www.DanaherMotion.com

950BASIC Reference Manual

Version 4.1
MA950-LR
Rev. G

Record of Manual Revisions

ISSUE	Date	Description of Revision
G	05/30/2003	Updated corporate information

Copyright Information

© Copyright 1996 - 2003 Danaher Motion - All rights reserved.
Printed in the United States of America.

NOTICE:

Not for use or disclosure outside of Danaher Motion except under written agreement. All rights are reserved. No part of this book shall be reproduced, stored in retrieval form, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise without the written permission from the publisher. While every precaution has been taken in the preparation of the book, the publisher assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

This document is proprietary information of Danaher Motion that is furnished for customer use ONLY. No other uses are authorized without written permission of Danaher Motion. Information in this document is subject to change without notice and does not represent a commitment on the part Danaher Motion. Therefore, information contained in this manual may be updated from time-to-time due to product improvements, etc., and may not conform in every respect to former issues.

WARRANTY AND LIMITATION OF LIABILITY

Includes software provided by Danaher Motion

Danaher Motion 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. In no event, however, shall Danaher Motion 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'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's satisfaction to be defective as a result of defective materials or workmanship, in accordance with Danaher Motion's stated warranty, provided; however, that written notice of claimed defects shall have been given to Danaher Motion 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, transportation prepaid by Customer, with written specifications of the claimed defect. Evidence acceptable to Danaher Motion must be furnished that the claimed defects were not caused by misuse, abuse, or neglect by anyone other than Danaher Motion.

Danaher Motion also warrants that each of the Pacific Scientific Motion Control Software Programs ("Program(s)") will, when delivered, conform to the specifications therefore set forth in 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 may be discovered only after the Programs have been used by Customer. Customer agrees that as Danaher Motion's sole liability, and as Customer's sole remedy, Danaher Motion will correct documented failures of the Programs to conform to Danaher Motion's specifications manual. DANAHER MOTION 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; (iv) unusual stress caused by Customer's equipment; or (v) any party other than 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 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. DANAHER MOTION'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 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's date of warranty.

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

Table of Contents

1 950BASIC LANGUAGE

This chapter describes the overall structure of a 950BASIC program, and the elements of the 950BASIC language. Topics covered are:

- scope
- program structure
  - setup parameters
  - global variables, constants and aliases
  - 'main' program, subroutines, functions and interrupt handlers
- language description
  - lexical conventions
  - identifiers
  - data types
  - constants
  - statements
  - built-in functions
  - pre-defined variables
  - expressions
  - function invocation
  - $include
  - arrays and parameter lists
  - optimizations

1.1 950BASIC Program Structure

Local Variables

The notion of 'scope' is a key concept in 950BASIC programs. By 'scope', we mean those parts of the program in which a particular name is 'visible'. There are two levels of scope in 950BASIC — global and local. Variables (and constant definitions, aliases, etc.) defined inside a 'main' definition, or a subroutine, function, or interrupt handler definition, are considered to be 'local' in scope (visible only within that function).

Global Variables

All other definitions (those occurring outside functions) are considered 'global' in scope (visible inside main, and inside any subroutine, function, or interrupt handler). For example, consider the following simple 950BASIC program:

```
dim i as integer
main
dim i as integer
    for i = 1 to 10
        print "the cube of ";i;" is ";cube(i)
        call increment
    next i
end main
function cube(i as integer) as integer
    cube = i * i * i
end function
sub increment
    i = i+1
end sub
```

This program prints a table of the cubes of the integers from 1 to 10. The first (global) definition of 'i' is visible inside subroutine 'increment', but 'shadowed' by the 'i' in main and function 'cube'. The definition of 'i' inside 'main' is local to 'main', and is NOT the same variable as the 'i' inside the function 'cube', or inside the subroutine 'increment'. These same scope rules apply to constant definitions and aliases, as well.

1.2 Program Sections

The major sections of a 950BASIC program are:

- setup parameter definitions
- global variables, constants, and aliases
- 'main' program, subroutines, functions, and interrupt handlers

Although these sections may appear in any order, we recommend that you keep them in the order shown, or at least, choose a single layout style and use it consistently.

Program Template

The program below is an example of the template generated automatically by 950IDE:

```
params
‘————— Parameter Values Header —————
‘ Drive:                    SC952
‘ Motor:                    R32G
‘ Performance Setting:      Medium
‘ Inertia Ratio:            0
‘————— params start ———————————————
ARF0                        = 150.000000
ARF1                        = 750.000000
Commoff                     = 0.000000
ILmtMinus                   = 100.000000
ILmtPlus                    = 100.000000
ItThresh                    = 60.000000
Kip                         = 144.513255
Kpp                         = 15.000000
Kvi                         = 5.000000
Kvp                         = 0.059626
Polecount                   = 4
BDIOMap1                    = Fault_Reset_Inp_Lo
BDIOMap2                    = CW_Inhibit_Inp_Lo
BDIOMap3                    = CCW_Inhibit_Inp_Lo
BDIOMap4                    = 0
BDIOMap5                    = Brake_Out_Hi
BDIOMap6                    = Fault_Out_Hi
‘————— params end ———————————————
end params
‘————— Define (dim) Global Variables ——
‘————— Main Program ————————————————-
main
end main
‘————— Subroutines and Functions ————
‘————— Interrupt Routines ————————————-
```

These sections are described in greater detail in the following paragraphs.

Setup Parameter Definitions

This section of the program defines the power-on default parameters for servocontroller tuning and configuration. It is executed immediately upon power-up, before entering main, and before any interrupts are enabled. The section begins with the keyword params and ends with the keywords end or end params (this is similar to the format used to define a subroutine or function). The only statements permitted in this section are assignment statements of the form:

<pre-defined variable> = <constant expression>

This section is automatically generated by 950IDE when File|New is selected from the main menu. Ordinarily, you do not need to modify the statements in this section — they are automatically given optimal values based on the New Program dialog, and should not be changed unless further tuning is necessary.

Global Variables, Constants, And Aliases

This section contains variables, constant definitions, and global alias expressions — they apply everywhere in the program, unless specifically overridden by another declaration at local scope (inside a subroutine, function, or interrupt handler). Global definitions may be placed almost anywhere in the program text — between subroutines, before or after 'main', and so on.

