(815) 226-2222
Fax (815) 226-3148
Email: customer.support@danahermotion.com
Web: www.danahermotion.com
Part# 903-075312-80
List Price $45 U.S.
May, 2003
Rev F

S C 7 5 0 S E R I E S

Programmable, Digital Brushless Servocontroller

ServoBASIC *Plus*™ Reference Manual

Version 2.8

WARRANTY AND LIMITATION OF LIABILITY

Includes software provided by Danaher Motion Pacific Scientific

Danaher Motion Pacific Scientific warrants its motors and controllers ("Product(s)") to the original purchaser (the "Customer"), and in the case of original equipment manufacturers or distributors, to their original consumer (the "Customer") to be free from defects in material and workmanship and to be made in accordance with Customer's specifications which have been accepted in writing by Danaher Motion Pacific Scientific. In no event, however, shall Danaher Motion Pacific Scientific be liable or have any responsibility under such warranty if the Products have been improperly stored, installed, used or maintained, or if customer has permitted any unauthorized modifications, adjustments, and/or repairs to such Products. Danaher Motion Pacific Scientific's obligation hereunder is limited solely to repairing or replacing (at its option), at its factory any Products, or parts thereof, which prove to Danaher Motion Pacific Scientific's satisfaction to be defective as a result of defective materials or workmanship, in accordance with Pacific Scientific's stated warranty, provided, however, that written notice of claimed defects shall have been given to Danaher Motion Pacific Scientific within two (2) years after the date of the product date code that is affixed to the product, and within thirty (30) days from the date any such defect is first discovered. The products or parts claimed to be defective must be returned to Danaher Motion Pacific Scientific, transportation prepaid by Customer, with written specifications of the claimed defect. Evidence acceptable to Danaher Motion Pacific Scientific must be furnished that the claimed defects were not caused by misuse, abuse, or neglect by anyone other than Danaher Motion Pacific Scientific.

Danaher Motion Pacific Scientific also warrants that each of the Control Software Programs ("Program(s)") will, when delivered, conform to the specifications therefore set forth in Danaher Motion Pacific Scientific's specifications manual. Customer, however, acknowledges that these Programs are of such complexity and that the Programs are used in such diverse equipment and operating environments that defects unknown to Danaher Motion Pacific Scientific may be discovered only after the Programs have been used by Customer. Customer agrees that as Danaher Motion Pacific Scientific's sole liability, and as Customer's sole remedy, Pacific Scientific will correct documented failures of the Programs to conform to Danaher Motion Pacific Scientific's specifications manual. DANAHER MOTION PACIFIC SCIENTIFIC DOES NOT SEPARATELY WARRANT THE RESULTS OF ANY SUCH CORRECTION OR WARRANT THAT ANY OR ALL FAILURES OR ERRORS WILL BE CORRECTED OR WARRANT THAT THE FUNCTIONS CONTAINED IN PACIFIC SCIENTIFIC'S PROGRAMS WILL MEET CUSTOMER'S REQUIREMENTS OR WILL OPERATE IN THE COMBINATIONS SELECTED BY CUSTOMER. This warranty for Programs is contingent upon proper use of the Programs and shall not apply to defects or failure due to: (i) accident, neglect, or misuse; (ii) failure of Customer's equipment; (iii) the use of software or hardware not provided by Danaher Motion Pacific Scientific; (iv) unusual stress caused by Customer's equipment; or (v) any party other than Danaher Motion Pacific Scientific who modifies, adjusts, repairs, adds to, deletes from or services the Programs. This warranty for Programs is valid for a period of ninety (90) days from the date Danaher Motion Pacific Scientific first delivers the Programs to Customer.

THE FOREGOING WARRANTIES ARE IN LIEU OF ALL OTHER WARRANTIES (EXCEPT AS TO TITLE), WHETHER EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR OF FITNESS FOR ANY PARTICULAR PURPOSE, AND ARE IN LIEU OF ALL OTHER OBLIGATIONS OR LIABILITIES ON THE PART OF DANAHER MOTION PACIFIC SCIENTIFIC. DANAHER MOTION PACIFIC SCIENTIFIC'S MAXIMUM LIABILITY WITH RESPECT TO THESE WARRANTIES, ARISING FROM ANY CAUSE WHATSOEVER, INCLUDING WITHOUT LIMITATION, BREACH OF CONTRACT, NEGLIGENCE, STRICT LIABILITY, TORT, WARRANTY, PATENT OR COPYRIGHT INFRINGEMENT, SHALL NOT EXCEED THE PRICE SPECIFIED OF THE PRODUCTS OR PROGRAMS GIVING RISE TO THE CLAIM, AND IN NO EVENT SHALL PACIFIC SCIENTIFIC BE LIABLE UNDER THESE WARRANTIES OR OTHERWISE, EVEN IF DANAHER MOTION PACIFIC SCIENTIFIC HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING WITHOUT LIMITATION, DAMAGE OR LOSS RESULTING FROM INABILITY TO USE THE PRODUCTS OR PROGRAMS, INCREASED OPERATING COSTS RESULTING FROM A LOSS OF THE PRODUCTS OR PROGRAMS, LOSS OF ANTICIPATED PROFITS, OR OTHER SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER SIMILAR OR DISSIMILAR, OF ANY NATURE ARISING OR RESULTING FROM THE PURCHASE, INSTALLATION, REMOVAL, REPAIR, OPERATION, USE OR BREAKDOWN OF THE PRODUCTS OR PROGRAMS, OR ANY OTHER CAUSE WHATSOEVER, INCLUDING NEGLIGENCE.

The foregoing shall also apply to Products, Programs, or parts for the same which have been repaired or replaced pursuant to such warranty, and within the period of time, in accordance with Danaher Motion Pacific Scientific's date of warranty.

No person, including any agent, distributor, or representative of Danaher Motion Pacific Scientific, is authorized to make any representation or warranty on behalf of Danaher Motion Pacific Scientific concerning any Products or Programs manufactured by Danaher Motion Pacific Scientific, except to refer purchasers to this warranty.

Table of Contents

1 Conventions 1-1

1.1 User-Defined Variable Names 1-1

1.2 Characters 1-2

1.3 Operators Used in Programming 1-2

1.4 Labels 1-4

1.5 Notation Conventions 1-5

1.6 ServoBASIC *Plus*™ Instruction Types 1-5

1.7 BASIC Language Instructions 1-7

1.7.1 BASIC Language Statements 1-7

1.7.2 BASIC Language Functions 1-9

2 Quick Reference of ServoBASIC *Plus* Instructions 2-1

3 ServoBASIC *Plus* Instructions (In Alphabetical Order) 3-1

Appendix A: Servo Loop A-1

Appendix B: ASCII Codes B-1

Appendix C: Reserved Words C-1

Appendix D: Faultcodes D-1

Appendix E: Motion Commands E-1

Appendix F: Multidrop Serial Communications F-1

Appendix G: Run Time Errors G-1

Index

1 Conventions

Introduction

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

Introduction

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

- INTEGER
- 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.

Note: ***Variable names are not case sensitive.***

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 *Plu*s. 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

Introduction

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; that is, multiplication is performed before addition, and so on. Also, integer division is not supported.***

Relational

Relational operators are used in IF-THEN-ELSE, WHILE-WEND, and FOR-NEXT statements. The 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_1
>=	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.

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 ServoBASIC *Plus* Instruction Types

Introduction

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 “Quick Reference” 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 an ASCII code to its equivalent screen display character. PacSci ServoBASIC *Plus* also supports string manipulation functions.

Pre-defined Variables

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 the 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 points* — 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

- *Integers* — 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 dependent 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

Program flow and decision-making control instructions are performed by BASIC language statements. 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.

BASIC language statements

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, executes 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)	Includes 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

Introduction

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 an angle.
SQR( )	Returns square root of an expression.
TAN( )	Returns the tangent of an angle.

String functions

String 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

Introduction This section contains functions, parameters, statements and variables for PacSci 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 Value	Page #
ABORT.MOTION	statement		3-2
ABS	function		3-3
ACCEL.GEAR	variable (integer)	16,000,000	3-4
ACCEL.RATE	variable (integer)	10,000	3-6
ACCEL.TYPE	variable (integer)	0	3-8
AD.OFFSET	parameter (float)	set up	3-10
ADF0	parameter (float)	set up	3-11
ANALOG.IN	variable (float R/O)		3-12
ANALOG.OUT	variable (float)	0	3-13
ARF0	parameter (float)	set up	3-14
ARF1	parameter (float)	set up	3-16
ASC( )	string function		3-18
ATAN	function		3-19
AUTOSTART	parameter (integer)	set up	3-20
AXIS.ADDR	variable (integer R/O)		3-21
AXIS.INTR	variable (integer R/O)		3-23
BEEP	statement		3-24
BLKTYPE	parameter (integer)	set up	3-25
CALL	statement		3-27
CCWINH	variable (integer R/O)		3-28
CCWOT	variable (integer)		3-29
CHR$( )	string function		3-30

Name	Type	Default Value	Page #
CINT	function		3-31
CLS	statement		3-32
CMDGAIN	parameter (float)	set up	3-33
CONST	statement		3-35
COS	function		3-36
COUNTER	variable (integer)		3-37
COUNTSPERREV	variable (integer)	4096	3-38
CWINH	variable (integer R/O)		3-39
CWOT	variable (integer)		3-40
DACMAP	parameter (integer)	set up	3-41
DACMON	variable (float R/O)		3-43
DECEL.GEAR	variable (integer)	16,000,000	3-45
DECEL.RATE	variable (integer)	10,000	3-47
DIM	statement		3-49
DIR	variable (integer)	0	3-52
DMF0	parameter (float)	set up	3-53
DMGAIN	parameter (float)	set up	3-54
ENABLE	variable (integer)	0	3-56
ENABLED	variable (integer R/O)		3-57
ENC.FREQ	variable (float R/O)		3-59
ENC.IN	variable (integer)	1024	3-60
ENC.OUT	variable (integer)	0	3-62
ENCPOS	variable (integer)		3-63
END	statement		3-65
ERR	variable (integer R/O)		3-66
ERRVAL	variable (float R/O)		3-67
ERRVAR	variable (integer R/O)		3-68
EXIT	statement (integer)		3-69
FAULTCODE	variable (integer)		3-70
FIX	function		3-73
FOR...NEXT	statement (integer)		3-74
FVEL.ERR	variable (float R/O)		3-76

Name	Type	Default Value	Page #
FWV	variable (integer R/O)		3-77
GEARERROR	variable (integer)		3-78
GEARING	variable (integer)	0	3-80
GEARLOCK	variable (integer R/O)		3-82
GO.ABS	statement		3-84
GO.HOME	statement		3-86
GO.INCR	statement		3-88
GO.VEL	statement		3-90
GOSUB...RETURN	statement		3-93
GOTO	statement		3-94
HEX$( )	string function		3-95
ICMD	variable (float R/O)		3-96
IFB	variable (float R/O)		3-97
IF...THEN...ELSE	statement		3-98
ILC	parameter (integer)	set up	3-101
ILMT.MINUS	parameter (integer)	set up	3-103
ILMT.PLUS	parameter (integer)	set up	3-104
INDEX.DIST	variable (integer)	4096	3-105
INKEY$( )	string function		3-106
IN.POS.LIMIT	variable (integer)	5	3-108
IN.POSITION	variable (integer R/O)		3-109
INPn	variable (integer R/O)		3-111
INPUT	statement		3-112
INPUTS	variable (integer R/O)		3-114
INSTR( )	string function		3-115
INT( )	function		3-116
INTERRUPT	statement		3-117
INTR.{Source Label}	variable (integer)	0	3-121
IPEAK	variable (float R/O)		3-123
ITF0	parameter (float)	set up	3-124
IT.FILT	variable (float R/O)		3-125

Name	Type	Default Value	Page #
IT.THRESH	parameter (float)	set up	3-127
KPP	parameter (float)	set up	3-129
KTEFF	variable (float)	set up	3-131
KVFF	parameter (float)	set up	3-132
KVI	parameter (float)	set up	3-134
KVP	parameter (float)	set up	3-136
LANFLT(n)	variable (float)	0	3-138
LANINT(n)	variable (integer)	0	3-139
LANINTERRUPT	statement		3-140
LCASE$( )	string function		3-141
LEFT$( )	string function		3-142
LEN( )	string function		3-143
LOG( )	function		3-144
LOG10( )	function		3-145
LOGGEDON	variable (integer)		3-146
LTRIM$( )	string function		3-147
MID$( )	string function		3-148
MOD	arithmetic operator		3-149
MODEL	variable (integer R/O)		3-150
MOVING	variable (integer R/O)		3-151
ON ERROR GOTO	statement		3-153
OUTn	variable (integer)	1	3-155
OUTPUTS	variable (integer)	4095	3-156
PAUSE	statement		3-157
PAUSE.TIME	variable (float)	1.00	3-158
POLECOUNT	parameter (integer)	set up	3-159
POS.CHKn	variable (integer)	0	3-160
POS.CHKn.OUT	variable (integer)	0	3-162
POS.COMMAND	variable (integer)		3-164
POS.ERROR	variable (integer R/O)		3-166
POS.ERROR.MOVING	variable (integer)	0	3-168
POS.ERROR.STOPPED	variable (integer)	0	3-169
POSITION	variable (integer R/O)		3-170
PRINT	statement		3-172
PULSES.IN	variable (integer)	1,000	3-173

Name	**Type**	**Default Value**	**Page #**
PULSES.OUT	variable (integer)	1,000	3-174
PWM12	variable (float)		3-175
RATIO	variable (float)	1.00	3-176
REG.DIST	variable (integer)	4096	3-178
REG.ENCPOS	variable (integer R/O)		3-180
REG.FLAG	variable (integer)		3-181
REG.FUNC	variable (integer)	0	3-183
REG.MODE	variable (integer)	0	3-184
REG.POS	variable (integer R/O)		3-186
REG.RESPOS	variable (integer R/O)		3-187
REM	statement		3-188
RESPOS	variable (integer R/O)		3-189
RESTART	statement		3-190
RIGHT$( )	string function		3-191
RUN.SPEED	variable (float)	1,000	3-192
RVEL	variable (float R/O)		3-193
SGN( )	function		3-194
SIN( )	function		3-195
SPACE$( )	string function		3-196
SQR( )	function		3-197
STATUS[AXIS#]	variable (integer R/O)		3-198
STEPDIR	variable (integer)	0	3-199
STOP	statement		3-201
STR$( )	string function		3-202
STRING$( )	string function		3-203
SUB	statement		3-204
SWAP	statement		3-205
TAN	function		3-207
TARGET.POS	variable (integer)	0	3-208
TIME	variable (float)		3-209
TMENABLEn	variable (integer)	0	3-211
TMOUTn	variable (integer R/O)		3-212
TMRSET	statement		3-213
UCASE$( )	string function		3-214
UPD.MOVE	statement		3-215

Name	Type	Default Value	Page #
VAL( )	string function		3-217
VEL.CMD	variable (float R/O)		3-218
VEL.ERR	variable (float R/O)		3-219
VELOCITY	variable (float R/O)		3-220
WHEN	statement		3-222
WHEN.ANALOG.IN	variable (float R/O)		3-225
WHEN.DACMON	variable (float R/O)		3-226
WHEN.ENCPOS	variable (integer R/O)		3-227
WHEN.ICMD	variable (float R/O)		3-228
WHEN.IFB	variable (float R/O)		3-229
WHEN.PCMD	variable (integer R/O)		3-230
WHEN.POS	variable (integer R/O)		3-231
WHEN.RESPOS	variable (integer R/O)		3-232
WHEN.RVEL	variable (float R/O)		3-233
WHEN.TIME	variable (float R/O)		3-234
WHEN.VELCMD	variable (float R/O)		3-235
WHILE...WEND	statement		3-236
WVSHP	variable (integer R/O)		3-237

Note: ***Parameters with the default value "set up" have default values that are established in the configuration set up of Motion Dialogue.***

3 ServoBASIC *Plus* Instructions

Introduction

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:

Purpose: The purpose of the instruction

Syntax: The complete notation of the instruction

Related instructions: Other ServoBASIC *Plus* commands that are similar to this particular instruction

Programming guidelines: Pertinent information about the instruction and its use in ServoBASIC *Plus*

Example program: Possible use of the instruction in a program

ABORT.MOTION

statement

Purpose

`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.

