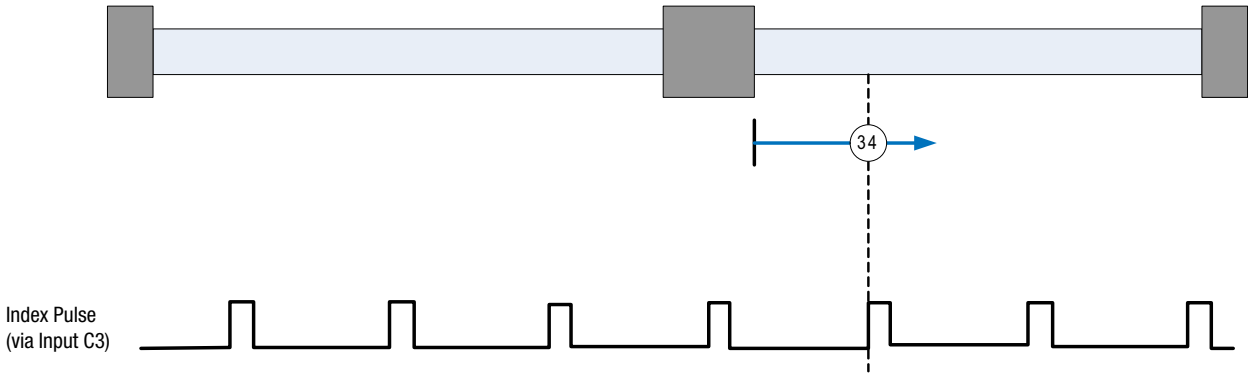


Figure 48: Homing Method 34

2.15.9.25 Homing Method 35: Using Current Position as Home

Using this method the current position of the axis is taken as the home position. There is no motion of the motor shaft during this procedure. Any offset specified (via the Var_Home_Offset Variable) will be added to the shaft's present position to create the home/zero position.

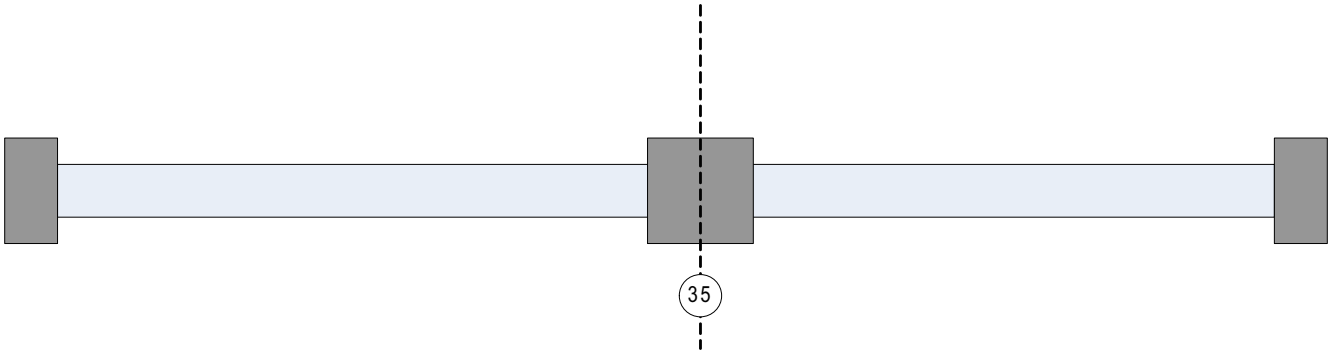


Figure 49: Homing Method 35

2.15.10 Homing Mode Operation Example

The following steps are needed to execute the homing operation from the user program or under interface control.

1. Set Fast homing speed: Variable #242
2. Set Slow homing speed: Variable #243
3. Set Homing accel/decel: Variable #239
4. Set home offset:
   a. In User Units Variable #240
   b. In encoder pulses Variable #241
5. Set Home Switch Input Variable #246
6. Select Home Method Variable #244

To start, execute the HOME command. Refer to the example herein.

There are two methods of starting pre-defined homing operation, the 'HOME' command and the Var_Start_Homing variable. When Homing is initiated from the user program the 'HOME' command should always be used. The HOME command is a blocking instruction that prevents further execution of the Main Program until homing operation is completed. Any events that are enabled whilst homing is carried out will continue to process.

WARNING!
If using firmware prior to 4.50 then execution of homing functionality does not prevent simultaneous execution of subsequent programming statements and it is required to immediately follow the HOME command with the following code line:

WAIT UNTIL VAR_EXSTATUS & 0x400000 == 0x400000.

Doing this ensures no further lines of code will be executed until homing is complete.

The home start variable (Var_Start_Homing) is used to initiate pre-defined homing functionality from a host interface. It should not be used if the drive contains or is executing a user program. Var_Start_Homing range is: 0 or 1. When set to 0, no action occurs. When set to 1, the homing operation is started.

```
;Program start-----------------------------------------------------------------------------
;
;
    UNITS=1                                             ;rps

    Accel=1000
    Decel=1000
    MaxV =20

;some program statements…
;
;
;Homing specific set up
    VAR_HOME_FAST_VEL=                      10          ;rps
    VAR_HOME_SLOW_VEL=                      1           ;rps
    VAR_HOME_ACCEL=                         100         ;rps/sec^2
    VAR_HOME_OFFSET=                        0           ;no offset from sensor
    VAR_HOME_SWITCH_INPUT=                  4           ;input B1 (0-A1, 1-A2…3-A4, 4-B1,…11-C4)
    VAR_HOME_METHOD=                        4           ;see table 21
    ENABLE
    HOME                                                ;start homing sequence
;The statement below MUST be included immediately after the Home command on drives containing
;firmware releases prior to version 4.50
    WAIT UNTIL VAR_EXSTATUS & 0x400000 ==   0x400000    ;wait for homing complete
;Drive homed

;Program statements…
END
```

3. Reference

3.1 Program Statement Glossary

Each programming statement is documented using the tabular format shown in Tables 22 and 23. The individual program statements are listed in this section in alphabetical order with detailed descriptions in Tables 24 through 62.

Table 22: Language Format

KEYWORD	Description	Type
Purpose		
Syntax	KEYWORD <ARGUMEMTS> ,[MODIFIERS]	
Remarks		
See Also		
Example		

Table 23: Field Descriptions

Field	Descriptions
KEYWORD:	The KEYWORD is the name of the programming statement as it would appear in a program.
Description:	The description is an interpretation of the keyword. For example: MOVEP is the keyword and Move to Position would be a description. The description is provided only as an aid to the reader and may not be used in a program.
Type:	The type field will identify the Keyword as either a Statement or a Pseudo statement. Statements are actual instructions converted to machine code by the compiler and form executable commands within the drive programming. Pseudo statements add convenience to the programmer but do not form instructions in their own right. They are therefore not executable code and are effectively removed when the program is compiled to it's native state by the compiler.
Purpose:	Purpose or Function of the Keyword (Programming Statement).
Syntax:	This field shows proper usage of the keyword. Arguments will be written in < > brackets. Optional arguments will be contained within [ ] brackets.
Arguments:	The data that is supplied with a statement that modifies the behavior of the statement. For example, MOVED=100. MOVED is the statement and 100 is the argument.
Remarks:	The remark field contains additional information about the use of the statement.
See Also:	This field contains a list of statements that are related to the purpose of the keyword.
Example:	The example field contains a code segment that illustrates the usage of the keyword

Table 24: ASSIGN

ASSIGN	Assign Input As Index Bit	Statement
Purpose	Assign keyword causes a specified input to be assigned to a particular bit of system variable INDEX. Up to 8 digital inputs can be assigned to the first eight bits (bits 0 - 7) of the INDEX system variable in any order or combination. The purpose of the Assign Keyword and INDEX system Variable is to allow the creation of a custom input word for inclusion in the user program. Good examples of it's use are for implementing easy selection of preset torque, velocity or position values within the user program.	
Syntax	ASSIGN INPUT <input name> AS BIT <bit #> Input name (IN_A1..IN_A2 etc.) Bit# INDEX variable bit number from 0 to 7	
Remarks	Assign statements typically appear at the start of the program (Initialize and set Variables section) but can be included in other code sections with the exception of Events and the Fault Handler.	
See Also	VAR_IOINDEX Variable (#220)	

Example:

```
ASSIGN INPUT IN_B1 AS BIT 0   ;index bit 0 state matches state of input B1
ASSIGN INPUT IN_B2 AS BIT 1   ;index bit 1 state matches state of input B2

Program Start:
; <statements>
If Index == 0                 ; If neither IN_B1 or IN_B2 is on
        MoveP 0               ; Move to Absolute Position 0
Endif

If Index == 1                 ; If IN_B1 is on and IN_B2 is off
        MoveP 10              ; Move to Absolute Position 10
Endif

; If Index == 2               .....
```

Table 25: DEFINE

DEFINE	Define name	Pseudo-statement
Purpose	DEFINE is used to define symbolic names for User Variables, constants, and Digital I/O for programming convenience. Define statements greatly enhance program understanding by allowing the user to program using symbolic strings (names) relevant to their application. DEFINE can be used also to substitute a symbolic string.	
Syntax	DEFINE <name> <synonym> name any symbolic string synonym User Variable, constant, or Digital I/O Flag that symbolic string will represent	
Remarks:	DEFINE statements can be located anywhere within the user program (with the exception of events and the fault handler). Normally practice however is to place definitions at the start of the program prior to any executable code.	
See Also		

Example:

```
Define Start_Button IN_B1                          ; Define a Digital Input
Define System_Stop Out2                            ; Define a Digital Output
Define Loop_Counter V5                             ; Define a User Variable
Define Loop_Increment 1                            ; Define a Constant Value

Program_Start:                                     ; Label Program Start
If Start_Button == 0                               ; If input B1 is off
   Disable                                         ; Disable Servo
   System_Stop = 1 ; Turn on Output 2
Else                                               ; Otherwise
   System_Stop = 0 ; Turn off Output 2
   Enable                                          ; Enable Servo
   MoveD 10                                        ; Move (increment) Distance 10
   Loop_Counter = Loop_Counter + Loop_Increment    ; Increment Variable V5 by 1
Endif
Goto Program Start ; Goto Label Program Start
```

Table 26: DISABLE

DISABLE	Disables the drive	Statement
Purpose	DISABLE turns OFF the drive output to the motor. Drive shows ‘Dis’ on display when in a disabled state.	
Syntax	DISABLE	
Remarks	Once the DISABLE statement is executed, the power to the motor is turned off and the motor can move freely. When disabled the drive will continue to monitor feedback and the actual position variable (APOS) will continue to update with the current position of the motor. The target position variable (TPOS) will be updated with the value of the actual position variable (APOS) on Enable to prevent unexpected motion from the motor shaft.	
See Also	ENABLE	

WARNING!
Work should not be carried out on the drive/system without the drive first being isolated from its mains supply. The disabled condition is not an indication that the motor or system is safe to work on as an Enable/run condition could result from execution of the programmers programming code, from a host interface, or controller.

Example:

```
If Start_Button == 0       ; If input B1 is off
   Disable                 ; Disable Servo
Else                       ; Otherwise
   Enable                  ; Enable Servo
   MoveD 10                ; Move (increment) Distance 10
Endif
```

Table 27: DO UNTIL

DO UNTIL	Do/Until	Statement
Purpose	The DO / UNTIL statement is used to execute a statement or set of statements repeatedly until a logical condition becomes true. The Do / Until statements enclose the program code to be repeatedly executed with the UNTIL statement containing the logical statement for exit of the loop.	
Syntax	DO {statement(s)}… UNTIL <condition> {statement(s)} any valid statement(s) <condition> The condition to be tested.	
Remarks	The statement or statements contained within a DO / UNTIL loop will always be executed at least once because the logical condition to be tested is contained within the UNTIL statement in the last statement of the loop.	
See Also	WHILE, IF	
Example:		

```
V0 = 0              ; Set V0 to Value 0
                    ; Create Loop to perform Move command 12 times
DO                  ; Start of Do Loop
   V0 = V0 + 1      ; Add 1 to Variable V0
   Moved 5          ; Move (incremental) distance 5
Until V0 == 12      ; Loop back to DO Statement, Repeat Until Logic True
```

Table 28: ENABLE

ENABLE	Enables the drive	Statement
Purpose	Enable turns on drive output to the motor. Drive shows ‘Run’ on display when in the enabled state.	
Syntax	ENABLE	
Remarks	Once a drive is enabled motion can be commanded from the user program. Commanding motion while the drive is disabled will result in fault trip (F_27).	
See Also	DISABLE	
Example:		

```
If Start_Button == 0     ; If input B1 is off
      Disable            ; Disable Servo
Else                     ; Otherwise
      Enable             ; Enable Servo
      MoveD 10           ; Move (increment) Distance 10
Endif
```

Table 29: END

END	END program	Statement
Purpose	This statement is used to terminate (finish) user program and its events.	
Syntax	END	
Remarks	END can be used anywhere in program	
See Also	DISABLE	
Example:		

```
END       ;end user program
```

Table 30: EVENT

EVENT	Starts Event handler	Statement

Purpose EVENT keyword is used to create scanned events within the user program. Statement also sets one of 4 possible types of events.

Syntax Any one of the 4 syntax examples herein may be used:

1. EVENT <name> INPUT <inputname> RISE
2. EVENT <name> INPUT <inputname> FALL
3. EVENT <name> TIME <period>
4. EVENT <name> <expression>

name	any valid alphanumeric string
inputname	any valid input “IN_A1 - IN_C4”
period	any integer number. Expressed in ms
expression	any arithmetic or logical expression

The following statements can ***not*** be used within event’s handler:

MOVE	MOVED	MOVEP	MOVEDR
MOVEPR	MDV	MOTION SUSPEND	MOTION RESUME
STOP MOTION	DO/UNTIL	GOTO	GOSUB
HALT	VELOCITY ON/OFF	WAIT	WHILE/ENDWHILE
ASSIGN	END	ON FAULT/END FAULT	RESUME
RETURN			

While GOTO or GOSUB are restricted, a special JUMP statement can be used for program flow change from within event handler. See JUMP statement description in Language Reference section.

Program labels are also not permitted within event code.

Remarks

For syntax 1 and 2:

The Event will occur when the input defined by the <name> transition from low to high, for syntax 1 (RISE) and from high to low for syntax 2 (FALL).

For syntax 3:

The Event will occur when the specified , <period>, period of time has expired. This event can be used as periodic event to check for some conditions.

For syntax 4

The Event will occur when the expression, <expression>, evaluates to be true. The expression can be any valid arithmetic or logical expression or combination of the two. This event can be used when implementing soft limit switches or when changing the program flow based on some conditions. Any variable, (user and system), or constants can be used in the expression. The event will only trigger when the logic transitions from False to True. Further occurrence of the event will not occur while the condition remains true.

See Also ENDEVENT, EVENT ON/OFF, EVENTS ON/OFF

Example:

```
EVENT InEvent IN_A1 RISE
   V0 = V0+1            ;V0 increments by 1 each time IN_A1 transitions from low to high
ENDEVENT
EVENT period TIME 1000  ;1000 ms = 1Sec
   V3=V0-V1             ;Event subtracts V1 from V0 and stores result in V3 every second
ENDEVENT
;-------------------------------------------------------------------------
   EVENT  InEvent  ON   ;Statements in main program to turn individual events on
   EVENT period    ON
   {program statements}
END
```

Table 31: ENDEVENT

ENDEVENT	END of Event handler	Statement
Purpose	Indicates end of the scanned event code	
Syntax	ENDEVENT	
Remarks		
See Also	EVENT, EVENT ON/OFF, EVENTS ON/OFF	

Example:	`EVENT InputRise IN_B4 RISE`
	`V0=V0+1`
	`ENDEVENT`	

Table 32: EVENT ON/OFF

EVENT ON/OFF	Turn events on or off	Statement
Purpose	Turns ON or OFF events created by an EVENT statement	
Syntax	EVENT <name> ON EVENT <name> OFF <name> Event name	
Remarks		
See Also	EVENT	

Example:

```
; Events Section
EVENT InputRise IN_B4 RISE
   V0=V0+1
ENDEVENT

; Main Program
EVENT InputRise ON
; statements...
EVENT InputRise OFF
```

Table 33: EVENTS ON/OFF

EVENTS OFF/ON	Globally Disables/re-enables events	Statement
Purpose	EVENTS OFF command when executed will disable any events currently enabled (running). EVENTS ON Command re-enables any events previously disabled through the EVENTS OFF command. EVENTS ON is not a global enable of all declared events. Events status is indicated through bit #30 of the DSTATUS register or by system flag 'F_EVENTSOFF'. EVENTS OFF/ON allows for easy disable and re-activation of events in sections of the main program or subroutines that the programmer doesn't want interrupted by event code.	
Syntax	EVENTS OFF Disables execution of all events EVENTS ON Restores execution of previously enabled events.	
Remarks	Events are globally disabled after a program reset is made or a fault occurs. Events are re-enabled by executing the individual EVENT <name> ON statement following either a Reset or a Fault Condition.	
See Also	EVENT	

Example:

```
*****************************************************************************
    EVENT SKIPOUT IN_B4 RISE      ;check for rising edge of input B4
      JUMP TOGGLE                 ;redirect code execution to TOGGLE
    ENDEVENT                      ;end the event
    EVENT OVERSHOOT IN_B3 RISE    ;check for rising edge of input B3
      JUMP SHUTDOWN               ;redirect code execution to SHUTDOWN
    ENDEVENT                      ;end the event
*****************************************************************************
    EVENT SKIPOUT ON
    EVENT OVERSHOOOT ON
*****************************************************************************
    ……..…User code……..
    EVENTS OFF                    ;turns off all events
    ……..…User code……..
    EVENTS ON                     ;turns on any event previously activated
```

Table 34: FAULT

FAULT	User generated fault	Statement
Purpose	Allows the user program to set a custom system fault. This is useful when the programmer needs to define a fault code and fault process for custom conditions like data supplied by interface out of range etc. Custom fault numbers must be in region of 128 to 240 (decimal)	
Syntax	FAULT <FaultNumber>	Sets system fault. <FaultNumber> constant in range 128-240
Remarks	Custom fault will be processed in the same way as a system fault. There will be a record in the fault log. Variables are not allowed in this statement.	
See Also	ON FAULT	