Global variables, constants, and aliases do not need to be defined before use — the only requirement is that they be defined at some point in the program text. You may have multiple instances of the global variables section throughout your program. However, as a matter of good programming style, we recommend that you keep all global definitions in one place, preferably at or near the beginning of your program.

Variable Definitions

The format of a global variable definition is:

```
dim a,b, as integer, x,y,z as float
dim ia(3,4) as integer
dim s1, s2 as string*80
dim sa(5,2) as string
```

Line 1 declares a and b as integers, x,y, and z as floats. Line 2 declares a 3 x 4 array of integers. Line 3 declares s1 and s2 as strings, each of length 80 Line 4 declares sa as a 5 x 2 array of strings, each with the default length of 32 characters.

In addition, global variables are specified as 'nv' to indicate their values are retained when power is turned off. All other global variables are automatically initialized when the program begins (strings are set to empty, and floats and integers are set to 0). There are no restrictions on the ordering of volatile vs. non-volatile user-variables. For ease of program maintenance, place all non-volatile variables definitions in a single section at the beginning of the program, and add new variables to the end of that section.

Constant Definitions

The format of a constant declaration is:

<name> = <constant_expression>

as in

```
const ARRAY_SIZE = 4 * NUMBER_OF_ENTRIES
const PI_SQUARE = 3.1415926535 ^ 2
const GREETING = "Hello"
const SALUTATION = GREETING + ", world!"
const NUMBER_OF_ENTRIES = 5
```

Names for constants follow the same rules as variable names. 'Forward definitions' are allowed. Circular definitions are detected and reported at compile-time. Although it is not required, it is convenient to adopt a convention of keeping all constants in UPPER_CASE, so you can easily distinguish between constants and variables in the program.

Constant definitions are entirely 'folded' at compile-time. Feel free to write maintainable constant expressions such as:

```
const LENGTH = 3
const WIDTH = 10
const AREA = LENGTH * WIDTH
```

The value of AREA is computed at compile-time, so the program does NOT need to compute this at run-time and the program is easier to maintain if LENGTH changes at some future date.

Alias Definitions

Aliases allow you to define your own names for system resources, such as input / output pins. The intention is to make it possible for you to use names that are meaningful to you in your particular application. The format of an alias expression is:

```
alias <name> = <expression>
```

For example, the following alias defines application-specific uses of input # 1:

```
alias CONVEYOR_IS_RUNNING = (inp1=0)
alias CONVEYOR_IS_STOPPED = (inp1=1)
if CONVEYOR_IS_RUNNING then print “running” else print “stopped”
```

An alias is much more powerful than a constant. Constant expressions are computable at compile-time, while an alias has a value that is only known (in general) at the time it is used. Use aliases with care — too much aliasing can make it very difficult for you to understand the program.

1.3 Main Program, Subroutines, Functions & Interrupt Handlers

These sections share the same fundamental structure:

```
<section>
    <declarations>
    <statements>
<section end>
```

An example of each of these sections follows, with an explanation of key points.

Main Definitions

For main, a typical definition is:

```
main
dim i as integer
    i = 1
    print i
end main
```

The variable ‘i’ defined above in the ‘dim’ statement is a local variable — it is not accessible to other functions, and inside ‘main’, its definition overrides any other variable named ‘i’ that might exist at global scope.

Unlike global variables, local variables MUST be defined at the beginning of the section — they must appear before any executable statement in main. For example, the following is illegal:

```
main
dim i as integer
i = 1
dim j as integer          'this is an error!
j = i
end main
```

You may also define local constant definitions and aliases, provided that like local variables, they appear before any executable statement. Local constant definitions override global definitions of the same name. For example, given the following global definitions,

```
const N = 1
main
const N = "Hello, world!"
print N
call sub1
end main
sub sub1
      print N
end sub
The program prints:
Hello world!
1
```

Because the *N* visible inside main is the constant defined there, while the *N* visible to sub1 is the global constant *N*, whose value is 1.

The main program is the section of your program that is executed immediately after the'params section, regardless of its position in the program text. Other functions, subroutines, and interrupt handlers are executed according to the flow of control defined in the program.

main does not accept arguments, and cannot be called from any other subroutine, function, or interrupt handler.

Subroutine Definition

For a subroutine such as print_sum, a typical definition is:

```
sub print_sum(i,j as integer)
        print i+j
end sub
```

The arguments to this subroutine are specified as integer variables, and are passed by value — any assignments to these variables has no effect on the arguments supplied by the caller. Subroutines are invoked by 'call' instructions, as in call print_sum(3,4).

Function Definition

For a function such as sum_squares, a typical definition is:

```
function sum_squares(i,j as integer) as integer
        sum_squares = i^2 + j^2
end function
```

The function above returns a value of type integer. The value of the function is assigned by assigning to the name of the function, as if it were a variable. However, it is not legal to use the function name as a variable on the right-hand-side of an assignment — a function name on the right-hand-side is always an INVOCATION of that function.

There must be at least one statement in the function that assigns a value to the function. It is not possible to detect at compile-time if the statement will actually execute. Functions are invoked by name, as in print sum_squares(3,4).

This is syntactically identical to an array reference.

Interrupt Handler Definition

For an interrupt handler such as i1hi, a typical definition is:

```
interrupt i1hi
        print "interrupt occurred on input 1"
        intri1hi = TRUE
end interrupt
```

The interrupt is re-enabled by the statement intri1hi = TRUE. A similar statement must be executed once before the interrupt is serviced. It is a run-time error to attempt to enable an interrupt for which no handler is defined.

Interrupt handlers do not return values and cannot have arguments. They declare local variables, constants, and aliases. Interrupt handlers are invoked when the 950 hardware detects that the designated interrupt condition is satisfied (provided that the interrupt is enabled).

1.4 Language Definition

Lexical Conventions

950BASIC is case-insensitive. String literals are not modified, but all other text is treated as if it were entered in upper case. This means that the identifiers spin, Spin, and SPIN all refer to the same entity.

