www.DanaherMotion.com

SC750 Series

Programmable, Digital Brushless ServorController
ServoBASIC Plus™ Reference Manual
Version 2.8
Part # 903-075312-80
Rev G

Record of Manual Revisions

Revision	Date	Description of Revision
A	1987	Initial Release
F	2003	Update corporate identity
G	09/2004	Update contact information, revise formatting

Copyright Information
© Copyright 1987-2004 Danaher Motion – All rights reserved.
Printed in the United States of America

NOTICE:
Not for use or disclosure outside of Danaher Motion except under written agreement. All rights are reserved. No part of this book shall be reproduced, stored in retrieval form, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise without the written permission from the publisher. While every precaution has been taken in the preparation of the book, the publisher assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.
This document is proprietary information of Danaher Motion that is furnished for customer use ONLY. No other uses are authorized without written permission of Danaher Motion. Information in this document is subject to change without notice and does not represent a commitment on the part of Danaher Motion. Therefore, information contained in this manual may be updated from time-to-time due to product improvements, etc., and may not conform in every respect to former issues.
Danaher Motion reserves the right to make engineering refinements on all its products. Such refinements may affect information in instructions. **USE ONLY THE INSTRUCTIONS PACKED WITH THE PRODUCT.**

Safety-alert symbols used in this document are:

WARNING

Alerts users to potential physical danger or harm. Failure to follow warning notices could result in personal injury or death.

CAUTION

Directs attention to general precautions, which if not followed, could result in personal injury and/or equipment damage.

NOTE

Highlights information critical to your understanding or use of the product.

Table of Contents

1. CONVENTIONS

This chapter contains a summary of conventions used with ServoBASIC Plus. Topics covered are:

- User-defined variable names
- Characters
- Operators used in programming
- Labels
- Notation conventions
- ServoBASIC Plus instruction types
- BASIC language instructions

1. 1 *USER-DEFINED VARIABLE NAMES*

User-defined variables are used with BASIC functions and statements for general programming tasks. There are three basic types of user-defined variables:

- INTERGER
- FLOAT
- STRING

User-defined variable names must be alphanumeric, less than 40 characters in length, and contain no spaces. They must begin with an alphabetic character. Variables must be declared using the DIM statement prior to their use within a program. Otherwise, a compilation error occurs.

Variable names are not case sensitive.

NOTE

Integer: INTEGER variables are specified as either INTEGER or LONG. However, each is represented as a 32-bit signed number.

Float: FLOAT variables can be designated as SINGLE, DOUBLE or FLOAT. However, all will be represented as a single precision IEEE floating-point number.

String: STRING variables are intended to permit storage of ASCII character strings. These variables are manipulated using the string functions.

1. 2 CHARACTERS

Along with ServoBASIC Plus instructions, alphabetic and numeric characters are used in creating programs.

Alphabetic: Any alphabetic character is legal in ServoBASIC Plus. Program instructions are NOT case sensitive. Alpha characters may be typed in either upper or lower case. ServoBASIC Plus processes all text in upper case after compilation. The drive does not recognize case when the text is part of a string that is text bracketed by quotes for printout or display.

Numeric: The digits 0 through 9 are legal for use in ServoBASIC Plus.

1. 3 OPERATORS USED IN PROGRAMMING

The operators used by ServoBASIC Plus are arithmetic, relational and logical, and are evaluated in that order of precedence. However, operations within parentheses are performed first. Inside the parentheses the usual order of precedence occurs.

Arithmetic The arithmetic operators are:

Arithmetic Operator	Description of Operation	Example
^	Exponential	2^16
- (one variable)	Negation of value	-3
*,/	Multiplication/Division	4.21*3, 10.5/2
MOD	Modulus (remainder)	5 MOD 2
+ / - (two variables)	Addition/Subtraction	27 = 8, 19 - 2

NOTE

When multiple arithmetic operators are used in an expression, they are performed in the order of precedence given in the table. Multiplication is performed before addition, and so on. Integer division is not supported.

Relational Relational operators are used in **IF-THEN-ELSE**, **WHILE-WEND**, and **FOR-NEXT** statements. Relational operators are:

Relational Operator	Description of Operation	Example
=	Equality	IF Value = 9 THEN GOTO LABEL_1
< >	Inequality	IF Value < > 9 THEN GOTO LABEL_1
<	Less than	IF Value < 99 THEN GOTO LABEL_1
>	Greater than	IF Value1 > Value2 THEN GOTO LABEL_1
<=	Less than or equal to	IF Value1 <= Value2 THEN GOTO LABEL_2
>=	Greater than or equal to	IF Value1 >= Value2 THEN GOTO LABEL_1

NOTE

Arithmetic operators are performed before relational operators in an executing program line. Relational operators are performed in the order of precedence shown in the table.

Logical: Logical operators are used in **IF-THEN-ELSE, WHILE-WEND**, and **FOR-NEXT** statements. The logical operators are:

Logical Operator	Description of Operation	Example
AND	Both conditions must be true	IF Value1 > 5 AND Value2 <= 3.00 THEN GOTO LABEL_1
OR	Either or both conditions must be true	IF Value1 =1 OR Value2 = 0 THEN GOTO LABEL_1
XOR	Either but not both conditions must be true	WHILE Value1 > 1 XOR ENCDR.POS = 102400

NOTE

Logical operators are performed in the order of precedence given in the table.

1. 4 *LABELS*

Labels reference particular lines within a program to permit access by program flow control statements, such as the GOTO statement. Labels must be alphanumeric and less than 40 characters in length.

Labels must begin with an alpha character and must be terminated with a colon.

For example, the following program performs a simple loop, printing the same message over and over until the program is terminated.

```
Loop_Again:
PRINT "All work and no play makes Jack a dull boy."
GOTO Loop_Again
```

1. 5 *NOTATION CONVENTIONS*

The following notation conventions are used in this manual when explaining ServoBASIC Plus language use.

Notation	Named	Indicates
[ ]	Square brackets	The entry within the brackets is optional
…	3 dots	The entry may be repeated multiple times
info	Information in computer typeface	Computer information displayed on the computer screen
Italics	Italicized information	For emphasis or definition

1. 6 *INSTRUCTION TYPES*

ServoBASIC Plus consists of programming statements or functions, and arithmetic operations permitted in the BASIC programming language. A complete list of these instructions is given in Section 2 of this manual.

Statements: Statements are of two types, BASIC and PacSci ServoBASIC Plus:

- BASIC statements control the flow of instructions within a program. They direct the execution of functions, for example comparing function results and going to specific points in the program based on the comparison, prompting for input, printing results of functions, and so on.
- PacSci ServoBASIC Plus statements control the motion of the motor in real time. Motion statements command the motor to move to a specified position or at constant velocity, display parameters and status of the drive, etc.

Functions: BASIC functions perform a computation and return a value that can be used in arithmetic expressions. For example, BASIC functions convert decimal numbers to integers, and convert and ASCII code to its equivalent screen display character. PacSci ServoBASIC Plus also supports string manipulation functions.

Pre-defined Variable: The SC750 contains a large number of pre-defined variables, which are used to extend the capabilities of the standard BASIC language to make ServoBASIC Plus capable of motion and I/O control. These variables may be functionally grouped as follows:

- Variables, which control functionality of ServoBASIC Plus motion and I/O statements. **RUN.SPEED, PAUSE.TIME** and **ACCEL.RATE** are examples of this type of predefined variable.
- Variables, which control motion and I/O directly. **GEARING, ENC.OUT**, and **OUTPUTS** are examples of this type of predefined variable.
- Variables that are maintained by the internal firmware and contain information about the present state of the controller. These variables are typically read-only and include **VELOCITY, POSITION, MOVING** and **INPUTS**.

Non-volatile Parameters: There is a relatively small subset of the predefined variables whose values are stored in the non-volatile memory of the SC750 controller. These predefined variables are referred to as non-volatile parameters. These variables, like **KVP** and **KPP** appear at the beginning of a ServoBASIC Plus program between **PARAMS START** and **PARAMS END**.

Pre-defined variable types: Variables are the values acted upon by functions, or as a result of arithmetic operations. Variables can be further categorized as Read/Write (R/W) or Read Only (R/O). Pre-defined variables are reserved for use with specific PacSci functions. These pre-defined variables are either:

Floating-point numbers with values to the right of the decimal place. Used with functions that require decimal numbers, for example the VELOCITY variable contains the motor speed in revolutions-per-minute.

or

Integer integers used with functions that require integers, for example the number of steps to move the motor. Some pre-defined variables are read-only, that is they cannot be altered from the keyboard or by the program. The INPUTS variable, for instance, is dependant solely on the state of the programmable inputs at the connector interface and cannot be altered from the keyboard.

1. 7 *BASIC LANGUAGE INSTRUCTIONS*

ServoBASIC Plus is based upon the BASIC programming language. The key program control statements and functions defined for BASIC are also utilized in ServoBASIC Plus. The instructions common to both BASIC and ServoBASIC Plus are functionally equivalent enhancements permitting motion control work within the framework of the BASIC language programming structure.

1.7.1. BASIC LANGUAGE STATEMENTS

BASIC language statements perform program flow and decision-making control instructions. Arithmetic and logical expressions are also valid elements of BASIC language statements. Although a BASIC language-programming guide is the best reference for BASIC language programming, the key statements supported by ServoBASIC Plus are summarized below.

Instruction	Description
BEEP	Transmits a speaker beep command to the serial port.
DIM	Declares variable type.
CALL SUB	Calls a subroutine, executes it and returns.
END, END IF, END SUB	Terminates the execution of a program or block structure.
FOR…NEXT	Allows a series of statements to be executed in a loop to be executed a specified number of times.
GOSUB…RETURN	Branches to a subroutine, execute it, and returns to instruction following GOSUB statement.
GOTO	Branches unconditionally to specified label and commences execution.
IF…THEN…ELSE, …ELSE IF, END IF	Permits conditional execution pending evaluation and outcome of Boolean expression.
INPUT	Reads a character string received by the serial communications port.
PRINT	Displays output on the terminal screen while the program is running.
REM or ‘ (Apostrophe)	Included comments in the program.
STOP	Stops the execution of the program.
SWAP	Exchanges the value of two variables.
WHILE…WEND	Executes a series of statements in a loop as long as the outcome of a Boolean expression is true.

1.7.2. BASIC LANGUAGE FUNCTIONS

BASIC functions consist of two fundamental types:

- Arithmetic functions
- String functions

Either type performs an operation on a specified argument and returns a result. Arithmetic functions perform a numerical calculation on an argument and return a numerical result. String functions operate on character string arguments. BASIC functions can be incorporated in expressions.

Arithmetic Functions	
Function	**Description**
ABS	Converts the associated value to an absolute value.
ATAN	Returns the arc tangent of an angle.
CINT	Converts x to an integer.
COS	Returns the cosine of an angle.
FIX	Returns truncated integer part of argument.
INT	Converts a variable's value to an integer.
LOG	Returns natural logarithm of expression.
LOG10	Returns base 10 logarithm of expression.
SGN	Returns sign of an argument.
SIN	Returns sine of and angle.
SQR	Returns square root of an expression.
TAN	Returns the tangent of an angle.

String Functions	
Function	**Description**
ASC	Returns a numeric value that is the ASCII code for the first character of the string.
CHR$	Converts an ASCII code to its equivalent character to display on the terminal.
HEX$	Converts a long integer to a hexadecimal ASCII string.
INKEY$	Returns one character read from the serial input port's buffer.
INSTR	Provides the location of a substring within a string.
LCASE$	Converts a variable to the lower case value.
LEFT$	Returns a string of the leftmost characters of the string.
LEN	Returns the number of characters in the string.
LTRIM$	Trims leading and trailing spaces.
MID$	Returns a substring of a string expression that begins at a specified offset location.
RIGHT$	Returns the rightmost characters of the string.
SPACE$	Returns a string of spaces.
STR$	Returns the string representation of a numeric expression.
STRING$	Returns a string of common characters.
UCASE$	Converts a value to upper case.
VAL	Returns the numerical value of a string.

2. QUICK REFERENCE

This section contains functions, parameters, statements and variables for ServoBASIC Plus. Below is a summary of the list of instructions.

NOTE

The default value for parameters designates the value of the instruction at power on and at program start. A numeric value designates the power on/program start default value of a parameter. Default values designated by set up are initialized to the value in the *PARAMS* ***section of the program. Parameters may also be modified during program execution but will always retain their power on value at the start of program execution.***

Name	Type	Default
ABORT.MOTION	Statement	
ABS	Function	
ACCEL.GEAR	Variable (integer)	16,000,000
ACCEL.RATE	Variable (integer)	10,000
ACCEL.TYPE	Variable (integer)	0
AD.OFFSET	Parameter (float)	Set up
ADF0	Parameter (float)	Set up
ANALOG.IN	Variable (float R/O)	
ANALOG.OUT	Variable (float)	0
ARF0	Parameter (float)	Set up
ARF1	Parameter (float)	Set up
ASC	String Function	
ATAN	Function	
AUTOSTART	Parameter (integer)	Set up
AXIS.ADDR	Variable (integer R/O)	
AXIS.INTR	Variable (integer R/O)	
BEEP	Statement	
BLKTYPE	Parameter (integer)	Set up
CALL	Statement	
CCWINH	Variable (integer R/O)	
CCWOT	Variable (integer)	0
CHR$	String Function	
CINT	Function	
CLS	Statement	
CMDGAIN	Parameter (float)	Set up
CONST	Statement	
COS	Function	
COUNTER	Variable (integer)	
COUNTSPERREV	Variable (integer)	4096
CWINH	Variable (integer R/O)	
CWOT	Variable (integer)	
DACMAP	Parameter (integer)	Set up
DACMON	Variable (float R/O)	
DECEL.GEAR	Variable (integer)	16,000,000
DECEL.RATE	Variable (integer)	10.000

Name	Type	Default
DIM	Statement	
DIR	Variable (integer)	0
DMF0	Parameter (float)	Set up
ENABLE	Variable (integer)	Set up
ENABLED	Variable (integer R/O)	
ENC.FREQ	Variable (float R/O)	
ENC.IN	Variable (integer)	1024
ENC.OUT	Variable (integer)	0
ENCPOS	Variable (integer)	
END	Statement	
ERR	Variable (integer R/O)	
ERRVAL	Variable (float R/O)	
ERRVAR	Variable (integer R/O)	
EXIT	Statement (integer)	
FAULTCODE	Variable (integer)	
FIX	Function	
FOR…NEXT	Statement (integer)	
FVEL.ERR	Variable (float R/O)	
FWV	Variable (integer R/O)	
GEARERROR	Variable (integer)	
GEARING	Variable (integer)	0
GEARLOCK	Variable (integer R/O)	
GO.ABS	Statement	
GO.HOME	Statement	
GO.INCR	Statement	
GO.VEL	Statement	
GOSUB…RETURN	Statement	
GOTO	Statement	
HEX$	String function	
ICMD	Variable (float R/O)	
IFB	Variable (float R/O)	
IF…THEN…ELSE	Statement	
ILC	Parameter (integer)	Set up
ILMT.MINUS	Parameter (integer)	Set up
ILMT.PLUS	Parameter (integer)	Set up
INDEX.DIST	Variable (integer)	4096
INKEY	String function	
IN.POS.LIMIT	Variable (integer)	5
IN.POSITION	Variable (integer R/O)	
INP*n*	Variable (integer R/O)	
INPUT	Statement	
INPUTS	Variable (integer R/O)	
INSTR	String function	
INT	Function	
INTERRUPT	Statement	
INTR.{Source Label}	Variable (integer)	0
IPEAK	Variable (float R/O)	
ITF0	Parameter (float)	Set up
IT.FILT	Variable (float R/O)	
IT.THRESH	Parameter (float)	Set up

Name	Type	Default
KPP	Parameter (float)	Set up
KTEFF	Variable (float)	Set up
KVFF	Parameter (float)	Set up
KVI	Parameter (float)	Set up
KVP	Parameter (float)	Set up
LANFLT	Variable (float)	0
LANINT	Variable (integer)	0
LANINTERRUPT	Statement	
LCASE$	String function	
LEFT$	String function	
LEN	String function	
LOG	Function	
LOG10	Function	
LOGGEDON	Variable (integer)	
LTRIM$	String function	
MID$	String function	
MOD	Arithmetic operator	
MODEL	Variable (integer R/O)	
MOVING	Variable (integer R/O)	
ON ERROR GOTO	Statement	
OUT*n*	Variable (integer)	1
OUTPUTS	Variable (integer)	4095
PAUSE	Statement	
PAUSE.TIME	Variable (float)	1.00
POLECOUNT	Parameter (integer)	Set up
POS.CHK*n*	Variable (integer)	0
POS.CKK*n*.OUT	Variable (integer)	0
POS.COMMAND	Variable (integer)	
POS.ERROR	Variable (integer R/O)	
POS.ERROR.MOVING	Variable (integer)	0
POS.ERROR.STOPPED	Variable (integer)	0
POSITION	Variable (integer R/O)	
PRINT	Statement	
PULSES.IN	Variable (integer)	1,000
PULSES.OUT	Variable (integer)	1,000
PWM12	Variable (float)	
RATIO	Variable (float)	1.00
REG.DIST	Variable (integer)	4096
REG.ENCPOS	Variable (integer R/O)	
REG.FLAG	Variable (integer)	
REG.FUNC	Variable (integer)	0
REG.MODE	Variable (integer)	0
REG.POS	Variable (integer R/O)	
REG.RESPOS	Variable (integer R/O)	
REM	Statement	
REPOS	Variable (integer R/O)	
RESTART	Statement	
RIGHT$	String function	
RUN.SPEED	Variable (float)	1.000
RVEL	Variable (float R/O)	

Name	Type	Default
SGN	Function	
SIN	Function	
SPACES$	String function	
SQR	Function	
STATUS	Variable (integer R/O)	
STEPDIR	Variable (integer)	
STOP	Statement	
STR$	String function	
STRING$	String function	
SUB	Statement	
SWAP	Statement	
TAN	Function	
TARGET.POS	Variable (integer)	0
TIME	Variable (float)	
TMENABLE*n*	Variable (integer)	0
TMOUT*n*	Variable (integer R/O)	
TMRSET	Statement	
UCASE$	String function	
UPD.MOVE	Statement	
VAL	String function	
VEL.CMD	Variable (float R/O)	
VEL.ERR	Variable (float R/O)	
VELOCITY	Variable (float R/O)	
WHEN	Statement	
WHEN.ANALOG.IN	Variable (float R/O)	
WHEN.DACMON	Variable (float R/O)	
WHEN.ENCPOS	Variable (integer R/O)	
WHEN.ICMD	Variable (float R/O)	
WHEN.IFB	Variable (float R/O)	
WHEN.PCMD	Variable (integer R/O)	
WHEN.POS	Variable (integer R/O)	
WHEN.RESPOS	Variable (integer R/O)	
WHEN.RVEL	Variable (float R/O)	
WHEN.TIME	Variable (float R/O)	
WHEN.VELCMD	Variable (float R/O)	
WHILE…WEND	Statement	
WVSHP	Variable (integer R/O)	

Note: Parameters with default value 'set up' have default values established in the configuration set up of Motion Dialogue.

3. SERVOBASIC PLUS INSTRUCTIONS

This section is an alphabetical reference to ServoBASIC Plus instructions:

- commands
- functions
- string functions
- parameters
- statements
- string variables
- variables

The name and type of each instruction is listed at the top of the page. The instruction is then described based on the following categories:

Syntax The complete notation of the instruction

Description Pertinent information about the instruction and its use in ServoBASIC Plus

Example Possible use of the instruction in a program

See Also Other ServoBASIC Plus commands that are similar to this particular instruction

ABORT.MOTION

Statement

Syntax **ABORT MOTION**

Description **ABORT MOTION** stops motor motion while allowing continued program execution. Deceleration is determined by the motor torque capability in conjunction with the current limit parameters.

Example Program line

```
‘This program segment commands the motor at constant
‘velocity until input 1 goes to a logic 0,
‘then the motor is commanded to stop.
‘Trapezoidal velocity profile
ACCEL.TYPE = 0
‘Set acceleration rate equal to 12,000 rpm/sec
ACCEL.RATE = 12000
‘Set deceleration rate equal to 12,000 rpm/sec
DECEL.RATE = 12000
‘Set run speed equal to 120 rpm
RUN SPEED = 120
GO.VEL
WHEN INP1 = 0, ABORT.MOTION
PRINT “MOVE ABORTED”
END
```

See Also ILMT.MINUS, ILMT.PLUS

ABS

Function

Syntax **ABS**

Description **ABS** converts the associated value (*x*) to an absolute value. If the value is negative, it is converted to a positive value. If the value is positive, it is not changed.

Enter the argument (the value) in parentheses immediately following the term ABS.

Example Program line

```
‘This program segment prints the absolute
‘value of INT1.
INT1 = 1000
PRINT ABS (INT1)
```

The following value is printed: 1000

ACCEL.GEAR

Variable (integer)

Syntax *x* = **ACCEL.GEAR** OR **ACCEL.GEAR** = *x*
Where *x* is controlled acceleration in rpm/sec

Range 1 to 16,000,000 rpm/sec

Resolution 1 rpm/sec

Default 16,000,000

Description **ACCEL.GEAR** sets the commanded acceleration rate when gearing is turned ON. The specified acceleration rate is used until **GEARLOCK** is achieved.

NOTE

ACCEL.GEAR ***is independent of*** *DECEL.GEAR****. Each variable must be set, independently, to the appropriate value for the desired motion.***

Set **ACCEL.GEAR** prior to turning gearing ON.