Example:

```
FAULT 200        ;Sets fault #200
```

Table 35: GOSUB

GOSUB	Go To subroutine	Statement
Purpose	GOSUB transfers control to subroutine.	
Syntax	GOSUB <subname>	
	<subname> a valid subroutine name	
Remarks	After return from subroutine program resumes from next statement after GOSUB	
See Also	GOTO, JUMP, RETURN	

Example:

```
DO
   GOSUB CALCMOVE     ;Go to CALCMOVE Subroutine
   MOVED V1           ;Move distance calculated in Subroutine
UNTIL INA1
END

SUB CALCMOVE:
   V1=(V2+V3)/2       ;Subroutine statement, Calculates value for V1
RETURN                ;Return to main program execution
```

Table 36: GOTO

GOTO	Go To	Statement
Purpose	Transfer program execution to label following the GOTO instruction.	
Syntax	GOTO <label>	
Remarks	<Label> must be a valid program reference label (alphanumeric string, 64 characters in length, and ending with a colon ":") contained within the user program. The GOTO statement can be located either above or below the program label in the user code.	
See Also	GOSUB, JUMP	

Example:

```
GOTO Label2
{Statements…}

Label2:

{Statements…}
```

Table 37: HALT

HALT	Halt the program execution	Statement
Purpose	Used to halt main program execution. Events are not halted by the HALT statement. Event code can restart main program execution by issuing the RESET statement or by executing a JUMP to a Main Program Label from the EVENT handler. With the RESET statement, Main Program execution will recommence on the code line immediately following the HALT Statement. With a Jump command program execution is forced to the program label defined within the argument of the JUMP command.	
Syntax	HALT	
Remarks	This statement is convenient when writing event driven programs.	
See Also	RESET, JUMP, EVENT	

Example:

```
{Statements…}
HALT                  ;halt main program execution and wait for event
```

Table 38: HOME

HOME	Execute homing routine	Statement
Purpose	Used to initiate homing.	
Syntax	HOME	
Remarks	Homing is initiated from the user program using the 'HOME' command. The HOME command is a blocking instruction that prevents further execution of the Main Program until homing operation is completed. Any events that are enabled while homing is carried out will continue to process.	
	WARNING! If using firmware prior to 4.50 then execution of homing functionality does not prevent simultaneous execution of subsequent programming statements and it is required to immediately follow the HOME command with the following code line: WAIT UNTIL VAR_EXSTATUS & 0x400000 == 0x400000.	
See Also		

Example:

```
{Statements…}
HOME                    ;initiate homing routine
```

Table 39: ICONTROL ON/OFF

ICONTROL ON/OFF	Enables interface control	Statement
Purpose	Enables/Disables interface control. Effects bit #27 in DSTATUS register and system flag F_ICONTROLOFF. All interface motion commands and commands changing any outputs will be disabled. See Host interface commands manual for details. This command is useful when the program is processing critical states (example limit switches) and can't be disturbed by the interface.	
Syntax	ICONTROL ON ICONTROL OFF	Enables Interface control Disables interface control
Remarks	After reset interface control is enabled by default.	
See Also		

Example:

```
EVENT LimitSwitch IN_A1 RISE    ;limit switch event
   Jump LimitSwitchHandler      ;jump to process limit switch
ENDEVENT

V0=0                            ;V0 will be used to indicate fault condition
EVENT LimitSwitch ON            ;Turn on event to detect limit switch activation
Again:
HALT                            ;system controlled by interface and Events
LimitSwitchHandler:
   EVENTS OFF                   ;turn off all events
   ICONTROL OFF                 ;disable interface control
   STOP MOTION QUICK
   DISABLE                      ;DISABLE
   V0=1                         ;indicate fault condition to the interface
   ICONTROL ON                  ;Enable Interface Control
   EVENTS ON                    ;turn on events turned off by 'EVENTS OFF'
   GOTO AGAIN
```

Table 40: IF

IF	IF/ENDIF	Statement
Purpose	The IF statement tests for a condition and then executes the specific action(s) between the IF and ENDIF statements if the condition is found to be true. If the condition is false, no action is taken and the instructions following the ENDIF statement are executed. Optionally, using the ELSE statement, a second series of statements may be specified to be executed if the condition is false.	

Syntax	IF <condition>
	{statements 1}
	ELSE
	{statements 2}
	ENDIF	

Remarks		
See Also	WHILE, DO	

Example:

```
IF APOS  > 4              ;If actual position is greater than 4 units
   V0=2
ELSE                      ;otherwise... (actual position equal or less than 4)
   V0=0
ENDIF
;-------------------------------------------------------------------------
If V1 <> V2 && V3>V4      ;If V1 doesn't equal V2 AND V3 is greater than V4
   V2=9
ENDIF
```

Table 41: JUMP

JUMP	Jump to label from Event handler	Statement
Purpose	This is a special purpose statement to be used only in the Event Handler code. When the EVENT is triggered and this statement is processed, execution of the main program is transferred to the <label> argument called out in the "JUMP" statement. The Jump statement is useful when there is a need for the program's flow to change based on some event(s). Transfer of program execution is to the instruction following the label. When a Jump statement is executed within an event, processing of subsequent events is suspended until the next event time cycle.	
Syntax	JUMP <label> <label> is any valid program label	
Remarks	Can be used in EVENT handler only.	
See Also	EVENT	

Example:

```
EVENT ExternalFault INPUT IN_A4 RISE      ;activate Event when IN_A4 goes high
   JUMP ExecuteStop                       ;redirect program to <ExecuteStop>
ENDEVENT
StartMotion:
   EVENT ExternalFault ON
   ENABLE
   MOVED 20
   MOVED -100
   {statements}
END
ExecuteStop:
   STOP MOTION                            ;Motion stopped here
   DISABLE                                ;drive disabled
   Wait Until !IN_A4                      ;Wait Until Input A4 goes low
   GOTO StartMotion
```

Table 42: LOADVARS

LOADVARS	EPM access statements LOADVARS Statement
Purpose	LOADVARS is the command to retrieve the values of the user variables (V0-V31) from the drive's EPM. Using this statement any combinations of variables V0-V31 can be retrieved from the EPM with a single statement. Loads the values of the user's variables (V0-V31) from EPM to the drive operating memory.
Syntax	LOADVARS [Va, Vx-Vy] a,x,y any number from 0 - 31
Remarks	Values that are stored in EPM memory for the User Variables V0-V31 (using host interface or StoreVars command) are automatically transferred into operational memory at power up (a LoadVar statement is not required). Should a User Variable be altered by the user program it is altered only in the operational memory of the drive and can be restored back to its EPM value using the LoadVars statement.
See Also	STOREVARS

Example:

```
…{statements}…
V1=12
STOREVARS [V1]            ;Store V1 variable to EPM memory
…{statements}…
LOADVARS [V1]             ;Retrieve V1 variable
…{statements}…
END                       ;End main program
-----------------------------------------------------------
;Example to specify multiple variables list in a single LoadVar Statement

LOADVARS [V0,V1,V5-V20]  ;load values of V0, V1, V5-V20
-----------------------------------------------------------
```

Table 43: MDV

MDV	Segment Move Statement
Purpose	MDV (Move-Distance-Velocity) defines individual motion segment by specifying distance and final velocity (for each segment) in User Units. Acceleration (or deceleration) is calculated automatically based on these two parameters. This technique allows complicated moves to be created that consist of many segments. Each MDV sequence (series of MDV segments) must start and end with a velocity of 0, hence an MDV sequence must have at least two segments. The MDV statement doesn't suspend execution of the main program. Each segment is loaded into the Motion Queue and the sequence executed sequentially. If the last segment in the Motion Queue doesn't have a final velocity of 0, the drive will generate a "Motion Stack Underflow" fault #24. If the "S" modifier is used in the statement, then the velocity acceleration/deceleration will be S-curved as opposed to linear.
Syntax	MDV <[-]segment distance>,<segment final velocity> [,S] [,S] optional modifier specifies S-curve acceleration / deceleration.
See Also	MOVE, MOVEP, MOVEPR, MOVED, MOVEDR, MOTION SUSPEND, MOTION RESUME

Example:

```
{Statements…}
MDV  5, 10              ;Move 5 user units and accelerate to a velocity of 10
MDV  10,10              ;Move 10 user units and maintain a velocity of 10
MDV 10,5                ;Move 10 user units and decelerate to velocity of 5
MDV  5,0                ;Move 5 user units and decelerate to velocity 0.
                        ;The last MDV must have a final velocity of 0.
{Statements…}
```

Table 44: MEMGET

MEMGET	Memory access statements MEMGET	Statement
Purpose	MEMGET provides command for simplified retrieval of data from the drives RAM memory file through transfer of data to the variables V0-V31. Using this statement any combinations of variables V0-V31 can be retrieved from the RAM file with a single statement.	
Syntax	MEMGET	<offset> [<varlist>]
	<offset>	specifies offset in RAM file where data will be retrieved. Range: -32767 to 32767
	<varlist>	any combinations of variables V0-V31
See Also	MEMSET	

Example:

```
MEMGET 5 [V0]                ;single variable will be retrieved from location 5
MEMGET V1 [V0,V3,V2]         ;variables V0,V3,V2 will be retrieved from
                             ;memory location starting at value held in V1
MEMGET 10 [V3-V7]            ;variables V3 to V7 inclusively will be retrieved
MEMGET V1 [V0,V2,V4-V8]      ;variables V0,V2, V4 through V8 will be retrieved
```

Table 45: MEMSET

MEMSET	Memory access statements MEMSET	Statement
Purpose	MEMSET provides command for simplified storage of data to the drives RAM memory file through transfer of data from variables V0-V31. Using this statement any combinations of variables V0-V31 can be stored in the RAM file with a single statement.	
Syntax	MEMSET	<offset> [<varlist>]
	<offset>	specifies offset in RAM file where data will be stored. Range: -32767 to 32767
	<varlist>	any combinations of variables V0-V31
See Also	MEMGET	

Example:

```
MEMSET 5 [V0]                ;single variable will be stored in location 5
MEMSET V1 [V0,V3,V2]         ;variables V0,V3,V2 will be stored in memory
                             ;location starting at value held in V1
MEMSET 10 [V3-V7]            ;variables V3 to V7 inclusively will be stored
MEMSET V1 [V0,V2,V4-V8]      ;variables V0,V2, V4 through V8 will be stored.
```

Table 46: MOTION RESUME

MOTION RESUME	Resume Motion	Statement
Purpose	Statement resumes motion previously suspended by MOTION SUSPEND. If motion was not previously suspended, this has no effect on operation.	
Syntax	MOTION RESUME	
Remarks	Any motion command executed in the user program while motion is suspended will be placed in the motion queue but not executed. Motion commands accumulated on the motion stack will be performed in the order they were loaded to the motion queue on execution of the Motion Resume statement.	
See Also	MOVE, MOVEP, MOVEDR, MOVED, MOVEPR ,MDV, MOTION SUSPEND	

Example:

```
…{statements}
MOTION RESUME ;Motion is resumed from first command in motion Queue (if any)
…{statements}
```

Table 47: MOTION SUSPEND

MOTION SUSPEND	Suspend	Statement
Purpose	This statement is used to temporarily suspend execution of motion. The Motion Suspend command does not flush the Motion Queue of any accumulated motion commands but will suspended further motion until the Motion Resume command is processed. If this statement is executed while a motion profile is already being processed, then the motion will not be suspended until after the completion of of the current motion profile. If executing a series of segment (MDV) moves, motion will not be suspended until after all the MDV segments have been processed. Any subsequent motion statement will be loaded into the queue and will remain there until the “Motion Resume” statement is executed. Any motion statements executed without the “C” modifier (except MDV statements) while motion is suspended will lock-up the User Program.	
Syntax	MOTION SUSPEND	
Remarks	Performing any MOVEx commands without “C” modifier will lock-up the user program. The programmer will be able to unlock program execution only by performing a Reset or by issuing a Motion Resume command from a host interface.	
See Also	MOVE, MOVEP, MOVEDR, MOVED, MOVEPR ,MDV, MOTION RESUME	

Example:

```
…{statements}
MOTION SUSPEND        ;Motion will be suspended after current motion
                      ;command is finished.
…{statements}
```

Table 48: MOVE

MOVE	Move	Statement
Purpose	There are two variations to the Move command, Move Until and Move While. MOVE UNTIL performs motion until a logical condition becomes TRUE. MOVE WHILE performs motion while a logical condition stays TRUE. The statement suspends the programs execution until the motion is completed, unless the statement is used with C modifier.	
Syntax	MOVE [BACK] UNTIL <condition> [,C] MOVE [BACK] WHILE <condition> [,C]	
	BACK	Changes direction of the move to negative.
	C (optional)	C[ontinue] - modifier allows the program to continue while motion is being performed. If a second motion profile is executed while the first profile is still in motion, the second profile will be loaded into the Motion Stack. The Motion Stack is 32 entries deep. If the queue becomes full, or overflows, then the drive will generate a fault.
	<condition>	The condition to be tested.
See Also	MOVEP, MOVED, MOVEPR, MOVEDR, MDV, MOTION SUSPEND, MOTION RESUME	

Example:

```
{Statements…}
MOVE UNTIL V0<3          ;Move until V0 is less than 3
MOVE BACK UNTIL V0>4     ;Move back until V0 is greater than 4
MOVE WHILE V0<3          ;Move While V0 is less than 3
MOVE BACK WHILE V0>4     ;Move While V0 is greater than 4
MOVE WHILE V0<3,C        ;Move While V0 < 3, continue program execution
```

Table 49: MOVED

MOVED	Move Distance	Statement
Purpose	MOVED performs incremental motion (distance) specified in User Units. This statement will suspend the program's execution until the motion is completed, unless the statement is used with the "C" modifier. If the "S" modifier is used then S-curve acceleration/deceleration is performed during the move.	
Syntax	MOVED <distance>[,S] [,C]	
	C[ontinue]	The "C" argument is an optional modifier which allows the program to continue executing while the motion profile is being executed. If the drive is in the process of executing a previous motion profile the new motion profile will be loaded into the Motion Stack. The Motion Stack is 32 entries deep. If the queue becomes full, or overflows, then the drive will generate a fault.
	S[-curve]	optional modifier specifies S-curve acceleration/deceleration.
Remarks	Maximum variable size is 2^32 * Units/QPPR. This is the max value for Var_APOS_Pulses. Maximum distance is then this maximum value that can be held in a variable divided by the feedback pulses. So assume a 4096 ppr encoder. Post quad = 16384. Max distance before register overflow = 131072. For resolver = 32768. For MoveD, absolute position is not a concern. If overloaded, the register will simply roll over.	
See Also	MOVE, MOVEP, MOVEPR, MOVEDR, MDV, MOTION SUSPEND, MOTION RESUME	

Example:

```
{Statements…}
MOVED 3           ;moves 3 user units forward
MOVED BACK 3      ;moves 3 user units backward
MOVED -3          ;moves 3 user units backward
MOVED V5          ;moves distance / direction determined by value in v5
{Statements…}
```

Table 50: MOVEDR

MOVEDR	Registered Distance Move	Statement
Purpose	MOVEDR performs incremental motion, specified in User Units, in search of the registration input. If during the move the registration input becomes activated (goes high) then the current position is recorded, and the displacement value (the second argument in the MOVEDR statement) is added to the captured registration position to form a new target position. The end of the move is then altered to this new target position. This statement suspends execution of the program until the move is completed, unless the statement is used with the "C" modifier. If the "S" modifier is used then S-curve acceleration/deceleration is performed during the move.	
Syntax	MOVEDR <distance>,<displacement> [,S] [,C]	
	C[ontinue]	The "C" argument is an optional modifier which allows the program to continue executing the User Program while a motion profile is being processed. If a new motion profile is requested while the drive is processing a move the new motion profile will be loaded into the Motion Stack. The Motion Stack is 32 entries deep. If the queue becomes full, or overflows, then the drive will generate a fault.
	S[-curve]	optional modifier specifies S-curve acceleration/deceleration.
See Also	MOVE, MOVEP, MOVEPR, MOVED, MDV, MOTION SUSPEND, MOTION RESUME	
Example:	This example moves the motor 3 user units while checking for the registration input. If registration isn't detected then the move is completed. If registration is detected, the registration position is recorded and the displacement value of 2 is added to the recorded registration position to calculate the new end position.	

```
{Statements…}
MOVEDR 3, 2
{Statements…}
```

Table 51: MOVEP

MOVEP	Move to Position	Statement
Purpose	MOVEP performs motion to a specified absolute position in User Units. This statement will suspend the program's execution until the motion is completed unless the statement is used with the "C" modifier. If the "S" modifier is used then an S-curve acceleration/deceleration is performed during the move.	
Syntax	MOVEP <absolute position>[,S] [,C]	
	C[ontinue]	The "C" argument is an optional modifier which allows the program to continue executing while the motion profile is being executed. If the drive is in the process of executing a previous motion profile the new motion profile will be loaded into the Motion Stack. The Motion Stack is 32 entries deep. If the queue becomes full, or overflows, then the drive will generate a fault.
	S[-curve]	optional modifier specifies S-curve acceleration/deceleration.
Remarks	Maximum variable size is 2^32 * Units/QPPR. This is the max value for Var_APOS_Pulses. Maximum distance is then this maximum value that can be held in a variable divided by the feedback pulses. So assume a 4096 ppr encoder. Post quad = 16384. Max distance before register overflow = 131072. For resolver = 32768. This matters because if the register overflows, then the absolute position is flipped up-side down.	
See Also	MOVE, MOVED, MOVEPR, MOVEDR, MDV, MOTION SUSPEND, MOTION RESUME	

Example:

```
{Statements…}
MOVEP 3          ;moves to 3 user units absolute position
MOVEP -3         ;moves to -3 user units absolute position
MOVEP V5         ;moves to absolute position determined by value in v5
{Statements…}
```