Syntax

`ABORT.MOTION`

Related instructions

`ILMT.MINUS` — sets the maximum commanded current in the counter-clockwise direction.

`ILMT.PLUS` — sets the maximum commanded current in the clockwise direction.

Program segment

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
```

ABS( )
function

Purpose

`ABS(x)`, 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.

Syntax

`ABS(x)`

Programming guidelines

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

Program segment

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)

Purpose

`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.

Syntax

`x = ACCEL.GEAR or ACCEL.GEAR = x`

where x is the controlled acceleration in rpm/sec

Range	x = 1 to 16,000,000 rpm/sec
Resolution	1 rpm/sec
Default	x = 16,000,000

Related instructions

`GEARING` - turns electronic gearing on or off.

`DECEL.GEAR` -specifies deceleration rate when gearing is turned OFF.

`GEARERROR` - specifies amount of position lag accumulated when electronic gearing is turned on.

`GEARLOCK` - indicates slave axis velocity is synchronized with motor.

Programming guidelines

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`.***

Program segment

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
```

ACCEL.RATE

variable

(integer)

Purpose	`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.*

Syntax	`ACCEL.RATE = x` where x is the desired acceleration rate in rpm/sec
Range	x = 1 to 16,000,000 rpm/sec
Resolution	1 rpm/sec
Default	x = 10,000

Related instructions

`ACCEL.TYPE` — specifies S-Curve (constant acceleration) velocity profiles.

`DECEL.RATE` — limits the maximum commanded deceleration rate.

`GO.ABS` — causes motor to move to the position specified by `TARGET.POS.`

`GO.HOME` — moves the motor shaft to the electrical home position.

`GO.INCR` — moves the motor shaft an incremental index from the current position.

`GO.VEL` — moves the motor shaft at a constant speed.

`RUN.SPEED` — sets the commanded velocity.

UPD.MOVE — updates the commanded motion (currently in progress) using specified `ACCEL.RATE`, `DECEL.RATE`, and `RUN.SPEED`.

Programming guidelines

- 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.**

Program segment

<u>Program line</u>

```
'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
```

ACCEL.TYPE

variable

(integer)

Purpose	`ACCEL.TYPE` (Acceleration Type) determines the use of constant acceleration or S-Curve velocity profiles. **Note:** ***S-Curve velocity profiles are only supported for positioning moves***.
Syntax	`ACCEL.TYPE = x`
Value	x = 0 for constant acceleration motion (trapezoidal) profiles x = 1 for S-curve velocity profiles
Default	x = 0

Related instructions

`ABORT.MOTION` — stops motion using available motor torque limited by current limit (`ILMT.PLUS, ILMT.MINUS`) parameters.

`ACCEL.RATE` — limits the maximum commanded acceleration rate.

`DECEL.RATE` — limits the maximum commanded deceleration rate.

`GO.ABS` — causes motor to move to the position specified by `TARGET.POS`.

`GO.HOME` — moves the motor shaft to the electrical home position.

`GO.INCR` — moves the motor shaft an incremental index from the current position.

`RUN.SPEED` — sets the commanded velocity.

`UPD.MOVE` — updates the commanded motion (currently in progress) using specified `ACCEL.RATE, DECEL.RATE`, and `RUN.SPEED`.

Programming guidelines

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.***

AD.OFFSET

parameter

(float)

Purpose

`AD.OFFSET` specifies the level of a signal summed with the digitized value of the analog input channel, in volts.

Syntax

`AD.OFFSET = x`

Range

- 12.50 < x < + 12.50 volts

Default

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

Related instructions

`ANALOG.IN` — contains the digitized value of the analog input channel.

`BLKTYPE` — specifies configuration as a position, velocity or torque block.

`CMDGAIN` — controls scale factor of analog input signal.

Programming guidelines

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

ADF0

parameter

(float)

Purpose

`ADF0` sets the analog input channel's filter corner frequency.

Syntax

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

Range

.011 to 12,222,726 (Hz)

Default

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

Related instructions

`ANALOG.IN` — contains the digitized value of the analog input channel.

`BLKTYPE` — specifies configuration as a position, velocity or torque block.

Programming guidelines

`ADF0` 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 digitized input signal.

Program segment

<u>Program line</u>

```
'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
```

ANALOG.IN

variable

(float)

(read only)

Purpose

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

Syntax

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

Range

x = -12.50 to +12.50 volts

Related instructions

`ANALOG.OUT` — sets the voltage level of the analog output channel.

`BLKTYPE` — specifies configuration as a position, velocity, or torque block.

`CMDGAIN` — controls scale factor of analog input signal.

`ADF0` — sets the analog input channels filter corner frequency.

Program segment

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 will continuously print out the message:

```
The Analog Input signal measures 1.000000 volts
```

ANALOG.OUT

variable

(float)

Purpose

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

Syntax

ANALOG.OUT = x

Range

-5.00 volts < x < 5.00 volts

Default

x = 0

Related instructions

DACMAP — selects signal sent to the analog output channel.

Programming guidelines

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

Program segment

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
```

ARF0

parameter

(float)

Purpose

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

Syntax

`ARF0 = x` where x is the corner frequency in Hz

Range

0.011 to 12,222,726 (Hz)

Default

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

Related instructions

`ARF1` — stage 1 anti-resonant filter break frequency.

`ILC` — sets the current loop gain compensation control.

`KVP` — sets the proportional gain of the velocity servo loop.

`KVI` — sets the integral gain of the velocity servo loop.

`KPP` — sets the proportional gain of the position loop.

`KVFF` — sets the amount of velocity feedforward signal applied to the position loop.

Programming guidelines

`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.

IMPORTANT NOTE

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.

Program segment

Program line

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

ARF0 = ANTIRES0
```

ARF1

parameter

(float)

Purpose

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

Syntax

`ARF1 = x` where x is the corner frequency in Hz

Range

0.011 to 12,222,726 (Hz)

Default

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

Related instructions

`ARF0` — stage 0 anti-resonant filter break frequency.

`ILC` — sets the current loop gain compensation control.

`KVP` — sets the proportional gain of the velocity.

`KVI` — sets the integral gain of the velocity.

`KPP`— sets the proportional gain of the position loop.

`KVFF` — sets the amount of velocity feedforward signal applied to the position loop.

Programming guidelines

`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.

IMPORTANT NOTE

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.**

Program segment

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

 ARF1 = ANTIRES1
```

ASC( )

string function

Purpose

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

Syntax

`ASC(x$)`

Related instructions

`CHR$` — converts a decimal ASCII code to its equivalent character to display on terminal.

Programming guidelines

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.***

Program segment

<u>Program line</u>

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

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

ATAN( )
function

Purpose

`ATAN` (arc tangent) returns the arc tangent of x in radians.

Syntax

```
ATAN(x)
```

Programming guidelines

The result is within the range of -π/2 to π/2.

The expression x may be any numeric type.

To convert from degrees to radians, multiply by π/180.

π/180 = 0.01745329

AUTOSTART

parameter

(integer)

Purpose

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

Syntax

```
AUTOSTART = x
```

x = 0 — AUTOSTART disabled, sign-on message displayed

x = 1 — AUTOSTART enabled, sign-on message displayed

x = 2 — AUTOSTART disabled, sign-on message surpressed

x = 3 — AUTOSTART disabled, sign-on message surpressed

Default

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

Programming guidelines

`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)

Purpose

`AXIS.ADDR` indicates the RS-485 or Multidrop address set by switch S1 on the controller.

Syntax

`x = AXIS.ADDR`

Range

1 to 255 (0 reserved for PacLAN globals)

(255 is the default RS-232 address)

Related instructions

`LOGGEDON` — indicates that the multidrop master has enabled multidrop serial communications with the multidrop master.

Programming guidelines

This variable can be inspected to insure the multidrop 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***.

AXIS.ADDR (continued)

Program segment

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
```

AXIS.INTR

variable

(integer)

(read only)

Purpose

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

Syntax

```
x  =  AXIS.INTR
```

Range

x = 1 to 255

Related instructions

`LANINTERRUPT[AXIS#]` - invokes an interrupt to the PacLAN controller specified by [Axis#].

Programming guidelines

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.

BEEP

statement

Purpose `BEEP` transmits a speaker beep command to the serial port.

Syntax

```
BEEP
```

BLKTYPE

parameter

(integer)

Purpose

`BLKTYPE` specifies configuration as a position, velocity, or torque block.

Syntax

`BLKTYPE = x`

Default

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

Related instructions

`CMDGAIN` — controls the scale factor of the analog input signal.

Programming guidelines

`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 will also provide alternative `BLKTYPE` settings.

Note: ***The servocontroller will be disabled whenever* BLKTYPE *is changed. The* ENABLE *variable must be set equal 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 the variable `CMDGAIN` as follows:

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
```

The parameter `CMDGAIN` (default value is defined within the parameter set up program section - typically 1.00) can be modified to customize the scale factor as required.

Program segment

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
```

CALL
statement

Purpose

CALL transfers program execution to a BASIC subroutine.

Syntax

```
CALL subroutine name
```

Related instructions

SUB — marks the beginning and the end of a subroutine.

Program segment

Program line

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

PRINT "This is Main Program"
CALL PRINTSUB
END

SUB PRINTSUB
PRINT "This is a Subroutine"
END SUB
```

CCWINH

variable

(integer)

(read only)

Purpose `CCWINH` indicates the current state of the `CCWINH` (`INH-`) input.

Syntax `x = CCWINH`

Value 0 or 1

Related instructions

`CCWOT` — sets the counter-clockwise overtravel limit.

`CWOT` — sets the clockwise overtravel limit.

`CWINH` — indicates the current state of the `CWINH` (`INH+`) input.

CCWOT

variable

(integer)

Purpose

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

Syntax `CCWOT = x`

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

Default 0

Related instructions

`CWOT` — sets the clockwise overtravel limit

Programming guidelines

`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.**

CHR$( )

string function

Purpose

`CHR$` converts an ASCII code to its equivalent character.

Syntax

```
CHR$ (n)
```

Related instructions

`ASC` — returns a decimal numeric value that is the ASCII code for the first character of the string expression.

Programming guidelines

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.***

Program segment

Program line

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

The upper case letter B is printed.

CINT( )
function

Purpose

`CINT(x)`, converts x to an integer by rounding the fractional portion. If the fractional portion is greater than .5, x is rounded up to the next integer; if less than .5, x is rounded down to the existing integer portion.

Syntax

`CINT (x)`

Range

-32,768 to 32,767

Related instructions

`INT` — converts a constant or variable into the largest integer that is less than or equal to x.

`FIX` — returns the truncated integer part of x.

Program segment

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.

CLS

statement

Purpose

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

Syntax

```
CLS
```

CMDGAIN

parameter

(float)

Purpose

`CMDGAIN` controls the scale factor of the analog input signal.

Syntax

`CMDGAIN = xx.xx` where xx.xx can be negative

Default

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

Related instructions

`BLKTYPE` — specifies configuration as a position, velocity or torque block.

`DACMAP` — specifies the signal sent to the monitor DAC driving the analog output channel.

Programming guidelines

IMPORTANT NOTE

CMDGAIN is automatically set to 1.0 any time the parameter 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)

CMDGAIN (continued)

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)

  Example: If analog input voltage is one volt and if CMDGAIN is equal to 1.5, then the commanded motor current will be equal to 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)

  Example: If the analog input voltage is positive one volt and if `CMDGAIN` is equal to 1.0, the motor will rotate 1/40 revolution from the resolver zero position. The zero position will be 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.***

Program segment

Program line

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

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

CONST

statement

Purpose

CONST declares numeric constants to be used in place of numeric values.

Syntax

```
CONST constant name = value
```

Programming guidelines

Unlike variable names declared by the DIM statement, constants can assume only one value in a program.

Program segment

<u>Program line</u>

```
'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

Purpose

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

Syntax

`COS(x)`

Related instructions

`SIN` — returns the sine of x in radians.

`TAN` — returns the tangent of x in radians.

Programming guidelines

x must be in radians. To convert from degrees to radians, multiply by π/180 (0.017453).

Program segment

Program line

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

This program prints the value 1.842122.

COUNTER

variable

(integer)

Purpose

`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).

Syntax

`x = COUNTER`

or

`COUNTER = x` where x is the starting count value

Range

0 to 32,767

Related instructions

`INP15` — reads the state of Input 15, the counter reset.

`INP16` — reads the state of Input 16.

`INPUTS` — reads the decimal value indicating the sum of the binary number of the inputs.

Programming guidelines

`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 kilohertz.

COUNTSPERREV

variable

(integer)

Purpose

`COUNTSPERREV` specifies the resolution of the position control.

Syntax

`x = COUNTSPERREV or COUNTSPERREV = x`

Range

x can only be 4096, 8192, 16384, 32768 or 65536

Default

x = 4096

Related instructions

`INDEX.DIST` - sets the distance the motor rotates with `GO.INCR`.

`POSITION` - indicates the actual motor position.

`POS.COMMAND` - displays or redefines position.

`TARGET.POS` - sets the target position used with `GO.ABS`.

Programming guidelines

`COUNTSPERREV` specifies the scaling, hence 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 effect accuracy.***

CWINH

variable

(integer)

(read only)

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

Syntax `x = CWINH`

Value 0 or 1

Related instructions

CCWOT — sets the counter-clockwise overtravel limit.

CWOT — sets the clockwise overtravel limit.

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

CWOT

variable

(integer)

Purpose

`CWOT` sets the clockwise overtravel limit. When the position of the motor becomes clockwise (more positive) to this threshold, a clockwise overtravel interrupt occurs (if enabled).

Syntax

```
CWOT = x
```

Range

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

Related instructions

`CCWOT` — sets the counter-clockwise overtravel limit.

Programming guidelines

`CWOT` should be set 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.**

DACMAP

parameter

(integer)

Purpose

`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.

Syntax

`DACMAP = x` where x selects the desired signal to be monitored as indicated by the following table:

Default

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 Filter	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

DACMAP (continued)

Related instructions

ANALOG.OUT — sets the voltage level of the analog output channel when DACMAP = 0.

DMGAIN — multiplies the default scale factor applied to the selected monitor.

DMF0 — selects the corner frequency of a low pass filter applied to the analog output signal.

Programming guidelines

- Select the desired variable to be monitored within your program. Modify the scale factor, if necessary, using the DMGAIN parameter. (Changing DACMAP resets the DMGAIN parameter to its default value, 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 +5 V or -5 V.
- 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 continue to decrease.

Program segment

Program line

```
 'Select ANALOG.OUT as the output signal source
DACMAP = 0
'Set analog output equal to 1 volt
ANALOG.OUT = 1.0
'Monitor the velocity error.  The default factor
'is 1 V/kRPM
DACMAP = 3
```

DACMON

variable

(float)

(read only)

Purpose

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

Syntax

```
x = DACMON
```

Related instructions

`DMF0` — selects the corner frequency of a low pass filter applied to the analog output signal.

`DACMAP` — selects signal output to analog output channel.

Programming guidelines

`DACMON` provides the ability to sense the signals that are output to the analog output channel, after they have been processed by 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.

Program segment

This program segment sends the variable `ICMD`, commanded motor current, to the analog output channel. A low pass filter with a 10 hertz 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 hertz 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
```

DECEL.GEAR

variable

(integer)

Purpose

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

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

Syntax

`x = DECEL.GEAR or DECEL.GEAR = x`

where x is the controlled deceleration rate in RPM/sec

Range

x = 1 to 16,000,000 RPM/sec

Resolution

1 RPM/sec

Default

x = 16,000,000

Related instructions

`ACCEL.GEAR` -specifies acceleration rate when gearing is turned ON.

`GEARING` - turns electronic gearing on and off.

`GEARERROR` - specifies amount of position lag accumulated when electronic gearing is turned on.

`GEARLOCK` - indicates slave axis velocity is synchronized with motor.

Programming guidelines

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

DECEL.GEAR (continued)

Program segment

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
```

DECEL.RATE

variable

(integer)

Purpose

`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.***

Syntax

`DECEL.RATE = x`

where x is the desired deceleration rate in rpm/sec

Range	x = 1 to 16,000,000 rpm/sec
Resolution	1 rpm/sec
Default	x = 10,000

Related instructions

`ACCEL.RATE` — limits the maximum commanded acceleration rate.