NOTE

The control of acceleration is independent of the control of deceleration. Deceleration is set using the command *DECEL.RATE.*

Example Program line

```
‘Clear accumulated error
GEARERROR = 0
‘Set up acceleration rate
ACCEL.GEAR = 1200
‘set up deceleration rate
DECEL.GEAR = 1200
‘turn on Gearing
WHEN INP1=0, CONTINUE
‘When INP1 goes low
GEARING = 1
‘Wait for Gearlock
WHILE GEARLOCK=0 : WEND
‘Setup for Correction Move
INDEX.DIST = GEARERROR
‘Perform Phase Correction
GO.INCR
```

See Also GEARING, DECEL.GEAR, GEARERROR, GEARLOCK

ACCEL.RATE

Variable (integer)

Syntax **ACCEL.RATE** = x
Where *x* is the desired acceleration rate in rpm/sec

Range 1 to 16,000,000 rpm/sec

Resolution 1 rpm/sec

Default 10,000

Description **ACCEL.RATE** (Acceleration Rate) sets the maximum commanded acceleration rate when speed is increased. During S-Curve velocity profiles **ACCEL.RATE** designates the average acceleration with the peak acceleration at twice **ACCEL.RATE.**

NOTE

ACCEL.RATE ***is independent of*** *DECEL.RATE****. Each variable must be set independently to the appropriate value for the desired motion.***

Set **ACCEL.RATE** prior to issuing any motion command statement. Acceleration rate can be updated using the **UPD.MOVE** statement.

NOTE

The control of acceleration is independent of the control of deceleration. Deceleration is set using the command *DECEL.RATE****.***

Example Program line

```
'Set to use trapezoidal velocity profiles
ACCEL.TYPE = 0
'Set run speed equal to 1,800 rpm
RUN.SPEED = 1,800
'Set acceleration rate equal to 10,000 rpm/sec
ACCEL.RATE = 10,000
'Set deceleration rate equal to 14,000 rpm/sec
DECEL.RATE = 14,000
'Begin acceleration of motor
GO.VEL
TIME = 0
'Wait one second
WHILE TIME < 1
WEND
'Command zero velocity
RUN.SPEED = 0
'Begin deceleration of motor
GO.VEL
```

See Also ACCEL.TYPE, DECEL.RATE, GO.ABS, GO.HOME, GO.INCR, GO.VEL, RUN.SPEED, UPD.MOVE

ACCEL.TYPE

Variable (integer)

Syntax **ACCEL.TYPE** = x

Range 0 for constant acceleration motion (trapezoidal) profiles
1 for S-curve velocity profiles

Default 0

Description **ACCEL.TYPE** (Acceleration Type) determines the use of constant acceleration of S-Curve velocity profiles.

NOTE

S-Curve velocity profiles are only supported for positioning moves.

Specify **ACCEL.TYPE** prior to issuing motion commands.

NOTE

UPD.MOVE ***does not work if*** *ACCEL.TYPE* ***= 1. S-Curve velocity profiles cannot be modified once the move has started.***

See Also ABORT.MOTION, ACCEL.RATE, DECEL.RATE, GO.ABS, GO.HOME, GO.INCR, RUN.SPEED, UPD.MOVE

AD.OFFSET

Parameter (float)

Syntax **AD.OFFSET** = *x*

Range -12.50 to +12.50 volts

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Description **AD.OFFSET** is the level of a signal summed with the digitized value of the analog input channel, in volts.

AD.OFFSET can be preconfigured for power on default. This is accomplished in the servo set up parameter section.

See Also ANALOG.IN, BLKTYPE, CMDGAIN

ADF0

Parameter (float)

Syntax **ADF0** = x where x is the filter's corner frequency in Hz

Range 0.011 to 12,222,726 (Hz)

NOTE

The default value of this parameter is set during the configuration set up of Motion Dialogue.

Description **ADF0** sets the analog input channel's filter corner frequency. The purpose of the filter is to attenuate the high frequency components from the digitized input signal.

Example Program line

```
'This program segment sets ADF0 based on user input
DIM ADFILTER AS FLOAT
INPUT "Break frequency of analog input channel filter (Hz)"; ADFILTER
ADF0 = ADFILTER
```

See Also ANALOG.IN, BLKTYPE

ANALOG.IN

Variable (float) (read only)

Syntax *x* = **ANALOG.IN** (value of J57-1 relative to J57-2)

Range -12.50 to +12.50 volts

Description **ANALOG.IN** (Analog input) contains the digitized value of the analog input channel (value of J57-1 relative to J57-2), in volts.

Example Program line

```
PRINT "Place a 1 volt dc signal into the analog channel"
WHILE 1 = 1
  PRINT "The Analog Input signal measures",
  ANALOG.IN, "volts"
  PAUSE
WEND
```

The program continuously prints out the message:
The Analog Input signal measures 1.000000 volts

See Also ANALOG.OUT, BLKTYPE, CMDGAIN, ADF0

ANALOG.OUT

Variable (float)

Syntax **ANALOG.OUT** = x

Range -5.00 volts to +5.00 volts

Default 0

Description **ANALOG.OUT** (Analog Output) sets the voltage level of the analog output channel.

The analog output signal can be controlled within a program using the **ANALOG.OUT** variable. However, **DACMAP** must be zero to configure the output channel for the **ANALOG.OUT** variable.

Example Program line

```
DIM OUTPUT_SIGNAL AS FLOAT
‘Select ANALOG.OUT as the source of analog
‘output signal
DACMAP = 0
WHILE 1 = 1
    INPUT “Desired output voltage”; OUTPUT_SIGNAL
    ANALOG_OUT = OUTPUT_SIGNAL
WEND
```

See Also DACMAP

ARF0

Parameter (float)

Syntax **ARF0** = *x* where *x* is the corner frequency in Hz

Range 0.011 to 12,222,726 (Hz)

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Description **ARF0** is the stage 0 anti-resonant single order low pass filter corner frequency.

ARF0 is the corner frequency in Hz of one of two single order low pass anti-resonant filters. The purpose of these anti-resonant filters is to attenuate the velocity loop gain at the mechanical resonant frequency.

CAUTION

A value is assigned to ARF0 by the controller Set Up feature (one of the menu options in Motion Dialogue). This value is appropriate for most applications and the user normally should not be concerned with setting it in the application program.

Example Program line

```
'This program segment sets ARF0 based on user input
DIM ANTIRES0 AS FLOAT
INPUT "Break frequency of first anti-resonant filter (Hz)"; ANDIRES0

ARF0 = ANTIRES0
```

See Also ARF1, ILC, KVP, KVI, KPP, KVFF

ARF1

Parameter (float)

Syntax **ARF1** = *x* where *x* is the corner frequency in Hz

Range 0.011 to 12,222,726 (Hz)

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Description **ARF1** is the stage 1 anti-resonant single order low pass filter corner frequency.

ARF1 is the corner frequency in Hz of one of two single order low pass anti-resonant filters. The purpose of these anti-resonant filters is to attenuate the velocity loop gain at the mechanical resonant frequency.

CAUTION

A value is assigned to** ARF1 **by the controller Set Up feature (one of the menu options in Motion Dialogue). This value is appropriate for most applications and the user normally should not be concerned with setting it in the application program.

Example Program line

```
'This program segment sets ARF1 based on user input
DIM ANTIRES1 AS FLOAT
INPUT "Break frequency of second anti-resonant filter (Hz)"; ANTIRES1
```

See Also ARF0, ILC, KVP, KVI, KPP, KVFF

ASC

String Function

Syntax **ASC**(*x*)

Description **ASC** (string expression) returns a decimal numeric value that is the ASCII code for the first character of the string expression(*x$*).

If the string begins with an uppercase letter, the value will be between 65 and 90. If the string begins with a lower-case letter, the range is between 97 and 122. Values 0 to 9 return 48 to 57.

NOTE

ASCII codes are listed in the table located in Appendix B: ASCII Codes.

Example Program line

```
DIM X$ AS STRING
X$ = "TEN"
PRINT ASC(x$)
```

The program above prints the decimal value 84. 84 is the ASCII code for the letter T.

See Also CHR$

ATAN

Function

Syntax **ATAN** (*x*)

Description **ATAN** (arc tangent) returns the arc tangent of *x* in radians. *x* may be any numeric type. To convert from degrees to radians, multiply by π/180.

Range –π/2 to π/2.

Example π /180 = 0.01745329

AUTOSTART

Parameter (integer)

Syntax **AUTOSTART** = x
x = 0 – **AUTOSTART** disabled, sign-on message displayed
x = 1 – **AUTOSTART** enabled, sign-on message displayed
x = 3 – **AUTOSTART** disabled, sign-on message suppressed
x = 4 – **AUTOSTART** enabled, sign-on message suppressed

NOTE

The default value of the parameter is set during the configuration set up in Motion Dialogue.

Description **AUTOSTART** specifies the automatic execution of a user program as soon as the servocontroller has AC control power applied.

AUTOSTART should be activated using the variable set/reset window, to initiate execution of a downloaded program upon the application of AC control power.

To execute a program with **AUTOSTART** turned Off requires a RUN command from Motion Dialogue.

To stop program execution when **AUTOSTART** is enabled requires a stop motion command from Motion Dialogue.

Changing the **AUTOSTART** variable within a program has no effect on the power-up default setting.

AXIS.ADDR

Variable (integer) (read only)

Syntax x = **AXIS.ADDR**

Range 1 to 255 (0 reserved for PacLAN globals)
(255 is the default RS-232 address)

Description This variable can be inspected to insure the mulitdrop subsystem has been configured for the proper address by switch S1.

NOTE

Refer to Section 3.1.1 of the SC750 Installation and Hardware Reference Manual to configure Dip Switch S1 for appropriate multidrop address.

Example Program line

```
'This program will toggle OUT1 if the controller
'is not configured for the proper address
'This controller is intended to be located
'at multidrop address 12
IF AXIS.ADDR < > 12 THEN
    PAUSE.TIME = 1
    WHILE 1 = 1
        OUT1 = 0
        PAUSE
        OUT1 = 1
        PAUSE
    WEND
ELSE
    OUT1 = 1
END IF
```

See Also LOGGEDON

AXIS.INTR

Variable (integer) (read only)

Syntax x = **AXIS.INTR**

Range x = 1 to 255

Description **AXIS.INTR** indicates the axis address number of the source address of a LANINTERRUPT.

Use **AXIS.INTR** when a PacLAN installation controller can have multiple interrupt sources and it is necessary to determine the source axis of the interrupt.

See Also LANINTERRUPT

BEEP

Statement

Syntax **BEEP**

Description **BEEP** transmits a speaker beep command to the serial port.

BLKTYPE

Parameter (integer)

Syntax **BLKTYPE** = x

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Description **BLKTYPE** specifies configuration as a position, velocity, or torque block. **BLKTYPE** allows the SC750 to be operated as a torque controller, an analog velocity controller, or an analog position controller with analog command in addition to the software controlled position controller (default):

BLKTYPE	Servo Configuration
BLKTYPE = 0	Torque block - analog command
BLKTYPE = 1	Velocity block – analog command
BLKTYPE = 2	Position loop under software control (default)
BLKTYPE = 3	Position loop – analog command

For a block diagram of the controller configuration, refer to Appendix A: Servo Loop. This also provides alternative **BLKTYPE** settings.

NOTE

The servocontroller is disabled whenever BLKTYPE is changed. Set ENABLE to 1 to re-enable the controller.

When used in any of the analog modes, the analog control is the differential voltage applied to the Analog In (+) and Analog In (-) inputs (J57-1 and J57-2 respectively). Scaling for the analog control input is determined by **CMDGAIN** as:

BLKTYPE	Scale Factor
BLKTYPE = 0	2.00 * CMDGAIN amps/volt *
BLKTYPE = 1	1.00 * CMDGAIN krpm/V
BLKTYPE = 3	0.025 * CMDGAIN revs/volts

**To achieve the indicated scale factor, velocity servo loop gains must be set as follows:*

KVP = 1.0 KVI = 0

CMDAGIN (default is defined within the parameter set up program section – typically 1.00) can be modified to customize the scale factor as required.

Example Program line

```
'This segment sets BLKTYPE based upon user input
    DIM CHOICE AS INTEGER
    PROMPT:
    PRINT ""
    PRINT "Desired function:"
    PRINT "Analog Torque Controller (0)"
    PRINT "Analog Velocity Controller (1)"
    PRINT "Software Controlled Positioning System (2)"
    PRINT "Analog Position Controller (3)"
    PROMPT:
    PRINT "Choice"
    CHOICE = -1
    WHILE (CHOICE < 0) OR (CHOICE > 3)
        INPUT CHOICE
    WEND
    BLKTYPE = CHOICE
    ENABLE = 1
```

See Also CMDGAIN

CALL

Statement

Syntax **CALL** *subroutine name*

Description **CALL** transfers program execution to a BASIC subroutine.

Example Program line

```
'LOCATE SUBROUTINES EXECUTED BY THE CALL STATEMENT
'AFTER THE END STATEMENT IN THE MAIN PROGRAM

PRINT "This is Main Program"
```

See Also SUB

CCWINH

Variable (integer) (read only)

Syntax x = **CCWINH**

Value: 0 or 1

Description **CCWINH** indicates the current state of the CCWINH (INH-) input.

See Also CCWOT, CWOT, CWINH

CCWOT

Type

Syntax **CCWOT** = x

Range -134,217,728 to 134,217,727 resolver steps

Default 0

Description **CCWOT** sets the counter-clockwise over-travel limit. When the position of the motor becomes counter-clockwise (more negative) on this threshold, a counter-clockwise over-travel interrupt occurs (if enabled).

CCWOT should be set before the **CCWOT** interrupt is enabled.

NOTE

Do not change POS.COMMAND after CCWOT, CWOT, TARGET.POS or POS.CHKn have been programmed. These absolute position variables change value if the electrical home position is changed.

See Also CWOT

CHR$

String Function

Syntax **CHR$**(*n*)

Description **CHR$** converts an ASCII code to its equivalent character. *n* is a value from 1 to 255. **CHR$**(0) returns a null string.

NOTE

ASCII codes are listed in the table located in Appendix B: ASCII Codes.

Example Program line

```
DIM A$ AS STRING
A$ = CHR$(66)
```

The upper case letter B is printed.

See Also ASC

CINT

Function

Syntax **CINT**(*x*)

Range *x* = -32,768 to 32,767

Description **CINT** converts *x* to an integer by rounding the fractional portion. If the fractional portion is greater than 0.5, *x* is rounded up to the next integer, if less than 0.5, *x* is rounded down to the existing integer portion.

Example Program line

```
PRINT CINT (45.67)
The value 46 is printed.

PRINT CINT (-12.11)
The value 12 is printed.

PRINT CINT (VELOCITY)
The value 1000 is printed. The motor is moving at 1000 rpm.
```

See Also INT, FIX

CLS

Statement

Syntax **CLS**

Description This command transmits 40 line feed characters (ASCII code=10) to the serial port. **CLS** clears the screen display of a terminal.

CMDGAIN

Parameter (float)

Syntax **CMDGAIN** = *xx.xx* where *xx.xx* can be negative

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Description **CMDGAIN** controls the scale factor of the analog input signal.

CAUTION

CMDGAIN ***is automatically set to 1.0 any time BLKTYPE is changed.***

CMDGAIN is a floating-point variable that sets the command gain of the analog input (voltage from J57-1 to J57-2) for **BLKTYPE** equal to:

- 0 (Analog torque block)
- 1 (Analog velocity block)
- 3 (Analog position loop)

Scaling is defined as:

- **BLKTYPE** = 0 (Analog torque block)

NOTE

To achieve indicated scale factor, velocity servo loop gains must be set as follows:

KVP = 1.0 ***KVI = 0***

Motor Current (amps) = 2 * CMDGAIN * analog-in (volts)

If the analog input voltage is one volt and GMDGAIN is equal to 1.5, the commanded motor current equals 2 * 1.5 * 1 = 3 amps.

- **BLKTYPE** = 1 (Analog velocity block)

  Motor Velocity (rpm) = 1000 * CMDGAIN * analog-in (volts)

  Example If the analog input voltage is one volt and if CMDGAIN is equal to 2.0, then the commanded motor speed will equal 1000 * 2.0 * 1 = 2000 rpm.

- **BLKTYPE** = 3 (Analog position block)

  Motor Position (rslvr counts) = (4096/40) * CMDGAIN * analog-in (volts)

If the analog input voltage is positive one volt and CMDGAIN is equal to 1.0, the motor rotates 1/40 revolution from the resolver zero position. The zero position is the closest position where the resolver angle is zero when setting **BLKTYPE** equal to 3.

NOTE

CMDGAIN ***can be negative. This inverts the sign of the associated block.***

Example

Program line

```
DIM SCALEFACTOR AS FLOAT
INPUT "Velocity scale factor (krpm/volt)"
SCALEFACTOR

BLKTYPE = 1
CMDGAIN = SCALEFACTOR
ENABLE = 1
```

See Also BLKTYPE, DACMAP

CONST

Statement

Syntax **CONST** *constant name = value*

Description **CONST** declares numeric constants to be used in place of numeric values. Unlike variable names declared by the **DIM** statement, constants can assume only one value in a program.

Example Program line

```
'Set variable x equal to 1800
CONST x = 1800
"Reset variable x to 2000
CONST x = 2000
RUN.SPEED = x
```

This program changes the value of RUN.SPEED from 1000 to 2000.

COS

Function

Syntax **COS**(*x*)

Description **COS**(*x*) (Cosine) returns the cosine of *x* in radians.

To convert degrees to radians, multiply by π/180 (0.017453)

Example Program line

```
X = 2 * COS(.4)
PRINT X
```

This program prints the value 1.842122.

See Also SIN, TAN

COUNTER

Variable (integer)

Syntax x = **COUNTER**

OR

COUNTER = x where x is the starting count value

Range 0 to 32,727

Description **COUNTER** specifies the current count of the hardware event counter feature using the discrete Input 16. This variable can also be preset to a starting count value (typically zero).

COUNTER contains the number of cycles input by a clock source applied to Input 16. This feature provides the ability to count events detected with the appropriate instrumentation and interface electronics.

COUNTER is cleared either by the hardware counter reset, Input 15, or by clearing the software variable. The maximum input frequency that can be counted is 10 kHz.

See Also INP15, 1NP16, INPUTS

COUNTSPERREV

Variable (Integer)

Syntax *x* = **COUNTSPERREV**
Or
COUNTSPERREV = *x*

Range Valid values are only: 4096, 8192, 16384, 32768 or 65536

Default 4096

Description **COUNTSPERREV** specifies the resolution of the position control. The default value is 4096 resolver steps per motor revolution (5.27 arc-min). This is consistent with earlier versions of the SC750. This variable can be changed to 8192 (2.63 arc-min), 16384 (1.31 arc-min), 32768 (0.66 arc-min) or 65536 (0.33 arc-min).

NOTE

This variable controls the resolution, or granularity of the positioning control. It does not affect accuracy.

See Also INDEX.DIST, POSITION, POS.COMMAND, TARGET.POS

CWINH

Variable (integer) (read only)

Syntax *x* = **CWINH**

Value 0 or 1

Description **CWINH** indicates the current state of the CWINH (INH+) input.

See Also CCWOT, CWOT, CCWINH

CWOT

Variable (integer)

Syntax **CWOT** = *x*

Range -134,217,728 to 134,217,727 resolver steps

Description Set **CWOT** before enabling the **CWOT** interrupt.

NOTE

Do not change** POS.COMMAND **after** CCWOT, CWOT, TARGET.POS **or** POS.CHKn **have been programmed. These absolute position variables change value if the electrical home position is changed.

See Also CCWOT

DACMAP

Parameter (integer)

Syntax **DACMAP** = *x*
where *x* selects the desired signal to monitor as indicated by the following table.

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Monitor #	Mnemonic	Description	DAC Out Units
0	ANALOG.OUT	ServoBASIC parameter	1 V/V
1	RVEL	Resolver Velocity	1 V/krpm
2	VEL.CMD	Net Velocity Command	1 V/krpm
3	VEL.ERR	Velocity Error	1 V/krpm
4	FVEL.ERR	Filtered Velocity Error	1 V/krpm
5	POSITION	Resolver Position	40 V/Rev
6	POS.ERR	Position Error	40 V/Rev
7	POS.COMMAND	Net Position Command	40 V/Rev
8	ICMD	Torque Current Command	0.5 V/Amp
9	IFB	Measure Torque Current	0.5 V/Amp
10	FAD	A/D After Filer	1 V/V
11	ENC.FREQ	Encoder Velocity	0.1 V/kHz
12	ENC.POS	Encoder Position	40 V/4096 Counts
13	IT.FILT	IT circuit output	0.5 V/amp

Description **DACMAP** specifies the signal sent to the monitor **DAC** driving the analog output channel.

If **DACMAP** equals 0, the monitor **DAC** output equals the value specified by **ANALOG.OUT**; otherwise **DACMAP** specifies the signal sent to the monitor **DAC**.

Select the desired variable to be monitored within you program. Modify the scale factor, if necessary, using **DMGAIN**. (Changing **DACMAP** resets **DMGAIN** to its default value of 1.

Polarity of the output signal is positive for clockwise torque/velocity/position (except for position monitoring).

If the selected signal exceeds the ±5 volt range of the analog output channel, the analog output is clamped at approximately either ±5V or –5V.

For position signals, the output signal “rolls over.” For signals of increasing value, this means increasing to +5 volts then rolling over to –5 volts and again increasing to +5 volts. For decreasing position, the output decreases to –5 volts, then rolls over to +5 volts and then continues to decrease.

Example Program line

```
‘Select ANALOG.OUT as the output signal source
DACMAP = 0
‘Set analog output equal to 1 volt
ANALOG.OUT = 0
‘Monitor the velocity error. The default factor
‘is 1 V/krpm
DACMPAP = 3
```

See Also ANALOG.OUT, DMGAIN, DMF0

DACMON

Variable (float) (read only)

Syntax *x* = **DACMON**

Description **DACMON** contains the value of the selected, filtered variable output to analog output channel.

DACMON provides the ability to sense the signals that are output to the analog output channel, after they have been processed a low pass filter controlled by the parameter DMF0, the filter’s corner frequency.

DACMAP Selects the source for **DACMON** and reports the value in natural units. That is, speed variables are in rpm, position variables are in steps (1 Rev = 4096 steps), current is in amps, voltages are in volts, and encoder variables are in counts and counts-per-second.

Example This program segment sends **ICMD** (commanded motor current) to the analog output channel. A low pass filter with a 10 Hz corner frequency is specified for the output channel. The filtered current feedback signal times the system torque constant KTEFF is then output to the serial I/O port to report shaft torque.