Table 52: MOVEPR

MOVEPR	Registered Distance Move	Statement
Purpose	MOVEPR performs absolute position moves, specified in User Units, in search of the registration input. If during a move the registration input becomes activated (goes high), then the current position is recorded, and the displacement value (the second argument in the MOVEPR statement) is added to the captured registration position to form a new target position. The end of the move is then altered to this new target position. This statement suspends the execution of the program until the move is completed, unless the statement is used with the C modifier. If the "S" modifier is used then S-curve acceleration/deceleration is performed during the move.	
Syntax	MOVEPR <distance>,<displacement> [,S] [,C]	
	C[ontinue]	The "C" argument is an optional modifier which allows the program to continue executing the User Program while a motion profile is being processed. If a new motion profile is requested while the drive is processing a move the new motion profile will be loaded into the Motion Stack. The Motion Stack is 32 entries deep. If the queue becomes full, or overflows, then the drive will generate a fault.
	S[-curve]	optional modifier specifies S-curve acceleration/deceleration.
See Also	MOVE, MOVEP, MOVEDR, MOVED, MDV, MOTION SUSPEND, MOTION RESUME	
Example:	This example moves the motor to the absolute position of 3 user units while checking for the registration input. If registration isn't detected, then the move is completed. If registration is detected, the registration position is recorded and the displacement value of 2 is added to the recorded registration position to calculate the new end position.	

```
{Statements…}
MOVEPR 3, 2
{Statements…}
```

Table 53: ON FAULT/ENDFAULT

ON FAULT/ ENDFAULT	Defines Fault Handler	Statement
Purpose	This statement defines the Fault Handler section of the User Program. The Fault Handler is a section of code which is executed when a fault occurs in the drive. The Fault Handler program must begin with the “ON FAULT” statement and end with the “ENDFAULT” statement. If a Fault Handler routine is not defined, then the User Program will be terminated and the drive disabled upon the drive detecting a fault. Subsequently, if a Fault Handler is defined and a fault is detected, the drive will be disabled, all scanned events will be disabled, and the Fault Handler routine will be executed. The RESUME statement can be used to redirect the program execution from the Fault Handler back to the main program. If this statement is not utilized then the program will terminate once the ENDFAULT statement is executed. The following statements ***can’t*** be used in fault handler: DO / UNTIL, ENABLE, EVENT (ON, OFF), EVENTS (ON, OFF), GOTO, GOSUB, HALT, HOME, JUMP, MDV, MOTION RESUME, MOTION SUSPEND, MOVE, MOVED, MOVEP, MOVEDR, MOVEPR, STOP MOTION (QUICK), VELOCITY ON/OFF, WAIT and WHILE / ENDWHILE	
Syntax	ON FAULT {…statements} ENDFAULT	
See Also	RESUME	

Example:

```
…{statements}              ;User program
FaultRecovery:             ;Recovery procedure
…{statements}

END
ON FAULT                   ;Once fault occurs program is directed here
…{statements}              ;code to deal with fault
RESUME FaultRecovery       ;Execution of RESUME ends Fault Handler and directs
                           ;execution back to User Program.
ENDFAULT                   ;If RESUME is omitted the program will terminate here
                           Fault routine must end with an ENDFAULT statement
```

Table 54: REGISTRATION ON

REGISTRATION ON	Registration On	Statement
Purpose	This statement arms the registration input (input IN_C3). When the registration is armed and the registration input activated the Flag “F_REGISTRATION” is set and the current position is captured and stored to the “RPOS” System Variable. The “REGISTRATION ON” statement, when executed will reset the “F_REGISTRATION” flag ready for detection of the next registration input.	
Syntax	REGISTRATION ON	Flag “F_REGISTRATION” is reset and registration input is armed
See Also	MOVEDR, MOVEPR	

Example:

```
; Moves until registration input is activated and then returns to the sensor position.
…{statements}
REGISTRATION ON             ;Arm registration input
MOVE UNTIL F_REGISTRATION   ;Move until registration flag is activated (triggered by
                            ;registration input to C3), (sensor hit)
                            ;Drive will decelerate to stop beyond Sensor position
MOVEP  RPOS                 ;Absolute move back to the position of the sensor
…{statements}
```

Table 55: RESUME

RESUME	Resume	Statement
Purpose	This statement redirects the code execution form the Fault Handler routine back to in the User Program. The specific line in the User Program to be directed to is called out in the argument <label> of the “RESUME” statement. This statement is only allowed in the fault handler routine.	
Syntax	RESUME <label>	
	<label>	Label in User Program to recommence program execution
See Also	ON FAULT	

Example:

```
;Main Program Section
…{statements}
FaultRecovery:
…{statements}
END

;Fault Handler Section
ON FAULT                ;Once fault occurs program is directed here
…{statements}           ;code to deal with fault
RESUME FaultRecovery    ;Execution of RESUME ends Fault Handler and directs
                        ;execution back the “FaultRecovery” label in the User
                        ;Program.
ENDFAULT                ;If RESUME is omitted the program will terminate here.
                        ;Fault routine must end with a ENDFAULT statement
```

Table 56: RETURN

RETURN	Return from subroutine	Statement
Purpose	This statement will return the code execution back from a subroutine to the point in the main program from where the subroutine was called. If this statement is executed without a previous call to subroutine, (GOSUB), fault #21 “Subroutine stack underflow” will result.	
Syntax	RETURN	
See Also	GOSUB	

Example:

```
;Main Program Section
…{statements}…
GOSUB MySub           ;Program jumps to Subroutine “MySub”
MOVED 10              ;Move to be performed once the Subroutine has executed
…{statements}
END                   ;main program end

;Subroutine Section
MySub:                ;Subroutine called out from User Program
…{statements}         ;Code to be executed in subroutine
RETURN                ;Returns execution to the line of code under the “GOSUB”
                      ;command, (MOVED 10 statement).
```

Table 57: SEND / SEND TO

SEND/SEND TO	Send network variable(s)	Statement
Purpose	This statement is used to share the value of Network Variables between drives on an Ethernet network. Network Variables are variables NV0 through NV31. The variables to be sent out or synchronized between drives, are called out in the “SEND” statement. For example, “SEND [NV5]” will take the current value of variable NV5 in the drive executing the command and load it into the NV5 variable of every other drive on the network. The SENDTO statement only updates network variables of the drives set with the same group ID given in the command.	
Syntax	SEND [NVa,NVb, NVx-NVy],	
	SENDTO GroupID [NVa,NVb, NVx-NVy]	
	a,b,x,y	Any number from 0 to 31
	GroupID	GroupID of the drives whose variables will be affected (synchronized)
See Also		

Example:

```
…{statements}…
NV1=12              ;Set NV1 equal to 12
SEND [NV1]          ;Set the NV1 variable to 12 in every drive in the Network.
SEND [NV5-NV10]     ;Sets the NV5 through NV10 variables in all drives on the Network.
NV20=25             ;Set NV20 equal to 25
SENDTO 2 [NV20]     ;Set the NV20 variable to 25 only in drives with GroupID=2
…{statements}
END                 ;End main program
```

Table 58: STOP MOTION

STOP MOTION [Quick]	Stop Motion	Statement
Purpose	This statement is used to stop all motion. When the “STOP MOTION” statement is executed all motion profiles stored in the Motion Queue are cleared, and motion will immediately be stopped via the deceleration parameter set in the “DECEL” variable. If the “QUICK” modifier is used, then the deceleration value will come from the “QDECEL” variable. The main use for this command is to control an emergency stop or when the End Of Travel sensor is detected. Note that the current position will not be lost after this statement is executed.	
Syntax	STOP MOTION	Stops using DECEL deceleration rate
	STOP MOTION QUICK	Stops using QDECEL deceleration rate
Remarks	Drive output is not disabled following a STOP MOTION [QUICK] command. Also Motion is not suspended after a STOP MOTION [QUICK] so any motion command processed subsequently will be loaded to the Motion Queue and will be executed.	
See Also	MOTION SUSPEND	

Example:

```
…{statements}…
DECEL = 100
QDECEL = 10000
…{statements}
STOP MOTION QUICK
```

Table 59: STOREVARS

STOREVARS	EPM access statements STOREVARS Statement
Purpose	STOREVARS is the command to store the values of the user variables (V0-V31) to the drive's EPM. Using this statement any combinations of variables V0-V31 can be stored to the EPM with a single statement. Stores the values of the user's variables (V0-V31) to the EPM. The purpose of the STOREVARS command is to store user variables from the drives operational memory to the EPM memory so they are retained on power down, or so they can be restored back to the operational memory should their values be altered during the execution of the user program (or by host interface).
Syntax	STOREVARS [Va, Vx-Vy] a,x,y any number from 0 - 31
Remarks	Values that are stored in EPM memory for the User Variables V0-V31 (using StoreVars command) are automatically transferred into operational memory at power up (a LoadVar statement is not required). Should a User Variable be altered by the user program, it is altered only in the operational memory of the drive and can be restored back to its EPM value using the LoadVars statement. Care must be taken with the STOREVAR statement not to exceed EPM write capacity of 1 million cycles.
See Also	LOADVARS

Example:

```
…{statements}…
V1=12                       ;Set V1=12 in drives operational memory (volatile)
…{statements}…
STOREVARS [V1]              ;store V1 variable to EPM Memory (non-volatile)
…{statements}…
LOADVARS [V1]               ;Restore value of V1 from EPM memory
END                         ;End main program
;-------------------------------------------------------
;Example to specify multiple variables list in a single STOREVARS statement

STOREVARS [V0,V1,V5-V20]    ;store values of V0, V1, V5-V20
```

Table 60: VELOCITY ON/OFF

VELOCITY ON/OFF	Velocity Mode Statement
Purpose	The VELOCITY ON statement enables the drive to simulate velocity mode operation while remaining in internal position mode. This allows the drive to transition between internal velocity and position mode while the drive is still enabled. The VELOCITY OFF statement disables velocity mode and returns drive to position mode. The velocity value for this mode is set by writing to the System Variable "VEL". All position related variables are valid in this mode.
Syntax	VELOCITY ON VELOCITY OFF
Remarks	The "VELOCITY ON" statement has to be implemented when the drive is enabled. If the "VELOCITY ON" statement is executed while the drive is disabled, fault # 27 - "Motion Attempted While Drive Disabled" will occur. Execution of any motion related profiles while the drive is in Velocity mode will be loaded into the Motion Queue. When the "VELOCITY OFF" statement is executed the drive will default back to Position mode and any motion commands contained in the Motion Queue will execute in sequence. Please note that the "VEL" variable can be set on the fly, allowing dynamic control of the velocity.
See Also	

Example:

```
VEL=0               ;Set velocity to 0
VELOCITY ON         ;Turn on Velocity Mode
VEL = 10            ;Set velocity to 10
…{statements}
VELOCITY OFF        ;Turn off Velocity Mode
```

Table 61: WAIT

WAIT	Wait	Statement
Purpose	This statement suspends the execution of the program until logical condition or conditions are met. Conditions can include logical expressions, time expiration, or completion of motion commands.	
Syntax	WAIT UNTIL <expression>	wait until expression becomes TRUE
	WAIT WHILE <expression>	wait while expression is TRUE
	WAIT TIME <time delay>	wait until <time delay> in mS is expired
	WAIT MOTION COMPLETE	wait until last motion in Motion Queue completes
	<expression>	Logical expression evaluating to True or False
	<time delay>	time delay expressed in milliseconds
Remarks	Events that have been declared and enabled will continue to process while the main program executes a WAIT statement. Events containing a JUMP statement could interrupt a WAIT statement and cause an immediate jump to another point in the main program.	
See Also		

Example:

```
WAIT UNTIL (APOS>2 && APOS<3)     ;Wait until Apos is > 2 and APOS < 3
WAIT WHILE (APOS <2 && APOS>1)    ;Wait while Apos is <2 and >1
WAIT TIME 1000                    ;Wait 1 Sec (1 Sec=1000mS)
WAIT MOTION COMPLETE              ;Wait until motion is done
```

Table 62: WHILE / ENDWHILE

WHILE/ ENDWHILE	While	Statement
Purpose	The WHILE <expression> executes statement(s) between keywords WHILE and ENDWHILE repeatedly while the expression contained in the WHILE statement evaluates to TRUE.	
Syntax	WHILE <expression> {statement(s)}… ENDWHILE	
Remarks	WHILE block of statements has to end with ENDWHILE keyword.	
See Also	DO/UNTIL	

Example:

```
WHILE  APOS<3         ;Execute the statements while Apos is <3
{statement(s)}..
ENDWHILE
```

3.2 Variable List

Table 63 provides a complete list of the accessible PositionServo variables. These variables can be accessed from the user's program or any supported communications interface protocol. From the user program, any variable can be accessed by either its variable name or by its index value (using the syntax: @<VARINDEX> , where <VARINDEX> is the variable index from Table 63). From the communications interface any variable can be accessed by its index value.

The column "**Type**" indicates the type of variable:

mtr	Motor: denotes a motor value
mtn	Motion: writing to an "mtn" variable could cause the start of motion ⚠
vel	Velocity: denotes a velocity or velocity scaling value

The column "**Format**" provides the native format of the variable:

W	32 bit integer
F	float (real)

When setting a variable via an external device the value can be addressed as floating or integer. The value will automatically adjusted to fit it's given form.

The column "**EPM**" shows if a variable has a non-volatile storage space in the EPM memory:

Y	Variable has non-volatile storage Space in EPM
N	Variable does not exist in EPM memory

The user's program uses a RAM (volatile) 'copy' of the variables stored on the EPM. At power up all RAM copies of the variables are initialized with the EPM values. The EPM's values are not affected by changing the variables in the user's program. When the user's program reads a variable it always reads from the RAM (volatile) copy of the variable. Communications Interface functions can change both the volatile and non-volatile copy of the variable. If the host interface requests a change to the EPM (non-volatile) value, this change is done both in the user program's RAM memory as well as in the EPM. Interface functions have the choice of reading from the RAM (volatile) or from the EPM (non-volatile) copy of the variable. LOADVARDS AND STOREVARD commands can be used to move user variables (V0-V31) between RAM and EPM memory.

The column "**Access**" lists the user's access rights to a variable:

R	read only
W	write only
R/W	read/write

Writing to an R (read-only) variable or reading from a W (write-only) variable is not permitted and many result in erroneous data.

The column "**Units**" shows units of the variable. Units unique to this manual that are used for motion are:

UU	user units
EC	encoder counts
S	seconds
PPS	pulses per sample. Sample time is 512μs - servo loop rate
PPSS	pulses per sample per sample. Sample time is 512μs - servo loop rate

Table 63: PositionServo Variable List

Index	Name	Type	Format	EPM	Access	Description	Units
1	VAR_IDSTRING			N	R	Drive's identification string	
2	VAR_NAME			Y	R/W	Drive's symbolic name	
3	VAR_SERIAL_NUMBER				R	Drive's serial number	
4	VAR_MEM_INDEX				R/W	Position pointer in RAM file (0 - 32767)	
5	VAR_MEM_VALUE				R/W	Value to be read or written to the RAM file	
6	VAR_MEM_INDEX_INCREMENT				R/W	Holds value the MEM_INDEX will increment once the R/W operation is complete	
7	VAR_VELOCITY_ACTUAL		F	N	R	Actual measured motor velocity	UU/sec
8	VAR_RSVD_2						
9	VAR_DFAULT Short Name: DFAULTS				R	Drive faults register. Holds current trip / fault code	
10	VAR_M_ID	mtr		Y	R/W*	Motor ID	
11	VAR_M_MODEL	mtr		Y	R/W*	Motor model	
12	VAR_M_VENDOR	mtr		Y	R/W*	Motor vendor	
13	VAR_M_ESET	mtr		Y	R/W*	Motor Feedback Resolver: 'Positive for CW' 1 - Positive for CW 0 - negative for clockwise	
14	VAR_M_HALLCODE	mtr		Y	R/W*	Hallcode index Range: 0 - 5	
15	VAR_M_HOFFSET	mtr		Y	R/W*	Reserved	
16	VAR_M_ZOFFSET	mtr		Y	R/W*	Resolver Offset Range: 0 - 360	
17	VAR_M_ICTRL	mtr		Y	R/W*	Reserved	
18	VAR_M_JM	mtr		Y	R/W*	Motor moment of inertia, Jm Range: 0 - 0.1	Kgm2
19	VAR_M_KE	mtr		Y	R/W*	Motor voltage or back EMF constant, Ke Range: 1 - 500	V/Krpm
20	VAR_M_KT	mtr		Y	R/W*	Motor torque or force constant, Kt Range: 0.01 - 10	Nm/A
21	VAR_M_LS	mtr		Y	R/W*	Motor phase-to-phase inductance, Lm Range: 0.1 - 500	mH
22	VAR_M_RS	mtr		Y	R/W*	Motor phase-to-phase resistance, Rm Range: 0.01 - 500	[Ohm]
23	VAR_M_MAXCURRENT	mtr		Y	R/W*	Motor's max current(RMS) Range: 0.5 - 50	[A]mp
24	VAR_M_MAXVELOCITY	mtr		Y	R/W*	Motor's max velocity Range: 500 - 20,000	RPM
25	VAR_M_NPOLES	mtr		Y	R/W*	Motor's poles number Rnage: 2 - 200	
26	VAR_M_ENCODER	mtr		Y	R/W*	Encoder resolution Range: 256 - 65536 * 12/Npoles	PPR
27	VAR_M_TERMVOLTAGE	mtr		Y	R/W*	Nominal Motor's terminal voltage Range: 50 - 800	[V]olt
28	VAR_M_FEEDBACK	mtr		Y	R/W*	Feedback type 1 - Encoder 2 - Resolver	

* These are all R/W variables that only become active after variable 247 is set.