Identifiers

Identifiers are alphanumeric and must start with an alphabetic character or underscore. In addition, they may include the underscore character ('_') and dollar sign ('$'). Identifiers denote variables, functions, subroutines, and statement labels, symbolic constants, and aliases. Identifiers are a maximum of 40 characters. User-defined identifiers may not include the period ('.'). Use of a longer identifier is a compile-time error. Several pre-defined variables that have special forms:

predefvar	{alpha} {alnum}* '.' {alnum}*
alpha	[A-Za-z_]
alnum	[A-Za-z_0-9$]

Many of these pre-defined variables have alternate spellings without the '.' character, such as index.dist and IndexDist. Although both forms are accepted for compatibility, the latter form is preferred. Although 950BASIC is case-insensitive, we recommend that you adopt a consistent naming convention, such as IndexDist, and avoid having indexDist, index.dist, and Indexdist in the same program.

Data Types

The pre-defined types are INTEGER, FLOAT, and STRING. LONG is used for INTEGER. SINGLE or DOUBLE are used for FLOAT. INTEGER variables are 32-bit signed integers. FLOAT variables are IEEE single-precision floating point numbers. STRING variables are represented internally as a maximum length, a current length, and an array of ASCII characters (can contain null characters).

When a FLOAT result is assigned to an INTEGER variable, or when a FLOAT argument is used where an INTEGER is expected, the value is coerced to an integer before use. Coercion from FLOAT to INT always rounds to the nearest integer. For example:

1.2 rounds to 1
1.7 rounds to 2
-1.2 rounds to –1
-1.7 rounds to -2

Scalar INTEGER and FLOAT coercion is automatically provided for function arguments. When passing ARRAYS as arguments, the types must match exactly because coercion is prohibitively expensive at run-time.

String assignment is checked at run-time. An attempt to copy a string to a destination too small results in a run-time error. String indexing is 1-origin. For example, mid$("abc",1,1) returns the string, a.

STRING variables have a firmware-imposed maximum length of 230 characters and a default maximum length of 32 characters. They may be assigned a different maximum length by declaring them to be of type STRING*_n_ where _n_ is a positive integer between 1 and 230 (inclusive).

Declare arrays of the pre-defined types. Arrays have a maximum rank of four dimensions. The upper-bound of each dimension has no compiler-defined limit. However, because of the limited data space of the controller, there is a logical upper-bound that depends on the controller model.

Array indexing is 1-origin. The indices in each dimension range from 1 to the upper-bound of the dimension. Every reference to an array element is checked at run-time. Any attempt to reference beyond the bounds of the array causes a run-time error. New types cannot be defined.

Literal Constants

String constants begin and end with the double-quotes ('"'). They cannot extend past the end of the input line. Any printable ASCII character appears in a string constant. An attempt to generate a string literal with non-ASCII characters causes a compile-time error. No check is made to verify that non-ASCII strings are not created at run-time, so avoid doing so.

Decimal Integer Constants

Decimal integer constants are a string of decimal digits with no decimal point. A leading '-' sign is optional and is parsed as a unary minus. For example:

```
1
-1
314159
```

are all valid decimal constants.

Hexadecimal Constants

Hexadecimal constants are denoted by a leading &H or &h, and cannot have a sign or decimal point. Hexadecimal constants are composed from the set [0-9A-Fa-f]. Upper- and lower-case may be mixed. For example:

```
&h00ff
&HFF00
&H1234abcd
```

are all valid hexadecimal constants.

Octal and binary constants are not supported.

Floating-Point Constants

Floating-point constants are specified in fixed-point or mantissa-exponent notation. A floating-point constant consists of one of the following.

digit	[0-9]
optsign	'+' | '-' | /* nothing */
fixed	optsign {digit}+ '.' {digit}*optsign '.' {digit}+
exp	fixed 'e' optsign {digit}+
float	fixed | exp

For example:

```
0.1
.1
-.1
-0.1
3.14159E-6
-1.0E6
```

are all valid floating point constants.

By design, "." is not a legal floating-point constant.

1.5 Statements

Statements are separated by a new line (CR-LF) or a colon (':'). The statements of the language are:

AbortMotion AbortMotion stops motor motion and allows continued program execution. Deceleration is determined by the motor torque capability in conjunction with the current limit parameters.

Alias

```
Alias <name> = <expression>
```

Create an alias for an identifier (not just any identifier). alias is either a pre-defined variable or another alias. id must be a legal variable name.

You cannot create an alias for an array element.

Like Const definitions, Alias definitions can be made to identifiers not yet defined. Circular definitions are not allowed.

Any duplicate definition of an identifier in the same scope is illegal. However, a local definition can shadow a definition from the global scope. Using a single identifier to denote two different objects is NOT allowed (i.e., you cannot have both a label and a variable named all_done).

Like constant, variable, and function declarations, Alias declarations made in the global scope are imported into all functions (including the main function).

Example

```
Alias speed = motor.speed    'save some keystrokes
```

Beep Sends the ASCII character, &h7, to the serial port.

Call

```
CALL sub[(arg1, arg2, ...)]
```

sub is the name of a subroutine. The current program counter is saved and sub is invoked. When sub finishes (by reaching either an exit sub or end sub statement, control is returned to the statement logically following Call.

A subroutine is essentially a function with no return value. The parameter passing conventions followed by subroutines are the same as those followed by functions.

Cls

This statement transmits 40 line-feed characters (ASCII code = 10) to the serial port. Cls clears the display of a terminal.

Const

```
Const name = x
```

Declares symbolic constants to be used instead of numeric values. Forward references are allowed, but circular references are not supported.

```
CONST x = y + 2
CONST y = 17
```

unsupported

```
CONST x = y + 2
CONST y = x - 2
```

Like alias, variable, and function declarations, Const declarations made in the global scope are imported into all functions (including the main function).

Dim

```
Dim var1 [, var2 [...]] as type [NV]
```

All variables must be declared. Local variables must be declared in the function before use. Global variables are defined in the module after use in a function (as can functions).

The *NV* specifier is used on a Dim statement in the global scope, in the main function, or a Static statement in function scope.