Program line

```
‘Set up output signal selection to monitor
‘command motor current
DACMAP = 8
‘Select 10 Hz corner frequency on the
‘output channel filter
DMF0 = 10.0
‘Output the filtered shaft torque every second
PAUSE.TIME = 1
WHILE 1 = 1
    PAUSE
    PRINT “Filtered shaft torque is
“,DACMON*KTEFF;
        “lb-in”
WEND
```

See Also DMF0, DACMAP

DECEL.GEAR

Variable (integer)

Syntax *x* = **DECEL.GEAR**
or
DECEL.GEAR = *x*
Where *x* is the controlled deceleration rate I rpm/sec

Range 1 to 16,000,000 rpm/sec

Resolution 1 rpm/sec

Default 16,000,000

Description **DECEL.GEAR** sets the deceleration rate commanded when gearing is turned OFF. The specified deceleration rate is used until geared motion had stopped.

NOTE

ACCEL.GEAR is independent of DECEL.GEAR. Each variable must be set, independently, to the appropriate value for the desired motion.

Set **DECEL.GEAR** prior to turning gearing OFF.

Example Program line

```
'Clear accumulated error
GEARERROR = 0
'Set up acceleration rate
ACCEL.GEAR = 1200
'Set up deceleration rate
DECEL.GEAR = 1200
'Turn on Gearing
WHEN INP1=0, CONTINUE
'When INP1 goes low
GEARING = 1
'Wait for Gearlock
WHILE GEARLOCK=0 : WEND
INDEX.DIST= GEARERROR
'Perform Phase Correction
GO.INCR
```

See Also ACCEL.GEAR, GEARING, GEARERROR, GEARLOCK

DECEL.RATE

Variable (integer)

Syntax **DECEL.RATE** = *x*
where *x* is the desired deceleration rate in rpm/sec

Range 1 to 16,000,000 rpm/sec

Resolution 1 rpm/sec

Default 10,000

Description **DECEL.RATE** (deceleration rate) sets the maximum rate commanded when speed is decreased. During s-curve velocity profiles, **DECEL.RATE** designates the average deceleration with the peak deceleration being twice **DECEL.RATE**.

NOTE

ACCEL.RATE ***is independent of*** *DECEL.RATE****. Each variable must be independently set to the appropriate value for the desired motion.***

Set **DECEL.RATE** prior to issuing any motion command statement. Deceleration rate can be updated using **UPD.MOVE**.

Example Program line

```
'Use trapezoidal velocity profiles
ACCEL.TYPE = 0
'Set run speed equal to 1,800 rpm
RUN.SPEED = 1800
'Set acceleration rate equal to 10,000 rpm/sec
ACCEL.RATE = 10,000
'Set deceleration rate equal to 14,000 rpm/sec
DECEL.RATE = 14,000
'Begin acceleration of motor
GO.VEL
TIME = 0
'Wait one second
WHLE TIME < 1
WEND
'Command zero velocity
RUN.SPEED – 0
'Begin deceleration of motor
GO.VEL
```

See Also ACCEL.RATE, ACCEL.TYPE, GO.ABS, GO.HOME, GO.INCR, GO.VEL, RUN.SPEED, UPD.MOVE

DIM

Statement

Syntax

1. Typical data allocation

   DIM variable [(subscript)] [,variable ({subscript}),…]

2. Non-volatile (NV) data allocation

   DIM variable [(subscript)] [,variable ({subscript}),…]

   AS type NV

In either type of data allocation, typical or NV, “variable” is the user defined variable name. The optional subscript designates the number of elements of an array variable. Type is either FLOAT, INTEGER, STRING [*LENGTH], SINGLE, DOUBLE or LONG.

NOTE

The [*length] option for the STRING variable type permits defining the maximum number of characters permitted in a STRING variable. The default is 34 characters. The maximum is 254 characters.

Description

DIM specifies and allocates storage for variables and arrays. The statement is also used to designate variables to be treated as non-volatile.

When the controller power is cycled, non-volatile variables retain the data present when the controller was powered down. All other variables reset to zero when control power is applied.

All non-volatile (NV) memory variables must be defined prior to typical data allocation. DIM statements specifying NV variables must occur prior to a non-NV variable DIM statement.

Non-volatile (NV) variables are to be used to save data that may change periodically (i.e. weekly, monthly) but are desired to be retained in memory after the controller power is cycled. A possible usage could be a cut-to-length process where the part’s length is modified for different productions runs.

CAUTION

Non-volatile (NV) variables are not intended to be used for typical data storage since variables can potentially power up to a different value each time power is cycled.

There are essentially three fundamental variable types:

- FLOAT
- INTEGER
- STRING

INTEGER, LONG variables defined as LONG or INTEGER are processed as INTEGER (32 bit signed number). These variables or array element require 4 bytes of storage.

FLOAT, SINGLE, DOUBLE variables defined as SINGLE, DOUBLE, or FLOAT are processed as FLOAT (single precision IEEE floating point). These variables, or array elements, each require 4 bytes of storage. FLOAT variables have seven significant digits.

STRING: SRING variables provide storage for ASCII character string data. The default string length is 34 characters, each requiring one byte of storage. The string size can be modified using the [*length] option on the STRING type designation in the DIM definition.

Example 1. Typical data allocation (variables other than non-volatile)

Dimension x as a floating point variable:

```
DIM x AS FLOAT
```

Specify 2 arrays, each containing 16 integer variables:

```
DIM i (16), j (16) AS INTEGER
```

Allocate storage for three character strings A$, B$ and C$, each permitted a maximum default length of 34 characters:

```
DIM A$, B$, C$ AS STRING
```

Allocate an array, named part_num$, of 26 string variables, each element permitting storage of a maximum of 8 characters:

```
DIM part_num$ (26) AS STRING*8
```

2. Non-volatile (NV) data allocation

Declare y as an NV floating point variable:

```
DIM y as FLOAT NV
```

Define a 10-element array, k, of NV integer variables:

```
DIM k(10) AS INTEGER NV
```

DIR

Variable (integer)

Syntax **DIR** = *x*

Range 0, rotation is clockwise when looking at the motor shaft end.
1, rotation is counter-clockwise when looking at the motor shaft end.

Default 0

Description **DIR** (Direction) sets the direction the motor turns when a GO.VEL function is executed.

NOTE

DIR ***does not define direction for the*** *GO.INCR* ***or*** *GO.ABS* ***motion functions. The sign of*** *INDEX.DIST* ***defines direction for the*** *GO.INCR* ***function, and*** *TARGET.POS* ***relative to present*** *POSITION* ***defines direction for the*** *GO.ABS* ***function.***

See Also GO.VEL, RUN.SPEED

DMF0

Parameter (float)

Syntax **DMF0** = *x*……where *x* is the filter corner frequency in Hz

Range 0.011 to 12,222,276 (Hz)

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Description **DMF0** sets the analog output channel's single order low pass filter corner frequency.

DMF0 is the corner frequency in Hz of a single order low pass filter. The purpose of the filter is to attenuate the high frequency components from the analog output signal.

Example Program line

```
'This program segment set DMF0 based on user input
DIM DAFILTER AS FLOAT
INPUT "Break frequency of analog output channel filter (Hz)"; DAFILTER
DMF0 = DAFILTER
```

See Also DACMAP

DMGAIN

Parameter (float)

Syntax **DMGAIN** = *x* where *x* can be negative

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Description **DMGAIN** specifies the multiplicative scale factor applied to monitor DAC when DACMAP < > 0

DMGAIN specifies a multiplication factor applied to signals output to the analog output channel. The voltage range of signals selected by the DACMAP parameter can be modified and result in a modified overall scale factor.

Setting DACMAP to a new value will result in DMGAIN being reset to unity gain (DMGAIN = 1)

DMGAIN can be set to a negative value to invert the signal polarity.

Example Program line

```
‘Monitor motor current command. Default scale
‘factor is equal to 0.5 V/amp
DACMAP = 8
‘Boost the scale factor 4 to 1. The new overall
‘scale factor is 2 V/amp.
DMGAIN = 4
.
.
.
‘Monitor the velocity command. This resets
‘DMGAIN to its default value of 1.
DACMAP = 2
```

See Also ANALOG.OUT, DACMAP, DMF0

ENABLE

Variable (integer)

Syntax **ENABLE** = *x*

Range 0 = disable the drive
1 = enable the drive

Default 0

Description **ENABLE** allows or prevents power flow to the motor.
To enable, that is, allow power to flow to the motor, verify that the following conditions are all true:

1. Drive is not faulted. (Status display 0 or 8)
2. Enable input (J53-4 to J53-10) connected to I/O Rtn.
3. Enable variable set to 1.

See Also ENABLED

ENABLED

Variable (integer) (read only)

Syntax *x* = **ENABLED**

Value 0 = controller disabled
1 = controller enabled

Description **ENABLED** indicates whether controller is enabled.
To enable, that is, allow power to flow to the motor, verify that the following conditions are all true:

1. Drive is not faulted. (Status display 0 or 8)
2. Enable input (J53-4 to J53-10) connected to I/O Rtn.
3. Enable variable programmed.

Example Program line

```
IF ENABLED = 0 THEN
    PRINT "Motor not enabled"
ELSE
    PRINT "Motor enabled"
END IF
END
```

See Also ENABLE

ENC.FREQ

Variable (float) (read only)

Syntax *x* = **ENC.FREQ**

Range -3,000,000 to +3,000,000

Calculation…….ENC.FREQ =

$$\text{ENC.IN}\ \left(\frac{\text{ENCODERCOUNTS}}{\text{REV}}\right)\ \text{x}\ \text{ENCODERVELOCITY}\left(\frac{\text{REV}}{\text{MIN}}\right)\text{x}\left(\frac{\text{MIN}}{60\ \text{sec}}\right)\text{x}\ 4$$

Description **ENC.FREQ** text (Encoder Frequency) contains the frequency in quadrature pulses per second of the external encoder input averaged over a 128 msec interval for filtering.

The value returned is a floating point variable. To convert the value to an integer, use **CINT**.

Example Program line

```
ENC.IN = 1024
PRINT "ENC.FREQ =", ENC.FREQ
```

Assuming the master encoder is moving at a rate of 3000 rpm, the output for this program is:

```
ENC.FREQ = 204,800
```

See Also ENC.IN

ENC.IN

Variable (integer)

Syntax **ENC.IN** = *x*…….where *x* is the line count

Range 1 to 65,535

Default 1024

Description **ENC.IN** specifies the line count of the encoder being used for electronic gearing. Install an incremental quadrature encoder with differential line driver-type outputs on the master axis.

Connect the encoder output from the master axis to the controller (J52) and verify that it is set to the correct ENC.IN line count.

NOTE

ENC.IN must be specified before RATIO is set. ENC.IN is not used for electronic gearing using PULSES.IN/PULSES.OUT.

Example Program line
`ENC.IN = 1024`

Because this is quadrature encoder, there are actually 4096 quadrature counts per revolution of the encoder.

See Also ENC.FREQ, ENC.OUT, GEARING, RATIO

ENC.OUT

Variable (integer)

Syntax **ENC.OUT** = *x*……where *x* is the desired pulses/rev.

Range 0, 500, 512, 1000, 1024, 200, 2048, 4096, 16384

Default 0

Description **ENC.OUT** selects the direction of the encoder ports and selects emulated encoder line count when the bi-directional encoder ports are set to "TRANSMIT.'

If **ENC.OUT** is set to zero then the encoder port is set to "RECEIVE" encoder pulses from an external encoder.

If **ENC.OUT** is set to any valid value within the specified range, the ports will be set to "TRANSMIT" emulated encoder pulses at the specified pulses per motor rev (also known as the emulated line count).

ENC.IN must be set to equal zero for the controller to execute electronic gearing.

The maximum encoder frequency is 750 kHz.

NOTE

Example Program line
`ENC.OUT = 1024`

The controller transmits 4096 quadrature counts for each rotation of the motor.

See Also ENC.IN

ENCPOS

Variable (integer)

Syntax *x* = **ENCPOS**

Range ±2,147,483,647 encoder quadrature counts

Description **ENCPOS** (Encoder Position) indicates the position of the external encoder. For example, with a 1024 line encoder, each increment of **ENCPOS** is equal to 1/4096 of a revolution of the encoder shaft.

Install an incremental quadrature encoder with differential line driver-type outputs on the master axis.

The maximum encoder frequency is 750 kHz.

NOTE

Connect the encoder output from the master axis to the controller (J52) and verify that it is set to the correct **ENC.IN** line count.

ENCPOS can be set equal to zero.

Example Program line

```
'Turn the encoder ½ revolution.
ENC.IN = 1024
ENCPOS = 0
PRINT "ENCPOS = "ENCPOS
```

The output is ENCPOS = 2048

See Also ENC.IN, ENC.FREQ

END

Statement

Syntax **END** [{|**SUB** | **IF** | **INTERRUPT**}]

Description **END** marks the end of a program, subroutine, **IF…THEN…ELSE** block structure, or **INTERRUPT** service routine.

Once the **END** statement is encountered the program, subroutine, interrupt service routine, or block structure is terminated.

- **EXIT** – Use the **END** statement by itself on a single line.
- **Subroutine** – Mark the end of a subroutine with "**END SUB**"
- **IF…THEN…ELSE Block Structure** – Close the block with "**END IF**."
- **INTERRUPT Service Routine** – End the routine with "**END INTERRUPT**."

See Also SUB, IF…THEN…ELSE

ERR

Variable (integer) (read only)

Syntax *x* = **ERR**

Description **ERR** contains the error code for the last error that occurred. After and error, **ERR** contains the code for the error. Because **ERR** returns a meaningful value only after an error, it is usually used in error-handling routines to determine the error and the corrective action. **ERR** is a read-only variable that cannot be used on the left-hand side of an assignment statement.

NOTE

Please refer to Appendix G: Run Time Errors for additional information.

See Also ERRVAL, ERRVAR, ON ERRO GOTO

ERRVAL

Variable (float) (read only)

Syntax *x* = **ERRVAL**

Description **ERRVAL** contains the value that caused the last error to occur if the error was a variable-related error.

After an error, **ERRVAL** contains the value that caused the error. **ERRVAL** only returns a meaningful value after a variable-related error (**ERR** = 9, 10, 14 15), and is usually used in error-handling routines to determine the error and the necessary corrective action.

See Also ERR, ERRVAR, ON ERROR GOTO

ERRVAR

Variable (integer) (read only)

Syntax **ERRVAR** = *x*

Description **ERRVAR** contains the index number of the variable that caused the last error to occur if the error was a variable-related error.

After an error, **ERRVAR** contains the index number of the variable that caused the error. **ERRVAR** only returns a meaningful value after a variable-related error and is usually used in error-handling routines to determine the error and the necessary corrective action.

ERRVAR is a read-only variable that cannot be used on the left-hand side of an assignment statement.

NOTE

Please refer to Appendix G: Run Time Errors for additional information.

See Also ERRVAL, ERRVAL, ON ERRO GOTO

EXIT

Statement

Syntax **EXIT** [{|**SUB | FOR | INTERRUPT | WHILE** }]

Description Once the **EXIT** statement is encountered, program execution within a subroutine, or a block structure is aborted.

In the case of a block structure, program execution continues at the first statement following the structure. Issuing the **EXIT** statement within a subroutine returns program execution to the point from which the routine was initiated.

- **Subroutine** – Mark the end of a subroutine with "**EXIT SUB**"
- **FOR…NEXT Block Structure** – Exit the block with "**EXIT FOR**."
- **INTERRUPT Service Routine** – End the routine with "**EXIT INTERRUPT**."
- **WHILE…WEND Block Structure** – Exit the block with "**EXIT WHILE**."

See Also SUB, FOR…NEXT, INTERRUPT, WHILE…WEND

FAULTCODE

Variable (integer)

Syntax *x* = **FAULTCODE**

SC752/SC753

Status Display	FAULTCODE	Description
0	0	No fault, disabled
1	1	Software resolver over-speed
2	2	Motor over-temperature
3	3	Servocontroller over-temperature
4	4	Servocontroller IT
5	5	Motor line-neutral fault
6	6	Control under-voltage
7	7	Bus OV/OC (highest priority)
8	0	No fault, enabled
9	9	Estimated shunt regulator IT fault
B	11	Encoder +5V low
C	12	Terminal +5V low
E	14	Micro-processor fault
F1	241	Following error overflow
F2	242	Program memory fault
F3	243	Parameter memory fault
F4	244	Run time error
F5	245	PacLAN error
F6	246	Incompatible Motion Dialogue
UC		Unconfigured controller

SC754/SC755/SC756

Status Display	FAULTCODE	Description
0	0	No fault, disabled
1	1	Software resolver over-speed
2	2	Motor over-temperature
3	3	Servocontroller over-temperature
4	4	Servocontroller IT
5	5	Bus over-current
6	6	Control under-voltage
7	7	Output over-current
8	0	No fault, enabled
9	9	Measured shunt regulator IT fault
A	10	Bus over-voltage or Hot control under-voltage
b	11	Encoder +5V low
C	12	Terminal +5V low
d	13	Power stage control under-voltage
E	14	Micro-processor fault
F1	241	Following error overflow
F2	242	Program memory fault
F3	243	Parameter memory fault
F4	244	Run time error
F5	245	PacLAN error
F6	246	Incompatible Motion Dialogue
UC		Unconfigured controller

Status displays F1, F2, F3, F4, F5, F6 and UC alternately flash between the two values.

There is no faultcode 8 – No faults, enabled. For status display 8, the faultcode is 0. The variable ENABLED indicates whether the controller is enabled.

Description **FAULTCODE** indicates that status of the controller. When this code is not equal to zero, a fault has occurred.

Program a fault code in and expression to detect faults that occur during operation.

If fault occurs, the servocontroller must be reset by asserting the fault reset input signal or cycling controller AC power.

FIX

Function

Syntax **FIX**(*x*)

Description **FIX** returns the truncated integer part of *x.*

FIX does not round off numbers; it simply eliminates the decimal point and all characters to the right of the decimal point.

Example Program line

```
PRINT FIX(58.75)
This segment prints the value 58.

PRINT FIX(-58.75)
This segment prints the value –58.
```

See Also INT, ABS

FOR…NEXT

Statement (integer)

Syntax **FOR** loop_counter = start value **TO** end value [**STEP** increment]
.
.
.
NEXT x = [loop_counter]

Description **FOR…NEXT** allows a series of statements to be executed in a loop a given number of times.

For the desired variable, program the range of expressions and the step size to control the increments (or decrements).

Program the desired instructions for the loop.

End the loop with the **NEXT** instruction.

If a step (increment must be constant) is not specified, the increment is assumed to be 1.

Example Program line
For example, the **FOR…NEXT** lines:

```
FOR x = 2 TO 10
PRINT x
NEXT x
PRINT "LOOP DONE"
```

Execute 9 times. The fist loop sets x to 2, prints out the value 2, then loops back. The second pass through the loop sets x = 3, prints this value and loops back. The statement continues in this fashion until the final expression, x = 10 is printed out, then the program exits the **FOR…NEXT** loop and executes the PRINT "**LOOP DONE**" statement.
If a value is not programmed for STEP, the statement increments by 1 after starting with the initial expression. If a value is programmed for **STEP**, the statement increments by the **STEP** value after starting with the initial expression.

For example, if STEP =.5, for an expression:

```
FOR x = 2 TO 10 STEP 0,5
PRINT x
NEXT x
```

After the first loop executes and prints the value 2, the statement increments to 2.5, performs the loop and so on. If a negative value is programmed, the loops decrement. In this case, the initial expression must be greater than the final expression. For Example

```
FOR x = 10 TO STEP = -0.5
PRINT x
NEXT x
```

See Also EXIT

FVEL.ERR

Variable (float) (read only)

Syntax *x* = **FVEL.ERR**

Description **FVEL.ERR** indicates the velocity servo error signal (**VEL.CMD** = {**VEL.CMD** – **RVEL**}), in rpm, after the anti-resonant filter section has processed it.

Example Program line

```
'This program segment sends the variable FVEL.ERR
'to the analog output channel.
'Set up Output signal selection
DACMAP = 4
```

See Also ARF0, ARF1, DACMAP, VELOCITY, VEL.CMD, VEL.ERR

FWV

Variable (integer) (read only)

Syntax *x* = **FWV**

Description **FWV** indicates the controller firmware version number.

Example Program line

```
PRINT "This controller is a SC";MODEL;"With Firmware
Version –";FWV
```

GEARERROR

Variable (integer)

Syntax *x* = **GEARERROR**
or
GEARERROR = *x*

Description **GEARERROR** specifies the amount of position lag that accumulates when electronic gearing is turned on.

After turning on electronic gearing, slave axis acceleration will be limited to the specified **ACCEL.GEAR**. After velocity synchronization with the master axis the variable **GEARLOCK** is set equal to one. At this point, a correction move can be performed to eliminate the position lag due to acceleration. After starting the correction move, set **GEARERROR** to zero.

Example Program line

```
'Clear accumulated error
GEARERROR = 0
'Set up Accel Rate
ACCEL.GEAR = 1200
'Set up Decel Rate
DECAL.GEAR = 1200
'Turn on Gearing
WHEN INP1=0, CONTINUE
'When INP1 goes low
GEARING = 1
'Wait for Gearlock
WHILE GEARLOCK=0 : WEND
'Setup for Correction Move
INDEX.DIST = GEARERROR
'Perform Phase Correction
GO.INCR
'Set GEARERROR to zero
GEARERROR = 0
```

See Also GEARLOCK, GEARING, ACCEL.GEAR, DECEL.GEAR, RATIO, PULSES.IN, PULSES.OUT

GEARING

Variable (integer)

Syntax **GEARING** = *x*

Value of GEARING	Description
X= 0	Off, no gearing
X = 1	Permits bi-directional motion slaved to the master's encoder input signal
X = 2	Permits only clockwise controller response to follow the master's encoder input signal
X = 3	Permits only counter-clockwise controller response to follow the master's encoder input signal

Default 0

Description **GEARING** turns electronic gearing on or off and sets allowed direction of motion. Electronic gearing slaves the motion of the controller's motor to a master encoder signal.