Index	Name	Type	Format	EPM	Access	Description	Units
29	VAR_ENABLE_SWITCH_TYPE		W	Y	R/W	Enable switch function 0 - inhibit only 1 - Run	Bit
30	VAR_CURRENTLIMIT		F	Y	R/W	Current limit	[A]mp
31	VAR_PEAKCURRENTLIMIT16		F	Y	R/W	Peak current limit for 16kHz operation	[A]mp
32	VAR_PEAKCURRENTLIMIT		F	Y	R/W	Peak current limit for 8kHz operation	[A]mp
33	VAR_PWMFREQUENCY		W	Y	R/W	PWM frequency selection 0 - 16kHz 1 - 8kHz	
34	VAR_DRIVEMODE		W	Y	R/W	Drive mode 0 - torque 1 - velocity 2 - position ⚠ **WARNING!** You can change operating modes when required during program execution but do **not** change modes on the fly (with drive enabled), as this may cause unexpected behavior of the motor.	
35	VAR_CURRENT_SCALE		F	Y	R/W	Analog input #1 current reference scale Range: model dependent	A/V
36	VAR_VELOCITY_SCALE	vel	F	Y	R/W	Analog input #1 velocity reference scale Range: -10,000 to +10,000	RPM/V
37	VAR_REFERENCE		W	Y	R/W	Reference selection: 1 - internal source 0 - external	
38	VAR_STEPINPUTTYPE		W	Y	R/W	External Position Mode - Input configuration 0 - Quadrature inputs (A/B) 1 - Step & Direction	
39	VAR_MOTORTHERMALPROTECT		W	Y	R/W	Motor thermal protection function: 0 - disabled 1 - enabled	
40	VAR_MOTORPTCRESISTANCE		F	Y	R/W	Motor thermal protection PTC cut-off resistance in Ohms	[Ohm]
41	VAR_SECONDENCODER		W	Y	R/W	Second encoder: 0 - Disabled 1 - Enabled	
42	VAR_REGENDUTY		W	Y	R/W	Regen circuit PWM duty cycle in % Range: 1-100%	%
43	VAR_ENCODERREPEATSRC		W	Y	R/W	Selects source for repeat buffers: 0 - Model 940 - Encoder Port P4 0 - Model 941 - 2nd Encoder Option Bay 1 - Model 940 - 2nd Encoder Option Bay 1 - Model 941 - Resolver Port P4	
44	VAR_VP_GAIN Short Name: VGAIN_P	vel	W	Y	R/W	Velocity loop Proportional gain Range: 0 - 32767	
45	VAR_VI_GAIN Short Name: VGAIN_I	vel	W	Y	R/W	Velocity loop Integral gain Range: 0 - 32767	
46	VAR_PP_GAIN Short Name: PGAIN P		W	Y	R/W	Position loop Proportional gain Range: 0 - 32767	
47	VAR_PI_GAIN Short Name: PGAIN I		W	Y	R/W	Position loop Integral gain Range: 0 - 16383	
48	VAR_PD_GAIN Short Name: PGAIN D		W	Y	R/W	Position loop Differential gain Range: 0 - 32767	
49	VAR_PI_LIMIT Short Name: PGAIN ILIM		W	Y	R/W	Position loop integral gain limit Range: 0 - 20000	

Index	Name	Type	Format	EPM	Access	Description	Units
50	VAR_SEI_GAIN					Not used	
51	VAR_VREG_WINDOW	vel	W	Y	R/W	Gains scaling coefficient Range: -16 to +4	
52	VAR_ENABLE		W	N	W	Software Enable/Disable 0 - disable 1 - enable	
53	VAR_RESET		W	N	W	Drive reset. Disables drive, halts program execution, reset active fault 0 - no action 1 - reset drive	
54	VAR_STATUS Short Name: DSTATUS		W	N	R	Drive's status register	
55	VAR_BCF_SIZE		W	Y	R	User's program Byte-code size	Bytes
56	VAR_AUTOBOOT		W	Y	R/W	User's program autostart flag. 0 - program has to be started manually (MotionView or interface) 1 - program started automatically after drive power up	
57	VAR_GROUPID		W	Y	R/W	Network group ID Range: 1 - 32767	
58	VAR_VLIMIT_ZEROSPEED		F	Y	R/W	Zero Speed window Range: 0 - 100	Rpm
59	VAR_VLIMIT_SPEEDWND		F	Y	R/W	At Speed window Range: 10 - 10000	Rpm
60	VAR_VLIMIT_ATSPEED		F	Y	R/W	Target Velocity for At Speed window Range: -10000 - +10000	Rpm
61	VAR_PLIMIT_POSERROR		W	Y	R/W	Position error Range: 1 - 32767	EC
62	VAR_PLIMIT_ERRORTIME		F	Y	R/W	Position error time (time which position error has to remain to set-off position error fault) Range: 0.25 - 8000	mS
63	VAR_PLIMIT_SEPOSERROR		W	Y	R/W	Second encoder Position error Range: 1 - 32767	EC
64	VAR_PLIMIT_SEERRORTIME		F	Y	R/W	Second encoder Position error time (time which position error has to remain to set-off position error fault) Range: 0.25 - 8000	mS
65	VAR_INPUTS Short Name: INPUTS		W	N	R	Digital inputs status variable. A1 occupies Bit 0, A2- Bit 1 … C4 - bit 11.	
66	VAR_OUTPUT Short Name: OUTPUTS		W	N	R/W	Digital outputs status variable. Writing to this variable sets/resets digital outputs, except outputs which have been assigned special function. Output 1 Bit0 Output 2 Bit 1 Output 3 Bit 2 Output 4 Bit 3	
67	VAR_IP_ADDRESS		W	Y	R/W	Ethernet IP address. IP address changes at next boot up. 32 bit value	
68	VAR_IP_MASK		W	Y	R/W	Ethernet IP NetMask. Mask changes at next boot up. 32 bit value	
69	VAR_IP_GATEWAY		W	Y	R/W	Ethernet Gateway IP address. Address changes at next boot up. 32 bit value	

Index	Name	Type	Format	EPM	Access	Description	Units
70	VAR_IP_DHCP		W	Y	R/W	Use DHCP 0-manual 1- use DHCP service	
71	VAR_AIN1 Short Name: AIN1		F	N	R	Analog Input AIN1 current value	[V]olt
72	VAR_AIN2 Short Name: AIN2		F	N	R	Analog Input AIN2 current value	[V]olt
73	VAR_BUSVOLTAGE		F	N	R	Bus voltage current value	[V]olt
74	VAR_HTEMP		F	N	R	Heatsink temperature Returns: 0 - for temperatures < 40C and actual heat sink temperature for temperatures >40 C	[c]
75	VAR_ENABLE_ACCELDECEL	vel		Y	R/W	Enable Accel/Decel function for velocity mode 0 - disable 1 - enable	
76	VAR_ACCEL_LIMIT System variable for ramp parameters in MotionView	vel	F	Y	R/W	Accel value for velocity mode Range: 0.1 - 5000000	Rpm*Sec
77	VAR_DECEL_LIMIT System variable for ramp parameters in MotionView	vel	F	Y	R/W	Decel value for velocity mode Range: 0.1 - 5000000	Rpm*Sec
78	VAR_FAULT_RESET		W	Y	R/W	Fault Reset configuration: 1 - on deactivation of Enable/Inhibit input A3 0 - on activation of Enable/Inhibit input (A3)	
79	VAR_M2SRATIO_MASTER		W	Y	R/W	Master to system ratio. Master counts range: -32767 - +32767 Value will be applied upon write to PID #80. Write to this PID followed by writing to PID#80 to apply new ratio pair	
80	VAR_M2SRATIO_SYSTEM		W	Y	R/W	Master to system ratio. System counts range: 1 - 32767 Writing to this PID also applies value currently held in PID #79. If you need to change both values Set #79 first then write to this PID desired value to apply new ratio.	
81	VAR_S2PRATIO_SECOND		W	Y	R/W	Secondary encoder to prime encoder ratio. Second counts range: -32767 - +32767 Value will be applied upon write to PID #82. Write to this PID followed by writing to PID#82 to apply new ratio pair	
82	VAR_S2PRATIO_PRIME		W	Y	R/W	Secondary encoder to prime encoder ratio. Prime counts range: 1 - 32767 Writing to this PID also applies value currently held in PID #81. If you need to change both values Set #81 first then write to this PID desired value to apply new ratio.	
83	VAR_EXSTATUS Short Name: DEXSTATUS		W	N	R	Extended status. Lower word copy of DSP status flags.	
84	VAR_HLS_MODE		W	Y	R/W	Hardware limit switches. 0 - not used 1 - stop and fault 2 - fault **NOTE:** When the Hard Limit Switches are activated, the drive will remember this state until the drive is disabled or a fault occurs.	

Index	Name	Type	Format	EPM	Access	Description	Units
85	VAR_AOUT_FUNCTION		W	Y	R/W	Analog output function range: 0 - 8 0 - Not assigned 1 - Phase Current (RMS) 2 - Phase Current (Peak Value) 3 - Motor Velocity 4 - Phase Current R 5 - Phase Current S 6 - Phase Current T 7 - Iq current 8 - Id current	
86	VAR_AOUT_VELSCALE		F	Y	R/W	Analog output scale for velocity quantities. Range: 0 - 10	mV/Rpm
87	VAR_AOUT_CURSCALE		F	Y	R/W	Analog output scale for current related quantities. Range: 0 - 10	V/A
88	VAR_AOUT Short Name: AOUT		F	N	W	Analog output value.(Used if VAR #85 is set to 0 - no function) Range: 0 - 10	V
89	VAR_AIN1_DEADBAND		F	Y	R/W	Analog input #1 dead-band. Applied when used as current or velocity reference. Range: 0 - 100	mV
90	VAR_AIN1_OFFSET			Y	R/W	Analog input #1 offset. Applied when used as current/velocity reference Range: -10,000 to +10,000	mV
91	VAR_SUSPEND_MOTION		W	N	R/W	Suspend motion. Suspends motion produced by trajectory generator. Current move will be completed before motion is suspended. 0 - motion suspended 1 - motion resumed	
92	VAR_MOVEP	⚠ mtn	W	N	W	Target position for absolute move. Writing value executes Move to position as per MOVEP statement using current values of acceleration, deceleration and max velocity.	UU
93	VAR_MOVED	⚠ mtn	W	N	W	Incremental position. Writing value executes Incremental move as per MOVED statement using current values of acceleration, deceleration and max velocity.	UU
94	VAR_MDV_DISTANCE		F	N	W	Distance for MDV move	UU
95	VAR_MDV_VELOCITY	⚠ mtn	F	N	W	Velocity for MDV move. Writing to this variable executes MDV move with Distance value last written to variable #94	UU
96	VAR_MOVE_PWI1	⚠ mtn	W	N	W	Writing value executes Move in positive direction while input true (active). Value specifies input # 0 - 3 : A1 -A4 4 - 7 : B1 - B4 8 - 11 : C1 - C4	
97	VAR_MOVE_PWI0	⚠ mtn	W	N	W	Writing value executes Move in positive direction while input false (not active). Value specifies input # 0 - 3 : A1 -A4 4 - 7 : B1 - B4 8 - 11 : C1 - C4	

Index	Name	Type	Format	EPM	Access	Description	Units
98	VAR_MOVE_NWI1	⚠ mtn	F	N	W	Writing value executes Move negative direction while input true (active). Value specifies input # 0 - 3 : A1 -A4 4 - 7 : B1 - B4 8 - 11 : C1 - C4	
99	VAR_MOVE_NWI0	⚠ mtn	F	N	W	Writing value executes Move negative direction while input false (not active). Value specifies input # 0 - 3 : A1 -A4 4 - 7 : B1 - B4 8 - 11 : C1 - C4	
100	VAR_V0 Short Name: V0		F	Y	R/W	User variable General purpose user defined variable	
101	VAR_V1 Short Name: V1		F	Y	R/W	User variable General purpose user defined variable	
102	VAR_V2 Short Name: V2		F	Y	R/W	User variable General purpose user defined variable	
103	VAR_V3 Short Name: V3		F	Y	R/W	User variable General purpose user defined variable	
104	VAR_V4 Short Name: V4		F	Y	R/W	User variable General purpose user defined variable	
105	VAR_V5 Short Name: V5		F	Y	R/W	User variable General purpose user defined variable	
106	VAR_V6 Short Name: V6		F	Y	R/W	User variable General purpose user defined variable	
107	VAR_V7 Short Name: V7		F	Y	R/W	User variable General purpose user defined variable	
108	VAR_V8 Short Name: V8		F	Y	R/W	User variable General purpose user defined variable	
109	VAR_V9 Short Name: V9		F	Y	R/W	User variable General purpose user defined variable	
110	VAR_V10 Short Name: V10		F	Y	R/W	User variable General purpose user defined variable	
111	VAR_V11 Short Name: V11		F	Y	R/W	User variable General purpose user defined variable	
112	VAR_V12 Short Name: V12		F	Y	R/W	User variable General purpose user defined variable	
113	VAR_V13 Short Name: V13		F	Y	R/W	User variable General purpose user defined variable	
114	VAR_V14 Short Name: V14		F	Y	R/W	User variable General purpose user defined variable	
115	VAR_V15 Short Name: V15		F	Y	R/W	User variable General purpose user defined variable	
116	VAR_V16 Short Name: V16		F	Y	R/W	User variable General purpose user defined variable	
117	VAR_V17 Short Name: V17		F	Y	R/W	User variable General purpose user defined variable	
118	VAR_V18 Short Name: V18		F	Y	R/W	User variable General purpose user defined variable	
119	VAR_V19 Short Name: V19		F	Y	R/W	User variable General purpose user defined variable	
120	VAR_V20 Short Name: V20		F	Y	R/W	User variable General purpose user defined variable	

Index	Name	Type	Format	EPM	Access	Description	Units
121	VAR_V21 Short Name: V21		F	Y	R/W	User variable General purpose user defined variable	
122	VAR_V22 Short Name: V22		F	Y	R/W	User variable General purpose user defined variable	
123	VAR_V23 Short Name: V23		F	Y	R/W	User variable General purpose user defined variable	
124	VAR_V24 Short Name: V24		F	Y	R/W	User variable General purpose user defined variable	
125	VAR_V25 Short Name: V25		F	Y	R/W	User variable General purpose user defined variable	
126	VAR_V26 Short Name: V26		F	Y	R/W	User variable General purpose user defined variable	
127	VAR_V27 Short Name: V27		F	Y	R/W	User variable General purpose user defined variable	
128	VAR_V28 Short Name: V28		F	Y	R/W	User variable General purpose user defined variable	
129	VAR_V29 Short Name: V29		F	Y	R/W	User variable General purpose user defined variable	
130	VAR_V30 Short Name: V30		F	Y	R/W	User variable General purpose user defined variable	
131	VAR_V31 Short Name: V31		F	Y	R/W	User variable General purpose user defined variable	
132	VAR_MOVEDR_DISTANCE		F	N	W	Registered move distance. Incremental motion as per MOVEDR statement	UU
133	VAR_MOVEDR_DISPLACEMENT	⚠ mtn	F	N	W	Registered move displacement Writing to this variable executes the move MOVEDR using value set by #132	UU
134	VAR_MOVEPR_DISTANCE		F	N	W	Registered move distance. Absolute motion as per MOVEPR statement	UU
135	VAR_MOVEPR_DISPLACEMENT	⚠ mtn	F	N	W	Registered move displacement Writing to this variable makes the move MOVEPR using value set by #134	UU
136	VAR_STOP_MOTION		W	N	W	Stops motion: 1 - stops motion 0 - no action	
137	VAR_START_PROGRAM		W	N	W	Starts user program 1 - starts program 0 - no action	
138	VAR_VEL_MODE_ON		W	N	W	Turns on Profile Velocity for Internal Position Mode. (Acts as statement VELOCITY ON) 0 - normal operation 1 - velocity mode on	
139	VAR_IREF Short Name: IREF		F	N	W	Internal Reference: Torque or Velocity mode Set value in Amps for Torque mode Set Value in RPM for Velocity Mode	RPS Amps
140	VAR_NV0 Short Name: NV0		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
141	VAR_NV1 Short Name: NV1		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
142	VAR_NV2 Short Name: NV2		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
143	VAR_NV3 Short Name: NV3		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	

Index	Name	Type	Format	EPM	Access	Description	Units
144	VAR_NV4 Short Name: NV4		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
145	VAR_NV5 Short Name: NV5		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
146	VAR_NV6 Short Name: NV6		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
147	VAR_NV7 Short Name: NV7		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
148	VAR_NV8 Short Name: NV8		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
149	VAR_NV9 Short Name: NV9		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
150	VAR_NV10 Short Name: NV10		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
151	VAR_NV11 Short Name: NV11		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
152	VAR_NV12 Short Name: NV12		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
153	VAR_NV13 Short Name: NV13		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
154	VAR_NV14 Short Name: NV14		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
155	VAR_NV15 Short Name: NV15		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
156	VAR_NV16 Short Name: NV16		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
157	VAR_NV17 Short Name: NV17		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
158	VAR_NV18 Short Name: NV18		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
159	VAR_NV19 Short Name: NV19		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
160	VAR_NV20 Short Name: NV20		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
161	VAR_NV21 Short Name: NV21		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
162	VAR_NV22 Short Name: NV22		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
163	VAR_NV23 Short Name: NV23		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
164	VAR_NV24 Short Name: NV24		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
165	VAR_NV25 Short Name: NV25		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
166	VAR_NV26 Short Name: NV26		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
167	VAR_NV27 Short Name: NV27		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
168	VAR_NV28 Short Name: NV28		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
169	VAR_NV29 Short Name: NV29		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	

Index	Name	Type	Format	EPM	Access	Description	Units
170	VAR_NV30 Short Name: NV30		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
171	VAR_NV31 Short Name: NV31		F	N	R/W	User defined Network variable. Variable can be shared across Ethernet network.	
172	VAR_SERIAL_ADDRESS		W	Y	R/W	RS485 drive ID. Range: 0 - 254	
173	VAR_MODBUS_BAUDRATE		W	Y	R/W	Baud rate for ModBus operations: 2 - 9600 3 - 19200 4 - 38400 5 - 57600 6 - 115200	
174	VAR_MODBUS_DELAY		W	Y	R/W	ModBus reply delay in mS Range: 0 - 1000	mS
175	VAR_RS485_CONFIG		W	Y	R/W	Rs485 configuration: 0 - normal IP over PPP 1 - ModBus	
176	VAR_PPP_BAUDRATE **NOTE:** Does NOT apply to MVOB.		W	Y	R/W	RS232/485 (normal mode) baud rate: 2 - 9600 3 - 19200 4 - 38400 5 - 57600 6 - 115200	
177	VAR_MOVEPS	⚠ mtn	F	N	W	Same as variable #92 but using S-curve acceleration/deceleration	
178	VAR_MOVEDS	⚠ mtn	F	N	W	Same as variable #93 but using S-curve acceleration/deceleration	
179	VAR_MDVS_VELOCITY	⚠ mtn		N	W	Velocity for MDV move using S-curve accel/deceleration. Writing to this variable executes MDV move with Distance value last written to variable #94 (unless motion is suspended by #91).	UU
180	VAR_MAXVEL Short Name: MAXV		F	N	R/W	Max velocity for motion profile	UU/S
181	VAR_ACCEL Short Name: ACCEL		F	N	R/W	Accel value for indexing	UU/S^2
182	VAR_DECEL Short Name: DECEL		F	N	R/W	Decel value for indexing	UU/S^2
183	VAR_QDECEL Short Name: QDECEL		F	N	R/W	Quick decel value	UU/S^2
184	VAR_INPOSLIM Short Name: INPOSLIM		W	N	R/W	Sets window for “In Position” Limits	UU
185	VAR_VEL Short Name: VEL		F	N	R/W	Velocity reference for “Profiled” velocity	UU/S
186	VAR_UNITS Short Name: UNITS		F	Y	R/W	User units	
187	VAR_MECOUNTER Short Name: MECOUNTER		W	N	R/W	A/B inputs reference counter value	Count
188	VAR_PHCUR Short Name: PHCUR		F	N	R	Phase current	A
189	VAR_POS_PULSES Short Name: TPOS PLS		W	N	R/W	Target position in encoder pulses	EC
190	VAR_APOS_PULSES Short Name: APOS PLS		W	N	R/W	Actual position in encoder pulses	EC