`ACCEL.TYPE` — specifies S-Curve or constant acceleration velocity profiles.

`GO.ABS` — causes motor to move to the position specified by `TARGET.POS`.

`GO.HOME` — moves the motor shaft to the electrical home position.

`GO.INCR` — moves the motor shaft an incremental index from the current position.

`GO.VEL` — moves the motor shaft at a constant speed.

`RUN.SPEED` — sets the commanded velocity.

`UPD.MOVE` — updates the commanded motion (currently in progress) using ACCEL.RATE, DECEL.RATE, and `RUN.SPEED`.

DECEL.RATE (continued)

Programming guidelines

Set `DECEL.RATE` prior to issuing any motion command statement.

Deceleration rate can be updated using the `UPD.MOVE` statement.

Program segment

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
WHILE TIME < 1
WEND
'Command zero velocity
RUN.SPEED = 0
'Begin deceleration of motor
GO.VEL
```

DIM

statement

Purpose

`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.

Syntax

1. Typical data allocation

```
DIM variable [(subscript)][,variable[(subscript)],...]
AS type
```

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.***

Programming guidelines

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

Non-volatile (NV) variables are intended to be used to save data that may change periodically (i.e. weekly, monthly) but is 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 production runs.

DIM (continued)

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 will be processed as INTEGER (32 bit signed number). These variables, or each array element, will require 4 bytes of storage.

FLOAT, SINGLE, DOUBLE

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

STRING

STRING variables are used to 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.

Program segment

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)

Purpose

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

Syntax

`DIR = x`

Value

x = 0, rotation is *clockwise* when looking at the motor shaft end.

x = 1, rotation is *counter-clockwise* when looking at the motor shaft end.

Default

x = 0

Related instructions

`GO.VEL` — moves the motor shaft at a constant speed.

`RUN.SPEED` — sets the commanded velocity.

Programming guidelines

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.**

DMF0

parameter

(float)

Purpose

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

Syntax

`DMF0 = x` where x is the filter corner frequency in Hz

Range

0.011 to 12,222,276 (Hz)

Default

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

Related instructions

`DACMAP` — specifies the signal output to the analog output channel.

Programming guidelines

`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.

Program segment

<u>Program line</u>

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

DMF0 = DAFILTER
```

DMGAIN

parameter

(float)

Purpose

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

Syntax

DMGAIN = x where x can be negative

Default

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

Related instructions

ANALOG.OUT — sets the voltage level of the analog output channel when DACMAP = 0. DMGAIN will not scale the ANALOG.OUT signal.

DACMAP — specifies the signal output sent to the analog output channel.

DMF0 — selects the corner frequency of a low pass filter applied to the analog output signal.

Programming guidelines

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.

Program segment

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
```

ENABLE

variable

(integer)

Purpose `ENABLE` allows or prevents power flow to the motor.

Syntax `ENABLE = x`

Value x = 0 to disable the drive

x = 1 to enable the drive

Default x = 0

Related instructions `ENABLED` — displays drive enable state.

Programming guidelines 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.

ENABLED
variable

(integer)

(read only)

Purpose

`ENABLED` indicates whether controller is enabled.

Syntax

```
x = ENABLED
```

Value

0 = controller disabled

1 = controller enabled

Related instructions

`ENABLE` — variable to enable drive in program.

Programming guidelines

To enable, that is, allow power to flow to the motor, verify that the following conditions are all true:

1. Drive is not faulted.
2. Enable input (J53-4 to J53-10) connected to I/O Rtn.
3. `ENABLE` variable programmed.

ENABLED (continued)

Program segment

Program line

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

ENC.FREQ

variable

(float)

(read only)

Purpose

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

Syntax

`x = ENC.FREQ`

Range

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

Calculation

`ENC.FREQ =`

$$ENC.IN\ \left(\frac{ENCODERCOUNTS}{REV}\right)\ x\ ENCODERVELOCITY\left(\frac{REV}{MIN}\right)\ x\ \left(\frac{MIN}{60sec}\right)\ x4$$

Related instructions

`ENC.IN` — sets the line count of the master encoder.

Programming guidelines

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

Program segment

<u>Program line</u>

```
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.
```

ENC.IN

variable

(integer)

Purpose

`ENC.IN` specifies the line count of the encoder being used for electronic gearing.

Syntax

`ENC.IN = x`

where x is the line count

Range

x = 1 to 65535

Default

x = 1024

Related instructions

`ENC.FREQ` —frequency in pulses per second.

`ENC.OUT` — must be set to zero for drive to receive external pulses.

`GEARING` — turns On or Off electronic gearing.

`RATIO` — the electronic gearing ratio of motor shaft movement to encoder shaft movement using encoder line count.

Programming guidelines

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.

Program segment

Program line

```
ENC.IN = 1024
```

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

ENC.OUT

variable

(integer)

Purpose

`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).

Syntax

```
ENC.OUT = x
```

where x is the desired pulses/rev.

Range

0, 500, 512, 1000, 1024, 2000, 2048, 4096, 16384

Default

x = 0

Related instructions

`ENC.IN` — sets the line count of the master encoder.

Programming guidelines

`ENC.OUT` must be set equal to zero for controller to execute electronic gearing.

Note: ***The maximum encoder frequency is 750 kHz.***

Program segment

Program line

```
ENC.OUT = 1024
```

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

ENCPOS

variable

(integer)

Purpose

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.

Syntax

```
x = ENCPOS
```

Value

x = ± 2,147,483,647 encoder quadrature counts

Related instructions

ENC.IN — specifies the line count of the master encoder being used.

ENC.FREQ — displays encoder frequency.

Programming guidelines

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

Note: ***The maximum encoder frequency is 750 kHz.***

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.

ENCPOS (continued)

Program segment

Program line

```
'Turn the encoder 1/2 revolution.
ENC.IN = 1024
ENCPOS = 0
PRINT "ENCPOS = "ENCPOS
```

The output is ENCPOS = 2048.

END

statement

Purpose

The `END` statement is used to mark the end of a program, subroutine, `IF...THEN...ELSE` block structure, or `INTERRUPT` service routine.

Syntax

```
END [{|SUB|IF|INTERRUPT}]
```

Related instructions

`SUB` — defines a subroutine.

`IF...THEN...ELSE` — decision tree.

`INTERRUPT` — defines an interrupt service routine.

Programming guidelines

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

- **Main Program** - 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`."

ERR

variable

(integer)

(read only)

Purpose

`ERR` contains the error code for the last error that occurred.

Syntax

```
x = ERR
```

Related instructions

`ERRVAL` - contains the value that caused the last error to occur if it was variable-related.

`ERRVAR` - contains the index number of the variable that caused the last error to occur if it was variable-related.

`ON ERROR GOTO` - enables error handling routine and specifies the first line of the routine.

Programming guidelines

After an 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.***

ERRVAL

variable

(float)

(read only)

Purpose

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

Syntax

```
x = ERRVAL
```

Related instructions

`ERR` - contains the error code for the last error that occurred.

`ERRVAR` - contains the index number of the variable that caused the last error to occur if it was variable-related.

`ON ERROR GOTO` - enables error handling routine and specifies the first line of the routine.

Programming guidelines

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

ERRVAR

variable

(integer)

(read only)

Purpose

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

Syntax

```
ERRVAR = x
```

Related instructions

ERR - contains the error code for the last error that occurred.

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

ON ERROR GOTO - enables error handling routine and specifies the first line of the routine.

Programming guidelines

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 corrective action necessary.

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.***

EXIT
statement

Purpose

The `EXIT` statement is used to exit a subroutine, an interrupt, `FOR...NEXT` or `WHILE...WEND` block structure.

Syntax

```
EXIT  [{|SUB|FOR|INTERRUPT|WHILE|}]
```

Related instructions

`SUB` — defines a subroutine.

`FOR...NEXT` — looping block structure.

`INTERRUPT` — defines an interrupt subroutine.

`WHILE...WEND` — is a looping block structure.

Programming guidelines

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 exit location of a subroutine with “`EXIT SUB`.”
- **FOR...NEXT Block Structure** - Exit the block with “`EXIT FOR`.”
- **INTERRUPT Subroutine** — Mark the exit location of a subroutine with an “`EXIT INTERRUPT`.”
- **WHILE...WEND Block Structure** — Exit the block with “`EXIT WHILE`.”

FAULTCODE

variable

(integer)

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

Syntax `x = FAULTCODE`

Status tables

SC752/SC753

Status Display	FAULTCODE	Description
0	0	No fault, disabled
1	1	Software resolver overspeed
2	2	Motor overtemperature
3	3	Servocontroller overtemperature
4	4	Servocontroller IT
5	5	Motor line-neutral fault
6	6	Control undervoltage
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	Microprocessor fault
F 1	241	Following error overflow
F 2	242	Program memory fault
F 3	243	Parameter memory fault
F 4	244	Run time error
F 5	245	PacLAN error
F 6	246	Incompatible Motion Dialogue
U C		Unconfigured controller

SC754/SC755/
SC756

Status Display	FAULTCODE	Description
0	0	No fault, disabled
1	1	Software resolver overspeed
2	2	Motor overtemperature
3	3	Servocontroller overtemperature
4	4	Servocontroller IT
5	5	Bus overcurrent
6	6	Control undervoltage
7	7	Output overcurrent
8	0	No fault, enabled
9	9	Measured shunt regulator IT fault
A	10	Bus overvoltage or Hot control undervoltage
b	11	Encoder +5V low
C	12	Terminal +5V low
d	13	Power stage control undervoltage
E	14	Microprocessor fault
F 1	241	Following error overflow
F 2	242	Program memory fault
F 3	243	Parameter memory fault
F 4	244	Run time error
F 5	245	PacLAN Error
F 6	246	Incompatible Motion Dialogue
U C		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 is 0. The variable ENABLED indicates whether the controller is enabled.

Programming guidelines

Program a fault code in an 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

Purpose

`FIX` returns the truncated integer part of x.

Syntax

`FIX(x)`

Related instructions

`INT` — converts a constant or variable into the largest integer that is less than or equal to x.

`ABS` — converts the associated value to an absolute value.

Programming guidelines

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

Program segment

Program line

```
PRINT FIX (58.75)
```

This segment prints the value 58.

```
PRINT FIX (-58.75)
```

This segment prints the value -58.

FOR...NEXT

statement

(integer)

Purpose

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

Syntax

FOR loop_counter = start value TO end value [STEP increment]

.

.

.

NEXT x = [loop_counter]

Related instructions

EXIT — permits conditional exit from FOR...NEXT loop.

Programming guidelines

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.

Program segment

Program line

For example, the `FOR...NEXT` lines:

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

execute 9 times. The first 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 = 0.5`, 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 2 STEP = -0.5
PRINT x
NEXT x
```

FVEL.ERR

variable

(float)

(read only)

Purpose

`FVEL.ERR` indicates the velocity servo error signal (`VEL.ERR = (VEL.CMD - RVEL)`), in RPM, after it has been processed by the anti-resonant filter section.

Syntax

`x = FVEL.ERR`

Related instructions

`ARF0, ARF1` — indicates anti-resonant filter corner frequencies.

`DACMAP` — selects signal output to analog output channel.

`VELOCITY` — indicates the resolver velocity.

`VEL.CMD` — indicates the velocity command.

`VEL.ERR` — indicates the velocity error.

Program segment

Program line

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

FWV
variable

(integer)

(read only)

Purpose `FWV` indicates the controller firmware version number.

Syntax

```
x = FWV
```

Program segment

Program line

```
PRINT "This Controller is a SC";MODEL;"With
Firmware Version -";FWV
```

GEARERROR

variable

(integer)

Purpose

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

Syntax

```
x = GEARERROR or GEARERROR = x
```

Related instructions

GEARLOCK — indicates slave axis velocity is synchronized with motor.

GEARING — turns electronic gearing on or off.

ACCEL.GEAR — limits the maximum commanded acceleration rate.

DECEL.GEAR — limits the maximum commanded deceleration rate.

RATIO — sets the electronic gearing rate.

PULSES.IN — specifies number of encoder inputs that PULSES.OUT is referenced to.

PULSES.OUT — selects resultant motor resolver motion count.

Programming guidelines

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.

Program segment

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
'Set GEARERROR to zero
GEARERROR = 0
```

GEARING

variable

(integer)

Purpose

`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.

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 counterclockwise controller response to follow the master's encoder input signal

Default

x = 0

Related instructions

`ENC.IN` — specifies the line count of the master encoder being used.

`RATIO` — the electronic gearing ratio of encoder shaft movement to motor shaft movement using encoder line count.

`ENCPOS` — displays the encoder position.

`PULSES.IN` — specifies the input pulse ratio (used to select exact gearing ratio).

`PULSES.OUT` — specifies the output pulse ratio (used to select exact gearing ratio).

Note: ***The effective ratio is determined by the most recently selected variables,*** **RATIO, PULSES.IN** ***or*** **PULSES.OUT.**

Programming guidelines

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`.

The variable 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.

Note: ***Other motion commands could result in motion in the disabled gearing direction.***

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

Program segment

Program line

```
ENC.IN = 1024
RATIO = 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 RPM.

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.

GEARLOCK

variable

integer

(read only)

Purpose

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

Syntax

```
x  =  GEARLOCK
```

where:

x = 0 indicates no slave velocity synchronization

x = 1 indicates that synchronization has been achieved.

Related instructions

`GEARERROR` — specifies amount of position lag accumulated when electronic gearing is turned on.

`GEARING` — turns electronic gearing on or off.

`ACCEL.GEAR` — limits the maximum commanded acceleration rate.

`DECEL.GEAR` — limits the maximum commanded deceleration rate.

`RATIO` — sets the electronic gearing rate.

`PULSES.IN` — specifies number of encoder inputs that `PULSES.OUT` is referenced to.

`PULSES.OUT` — selects resultant motor resolver motion count.

Programming guidelines

After turning on electronic gearing, slave axis acceleration is 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.

Program segment

<u>Program line</u>

```
'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
```

GO.ABS

statement

Purpose

`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.TYPE`, `ACCEL.RATE`, and `DECEL.RATE`. 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`.

Syntax

`GO.ABS`

Related instructions

`ACCEL.TYPE` — sets either S-Curve or trapezoidal velocity profiles.

`ACCEL.RATE` — limits the maximum commanded acceleration rate.

`DECEL.RATE` — limits the maximum commanded deceleration rate.

`POS.COMMAND` — displays or redefines position.

`RUN.SPEED` — sets the commanded velocity.

`ABORT.MOTION` — stops motion using available motor torque limited by current limit (`ILMT.PLUS`, `ILMT.MINUS`) parameters.

`TARGET.POS` — sets the target position used with `GO.ABS`.

`UPD.MOVE` — updates the commanded motion (currently in progress) using specified `ACCEL.RATE`, `DECEL.RATE`, and `RUN.SPEED`.

Programming guidelines

- 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 a `ABORT.MOTION` statement.

Program segment

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
```

GO.HOME

statement

Purpose

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 as setting TARGET.POS to zero and executing a GO.ABS function.

Syntax

```
GO.HOME
```

Related instructions

ACCEL.TYPE — sets either S-Curve or constant acceleration velocity profiles.

ACCEL.RATE — limits the maximum commanded acceleration rate.

DECEL.RATE — limits the maximum commanded deceleration rate.

RUN.SPEED — sets the commanded velocity.

ABORT.MOTION — stops motion using available motor torque limited by current limit (ILMT.PLUS, ILMT.MINUS) parameters.

UPD.MOVE — updates the commanded motion (currently in progress) using specified ACCEL.RATE, DECEL.RATE, and RUN.SPEED.

Programming guidelines

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

Program segment

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
```

GO.INCR

statement

Purpose

`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`.