Follow these guidelines to program the **GEARING** variable:

- Connect an encoder output from the master axis to the controller (J52). Specify the correct encoder line count **ENC.IN**
- Specify **RATIO** before programming **GEARING**.

MOVING does not recognize movement caused by GEARING.

If unidirectional gearing is set (x=2 or 3), gearing motion in the allowed direction occurs only when the master encoder returns to the point where it originally reversed direction.

Other motion commands could result in motion in the disabled gearing direction.

NOTE

Incremental moves may be executed while gearing is active. This results in a move referenced to the instantaneous gearing effect.

Example Program line

```
ENC.IN = 1024
RATION =1
RUN.SPEED = 100
INDEX.DIST = 4096
GEARING = 1
GO.INCR
```

When the **GO.INCR** is executed if the master encoder is not moving, the gearing effect is 0 and the motor performs a one-revolution move with a maximum speed of 100 PRM.

If the master encoder is moving at a rate of 1500 rpm, the motor is moving at 1500 rpm due to **GEARING**, and moves an additional revolution by increasing its speed to 1600 rpm for a pre-determined period of time.

See Also ENC.IN, RATION, ENCPOS, PULSES.IN, PULSES.OUT

GEARLOCK

Variable integer (read only)

Syntax *x* = **GEARLOCK**

Range 0 = no slave velocity synchronization
1 = that synchronization has been achieved.

Description **GEARLOCK** indicates slave axis velocity is synchronized with the master, when performing electronic gearing.

After turning on electronic gearing, slave axis acceleration is limited to the specified **ACCEL.GEAR**. After velocity synchronization with the master axis, **GEARLOCK** is set to one. At this point, a correction move can be performed to eliminate the position lag due to acceleration.

Example Program line

```
'Clear accumulated error
GEARERROR = 0
'Set Up Accel Rate
ACCEL.GEAR = 1200
'Set up Decel Rate
DECEL.GEAR = 1200
'Turn on Gearing
WHEN INP1=0, CONTINUE
'When INP1 goes low
GEARING = 1
'Wait for Gearlock
WHILE GEARLOCK= 0:WEND
'Setup for Correction Move
INDEX.DIST= GEARERROR
'Perform Phase Correction
GO.INCR
```

See Also GEARERROR, GEARING, ACCEL.GEAR, DECEL.GEAR, RATIO, PULSES.IN, PULSES.OUT

GO.ABS

Statement

Syntax **GO.ABS**

Description **GO.ABS** (Go Absolute) causes the motor to move to the position specified by **TARGET.POS**. This position is based on a zero position at electrical home.

The motor speed follows a velocity profile as specified by **ACCEL.TYPR, ACCEL.RATE**, and **DECEL.REAT**. Direction of travel depends on current position and target position only (DIR has no effect).

NOTE

The program does not wait for** GO.ABS **completion. After the program initiates this move, it immediately goes to the next instruction.

Variables may be changed during a move using **UPD.MOVE**.

- Set desired **ACCEL.TYPE, ACCEL.RATE, DECEL.RATE, RUN.SPEED** and **TARGET.POS**.
- Issue motion command **GO.ABS**.
- (Optional) Update motion parameters, then issue **UPD.MOVE**.
- If desired, abort commanded motion by executing an **ABORT.MOTION** statement.

Example Program line

```
‘This program segment establishes the current motor
‘position as 0 – electrical home, then performs
‘an absolute move to 10 revolutions clockwise
‘from electrical home using specified acceleration
‘and velocity parameters
‘Set up motion constraints
‘Set for constant acceleration velocity profile
ACCEL.TYPE = 0
‘Set acceleration rate equal to 12,000 rpm/sec
ACCEL.RATE = 12000
‘Set deceleration rate equal to 12,000 rpm/sec
DECEL.RATE = 12000
‘Set run speed equal to 120 rpm
RUN.SPEED = 120
‘Establish current position as electrical home
POS.COMMAND = 0
‘Move ten revolutions
TARGET.POS = 4096*10
GO.ABS
WHILE MOVING = 1 : WEND
PRINT “MOVE COMPLETE”
STOP
END
```

See Also ACCEL.TYPE, ACCEL.RATE, DECEL.RATE, POS.COMMAND, RUN.SPEED, ABORT.MOTION, TARGET.POS, UPD.MOVE

GO.HOME

Statement

Syntax **GO.HOME**

Description **GO.HOME** moves the motor shaft to the electrical home position (POSITION = 0).

The motor speed follows a velocity profile as specified by **ACCEL.RATE, RUN.SPEED**, and **DECEL.RATE**.

NOTE

The program does not wait for** GO.HOME **completion. After the program initiates this move it immediately goes to the next instruction.

GO.HOME performs the same action setting **TARGET.POS** to zero and executing a **GO.ABS** function.

Set desired **ACCEL.TYPE, ACCEL.RATE, DECEL.RATE, RUN.SPEED, DIR, TARGET.POS** and **INDEX.DIST**.

Example Program line

```
‘This program segment initializes the motion
‘constraints, then performs an absolute move
‘to the electrical home using specified
‘acceleration and velocity parameters
‘Set up motion constraints
‘Set for trapezoidal velocity profile
ACCEL.TYPE = 0
‘Set acceleration rate equal to 12,000 rpm/sec
ACCEL.RATE = 12000
‘Set deceleration rate equal to 12,000 rpm/sec
DECEL.RATE = 12000
‘Set run speed equal to 120 rpm
RUN.SPEED = 120

‘This statement is included for demonstration
‘purposes only – to establish current
‘position 4 revs from electrical home
POS.COMMAND = 4*4096
GO.HOME
WHEN IN.POSITION. WEND
WHILE MOVING = 1 : WEND
PRINT “MOVE COMPLETE”
STOP
END
```

See Also ACCEL.TYPE, ACCEL.RATE, DECEL.RATE, RUN.SPEED, ABORT.MOTION, UPD.MOVE

GO.INCR

Statement

Syntax **GO.INCR**

Description **GO.INCR** (Go Incremental) moves the motor shaft an incremental index from the current position.

Distance, as specified in **INDEX.DIST**, may be positive or negative. The motor speed follows a trapezoidal velocity profile as specified by **ACCEL.TYPE, ACCEL.RATE, RUN.SPEED** and **DECEL.RATE**

NOTE

The program does not wait for motion completion. After the program initiates this move it immediately goes to the next instruction.

Parameters may be changed during a move using **UPD.MOVE**.

- Set desired **ACCEL.TYPE, ACCEL.RATE, DECEL.RATE, RUN.SPEED** and **INDEX.DIST**
- Issue motion command **GO.INCR**.
- (Optional) Update motion parameters, then issue **UPD.MOVE**.
- If desired, abort commanded motion by executing an **ABORT.MOTION** statement.

Example Program line
#

See Also ACCEL.TYPE, ACCEL.RATE, DECEL.RATE, INDEX.DIST, RUN.SPEED, ABORT.MOTION, UPD.MOVE

GO.VEL

Statement

Syntax **GO.VEL**

Description **GO.VEL** (Go Velocity) moves the motor shaft at a constant speed.

The motor accelerates and reaches maximum speed as specified by **ACCEL.RATE** and **RUN.SPEED**, with direction determined by **DIR**. Stop motion by:

Programming **ABORT.MOTION** for maximum deceleration allowed by current limits.

Programming **RUN.SPEED** = 0 for deceleration at rate set by **DECEL.RATE**

After the program initiates a** GO.VEL, **it immediately goes to the next instruction.

NOTE

Variables may be changed during a move using **UPD.MOVE**.

Set desired **ACCEL.TYPE**, **ACCEL.RATE**, **DECEL.RATE**, **RUN.SPEED** and **DIR**.

Issue motion command **GO.VEL**.

(Optional) Update motion parameters, then issue **UPD.MOVE**.

If desired, abort commanded motion by executing an **ABORT.MOTION** statement.

GO.VEL always uses trapezoidal profiles, regardless of **ACCEL.TYPE**.

Example Program line

```
'This program segment drives the motor clockwise for
'1 second, stops for 1 second, drives
'counter-clockwise for 1 second, the repeats
'Set for constant acceleration velocity profile
ACCEL.TYPE = 0
'Set acceleration rate equal to 12,000 rpm/sec
ACCEL.RATE = 12000
'Set deceleration rate equal to 12,000 rpm/sec
DECEL.RATE = 12000
'Set run speed equal to 120 rpm
RUN.SPEED = 120
PAUSE.TIME = 1.0
WHILE 1 = 1
    DIR = 0
    'Move in the clockwise direction for one second
    RUN.SPEED = 120
    GO.VEL
    PAUSE
    'Stop motion
    RUN.SPEED = 0
    GO.VEL
    PAUSE
    'Move in the counter-clockwise direction for one
    second
    DIR = 1
    RUN.SPEED = 120
    GO.VEL
    PAUSE
    'Stop motion
    RUN.SPEED = 0
    GO.VEL
    'Wait for one second
    PAUSE
    'Repeat loop (indefinitely)
    WEND
END
```

See Also ACCEL.TYPE, ACCEL.RATE, DECEL.RATE, DIR, RUN.SPEED, ABORT.MOTION, UPD.MOVE

GOSUB…RETURN

Statement

Syntax **GOSUB** subroutine label

Description **GOSUB…RETURN** (Go to subroutine) branches program execution to a subroutine, executes it, and returns.

Subroutines branched to with a GOSUB…RETURN should be located at the end of the main program.

Example Program line

```
'Define motion constraints
RUN.SPEED = 1000
INDEX.DIST = 4096
GO.INCR
'Use sub to check for 'in position'
GOSUB MOVCHK
PRINT "Back from Sub"
END
SUB MOVCHK
WHILE IN.POSITION = 0 : WEND
    PRINT "MOVE COMPLETE"
    RETURN
END SUB
```

See Also CALL, SUB

GOTO

Statement

Syntax **GOTO** Label

Description **GOTO** causes software to jump to a specific label and continue executing.

NOTE

GOTO ***is not a recommended looping technique. Use of*** *GOTO* ***leads to disorganized and confusing programs. Looping techniques such as*** *FOR…NEXT, IF…THEN…ELSE* ***and*** *WHILE…WEND* ***are recommended for writing structured code.***

Example

Program line	Explanation
`X = 1`	`Execution leaves off here.`
`.`	
`.`	
`Label_1:`	`Execution continues here.`
`.`	
`.`	
`PRINT x`	

HEX$

String Function

Syntax **HEX$**(x)

Description **HEX$** (numeric-expression) converts an integer or floating-point variable to its equivalent hexadecimal ASCII string.

Hexadecimal numbers are numbers to the base 16 (rather than base 10).

x is rounded to an integer before **HEX$** is evaluated.

Example Program line

```
DIM A$ AS STRING
DIM X AS FLOAT
x = 32
A$ = HEX$(x)
PRINT x “Decimal is “A$” Hexadecimal”
```

The program prints: 32 decimal is 20 hexadecimal.

ICMD

Variable (float) (read only)

Syntax *x* = **ICMD**

Description **ICMD** indicates the commanded motor torque current in amps.

Example Program line

```
‘This program segment sends the variable ICMD to
‘the analog output channel
‘Set up Output signal selection
DACMAP = 8
```

See Also DACMAP, IFB, ILMT.MINUS, ILMT.PLUS, KTEFF

IFB

Variable (float) (read only)

Syntax *x* = **IFB**

Description **IFB** indicates the measured motor current amplitude in amps.

Example Program line

```
‘This program segment sends the variable IFB to
‘the analog output channel.
‘Set up Output signal selection
DACMAP = 9
```

See Also DACMAP, ICMD, ILMT.MINUS, ILMT.PLUS

IF…THEN…ELSE

Statement

Syntax

Single Line: **IF** *expression* **THEN** *statement* [**ELSE** *statement*]

Block:
IF *expression* **THEN**
[*statement block*]
[**ELSEIF** *expression* **THEN**
[*statement block*]]
[**ELSEIF** *expression* **THEN**
[*statement block*]]
.
.
[**ELSE**
[*statement block*]]
END IF

Description

IF…THEN…ELSE statements control program execution based on the evaluation of numeric expressions. The **IF…THEN…ELSE** decision structure permits the execution of program statements or allows branching to other parts of the program based on the evaluation of the expression.

There are two structures of **IF…THEN…ELSE** statements, single line and block formats.

Single Line: In this format, the expression following the IF clause is evaluated. If true, the statement following **THEN** is executed. If the expression is false, the program executes the statements following the [optional] **ELSE** keyword (if used). Otherwise, execution continues on the next program line.

Block: In this form, the first line of the structure contains only the **IF** keyword, the expression, and the **THEN** keyword.

If the expression is true, statements within the first statement block are executed. If the expression is false, [optional] **ELSEIF** clauses are first evaluated (top-to-bottom) and the first **ELSEIF** clause to be true results in the statement block following the **ELSEIF** to be executed. If neither the **IF** or [optional] **ELSEIF** are true, the statements following the [optional] **ELSE** clause are executed. The block structure **IF…THEN…ELSE** requires termination with and **END IF** statement.

Example

Program line

Single Line:

```
'SINGLE LINE IF…THEN…ELSE
DIM a A$ INTEGER
LOOP2:
IF a > 0 THEN PRINT "a > 0" ELSE PRINT "a <= 0"
```

Block Type

```
'BLOCK IF…THEN…ELSE
IF POSITION > 0 THEN
    IF POSITION <= 0 THEN
        PRINT "0<POSITION<=4096"
    ELSE
        PRINT "POSITION>4096"
    END IF
```

Block Type

```
'BLOCK IF…THEN…ELSE
IF POSITION > 0 THEN
    IF POSITION <= 0 THEN
        PRINT "0<POSITION<=4096"
    ELSE
        PRINT "POSITION>4096"
    END IF
ELSE
    IF POSITION => -4096 THEN
        PRINT "-4096 <= POSITION"
    ELSE
        PRINT "POSITION < -4096"
    END IF
END IF
```

Block Type

```
'BLOCK IF…THEN…ELSE WITH THE USE OF THE
'ELSE IF CLAUSE
IF POSITION > 8148 THEN
    PRINT "POSITION > 8148"
ELSEIF POSITION > 4096"
    PRINT "POSITION > 4096"
ELSEIF POSITION > 2048 THEN
    PRINT "POSITION > 2048"
ELSEIF POSITION > 0 THEN
    PRINT "POSITION > 0"
ELSE
    PRINT "POSITION <= 0"
END IF
```

ILC

Parameter (integer)

Syntax **ILC** = *x*

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Description **ILC** sets the current loop proportional gain. **ILC** is an integer between 1 and 255, which defines the current loop's proportional gain (the actual gain varies inversely with **ILC). ILC** is determined as follows: ILC = (10.06 / (Ipeak * L1=1(henries)))

Where:

I_{peak} is the drive's peak current rating:

Model	Peak Current Rating
SC752	7.5
SC753	15
SC754	30
SC755	60
SC756	120

L_{1-1} is the motor's line-to-line inductance in henries.

Example For an SC752 (I_{peak} = 7.5) and R32G motor (L_{1-1} = 23mh),

ILC = Closest integer to 10.06/(7.5 * 23x10-3) = 58

ILC should be set to the integer closest to the value determined by this equation.

CAUTION

IMPORTANT NOTE: A value assigned to ILC by the controller Set Up feature (one of the menu options in Motion Dialogue). This value is appropriate for most applications and the user normally should not be concerned with setting it in the application program.

Example Program line

```
'This program segment prints the value of ILC
PRINT "The value of ILC is "; ILC
```

See Also ARF0, ARF1, KVI, KVP, KPP, KVFF

ILMT.MINUS

Parameter (integer)

Syntax **ILMT.MINUS** = *x* where *x* is the desired percent

Range 0 to 100 percent

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Description **ILMT.MINUS** (Counter-clockwise Current Limit) sets the maximum allowable current in the counter-clockwise direction. This is a percentage of the controller's peak current rating. (I_{peak}).

Only integer values may be entered (i.e. no fractional numbers).

See Also ILMT.PLUS, ABORT.MOTION

ILMT.PLUS

Parameter (integer)

Syntax **ILMT.PLUS** = *x* where *x* is the desired percent

Range 0 to 100 percent

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Description **ILMT.PLUS** sets the maximum allowable current in the clockwise direction as a percentage of the controller's peak current rating (I_{peak}).

Only integer values may be entered (i.e. no fractional numbers).

See Also ILMT.MINUS, ABORT.MOTION

INDEX.DIST

Variable (integer)

Syntax **INDEX.DIST** = +*x*

Where positive values move clockwise and negative values move counter-clockwise.

Range -134,217,728 to 134,217,727 resolver steps

NOTE

1 step = 1/4096 revolution

Default 4096

Description **INDEX.DIST** sets the distance the motor rotates during each incremental index. Specify **INDEX.DIST** prior to issuing a **GO.INCR** command.

See Also GO.INCR

INKEY$

String Variable (read only)

Syntax x = **INKEY$**

Description **INKEY$** returns one string character read from the serial input port's buffer. If no character is pending in the serial buffer, a null string (length zero) is returned. If several characters are pending, only the first is returned. Once a character is read from the buffer, it is removed from the buffer.

NOTE

ASCII Codes are listed in the table located in Appendix B: ASCII Codes.

NOTE

If you are using multidrop serial communication, and the SC750 controller is not "logged on" then no characters are available from the serial port and the INKEY$ variable will return a null string. Refer to LOGGEDON and Appendix F "Multidrop Serial Communication."

Example Program line

```
DIM A$ AS STRING
A$ = "" 'Preset A$ to null string
WHILE A$ = "" 'Check for new character
WEND
PRINT A$
```

See Also CHR$

IN.POS.LIMIT

Variable (integer)

Syntax **IN.POS.LIMIT** = *x*

Resolution 1/4096 of a motor revolution

Default 5

Description The **IN.POS.LIMIT** specifies the tolerance of commanded position minus actual position (POS.ERROR) within which region the position limit flag is set.

See Also IN.POSITION

IN.POSITION

Variable (integer) (read only)

Syntax x = **IN.POSITION**

Range: 0 or 1

Description **IN.POSITION** indicates whether or not the motor has achieved commanded position. IN.POSITION is useful to monitor position commands to ensure that desired motion has been completed. IN.POSITION is always either 1 (true) or 0 (false).

IN.POS.LIMIT – specifies position tolerance of the commanded position minus the actual position.

Example Program line

```
'Set electrical home to zero
POS.COMMAND = 0
'Set to move 10 revolutions
TARGET.POS = 4096 X 10
'Set velocity equal to 1 rev/sec
RUN.SPEED = 60
'Set run speed equal to 60 rpm
IN.POS.LIMIT = 10
GO.ABS
TIME = 0
'Move
WHILE IN.POSITION = 0
PRINT TIME, POSITION
WEND
```

The move should take 10 seconds.

See Also IN.POSITION

INPN

Variable (integer) (read only)

Syntax *x* = **INP*n*** where *n* = 1 to 16

Range 0 – Input low (On)
1 – Input high (Off)

Description **INP*n*** reads the state of an individual discrete input. This is a read only variable and cannot be set by the software.

- 0 indicates logic low input
- 1 indicates logic high input.

Example Program line

```
'Check both inputs before beginning operation
WHILE (INP1 = 0) OR (INPUT2 = 0) : WEND
PRINT "Machine set up"
END
```

See Also INPUTS, OUTPUTS, OUTn

INPUT

Statement

Syntax **INPUT** [“ *prompt* “] [,|;] *variable*

Range As an option, the “prompt” message is transmitted when the INPUT statement is encountered. If a semicolon follows the prompt string, a question mark is printed at the end of the prompt string. If a comma follows the prompt string, no question mark is printed.

Description **INPUT** reads a character string received by the serial communications port, terminated by a carriage return.

The input variables can be integer, float, or string variable types.

If invalid data is entered for a variable, the message “Redo from start” is transmitted and the data must be re-entered.

Example Program line

```
‘Declare variables as floating point
DIM PART_LEN, CF1, FEED_RATE, CF2 AS FLOAT
‘Convert to resolver counts
CF1 = 4096/0.5
‘Convert in/sec to rpm
CF2 = 60/0.5
‘Enter length
INPUT “Enter part length (in)”, PART_LEN
‘Enter feed rate
INPUT “Enter feed rate (in/sec)”, FEED_RATE
‘Define move distance
INDEX.DIST = CF1 * PART_LEN
‘Define speed
RUN.SPEED = CF2 * FEED_RATE
‘Execute move
GO.INCR
END
```

See Also INKEY$

INPUTS

Variable (integer) (read only)

Syntax *x* = **INPUTS**

Range A decimal value corresponding to the sum of the binary number of the inputs.

Input	Decimal value indicating Off	Input	Decimal value indicating Off
1	1	7	64
2	2	8	128
3	4	9	256
4	8	10	512
5	16	11	1024
6	32	12	2048

Range 0 to 65535

Default 65535 (inputs disconnected or all inputs Off)

Description **INPUTS** reads the state of discrete inputs. This is a read-only variable determined by the voltage levels applied to the discrete input pins.

Set the variable equal to the sum of the x values for Off (high) inputs. For example:

- Inputs 1 to 16 Off (high): **INPUTS** = 65535
- All inputs On (low): **INPUTS** = 0
- Input 5 Off (all others On): **INPUTS** = 16

INSTR

String Function

Syntax **INSTR** ([*n*],*x*$, *y*$) where *x*$ is the string and *y*$ is the substring. *n* optionally sets the start of the search

Range 1 to 255

Description **INSTR** provides the location of a substring within a string. *n* must be within the range of 1 to 255. INSTR returns 0 if:

- *n* > **LEN** (*x*$)
- *y*$ cannot be found

If *y*$ is null, **INSTR** returns *n*.

Example Program line

```
DIM X$, Y$, A$ AS STRING
X$ = "ABCDEBXYZ"
Y$ = "B"
PRINT INSTR (x$, y$) ; INSTR (4, x$, y$)
```

This program prints: 2 6

The interpreter searches the string and finds the first occurrence of "B" at position 2. It then starts another search at Position 4 (D) and finds another match at Position 6 (B).