Index	Name	Type	Format	EPM	Access	Description	Units
191	VAR_POSERROR_PULSES Short Name: PERROR_PLS		W	N	R	Position error in encoder pulses	EC
192	VAR_CURRENT_VEL_PPS		F	N	R	Set-point (target) velocity in PPS	PPS
193	VAR_CURRENT_ACCEL_PPSS		F	N	R	Set-point (target) acceleration (demanded value) value	PPSS
194	VAR_IN0_DEBOUNCE		W	Y	R/W	Input A1 de-bounce time in mS Range: 0 - 1000	mS
195	VAR_IN1_DEBOUNCE		W	Y	R/W	Input A2 de-bounce time in mS Range: 0 - 1000	mS
196	VAR_IN2_DEBOUNCE		W	Y	R/W	Input A3 de-bounce time in mS Range: 0 - 1000	mS
197	VAR_IN3_DEBOUNCE		W	Y	R/W	Input A4 de-bounce time in mS Range: 0 - 1000	mS
198	VAR_IN4_DEBOUNCE		W	Y	R/W	Input B1 de-bounce time in mS Range: 0 - 1000	mS
199	VAR_IN5_DEBOUNCE		W	Y	R/W	Input B2 de-bounce time in mS Range: 0 - 1000	mS
200	VAR_IN6_DEBOUNCE		W	Y	R/W	Input B3 de-bounce time in mS Range: 0 - 1000	mS
201	VAR_IN7_DEBOUNCE		W	Y	R/W	Input B4 de-bounce time in mS Range: 0 - 1000	mS
202	VAR_IN8_DEBOUNCE		W	Y	R/W	Input C1 de-bounce time in mS Range: 0 - 1000	mS
203	VAR_IN9_DEBOUNCE		W	Y	R/W	Input C2 de-bounce time in mS Range: 0 - 1000	mS
204	VAR_IN10_DEBOUNCE		W	Y	R/W	Input C3 de-bounce time in mS Range: 0 - 1000	mS
205	VAR_IN11_DEBOUNCE		W	Y	R/W	Input C4 de-bounce time in mS Range: 0 - 1000	mS
206	VAR_OUT1_FUNCTION		W	Y	R/W	Programmable Output function 0 - Not Assigned 1 - Zero Speed 2 - In Speed Window 3 - Current Limit 4 - Run time fault 5 - Ready 6 - Brake 7 - In position	
207	VAR_OUT2_FUNCTION		W	Y	R/W	Programmable Output Function. See range (settings) for Variable #206	
208	VAR_OUT3_FUNCTION		W	Y	R/W	Programmable Output Function. See range (settings) for Variable #206	
209	VAR_OUT4_FUNCTION		W	Y	R/W	Programmable Output Function. See range (settings) for Variable #206	
210	VAR_HALLCODE		W	N	R	Current hall code Bit 0 - Hall 1 Bit 1 - Hall 2 Bit 2 - Hall 3	
211	VAR_ENCODER		W	N	R	Primary encoder current value	EC
212	VAR_RPOS_PULSES Short Name: RPOS PLS		W	N	R	Registration position in encoder pulses	EC
213	VAR_RPOS Short Name: RPOS		F	N	R	Registration position	UU

Index	Name	Type	Format	EPM	Access	Description	Units
214	VAR_POS Short Name: TPOS		F	N	R/W	Target position	UU
215	VAR_APOS Short Name: APOS		F	N	R/W	Actual position	UU
216	VAR_POSERROR Short Name: PERROR		W	N	R	Position error	EC
217	VAR_CURRENT_VEL Short Name: TV		F	N	R	Set-point (target) velocity (demanded value)	UU/S
218	VAR_CURRENT_ACCEL Short Name: TA		F	N	R	Set-point (target) acceleration (demanded value)	UU/S^2
219	VAR_TPOS_ADVANCE Short Name: TPOS_ADV		W	N	W	Target position advance. Every write to this variable adds value to the Target position summing point. Value gets added once per write. This variable useful when loop is driven by Master encoder signals and trying to correct phase. Value is in encoder counts	EC
220	VAR_IOINDEX Short Name: INDEX		W	N	R/W	Same as INDEX variable in user's program. See "INDEX" in Language Reference section of this manual.	
221	VAR_PSLIMIT_PULSES		W	Y	R/W	Positive Software limit switch value in Encoder counts	EC
222	VAR_NSLIMIT_PULSES		W	Y	R/W	Negative Software limit switch value in Encoder counts	EC
223	VAR_ SLS_MODE		W	Y	R/W	Soft limit switch action code: 0 - no action 1- Fault. 2- Stop and fault (When loop is driven by trajectory generator only. With all the other sources same action as 1) --	
224	VAR_PSLIMIT		F	Y	R/W	Same as var 221 but value in User Units	UU
225	VAR_NSLIMIT		F	Y	R/W	Same as var 222 but value in User Units	UU
226	VAR_SE_APOS_PULSES		W	N	R	2nd encoder actual position in encoder counts	EC
227	VAR SE POSERROR PULSES		W	N	R	2nd encoder position error in encoder counts	EC
228	VAR_MODBUS_PARITY		W	Y	R/W	Parity for Modbus Control: 0 - No Parity 1 - Odd Parity 2 - Even Parity	
229	VAR_MODBUS_STOPBITS		W	Y	R/W	Number of Stopbits for Modbus Control: 0 - 1.0 1 - 1.5 2 - 2.0	
230	VAR_M_NOMINALVEL		F	Y	R/W	Induction Motor Nominal Velocity Range: 500 - 20000 RPM	RPM
231	VAR_M_COSPHI		F	Y	R/W	Induction Motor Cosine Phi Range: 0 - 1.0	
232	VAR_M_BASEFREQUENCY		F	Y	R/W	Induction Motor Base Frequency: Range: 0 - 400Hz	Hz
233	VAR_M_SERIES					Induction Motor Series	

Index	Name	Type	Format	EPM	Access	Description	Units
234	VAR_CAN_BAUD_EPM		W	Y	R/W	CAN Bus Parameter: Baud Rate: 1 - 8 1 - 10k 2 - 20k 3 - 50k 4 - 125k 5 - 250k 6 - 500k 7 - 800k 8 - 1000k	
235	VAR_CAN_ADDR_EPM		W	Y	R/W	CAN Bus Parameter: Address: 1-127	
236	VAR_CAN_OPERMODE_EPM		W	Y	R/W	CAN Bus Parameter: Boot-up Mode: 0 - 2 (Operational State Control) 0 - enters into pre-operational state 1 - enters into operational state 2 - pseudo NMT: sends NMT Start Node command after delay (set by variable 237)	
237	VAR_CAN_OPERDELAY_EPM		W	Y	R/W	CAN Bus Parameter: pseudo NMT mode delay time in seconds (refer to variable 236)	sec
238	VAR_CAN_ENABLE_EPM		W	Y	R/W	CAN Bus Parameter: Mode Control: 0, 1, 2 0 - Disable CAN interface 1 - Enable CAN interface in DS301 mode Concurrent user's program execution possible 2 - Enable CAN interface in DS402 mode Concurrent user's program execution possible 3 - Enable DeviceNet 4 - Enable PROFIBUS DP	
239	VAR_HOME_ACCEL		F	Y		Homing Mode: ACCEL rate: 0 - 10000000.0	UU/sec^2
240	VAR_HOME_OFFSET		F	Y	R/W	Homing Mode: Home Position Offset Range: -32767 to +32767	UU
241	VAR_HOME_OFFSET_PULSES		W	Y	R/W	Homing Mode: Home Position Offset in encoder counts Range: +/- 2,147,418,112	EC
242	VAR_HOME_FAST_VEL		F	Y	R/W	Homing Mode: Fast Velocity Range: -10,000 to +10,000	UU/sec
243	VAR_HOME_SLOW_VEL		F	Y	R/W	Homing Mode: Slow Velocity Range: -10,000 to +10,000	UU/sec
244	VAR_HOME_METHOD		W	Y	R/W	Homing Mode: Homing Method Range: 1 - 35	
245	VAR_START_HOMING Short Name: HOME		W	N	W	Homing Mode: Start Homing: 0, 1 0 - No action 1 - Start Homing	
246	VAR_HOME_SWITCH_INPUT		W	Y	R/W	Homing Mode: Switch Input Assignment: Range: 0 - 11 0 - 3: A1 - A4 4 - 7: B1 - B4 8 - 11: C1 - C4 Warning: If using A1, A2 or C3 refer to the homing section. Do not use input A3 as homing switch.	

Index	Name	Type	Format	EPM	Access	Description	Units
247	VAR_M_VALIDATE_MOTOR		W	N	W	Initiate / accept drive motor parameters entered in motor data PIDs. Motor parameters are variables whose identifier starts with VAR_M_xxxxxx 0 - No Action 1 - Validate Motor Data	
248	VAR_M_I2T		F	Y	R/W	Not used	
249	VAR_M_EABSOLUTE		F	Y	R/W	Indicates type of ABS encoder for models with ABS encoder support. Otherwise currently not active.	
250	VAR_M_ABSWAP		F	Y	R/W	Motor Encoder Feedback: B leads A 0 - No Action 1 - B leads A for forward checked (active)	
251	VAR_M_HALLS_INVERTED		F	Y	R/W	Motor Encoder Feedback: Halls 0 - No Action 1 - Inverted Halls Box checked (active)	
252	RESERVED					Do NOT use	
253	RESERVED					Do NOT use	
254	RESERVED					Do NOT use	
255	RESERVED					Do NOT use	
256	RESERVED					Do NOT use	
257	RESERVED					Do NOT use	
258	RESERVED					Do NOT use	
259	RESOLVER_EMU_TRK		W	Y	R/W	Resolver Emulation Track Number Range: 0 - 15 0 - 1024 1 - 256 2 - 360 3 - 400 4 - 500 5 - 512 6 - 720 7 - 800 8 - 1000 9 - 1024 10 - 2000 11 - 2048 12 - 2500 13 - 2880 14 - 250 15 - 4096	
260	RESERVED						
261	VAR_CIP_LINK_A_IN_CTRL		W	Y	R/W	Datalink “A” for input assembly (Refer to Ethernet/IP manual for details)	
262	VAR_CIP_LINK_B_IN_CTRL		W	Y	R/W	Datalink “B” for input assembly (Refer to Ethernet/IP manual for details)	
263	VAR_CIP_LINK_C_IN_CTRL		W	Y	R/W	Datalink “C” for input assembly (Refer to Ethernet/IP manual for details)	
264	VAR_CIP_LINK_D_IN_CTRL		W	Y	R/W	Datalink “D” for input assembly (Refer to Ethernet/IP manual for details)	
265	VAR_CIP_LINK_A_OUT_CTRL		W	Y	R/W	Datalink “A” for output assembly (Refer to Ethernet/IP manual for details)	

Index	Name	Type	Format	EPM	Access	Description	Units
266	VAR_CIP_LINK_B_OUT_CTRL		W	Y	R/W	Datalink “B” for output assembly (Refer to Ethernet/IP manual for details)	
267	VAR_CIP_LINK_C_OUT_CTRL		W	Y	R/W	Datalink “C” for output assembly (Refer to Ethernet/IP manual for details)	
268	VAR_CIP_LINK_D_OUT_CTRL		W	Y	R/W	Datalink “D” for output assembly (Refer to Ethernet/IP manual for details)	
269	VAR_CIP_DAT_REG_CTRL		W	Y	R/W	Data format control for Ethernet/IP assemblies (Refer to Ethernet/IP manual for details)	
270	VAR_CIP_CTRL_REG		W	Y	R/W	Control register for control via Ethernet/IP (Refer to Ethernet/IP manual for details)	
271	VAR_CIP_STATUS_REG		W	N	R	Status register 2 (Fromat for Ethernet/IP) (Refer to Ethernet/IP manual for details)	
272	VAR_CIP_HEART_BEAT		W	Y	R/W	CIP Heart beat timer (Ethernet/IP)	
273	VAR_EIP_MCACT_TTL		W	Y	R/W	Ethernet/IP multicast “time to leave” parameter	
274	VAR_EIP_MCAST_CTRL		W	Y	R/W	Multicast enable/disable control register (Ethernet/IP)	
275	EIP_MCAST_ADDRESS		W	Y	R/W	Multicast address (Default = 239,192,15,32)	
276	DNET_SCALE_POLL_IO		W	Y	R/W	DeviceNet polled IO data scale factor (Refer to DeviceNet manual for details)	
277	TCP_REPLY_DELAY		W	Y	R/W	TCP reply delay value	
278	RESERVED					Do NOT use	
279	RESERVED					Do NOT use	
280	RESERVED					Do NOT use	
281	RESERVED					Do NOT use	
282	RESERVED					Do NOT use	

Index	Name	Type	Format	EPM	Access	Description	Units
i	**NOTE**: PIDs 283 - 309 are for REFERENCE ONLY. These variables are set through MotionView. Do **NOT** use directly.						
283	PBUS_ADDR		W	Y	R/W	Profibus address	
284	PBUS_DOUT_SIZE		W	Y	R/W	Number of Profibus Data Out channels Range: 0 - 12	
285	PBUS_DIN_SIZE		W	Y	R/W	Number of Profibus Data In channels Range: 0 - 12	
286	PBUS_OUT_LINK1		W	Y	R/W	Profibus Data Out, Channel link 1 PID map	
287	PBUS_OUT_LINK2		W	Y	R/W	Profibus Data Out, Channel link 2 PID map	
288	PBUS_OUT_LINK3		W	Y	R/W	Profibus Data Out, Channel link 3 PID map	
289	PBUS_OUT_LINK4		W	Y	R/W	Profibus Data Out, Channel link 4 PID map	
290	PBUS_OUT_LINK5		W	Y	R/W	Profibus Data Out, Channel link 5 PID map	
291	PBUS_OUT_LINK6		W	Y	R/W	Profibus Data Out, Channel link 6 PID map	
292	PBUS_OUT_LINK7		W	Y	R/W	Profibus Data Out, Channel link 7 PID map	
293	PBUS_OUT_LINK8		W	Y	R/W	Profibus Data Out, Channel link 8 PID map	
294	PBUS_OUT_LINK9		W	Y	R/W	Profibus Data Out, Channel link 9 PID map	
295	PBUS_OUT_LINK10		W	Y	R/W	Profibus Data Out, Channel link 10 PID map	
296	PBUS_OUT_LINK11		W	Y	R/W	Profibus Data Out, Channel link 11 PID map	
297	PBUS_OUT_LINK12		W	Y	R/W	Profibus Data Out, Channel link 12 PID map	
298	PBUS_IN_LINK1		W	Y	R/W	Profibus Data In, Channel link 1 PID map	
299	PBUS_IN_LINK2		W	Y	R/W	Profibus Data In, Channel link 2 PID map	
300	PBUS_IN_LINK3		W	Y	R/W	Profibus Data In, Channel link 3 PID map	
301	PBUS_IN_LINK4		W	Y	R/W	Profibus Data In, Channel link 4 PID map	
302	PBUS_IN_LINK5		W	Y	R/W	Profibus Data In, Channel link 5 PID map	
303	PBUS_IN_LINK6		W	Y	R/W	Profibus Data In, Channel link 6 PID map	
304	PBUS_IN_LINK7		W	Y	R/W	Profibus Data In, Channel link 7 PID map	
305	PBUS_IN_LINK8		W	Y	R/W	Profibus Data In, Channel link 8 PID map	
306	PBUS_IN_LINK9		W	Y	R/W	Profibus Data In, Channel link 9 PID map	
307	PBUS_IN_LINK10		W	Y	R/W	Profibus Data In, Channel link 10 PID map	
308	PBUS_IN_LINK11		W	Y	R/W	Profibus Data In, Channel link 11 PID map	
309	PBUS_IN_LINK12		W	Y	R/W	Profibus Data In, Channel link 12 PID map	
310	PBUS_ACYC_MODE		W	Y	R/W	Profibus Acyclic Mode Type Refer to Profibus Manual (P94PFB01)	
i	**NOTE**: PIDs 311 - 406 are for REFERENCE ONLY. These variables are set through MotionView. Do **NOT** use directly These variables are used by MotionView for non-volatile settings of CAN TPDO/RPDO.						
311	VAR_RPDO_1_COM					Receive PDO	
312	VAR_RPDO_2_COM						
313	VAR_RPDO_3_COM						
314	VAR_RPDO_4_COM						

Index	Name	Type	Format	EPM	Access	Description	Units
315	VAR_RPDO_5_COM						
316	VAR_RPDO_6_COM						
317	VAR_RPDO_7_COM						
318	VAR_RPDO_8_COM						
319	VAR_RPDO_1_MAP1					RPDO Mapping	
320	VAR_RPDO_1_MAP2						
321	VAR_RPDO_1_MAP3						
322	VAR_RPDO_1_MAP4						
323	VAR_RPDO_2_MAP1						
324	VAR_RPDO_2_MAP2						
325	VAR_RPDO_2_MAP3						
326	VAR_RPDO_2_MAP4						
327	VAR_RPDO_3_MAP1						
328	VAR_RPDO_3_MAP2						
329	VAR_RPDO_3_MAP3						
330	VAR_RPDO_3_MAP4						
331	VAR_RPDO_4_MAP1						
332	VAR_RPDO_4_MAP2						
333	VAR_RPDO_4_MAP3						
334	VAR_RPDO_4_MAP4						
335	VAR_RPDO_5_MAP1						
336	VAR_RPDO_5_MAP2						
337	VAR_RPDO_5_MAP3						
338	VAR_RPDO_5_MAP4						
339	VAR_RPDO_6_MAP1						
340	VAR_RPDO_6_MAP2						
341	VAR_RPDO_6_MAP3						
342	VAR_RPDO_6_MAP4						
343	VAR_RPDO_7_MAP1						
344	VAR_RPDO_7_MAP2						
345	VAR_RPDO_7_MAP3						
346	VAR_RPDO_7_MAP4						
347	VAR_RPDO_8_MAP1						
348	VAR_RPDO_8_MAP2						
349	VAR_RPDO_8_MAP3						
350	VAR_RPDO_8_MAP4						
351	VAR_TPDO_1_COM					Transmit PDO	
352	VAR_TPDO_2_COM						
353	VAR_TPDO_3_COM						
354	VAR_TPDO_4_COM						
355	VAR_TPDO_5_COM						
356	VAR_TPDO_6_COM						