Variables in the global scope are automatically imported into functions and subroutines. Variables in function scope (including inside the main function) are not accessible in other functions.

Arrays cannot be assigned directly (i.e., the following is not allowed):

```
DIM X(5), Y(5) AS INTEGER
X = Y
```

Instead, a loop is needed:

```
DIM X(5), Y(5), I AS INTEGER
    FOR I = 1 to 5
        X(I) = Y(I)
    NEXT I
```

Exit

```
Exit {{Sub|Function|Interrupt|For|While}]
```

Exits the closest enclosing context of the specified type. It is a compile-time error to EXIT a construct not currently in scope.

For…Next

```
For loop_counter = Start_Value To End_Value [Step increment]
...statements...
Next
```

If step increment is not specified, uses 1 as the step increment. If step increment is positive, continues to the value of End_Value. If step increment is negative, continues to the value of var = limit.

The loop index variable must be a simple identifier, not an array element or a pre-defined variable and must be a numeric variable (integer or float).

The semantics of a For loop are defined in terms of the following transformation:

```
FOR var = init TO limit STEP delta
        stlist
NEXT var
becomes:
var = init
delta_val = delta
limit_val = limit
test:
IF delta_val 0 AND var limit_val THEN
        GOTO done
ELSEIF delta_val 0 AND var limit_val THEN
        GOTO done
ENDIF
stlist
var = var + delta_val
GOTO test
done:
...
```

Substantially more efficient code is generated if* delta *is a constant (i.e., the default value of 1 is used, or specified as an expression that is evaluated at compile-time).

Function

```
Function function-name [(argument-list)] as function-type
...statements...
End Function
```

On function entry, all local variable strings are "" and all numeric locals are zero (including all elements of local arrays).

If the function takes no arguments, omit the paramlist. An empty paramlist is illegal.

The value returned from the function is specified by assigning an identifier with the name of the function.

Example

```
FUNCTION cube(x AS FLOAT) AS FLOAT
    cube = x * x * x
END FUNCTION
```

Arguments are passed by value.

Arrays can not be returned by a function. Arrays passed to a function are passed by value.

If the return value is not set, a runtime error condition is generated (caught with ON ERROR).

Array actuals must conform with formals to the extent that they have the same number of dimensions, and EXACTLY the same type. The size of each dimension is available to the function through the use of local constants that are bound on function entry.

Example

```
FUNCTION sum(x(N) AS INTEGER) AS INTEGER
    DIM i, total AS INTEGER
    sum = 0
    FOR I = 1 TO N
        total = total + x(i)
    next
    sum = total
END FUNC
```

This function exploits the fact that the variable *N* is automatically assigned a value when the function is called and the value is the extent of the array passed on invocation. *N* is a read-only variable in this context. Attempts to write to *N* cause compile-time errors.

The local variable, total is automatically initialized to 0 upon function entry.

GoAbs

GoAbs (Go Absolute) moves the motor to the position specified by TargetPos. This position is based on a zero position at electrical home.

The motor speed follows a velocity profile as specified by AccelType, AccelRate, and DecelRate . Direction of travel depends on current position and target position only (DIR has no effect).

After the program initiate GoAbs, it immediately goes to the next instruction.

Change variables during a move using UpdMove.

GoHome

GoHome moves the motor shaft to the electrical home position (Position = 0).

The motor speed follows a velocity profile as specified by AccelRate, RunSpeed, and DecelRate.

After the program initiates GoHome, it immediately goes to the next instruction.

GoHome performs the same action as setting TargetPos to zero and executing a GoAbs function.

GoIncr

GoIncr (Go Incremental) moves the motor shaft an incremental index from the current position.

Distance, as specified in IndexDist, is either positive or negative. The motor speed follows a trapezoidal velocity profile as specified by AccelType, AccelRate, RunSpeed, and DecelRate.

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

Change variables during a move using UpdMove.

GoVel

GoVel (Go Velocity) moves the motor shaft at a constant speed.

The motor accelerates and reaches maximum speed as specified by AccelRate and RunSpeed, with direction determined by DIR. Stop motion by:

- Programming AbortMotion for maximum deceleration allowed by current limits.
- Programming RunSpeed = 0 for deceleration at rate set by DecelRate.

After the program initiate GoVel, it immediately goes to the next instruction.

Change variables during a move using UpdMove.

GoTo

```
GoTo label
```

A program can only GoTo a label in the same scope. A GoTo may jump out of a For or While loop, but not INTO one.

If…Then…Else

```
IF condition1 THEN
...statement block1...
[ELSEIF condition2 THEN
...statement block2...]
[ELSE
...statement block3...]
END IF
```

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.

$Include

```
$INCLUDE inclfile
$Include include-file-name
```

Textually include inclfile at this point in the compilation.

There can be no space between $ and include. The $include directive must start at the beginning of the line.

Input

Input [prompt-string][,|;]input-variable

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

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.

Interrupt … End Interrupt

Interrupt {Interrupt-Source-Name}
..program statements...
End Interrupt

Interrupt handlers can be located anywhere in the program text (e.g., before main).

Laninterrupt[ ]

Laninterrupt '['axis']'
Laninterrupt invokes an interrupt to the PacLAN controller specified by [*AXIS#*].

This command is only available with PacLAN controllers.

On Error GoTo

On Error Goto *Error-Handler-Name*

or

On Error Goto 0

When a firmware runtime error condition occurs, *Error-Handler-Name* is called, the error handler is de-installed, and an internal flag (in-error-handler) is set. Any subsequent runtime error (including attempting to set the error handler, or return from the On Error handler) causes an immediate Stop.

On Error Goto 0 disables the current On Error handler. If an error occurs when no error handler is installed, Stop is invoked.

Pause( )

Pause(*Pause_Time*) causes the program to pause the amount of time specified by the *Pause_Time* argument. The motion of the motor is not affected.

This implementation differs from the SC750.

Print

Print *expression1* [[,;] *expression2*] [;]

Print a list of *expressions*, separated by delimiters. Any number of delimiters (including zero) can appear before or after the list of expressions. At least one delimiter must appear between each pair of expressions in the print list.

Expressions are optional.