INT

Function

Syntax **INT** (x)

Description **INT** (Convert to Largest Integer) truncates an expression to a whole number.

Example Program line
`PRINT INT (99.89)`
Prints the value 99.

`PRINT INT (-12.11)`
Prints the value –13.

See Also CINT, FIX

INTERRUPT

Statement

Syntax **INTERRUPT** (Source Label)

User Program Statements

[**INTR**.{*Source Label*} = 1]
END INTERRUPT

Description **INTERRUPT** marks the beginning and the end of an interrupt service routine. **INTERRUPT** permits user-defined program execution upon receipt of a hardware interrupt signal or predefined interrupt event. The interrupt service routine is defined by a program structure resembling a subroutine.

Interrupts are triggered by predefined events or external hardware sources as indicated in the table. The interrupt source label and enable flag are unique for each interrupt type.

Receiving an interrupt suspends execution of the main program (after completing the execution of the instruction being processed when the interrupt was triggered). The interrupt service routine is executed and the program continues executing at the point from which it was interrupted.

Interrupt service routines must be defined after the END statement in the main program. The body of the interrupt service routine contains the ServoBASIC Plus statements implementing a user-defined service algorithm.

Interrupts are enabled or disabled by setting or resetting the appropriate enable flag. Interrupts are disabled until explicitly turned on. After an interrupt is triggered, it is disabled. Many situations may find it desirable to enable the flag prior to exiting the service routine.

NOTE

General-purpose input sources are edge triggered. For example, I1HI triggers as input 1 transitions from low to high. I1LO triggers as input 1 transitions from high to low.

Example Program line

```
'Enable the input 1, high level interrupt.
INTR.I1HI = 1
'Endless loop
loop:
GOTO loop
END
INTERRUPT I1HI
PRINT "input 1 is high – Interrupt Occurred"
'Re-enable the interrupt
INTR.I1HI = 1
END INTERRUPT
```

See Also RESTART

Source Code Table	
Interrupt Source	**Source Label**
Counter-Clockwise Inhibit Active	CCWINH
Counter-Clockwise Over-Travel	CCWOT
Serial Communications Port	CHAR
Clockwise Inhibit Active	CWINH
Clockwise Over-Travel	CWOT
Disabled	DISABLE
Controller Fault	FAULT
General Purpose Input 1 – High State	I1HI
General Purpose Input 1 – Low State	I1LO
General Purpose Input 2 – High State	I2HI
General Purpose Input 2 – Low State	I2LO
General Purpose Input 3 – High State	I3HI
General Purpose Input 3 – Low State	I3LO
General Purpose Input 4 – High State	I4HI
General Purpose Input 4 – Low State	I4LO
General Purpose Input 5 – High State	I5HI
General Purpose Input 5 – Low State	I5LO
General Purpose Input 6 – High State	I6HI
General Purpose Input 6 – Low State	I6LO
General Purpose Input 7 – High State	I7HI
General Purpose Input 7 – Low State	I7LO
General Purpose Input 8 – High State	I8HI
General Purpose Input 8 – Low State	I8LO
General Purpose Input 9 – High State	I9HI
General Purpose Input 9 – Low State	I9LO
General Purpose Input 10 – High State	I10HI
General Purpose Input 10 – Low State	I10LO
General Purpose Input 11 – High State	I11HI
General Purpose Input 11 – Low State	I11LO
General Purpose Input 12 – High State	I12HI
General Purpose Input 12 – Low State	I12LO
General Purpose Input 13 – High State	113HI
General Purpose Input 13 – Low State	I13LO
General Purpose Input 14 – High State	I14HI
General Purpose Input 14 – Low State	I14LO
General Purpose Input 15 – High State	I15HI
General Purpose Input 15 – Low State	I15LO
General Purpose Input 16 – High State	I16HI
General Purpose Input 16 – Low State	I16LO
PacLAN	PacLAN
Position Error	POS.ERROR

INTR.{SOURCE LABEL}

Variable (integer)

Syntax **INTR.{*Source Label*}** = *x*

Range 0 disables interrupts
1 enables interrupts

Default 0

Description **INTR.{*Source Label*}** parameters enable and disable the interrupts. The interrupt feature permits user-defined program execution upon receipt of a hardware interrupt signal or predefined interrupt event.

Interrupts must be individually enabled (or disabled) to permit (or lock-out) execution of the corresponding interrupt subroutine by setting the interrupt enable flag corresponding to the interrupt source.

The default state for interrupts is disabled. Interrupts remain disabled until explicitly turned on. The interrupt enable settings can be polled during program execution to determine which interrupts are enabled. Disabling (currently enabled) interrupts requires clearing the appropriate interrupt enable flag

NOTE

When an interrupt is given, and the interrupt service routing is executed, the interrupt is disabled. The interrupt service routine must re-issue the interrupt enable instruction if it is desired to leave it enabled.

Example Program line
`INTR.I3HI = 1`
Enables an interrupt for input 3, when it transitions to the logical high state.
`INTR.CHAR = 1`
Enables an interrupt for an input character via the serial input channel.
`INTR.I1LO = 0`
Disables the interrupt when discrete Input 1 goes to a low logic level.

See Also INTERRUPT, RESTART

IPEAK

Variable (float) (read only)

Syntax *x* = **IPEAK**

Description **IPEAK** contains the peak current rating of the controller in amps. I_{peak} is the drive's peak current rating:

Model	Peak Current Rating
SC752	7.5
SC753	15
SC754	30
SC755	60
SC756	120

ITF0

Parameter (float)

Syntax **ITF0** = *x* where *x* is the filter's corner frequency (Hz)

Range 0.017 to 373 (Hz)

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Description **ITF0** with **IT.THRESH** specifies the thermal protection circuit for the servocontroller**. ITF0** represents the corner frequency of a low pass filter, which processes, the absolute value of the measured motor current. The filter's output is **IT.FILT**. Increasing **ITF0** adds protection to the controller by permitting faster response to over-current conditions.

NOTE

The minimum frequency for ITF0 (slowest response) is limited to protect the servocontroller's power electronics.

See Also IT.THRESH, IT.FILT

IT.FILT

Variable (float) (read only)

Syntax *x* = **IT.FILT** where *x* contains amps of motor current

Description **IT.FILT** contains the output of the I*T controller thermal protection filter circuit.

IT.FILT provides a means of evaluating the I*T protection circuit. When **IT.FILT** exceeds the threshold specified by **IT.THRESH**, the controller indicates an IT fault and disables itself.

This variable can be output to the Analog Output Circuit (by setting **DACMAP** = 13) and compared to the actual motor current for evaluation of thermal characteristics.

Example Program line

```
'This program segment sends the variable IT.FILT to
'the analog output channel.
'Set up Output signal selection
DACMAP = 13
```

See Also ITF0, IT.THRESH, DACMAP

IT.THRESH

Parameter (float)

Syntax **IT.THRESH** = *x*
where *x* is a percentage of the controller's peak current capability

Model	Maximum (%)
SC752	75%
SC753	60%
SC754	75%
SC755	66%
SC756	60%

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Description **IT.THRESH** designates the threshold (trip level) at which the IT thermal protection circuit triggers an over-current fault.

IT.THRESH in conjunction with the peak current for a servocontroller (IPEAK) specifies the level of filtered motor current (**IT.FILT**), which activates a thermal fault. This threshold specifies the maximum continuous (steady-state) motor current permitted. Reduce this parameter to set a lower threshold of motor over-current protection.

NOTE

The maximum value for IT.THRESH is limited to protect the servocontroller power electronics.

See Also ITF0, IT.FILT, IPEAK

KPP

Parameter (float)

Syntax **KPP** = *x*

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Description **KPP** sets the proportional gain of the position loop. **KPP** is the position loop's proportional gain. It has units of Hz and is defined as:

KPP (Hz) = [Commanded velocity (rad/sec) / Position Error (rad)] / (2 * pi)

CAUTION

A value is assigned to** KPP **by the controller Set Up feature (one of the menu options in Motion Dialogue). This value is appropriate for most applications and you normally should not set it in the application program.

Example Program line

```
'This program segment computes the expected
'following error in resolver counts for a
'given value of RUN.SPEED and KPP
DIM PI, FOLLOWERR AS FLOAT
PI = 3.1416
FOLLOWERR = 4096 * {(RUN.SPEED/60)/KPP}/ (2*pi)
```

See Also ARF0, ARF1, ILC, KVI, KVP, KVFF

KTEFF

Variable (float) (read only)

Syntax *x* = **KTEFF**

Where *x* is the torque constant in lb-in/amp

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Description **KTEFF** is the torque constant of the system in lb-in/amp. Instantaneous motor shaft torque in lb-in is **ICMD***__KTEFF__.

See Also ICMD

KVFF

Parameter (float)

Syntax **KVFF** = *xx*

Range 0 to 375 (*xx* is the percentage of velocity feedforward)

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Description **KVFF** sets the proportion of velocity feedforward signal added to the velocity command from differentiated position command.

With no velocity feedforward **KVFF** = 0), the commanded velocity results entirely from position error for position loops. Velocity feedforward adds a term to the commanded velocity equal to the expected velocity times (**KVFF**/100). Increasing **KVFF** reduces following error and gives faster response time but, if too large, it can produce overshoot. Typically, **KVFF** is not set larger than 80 for good dynamics and acceptable overshoot, but is set to 100 for minimum following error.

CAUTION

A value is assigned to** KVFF **by the controller Set Up feature (one of the menu options in Motion Dialogue). This value is appropriate for most applications and you normally should not set it in the application program.

Example Program line

```
DIM FEEDFORWARD AS FLOAT
LOOP:
INPUT “Set velocity feedforward gain (0 to 100)”;
FEEDFORWARD

IF FEEDFORWARD < 0 THEN GOTO LOOP
ELSE IF FEEDFORWARD > 100 THEN GOTO LOOP
ELSE KVFF = FEEDFORWARD
END IF
```

See Also ARF0, ARF1, ILC, KVI, KVP, KPP

KVI

Parameter (float)

Syntax **KVI** = *x*

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Description **KVI** sets the integral gain of the velocity servo loop.It has units of Hz and defines the frequency where the velocity loop compensation transitions from integral characteristics (gain decreasing inversely with frequency) to proportional characteristics (constant gain).

CAUTION

A value is assigned to KVI by the controller Set Up feature (one of the menu options in Motion Dialogue). This value is appropriate for most applications and you normally should not set it in the application program.

Example Program line

```
'This program segment sets KVI based on user input
DIM LAGBREAK AS FLOAT
INPUT "Desired velocity loop lag-break frequency (Hz)" LAGBREAK
KVI = LAGBREAK
```

See Also ARF0, ARF1, ILC, KVP, KPP, KVFF

KVP

Parameter (float)

Syntax **KVP** = *x*

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Description **KVP** sets the proportional gain of velocity servo loop. It has units of amps/(rad/sec) and is defined by the following relationship:

KVP {amps/(rad/sec)} = Commanded motor current (amps)/Velocity error (rad/sec)

The example program shows how **KVP** can be computed to give a desired velocity loop bandwidth (unity-gain crossover frequency) in Hz given the total load inertia and motor torque constant (K_T).

CAUTION

A value is assigned to KVP by the controller Set Up feature (menu option in Motion Dialogue). This value is appropriate for most applications and you normally should not set it in the application program.

Example Program line

```
'This program segment calculates KVP based
'on desired velocity-loop bandwidth, total inertia,
'and motor torque constant
DIM PI AS FLOAT
INPUT "Motor plus Load Inertia (lb-in-sec^2) "; J_tot
INPUT "Desired Velocity Loop Bandwidth (Hz)"; BW
KVP = 2 * pi * BW * J_tot / KTEFF
```

See Also ARF0, ARF1, ILC, KVI, KPP, KTEFF, KVFF

LANFLT

Variable (float)

Syntax The floating-point variable array syntax is specified as:

x = **LANFLT**(*n*) [*axis#*]

or

LANFLT(*n*) [*axis#*] = *x*

where *n* is the array subscript, ranging from 1 to 32, and *x* is a floating-point value. [*axis#*] specifies the axis location of the LAN floating-point variable array.

Description **LANFLT(*n*)** is an array of 32 floating point variables globally accessible over PacLAN.

To read a **LANFLT(*n*)** variable from another SC750 connected with PacLAN specify:

x = **LANFLT**(*n*) [*axis#*]

where [*axis#*] is the axis address of the controller being read from.

To write a **LANFLT(n)** variable into another SC750 connected with PacLAN use the statement:

LANFLT(*n*) [*axis#*] = *x*

where [*axis#*] is the axis address of the controller being written to.

NOTE

The [axis#] designation can be omitted when accessing (reading or writing) and axis' own LANFLT(n) variables.

See Also LANINT

LANINT

Variable (integer)

Syntax The integer variable array syntax is specified as

x = **LANINT**(*n*) [*axis#*]

or

LANINT(*n*) [*axis#*] = *x*

where *n* is the array subscript, ranging from 1 to 32, and *x* is an integer. [*axis#*] specifies the axis location of the LAN integer variable array.

Description **LANINT** is an array of 32 integer variables globally accessible over PacLAN.

To read a **LANINT(*n*)** variable from another SC750 connected with PacLAN into the variable x specify:

x = **LANINT**(*n*) [*axis#*]

where [*axis#*] is the axis address of the controller being read from.

To write a **LANINT(n)** variable into another SC750 connected with PacLAN into the variable x use the statement:

LANINT(*n*) [*axis#*] = *x*

where [*axis#*] is the axis address of the controller being written to.

NOTE

The [*axis#*] designation can be omitted when accessing (reading or writing) an axis' own LANINT variables.

See Also LANFLT

LANINTERRUPT

Statement

Syntax **LANINTERRUPT [*axis#*]**

where [*axis#*] is the address of destination of the interrupt

Description **LANINTERRUPT** invokes an interrupt to the PacLAN controller specified by [*axis#*].

Before issuing this command, insure that the destination axis is connected to PacLAN, or a run time error is generated on the source axis.

See Also INTR.PacLan, INTERRUPT, AXIS.INTR, STATUS

LCASES$

String Function

Syntax **LCASE$**(x)

Description **LCASE$** converts expression to lower case characters.

Example Program line

```
DIM A$ AS STRING
A$ = "U.S.A."
PRINT LCASE$(AS)
```

This program segment prints: u.s.a

See Also UCASE$

LEFT$

String Function

Syntax **LEFT$** = (*x*$, *n*)

Range 0 to 255

Description **LEFT$** returns a string of the n leftmost characters of the string expression.

If *n* is greater than **LEN**(*x$*)**,** the entire string (*x$*) is returned.

Example Program line

```
A$ = "Mississippi"
PRINT LEFT (A$, 5)
```

This program segment prints: Missi

See Also MID$, RIGHT$

LEN$

String Function

Syntax **LEN** = *x$*

Description **LEFT$** returns a string of characters in the string (*x$*). Non-printing characters and blanks are included.

Example Program line

```
DIM A$ AS STRING
x$ = "New York, New York"
PRINT LEN(x$)
```

This program segment prints the decimal value 18.

NOTE

The comma and spaces are included in the length of the string.

LOG

Function

Syntax **LOG**(*x*)

Description **LOG** returns natural logarithm value (*x*). *x* must be greater than zero.

Example Program line

```
PRINT LOG (45/7)
Prints the decimal value 1.860752.
PRINT LOG (1)
Prints the decimal value 0.
PRINT LOG (2)
Prints the decimal value 0.6931471
```

See Also LOG10

LOG10

Function

Syntax **LOG10** (*x*)

Description **LOG10** returns base 10 logarithm of expression. *x* must be greater than zero.

See Also LOG

LOGGEDON

Variable Integer

Syntax *x* = **LOGGEDON**

Range 0= logged off
1=logged on

Description **LOGGEDON** indicates that the multidrop master has selected the controller as the enabled multidrop subsystem.

When the multidrop master transmits a subsystem selection message that corresponds to the multidrop subsystem address **(AXIS.ADDR),** the subsystem is permitted to transmit messages to the multidrop master.

If the subsystem is "Logged On" (**LOGGEDON**=1), data transmitted from the multidrop master are read using either **INKEY$** or **INPUT**. If the subsystem is not **LOGGEDON**, data transmitted by the multidrop master are not available to the subsystem.

See Also AXIS.ADDR

LTRIM$

String Function

Syntax **LTRIM$ (string_expression)**

Description **LTRIM$** removes the leading blank characters from a string expression. String_expression can be any string expression.

Example Program line

```
PRINT ltrim$ (“       Trim test”)
```

The program line above produces the following output:

Trim test

MID$

String Function

Syntax **MID$** *(string_expression, start_offset [, length])*

Description **MID$** returns a substring of a string expression that begins at the specified offset location. *string_expression* is any string expression. *start_offset* is the position of the first character of the substring. *length* is the number of characters in the substring.

NOTE

If the length is omitted, MID$ returns all characters from start_offset through the end of the string.

Example Program line

```
DIM A$ AS STIRNG
A$ = “ABCDEFGHI”
PRINT MID$ (a$, 1, 5)
PRINT MID$ (a$, 6)
```

This program prints the following:

ABDCE
FGHI

See Also INSTR, LEFT$, LEN, RIGHT$

MOD

Arithmetic Operator

Syntax *x = numeric expression MOD numeric expression*

Description **MOD** is the modulus or remainder. It divides one number by another and returns the remainder.

Example Program line

```
PRINT 19 MOD 5
```

This program prints: 4

MODEL

Variable (integer) (read only)

Syntax *x* = **MODEL**

Range Valid values are only: 752, 753, 754, 755 or 756

Description **MODEL** indicates the servocontroller model number (power level). This is a read only parameter.

Example Program line
```
PRINT MODEL
```
Prints the model number of the servocontroller being used.

MOVING

Variable (integer) (read only)

Syntax *x* = **MOVING**

Range 0 = Commanded motion profile complete
1 = Commanded motion profile in progress

Description **MOVING** provides an indication that a commanded motion profile is complete. The motion profile is based on the mode of commanded motion and the associated constraints.

NOTE

Actual motion is affected not only by the commanded motion, but also by motor capabilities, servo tuning and process dynamics. *DO NOT* ***use the*** *MOVING* ***variable to verify that no motion is in process.***

Example Program line
```
RUN.SPEED = 1000
ACCEL.RATE = 10000
DECEL.RATE = 10000
INDEX.DIST = 20 * 4096
GO.INCR
TIME = 0
    WHILE TIME < 4
PRINT VELOCITY, MOVING
WENT
```

See Also IN.POSITION

ON ERROR GOTO {LABEL}

Statement

Syntax

```
ON ERROR GOTO {Label)
goto DoneProgram          'prevents the main program from exiting to
                          the Error Handler
{Label}:                  'Code to handle the Error
RESTART                   'start again from the beginning of the program
    OR
ON ERROR GOTO 0           'call the default error handler (halt program,
                          abort motion, generate "F4" fault)
Done Program:
END
```

NOTE

Both** RESTART **and** ON ERROR GOTO 0 **work in this routine.

Description **ON ERROR GOTO** allows you to define a ServoBASIC Plus Run-Time Error Handler to prevent Run-Time Errors from halting program execution.

NOTE

Different Error Handlers can be defined for different sections of the program; an Error Handler is valid from when the** ON ERROR GOTO **statement is executed until another one is encountered.

The label argument is the label of the first line in the error-handling routine.

If no active error handler is found, an error message is printed and program execution halts. The specific error message depends on the type of error.

A label of 0 disables error handling. It does not specify line 0 as the start of the error-handling code, even if the program contains a line numbered 0. Subsequent errors print and error message and halt the program. Once error handling is enabled, any error that can be trapped causes a jump to the specified error-handling routine.

Inside an error handler, executing an **ON ERROR** with a label of 0 halts program execution and prints the error message for the error that caused the trap. This is a convenient way to halt a program in response to errors that cannot be processed by the error-handling routine.

NOTE

An error-handling routine is** NOT **a** SUBROUTINE**. An error-handling routine is a block of code marked by a label.

NOTE

Errors occurring within an error-handling routine are not trapped. These errors halt program execution after printing an error message.

See Also ERR, ERRVAL, ERRVAR

*OUT*N

Variable (integer)

Syntax **OUT*n*** = *x* with *n* = 1 to 12

Range 0 = specific outputs (1 to 12) to be On (pulled low)
1 = specific outputs (1 to 12) to beOff (open circuit)

Default 1

Description **OUT*n*** (Outputs 1 to 12) sets the state of the individual discrete output.

Set the individual variable equal to 0 to turn an output On or to 1 to turn and output Off.

To set all outputs equal to 1, use outputs = 8191

NOTE

Outputs 1 to 3 can also be controlled by *POS.CHKn.OUT.*

See Also OUTPUTS, POS.CHK*n*.OUT

OUTPUTS

Variable (integer)

Syntax **OUTPUTS** = *x*

Range 0 to 4095

Default 4095

Description **OUTPUTS** specifies the state of the discrete outputs. For example, set the variable equal to the sum of the *x* values for Off (high) inputs.

- Outputs 1 to 12 Off (high): **OUTPUTS** = 4095
- All outputs On (low): **OUTPUTS** = 0
- Output 5 Off (all others On)" **OUTPUTS** = 16

See Also OUT*n*

PAUSE

Statement

Syntax **PAUSE**

Description **PAUSE** causes the program to pause the amount of time specified by the **PAUSE.TIME** variable. The motion of the motor is not affected. Issue **PAUSE** fin place of software loops (e.g. **FOR…NEXT**) for precise control of timing.

Example Program line

```
‘This program waits 1 second
‘between every print statement
PAUSE.TIME = 1
WHILE 1 = 1
    PRINT TIME
    PAUSE
WEND
```

See Also PAUSE.TIME

PAUSE.TIME

Variable (float)

Syntax **PAUSE.TIME** = *x*

Range 0.001 to 999,999

Default 1.00

Description Designates the amount of time to pause.

Example Program line

```
‘This program segment waits 1 second
‘between every print statement
PAUSE.TIME = 1
WHILE 1 = 1
    PRINT TIME
    PAUSE
WEND
```

See Also PAUSE

POLECOUNT

Parameter (integer)

Syntax **POLECOUNT** = *x*

NOTE

The default value of this parameter is set during the configuration set up in Motion Dialogue.