Syntax

```
GO.INCR
```

Related instructions

`ACCEL.TYPE` — sets either S-Curve or constant acceleration velocity profiles.

`ACCEL.RATE` — limits the maximum commanded acceleration rate.

`DECEL.RATE` — limits the maximum commanded deceleration rate.

`INDEX.DIST` — sets the distance the motor rotates

`RUN.SPEED` — sets the commanded velocity.

`ABORT.MOTION` — stops motion using available motor torque limited by current limit (`ILMT.PLUS, ILMT.MINUS`) parameters.

`UPD.MOVE` — updates the commanded motion (currently in progress) using specified `ACCEL.RATE, DECEL.RATE`, and `RUN.SPEED`.

Programming guidelines

- 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 a `ABORT.MOTION` statement.

Program segment

Program line

```
'This program segment performs an incremental move
'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
'20 revolutions in the counter-clockwise direction
INDEX.DIST = -4096*20
'Issue move command
GO.INCR
WHILE MOVING = 1 : WEND
PRINT "MOVE COMPLETE"
STOP
END
```

GO.VEL

statement

Purpose

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.

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

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

Syntax

```
GO.VEL
```

Related instructions

ACCEL.TYPE — sets either S-Curve or constant acceleration velocity profiles.

ACCEL.RATE — limits the maximum commanded acceleration rate.

DECEL.RATE — limits the maximum commanded deceleration rate.

DIR — specifies the direction the motor is to turn.

RUN.SPEED — sets the commanded velocity.

ABORT.MOTION — stops motion using available motor torque limited by current limit (ILMT.PLUS, ILMT.MINUS) parameters.

UPD.MOVE — updates the commanded motion (currently in progress) using specified ACCEL.RATE, DECEL.RATE and RUN.SPEED.

Programming guidelines

- 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 a `ABORT.MOTION` statement.
- `GO.VEL` always uses trapezoidal profiles, regardless of `ACCEL.TYPE`

Program segment

Program line

```
'This program segment drives the motor clockwise for
'1 second, stops for 1 second, drives
'counter-clockwise for 1 second, then 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
```

GO.VEL (continued)

Program line

```
'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
```

GOSUB...RETURN

statement

Purpose

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

Syntax

GOSUB subroutine label

Related instructions

CALL, SUB — alternative and preferred method of defining and executing a subroutine.

Programming guidelines

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

Program segment

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
```

GOTO

statement

Purpose

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

Syntax

`GOTO` Label

Programming guidelines

Note: ***GOTO is not a recommend 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.

Program segment

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

HEX$( )

string function

Purpose

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

Syntax

```
HEX$(x)
```

Programming guidelines

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

x is rounded to an integer before `HEX$(x)` is evaluated.

Program segment

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)

Purpose

`ICMD` indicates the commanded motor torque current in amperes.

Syntax

```
x = ICMD
```

Related instructions

`DACMAP` — selects signal output to analog output channel.

`IFB` — indicates measured motor current.

`ILMT.MINUS` — sets the maximum allowable current in the counter-clockwise direction.

`ILMT.PLUS` — sets the maximum allowable current in the clockwise direction.

`KTEFF` — sets the system torque constant. `ICMD * KTEFF` is the motor shaft torque in lb-in.

Program segment

Program line

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

IFB

variable

(float)

(read only)

Purpose

IFB indicates the measured motor current amplitude in amperes.

Syntax

```
x = IFB
```

Related instructions

DACMAP — selects signal output to analog output channel.

ICMD — indicates the commanded motor torque current.

ILMT.MINUS — sets the maximum allowable current in the counter-clockwise direction.

ILMT.PLUS — sets the maximum allowable current in the clockwise direction.

Program segment

Program line

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

IF...THEN...ELSE

statement

Purpose

`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.

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
```

Programming guidelines

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). Otherwis,e execution continues on the next program line.

Block structure 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` clauses are true, the statements following the [optional] `ELSE` clause are executed. The block structure `IF...THEN...ELSE` requires termination with an `END IF` statement.

Program segment Program line

Single line

```
'SINGLE LINE IF...THEN...ELSE
DIM a AS 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 <= 4096 THEN
    PRINT "0 < POSITION <= 4096 "
 ELSE
    PRINT "POSITION > 4096 "
 END IF
```

Program line

Block type

```
‘BLOCK IF...THEN...ELSE
IF POSITION > 0 THEN
   IF POSITION <= 4096 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 THEN
  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)

Purpose `ILC` sets the current loop proportional gain.

Syntax `ILC = x`

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

Related instructions

`ARF0` — stage 0 anti-resonant single low pass filter corner frequency.

`ARF1` — stage 1 anti-resonant single low pass filter corner frequency.

`KVI` — sets the integral gain of the velocity servo loop.

`KVP` — sets the proportional gain of the velocity servo loop.

`KPP` — sets the proportional gain of the position loop.

`KVFF` — sets the amount of velocity feedforward signal applied to the position loop.

Programming guidelines

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 * Ll-l(henries)))
```

where:

Ipeak is the drive's peak current rating:

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

$L_{l\text{-}l}$ is the motor's line to line inductance in henries.

Example

For an SC752 (I_{peak} = 7.5) and an R32G motor ($L_{l\text{-}l}$ = 23 mh),

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.

IMPORTANT NOTE

A value is 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.**

Program segment

<u>Program line</u>

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

ILMT.MINUS

parameter

(integer)

Purpose

`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}).

Syntax

`ILMT.MINUS = x`

where x is the desired percent

Range

0 to 100 percent

Default

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

Related instructions

`ILMT.PLUS` — sets the maximum allowable current in the clockwise direction.

`ABORT.MOTION` — stops motion using available motor torque limited by current limit (`ILMT.PLUS`, `ILMT.MINUS`) parameters.

Programming guidelines

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

ILMT.PLUS

parameter

(integer)

Purpose

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

Syntax

`ILMT.PLUS = x`

where x is the desired percent

Range

0 to 100 percent

Default

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

Related instructions

`ILMT.MINUS` — sets the maximum allowable current in the counter-clockwise direction.

`ABORT.MOTION` — stops motion using available motor torque limited by current limit (`ILMT.PLUS`, `ILMT.MINUS`) parameters.

Programming guidelines

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

INDEX.DIST

variable

(integer)

Purpose	`INDEX.DIST` sets the distance the motor rotates during each incremental index.
Syntax	`INDEX.DIST = + x` where positive values move clockwise and negative values move counter-clockwise.
Range	x = -134,217,728 to 134,217,727 resolver steps **Note:** ***(1 step = 1/4096 revolution)***
Default	x = 4096
Related instructions	`GO.INCR` —moves the motor shaft an incremental index from the current position.
Programming guidelines	Specify `INDEX.DIST` prior to issung a `GO.INCR` command.

INKEY$

string variable

(read only)

Purpose

`INKEY$` returns one character read from the serial input port's buffer.

Syntax

```
x = INKEY$
```

Related instructions

`CHR$` — converts an ASCII code to its equivalent character.

Programming guidelines

`INKEY$` returns a string character.

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 Communications."***

Program segment

Program line

```
DIM A$ AS STRING
A$ = “”          ‘Preset A$ to null string
WHILE A$ = “”    ‘Check for null string
 A$ = INKEY$‘Check for new character
WEND
PRINT A$
```

IN.POS.LIMIT

variable

(integer)

Purpose	The `IN.POS.LIMIT` specifies the tolerance of commanded position minus actual position (`POS.ERROR`) within which region the position limit flag is set.
Syntax	`IN.POS.LIMIT = x`
Resolution	1/4096 of a motor revolution
Default	x = 5
Related instructions	`IN.POSITION` — indicates whether the motor is “in position.”

IN.POSITION

variable

(integer)

(read only)

Purpose

`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).

Syntax

```
x = IN.POSITION
```

Value

x = 0 or 1

Related instructions

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

Programming guidelines

`IN.POSITION` is always true (1) if the motor is disabled.

Program segment

Program line

```
'Set electrical home to zero
POS.COMMAND = 0
'Set to move 10 revolutions
TARGET.POS = 4096 x10
'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.

INPn

variable

(integer)

(read only)

Purpose

`INPn` reads the state of an individual discrete input.

Syntax

`x = INPn` where n = 1 to 16

Value

x = 0 — Input low (On)

x = 1 — Input high (Off)

Related instructions

`INPUTS` — reads state of discrete inputs.

`OUTPUTS` — sets the state of discrete outputs.

`OUTn` — sets state of individual output.

Programming guidelines

This is a read only variable and cannot be set by the software.

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

Program segment

Program line

```
'Check both inputs before beginning operation
WHILE (INP1 = 0) OR (INP2 = 0) : WEND
PRINT "Machine set up"
END
```

INPUT

statement

Purpose

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

Syntax

`INPUT [" prompt "] [,|;] variable`

Value

As an option, the "prompt" message is transmitted when the `INPUT` statement is encountered. If the prompt string is followed by a semicolon, a question mark is printed at the end of the prompt string. If a comma follows the prompt string, no question mark is printed.

Related instructions

`INKEY$` — reads a single string character from the serial input port's buffer.

Programming guidelines

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.

Program segment

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
```

INPUTS

variable

(integer)

(read only)

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

Syntax `x = INPUTS`

Value where x is 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)

Programming guidelines 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

Purpose

`INSTR( )` provides the location of a substring within a string.

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

Programming guidelines

n must be within the range 1 to 255.

`INSTR` returns 0 if:

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

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

Program segment

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

Purpose

`INT(x)` (Convert to Largest Integer) truncates an expression to a whole number.

Syntax

```
INT (x)
```

Related instructions

`CINT` — converts a constant or variable to an integer by rounding the fractional portion.

`FIX` — returns the truncated integer part of x.

Program segment

Program line

```
PRINT  INT ( 99.89 )
```

Prints the value 99.

```
PRINT  INT ( -12.11 )
```

Prints the value -13.

INTERRUPT

statement

Purpose

The `INTERRUPT` statement marks the beginning and the end of an interrupt service routine. The Interrupt feature 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.

Syntax

```
INTERRUPT {Source Label}
           .
           .
           .
   User Program Statements
           .
           .
           .
[INTR.{Source Label} = 1]
END INTERRUPT
```

Related instructions

`INTR.{Source Label}` - interrupt enable flag.

`RESTART` - performs a program restart.

Source 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

Source label (cont'd)

Interrupt Source	Source Label
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	I13HI
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

Programming guidelines

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.***

Program segment

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
```

INTR.{Source Label}

variable

(integer)

Purpose

The `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.

Syntax

`INTR.{Source Label} = x`

Values

x = 0 disables interrupts
x = 1 enables interrupts

Default

x = 0

Related instructions

`INTERRUPT` — marks the beginning and the end of an interrupt service routine.

`RESTART` — performs a program restart.

Programming guidelines

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 routine 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.***

Program segment

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.

IPEAK

variable

(float)

(read only)

Purpose

`IPEAK` contains the peak current rating of the controller in amperes.

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

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

Syntax

```
x = IPEAK
```

Related instructions

`MODEL` — indicates the SC750 servocontroller model number.

ITF0

parameter

(float)

Purpose

`ITF0` specifies the corner frequency of the low pass filter implementing the IT controller thermal protection circuit.

Syntax

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

Range

0.017 to 373 (Hz)

Default

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

Related instructions

`IT.THRESH` —designates the threshold (trip level) at which the IT thermal protection circuit triggers an overcurrent fault.

`IT.FILT` — contains the output of the I*T filter circuit.

Programming guidelines

`ITF0`, in conjunction 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.**

IT.FILT

variable

(float)

(read only)

Purpose

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

Syntax

`x = IT.FILT` where x contains amperes of motor current.

Related instructions

`ITF0` — specifies the corner frequency of a low pass filter, applied to the absolute value of the measured motor current, used for the controller's I*T protection.

`IT.THRESH` — designates the threshold (trip level) at which the IT thermal protection circuit triggers an overcurrent fault.

`DACMAP` — specifies variables output to analog channel.

Programming guidelines

`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.

IT.FILT (continued)

Program segment

Program line

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

IT.THRESH

parameter

(float)

Purpose

`IT.THRESH` designates the threshold (trip level) at which the IT thermal protection circuit triggers an overcurrent fault.

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 %

Default

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

Related instructions

`ITF0` — specifies the corner frequency of a low pass filter, applied to the absolute value of the measured motor current, used for the controller's I*T protection.

`IT.FILT` — contains the output of the I*T filter circuit.

`IPEAK` — denotes the peak current level for a servo controller.

Programming guidelines

`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.***

KPP

parameter

(float)

Purpose KPP sets the proportional gain of the position loop.

Syntax `KPP = x`

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

Related instructions

ARF0 — stage 0 anti-resonant single order low pass filter corner frequency.

ARF1 — stage 1 anti-resonant single order low pass filter corner frequency.

ILC — sets the current loop gain compensation control.

KVI — sets the integral gain of the velocity servo loop.

KVP — sets the proportional gain of the velocity servo loop.

KVFF — sets the amount of velocity feedforward signal applied to the position loop.

Programming guidelines

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)

IMPORTANT NOTE

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.

Program segment

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)
```

KTEFF

variable

(float)

(read only)

Purpose

`KTEFF` is the torque constant of the system in lb-in/Amp. Instantaneous motor shaft torque in lb-in is `ICMD*KTEFF`.

Syntax

`x = KTEFF` where x is the torque constant in lb-in/Amp

Default

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

Related instructions

`ICMD` — indicates the commanded motor torque current in amperes.

KVFF

parameter

(float)

Purpose	`KVFF` sets the proportion of velocity feedforward signal added to the velocity command from differentiated position command.
Syntax	`KVFF = xx`
Range	0 <= xx <= 375 xx is the percentage of velocity feedforward
Default	**Note:** ***The default value of this parameter is set during the configuration set up in Motion Dialogue.***

Related instructions

`ARF0` — stage 0 anti-resonant single order low pass filter corner frequency.

`ARF1` — stage 1 anti-resonant single order low pass filter corner frequency.

`ILC` — sets the current loop gain compensation control.

`KVI` — sets the integral gain of the velocity servo loop.

`KVP` — sets the proportional gain of the velocity servo loop.

`KPP` — sets the proportional gain of the position loop.

Programming guidelines

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.

IMPORTANT NOTE

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 the user normally should not be concerned with setting it in the application program.**

Program segment

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
```

KVI

parameter

(float)

Purpose `KVI` sets the integral gain of the velocity servo loop.

Syntax `KVI = x`

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

Related instructions

`ARF0` — stage 0 anti-resonant single order low pass filter corner frequency.

`ARF1` — stage 1 anti-resonant single order low pass filter corner frequency.

`ILC` — sets the current loop gain compensation control.

`KVP` — sets the proportional gain of the velocity servo loop.

`KPP` — sets the proportional gain of the position loop.

`KVFF` — sets the amount of velocity feedforward signal applied to the position loop.

Programming guidelines

`KVI` is the velocity loop's integral gain. 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).

IMPORTANT NOTE

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 do not set it in the application program.

Program segment

<u>Program line</u>

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

KVI = LAGBREAK
```

KVP

parameter

(float)

Purpose `KVP` sets the proportional gain of the velocity servo loop.

Syntax `KVP = x`

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

Related instructions

`ARF0` — stage 0 anti-resonant filter break frequency.

`ARF1` — stage 1 anti-resonant filter break frequency.

`ILC` — sets the current loop gain compensation control.

`KVI` — sets the integral gain of the velocity.

`KPP` — sets the proportional gain of the position loop.

`KTEFF` — contains the torque constant of the system in lb-in/Amp.

`KVFF` — sets the amount of velocity feedforward signal applied to the position loop.

Programming guidelines

`KVP` is the velocity loop's proportional gain. 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).