Example

```
PRINT           ‘ print a newline
PRINT ,         ‘ advance a single tab stop
PRINT a,b       ‘ print a and b, tab between
PRINT a,b,      ‘ print a and b, tab between and at end
PRINT ,,,x,,,   ‘ tab tab tab x tab tab tab
```

Restart

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.

Restart does not clear the data area or change any program or motion variables.

Select Case

```
Select Case test-expression
Case expression-list1
...statement block1...
Case expression-list2
...statement block1...
Case expression-list3
...statement block1...
Case Else
...else block...
End Select
```

test-expression must evaluate to an INTEGER or FLOAT value.

expression-list1 is a non-empty list of case-defn, separated by commas.

There can be only one Case Else and, if present, it must appear as the last case. It is selected only if all other tests fail.

case-defn can be any of the following:

expr	
expr TO expr	(tests inclusive (closed range))
IS relop expr	(<, £, =, ³, >)
IS expr	(equiv to “IS = expr”)

Select-case statements where the case-defn expressions are composed solely of integer constants are evaluated much quicker at run-time. (Cases involving variables must be transformed to logically equivalent if-then-else statements.)

Static

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.

Stop

Stops the execution of the program.

Sub…End Sub

```
Sub [argument-list]
...body of the sub-procedure...
End Sub
```

Declare a subroutine. Invoked via Call. Optionally takes arguments. As with Function, it is illegal to provide an empty parameter list (‘()’) if the subroutine takes no parameters.

Swap

Swap x, y
Swaps the values of the variables. The variable types must be the same. Does not work on arrays or strings.

UpdMove

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

When

When *when-condition* , *when-action*

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. The program 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:

WhenEncpos	WhenRespos
WhenPosCommand	WhenTime
WhenPosition	

The software checks for the defined condition every 0.5 millisecond and performs the action within 0.5 ms of condition satisfaction.

While…Wend

While condition
...statement block...
Wend

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 not true, the statement immediately following the Wend statement is executed.

1.6 Built-in Functions

A function that takes a numeric argument (either FLOAT or INTEGER) returns the same type. Coercion between INTEGER and FLOAT is not performed unless necessary. (notation — the arguments *n* and *m* refer to INTEGER types, as in the definition of the MID$ function, whose signature is MID$(string, integer, integer).

Name	Args	Return	Semantics
ABS	numeric	numeric	absolute value
ATAN	float	float	arc tangent (radians)
CINT	numeric	int	truncate (round to nearest int)
COS	float	float	cosine
EXP	float	float	e ^ arg, arg 88.02969 (o/w overflow)
FIX	numeric	int	truncate (round toward zero)
INT	numeric	int	truncate (round towards -INFINITY)
LOG	float	float	natural log
LOG10	float	float	log base 10
SGN	numeric	integer	sign of argument: -1, 0, 1
SIN	float	float	sine (radians)
SQR	float	float	square root of arg
TAN	float	float	tangent (radians)

String function			Description
ASC	string	int	ASCII code for 1st char
CHR$	int	string	One-character string containing the character with the ASCII code of arg. If arg 255, returns CHR$(arg % 256).
HEX$	int	string	Printable hexadecimal rep of arg (without leading &H).
INKEY$		string	One-character string, read from serial port.Returns “” if no char available.
INSTR	[pos],str1,str2	int	Index of str2 in str1, or 0 if not found. Optional first arg specifies where to start search (defaults to position 1).
LCASE$	str	str	Returns lower-case copy of arg.
LEFT$	str,n	str	Returns n leftmost chars of str.
LEN	str	int	Returns length of str in bytes.
LTRIM$	str	str	Trim leading spaces.
MID$	str,n[,m]	str	Returns substring starting at position n [for up to to m bytes].
OCT$	n	str	Octal string representation of arg.
RIGHT$	str,n	str	Rightmost n chars of str.
RTRIM$	str	str	Trim trailing spaces.
SPACE$	n	str	Returns a string of n spaces.
STR$	n	str	Decimal string representation of str.
STRING$	n,str	str	Return n copies of first char of str.
STRING$	n,ch	str	Return n copies of char.
TRIM$	str	str	Trim leading AND trailing spaces.
UCASE$	str	str	Returns upper-case copy of arg.
VAL	str	numeric	Returns numeric value of str.

Pre-defined Variables and Commands

The 950BASIC language is augmented by a set of pre-defined variables, whose purpose is to set motor-specific control parameters, and by a set of pre-defined commands, whose purpose is to control the motor.

For example, AccelRate, DecelRate, and RunSpeed are used to set the acceleration rate, deceleration rate, and commanded motor speed for the next commanded move:

```
AccelRate = 1000.0
DecelRate = 1000.0
RunSpeed = 500.0
GoVel
```

The program fragment above sets up the relevant motion parameters, and commands the motor to move in velocity mode.

You cannot create variables (or function names, etc.) that shadow pre-defined ones. For a complete list of pre-defined variables and commands, refer to the detailed Language Reference section in this manual.

1.7 Expressions

Arithmetic Expressions

Arithmetic expressions (expressions involving INTEGER and FLOAT values) use the following operators.

Operators higher in the table have greater precedence than those below.

Numeric Operators

Operator	Assoc	Name
^	right	exponentiation
-	right	unary minus
*	left	multiply
/	left	divide
MOD	left	modulo
+	left	add
-	left	subtract

Logical Operators

Operator	Assoc	Explanation
=, < >, ³, £, <, >	left	the usual
NOT, BITNOT	right	not, boolean not
AND, BITAND	left	and, boolean and
OR, BITOR, XOR, BITXOR	left	or, boolean or, xor, boolean xor

Logical expressions (as, for example, in the condition of an 'if' statement) also use these operators. Strings are concatenated with the '+' operator. Logical expressions are formed from strings, using the comparison operators, NOT, AND, OR, and XOR, with the meaning of an empty string being FALSE, and a non-empty string being TRUE.

Integer values are coerced to floating point values as needed. Floating-point values are rounded when coerced to integer values.