Range Valid values are only 4, 6, 8, or 12

Description **POLECOUNT** matches the controller for the appropriate motor pole count.

CAUTION

A value is assigned to** POLECOUNT **by the controller Set Up feature (one of the menu options in Motion Dialogue). This value is appropriate for the selected motor.** POLECOUNT **need only be set by the user if a brushless motor not supported by the Set Up menu is used.

Example Program line

```
'This program segment prints the value of POLECOUNT
PRINT "The value of POLECOUNT is "; POLECOUNT
```

POS.CHKN

Variable (integer)

Syntax **POS.CHK1** = *x*
POS.CHK2 = *x*
POS.CHK3 = *x*

Range Any valid arithmetic expression (-134,217,728 to 134,217,728 resolver steps).

NOTE

1 revolution = 4096

Default 0

Description **POS.CHK*n*** (Position Check trigger 1, 2 or 3) is the position at which outputs 1, 2 and 3 are switched to the polarity designated by **POS.CHK*n*.OUT**. Position check acts as a programmable limit switch output.

Program **POS.CHK*n*.OUT** to enable the **POS.CHK*n*.**

Refer to **POS.CHK*n*.OUT** for more information.

NOTE

Make sure to program** POS.CHKn **after establishing electrical home with** POS.COMMAND. POS.CHKn **is an absolute position variable that is changed when electronic home is changed.

Example Program line

```
'Reset electrical zero
POS.COMMAND = 0
POS.CHK1 = 10,000
POS.CHK2 = 20,000
POS.CHK3 = 30,000
    'Set outputs to logic 1
POS.CHK1.OUT = 11
POS.CHK2.OUT = 11
POS.CHK3.OUT = 11
TARGET.POS = 31,000
GO.ABS
'Wait for outputs to turn on
WHILE MOVING = 1 : WEND
```

See Also POS.CHK*n*.OUT

POS.CHKN.OUT

Type

Syntax **POS.CHK*n*.OUT** = 0

Range Valid values are only 0, 10, 11

Default 0

Description **POS.CHK*n*.OUT** (position check output specifier)is used in conjunction with **POS.CHK*n*** to implement Position Check *n*. Position check functions as a programmable limit switch output.

POS.CHK*n*.OUT can be set to one of three values:

Value	Description
0	Position check *n* disabled
10	Position check *n* enabled
	If (POSITION ≥ POS.CHK*n*) then OUT*n* = 0
	If (POSITION < POS.CHK*n*) then OUT*n* = 1
11	Position check n enabled
	If (POSITION ≥ POS.CHK*n*) then OUT*n* = 1
	If (POSITION < POS.CHK*n*) then OUT*n* = 0

- OUT1 to OUT3 (Outputs 1 to 3) cannot be programmed if the outputs are enabled using **POS.CHK1.OUT** to **POS.CHK3.OUT**.
- Set the **POS.CHK*n*** position before programming **POS.CHK*n*.OUT**.

Example Program line

```
POS.COMMAND = 0
POS.CHK1.OUT = 10
POS.CHK1 = 10 * 4096
DIR = 0
GO.VEL
REM OUT1 is 0 until the motor moves 10
'revolutions then OUT1 is 1
```

See Also POS.CHK*n*

POS.COMMAND

Variable (integer)

Syntax **POS.COMMAND** = *x*

Range -134,217,728 to 134,217,727 resolver steps

NOTE

1 revolution = 4096 resolver steps.

Description **POS.COMMAND** (position command) contains the current position command and can also be set to initialize or redefine the electrical home position.

NOTE

Do not change** POS.COMMAND **after** CCW.OT, CW.OT, TARGET.POS**, or** POS.CHKn **have been programmed. These absolute position variables change value if the electrical home position is changed.

Example Program line

```
'Set electrical home position (POS.COMMAND = 0)
'where home switch, connected to INP1, transitions
'from 0 to 1. INP1 is assumed to be 0 initially.
'Set up slow move to search for edge of home switch.
ENABLE = 1
RUN.SPEED = 10 : DIR = 0
GO.VEL
WHEN INP1 = 1, ABORT.MOTION
'Wait for POS.COMMAND to reach fixed value
WHILE MOVING = 1
WEND
'Use value of POS.COMMAND to set POS.COMMAND
POS.COMMAND = POS.COMMAND - WHENPCMD
```

NOTE

Use this method instead of setting POS.COMMAND to 0 because the motor does not stop instantly when transition occurs.

See Also GO.HOME, POSITION, WHENPCMD

POS.ERROR

Variable (integer) (read only)

Syntax *x* = **POS.ERROR**

Range -134,217,728 to 134,217,727 resolver steps

1 revolution = 4096 resolver steps.

NOTE

Description **POS.ERROR** (Actual Position Error) is the difference between the position command and the actual position.

The internal software sets this variable.

NOTE

Example Program line

```
‘Record largest value of POS.ERROR during an
‘incremental move
DIM MAXERROR, TEMP AS FLOAT
MAXERROR = 0
‘Initialize MAXERROR
ENABLE = 1
INDEX.DIST = 10 * 4096
GO.INCR
WHILE MOVING = 1
    TEMP = ABS(POS.ERROR)
    IF TEMP > MAXERROR THEN
        MAXERROR = TEMP
    END IF
WEND
PRINT “Max position error during move = “;
MAXERROR
```

See Also POSITION, POS.COMMAND

POS.ERROR.MOVING

Variable (integer)

Syntax **POS.ERROR.MOVING** = *x*

Range 0 to +134,217,727 resolver steps

Default 0

1 revolution = 4096 resolver steps.

NOTE

Description **POS.ERROR.MOVING** defines the maximum value allowed for POS.ERROR when the motor is moving before a POS.ERROR interrupt is generated, if enabled.

NOTE

Before enabling the** POS.ERROR **Interrupt (Setting** INTR.POS.ERROR **= 1), set** POS.ERROR.MOVING **and** POS.ERROR.STOPPED **to non-zero values. Both variables default to 0, generating a** POS.ERROR **interrupt as soon as it is enabled.

See Also POS.ERROR, POS.ERROR.STOPPED

POS.ERROR.STOPPED

Variable (integer)

Syntax **POS.ERROR.STOPPED** = *x*

Range 0 to +134,217,727 resolver steps

Default 0

1 revolution = 4096 resolver steps.

NOTE

Description **POS.ERROR.STOPPED** defines the maximum value allowed for POS.ERROR when the motor is stopped before a POS.ERROR interrupt is generated, if enabled.

NOTE

Before enabling the** POS.ERROR **Interrupt (Setting** INTR.POS.ERROR **= 1), set** POS.ERROR.MOVING **and** POS.ERROR.STOPPED **to non-zero values. Both variables default to 0, generating a** POS.ERROR **interrupt as soon as it is enabled.

See Also POS.ERROR, POS.ERROR.MOVING

POSITION

Variable (integer) (read only)

Syntax *x* = **POSITION**

Range -134,217,727 to +134,217,727 resolver steps

NOTE

1 revolution = 4096 resolver steps.

Description **POSITION** (Actual Position) indicates the actual motor position. This is a read-only variable and cannot be set by the software. This is the absolute motor position relative to the electrical home position.

NOTE

If you change** POS.COMMAND, POSITION **and the electrical home position also change. Assigning a new value to** POS.COMMAND **does not cause the motor to move.

Example Program line

```
'Run motor at 500 rpm for 10 turns. Then increase
'speed to 1000 rpm
ENABLE = 1
'Set to move 500 rpm in the clockwise direction
RUN.SPEED = 500
DIR = 0
'Set the present commanded position to zero
POS.COMMAND = 0
GO.VEL
    RUN.SPEED = 1000
WHEN POSITION > 40960, GO.VEL
```

See Also POS.COMMAND, WHEN.POS

PRINT

Statement

Syntax **PRINT** *expression* [(,;] *expression*] [;]

Expressions can be:

- Variables
- Calculations with numeric variables and constants
- String constants enclosed in quotes

Description **PRINT** displays output on the terminal screen while the program is running. ServoBASIC Plus defines zones of 13 characters used to produce output in columns.

- If a list of expressions is separated by commas (,) or spaces ( ), each subsequent expression is printed in the next available zone.
- If a list of expressions is separated by semi-colons (;), the zones are ignored and consecutive expressions are printed in the next character space.
- If the **PRINT** statement ends with a comma or a semi-colon, the carriage return/line feed at the end of the screen output is suppressed.

Example Program line

```
INT1 = 25
PRINT 'The total is "; INT1; "so far this shift"
```

This program segments prints "The total is 25 so far this shift."

PULSES.IN

Variable (integer)

Syntax **PULSES.IN** = *x*

Range -32,768 to +32,767

Default 1,000

Description **PULSES.IN** specifies the number of input encoder counts used for selecting an exact gear ratio.

PULSES.IN and **PULSES.OUT** must be set more recently than **RATIO** for the effective ratio to be set by them.

Example Program line

```
PULSES.IN = 2
PULSES.OUT = 3
```

For every 2 encoder pulse input counts, the motor moves 3 resolver counts.

See Also PULSES.OUT, RATIO, GEARING

PULSES.OUT

Variable (integer)

Syntax **PULSES.OUT** = *x*

Range -2048 to +2047

Default 1,000

Description **PULSES.OUT** specifies the number of resolver counts the motor moves for a specified number of encoder counts. (**PULSES.IN**)

PULSES.IN and **PULSES.OUT** must be set more recently than **RATIO** for the effective ratio to be set by them.

Example Program line

```
PULSES.IN = 2
PULSES.OUT = 3
```

For every 2 encoder pulse input counts, the motor moves 3 resolver counts.

See Also PULSES.IN, RATIO, GEARING

PWM12

Variable (float)

Syntax **PWM** = *x*

Range 0 to 100 percent

Description **PWM12** specifies the function of discrete Output 12 / **PWM** as a Pulse Width Modulated (**PWM**) signal and controls the duty cycle.

The function of Output 12, as either a **PWM** signal, or as a general purpose discrete is determined by the last instruction, **PWM12** or **OUT12**, to control the output pin's function. When configured as **PWM** source, **PWM12** controls the duty cycle of Output 12 from ON (0%) to FULL OFF (100%). The frequency is fixed at 11.72 kHz.

Example Program line

```
'This program segment sets the PWM output to the
'ration of the measured motor current to that
'of the drive's peak current capability.
PWM12 = IFB/IPEAK * 100
```

See Also OUT12, OUTPUTS

RATIO

Variable (float)

Syntax **RATIO** = +*x*

Range -2,000 TO +2,000

Resolution 0.00001

Default 1.00

Description **PROGRAM** sets the electronic gearing ration (Rev/Rev) between the encoder shaft and the motor shaft. Follow these guidelines to program the **RATIO** variable:

- For and encoder input, install an encoder input from the master and verify that it is set to the correct encoder line count via **ENC.IN**
- **ENC.IN** must be non-zero to set **RATIO**.
- A negative value for **RATIO** causes motion opposite to the encoder shaft.
- The gearing ratio can be specified by either **PULSES.IN** and **PULSES.OUT** or by **RATIO**. The last parameter specified determines the gearing ratio.

Example Program line

```
'Set motor to move three revolutions for
'every one revolution of the master encoder
RATIO = 3
GEARING = 1
```

See Also GEARING, ENC.IN, PULSES.IN, PULSES.OUT

REG.DIST

Variable (integer)

Syntax **REG.DIST** = *x*

Range - 134,217,728 to 134,217,727 resolver steps.

NOTE

1 revolution = 4096 resolver steps.

Default 4096

Description **REG.DIST** (Registration Distance) sets the distance that is moved automatically when a resolver registration input is applied. This function, specified when **REG.FUNC** = 1, performs an incremental registration move with a microsecond response to the input.

NOTE

The controller must be in motion and executing a motion command to perform the registration index.

For registration input(s) information, please refer to **REG.MODE** for further details. Program **REG.FUNC** = 1 to specify allowing **REG.DIST**. Refer to **REG.FUNC** for more information.

Example Program line

```
'Beginning of the loop
WHILE 1 =1     'Prompt for Registration Distance
INPUT "Enter registration distance", REG.DIST
'Clear encoder and resolver registers
REG.FLAG = 0   'Turn registration distance flag on
REG.FUNC = 1   'Start motor
GO.VEL         'Beginning of loop 2
WHILE IN.POSITION = 0 : WEND
'* Display registration complete
PRINT "Registration Complete"
'Go back to loop 1
WEND
```

*This message displays only if a registration input was triggered and the motor has moved into position.

See Also REG.ENCPOS, REG.FLAG, REG.FUNC, REG.MODE, REG.POS, REG.RESPOS

REG.ENCPOS

Variable (integer) (read only)

Syntax *x* = **REG.ENCPOS**

Range - 2,147,483,648 to 2,147,483,647 encoder quadrature counts.

Description **REG.ENCPOS** (Registration Encoder Position) records the encoder position when and encoder registration input triggers.

Attach registration inputs to appropriate location. Refer to **REG.MODE** and **REG.FUNC** for more information.

See Also REG.DIST, REG.FLAG, REG.FUNC, REG.MODE, REG.POS, REG.RESPOS

REG.FLAG

Variable (integer) (read only)

Syntax *x* = **REG.FLAG, REG.FLAG** = 0 (Resets Flag)

Range The table below indicates which input has tripped.

REG.FLAG	
0	No trip
1	Encoder trip only
2	Resolver trip only
3	Encoder & Resolver trip

Description **REG.FLAG** (Registration Flag) indicates that the Registration input has triggered. This variable also indicates which input has triggered **REG.FLAG** must be reset to re-enable registration.

Attach registration input(s) to the appropriate location. Refer to **REG.MODE** for more information.

To clear the flag, set **REG.FLAG** = 0. This enables the controller to respond to the next registration pulse. Program **REG.DIST** for the appropriate distance after specifying **REG.FUNC** = 1. Refer to **REG.FUNC** for more information.

NOTE

Registration is triggered on the rising edge of the input signal.

See Also REG.DIST, REG.ENCPOS, REG.FUNC, REG.MODE, REG.POS, REG.RESPOS

REG.FUNC

Variable (integer)

Syntax **REG.FUNC** = *x*

Range 1 allows **REG.DIST** to move upon Registration trigger.
0 disallows **REG.DIST** to move upon Registration trigger.

Default 0

Description **REG.FUNC** (Registration Functionality) specifies whether **REG.DIST** is the distance moved automatically when a resolver registration input is applied. This function performs an incremental move with microsecond response to the input.

NOTE

The controller must be in motion and executing a motion command to perform the registration index.

- Attach registration inputs to the appropriate location. Refer to **REG.MODE** for more information.
- Set **REG.FUNC** = 1

NOTE

Registration is triggered on the rising edge of the input signal.

See Also REG.DIST, REG.ENCPOS, REG.FLAG, REG.MODE, REG.POS, REG.RESPOS

REG.MODE

Variable (integer)

Syntax **REG.MODE** = *x*

Value: The table below shows the inputs to latch the encoder or resolver based upon **REG.MODE**.

REG.MODE	Encoder Registration	Resolver Registration
0	Z	Z
1	Z	INP7
2	INP7	INP7
3	INP8	INP7

Description **REG.MODE** specifies the discrete input signal that triggers and latches the encoder and resolver positions during registration. A resolver registration trigger latches **REG.POS** and **REG.RESPOS** and permits an optional incremental move specified by **REG.DIST** and enabled by **REG.FUNC**. An encoder registration trigger latches **REG.ENCPOS**.

Attach registration input(s) to appropriate location. Refer to the *SC750 Installation and Hardware Reference Manual* for wiring information.

The Z Channel input requires a differential input signal. If using a single-ended device, use either INP7 or INP8.

NOTE

Registration is triggered on the rising edge of the input signal.

NOTE

See Also REG.DIST, REG.ENCPOS, REG.FLAG, REG.FUNC, REG.POS, REG.RESPOS

REG.POS

Variable (integer) (read only)

Syntax *x* = **REG.POS**

Range -134,217,728 to 134,217,727 resolver steps.

1 revolution = 4096 resolver steps.

NOTE

Description **REG.POS** (Registration Motor Position) records the motor position when a resolver registration input triggers.

Attach registration input(s) to appropriate location. Refer to **REG.MODE** for more information.

See Also REG.DIST, REG.ENCPOS, REG.FLAG, REG.FUNC, REG.MODE, REG.RESPOS

REG.RESPOS

Variable (integer) (read only)

Syntax *x* = **REG.RESPOS**

Range 0 to 4095 resolver steps

1 revolution = 4096 resolver steps.

NOTE

Description **REG.RESPOS** (Registration Resolver Position) records the resolver position relative to the motor housing when a resolver registration input triggers.

Attach registration input(s) to appropriate location. Refer to **REG.MODE** for more information.

See Also REG.DIST, REG.ENCPOS, REG.FLAG, REG.FUNC, REG.POS

REM OR ‘

Statement

Syntax **REM** [text of comment] OR ‘ [text of comment]

Description **REM** (Remark) includes explanatory remarks or comments in the program.

A **REM** may appear anywhere within the line and anything following the **REM** is treated as a comment. Comments may also appear at the end of any program line, by the use of the apostrophe (‘). This statement has no affect on the program.

Example Program line

```
REM This is a comment
‘So is this
RUN.SPEED = 1800
```

RESPOS

Variable (integer) (read only)

Syntax *x* = **RESPOS**

Range 0 to 4095 resolver steps

1 revolution = 4096 resolver steps.

NOTE

Description **RESPOS** (Resolver Position) contains the mechanical orientation of the resolver relative to the motor housing.

RESPOS varies from 0 to 4095 and back to zero as the motor rotates clockwise through one complete revolution.

Example Program line

```
‘ROTATE MOTOR TO A FIXED ABSOLUTE ANGLE WHERE
‘RESPOS EQUALS 1000.
ENABLE = 1
INDEX.DIST = 1000 – RESPOS
GO.INCR
END
```

RESTART

Statement

Syntax **RESTART**

Description **RESTART** clears the run time error variables and causes program execution to start again from the beginning of the program. Any Interrupts, Subroutines, **WHEN** statements or loops in process are aborted. This statement is used to continue program execution after a Run Time Error Handler or to abort from **WHEN** statements without satisfying the condition.

NOTE

RESTART ***does not clear the data area or, in itself, change any program or motion variables.***

NOTE

If the *RESTART* ***instruction is used to exit from an Error-Handler, an infinite loop occurs if the error condition is not cleared.***

See Also ERR, ERRVAL, ERRVAR, INTERRUPT, ON ERROR GOTO, WHEN

RIGHT$

String Function

Syntax **RIGHT$** *(x$, n)*

Description **RIGHT$** returns the right-most characters of the string. If *n* is equal to or greater than **LEN**, **RIGHT$** returns *x*$. If *n*=zero, a null string is returned.

Example Program line

```
DIM A$ AS STRING
A$ = "DISK ServoBASIC"
PRINT RIGHT (A$, 10)
```

Prints ServoBASIC (the rightmost 10 characters of the string A$)

See Also MID$, LEFT$

RUN.SPEED

Variable (float)

Syntax **RUN.SPEED** = *x*

Range 0 to 12,000 rpm

Resolution 0.001 rpm

Default 1,000 rpm

Description **RUN.SPEED** sets the maximum speed for an incremental or absolute move. It is also sets the command velocity for a **GO.VEL** command. Specify **RUN.SPEED** prior to issuing motion commands.

See Also DIR, GO.ABS, GO.INCR, GO.VEL, UPD.MOVE

RVEL

Variable (float) (read only)

Syntax *x* = **RVEL**

Range -13,733 rpm to +13,733 rpm

Description **RVEL** indicates the actual speed at which the motor is running. This variable is unfiltered.

See Also RUN.SPEED, VELOCITY

SGN

Function

Syntax **SGN**(*x*) *x* = any numeric expressions

Range If *x* > 0, SGN(*x*) returns 1.
If *x* >=0, SGN(*x*) returns 0.
If *x* < 0, SGN(*x*) returns -1.

Description **SGN** returns the sign of *x*.

Example Program line

```
A$ = 1.5
PRINT SGN(A$)
```

This program segment prints the value, 1.

See Also SIN

SIN

Function

Syntax **SIN**(*x*)

Description **SIN** returns the trigonometric sine of *x* in radians.

Example Program line

```
PRINT SIN(1.5)
```

This program segment prints the value 0.9974951 (the sine of 1.5 radians).

See Also COS, TAN

SPACE$

String Function

Syntax **SPACE$***(x)*

Range 0 to 255

Description **SPACE$** returns a string of spaces. *x* is rounded to an integer.

Example Program line

```
DIM N AS INTEGER
DIM X$ AS STRING
FOR N = 1 TO 5
X$ = SPACE$ (n)
NEXT N
```

The following is printed:

```
1
 2
  3
   4
    5
```

(One space is added for each loop execution.)

SQR

Function

Syntax **SQR**(*x*)

Description **SQR** returns the square root of an expression.

NOTE

x must be greater than or equal to 0.

Example Program line

```
x = 10
PRINT SQR(x)
```

This program segment prints the value 3.162278.

STATUS

Variable (integer) (read-only)

Syntax *x* = **STATUS[*axis#*]**

Range 0 = an external axis is not connected to PacLAN
1 = an external axis is connected to PacLAN

Description **STATUS** used within a program determines if an external axis is connected to PacLAN.

Example Program line
The following program checks the STATUS of all PacLAN Axes and prints a message indicating their presence to the serial I/O port.

```
DIM i as Integer
For i = 1 to 255
IF STATUS(i) = 1 THEN PRINT “Axis “;i;” is
Connected”
NEXT i
```

STEPDIR

Variable (integer)

Syntax **STEPDIR** = *x*

Range 0 or 1

Value of STEPDIR	Description
x = 0	Selects quadrature encoder pulses
x = 1	Selects step and direction input signals

NOTE

A default value of 4096 steps per revolution (*RATIO* **= 1) is used. Further scaling can be controlled using the** *RATIO* **parameter.***

Default 0

Description **STEPDIR** specifies response to either quadrature encoder signals or step and direction input signals when electronic gearing is in use.