IMPORTANT NOTE

A value is assigned to KVP **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.**

Program segment

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)"; Jtot

INPUT "Desired Velocity Loop Bandwidth (Hz)"; BW
KVP = 2 * pi * BW * Jtot / KTEFF
```

LANFLT(n)[Axis#]

variable

(float)

Purpose

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

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.

Related instructions

`LANINT(n)[Axis#]` — array of 32 integer variables accessible over PacLAN.

Programming guidelines

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) an Axis' own `LANFLT(n)` variables.

LANINT(n)[Axis#]

variable

(integer)

Purpose

`LANINT(n)` is an array of 32 integer variables globally accessible over PacLAN.

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 a integer. [Axis#] specifies the axis location of the LAN integer variable array.

Related instructions

`LANFLT(n)[Axis#]` — array of 32 floating point variables accessible over PacLAN.

Programming guidelines

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(n) variables.

LANINTERRUPT[AXIS#]

statement

Purpose

`LANINTERRUPT` invokes an interrupt to the PacLAN controller specified by [AXIS#].

Syntax

`LANINTERRUPT[AXIS#]`

where [AXIS#] is the address of destination of the interrupt

Related instructions

`INTR.PacLAN` — Source label of the LAN interrupt on a destination axis

`INTERRUPT` — Makes the beginning and the end of an interrupt service routine.

`AXIS.INTR` — specifies the source axis of a `LANINTERRUPT`

`STATUS [AXIS#]` — determines whether an external axis is connected to PacLAN

Programming guidelines

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

LCASE$( )

string function

Purpose

`LCASE$(string-expression)` converts expression to lower case characters.

Syntax

`LCASE$(x)`

Related instructions

`UCASE$` — converts expression to upper case characters.

Program segment

Program line

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

This program segment prints: u.s.a.

LEFT$( )

string function

Purpose

`LEFT$(string-expression, n)` returns a string of the n leftmost characters of the string expression.

Syntax

`LEFT$(x$, n)`

Range

n = 0 to 255

Related instructions

`MID$` — returns a substring of a string expression.

`RIGHT$` — returns a string of the rightmost characters of the string.

Programming guidelines

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

Program segment

<u>Program line</u>

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

This program segment prints: Missi

LEN( )

string function

Purpose

`LEN(string-expression)` returns the number of characters in the string (x$).

Syntax

`LEN(x$)`

Programming guidelines

Non-printing characters and blanks are included.

Program segment

<u>Program line</u>

```
DIM X$ 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

Purpose

`LOG` returns natural logarithm value (x).

Syntax

`LOG(x)`

Related instructions

`LOG10` — returns base 10 logarithm of expression.

Programming guidelines

x must be greater than zero.

Program segment

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 .6931471.

LOG10( )

function

Purpose	`LOG10` returns base 10 logarithm of expression.
Syntax	`LOG10(x)`
Related instructions	`LOG` — returns the natural logarithm of expression.
Programming guidelines	x must be greater than zero.

LOGGEDON

variable

integer

Purpose

`LOGGEDON` indicates that the controller has been selected as the enabled multidrop subsystem by the multidrop master.

Syntax

```
x = LOGGEDON
```

x = 0 logged off

x = 1 logged on

Related instructions

`AXIS.ADDR` — indicates the multidrop address set by switch S1.

Programming guidelines

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" (i.e. `LOGGEDON = 1`), data transmitted from the multidrop master are read using the `INKEY$` function or the `INPUT` statement. If the subsystem is not `LOGGEDON`, data transmitted by the multidrop master are not available to the subsystem.

LTRIM$( )

string function

Purpose

`LTRIM$()` removes the leading blank characters from a string expression.

Syntax

`LTRIM$(string_expression)`

Programming guidelines

String_expression can be any string expression.

Program segment

Program line

```
PRINT LTRIM$("            Trim test")
```

The program line above produces the following output:

```
      Trim test
```

MID$( )

string function

Purpose

`MID$` returns a substring of a string expression that begins at the specified offset location.

Syntax

```
MID$(string_expression, start_offset [,
length])
```

Related instructions

`INSTR` — provides the location of a substring within a string.

`LEFT$` — returns a string of the leftmost characters of the string.

`LEN` — returns the number of characters in the string.

`RIGHT$` — returns the rightmost characters of a string.

Programming guidelines

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*.**

Program segment

Program line

```
DIM A$ AS STRING
A$ = "ABCDEFGHI"
PRINT MID$(a$, 1, 5)
PRINT MID$(a$, 6)
```

This program prints the following: ABCDE
FGHI

MOD

arithmetic operator

Purpose

`MOD` is the modulus or ”remainder” operator. It divides one number by another and returns the remainder.

Syntax

```
x = numeric expression MOD numeric
expression
```

Program segment

Program line

```
PRINT 19 MOD 5
```

This program prints: 4

MODEL

variable

(integer)

(read only)

Purpose

`MODEL` indicates the servocontroller model number (power level).

Syntax

```
x = MODEL
```

where x = 752, 753, 754, 755 or 756

Programming guidelines

This is a read only parameter.

Program segment

Program line

```
PRINT MODEL
```

Prints tThe model number of the servocontroller being used.

MOVING

variable

(integer)

(read only)

Purpose

`MOVING` indicates when a commanded motion profile has been completed.

Syntax

```
x  =  MOVING
```

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

Related instructions

`IN.POSITION` -indicates whether the motor is "In Position."

Programming guidelines

`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 progress.**

Program segment

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
WEND
```

ON ERROR GOTO {Label}

statement

Purpose

`ON ERROR GOTO {Label}` 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.***

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)

DoneProgram:

END
```

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

ON ERROR GOTO {Label} (continued)

Related intstructions

`ERR` — contains the error code for the last error that occurred.

`ERRVAL` — contains the value that caused the last error to occur if the it was variable related.

`ERRVAR` — contains the index number of the variable that caused the last error if it was variable related.

Programming guidelines

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 an 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` statement 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.***

OUTn

variable

(integer)

Purpose	`OUTn` (Outputs 1 to 12) sets the state of the individual discrete output.
Syntax	`OUTn = x` with n = 1 to 12
Value	`OUTn = 0` for specific outputs (1 to 12) to be On (pulled low) `OUTn = 1` for specific outputs (1 to 12) to be Off (open circuit)
Default	x = 1
Related instructions	`OUTPUTS` — allows you to set a group of outputs. `POS.CHKn.OUT` — sets outputs 1 to 3 based on position.
Programming guidelines	Set the individual variable equal to 0 to turn an output On or to 1 to turn an 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.**

OUTPUTS

variable

(integer)

Purpose	`OUTPUTS` specifies the state of the discrete outputs.
Syntax	`OUTPUTS = x`
Range	0 to 4095
Default	x = 4095
Related instructions	`OUTn` —controls individual output signals.

Programming guidelines

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

PAUSE
statement

Purpose

`PAUSE` causes the program to pause the amount of time specified by the `PAUSE.TIME` variable. The motion of the motor is not affected.

Syntax

`PAUSE`

Related instructions

`PAUSE.TIME` — sets time for pause.

Programming guidelines

The `PAUSE` function is used in place of software loops (e.g. `FOR...NEXT`) for precise control of timing.

Program segment

Program line

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

PAUSE.TIME

variable

(float)

Purpose

PAUSE.TIME defines the length of time delay during a pause statement.

Syntax

PAUSE.TIME = x

Range

x = 0.001 to 999,999

Default

x = 1.00

Related instructions

PAUSE — causes the program to wait as specified by PAUSE.TIME.

Program segment

<u>Program line</u>

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

POLECOUNT

parameter

(integer)

Purpose

POLECOUNT matches the controller for the appropriate motor pole count.

Syntax

```
POLECOUNT = x
```

Default

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

Programming guidelines

POLECOUNT defines the number of motor poles. Allowable values are 4, 6, 8, or 12.

IMPORTANT NOTE

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.**

Program segment

Program line

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

POS.CHKn

variable

(integer)

Purpose

`POS.CHKn` (Position Check trigger 1, 2, or 3) specifies the position at which outputs 1, 2, and 3 are switched to the polarity designated by the `POS.CHKn.OUT` parameter. Position check functions as a programmable limit switch output.

Syntax

```
POS.CHK1 = x
POS.CHK2 = x
POS.CHK3 = x
```

Value	x is any valid arithmetic expression
Range	-134,217,728 to 134,217,727 resolver steps **Note:** ***1 revolution = 4096 resolver steps.***
Default	x = 0

Related instructions

`POS.CHKn.OUT` — defines output when `POS.CHKn` exceeded.

Programming guidelines

Program `POS.CHKn.OUT` to enable the `POS.CHKn`.

Refer to `POS.CHKn.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.***

Program segment

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
```

POS.CHKn.OUT

variable

(integer)

Purpose

`POS.CHKn.OUT` (Position Check Output Specifier) is used in conjunction with `POS.CHKn` to implement Position Check n. Position check functions as a programmable limit switch output.

`POS.CHKn.OUT` can be set to one of three values:

Value	Description
0	Position check n disabled
10	Position check n enabled
^	If (POSITION >= POS.CHKn) then OUTn = 1
^	If (POSITION < POS.CHKn) then OUTn = 0
11	Position check n enabled
^	If (POSITION >= POS.CHKn) then OUTn = 0
^	If (POSITION < POS.CHKn) then OUTn = 1

Syntax

```
POS.CHKn.OUT  =  0
POS.CHKn.OUT  =  10
POS.CHKn.OUT  =  11
```

Default

x = 0

Related instructions

`POS.CHKn` — position to trigger `POS.CHKn.OUT`.

Programming guidelines

- 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.CHKn position before programming POS.CHKn.OUT.

Program segment

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
```

POS.COMMAND

variable

(integer)

Purpose

`POS.COMMAND` (Position Command) contains the current position command.

Syntax

`POS.COMMAND = x`

Range

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

Note: ***1 revolution = 4096 resolver steps.***

Related instructions

`GO.HOME` — moves the motor to `POS.COMMAND = 0` (electrical home position).

`POSITION` — indicates the actual motor position.

`WHEN.PCMD` —specifies the motor position when the `WHEN` condition is satisfied.

Programming guidelines

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.***

Program segment

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.***

POS.ERROR

variable

(integer)

(read only)

Purpose

`POS.ERROR` (Actual Position Error) is equal to the difference between the position command and the actual position.

Note: ***This variable is set by the internal software.***

Syntax

`x = POS.ERROR`

Range

x = -134,217,728 to 134,217,727 resolver steps

Note: ***1 revolution = 4096 resolver steps.***

Related instructions

`POSITION` — indicates the actual motor position.

`POS.COMMAND` — contains the current position command.

Program segment

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
```

POS.ERROR.MOVING

variable

(integer)

Purpose

`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.

Syntax

`POS.ERROR.MOVING = x`

Range

0 to +134,217,727 resolver steps

Default

0

Note: ***1 revolution = 4096 resolver steps***

Related instructions

`POS.ERROR` — the difference between the position command and the actual position.

`POS.ERROR.STOPPED` — sets the maximum value for `POS.ERROR` when the motor is stopped before a `POS.ERROR` interrupt is generated.

Programming guidelines

Note: ***Before enabling the*** **POS.ERROR** ***Interrupt (Setting*** **INTR.POS.ERROR =** ***1), set both*** **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.***

POS.ERROR.STOPPED

variable

(integer)

Purpose	`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.
Syntax	`POS.ERROR.STOPPED = x`
Range	0 to +134,217,727 resolver steps
Default	0
	Note: ***1 revolution = 4096 resolver steps***
Related instructions	`POS.ERROR` — the difference between the position command and the actual position. `POS.ERROR.MOVING` — sets the maximum value for `POS.ERROR` when the motor is moving before a `POS.ERROR` interrupt is generated.
Programming guidelines	**Note:** ***Before enabling the* POS.ERROR *Interrupt (Setting* INTR.POS.ERROR = *1), set both* 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.***

POSITION

variable

(integer)

(read only)

Purpose

`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.

Syntax

```
x = POSITION
```

Range

`POSITION` = -134,217,728 to 134,217,727 resolver steps

Note: ***1 revolution = 4096 resolver steps.***

Related instructions

`POS.COMMAND` — displays or redefines position.

`WHEN.POS` — is set to the value of `POSITION` when the `WHEN` condition is satisfied.

Program segment

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
```

PRINT
statement

Pur pose

`PRINT` displays output on the terminal screen while the program is running.

Syntax

```
PRINT expression [[,;] expression ] [ ; ]
```

Expressions can be:

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

Programming guidelines

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.

Program segment

Program line

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

This program segment prints “The total is 25 so far this shift.”

PULSES.IN

variable

(integer)

Purpose

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

Syntax

```
PULSES.IN = x
```

Range

x = -32,768 to +32,767

Default

x = 1,000

Related instructions

`PULSES.OUT` — selects resultant motor resolver motion count.

`RATIO` — sets the electronic gearing ratio (used instead of `PULSES.IN` and `PULSES.OUT`).

`GEARING` — enables gearing function.

Programming guidelines

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

Program segment

Program line

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

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

PULSES.OUT

variable

(integer)

Purpose

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

Syntax

`PULSES.OUT = x`

Range

x = -2048 to +2047

Default

x = 1000

Related instructions

`RATIO` — sets the electronic gearing ratio (used instead of `PULSES.IN` and `PULSES.OUT`).

`PULSES.IN` — specifies the number of encoder inputs to which `PULSES.OUT` is referenced.

Programming guidelines

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

Program segment

<u>Program line</u>

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

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

PWM12
variable

(float)

Purpose

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

Syntax

```
PWM12  =  x
```

Range

x = 0 to 100%

Related instructions

OUT12 — sets the state of the individual discrete output.

OUTPUTS — specifies the state of the discrete outputs.

Programming guidelines

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 kilohertz.

Program segment

Program line

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

RATIO

variable

(float)

Purpose	`RATIO` sets the electronic gearing ratio (Rev/Rev) between the encoder shaft and the motor shaft.
Syntax	`RATIO = + x`
Range	x = -2,000 to +2,000
Resolution	0.000001
Default	x = 1.00

Related instructions

`GEARING` — turns electronic gearing On or Off.

`ENC.IN` — sets the line count of the master encoder.

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

`PULSES.OUT` — specifies the number of resolver counts the motor moves for a specified number of encoder counts.

Programming guidelines

Follow these guidelines to program the `RATIO` variable:

- For an 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 of the parameters to be specified will determine the gearing ratio.

RATIO (continued)

Program segment

<u>Program line</u>

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

REG.DIST

variable

(integer)

Purpose

`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.***

Syntax

`REG.DIST = x`

Value

x = -134,217,728 to 134,217,727 resolver steps

Note: ***1 resolver step = 1/4096 revolution.***

Default

x = 4096

Related instructions

`REG.ENCPOS` — encoder position when registration input triggers.

`REG.FLAG` — flag to indicate which registration input triggered.

`REG.FUNC` — specifier to perform `REG.DIST` index move when registration input triggers.

`REG.POS` — position when registration input triggers.

`REG.RESPOS` — resolver position when registration input triggers.

`REG.MODE` — specifier to determine which input(s) are used to latch position registers.

Programming guidelines

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.