Logical operators are NOT short-circuiting (i.e., when executing the code).

```
if a(x) or b(y) or c(z) then ...
if a(x) is true, b(y) and c(z) are still invoked.
```

BITxxx boolean operators are provided to support bitwise operations on integer values. They operate quite differently from their logical equivalents. For example:

2 and 1 has the value –1
(TRUE, since each operand is 'true'),

but

2 bitand 1 has the value 0
(since no matching bits are 1).

Similarly,

3 or 4 has the value –1
(TRUE since at least one operand is not FALSE),

while

3 bitor 4 has the value 7
(the three lsb's are set).

Remember that relational and logical operators return numeric values — 0 for FALSE and -1 for TRUE. Any value not equal to FALSE is considered to be logically equivalent to TRUE for purposes of the logical operators.

It is syntactically incorrect to code:

```
DIM a, b, c, x AS INTEGER
x = a < b < c
```

String Operators

Operator	Assoc	Name
<, >, £, ³	nonassoc	string comparisons
=, <>	nonassoc	string comparisons
	left	string concatenation

There is no implicit coercion between strings and numeric types.

String comparison is case-sensitive. Relative comparisons are made using ASCII lexical ordering. The empty string sorts before all other strings.

String comparison operators are non-associative because they evaluate to a numeric value.

Example

It makes no sense to say a$ = b$ = c$.

It is sensible to say x = a$ = b$
x is assigned the value TRUE if a$ is the same as b$, and FALSE otherwise.

1.8 Function Invocation

A function invocation is denoted as:

```
var = func(arg1, arg2, ..., argn)
```

The arguments are passed by value (i.e., modifications made to the formal parameters inside a function are not reflected in the actuals). Arrays are also passed by value to functions. Arrays cannot be returned by a function. A function of no arguments is invoked by using the function name alone. For example, if func_none takes no arguments, then func_none is correct and func_none() is invalid.

The return value of a function may not be ignored by the caller. If the return value of a function is regularly ignored, the function should be rewritten as a subroutine (a function with no return value).

$INCLUDE

Use $INCLUDE to textually include one file in another. The $INCLUDE facility is a simple, powerful way to create a consistent family of applications. By including source files containing commonly used functions, subroutines, constant definitions, aliases, etc., you have control over the source for each application. When you change the source, you update each application simply by recompiling (see Optimizations).

A file cannot include itself, either directly or indirectly. Include file nesting is allowed, but limited to a pre-defined maximum depth (currently 16).

The path of an include file is relative to the directory of the included file, not the current working directory of the compiler. Suppose, for example, the source program is in directory C:\WORK, and includes the file .C\H\HEADER, and the file HEADER includes COMMON. The compiler looks for COMMON in C:\H, not in C:\WORK.

```
C:\WORK
    A.BAS
        $INCLUDE "..\H\HEADER"
C:\H
    HEADER
        $INCLUDE "COMMON"
```

Compilation errors occur when a file is included multiple times. For example, if B.BAS includes files MATH and INCL, and INCL also includes MATH, MATH is included twice, causing a compile-time error.

```
B.BAS
    $INCLUDE "MATH"
    $INCLUDE "INCL"
INCL
    $INCLUDE "MATH"
```

1.9 Arrays and Function Parameter Lists

When an array parameter (formal) of a function or subroutine is declared, the number of dimensions is specified, but the extent of (number of elements in) each dimension is not specified. This allows the programmer some freedom when invoking such a function.

For example, a function may be defined to take a one-dimensional array and compute the sum of the elements in the array. A single function can be written to take a one-dimensional array of any size and correctly compute the sum.

(Because 950BASIC checks array bounds at run time on each access, there is no risk that a function will read or write outside the bounds of the array.)

When a formal parameter to a function is an array, instead of specifying the extent of each dimension, a list of variables is used to both implicitly specify the number of dimensions and to hold the extent of each dimension. These variables are read-only and cannot be modified within the function.

Adopt a convention for assigning names to placeholders. One such convention is to use the name of the array with a numerical suffix. For example,

```
function f(a(a1,a2,a3) as integer) as integer
```

where a1, a2, and a3 are the variables that get the extents of the array, a.

The function f above would be called as follows:

```
dim x_array(3,4,5) as integer
dim y_array(1,2,10) as integer
print f(x_array()) + f(y_array())
```

In both invocations of f, the function correctly determines the extent of each dimension of the passed array.

Remember that when passing an array to a function, the type of the array must match EXACTLY with the type expected by the function. Unlike scalar arguments (implicitly coerced from float to int or int to float), arrays are NOT coerced. An attempt to pass an integer array to a function that expects a float array results in a compile-time error.

Optimizations

As mentioned in an earlier section, constant definitions are completely 'folded' at the point of definition. This is efficient code. Constant expressions inside 950BASIC statements are also folded under certain conditions. For example, in the statement:

```
const PI = 3.1415926535
main
    print PI^2
end main
```

The value of PI^2 is not computed at run-time. It is detected as a constant value and pre-computed by the compiler as a single literal constant to be printed.

Similarly, the literal constant *3*4*PI* in

```
x = 3 * 4 * PI * x
```

is folded at compile-time, leaving only one multiplication to be performed at run-time.

However, certain constant expressions are not folded. For example:

```
x = 3 * PI * x * 4
```

is computed at run-time, involving 3 multiplications because the analysis of constant expressions does not attempt to exploit algebraic commutativity laws. Since the basic arithmetic operators are 'left associative', you can ensure the best performance by grouping constant factors together towards the left (or using a new constant definition).

If a function is not referenced (transitively from MAIN, plus any interrupt handlers), the compiler does not generate code for it. So, you can freely $include libraries with unused code (e.g., a comprehensive library containing functions supporting several possible axis configurations). Although the compiler parses and type-checks all the included source, it does not generate code into the downloaded program.

If select-case cases are all constants, more efficient code is generated. If a case is a variable, the generated code is equivalent to a string of if-then-else statements for all cases.

If any of the cases is an open-ended range (e.g., is 10), or covers a large range (e.g., 1 to 1000), a fast table-lookup is generated.

If all of the cases are constant, and can be grouped into locally dense subsets, the fastest possible code is generated — a binary search of dispatch tables, followed by an indirect jump through the table. If speed is a consideration, keep your cases constant and close together. (values form a reasonably dense set.)