Index	Name	Type	Format	EPM	Access	Description	Units
357	VAR_TPDO_7_COM						
358	VAR_TPDO_8_COM						
359	VAR_TPDO_1_MAP1					TPDO Mapping	
360	VAR_TPDO_1_MAP2						
361	VAR_TPDO_1_MAP3						
362	VAR_TPDO_1_MAP4						
363	VAR_TPDO_2_MAP1						
364	VAR_TPDO_2_MAP2						
365	VAR_TPDO_2_MAP3						
366	VAR_TPDO_2_MAP4						
367	VAR_TPDO_3_MAP1						
368	VAR_TPDO_3_MAP2						
369	VAR_TPDO_3_MAP3						
370	VAR_TPDO_3_MAP4						
371	VAR_TPDO_4_MAP1						
372	VAR_TPDO_4_MAP2						
373	VAR_TPDO_4_MAP3						
374	VAR_TPDO_4_MAP4						
375	VAR_TPDO_5_MAP1						
376	VAR_TPDO_5_MAP2						
377	VAR_TPDO_5_MAP3						
378	VAR_TPDO_5_MAP4						
379	VAR_TPDO_6_MAP1						
380	VAR_TPDO_6_MAP2						
381	VAR_TPDO_6_MAP3						
382	VAR_TPDO_6_MAP4						
383	VAR_TPDO_7_MAP1						
384	VAR_TPDO_7_MAP2						
385	VAR_TPDO_7_MAP3						
386	VAR_TPDO_7_MAP4						
387	VAR_TPDO_8_MAP1						
388	VAR_TPDO_8_MAP2						
389	VAR_TPDO_8_MAP3						
390	VAR_TPDO_8_MAP4						
391	VAR_TPDO_1_COM_ET						
392	VAR_TPDO_2_COM_ET						
393	VAR_TPDO_3_COM_ET						
394	VAR_TPDO_4_COM_ET						
395	VAR_TPDO_5_COM_ET						
396	VAR_TPDO_6_COM_ET						

NOTE: PIDs 311 - 406 are for REFERENCE ONLY. Do **NOT** use directly. These variables are used by MotionView for non-volatile settings of CAN TPDO/RPDO

Index	Name	Type	Format	EPM	Access	Description	Units
397	VAR_TPDO_7_COM_ET						
398	VAR_TPDO_8_COM_ET						
399	VAR_TPDO_1_COM_IT						
400	VAR_TPDO_2_COM_IT						
401	VAR_TPDO_3_COM_IT						
402	VAR_TPDO_4_COM_IT						
403	VAR_TPDO_5_COM_IT						
404	VAR_TPDO_6_COM_IT						
405	VAR_TPDO_7_COM_IT						
406	VAR_TPDO_8_COM_IT						
407	VAR_CAN_HEARTBEAT				R/W	CAN Heartbeat rate (0x1017) Range: 0 - 65335 milliseconds	
408	VAR_PBUS_STATUS				R	PROFIBUS Status	
409	VAR_PBUS_MASTER_TIMEOUT_VAL				R/W	Timeout Value for PROFIBUS master	
410	VAR_PBUS_DATA_EXCHANGE_TIMEOUT				R/W	Data Exchange Timeout for PROFIBUS Range: 0 - 327680 milliseconds	
411	VAR_PTC_RX				R	PTC resistance in ohms	
412	VAR_PBUS_FIRMWARE_REV					PROFIBUS firmware revision (hex number): 1ST word = major; Least word = minor Example: 0x00010001 = rev 1.1	
413	VAR_PBUS_TIMEOUT_ACTION_CFG			Y		PROFIBUS timeout action. Bits encoded as: Data Exchange Timeout: Bit 0 = 1 Fault, Bit 0 = 0 No Action Master Monitor TImeout: Bit 1 = 1 Fault, Bit 1 = 0 No Action Module Timeout (card not present): Bit 2 = 1 Fault, Bit 2 = 0 No Action	
433	VAR_BRAKE_RELEASE_DELAY				R/W	Range: 0 - 1000 milliseconds; Default 0mS	mS

NOTE: PIDs 311 - 406 are for REFERENCE ONLY. Do **NOT** use directly. These variables are used by MotionView for non-volatile settings of CAN TPDO/RPDO

3.3 Quick Start Examples

Contained in the following four sections are the connections and parameter settings to quickly setup a PositionServo drive for External Torque/Velocity, External Positioning, Internal Torque/Velocity and Internal Positioning modes. These Quick Start reference tables are NOT a substitute for reading the PositionServo User Manual. Observe all safety notices in the PositionServo User and Programming Manuals.

3.3.1 Quick Start - External Torque/Velocity

Table 64: Connections for External Torque/Velocity Mode

I/O (P3)		
Pin	**Name**	**Function**
5	GND	Drive Logic Common
6	+5V	+5V Output (max 100mA)
7	BA+	Buffered Encoder Output: Channel A+
8	BA-	Buffered Encoder Output: Channel A-
9	BB+	Buffered Encoder Output: Channel B+
10	BB-	Buffered Encoder Output: Channel B-
11	BZ+	Buffered Encoder Output: Channel Z+
12	BZ-	Buffered Encoder Output: Channel Z-
22	ACOM	Analog common
23	AO1	Analog output
24	AIN1+	Positive (+) of Analog signal input
25	AIN1 -	Negative (-) of Analog signal input
26	IN_A_COM	Digital input group A COM terminal
27	IN_A1	Digital input A1
28	IN_A2	Digital input A2
29	IN_A3	Digital input A3
41	RDY+	Ready output Collector
42	RDY-	Ready output Emitter
43	OUT1-C	Programmable output #1 Collector
44	OUT1-E	Programmable output #1 Emitter
45	OUT2-C	Programmable output #2 Collector
46	OUT2-E	Programmable output #2 Emitter
47	OUT3-C	Programmable output #3 Collector
48	OUT3-E	Programmable output #3 Emitter
49	OUT4-C	Programmable output #4 Collector
50	OUT4-E	Programmable output #4 Emitter

Note 1: Connections highlighted in BLUE are mandatory/necessary for operation in this mode.

Table 65: Parameter Settings for External Torque/Velocity Mode

<table>
<tr><th>MVOB Folder</th><th>Sub-Folder</th><th>Setting</th></tr>
<tr><td>Parameters</td><td>--</td><td>
<table>
<tr><th>Parameter Name</th><th>Description</th></tr>
<tr><td>Drive Mode</td><td>Set to [Torque] for Torque Mode; [Velocity] for Velocity Mode</td></tr>
<tr><td>Analog Input (Current Scale)</td><td>Torque Mode Only: Set to Required Amps per Volt</td></tr>
<tr><td>Analog Input (Velocity Scale)</td><td>Velocity Mode Only: Set to Required RPM per Volt</td></tr>
<tr><td>Enable Accel/Decel Limits</td><td>Velocity Mode Only: Set to [Enable] to switch on velocity ramp rates; Set to [Disable] to switch OFF (accelerate at current limit)</td></tr>
<tr><td>Accel Limit</td><td>Velocity Mode Only: Set Acceleration Limit in RPM/Sec</td></tr>
<tr><td>Decel Limit</td><td>Velocity Mode Only: Set Deceleration Limit in RPM/Sec</td></tr>
<tr><td>Reference</td><td>Set to [External] for external Torque/Velocity Mode</td></tr>
<tr><td>Enable Switch Input</td><td>Set to [Run] to allow Enable/Disable of the PositionServo to be controlled via Input A3 (Dedicated Enable)</td></tr>
</table>
</td></tr>
<tr><td>IO</td><td>Digital IO</td><td>
<table>
<tr><th>Parameter Name</th><th>Description</th></tr>
<tr><td>Output 1 Function</td><td rowspan="4">Output # indicates Digital Output No. 1-4;
Set value to select Output Functionality;
Output Function Values: 1=Not Assigned; 2=Zero Speed;
3=In Speed Window; 4=Current Limit; 5=Run Time Fault; 6=Ready;
7=Brake; 8=In Position</td></tr>
<tr><td>Output 2 Function</td></tr>
<tr><td>Output 3 Function</td></tr>
<tr><td>Output 4 Function</td></tr>
</table>
</td></tr>
<tr><td>IO</td><td>Analog IO</td><td>
<table>
<tr><th>Parameter Name</th><th>Description</th></tr>
<tr><td>Analog Input Dead Band</td><td>Set Zero Speed Dead Band in mV for Torque/Velocity Reference on Analog Input 1</td></tr>
<tr><td>Analog Input Offset</td><td>Set Torque/Velocity Reference Input Offset on Analog Input 1 to match Controller Offset</td></tr>
<tr><td>Adjust Analog Input Zero Offset</td><td>Tool to automatically learn the Analog Input Offset (of Analog Input 1)</td></tr>
</table>
</td></tr>
<tr><td>Limits</td><td>Velocity Limits</td><td>
<table>
<tr><th>Parameter Name</th><th>Description</th></tr>
<tr><td>Zero Speed</td><td>Velocity Mode Only: Set a bandwidth (around ORPM) for activation of the Zero Speed Output/Flag. Set digital output function to ‘2’.</td></tr>
<tr><td>At Speed</td><td>Velocity Mode Only: Set a Target Speed for activation of the At Speed Output/Flag</td></tr>
<tr><td>Speed Window</td><td>Velocity Mode Only: Set a bandwidth (around At Speed parameter) for activation of the At Speed Output/Flag. Set digital output function to ‘3’.</td></tr>
</table>
</td></tr>
<tr><td>Compensation</td><td>--</td><td>
<table>
<tr><th>Parameter Name</th><th>Description</th></tr>
<tr><td>Velocity P-Gain</td><td>Velocity Mode Only: Set P-Gain for Velocity Loop</td></tr>
<tr><td>Velocity I-Gain</td><td>Velocity Mode Only: Set I-Gain for Velocity Loop</td></tr>
<tr><td>Gain Scaling</td><td>Velocity Mode Only: Apply Scaling Factor to Velocity Gain Set</td></tr>
</table>
</td></tr>
</table>

Note 1: Parameters highlighted in BLUE are mandatory/necessary for operation in this mode.

3.3.2 Quick Start - External Positioning

Table 66: Connections for External Positioning Mode

I/O (P3)		
Pin	Name	Function
1	MA+	Master Encoder A+ / Step+ input
2	MA-	Master Encoder A- / Step- input
3	MB+	Master Encoder B+ / Direction+ input
4	MB-	Master Encoder B- / Direction- input
5	GND	Drive Logic Common
6	+5V	+5V Output (max 100mA)
7	BA+	Buffered Encoder Output: Channel A+
8	BA-	Buffered Encoder Output: Channel A-
9	BB+	Buffered Encoder Output: Channel B+
10	BB-	Buffered Encoder Output: Channel B-
11	BZ+	Buffered Encoder Output: Channel Z+
12	BZ-	Buffered Encoder Output: Channel Z-
26	IN_A_COM	Digital input group A COM terminal
27	IN_A1	Digital input A1
28	IN_A2	Digital input A2
29	IN_A3	Digital input A3
41	RDY+	Ready output Collector
42	RDY-	Ready output Emitter
43	OUT1-C	Programmable output #1 Collector
44	OUT1-E	Programmable output #1 Emitter
45	OUT2-C	Programmable output #2 Collector
46	OUT2-E	Programmable output #2 Emitter
47	OUT3-C	Programmable output #3 Collector
48	OUT3-E	Programmable output #3 Emitter
49	OUT4-C	Programmable output #4 Collector
50	OUT4-E	Programmable output #4 Emitter

Note 1: Connections highlighted in BLUE are mandatory/necessary for operation in this mode.
Note 2: Connections highlighted in GREEN are frequently required in applications of this type.

Table 67: Parameter Settings for External Positioning Mode

<table>
<tr><th>MVOB Folder</th><th>Sub-Folder</th><th>Setting</th></tr>
<tr><td>Parameters</td><td>--</td><td>
<table>
<tr><th>Parameter Name</th><th>Description</th></tr>
<tr><td>Drive Mode</td><td>Set to [Position] for Position Mode</td></tr>
<tr><td>Reference</td><td>Set to [External] for external Position Mode</td></tr>
<tr><td>Step Input Type</td><td>Set to either [Step and Direction] or [Master Encoder] to match the Position Controller</td></tr>
<tr><td>System to Master Ratio</td><td>Set Electronic Gear Ratio for Reference Signal to the PositionServo Motor Output</td></tr>
<tr><td>Enable Switch Input</td><td>Set to [Run] to allow Enable/Disable of the PositionServo to be controlled via Input A3 (Dedicated Enable)</td></tr>
<tr><td>Resolver Track</td><td>If using Resolver Feedback, set value that represents the pulses per revolution required on the PositionServo simulated encoder.
0=1024ppr; 1=256ppr; 2=360ppr; 3=400ppr; 4=500ppr; 5=512ppr; 6=720ppr; 7=800ppr; 8=1000ppr; 9=1024ppr; 10=2000ppr; 11=2048ppr; 12=2500ppr; 13=2880ppr; 14=250ppr; 15=4096ppr</td></tr>
</table>
</td></tr>
<tr><td>IO</td><td>Digital IO</td><td>
<table>
<tr><th>Parameter Name</th><th>Description</th></tr>
<tr><td>Output 1 Function</td><td rowspan="4">Output # indicates Digital Output No. 1-4;
Set value to select Output Functionality;
Output Function Values: 1=Not Assigned; 2=Zero Speed; 3=In Speed Window; 4=Current Limit; 5=Run Time Fault; 6=Ready; 7=Brake; 8=In Position</td></tr>
<tr><td>Output 2 Function</td></tr>
<tr><td>Output 3 Function</td></tr>
<tr><td>Output 4 Function</td></tr>
<tr><td>Hard Limit Switches Action</td><td>Set to Enable Inputs A1 and A2 to act as System Hard Limit Switches and define functionality in the event of an active input.</td></tr>
</table>
</td></tr>
<tr><td>Limits</td><td>Position Limits</td><td>
<table>
<tr><th>Parameter Name</th><th>Description</th></tr>
<tr><td>Position Error</td><td>Set Position Error Limit at which Position Error Timer starts counting</td></tr>
<tr><td>Max Error Time</td><td>Set Maximum Error Time for Position Error Correction before position error trip occurs.</td></tr>
</table>
</td></tr>
<tr><td>Compensation</td><td>--</td><td>
<table>
<tr><th>Parameter Name</th><th>Description</th></tr>
<tr><td>Velocity P-Gain</td><td>Set P-Gain for Velocity Loop</td></tr>
<tr><td>Velocity I-Gain</td><td>Set I-Gain for Velocity Loop</td></tr>
<tr><td>Position P-Gain</td><td>Set P-Gain for Position Loop</td></tr>
<tr><td>Position I-Gain</td><td>Set I-Gain for Position Loop</td></tr>
<tr><td>Position D-Gain</td><td>Set D-Gain for Position Loop</td></tr>
<tr><td>Position I-Limit</td><td>The Position I-Limit will clamp the Position I-Gain compensator to prevent excessive torque overshoot caused by an over-accumulation of I-Gain.</td></tr>
<tr><td>Gain Scaling</td><td>Apply Scaling Factor to Velocity Gain Set</td></tr>
</table>
</td></tr>
</table>

Note 1: Parameters highlighted in BLUE are mandatory/necessary for operation in this mode.

3.3.3 Quick Start - Internal Torque/Velocity

Table 68: Internal Torque/Velocity Mode

Connections for Internal Torque/Velocity: I/O (P3)

Pin	Name	Function
20	AIN2+	Positive (+) of Analog signal input
21	AIN2-	Negative (-) of Analog signal input
22	ACOM	Analog common
23	AO1	Analog output
24	AIN1+	Positive (+) of Analog signal input
25	AIN1 -	Negative (-) of Analog signal input
26	IN_A_COM	Digital input group A COM terminal
27	IN_A1	Digital input A1
28	IN_A2	Digital input A2
29	IN_A3	Digital input A3
30	IN_A4	Digital input A4
31	IN_B_COM	Digital input group B COM terminal
32	IN_B1	Digital input B1
33	IN_B2	Digital input B2
34	IN_B3	Digital input B3
35	IN_B4	Digital input B4
36	IN_C_COM	Digital input group C COM terminal
37	IN_C1	Digital input C1
38	IN_C2	Digital input C2
39	IN_C3	Digital input C3
40	IN_C4	Digital input C4
41	RDY+	Ready output Collector
42	RDY-	Ready output Emitter
43	OUT1-C	Programmable output #1 Collector
44	OUT1-E	Programmable output #1 Emitter
45	OUT2-C	Programmable output #2 Collector
46	OUT2-E	Programmable output #2 Emitter
47	OUT3-C	Programmable output #3 Collector
48	OUT3-E	Programmable output #3 Emitter
49	OUT4-C	Programmable output #4 Collector
50	OUT4-E	Programmable output #4 Emitter

Variable References for Internal Torque/Velocity

Index	Name	EPM	R/W	Description
29	VAR_ENABLE_SWITCH_TYPE	Y	R/W	Enable switch function: 0-inhibit only, 1- Run
34	VAR_DRIVEMODE	Y	R/W	Drive mode selection: 0-torque 1-velocity, 2-position
37	VAR_REFERENCE	Y	R/W	Reference source: set to: 1 - internal (for 'internal torque' or 'internal velocity' mode)
44	VAR_VP_GAIN	Y	R/W	Velocity loop Proportional gain Range: 0 - 32767
45	VAR_VI_GAIN	Y	R/W	Velocity loop Integral gain Range: 0 - 16383
51	VAR_VREG_WINDOW	Y	R/W	Gains scaling coefficient Range: -5 - +4
52	VAR_ENABLE	N	W	Software Enable/Disable: 0 – disable, 1 - enable
58	VAR_VLIMIT_ZEROSPEED	Y	R/W	Zero Speed value Range: 0 - 100
59	VAR_VLIMIT_SPEEDWND	Y	R/W	Speed window Range: 10 - 10000
60	VAR_VLIMIT_ATSPEED	Y	R/W	Target speed for velocity window Range: -10000 - +10000
71	VAR_AIN1	N	R	Analog Input AIN1 current value
72	VAR_AIN2	N	R	Analog Input AIN2 current value
75	VAR_ENABLE_ACCELDECEL	Y	R/W	Enable Accel/Decel (velocity mode), 0 – disable, 1 - enable
76	VAR_ACCEL_LIMIT	Y	R/W	Accel value for velocity mode Range: 0.1 - 5000000
77	VAR_DECEL_LIMIT	Y	R/W	Decel value for velocity mode Range: 0.1 - 5000000
139	VAR_IREF	N	R/W	Internal ref Current or Velocity mode
192	VAR_CURRENT_VEL_PPS	N	R	Current velocity in PPS (pulses per sample)
193	VAR_CURRENT_ACCEL_PPSS	N	R	Current acceleration (demanded value) value
217	VAR_CURRENT_VEL	N	R	Current velocity (demanded value)
218	VAR_CURRENT_ACCEL	N	R	Current acceleration (demanded value)

Positional Mode Language Reference - Enable/Disable

Command	Syntax	Long Name
DISABLE	DISBALE	Turns OFF Servo output
ENABLE	ENABLE	Turns ON Servo output

Note 1: Connections highlighted in BLUE are mandatory/necessary for operation in this mode.