Program segment

<u>Program line</u>

```
'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 has been triggered and the motor has moved into position.

REG.ENCPOS

variable

(integer)

(read only)

Purpose

`REG.ENCPOS` (Registration Encoder Position) records the encoder position when an encoder registration input triggers.

Syntax

```
x = REG.ENCPOS
```

Range

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

Related instructions

`REG.DIST` — distance moved upon Registration input.

`REG.FLAG` — flag to indicate which Registration input triggered.

`REG.FUNC` — specifier to perform `REG.DIST` index move when Registration input triggers.

`REG.POS` — position when registration input triggers.

`REG.RESPOS` — resolver position when registration input triggers.

`REG.MODE` — specifier to determine which input(s) are used to latch position registers.

Programming guidelines

Attach registration inputs to appropriate location. Refer to `REG.MODE` for more information.

Refer to `REG.FUNC` for more information.

REG.FLAG

variable

(integer)

(read only)

Purpose

`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.

Syntax

`x = REG.FLAG, REG.FLAG = 0` (Resets Flag)

Value

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

Related instructions

`REG.DIST` — distance moved upon Registration input.

`REG.ENCPOS` — encoder position when Registration input triggers.

`REG.FUNC` — specifier to perform `REG.DIST` index move when Registration input triggers.

`REG.POS` — position when registration input triggers.

`REG.RESPOS` — resolver position when registration triggers.

`REG.MODE` — specifier to determine which input(s) are used to latch position registers.

REG.FLAG (continued)

Programming guidelines

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.***

REG.FUNC

variable

(integer)

Purpose

`REG.FUNC` (Registration Functionality) specifies whether `REG.DIST` is the distance that is 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.***

Syntax

`REG.FUNC = x`

x = 1 allow `REG.DIST` move upon Registration trigger.

x = 0 disallow `REG.DIST` move upon Registration trigger.

Default

x = 0

Related instruction

`REG.DIST` — distance moved upon registration input.

`REG.ENCPOS` — encoder position when Registration input triggers.

`REG.FLAG` — flag to indicate which Registration input triggered.

`REG.POS` — position when registration input triggers.

`REG.RESPOS` — resolver position when registration input triggers.

`REG.MODE` — specifier to determine which input(s) are used to latch position registers.

Programming guidelines

- 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.***

REG.MODE

variable

(integer)

Purpose

`REG.MODE` specifies the discrete input signal used to trigger and latch the encoder and resolver positions during registration.

Syntax

`REG.MODE = x`

Value

The table below indicates the inputs to be used 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

Default

x = 0

Related instructions

`REG.DIST` — distance moved upon registration input.

`REG.ENCPOS` — encoder position when registration input triggers.

`REG.FUNC` — specifier to perform `REG.DIST` index move when registration input triggers.

`REG.FLAG` — flag to indicate which registration input triggered.

`REG.POS` — position when registration input triggers.

`REG.RESPOS` — resolver position when registration input triggers.

Programming guidelines

Receipt of a resolver registration trigger source latches `REG.POS` and `REG.RESPOS` and permits an optional incremental move specified by `REG.DIST` and enabled by `REG.FUNC`.

Receipt of an encoder registration trigger source latches `REG.ENCPOS`.

Attach registration input(s) to appropriate location. Refer to section 2.5.2 of the "SC750 Installation and Hardware Reference Manual" for wiring information.

Note: ***The Z Channel input requires a differential input signal. If using a single-ended device, INP7 or INP8 may be used.***

Note: ***Registration is triggered on the rising edge of the input signal.***

REG.POS

variable

(integer)

(read only)

Purpose

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

Syntax

```
x = REG.POS
```

Range

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

Note: *1 resolver step = 1/4096 revolution.*

Related instructions

REG.DIST — distance moved upon registration input.

REG.ENCPOS — encoder position when registration input triggers.

REG.FLAG — flag to indicate which registration input triggered.

REG.FUNC — specifier to perform REG.DIST index move when Registration input triggers.

REG.RESPOS — resolver position when registration input triggers.

REG.MODE — specifier to determine which input(s) are used to latch position registers.

Programming guidelines

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

REG.RESPOS

variable

(integer)

(read only)

Purpose

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

Syntax

`x = REG.RESPOS`

Range

0 to 4095 resolver steps

Note: ***1 revolution = 4096 resolver steps.***

Related instructions

`REG.DIST` — distance moved upon registration input.

`REG.ENCPOS` — encoder position when registration input triggers.

`REG.FLAG` — flag to indicate that registration input triggered.

`REG.FUNC` — specifier to perform `REG.DIST` index move when registration input triggers.

`REG.POS` — position when registration input triggers.

Programming guidelines

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

REM or ‘

statement

Purpose

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.

Syntax

```
REM [text of comment]
```

or

```
‘  [text of comment]
```

Program segment

Program line

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

RESPOS

variable

(integer)

(read only)

Purpose

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

Syntax

```
x = RESPOS
```

Range

0 to 4095 resolver steps

Note: ***1 revolution = 4096 resolver steps***.

Programming guidelines

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

Program segment

Program line

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

RESTART

statement

Purpose

`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.

Syntax

```
RESTART
```

Related instructions

`ERR` — contains the error code for the last error that occurred.

`ERRVAL` — contains the value that caused the last erro to occur if it was variable related.

`ERRVAR` — contains the index number of the variable that caused the error.

`INTERRUPT` — defines the start and the finish of an Interrupt Subroutine.

`ON ERROR GOTO` — enables error handling and specifies the first line of the routine.

`WHEN` — performs the `WHEN` function.

Programming guidelines

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

RIGHT$( )

string function

Purpose

`RIGHT$(string expression, n)` returns the right-most characters of the string.

Syntax

`RIGHT$(x$, n)`

Related instructions

`MID$` — return a substring of a string expression.

`LEFT$` — returns the leftmost characters of a string.

Programming guidelines

If n is equal to or greater than `LEN(x$), RIGHT$` returns x$.

If n = zero, a null string is returned.

Program segment

Program line

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

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

RUN.SPEED

variable

(float)

Purpose	`RUN.SPEED` sets the maximum speed used in making an incremental or absolute move. It is also used to set the command velocity for a `GO.VEL` command.
Syntax	`RUN.SPEED = x`
Range	x = 0 to 12,000 RPM
Resolution	0.001 RPM
Default	x = 1,000 RPM
Related instructions	`GO.ABS` — causes motor to move to the position specified by `TARGET.POS`. `GO.INCR` — moves the motor shaft an incremental index from the current position. `GO.VEL` — moves the motor shaft at a constant speed. `UPD.MOVE` — updates parameters of move in progress. `DIR` — specifies the direction of a `GO.VEL` command.
Programming guidelines	Specify `RUN.SPEED` prior to issuing motion commands.

RVEL

variable

(float)

(read only)

Purpose

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

Syntax

```
x = RVEL
```

Range

x = -13,733 RPM to +13,733 RPM

Related instructions

`RUN.SPEED` — sets the maximum speed used in making an incremental or an absolute move.

`VELOCITY` — indicates motor velocity, updated every 128 milliseconds.

SGN( )

function

Purpose

SGN returns the sign of x.

Syntax

SGN(x)

x = any numeric expression

Value

If x > 0, SGN(x) returns 1.

If x = 0, SGN(x) returns 0.

If x < 0, SGN(x) returns -1.

Related instructions

SIN — returns a trigonometric function (sine) in radians.

Program segment

<u>Program line</u>

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

This program segment prints the value 1.

SIN( )
function

Purpose `SIN` (Sine) returns the trigonometric sine of x in radians.

Syntax `SIN(x)`

Related instructions

`COS` — returns the cosine of x in radians.

`TAN` — returns the tangent of x in radians.

Program segment

Program line

```
PRINT SIN(1.5)
```

This program segment prints the value .9974951 (the sine of 1.5 radians).

SPACE$( )

string function

Purpose `SPACE$(n)` returns a string of spaces.

Syntax `SPACE$(x)`

Range x = 0 to 255

Programming guidelines x is rounded to an integer.

Program segment

Program line

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

The following is printed:

```
1
 2
  3
   4
    5
```

(One space is added for each loop execution.)

SQR( )
function

Purpose

`SQR` returns the square root of an expression.

Syntax

```
SQR(x)
```

Programming guidelines

Note: ***x must be >= 0***.

Program segment

Program line

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

This program segment prints the value 3.162278.

STATUS[Axis#]

variable

(integer)

(read-only)

Purpose

`STATUS[Axis#] is used within a program to determine if an external axis is connected to PacLAN.`

Syntax

```
x = STATUS[Axis#]
```

where x = 0 indicates the external axis is not connected to PacLAN and x = 1 indicates that it is.

Program segment

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)

Purpose

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

Syntax

`STEPDIR = x`

Range

x = 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

x = 0

Related instructions

`GEARING` — turns gearing functionality On.

`ENC.IN` — specifies encoder is being used.

`RATIO` — the electronic gearing ratio of encoder shaft movement to motor shaft movement using encoder line count.

STEPDIR (continued)

Programming guidelines

Step and direction signals are similar to those output 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-.

STOP

statement

Purpose

`STOP` stops the execution of the program.

Syntax

```
STOP
```

Related instructions

`ABORT.MOTION` — stops motion using available motor torque limited by current limit (`ILMT.PLUS`, `ILMT.MINUS`) parameters.

STR$( )

string function

Purpose

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

Syntax

```
STR$(x)
```

Related instructions

VAL — returns the numerical value of the string

Program segment

Program line

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

This program will print out: 123456

STRING$( )

string function

Purpose

`STRING$` returns a string containing the specified number of occurrences of a character.

Syntax

`STRING$ (num_char, ascii_character)`

or

`STRING$ (num_char, string_expression)`

Programming guidelines

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,* STRING$ uses the first character of the string.

Program segment

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

Purpose

`SUB` defines the beginning and ending of a subroutine.

Syntax

```
SUB subroutine name
.
.
[EXIT SUB]
.
.
END SUB
```

Related instructions

`CALL` — transfers program execution to a subroutine.

Programming guidelines

Subroutines are located after the `END` statement in the main program.

Program segment

Program line

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

SWAP

statement

Purpose

SWAP exchanges the values of two variables.

Syntax

```
SWAP variable1, variable2
```

Programming guidelines

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.

Program segment

Program line

```
DIM A$, B$, C$ AS STRING
A$ = “ONE” :B$ = “FOR”:"C$ = “ALL ”
PRINT A$ B$ C$


SWAP A$, C$
PRINT A$ B$ C$
```

The program segment prints: “ONE FOR ALL” first and then prints “ALL FOR ONE”.

TAN( )
function

Purpose `TAN` (Tangent) returns the trigonometric tangent of x in radians.

Syntax `TAN(x)`

Related instructions

`COS` — returns the cosine of x in radians.

`SIN` — returns the sine of x in radians.

TARGET.POS

variable

(integer)

Purpose `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.

Syntax `TARGET.POS = x`

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

Note: ***1 resolver step = 1/4096 revolution.***

Default x = 0

Related instructions

`POS.COMMAND` — displays or redefines position.

`GO.ABS` — moves motor shaft to position specified by `TARGET.POS`.

`GO.HOME` — moves motor shaft to electrical home.

Programming guidelines

Note: ***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* positi**on.

TIME

variable

(float)

Purpose	`TIME` contains the current value in 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 `TIME=2` is executed, 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.
Syntax	`TIME = x`
Range	0 to 1,832,519.38 seconds
Resolution	0.000427 sec or 1 part in 2^{23} whichever is larger
Programming guidelines	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.

TIME (continued)

Program segment

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)

Purpose

`TMENABLEn` enables or disables the programmable timers.

Syntax

Disabling or enabling the timers is performed by setting the parameter

```
TMENABLEn = 0|1,
```

where

```
0 = Disable Timer,
```

and

```
1 = Enable Timer.
```

Default

Timers Disabled

Related instructions

`TMRSET` — specifies the operation of the programmable timers

TMOUTn

variable

(integer)

(read-only)

Purpose `TMOUTn` indicates the state of an output controlled by a programmable timer.

Related instructions `TMRSET` — specifies the operation of the programmable timers.

`OUTn` — controls individual output signals

TMRSET

statement

Purpose

`TMRSET` specifies the operation of the programmable timers

Syntax

```
TMRSET(timer#n,LEVEL|PULSE,delay_time,OUT
n=0|1,INPn=0|1,  POUTn=0|1)
```

where:

timer# -Specifies the timer number. (n is from 1 to 6)

LEVEL|PULSE -Indicates whether the output is to change its output state continuously (LEVEL), or for a fixed duration of time (PULSE).

delay_time -Designates the delay time, in seconds, for LEVEL timers or the pulse duration for the PULSE timer function.

OUTn=0|1 -Specifies the discrete output to be activated. (n is from 1 to 12)

INPn=0|1 -Denotes the timer trigger input source. (n is from 1 to 16)

POUTn=0|1 -Determines whether a position check output function is to be logically anded with the timer trigger input source.

Note: ***The*** **POUTn** ***field must be left blank if not used. The state of a timer's output is available in the read-only variable*** **TMOUTn*****, where n specifies the timer number.***

Related instructions

`TMENABLEn` — enables or disables programmable timers.

Programing guidelines

Execute `TMRSET` before enabling or disabling the timer using `TMENABLEn`.

UCASE$( )

string function

Purpose

`UCASE$(string-expression)` converts value to upper case.

Syntax

`UCASE$(x)`

Related instructions

`LCASE$` — converts value to lower case.

Program segment

Program line

```
DIM X AS STRING
x = “Washington, D.C.”
x = UCASE$(x)
PRINT x
```

This program segment prints: WASHINGTON, D.C.

UPD.MOVE

statement

Purpose

`UPD.MOVE` (Update Move) updates a move in process with new variables. This allows you to change motion "on the fly" without having to stop motion and restart the motion function again with new variables.

Syntax

`UPD.MOVE`

Related instructions

`ACCEL.TYPE` — sets either S-Curve or trapezoidal velocity profiles.

`ACCEL.RATE` — limits the maximum commanded acceleration rate.

`DECEL.RATE` — limits the maximum commanded deceleration rate.

`RUN.SPEED` — sets the commanded velocity.

Programming guidelines

Update desired `ACCEL.RATE, DECEL.RATE, RUN.SPEED`, and `DIR.`

Issue motion command before `UPD.MOVE`.

`UPD.MOVE` does not work if `ACCEL.TYPE = 1`.

Program segment

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.COMMAND = 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
```

VAL( )

string function

Purpose

`VAL(string-expression)` returns the numerical value of the string.

Syntax

`VAL(x$)`

Related instructions

`STR$( )` — returns a string representation of the value of a numeric expression

Programming guidelines

If the first character of x$ is not numeric, the `VAL(x$)` function will return zero.

VEL.CMD

variable

(float)

(read only)

Purpose

VEL.CMD (Velocity Command) indicates the net velocity servo loop command signal in RPM.

Syntax

```
x = VEL.CMD
```

Related instructions

DACMAP — selects the signal output to analog output channel.

VELOCITY — indicates the resolver velocity.

VEL.ERR — indicates the velocity error.

Program segment

Program line

```
'This program segment sends the variable
'VEL.CMD to the analog output channel
' Set up Output signal selection
DACMAP = 2
```

VEL.ERR

variable

(float)

(read only)

Purpose

VEL.ERR (Velocity Error) indicates the velocity servo error signal in RPM. VEL.ERR is the difference between the commanded and the actual resolver velocity:

```
        VEL.CMD - RVEL
```

Syntax

```
x = VEL.ERR
```

Related instructions

DACMAP — selects signal output to analog output channel.

VELOCITY — indicates the resolver velocity.

VEL.CMD — indicates velocity command signal.

Program segment

Program line

```
This program segment sends the variable
'VEL.ERR to the analog output channel
' Set up Output signal selection
DACMAP = 3
```

VELOCITY

variable

(float)

(read only)

Purpose

`VELOCITY` indicates the actual speed at which the motor shaft is rotating averaged over a 128 msec interval. This is a read-only variable.

Syntax

```
x = VELOCITY
```

Range

x = -13,733 RPM to +13,733 RPM

Related instructions

`RUN.SPEED` — sets the maximum speed used in making an incremental or an absolute move.

Program segment

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
         OUT5 = 1
  ELSE
         OUT5 = 0
  END IF
WEND
ABORT.MOTION
END
```

WHEN
statement

Purpose

`WHEN` is used for very fast output response to certain input conditions. You specify the condition and action. 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 millisecond and performs the action within 0.5 millisecond of condition satisfaction.

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`

The action must be:

- `OUTn = 1 or 0`
- `RATIO = value`
- Any of the following:

  `GO.ABS`
  `GO.HOME`
  `GO.INCR`
  `GO.VEL`
  `PAUSE`
  `ABORT.MOTION`
  `CONTINUE` (allows program execution ot continue)
  `UPD.MOVE`

Related instructions

`WHEN.ANALOG.IN` — records the digitized value of the analog input channel latched when the `WHEN` condition is satisfied.

`WHEN.DACMON` — records the value of the filtered variable output latched when the `WHEN` condition is satisfied.

`WHEN.ENCPOS` — specifies the encoder position (`ENCPOS`) latched when the `WHEN` condition is satisfied.

`WHEN.ICMD` — records the commanded motor torque current latched when the `WHEN` condition is satisfied.

`WHEN.IFB` — records the measured motor current amplitude latched when the `WHEN` condition is satisfied.

`WHEN.POS` — specifies the motor position (`POSITION`) latched when the `WHEN` condition is satisfied.

`WHEN.PCMD` — specifies the motor position command (`POS.COMMAND`) latched when the `WHEN` condition is satisfied.

`WHEN.RESPOS` — specifies the mechanical orientation of the resolver (`RESPOS`) latched when the `WHEN` condition is satisfied.

`WHEN.RVEL` — specifies the variable `RVEL`, raw motor velocity, when the `WHEN` condition is satisfied.

WHEN.TIME — specifies the variable TIME when the WHEN condition is satisfied.

WHEN.VELCMD — records the net velocity servo loop command signal latched when the WHEN condition is satisfied.

Programming guidelines

Program the WHEN statement followed by the valid condition and action separated by a comma.

Program segment

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
```

WHEN.ANALOG.IN

variable

(float)

(read only)

Purpose

`WHEN.ANALOG.IN` (WHEN Analog Input) records the digitized value of the analog input channel, in volts, at the time the `WHEN` statement is satisfied.