The compiler performs limited dead-code elimination based on simple constant analysis. For example:

```
const DEBUGGING = FALSE
main
dim i, sum as integer
    for i = 1 to 10
        sum = sum + i
        if DEBUGGING then print “partial sum is ”;sum
    next i
end main
```

Since the value of DEBUGGING is FALSE, the compiler recognizes that the printing of the partial sum never happens and does not generate the print statement. This allows you to place debugging code in strategic locations in your programs and effectively disable it when shipping a production version (shrinks the size of the generated code).

This dead-code elimination also applies to functions whose only point of reference lies in eliminated code. The functions themselves become dead-code and no code is generated for their definitions.

The compiler does not eliminate the print statement from the following program:

```
dim DEBUGGING as integer
main
dim i, sum as integer
    DEBUGGING = FALSE
    for i = 1 to 10
        sum = sum + i
        if DEBUGGING print “partial sum is ”;sum
    next i
end main
```

In this case, the print statement never executes, but the code to implement is generated because the value of the integer DEBUGGING could be changed by the 950’s Integrated Development Environment Debugger at runtime, causing the print statement to be executed!

1.10 PACLAN

PACLAN is a local area network (LAN) providing high-speed (2.5 MBaud) inter-axis serial communication between Pacific Scientific SC950 single-axis programmable position controllers. The PACLAN provides support for up to 255 SC950 controllers. Information is passed between any two axes on a peer-to-peer basis. This capability is supported by specific features built into the BASIC language on the OC950.

PACLAN connectivity is an option and is only available on the OC950-503-01 and OC950-504-01 and OC950-603-01 and OC950-604-01 models. Use ModelExt to determine what type of OC950 you have.

Pre-defined variables on any other SC950 connected to the PACLAN are read using PACLAN. You can also generate interrupts on any of those axes, causing them to perform specific actions.

Configuration

Implementing a PACLAN network involves the following simple steps:

- Configure each SC950 on the PACLAN with a unique address using the address selection DIP switch on the OC950 card.
- Connect the SC950s with RG62 coax cable, terminating it at both ends with a 93 W terminator.
- Develop programs for the axes that incorporate inter-axis communications.

See Section 3.5 in MA950 - 0C950 Hardware and Installation Manual for cabling and hardware information.

Reading and Writing Pre-defined Variables

PACLAN provides interaxis communication of the pre-defined variables and PACLAN array variables. Inter-axis pre-defined variables are used in the same manner as local pre-defined variables. The SC950 accesses the variables over PACLAN.

Within a program, all off-axis variable accesses require the variable name to be appended with the axis address in square brackets. Axis designation is not required for on-axis variable usage.

Accessing Pre-defined Variables Over PACLAN

PACLAN provides read/write access to all pre-defined variables on all SC950s connected to the PACLAN. Use care in writing to pre-defined variables on another axis because extensive use of this capability leads to programs that are difficult to debug.

Each SC950 contains two uncommitted variable arrays (LANFlt and LANInt) specifically intended for inter-axis communications. These array variables also have read-write capability. See LANFlt( ) and LANInt( ).

Attempting to read from or write to a controller not present on the PACLAN results in a run-time error on the initiating controller. Use the pre-defined variable* Status[Axis #] *to determine if an axis is present on the PACLAN.

Example

PACLAN accesses any pre-defined variable on any other axis by appending the axis address in square brackets after the variable name.

For instance, to set the variable *x* equal to the value of Velocity on axis 3, use:

```
x = Velocity[3]
```

To set index distance on axis 5 equal to 10,000 counts, use:

```
IndexDist[5] = 10000
```

Pre-defined variables with an axis specifier are used wherever any other variables are used, with the exception of the WHEN statement.

LANInt() and LANFlt() Arrays

Two general purpose read/write variable arrays (one integer, one floating point) are available for user-defined inter-axis message passing. There are 32 elements in each array. These arrays are pre-defined variables with no pre-defined functionality. The integer array syntax is designated as:

x = LANInt(*y*)[*n*]
LANInt(*y*)[*n*] = x

where *y* is the array element (1-32) and *n* specifies the axis address containing the LAN array. The floating point array syntax is designated as:

x = LANFlt(*y*)[*n*]
LANFlt(*y*)[*n*] = x

where *y* is the array element (1-32) and *n* specifies the axis address containing the LAN array. For additional informationm see LANint( ) and LANFlt( ).

PACLAN Interrupts

PACLAN sends interrupts from a source axis to a destination axis. To send an interrupt to a program running on another axis, use SendLANInterrupt. This function allows you to specify the axis address of the program to which the interrupt is being sent. SendLANInterrupt allows you to send an integer argument along with the interrupt.

The receiving axis must have a PACLAN interrupt handler defined or SendLANInterrupt fails. There is a queue on each axis allowing each axis to buffer PACLAN interrupt requests.

Example

If axis 3 receives an interrupt from axis 5, it automatically jumps to a PACLAN interrupt handler and starts servicing the PACLAN interrupt. If axis 3 receives a PACLAN interrupt request from axis 2 before the request from axis 5 is complete, it buffers that request and services it after the request from axis 5. This queue holds 32 interrupt requests.

1.11 ModBus

The following functionality applies only to OC950s with** Enhanced Firmware**. Standard OC950s are are not capable of communicating on a ModBus network.

ModBus is a serial (RS232 or RS485) communications protocol consisting of one master and multiple slaves. The ModBus master initiates all transactions on the ModBus network. These transactions consist primarily of messages to read the values of data on a slave or to write new data values to a slave. The ModBus slaves generates responses to messages initiated by the master.

An OC950 is configured to operate as either a ModBus master or slave. In either case, there must be a program running on the OC950 for it to communicate on ModBus. When there is no program running on the OC950, the OC950 communicates using its native protocol.

ModBus Register and Data Types

There are two fundamental data types defined by ModBus: bits and registers

Bits

Bits are one bit of information. Bits are located at addresses 1-9999 (0x references) and 10001-19999 (1x references) in the ModBus address space. In ModBus terminology, bits are either coils (0x references) or inputs (1x references). Inputs are read-only while Coils are read-write.