Step and direction signals are similar to those outputs from a stepper motor indexer/driver.

- Use J52-2(+) and J52-3(-) for step inputs.
- Use J52-4(+) and J52-5(-) for direction inputs.

STEP is sensed by rising edge transitions of **STEP+** with respect to **STEP-.** Clockwise direction is commanded by a high logic signal on **DIR+** with respect to **DIR-.**

See Also GEARING, ENC.IN, RATIO

STOP

Statement

Syntax **STOP**

Description **STOP** stops the execution of the program.

See Also ABORT.MOTION

STR$

String Function

Syntax **STR$**(*x*)

Description **STR$** returns a string representation of the value of a numeric expression.

Example Program line

```
DIM A$, B$, C$ AS STRING
A$ = STR$(123)
B$ = STR$(456)
C$ = A$ + B$
PRINT C$
```

This program prints out: 123456

See Also VAL

STRING$

String Function

Syntax **STRING$** (*num_char*, *ascii_character*)
or
STRING$ (*num_char*, *string_expression*)

Description **STRING$** returns a string containing the specified number or occurrences of a character.
num_char is the desired number of occurrences of a character.
ascii_character is the ASCII code of the character.
string_expression is any string expression.

NOTE

If you provide a string, STRINGS uses the first character of the string.

Example Program line

```
DIM x$ AS STRING(10)
'x$ is initialized by HEX(45), which is a minus sign ("-").
x$ = STRING$(10, 45)
PRINT x$;"Monthly Report";x$
```

This program prints:

---------Monthly Report----------

SUB

Statement

Syntax **SUB** *subroutine name*
[*EXIT SUB*]
END SUB

Description **SUB** defines the beginning and ending of a subroutine. Subroutines are located after the END statement in the main program.

Example Program line

```
'Main program
GO.VEL
PAUSE
CALL PRTVEL
END
'Define subroutine
SUB PRTVEL
PRINT "Velocity is", VELOCITY
END SUB
```

See Also CALL

SWAP

Statement

Syntax **SWAP** *variable1*, *variable2*

Description **SWAP** exchanges the values of two variables. Any type variable may be swapped (integer, single-precision, string), but the two variables must be of the same type or a "Type mismatch" error results.

Example Program line

```
DIM A$, B$, C$ AS STRING
PRIT A$ B$ C$
SWAP A$, C$
PRINT A$ B$ C$
```

The program segment prints:
ONE FOR ALL
ALL FOR ONE

TAN

Function

Syntax **TAN**(*x*)

Description **TAN** (Tangent) returns the trigonometric tangent of *x* in radians.

See Also COS, SIN

TARGET.POS

Variable (integer)

Syntax **TARGET.POS** = *x*

Range -134,217,728 to 134,217,727 resolver steps.

1 resolver step – 1/4096 revolution.

NOTE

Default 0

Description **TARGET.POS** (Target Position) sets the target position used with the **GO.ABS** function.

The target position is the absolute position relative to the electrical home position.

Do not program a new value for** POS.COMMAND **after** TARGET.POS **has been programmed. Target Position is an absolute position variable based on the existing** POS.COMMAND **position.

NOTE

See Also GO.ABS, GO.HOME, POS.COMMAND

TIME

Variable (float)

Syntax **TIME** = *x*

Range 0 to 1,832,519.38 seconds

Resolution 0.000427 sec or 1 part in 2^{23} whichever is larger

Description **TIME** contains the current value is seconds of a free-running timer maintained by the internal software. If you enter a value for **TIME**, the timer resets to this new time and continues counting.

For example, when executing **TIME**=2, the timer resets to 2 seconds, counts up to 1,832,519.38 seconds, goes to zero, and continues counting until it rolls over to zero again.

Set **TIME** equal to a value that represents the starting time for the count. To get an accurate reading of the time of a given event (such as a switch closing), set a floating-point variable equal to **TIME** and **PRINT** that variable because the **PRINT** statement takes a relatively long time to execute.

To accurately measure time intervals less than 1 second by subtracting two consecutive **TIME** values, the **TIME** variable should be preset so **TIME** stays less than 3600 seconds (1 hour) during the time interval.

Example Program line

```
'This program segment waits 10 seconds for the position
'servo to be IN.POSITION following an incremental move
GO.INCR
TIME = 0
TIME_CHECK:
WHILE (TIME < 10) AND (IN.POSITION = 0) : WEND
IF TIME > 10 THEN
    PRINT "SERVO NOT POSITIONED IN 10 SEC"
ELSE
    PRINT "SERVO IS POSITIONED IN 10 SEC"
END IF
```

*TMENABLE*N

Variable (integer)

Syntax Disable or enable timers by setting the parameter **TMENABLE*n*** = 0|1.

Range 0 = Disable Timer
1 = Enable Timer

Default Timers Disabled

Description **TMENABLE*n*** enables or disables the programmable timers.

See Also TMRSET

*TMOUT*N

Variable (integer) (read-only)

Syntax **PROGRAM** = *x*

Description **TMOUT*n*** indicates the state of an output controlled by a programmable timer.

See Also TMRSET, OUT*n*

TMRSET

Statement

Syntax **TMRSET** (*timer#n*, LEVEL | PULSE, *delay_time*, OUT*n*= 0|1, INP*n* = 0|1, POUT*n* = 0|1)

Range *timer#n* (timer number) = 1 to 6)

LEVEL|PULSE = LEVEL (output changes state continuously)
PULSE (output changes for a fixed duration of time)

OUT*n* = 0 discrete output not activated
1 discrete output is activated

INP*n* = 0 timer trigger input source is off
1 timer trigger input source is on

POUT*n* = 0 position check output function is not logically ended with the timer trigger input source.
1 position check output function is logically ended with the timer trigger input source.

NOTE

*POUT*n ***must be left blank if unused. The state of*** *TMOUT***n** ***where*** **n** ***specifies the timer number.***

Description **TMRSET** specifies the operation of the programmable timers. Execute **TMRSET** before enabling or disabling the timer using **TMENABLE*n***.

delay_time designates the delay time, in seconds, for LEVEL timers or the pulse duration for the PULSE timer function.

POUT*n* position check output function where *n* is 1 to 12.

INP*n* timer trigger input source where *n* is 1 to 16.

See Also TMENABLE*n*

UCASE$

String Function

Syntax **UCASE$**(*x*)

Description **UCASE$** converts value to upper case.

Example Program line

```
DIM x AS STRING
x = "Washington, D.C."
x = UCASE$ (X)
PRINT X
```

This program segment prints: WASHINGTON, D.C.

See Also LCASE$

UPD.MOVE

Statement

Syntax **UPD.MOVE**

Description Updates a move in process with new variables. This allows you to change motion on-the-fly, without having to stop and restart the motion with new variables.

Update desired **ACCEL.RATE, DECEL.RATE, RUN.SPEED** and **DIR**. Issue motion before **UPD.MOVE**. **UPD.MOVE** does not work if **ACCEL.TYPE**=1.

Example Program line

```
‘This program segment establishes the current
‘motor position as 0 – electrical home, then
‘performs an absolute move to 50 revolutions
‘clockwise from electrical home using
‘specified acceleration and velocity
‘parameters: 100 rpm, 1200 rpm/sec. After
‘reaching 25 revolutions, the velocity will
‘be increased to 1000 rpm using the
‘deceleration rate of 5000 rpm/sec
‘Set up motion constraints
‘Set for trapezoidal velocity profile
ACCEL.TYPE = 0
‘Set acceleration rate equal to 1200 rpm/sec
ACCEL.RATE = 1200
‘Set deceleration rate equal to 1200 rpm /sec
DECEL.RATE = 1200
‘Set run speed equal to 100 rpm
RUN.SPEED = 100
‘Establish current position as electrical home
POS.COMMMAND = 0
‘Move motor 50 revolutions
TARGET.POS = 4096*50
GO.ABS
‘Set up motion constraints for update
RUN.SPEED = 1000
DECEL.RATE = 5000
LOOP:
WHEN POSITION > 4096*25, UPD.MOVE
WHILE MOVING = 1 : WEND
PRINT “MOVE COMPLETE”
STOP
END
```

See Also ACCEL.TYPE, ACCLE.RATE, DECEL.RATE, RUN.SPEED

VAL

String Function

Syntax **VAL**(*x$)*

Description **VAL** returns the numerical value of the string. If the first character of *x$* is not numeric, **VAL** returns zero.

See Also STR$

VEL.CMD

Variable (float) (read only)

Syntax *x* = **VEL.CMD**

Description **VEL.CMD** (Velocity Command) is the net velocity servo loop signal in rpm.

Example Program line

```
'This program segment sends the variable
'VEL.CMD to the analog output channel
'Set up Output signal selection
DACMAP = 2
```

See Also DACMAP, VELOCITY, VEL.ERR

VEL.ERR

Variable (float) (read only)

Syntax **PROGRAM** = *x* = **VEL.ERR**

Description **VEL.ERR** (Velocity Error) is the velocity servo error in rpm and is the difference between the commanded and the actual resolver velocity.

VEL.CMD – RVEL

Example Program line

```
'This program sends VEL.ERR to the analog output channel
'Set up Output signal selection
DACMAP = 3
```

See Also DACMAP, VELOCITY, VEL.CMD

VELOCITY

Variable (float) (read only)

Syntax *x* = **VELOCITY**

Range -13,733 to +13,733 rpm

Description **VELOCITY** is the actual speed the motor shaft is rotating averaged over a 128 ms interval.

Example Program line

```
'Run at commanded velocity as long as INP2 is zero.
'Indicate velocity within 1% of RUN.SPEED by setting
'OUT5 to zero
DIM TEMP AS FLOAT
ENABLE = 1
RUN.SPEED = 1000
WHILE INP2 = 0
GO.VEL
    TEMP = VELOCITY
    IF TEMP > 1.01 * RUN.SPEED THEN
        OUT5 = 1
    ELSE IF TEMP <.99 * RUN.SPEED THEN
        OUT 5 = 1
    ELSE
        OUT 5 = 0

    END IF
WEND
ABORT.MOTION
END
```

See Also RUN.SPEED

WHEN

Statement

Syntax **WHEN** *condition, action*

The condition must be:

- ANALOG.IN > value
- ANALOG.IN < value
- INPn = 1 or 0
- ENCPOS > value
- ENCPOS < value
- POS.COMMAND > value
- POS.COMMAND < value
- POSITION > value
- POSITION < value
- RVEL > value
- RVEL < value
- TIME > value
- TIME < value

The action must be:

- OUTn = 1 or 0
- RATIO = value

Any of the following:

GO.ABS
GO.HOME
GO.INCR
GO.VEL
PAUSE
ABORT.MOTIN
CONTINE (allows program execution to continue)
UPD.MOVE

Description Upon encountering **WHEN**, program execution waits until the defined condition is satisfied. Then immediately executes the action and continues with the next line of the program.

The **WHEN** statement provides latching of several variables when the **WHEN** condition is satisfied. These variables are:

WHEN.ANALOG.IN	WHEN.IFB	WHEN.RVEL
WHEN.DACMON	WHEN.PCMD	WHEN.TIME
WHEN.ENCPOS	WHEN.POS	WHEN.VELCMD
WHEN.ICMD	WHEN.RESPOS	

The software checks for the defined condition every 0.5 ms and performs the action within 0.5 ms of condition satisfaction. Follow the **WHEN** statement by the valid condition and action separated by a comma.

Example Program line

```
'Suspend program execution until input 1 goes low
WHEN INP1 = 0, CONTINUE
'Reset electrical home position
POS.COMMAND = 0
'Run at constant velocity in clockwise direction for
'10 revolutions
GO.VEL
DIR = 0
WHEN POSITION > 4096 * 10, ABORT.MOTION
PRINT WHEN.PCMD
PRINT WHEN.POS
PRINT WHEN.ENCPOS
```

See Also WHEN.ANALOG.IN, WHEN.DACMON, WHEN.ENCPOS, WHEN.ICMD, WHEN.IFB, WHEN.PCMD, WHEN.POS, X WHEN.RESPOS, WHEN.RVEL, WHEN.TIME, WHEN.VELCMD

WHEN.ANALOG.IN

Variable (float) (read only)

Syntax *x* = **WHEN.ANALOG.IN**

Range -12.50 to +12.50 volts

Description **WHEN.ANALOG.IN** (**WHEN** Analog Input) records the digitized value of the analog input channel (in volts), at the time **WHEN** statement is satisfied.

Example Program line

```
'Latch Analog input value when input 1 goes low
WHEN INP1 = 0, CONTINUE
PRINT WHEN.ANALOG.IN
```

See Also WHEN, ANALOG.IN

WHEN.DACMON

Variable (float) (read only)

Syntax *x* = **WHEN.DACMON**

Description **WHEN.DACMON** (**WHEN** DACMON) records the value of the selected, filtered variable output to the analog output channel at the time the **WHEN** statement is satisfied.

Example Program line

```
'Latch DACMON when input 1 goes low
WHEN INP1 = 0, CONTINUE
PRINT WHEN.DACMON
```

See Also DACMAP, DACMON, DMF0, WHEN

WHEN.ENCPOS

Variable (integer) (read only)

Syntax *x* = **WHEN.ENCPOS**

Range –2,147,483,648 to 2,147,483,647 external encoder quadrature counts.

Description **WHEN.ENCPOS** (**WHEN** Encoder Position) records the encoder position at the time the **WHEN** statement is satisfied. This variable is checked every 1.024 ms intervals.

Example Program line

```
'Latch encoder position when input 1 goes low
WHEN INP1 = 0, CONTINUE
PRINT WHEN.ENCPOS
```

See Also WHEN

WHEN.ICMD

Variable (float) (read only)

Syntax *x* = **WHEN.ICMD**

Description **WHEN.ICMD** (**WHEN ICMD**) records the commanded motor torque current (in amps), at the time the **WHEN** statement is satisfied.

Example Program line

```
'Latch current command when input 1 goes low
WHEN INP1 = 0, CONTINUE
PRINT WHEN.ICMD
```

See Also ICMD, WHEN

WHEN.IFB

Variable (float) (read only)

Syntax *x* = **WHEN.IFB**

Description **WHEN.IFB** (**WHEN IFB**) records the measured motor current amplitude (in amps), at the time the **WHEN** statement is satisfied. **WHEN** permits millisecond response in latching of the variable.

Example Program line

```
'Latch measured motor current when input 1 goes low
WHEN INP1 = 0, CONTINUE
PRINT WHEN.IFB
```

See Also IFB, WHEN

WHEN.PCMD

Variable (integer) (read only)

Syntax *x* = **WHEN.PCMD** (resolver steps)

1 resolver step = 1/4096 revolution.

NOTE

Description **WHEN.PCMD** (**WHEN** Position Command) specifies the motor position when the **WHEN** condition is satisfied.

Example Program line

```
'Latch position command when input 1 goes low.
WHEN INP1 = 0, CONTINUE
PRINT WHEN.PCMD
```

See Also WHEN

WHEN.POS

Variable (integer) (read only)

Syntax *x* = **WHEN.POS**

Range -134,217,728 to 134,217,727 resolver steps.

1 resolver step = 1/4096 revolution.

NOTE

Description **WHEN.POS** (**WHEN POSITION**) is set to the value of **POSITION** when the **WHEN** condition is satisfied.

This variable is set by the internal software.

NOTE

Example Program line

```
'Latch position command when input 1 goes low.
WHEN INP1 = 0, CONTINUE
PRINT WHEN.POS
```

See Also WHEN

WHEN.RESPOS

Variable (integer) (ready only)

Syntax *x* = **WHEN.RESPOS**

Range 0 to 4095 resolver steps

1 resolver step = 1/4096 revolution.

NOTE

Description **WHEN.RESPOS** (**WHEN** Resolver Position) records the resolver's mechanical orientation relative to the motor housing at the time the **WHEN** statement is satisfied.

WHEN.RESPOS varies from 0 to 4095 then back to zero as the motor rotates one revolution. **WHEN** permits ms response in latching the variable.

Example Program line

```
'Latch position command when input 1 goes low.
WHEN INP1 = 0, CONTINUE
PRINT WHEN.RESPOS
```

See Also WHEN

WHEN.RVEL

Variable (integer) (read only)

Syntax *x* = **WHEN.RVEL**

Value -13,733 to +13,733 rpm

Description **WHEN.RVEL** (**WHEN** Raw Velocity) records the raw motor velocity at the time the **WHEN** statement is satisfied.

This variable is set by the internal software.

NOTE

Example Program line

```
'Latch position command when input 1 goes low.
WHEN INP1 = 0, CONTINUE
PRINT WHEN.RVEL
```

See Also WHEN

WHEN.TIME

Variable (float) (read only)

Syntax x = **WHEN.TIME**

Range 0 to 1,833,951 seconds

Description Designates the amount of time to satisfy **WHEN**.

Example Program line

```
'Latch position command when input 1 goes low.
WHEN INP1 = 0, CONTINUE
PRINT WHEN.TIME
```

See Also TIME, WHEN

WHEN.VELCMD

Variable (float) (read only)

Syntax *x* = **WHEN.VELCMD**

Description **WHEN.VELCMD** (**WHEN** Velocity Command) records the net velocity servo loop command signal (in rpm), at the time the **WHEN** statement is satisfied. **WHEN** permits ms response in latching of the variable.

Example Program line

```
'Latch position command when input 1 goes low.
WHEN INP1 = 0, CONTINUE
PRINT WHEN.VELCMD
```

See Also VEL.CMD, WHEN

WHILE…WEND

Statement

Syntax **WHILE** expression
.
(loop statements)
.
WEND
Expression is any numeric of Boolean expression

Description **WHILE…WEND** executes a series of statements as long as an expression after the **WHILE** statement is true. The expression is evaluated again and if the expression is still true, the loop statements are executed again. This continues until the expression is no longer true. If the expression is false, the statement immediately following the **WEND** statement is executed.

WHILE…WEND loops may be nested. Each **WEND** is matched to the most recent **WHILE**. Unmatched **WHILE** or **WEND** statements cause compilation errors.

WVSHP

Variable (integer) (read only)

Syntax *x* = **WVSHP**

Description **WVSHP** (integer) is composed of the ASCII codes for the characters in the name of the motor back EMF wave shape the controller is using to shape the motor current. To convert the **WVSHP** integer back to and ASCII string:

CHR$(WVSHP)+CHR$(WVSHP/28)+CHR$(WVSHP?216)+CHR$(WVSHP/22^4)

See Also CHR$

CUSTOMER SUPPORT

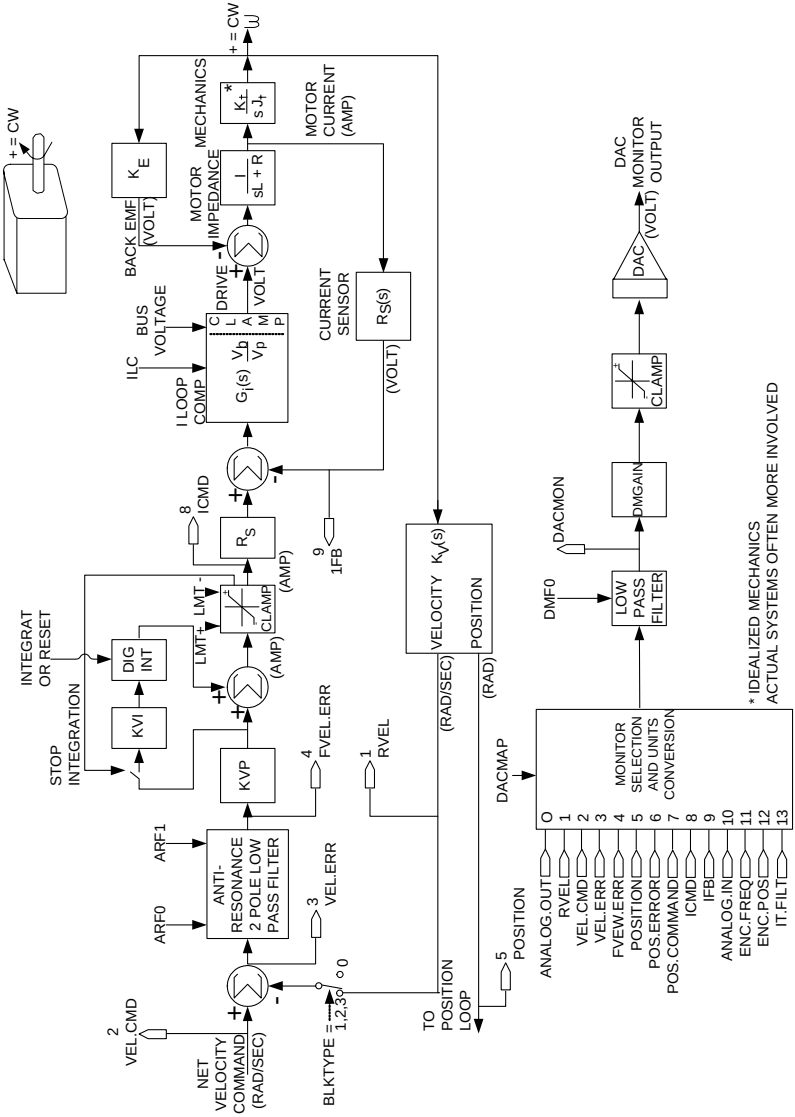
Danaher Motion products are available worldwide through an extensive authorized distributor network. These distributors offer literature, technical assistance, and a wide range of models off the shelf for the fastest possible delivery.