Syntax

`x = WHEN.ANALOG.IN`

Range

x = -12.50 to +12.50 volts

Related instructions

`WHEN` — performs the `WHEN` function.

`ANALOG.IN` — contains digitized value of analog input channel.

Program segment

Program line

```
'Latch Analog input value when input 1 goes low
WHEN INP1 = 0, CONTINUE
PRINT WHEN.ANALOG.IN
```

WHEN.DACMON

variable

(float)

(read only)

Purpose

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.

Syntax

```
x = WHEN.DACMON
```

Related instructions

WHEN — performs the WHEN functions.

DACMON — contains the value of variable output to analog output channel.

DMF0 — selects corner frequency of a low pass filter applied to analog output signal.

DACMAP — selects signal output to analog output channel.

Program segment

<u>Program line</u>

```
'Latch DACMON when input 1 goes low
WHEN INP1 = 0, CONTINUE
PRINT WHEN.DACMON
```

WHEN.ENCPOS

variable

(integer)

(read only)

Purpose

`WHEN.ENCPOS` (WHEN Encoder Position) records the encoder position at the time the WHEN statement is satisfied. This variable is checked for at 1.024 millisecond intervals.

Syntax

```
x = WHEN.ENCPOS
```

Value

x is -2,147,483,648 to 2,147,483,647 external encoder quadrature counts.

Related instructions

`WHEN` — performs the `WHEN` function.

Program segment

<u>Program line</u>

```
'Latch encoder position when input 1 goes low
WHEN INP1 = 0, CONTINUE
PRINT WHEN.ENCPOS
```

WHEN.ICMD

variable

(float)

(read only)

Purpose

WHEN.ICMD (WHEN ICMD) records the commanded motor torque current, in amperes, at the time the WHEN statement is satisfied.

Syntax

```
x = WHEN.ICMD
```

Related instructions

WHEN — performs the WHEN function.

ICMD — indicates the commanded motor torque current in amperes.

Program segment

Program line

```
'Latch current command when input 1 goes low
WHEN INP1 = 0, CONTINUE
PRINT WHEN.ICMD
```

WHEN.IFB

variable

(float)

(read only)

Purpose

`WHEN.IFB` (WHEN IFB) records the measured motor current amplitude, in amperes, at the time the `WHEN` statement is satisfied.

Syntax

```
x = WHEN.IFB
```

Related instructions

`WHEN` — performs the `WHEN` functions.

`IFB` — indicates the measured motor current amplitude in amperes.

Programming guidelines

The `WHEN` instruction permits millisecond response in latching of the variable.

Program segment

Program line

```
'Latch measured motor current when input 1 goes low
WHEN INP1 = 0, CONTINUE
PRINT WHEN.IFB
```

WHEN.PCMD

variable

(integer)

(read only)

Purpose

`WHEN.PCMD` (WHEN Position Command) specifies the motor position when the `WHEN` condition is satisfied.

Syntax

`x = WHEN.PCMD` (resolver steps)

Note: *1 resolver step = 1/4096 revolution.*

Related instructions

`WHEN` — performs the `WHEN` function.

Program segment

Program line

```
'Latch position command when input 1 goes low.
WHEN INP1 = 0, CONTINUE
PRINT WHEN.PCMD
```

WHEN.POS

variable

(integer)

(read only)

Purpose

`WHEN.POS` (WHEN Position) is set to the value of `POSITION` when the `WHEN` condition is satisfied.

Note: *This variable is set by the internal software.*

Syntax

`x = WHEN.POS`

Range

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

Note: ***1 resolver step = 1/4096 revolution.***

Related instructions

`WHEN` — performs the `WHEN` function.

Program segment

Program line

```
'Latch position when input 1 goes low
WHEN INP1 = 0, CONTINUE
PRINT WHEN.POS
```

WHEN.RESPOS

variable

(integer)

(read only)

Purpose

`WHEN.RESPOS` (WHEN Resolver Position) records the resolver's mechanical orientation relative to the motor housing at the time the `WHEN` statement is satisfied.

Syntax

`x = WHEN.RESPOS`

Range

0 to 4095 resolver steps

Note: ***1 resolver step = 1/4096 revolution***.

Related instructions

`WHEN` - performs the `WHEN` function.

Programming guidelines

`WHEN.RESPOS` varies from 0 to 4095 then back to zero as the motor rotates one revolution.

The `WHEN` instruction will permit millisecond response in latching of the variable.

Program segment

Program line

```
'Latch resolver position when input 1 goes low
WHEN INP1 = 0, CONTINUE
PRINT WHEN.RESPOS
```

WHEN.RVEL

variable

(integer)

(read only)

Purpose

WHEN.RVEL (WHEN Raw Velocity) records the raw motor velocity at the time the WHEN statement is satisfied.

Note: ***This variable is set by the internal software.***

Syntax

```
x  =  WHEN.RVEL
```

Value

x = -13,733 RPM to +13,733 RPM

Related instructions

WHEN — performs the WHEN function.

Program segment

Program line

```
'Latch raw velocity when input 1 goes low
WHEN INP1 = 0, CONTINUE
PRINT WHEN.RVEL
```

WHEN.TIME

variable

(float)

(read only)

Purpose

`WHEN.TIME` (WHEN Time) records the variable TIME at the time the `WHEN` statement is satisfied.

Note: ***This variable is set by the internal software.***

Syntax

`x = WHEN.TIME`

Value

x = 0 to 1,833,951 seconds

Related instructions

`WHEN` — performs the `WHEN` function.

`TIME` — contains the current value of a resettable free running timer that is maintained by the internal software.

Program segment

<u>Program line</u>

```
'Latch TIME when input 1 goes low
WHEN INP1 = 0, CONTINUE
PRINT WHEN.TIME
```

WHEN.VELCMD

variable

(float)

(read only)

Purpose

WHEN.VELCMD (WHEN Velocity Command) records the net velocity servo loop command signal (in rpm), at the time the WHEN statement is satisfied.

Syntax

```
x  =  WHEN.VELCMD
```

Related instructions

WHEN — performs the WHEN functions.

VEL.CMD — indicates the net velocity servo loop command signal in RPM.

Programming guidelines

The WHEN instruction permits millisecond response in latching of the variable.

Program segment

Program line

```
'Latch velocity command when input 1 goes low
WHEN INP1 = 0, CONTINUE
PRINT, WHEN.VELCMD
```

WHILE...WEND

statement

Purpose

`WHILE...WEND` tells the program to execute a series of statements as long as an expression after the WHILE statement is true.

If the expression is true, the loop statements between `WHILE` and `WEND` are executed. 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.

Syntax

```
WHILE expression

.

(loop statements)

.

WEND
```

expression is any numeric or boolean expression

Programming guidelines

`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)

Purpose

WVSHP is an integer 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 an ASCII string:

CHR\$(WVSHP)+CHR\$(WVSHP/28)+CHR\$(WVSHP/216)+CHR\$(WVSHP/22^{4})

Syntax

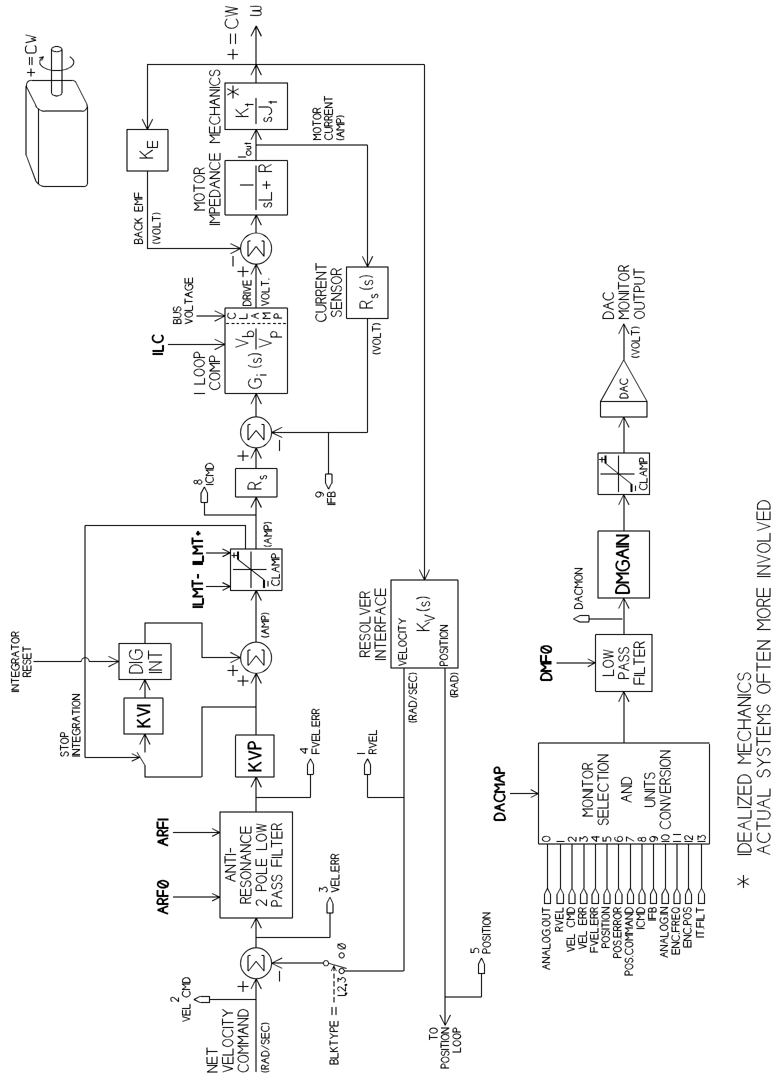
```
x = WVSHP
```

Related instructions

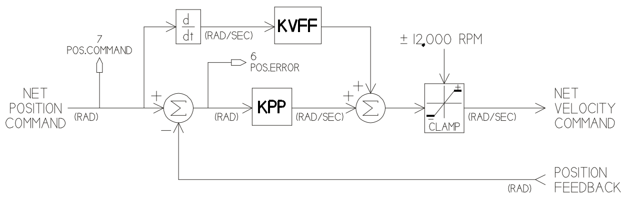
CHR$ — converts an ASCII code to its equivalent character.

This page intentionally left blank.

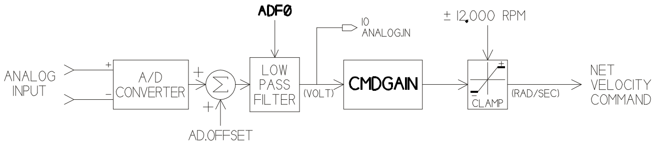
Appendix A Servo Loop



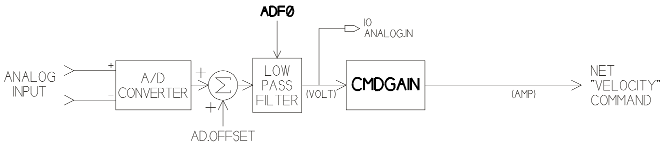
BLOCKTYPE 2: SERVO BASIC POSITION



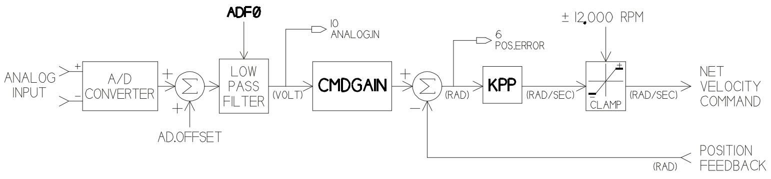
BLOCKTYPE 1: ANALOG VELOCITY



BLOCKTYPE 0: ANALOG CURRENT



BLOCKTYPE 3: ANALOG POSITION



Appendix B ASCII Codes

ASCII Code Result			ASCII Code Result		ASCII Code Result		ASCII Code Result	
0	^@	NUL	32		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	BEL	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	SYN	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	

This page intentionally left blank.

Appendix C Reserved Words

Introduction

The following list is comprised of all ServoBASIC *Plus* reserved words.

ABORT.MOTION	CFGD	DMGAIN***
ABS	CHAR	ENABLE
ACCEL.GEAR	CHR$	ENABLED
ACCEL.RATE	CINT	ENC.FREQ
ACCEL.TYPE	CLS	ENC.IN
AD.OFFSET***	CMDGAIN***	ENC.OUT
ADF0***	COMMENBL	ENCDR.POS
ANALOG.IN	COMMOFF*	ENCPOS
ANALOG.OUT	CONST	END
AND	COS	EQV
ARF0***	COUNTER	EREG**
ARF1***	COUNTSPERREV	ERR
ASC	CWINH	ERRVAL
ATAN	CWOT	ERRVAR
AUTOSTART	DACMAP***	EXIT
AXIS.ADDR	DACMON	EXP
AXIS.INTR	DBGn	EXTFAULT
BASE	DECEL.GEAR	FAULT
BEEP	DECEL.RATE	FAULTCODE
BLKTYPE***	DIM	FIX
CALL	DIR	FOR...NEXT
CCWINH	DISABLE*	FVEL.ERR
CCWOT	DMF0***	_FWDATE
		FWV
		GEARERROR

* Factory set motor parameter

** Reserved for future use

*** Non-Volatile paramete*r*

Reserved Words

GEARING
GEARLOCK
GO.ABS
GO.HOME
GO.INCR
GO.VEL
GOSUB...RETURN
GOTO
HEX$
HWV
InHI
InLO
ICMD
IFB
IF...THEN...ELSE
ILC***
ILMT.MINUS***
ILMT.PLUS***
IMP
INDEX.DIST
INKEY$
IN.POS.LIMIT
IN.POSITION
INPn
INPUT
INPUTS
INSTR
INT

INTERRUPT
INTR.xxx
IPEAK
IT.FILT
ITF0***
IT.THRESH***
KPP***
KTEFF
KVFF***
KVI***
KVP***
LANFLTn
LANINTn
LANINTERRUPT
LBOUND
LCASE$
LEFT$
LEN
LTRIM$
LOG
LOG10
LOGGEDON
MID$
MKL$
MKS$
MOD
MODEL
MOVING
NOT

OCT$
ON ERROR GOTO
OPTION
OR
OT.ERROR
OUTn
OUTPUTS
PARAMS
PAUSE
PAUSE.TIME
PERR
POLECOUNT***
POS.CHKn
POS.CHKn.OUT
POS.COMMAND
POS.ERROR
POS.ERROR.MOVING
POS.ERROR.STOPPED
POSITION
PRINT
PULSES.IN
PULSES.OUT
PWM12
RATIO
REG.DIST
REG.ENCPOS
REG.FLAG
REG.FUNC
REG.MODE

REG.POS	SWAP	WHEN.TIME
REG.RESPOS	TAN	WHEN.VELCMD
REM	TARGET.POS	WHILE...WEND
RESPOS	TIME	WRITE
RESTART	TMENABLEn	WVSHP
RIGHT$	TMOUTn	
RREG**	TMRSET	
RUN.SPEED	UBOUND	
RVEL	UCASE$	
SGN	UPD.MOVE	
SIG.THRESH*	VAL	
SIN	VEL.CMD	
SIX.THRESH*	VEL.ERR	
SPACE$	VELOCITY	
SQR	WHEN	
STEPDIR	WHEN.ANALOG.IN	
START	WHEN.DACMON	
STATUS	WHEN.ENCPOS	
STOP	WHEN.ICMD	
STR$	WHEN.IFB	
STRING$	WHEN.PCMD	
SUB	WHEN.POS	
	WHEN.RESPOS	
	WHEN.RVEL	

* Factory set motor parameter
** Reserved for future use
*** Non-volatile parameter

This page intentionally left blank.