Example Internal Torque Program

```
;Program slowly increases Motor Torque until nominal motor current is reached
VAR_DriveMode = 0                    ;Set Drive to Torque mode
VAR_Reference = 1                    ;Set Reference to Internal control
Program Start:
IREF = 0                             ;Reset Torque Reference to 0(Amps)
Wait While !In_A3                    ;Wait while Enable input is OFF
Enable                               ;Enable Drive
Torque_Loop:
Wait Time 500                        ;Set time between step increases in Torque
    If REF < VAR_CurrentLimit        ;If Set Torque < Motor Nominal Torque
    IREF = IREF+0.1                  ;Then increase by 0.1(Amps)
    GOTO Torque_Loop                 ;Loop to next torque increase
Else
    Goto Program_Start               ;Else restart program
Endif
END
```

Example Internal Velocity Program

```
;Program slowly increases and decreases Motor Velocity between Maximum Velocity Forward direction and
;Maximum Velocity Reverse direction producing a saw-tooth velocity profile.
Define MaxVelocityRPS 60             ;Enter Maximum Velocity (RPS) value here
Define VelocityStepRPS 1             ;Define Velocity INC/DEC per Step/Program Loop (RPS)
Define VelocityStepTime 200          ;Define Time for Velocity Steps in mS
Define Velocity_Inc_Dec V0           ;Define a Variable to identify if Velocity is currently INC/DECreasing
VAR_DriveMode = 1                    ;Set Drive to Velocity mode
VAR_Reference = 1                    ;Set Reference to Internal control
VAR_Enable_AccelDecel = 1            ;Enable Accel/Decel Ramps
VAR_Accel_Limit = 3000               ;Set Accel Rate required in RPS^2
VAR_Decel_Limit = 3000               ;Set Decel Rate required in RPS^2
Program Start:
IREF = 0                             ;Reset Velocity Reference to 0(RPS)
Wait While !In_A3                    ;Wait while Enable input is OFF
Enable                               ;Enable Drive
Velocity_Loop:
Wait Time VelocityStep Time          ;Set Time between Step Increases/Decreases in Velocity (mS)
    If REF <= MaxVelocityRPS         ;If Current Motor Velocity < MaxVelocityRPS
    IREF = IREF+VelocityStepRPS      ;Then increase Velocity by VelocityStepRPS
    Else
    Velocity_Inc_Dec = 1             ;Set Variable to start decreasing velocity
    Endif
Else                                 ;If Speed Decreasing
    If REF >= -1* MaxVelocityRPS     ;If Current Motor Velocity > -MaxVelocityRPS
    IREF = IREF-VelocityStepRPS      ;Then decrease Velocity by VelocityStepRPS
    Else
    Velocity_Inc_Dec = 0             ;Set Variable to start increasing velocity
    Endif
Endif
Goto Velocity_Loop                   ;Loop to next Velocity Increase/Decrease
END                                  ;End Code - Never Reached
On Fault                             ;Fault Handler
    Resume Program_Start             ;Resume at Program Start
EndFault
```

3.3.4 Quick Start - Internal Positioning

Table 69: Internal Positioning

Connections: I/O (P3)

Pin	Name	Function
26	IN_A_COM	Digital input group A COM terminal
27	IN_A1	Digital input A1
28	IN_A2	Digital input A2
29	IN_A3	Digital input A3
30	IN_A4	Digital input A4
31	IN_B_COM	Digital input group B COM terminal
32	IN_B1	Digital input B1
33	IN_B2	Digital input B2
34	IN_B3	Digital input B3
35	IN_B4	Digital input B4
36	IN_C_COM	Digital input group C COM terminal
37	IN_C1	Digital input C1
38	IN_C2	Digital input C2
39	IN_C3	Digital input C3
40	IN_C4	Digital input C4
41	RDY+	Ready output Collector
42	RDY-	Ready output Emitter
43	OUT1-C	Programmable output #1 Collector
44	OUT1-E	Programmable output #1 Emitter
45	OUT2-C	Programmable output #2 Collector
46	OUT2-E	Programmable output #2 Emitter
47	OUT3-C	Programmable output #3 Collector
48	OUT3-E	Programmable output #3 Emitter
49	OUT4-C	Programmable output #4 Collector
50	OUT4-E	Programmable output #4 Emitter

Language Reference

Enable/Disable		
Command	**Syntax**	**Long Name**
DISABLE	DISBALE	Turns OFF Servo output
ENABLE	ENABLE	Turns ON Servo output

Program Structure		
Command	**Syntax**	**Long Name**
STOP MOTION	STOP MOTION	Stop AA Motion - Clear
STOP MOTION QUICK	STOP MOTION QUICK	Motion Slack
WAIT	WAIT MOTION COMPLETE	Wait

Move / Motion Commands		
Command	**Syntax**	**Long Name**
MOVE	MOVE [BACK] UNTIL <condition> [,C]	Move
MOVED	MOVED <distance> [,S] [,C]	Move Distance
MOVEP	MOVEP <absolute position> [,S] [,C]	Move to Position
MOVEDR	MOVEDR <distance> , <displacement> [,C]	Registered Distance Move
MOVEPR	MOVEPR <distance> , <displacement> [,C]	Registered Position Move
MDV	MDV <[-]segment distance>,<segment final velocity>[,S]	Segmented Move
MOTION SUSPEND	MOTION SUSPEND	Temporarily Suspend Motion
MOTION RESUME	MOTION RESUME	Statement Resumes Motion

Example Internal Positioning Program

```
;****************************************** HEADER ********************************************
;Title:                         Pick and Place example program
;Author:                        940 Product Management
;Description:                   This is a sample program that shows a simple application that
;                               picks up a part moves to a set position and drops the part

;*****************************************  I/O List  *****************************************
;   Input A1            -       not used
;   Input A2            -       not used
;   Input A3            -       Enabled
;   Input A4            -       not used
;   Input B1            -       not used
;   Input B2            -       not used
;   Input B3            -       not used
;   Input B4            -       not used
;   Input C1            -       not used
;   Input C2            -       not used
;   Input C3            -       not used
;   Input C4            -       not used
;
;   Output 1            -       Pick Arm
;   Output 2            -       Gripper
;   Output 3            -       not used
;   Output 4            -       not used

;******************************** Initialize and Set Variables ********************************
UNITS = 1
ACCEL = 75
DECEL =75
MAXV = 10
APOS = 0

;******************************************** Events **********************************************
;Set Events handling here

;****************************************** Main Program ******************************************
RESET_DRIVE:
WAIT UNTIL IN_A3                ;Check the Enable / Inhibit switch is made before continuing
ENABLE                          ;Enable the Drive
PROGRAM_START:
MOVEP  0                        ;Move to Pick position
OUT1 = 1                        ;Turn on output 1 on to extend Pick arm
WAIT TIME 1000                  ;Delay 1 sec to extend arm
OUT2 = 1                        ;Turn on output 2 to Engage gripper
WAIT TIME 1000                  ;Delay 1 sec to Pick part
OUT1 = 0                        ;Turn off output 1 to Retract Pick arm
MOVEP 100                       ;Move to Place position
OUT1 = 1                        ;Turn on output 1 on to extend Pick arm
WAIT TIME 1000                  ;Delay 1 sec to extend arm
OUT2 = 0                        ;Turn off output 1 to Disengage gripper
WAIT TIME 1000                  ;Delay 1 sec to Place part
OUT1 = 0                        ;Retract Pick arm
GOTO PROGRAM_START
END

;****************************************** Sub-Routines ******************************************
;       Enter Sub-Routine code here

;************************************* Fault Handler Routine **************************************
;       Enter Fault Handler code here
ON FAULT

ENDFAULT
```

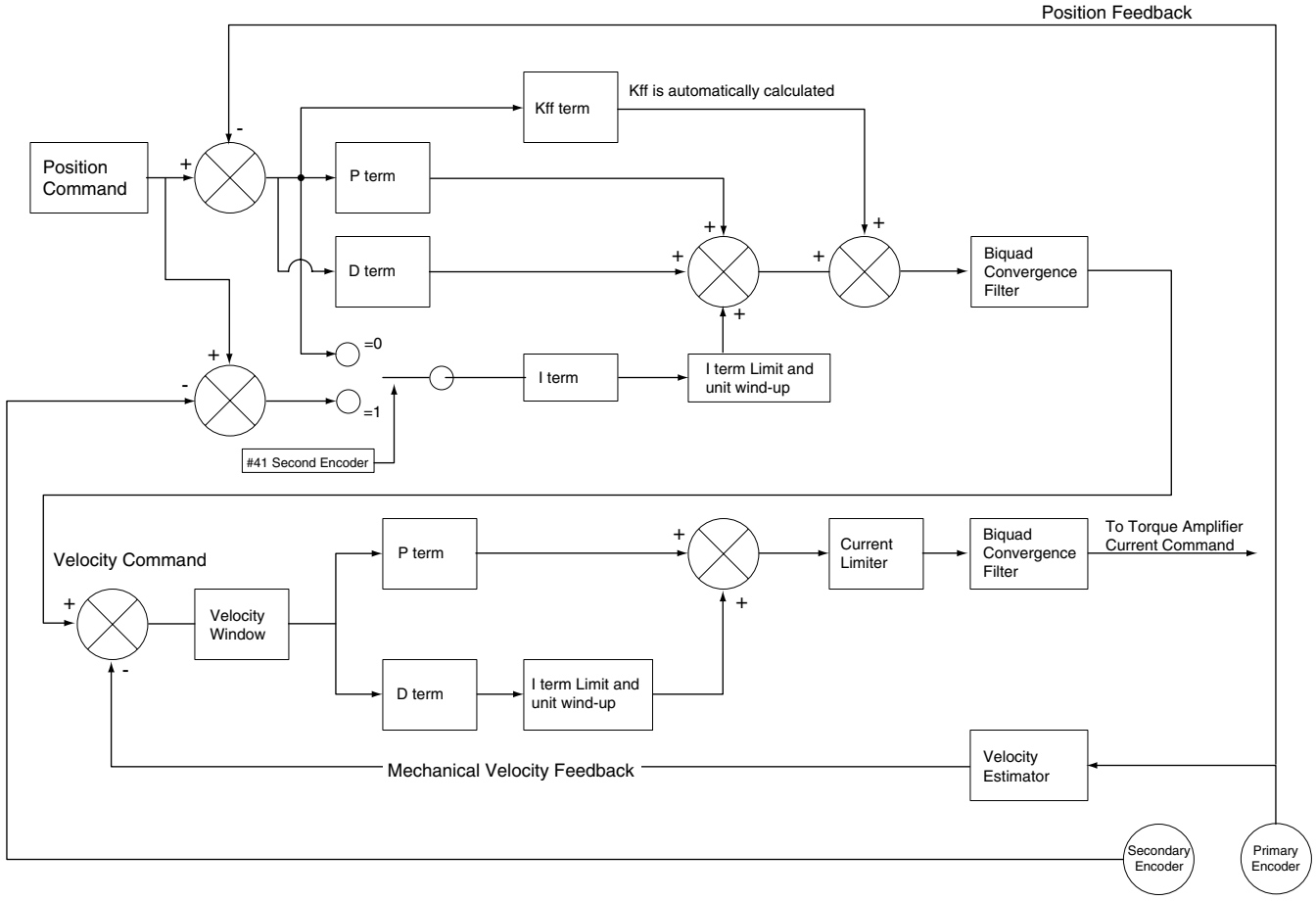
3.4 PositionServo Reference Diagrams

This section contains the process flow diagrams listed in Table 70. These diagrams are for reference only.

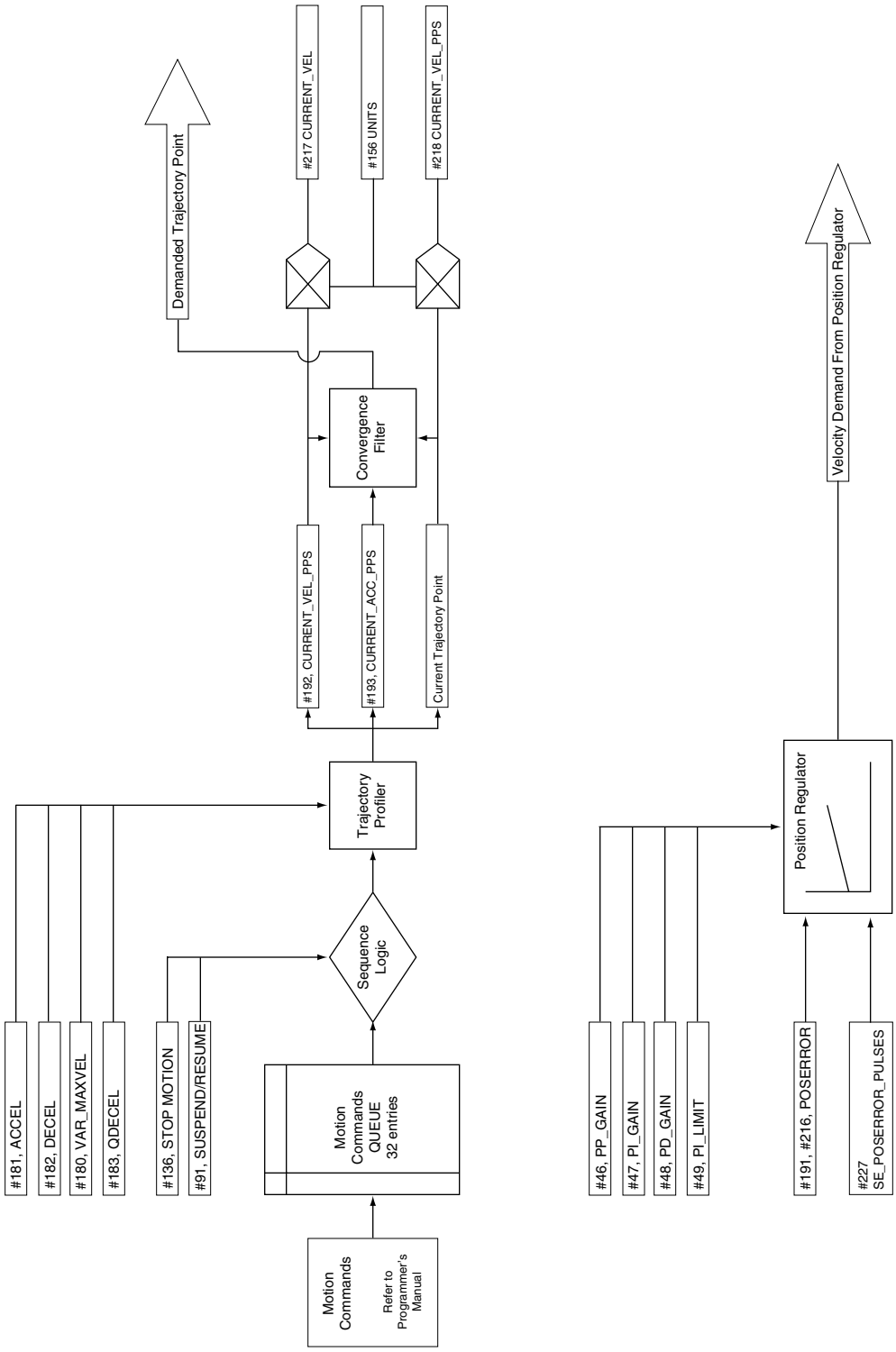
Table 70: PositionServo Process Flow Diagrams

Drawing #	Description
S999	Position and Velocity Regulator
S1000	Motion Commands -> Motion Queue -> Trajectory Generator
S1001	Current Command -> Motor
S1002	Encoder Inputs
S1003	Analog Inputs
S1004	Analog Outputs
S1005	Digital Inputs
S1006	Digital Outputs