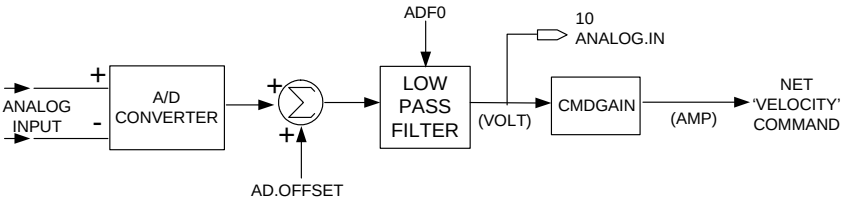
Danaher Motion sales engineers are conveniently located to provide prompt attention to customer needs. Call the nearest office for ordering and application information and assistance or for the address of the closest authorized distributor. If you do not know who your sales representative is, contact us at:

Danaher Motion
203A West Rock Road
Radford, VA 24141 USA
Phone: 1-540-633-3400
Fax: 1-540-639-4162
Email: customer.support@danahermotion.com
Website: www.DanaherMotion.com

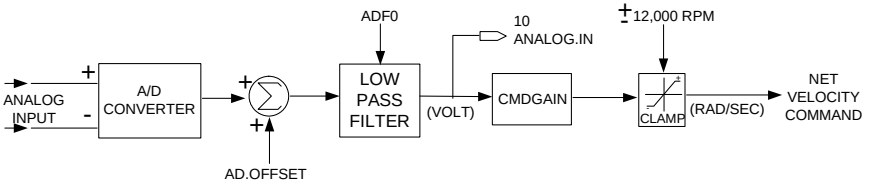
APPENDIX

SERVO LOOP

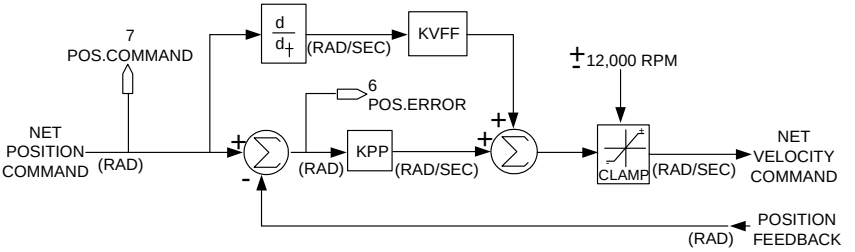




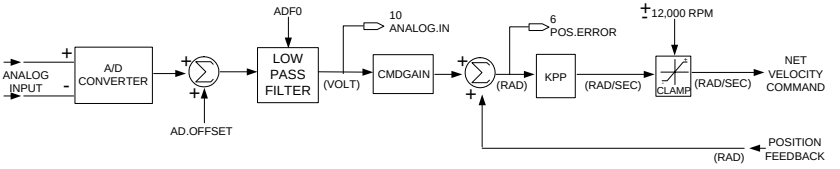
Blocktype 0: Analog Current



Blocktype 1: Analog Velocity



Blocktype 2: Servo Basic Position



Blocktype 3: Analog Position

ASCII TABLE

Dec Code		ASCII Char	Dec Code	ASCII Char	Dec Code	ASCII Char	Dec Code	ASCII Char
0	^@	NUL	32	Space	64	@	96	`
1	^A	SOH	33	!	65	A	97	a
2	^B	STX	34	\	66	B	98	b
3	^C	ETX	35	#	67	C	99	c
4	^D	EOT	36	$	68	D	100	d
5	^E	ENQ	37	%	69	E	101	e
6	^F	ACK	38	&	70	F	102	f
7	^G	BELL	39	‘	71	G	103	g
8	^H	BS	40	(	72	H	104	h
9	^I	HT	41	)	73	I	105	I
10	^J	LF	42	*	74	J	106	j
11	^K	VT	43	+	75	K	107	k
12	^L	FF	44	,	76	L	108	l
13	^M	CR	45	-	77	M	109	m
14	^N	SO	46	.	78	N	110	n
15	^O	SI	47	/	79	O	111	o
16	^P	DLE	48	0	80	P	112	p
17	^Q	DC1	49	1	81	Q	113	q
18	^R	DC2	50	2	82	R	114	r
19	^S	DC3	51	3	83	S	115	s
20	^T	DC4	52	4	84	T	116	t
21	^U	NAK	53	5	85	U	117	u
22	^V	SYNC	54	6	86	V	118	v
23	^W	ETB	55	7	87	W	119	w
24	^X	CAN	56	8	88	X	120	x
25	^Y	EM	57	9	89	Y	121	y
26	^Z	SUB	58	:	90	Z	122	z
27	^[	ESC	59	;	91	[	123	{
28	^\	FS	60	<	92	\	124	|
29	^]	GS	61	=	93	]	125	}
30	^^	RS	62	>	94	^	126	~
31	^_	US	63	?	95	_	127	

RESERVED WORDS

The following list is comprised of all ServoBASIC Plus reserved words.

ABORT.MOTION	CWINH	GEARING
ABS	CWOT	GEARLOCK
ACCEL.GEAR	DACMAP***	GO.ABS
ACCEL.RATE	DACMON	GO.HOME
ACCEL.TYPE	DBGn	GO.INCR
AD.OFFSET***	DECEL.GEAR	GO.VEL
ADF0***	DECEL.RATE	GOSUB…RETURN
ANALOG.IN	DIM	GOTO
ANALOG.OUT	DIR	HEX$
AND	DISABLE*	HWV
ARF0***	DMF0***	ICMD
ARF1***	DMGAIN***	IFB
ASC	ENABLE	IF…THEN…ELSE
ATAN	ENABLED	ILC***
AUTOSTART	ENC.FREQ	ILMT.MINUS***
AXIS.ADDR	ENC.IN	ILMT.PLUS***
AXIS.INTR	ENC.OUT	IMP
BASE	ENCPOS	InHI
BEEP	END	InLO
BLKTYPE***	EQV	INDEX.DIST
CALL	EREG**	INKEY
CCWINH	ERR	IN.POS.LIMIT
CCWOT	ERRVAL	IN.POSITION
CFGD	ERRVAR	INP*n*
CHAR	EXIT	INPUT
CHR$	EXP	INSTR
CINT	EXTFAULT	INT
CLS	FAULT	INTERRUPT
CMDGAIN***	FAULTCODE	INTR.xxx
COMMENBL	FIX	IPEAK
COMMOFF*	FOR…NEXT	ITF0***
CONST	FVEL.ERR	IT.FILT
COS	_FWDATE	IT.THRESH***
COUNTER	FWV	KPP***
COUNTSPERREV	GEARERROR	KTEFF
KVFF***	POS.CHK*n*	STEPDIR
KVI***	POS.CKK*n*.OUT	STOP
KVP***	POS.COMMAND	STR$
LANFLT(*n*)	POS.ERROR	STRING$

LANINT(*n*)	POS.ERROR.MOVING	SUB
LANINTERRUPT	POS.ERROR.STOPPED	SWAP
LBOUND	POSITION	TAN
LCASE$	PRINT	TARGET.POS
LEFT$	PULSES.IN	TIME
LEN	PULSES.OUT	TMENABLE*n*
LOG	PWM12	TMOUT*n*
LOG10	RATIO	TMRSET
LOGGEDON	REG.DIST	UBOUND
LTRIM$	REG.ENCPOS	UCASE$
MID$	REG.FLAG	UPD.MOVE
MKL$	REG.FUNC	VAL
MKS$	REG.MODE	VEL.CMD
MOD	REG.POS	VEL.ERR
MODEL	REG.RESPOS	VELOCITY
MOVING	REM	WHEN
NOT	REPOS	WHEN.ANALOG.IN
OCT$	RESTART	WHEN.DACMON
ON ERROR GOTO	RIGHT$	WHEN.ENCPOS
OPTION	RREG**	WHEN.ICMD
OR	RUN.SPEED	WHEN.IFB
OT.ERROR	RVEL	WHEN.PCMD
OUT*n*	SGN	WHEN.POS
OUTPUTS	SIG.THRESH*	WHEN.RESPOS
PARAMS	SIN	WHEN.RVEL
PAUSE	SPACES$	WHEN.TIME
PAUSE.TIME	SQR	WHEN.VELCMD
PERR	START	WHILE…WEND
POLECOUNT***	STATUS	WRITE
		WVSHP

** Factory set motor parameter*

*** Reserved for future use*

**** Non-volatile parameter*

STATUS DISPLAYS

The status display codes and their corresponding faultcode values are listed below.

SC752/SC753

Status Display	FAULTCODE	Description
0	0	No fault, disabled
1	1	Software resolver over-speed
2	2	Motor over-temperature
3	3	Servocontroller over-temperature
4	4	Servocontroller IT
5	5	L-N fault
6	6	Control under-voltage
7	7	Bus OV/OC (highest priority)
8	0	No fault, enabled
9	9	Estimated shunt regulator IT fault
b	11	Encoder +5 V low
C	12	Terminal +5 V low
E	14	Microprocessor fault
F1	241	Following error overflow
F2	242	Program memory fault
F3	243	Parameter memory fault
F4	244	Run time error
F5	245	PacLAN error
F6	246	Incompatible motion dialogue
UC		Unconfigured controller

SC754/SC755/SC756

Status Display	FAULTCODE	Description
0	0	No fault, disabled
1	1	Software resolver over-speed
2	2	Motor over-temperature
3	3	Servocontroller over-temperature
4	4	Servocontroller IT
5	5	Bus over-current
6	6	Control under-voltage
7	7	Output over-current
8	0	No fault, enabled
9	9	Measured shunt regulator IT fault
A	10	Bus over-voltage or Hot control under-voltage
b	11	Encoder +5 V low
C	12	Terminal +5 V low
d	13	Power stage control under-voltage
E	14	Microprocessor fault
F1	241	Following error overflow
F2	242	Program memory fault
F3	243	Parameter memory fault
F4	244	Run time error
F5	245	PacLAN error
F6	246	Incompatible motion dialogue
UC		Unconfigured controller

NOTE

Status displays F1, F2, F3, F4, F5, F6 and UC alternately flash between the two values.

NOTE

There is no faultcode 8 – No faults, enabled. For status display 8, the faultcode indicates 0. The variable ENABLED indicates whether the controller is enabled.

MOTION COMMAND

The table below indicates which ServoBASIC Plus variables must be used to perform certain motion moves. For each motion type listed, the variables that must be used are marked.

		ACCEL.TYPE	ACCEL.RATE	DECEL.RATE	RUN.SPEED	DIR	TARGET.POS	INDEX.DIST	GEARING	ENC.IN	RATIO	PULSES.IN	PULSESE.OUT
CONSTANT VELOCITY	GO.VEL	X	X	X	X	X							
ABSOLUTE MOVE	GO.ABS	X	X	X	X		X						
INCREMENTAL MOVE	GO.INCR	X	X	X	X			X					
HOMING MOVE	GO.HOME	X	X	X	X								
UPDATE PARAMETERS	UPD.MOVE		X	X	X	X							
ABORT MOTION	ABORT. MOTION												
ELECTRONIC GEARING									X	X	X	X	X

**Velocity moves only*

MULTIDROP SERIAL COMMUNICATIONS

NOTE

The default serial communications format for the SC750 is RS-232. The address value is axis 255 (all S1 switches in the UP or OFF position). Multidrop protocol does not apply to RS-232 communications.

SC750 MULTI-DROP PROTOCOL

A multidrop system consists of a multidrop master and 1 to 32 multidrop subsystems. Each subsystem has a unique address ranging from 0 to 254. The address of a subsystem is configured using the dip switch S1, as described in the *SC750 Installation and Hardware Reference Manual*. The subsystem address is indicated in the software variable **AXIS.ADDR**.

NOTE

Only one multidrop subsystem can transmit data back to the multidrop master at any given time.

Configuring a multidrop subsystem to transmit data to the multidrop master requires the multidrop subsystem as the unique logged on system. A variable indicating the logon status (**LOGGEDON**) is set appropriately in all the multidrop subsystems connected on the multidrop interface.

A logged on multidrop subsystem can input received data during program execution using either **INKEY$** or **INPUT**. If subsystems are not logged on, they cannot access data transmitted by the multidrop master. **INPUT** program execution effectively is suspended until the subsystem is issued a logon message and receives valid input data terminated with the carriage return character. **INKEY$** returns a null string (value of 0), if a subsystem is not logged on.

A logged on subsystem can transmit data to the multidrop master using **PRINT** while the program is executing. If a subsystem is not currently logged on, its multidrop transmitter is disabled and program execution does not halt at the **PRINT** statement.

SUBSYSTEM SELECTION

A subsystem's address is configured using the S1 dip switch located underneath the small panel on top of the controller. Setting up this switch for a particular address is described in the *SC750 Installation and Hardware Reference Manual*. The subsystem address is indicated in the **AXIS.ADDR**.

NOTE

The subsystem address S1 switch setting is polled only when power is applied to the controller. If the switch setting is modified, power must be cycled to the controller before the new address takes effect.

When a multidrop subsystem is logged on, its multidrop transmitter can be enabled whenever data must be transmitted. If a unit is not been logged on (since AC power was applied) or if a valid logon has been issued to another SC750 multidrop subsystem, the subsystem's transmitter is disabled.

The multidrop master must transmit a multidrop subsystem selection or logon command, message using the format, */nnn*,

This permits the multidrop subsystem's transmitter to be enabled. *nnn* is a valid SC750 subsystem address, ranging from 0 (all switches ON) to 254.

NOTE

Address 255 (all switches OFF) indicates the unit is configured for RS-232 serial communications.

Once the / (slash) is transmitted, the subsystem address is defined by the three digit numeric code. If there are less than three numeric digits, the address is terminated by the first non-numeric character.

When a multidrop subsystem currently logged on recognizes the selection of another subsystem, it disables its multidrop transmitter.

LOGGEDON indicates that a multidrop subsystem is selected to transmit to the multidrop master. This variable is updated in all multidrop subsystems after the multidrop master issues the logon message. This variable can be used to determine the status of a multidrop subsystem.

CHANGE THE SUBSYSTEM ADDRESS

If a multidrop master changes the subsystem address by issuing a new logon command, a subsystem previously logged on suspends its multidrop transmit and receives functions as:

- **INKEY$** does not indicate data characters received by the multidrop interface and always returns a null string (value of zero.)
- If an **INPUT** statement is encountered after the address changes, the subsystem effectively suspends program execution until it has been re-selected as the multidrop subsystem and **INPUT** receives valid data terminated with a carriage return. If a logon command is received while an **INPUT** data message is being received (embedded within the data), **INPUT** ignores the data transmitted prior to the logon command, awaits its subsystem to be addressed with a new logon command, re-prompts with a Redo from Start message, and waits for valid data to be input terminated with a carriage return.
- **PRINT** completes transmission of the most recent (single) character being output to the multidrop transmitter, prior to issuing the change of the subsystem address. The subsystem's multidrop transmitter is disabled when subsequent characters are output. However, **PRINT** continues execution regardless of the logon state.

  Due to the potential suspension or hanging of program execution, if the selected subsystem address changes while a subsystem is receiving input data, proper synchronization of the master and multidrop subsystems should be carefully developed. You may want to perform software handshaking to support communication between the multidrop master and subsystems.

RS-232

Serial data from the RS-232 RXD input is wire ORed with the multidrop (RS-485) RXD input channel. Do not use multidrop input communications simultaneously with RS-232 input sources.

Executing a **PRINT** when a multidrop subsystem is logged off results in serial data being transmitted only on the RS2-32 channel. When the subsystem is enabled, data is transmitted on both the RS-232 and multidrop (RS-485) output channels.

RUN TIME ERRORS

Error Number (ERR)	Run Time Error
1	Divisor in equation is zero
2	Too many arguments in an arithmetic expression
3	Internal failure or download failure
4	Exceeded maximum nesting level of subroutine calls and interrupt
5	RETURN statement encountered without matching GOSUB
6	Insufficient temporary memory for intermediary results of string variable processing
7	Reserved for future use
8	Nested WHEN statements (via interrupt subroutines)
9	Predefined integer variable assigned a value out of its specified range (See note below)
10	Predefined floating point variable assigned a value out of its specified range (See note below)
11	Error in argument of WHEN condition or WHEN action (i.e. WHEN condition: TIME < current time)
12	Internal failure
13	Internal failure
14	Attempted assignment to a predefined read-only integer variable
15	Attempted assignment to a predefined read-only floating point variable
16	Attempted to write to a volatile memory location outside the allocated memory space
17	Attempted to write to a non-volatile memory location outside the allocated memory space
18	Reserved for future use
19	Attempted to enable a software timer that has not been set with the TMRSET statement
20	Internal failure
21	Out of range timer number in TMRSET statement
22	Interrupt enabled without an interrupt subroutine
23	Error in argument of TMRSET statement
24	Error transmitting (writing) data to another PacLAN controller
25	Error receiving (reading) data from another PacLAN controller
26	Reserved for future use
27	Attempt to transmit a PacLAN interrupt to another controller has failed.

NOTE

When this error occurs, a message indicates a run time error as well as an index number that can be used to determine which variable was assigned a number out of its specified range. The following tables cross-reference the index numbers to the variables.

Pre-defined Floating-Point Variable Cross-Reference			
Index Number (ERRVAR)	**Predefined Variable Name**	**Index Number (ERRVAR)**	**Predefined Variable Name**
0	ANALOG.IN	21	DMGAIN
1	ANALOG.OUT	22	CMDGAIN
2	ENC.FREQ	23	COMMOFF
3	PAUSE.TIME	24	PWM12
4	RATIO	25	IT.FILT
5	TIME	26	VEL.CMD
6	VELOCITY	27	VEL.ERR
7	RUN.SPEED	28	ICMD
8	ARF0	29	IFB
9	ARF1	30	FVEL.ERR
10	KVI	31	DACMON
11	ITF0	32	WHEN.TIME
12	SIG.THRESH	33	WHEN.RVEL
13	SIX.THRESH	34	RVEL
14	KPP	35	KTEFF
15	KVP	36	ERRVAL
16	KVFF	37	WHEN.ANALOG.IN
17	DMF0	38	WHEN.DACMON
18	ADF0	39	WHEN.ICMD
19	AD.OFFSET	40	WHEN.IFB
20	IPEAK	41	WHEN.VELCMD

Pre-defined Integer Variable Cross-Reference			
Index Number (ERRVAR)	**Predefined Variable Name**	**Index Number (ERRVAR)**	**Predefined Variable Name**
0	ACCEL.RATE	44	OUT10
1	ACCEL.TYPE	45	OUT11
2	CCWOT	46	OUT12
3	Internal use	47	OUTPUTS
4	ILMT.MINUS	48	POS.CHK1
5	ILMT.PLUS	49	POS.CHK2
6	CWOT	50	POS.CHK3
7	DACMAP	51	POS.CHK1.OUT
8	DECEL.RATE	52	POS.CHK2.OUT
9	DIR	53	POS.CHK3.OUT
10	ENABLE	54	POS.COMMAND
11	ENABLED	55	POS.ERROR
12	ENC.IN	56	RESPOS
13	ENCPOS	57	POSITION
14	PAULTCODE	58	PULSES.IN
15	GEARING	59	PULSES.OUT
16	INDEX.DIST	60	REG.DIST
17	INP1	61	REG.FLAG

Pre-defined Integer Variable Cross-Reference			
Index Number (ERRVAR)	**Predefined Variable Name**	**Index Number (ERRVAR)**	**Predefined Variable Name**
18	INP2	62	REG.ENCPOS
19	INP3	63	REG.POS
20	INP4	64	REG.RESPOS
21	INP5	65	TARGET.POS
22	INP6	66	WHEN.ENCPOS
23	INP7	67	WHEN.POS
24	INP8	68	WHEN.RESPOS
25	INP9	69	ENC.OUT
26	INP10	70	REG.MODE
27	INP11	71	STEPDIR
28	INP12	72	Reserved for future use
29	INP13	73	Reserved for future use
30	INP14	74	Reserved for future use
31	INP15	75	Reserved for future use
32	INP16	76	IN.POS.LIMIT
33	INPUTS	77	MODEL
34	Internal use	78	AXIS.ADDR
35	OUT1	79	MOVING
36	OUT2	80	WHEN.PCMD
37	OUT3	81	IN.POSITION
38	OUT4	82	IT.THRESH
39	OUT5	83	REG.FUNC
40	OUT6	84	FWV
41	OUT7	85	BLKTYPE
42	OUT8	86	Internal use
43	OUT9	87	Internal use
88	Internal use	128	INTR.CWINH
89	AUTO START	129	INTR.CCWINH
90	COUNTER	130	HMV
91	INTR.I1LO	131	COMMENBL
92	INTR.I1HI	132	LOGGEDON
93	INTR.I2LO	133	EXTFAULT
94	INTR.I2HI	134	WVSHP
95	INTR.I3LO	135	Internal Use
96	INTR.I3HI	136	ERR
97	INTR.I4LO	137	ERRVAR
98	INTR.I4HI	138	STATUS
99	INTR.I5LO	139	POUT1
100	INTR.I5HI	140	POUT2
101	INTR.I6LO	141	POUT3
102	INTR.I6HI	142	TMENABLE1
103	INTR.I7LO	143	TMENABLE2
104	INTR.I7HI	144	TMENABLE3
105	INTR.I8LO	151	TMOUT4

Pre-defined Integer Variable Cross-Reference			
Index Number (ERRVAR)	**Predefined Variable Name**	**Index Number (ERRVAR)**	**Predefined Variable Name**
106	INTR.I8HI	152	TMOUT5
107	INTR.I9LO	153	TMOUT6
108	INTR.I9HI	154	GEARLOCK
109	INTR.I10LO	155	GEARERROR
110	INTR.I10HI	156	Internal Use
111	INTR.I11LO	157	Internal Use
112	INTR.I11HI	158	Internal Use
113	INTR.I12LO	159	INTR.POS.ERR
114	INTR.I12HI	160	Internal Use
115	INTR.I13LO	161	Internal Use
116	INTR.I13HI	162	INTR.PACLAN
117	INTR.I14LO	163	CWINH
118	INTR.I14HI	164	CCWINH
119	INTR.I15LO	165	COUNTSPERREV
120	INTR.I15HI	166	AXIS.INTR
121	INTR.I16LO	167	ACCEL.GEAR
122	INTR.I16HI	168	DECEL.GEAR
123	INTR.CHAR	169	Internal use
124	INTR.CWOT	170	Internal use
125	INTR.CCWOT	171	POS.ERROR.MOVING
126	INTR.DISABLE	172	POS.ERROR.STOPPED
127	INTR.FAULT		

INDEX

P

R

S

T

U

V

W