Appendix D Status Displays

Introduction This appendix shows the status display codes and their corresponding faultcode values.

SC752/SC753

Status Display	FAULTCODE	Description
0	0	No fault, disabled
1	1	Software resolver overspeed
2	2	Motor overtemperature
3	3	Servocontroller overtemperature
4	4	Servocontroller IT
5	5	L-N fault
6	6	Control undervoltage
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	Microprocessor fault
F 1	241	Following error overflow
F 2	242	Program memory fault
F 3	243	Parameter memory fault
F 4	244	Run time error
F 5	245	PacLAN error
F 6	246	Incompatible motion dialogue
U C		Unconfigured controller

Status Display	FAULTCODE	Description
0	0	No fault, disabled
1	1	Software resolver overspeed
2	2	Motor overtemperature
3	3	Servocontroller overtemperature
4	4	Servocontroller IT
5	5	Bus overcurrent
6	6	Control undervoltage
7	7	Output overcurrent
8	0	No fault, enabled
9	9	Measured shunt regulator IT fault
A	10	Bus overvoltage or Hot control undervoltage
b	11	Encoder +5V low
C	12	Terminal +5V low
d	13	Power stage control undervoltage
E	14	Microprocessor fault
F 1	241	Following error overflow
F 2	242	Program memory fault
F 3	243	Parameter memory fault
F 4	244	Run time error
F 5	245	PacLAN error
F 6	246	Incompatible motion dialogue
U C		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.**

Appendix E Motion Commands

Introduction

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 with a check.

		MOTION VARIABLES											
MOTION TYPE	SERVOBASIC PLUS STATEMENT	ACCEL.TYPE	ACCEL.RATE	DECEL.RATE	RUN.SPEED	DIR	TARGET.POS	INDEX.DIST	GEARING	ENC.IN	RATIO	PULSES.IN	PULSES.OUT
CONSTANT VELOCITY	GO.VEL	✓	✓	✓	✓	✓							
ABSOLUTE MOVE	GO.ABS	✓	✓	✓	✓		✓						
INCREMENTAL MOVE	GO.INCR	✓	✓	✓	✓			✓					
HOMING MOVE	GO.HOME	✓	✓	✓	✓								
UPDATE PARAMETERS	UPD.MOVE		✓	✓	✓	* ✓							
ABORT MOTION	ABORT MOTION												
ELECTRONIC GEARING									✓	✓	✓	✓	✓

* Velocity moves only

Motion Commands

Appendix F 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 Multidrop 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 Section 3.1.1 of 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 master to transmit a message which selects the multidrop subsystem as the unique logged on system. A variable indicating the logon status (LOGGEDON) will be 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 the INKEY$ function or INPUT statement. If subsystems are not logged on, they cannot access data transmitted by the multidrop master. Also, an INPUT statement program execution will effectively be suspended until the subsystem has been issued a logon message and receives valid input data terminated with the carriage return character. The INKEY$ statement will return a null string, with a value of 0, if a subsystem is not logged on.

A logged on subsystem can transmit data to the multidrop master using the PRINT statement while the program is executing. If a subsystem is not currently logged on, its multidrop transmitter is disabled and program execution will 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 Section 3.1.1 of the *SC750 Installation and Hardware Reference Manual.* The subsystem address is indicated in the software variable AXIS.ADDR.

Note: ***The subsytem address S1 switch setting is polled only when power is applied to the controller. If the switch setting is modified, then power must be cycled to the controller for the new address to take effect.***

When a multidrop subsystem is logged on, its' multidrop transmitter can be enabled whenever data is to be transmitted. If a unit has not been logged on (since AC power was applied) or if a valid logon command 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) has been 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 that is currently logged on recognizes selection of another subsystem, it will disable its multidrop transmitter.

A logon variable (LOGGEDON) indicates that a multidrop subsystem has been selected to transmit to the multidrop master. This variable will be updated in all multidrop subsystems after the multidrop master has issued 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, then a subsystem that was previously logged on will suspend its multidrop transmit and receive functions as follows:

- The INKEY$ function will not indicate data characters received by the multidrop interface and will always return a null string (value of zero).
- If an INPUT statement is encountered after the address is changed, the subsystem will effectively suspend program execution until it has been re-selected as the multidrop subsystem and the INPUT statement 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), the INPUT statement 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.
- The PRINT statement 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 the PRINT statement(s) continue execution regardless of the logon state.

Due to the potential suspension or hanging of program execution, if the selected subsystem address is changed 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 the communications between the multidrop master and subsystems.

RS-232 notes

Serial data from the RS-232 RXD input is wire "or"-ed with the multidrop (RS-485) RXD input channel. Multidrop input communications are not intended to be used simultaneously with RS-232 input sources.

Executing a PRINT statement when a multidrop subsystem is logged off will result in serial data being transmitted on the RS2-32 channel only. When the subsystem is enabled, data will be transmitted on both the RS-232 and multidrop (RS-485) output channels.

Appendix G Run Time Errors

SC750 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 Routines.
5	RETURN Statement Encountered Without Matching GOSUB
6	Insufficient temporary memory for intermediary results of string variables 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.

SC750 Run Time Errors (continued)

Error No. (ERR)	**Run Time Error**
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	32	INP16
1	ACCEL.TYPE	33	INPUTS
2	CCWOT	34	Internal Use
3	Internal Use	35	OUT1
4	ILMT.MINUS	36	OUT2
5	ILMT.PLUS	37	OUT3
6	CWOT	38	OUT4
7	DACMAP	39	OUT5
8	DECEL.RATE	40	OUT6
9	DIR	41	OUT7
10	ENABLE	42	OUT8
11	ENABLED	43	OUT9
12	ENC.IN	44	OUT10
13	ENCPOS	45	OUT11
14	FAULTCODE	46	OUT12
15	GEARING	47	OUTPUTS
16	INDEX.DIST	48	POS.CHK1
17	INP1	49	POS.CHK2
18	INP2	50	POS.CHK3
19	INP3	51	POS.CHK1.OUT
20	INP4	52	POS.CHK2.OUT
21	INP5	53	POS.CHK3.OUT
22	INP6	54	POS.COMMAND
23	INP7	55	POS.ERROR
24	INP8	56	RESPOS
25	INP9	57	POSITION
26	INP10	58	PULSES.IN
27	INP11	59	PULSES.OUT
28	INP12	60	REG.DIST
29	INP13	61	REG.FLAG
30	INP14	62	REG.ENCPOS
31	INP15	63	REG.POS

Pre-Defined Integer Variable Cross-Reference (continued)

Index Number (ERRVAR)	Predefined Variable Name	Index Number (ERRVAR)	Predefined Variable Name
64	REG.RESPOS	95	INTR.I3LO
65	TARGET.POS	96	INTR.I3HI
66	WHEN.ENCPOS	97	INTR.I4LO
67	WHEN.POS	98	INTR.I4HI
68	WHEN.RESPOS	99	INTR.I5LO
69	ENC.OUT	100	INTR.I5HI
70	REG.MODE	101	INTR.I6LO
71	STEPDIR	102	INTR.I6HI
72....75	Reserved for Future Use	103	INTR.I7LO
76	IN.POS.LIMIT	104	INTR.I7HI
77	MODEL	105	INTR.I8LO
78	AXIS.ADDR	106	INTR.I8HI
79	MOVING	107	INTR.I9LO
80	WHEN.PCMD	108	INTR.I9HI
81	IN.POSITION	109	INTR.I10LO
82	IT.THRESH	110	INTR.I10HI
83	REG.FUNC	111	INTR.I11LO
84	FWV	112	INTR.I11HI
85	BLKTYPE	113	INTR.I12LO
86	Internal Use	114	INTR.I12HI
87	Internal Use	115	INTR.I13LO
88	Internal Use	116	INTR.I13HI
89	AUTOSTART	117	INTR.I14LO
90	COUNTER	118	INTR.I14HI
91	INTR.I1LO	119	INTR.I15LO
92	INTR.I1HI	120	INTR.I15HI
93	INTR.I2LO	121	INTR.I16LO
94	INTR.I2HI	122	INTR.I16HI

Pre-Defined Integer Variable Cross-Reference (continued)

Index Number (ERRVAR)	Predefined Variable Name	Index Number (ERRVAR)	Predefined Variable Name
123	INTR.CHAR	151	TMOUT4
124	INTR.CWOT	152	TMOUT5
125	INTR.CCWOT	153	TMOUT6
126	INTR.DISABLE	154	GEARLOCK
127	INTR.FAULT	155	GEARERROR
128	INTR.CWINH	156	Internal Use
129	INTR.CCWINH	157	Internal Use
130	HMV	158	Internal Use
131	COMMENBL	159	INTR.POS.ERR
132	LOGGEDON	160	Internal Use
133	EXTFAULT	161	Internal Use
134	WVSHP	162	INTR.PACLAN
134	Internal Use	163	CWINH
136	ERR	164	CCWINH
137	ERRVAR	165	COUNTSPERREV
138	STATUS	166	AXIS.INTR
139	POUT1	167	ACCEL.GEAR
140	POUT2	168	DECEL.GEAR
141	POUT3	169	Internal Use
142	TMENABLE1	170	Internal Use
143	TMENABLE2	171	POS.ERROR.MOVING
144	TMENABLE3	172	POS.ERROR.STOPPED

Index

A

ABORT.MOTION, 3-2
ABS( ), 3-3
ACCEL.GEAR, 3-4
ACCEL.RATE, 3-6
ACCEL.TYPE, 3-8
AD.OFFSET, 3-10
ADF0, 3-11
Alphabetic characters, 1-2
ANALOG.IN, 3-12
ANALOG.OUT, 3-13
ARF0, 3-14
ARF1, 3-16
Arithmetic operators, 1-2
ASC, 3-18
ASCII codes, B-1
ATAN, 3-19
AUTOSTART, 3-20
AXIS.ADDR, 3-21
AXIS.INTR, 3-23

B

BASIC functions,
  arithmetic, 1-9
  string, 1-10
BASIC instructions, 1-7
  functions, 1-8
  statements, 1-7
BEEP, 3-24
BLKTYPE, 3-25

C

CALL, 3-27
CCWINH, 3-28
CCWOT, 3-29
Characters, 1-2
  alphabetic, 1-2
  numeric, 1-2
CHR$, 3-30
CINT, 3-31
CLS, 3-32
CMDGAIN, 3-33
Communications,
  multidrop serial, F-1
CONST, 3-35
Conventions, overview, 1-1
COS( ), 3-36
COUNTER, 3-37
COUNTSPERREV, 3-38
CWINH, 3-39
CWOT, 3-40

D

DACMAP, 3-41
DACMON, 3-43
DECEL.GEAR, 3-45
DECEL.RATE, 3-47
DIM, 3-49
DIR, 3-52
DMF0, 3-53
DMGAIN, 3-54

E

ENABLE, 3-56
ENABLED, 3-57
ENC.FREQ, 3-59
ENC.IN, 3-60
ENC.OUT, 3-62
ENCPOS, 3-63
END, 3-65
ERR, 3-66
ERRVAL, 3-67
ERRVAR, 3-68
Error codes, 3-71, D-1
EXIT, 3-69

F

FAULTCODE, 3-70
FIX( ), 3-73
Float variables, 1-1, 1-6
FOR...NEXT, 3-74
FVEL.ERR, 3-76
FWV, 3-77

G

GEARERROR, 3-78
GEARING, 3-80
GEARLOCK, 3-82
GO.ABS, 3-84
GO.HOME, 3-86
GO.INCR, 3-88
GO.VEL, 3-90
GOSUB...RETURN, 3-93
GOTO, 3-94

H

HEX$( ), 3-95

I

ICMD, 3-96
IFB, 3-97
IF...THEN...ELSE, 3-98
ILC, 3-101
ILMT.MINUS, 3-103
ILMT.PLUS, 3-104
INDEX.DIST, 3-105
INKEY$, 3-106
IN.POS.LIMIT, 3-108
IN.POSITION, 3-109
INPn, 3-111
INPUT, 3-112
INPUTS, 3-114
INSTR( ), 3-115
Instruction types, 1-5
  functions, 1-6
  parameters, 1-6
  statements, 1-5
  variables, 1-6
INT( ), 3-116
Integer variables, 1-1, 1-6
INTERRUPT, 3-117
INTR.{Source Label}, 3-121
IPEAK, 3-123
ITF0, 3-124
IT.FILT, 3-125
IT.THRESH, 3-127

K

KPP, 3-129
KTEFF, 3-131
KVFF, 3-132
KVI, 3-134
KVP, 3-136

L

Labels, 1-4
LANFLT(n), 3-138
LANINT(n), 3-139
LANINTERRUPT, 3-140
LCASE$, 3-141
LEFT$( ), 3-142
LEN( ), 3-143
Liability, i
Logical operators, 1-2
LOG( ), 3-144
LOG10( ), 3-145
LOGGEDON, 3-146
LTRIM$, 3-147

M

MID$( ), 3-148
MOD, 3-149
MODEL, 3-150
Motion Commands, E-1
MOVING, 3-151
Multidrop serial communications, F-1

N

Notation conventions, 1-5
Numeric characters, 1-2

O

ON ERROR GOTO, 3-153
Operators, 1-2
  arithmetic, 1-2
  logical, 1-4
  relational, 1-3
OUTn, 3-155
OUTPUTS, 3-156

P

PAUSE, 3-157
PAUSE.TIME, 3-158
POLECOUNT, 3-159
POS.CHKn, 3-160
POS.CHKn.OUT, 3-162
POS.COMMAND, 3-164
POS.ERROR, 3-166
POS.ERROR.MOVING, 3-168
POS.ERROR.STOPPED, 3-169
POSITION, 3-170
PRINT, 3-172
PULSES.IN, 3-173
PULSES.OUT, 3-174
PWM12, 3-175

Q

Quick reference
(ServoBASIC *Plus*), 2-1

R

RATIO, 3-176
Reference, quick, 2-1
REG.DIST, 3-178
REG.ENCPOS, 3-180
REG.FLAG, 3-181
REG.FUNC, 3-183
REG.MODE, 3-184
REG.POS, 3-186
REG.RESPOS, 3-187
REM, 3-188
RESPOS, 3-189
RESTART, 3-190
Reserved Words, C-1
RIGHT$( ), 3-191
RUN.SPEED, 3-192
Run time errors, G-1
RVEL, 3-193

S

ServoBASIC *Plus*,
  instruction types, 1-5
  quick reference, 2-1
Servo loop, A-1
SGN( ), 3-194
SIN( ), 3-195
SPACE$( ), 3-196
SQR( ), 3-197
Statements, 1-5
Status displays, 3-52, D-1
STATUS[AXIS#], 3-198
STEPDIR, 3-199
STOP, 3-201
STR$, 3-202
STRING$( ), 3-203
SUB, 3-204
SWAP, 3-205

T

TAN( ), 3-206
TARGET.POS, 3-207
TIME, 3-208
TMENABLEn, 3-210
TMOUTn, 3-211
TMRSET, 3-212

U

UCASE$( ), 3-213
UPD.MOVE, 3-214
User-defined variables, 1-1

V

VAL( ), 3-216
Variables,
  pre-defined, 1-6
  user-defined, 1-1
VEL.CMD, 3-217
VEL.ERR, 3-218
VELOCITY, 3-219

W

Warranty, i
WHEN, 3-221
WHEN.ANALOG.IN, 3-224
WHEN.DACMON, 3-225
WHEN.ENCPOS, 3-226
WHEN.ICMD, 3-227
WHEN.IFB, 3-228
WHEN.PCMD, 3-229
WHEN.POS, 3-230
WHEN.RESPOS, 3-231
WHEN.RVEL, 3-232
WHEN.TIME, 3-233
WHEN.VELCMD, 3-234
WHILE...WEND, 3-235
WVSHP, 3-236