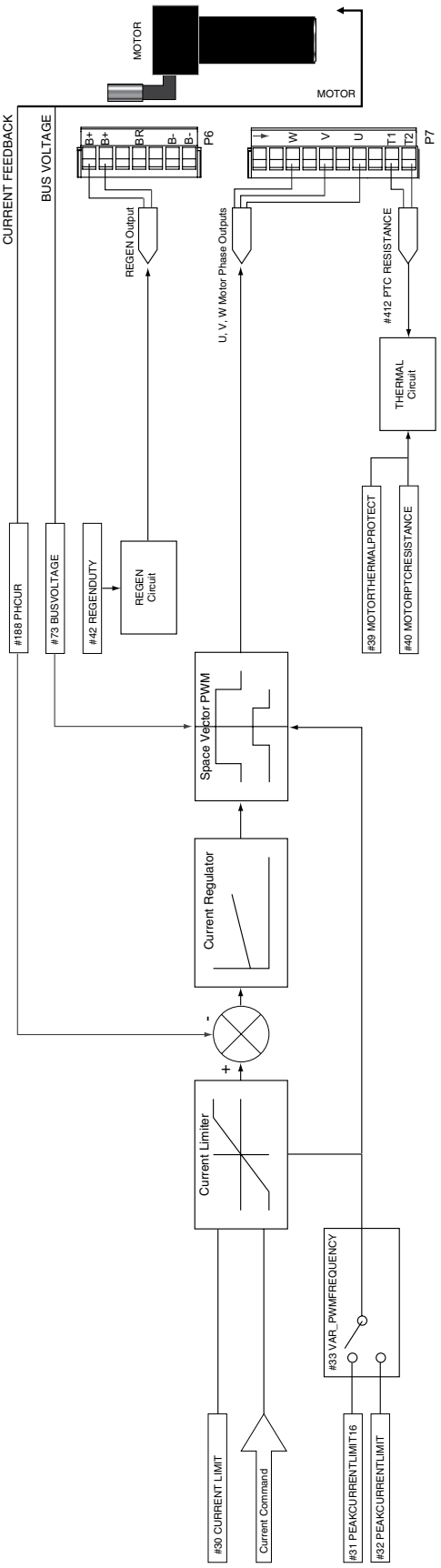
Position and Velocity Regulators



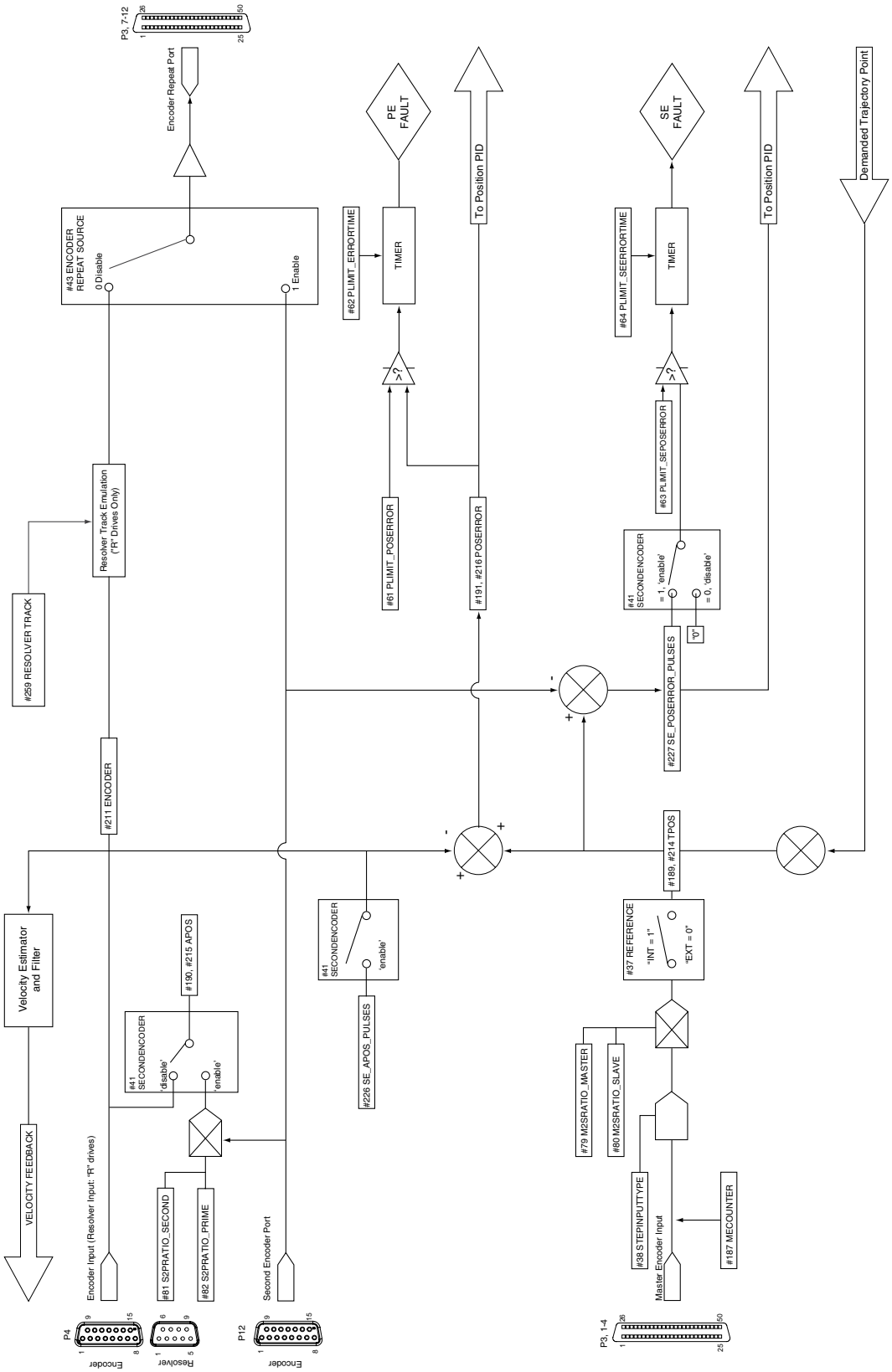
Motion Commands, Motion Queue & Trajectory Generator



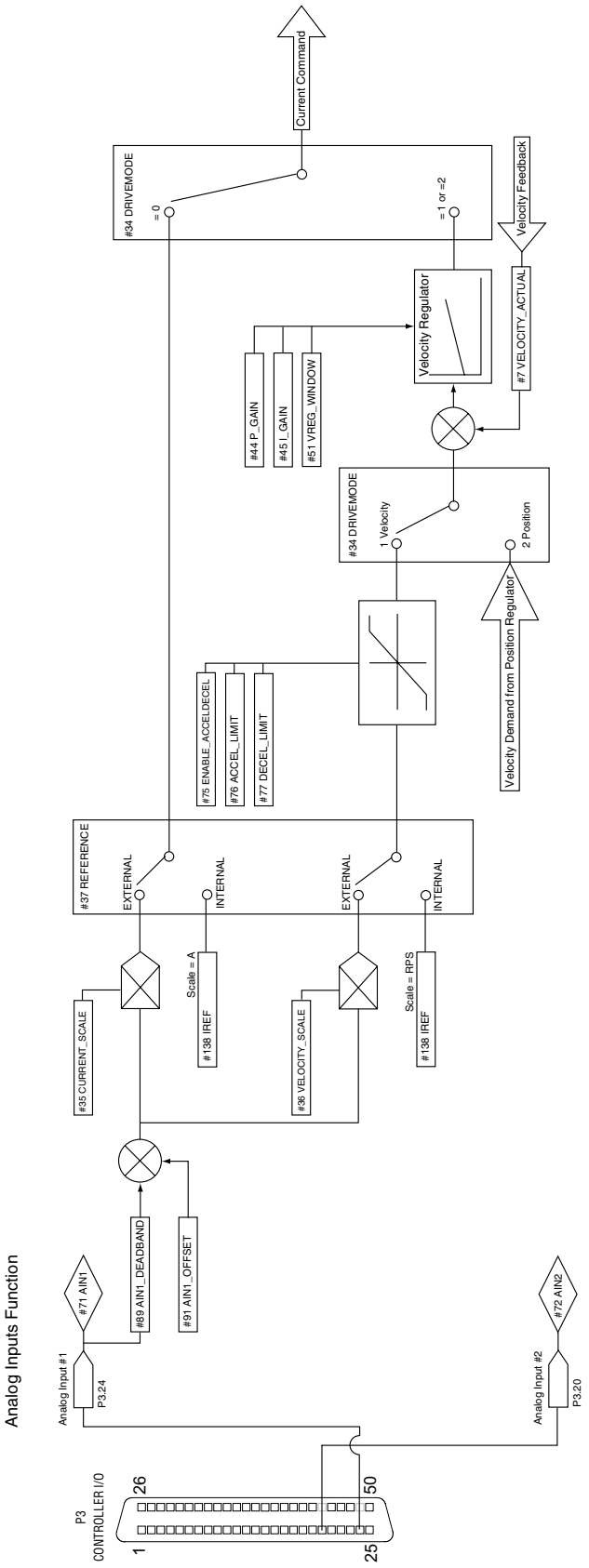
Current Command --> Motor



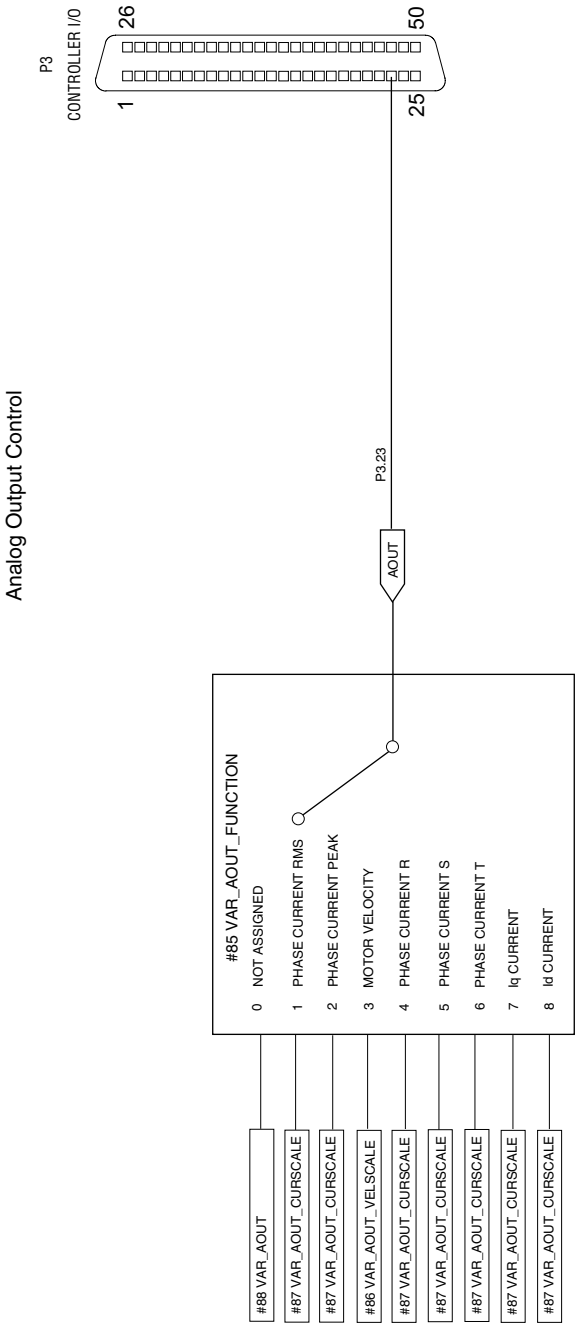
Encoder Inputs



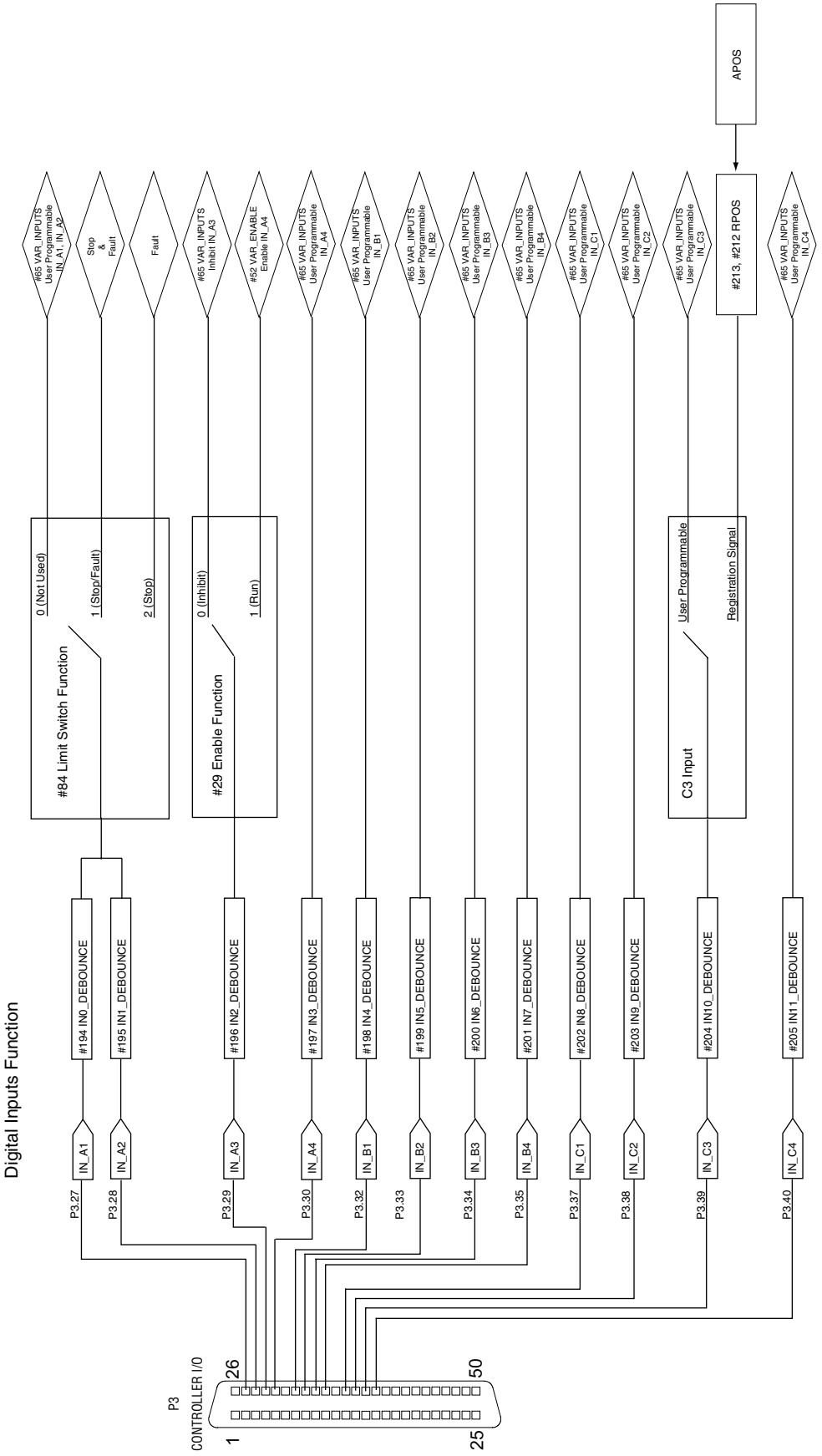
Analog Inputs



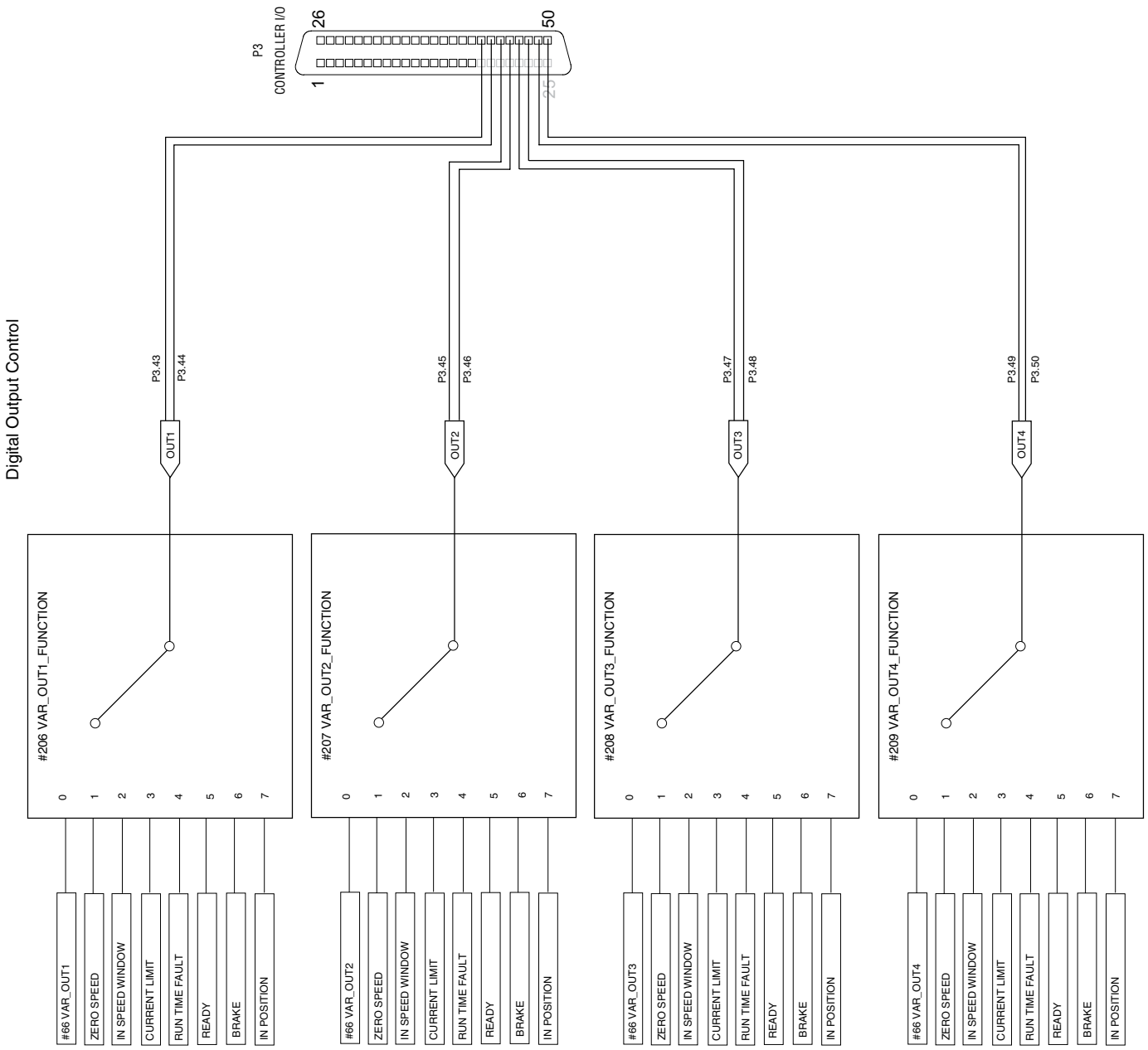
Analog Output



Digital Inputs



Digital Outputs



Lenze AC Tech Corporation

630 Douglas Street • Uxbridge, MA 01569 • USA
Sales 800 217 9100 • Service 508 278 9100
www.lenzeamericas.com

PM94201A