An MMI or touchscreen uses a bit reference to read the value of the OC950's Moving pre-defined variable or to write a new value to the Dir variable.

Registers

Registers contain 16 bits of information. In the ModBus address space, registers are located at addresses 30001-39999 (3x references) and 40001-49999 (4x references). In ModBus terminology, registers are either Input Registers (3x references) or Holding Registers (4x references). Input Registers are read-only while Holding Registers are read-write.

Examples of using register references include an MMI or touchscreen using a register reference to read the value of Velocity or write a new value to IndexDist.

Floating-Point and 32 bit Integer Registers

There are two additional register data types which, while not explicitly defined by ModBus, are supported by many ModBus devices. These are 32-bit integer registers and 32-bit IEEE floating-point registers. Each of these extended types uses two adjacent 16-bit registers to hold the 32-bit value. The OC950 supports 32-bit integers and 32-bit floating-point as both a master and slave. The word-order of the two adjacent 16-bit registers are combined to form the extended type is configurable using MB32WordOrder and MBFloatWordOrder.

Using an OC950 as a ModBus Slave

Set up the OC950 as a ModBus slave to allow a ModBus master, such as a touchscreen or an MMI, to read and/or write values on the OC950. Configuring an OC950 to operate as a ModBus slave consists of adding the following items to your program:

1. An MBInfo block to map pre-defined variables and/or user-global variables to specific ModBus addresses.

   The MBInfo block contains multiple $MBMap<xxx> statements that specify this mapping. You can use the ModBus Map Wizard in the 950 IDE to assist you in creating this map. There is also an example program MBDEMO.BAS in the examples directory (\950win\examples) that contains a complete MBInfo block.

2. Adding a line to set RuntimeProtocol to 2 (ModBus Slave).

   You must set RuntimeProtocol to 2 to tell the OC950 to operate as a ModBus slave. After you set this, the OC950 responds to ModBus messages (both read and write), without any intervention from the user program.

   Keep in mind the following when configuring an OC950 as a ModBus slave:

   - the OC950 baud rate must match the master's. See BaudRate variable.
   - the OC950 parity must match the master's. See RuntimeParity.
   - the OC950 supports 1 start bit, 8 data bits and 1 stop bit
   - the OC950 does not require or support hardware handshaking. If the master requires it, defeat it on the master.
   - 255 is not a valid ModBus slave address. Setting RuntimeProtocol to 2 with an AxisAddr of 255 causes Runtime Error 38.

Using an OC950 as a ModBus Master

The ModBus Master functionality allows an OC950 to communicate with one or more ModBus slaves. Use an OC950 as a ModBus master to communicate with a Modicon PLC or some other device that operate only as a ModBus slave. As ModBus master, the OC950 initiates all traffic on the ModBus network.

To use an OC950 as ModBus master, set RuntimeProtocol to 3 (ModBus Master) and use any of the eight ModBus functions and statements to implement ModBus master functionality. If try to use one of these functions or statements without first setting RuntimeProtocol to 3, you'll get Runtime Error 37.

There are four ModBus statements added to the OC950 BASIC language to allow the OC950 to operate as a ModBus master to write data to a ModBus Slave. These are:

MBWriteBit(a, b, c)	write a bit (0x or 1x reference)
MBWrite16(a, b, c)	write a 16 bit integer (3x or 4x reference)
MBWrite32(a, b, c)	write a 32 bit integer (double 3x or 4x reference)
MBWriteFloat(a, b, c)	write a float (double 3x or 4x reference)

where, in each case:

a is the slave's ModBus address
b is the register address where the data is to be written
c is the new data

There are four ModBus functions added to the OC950 BASIC language to allow the OC950 to operate as a ModBus master to read data from a ModBus slave. These are:

x = MBReadBit(a, b)	read a bit (0x or 1x reference)
x = MBRead16(a, b)	read a 16 bit integer (3x or 4x reference)
x = MBRead32(a, b)	read a 32 bit integer (double 3x or 4x reference)
x = MBReadFloat(a, b)	read a float (double 3x or 4x reference)

where, in each case:

a is the slave's ModBus address
b is the register address containing the data being read

When one of these functions or statements is executed in your program, the OC950 sends a ModBus message to the specified slave and waits to process the response message. If any error occurs while sending or receiving the message, it is indicated in the variable MBErr.

ModBus master statements and functions cannot be nested. If you get an interrupt while waiting for the response to a ModBus master message, you cannot initiate another ModBus transaction (by executing one of the eight ModBus functions or statements) in the interrupt service routine. If you try, you generate Runtime Error 36.

Keep in mind the following when configuring an OC950 as a ModBus master:

- the OC950 baud rate must match the slaves'. See BaudRate variable.
- the OC950 parity must match the slaves'. See RuntimeParity.
- the OC950 supports 1 start bit, 8 data bits and 1 stop bit
- the OC950 does not require or support hardware handshaking. If a slave requires it, defeat it on the slave.

ModBus Reference

Refer to the following items in the reference section for additional information on ModBus:

Item	Used for Master or Slave?
BaudRate	Both
MB32WordOrder	Both
MBErr	Master
MBFloatWordOrder	Both
MBInfo Block	Slave
MBMap16	Slave
MBMap32	Slave
MBMapBit	Slave
MBMapFloat	Slave
MBRead16	Master
MBRead32	Master
MBReadBit	Master
MBReadFloat	Master
MBWrite16	Master
MBWrite32	Master
MBWriteBit	Master
MBWriteFloat	Master
RuntimeParity	Both
RuntimeProtocol	Both

1.12 Allen-Bradley DF1 Communications Protocol

The following functionality applies only to OC950s with** Enhanced Firmware**. Standard OC950s are are not capable of communicating on an Allen Bradley Communications network.

Allen-Bradley DF1 is a communications utility based on the DF1 peer-to-peer communications protocol. The functionality allows the SC950 to communicate with other devices supporting AB DF1on a peer-to-peer basis.

The SC950 is capable of responding to messages initiated by other devices (unsolicited commands) as well as initiating messages to read and write registers on other devices (solicited commands).

The SC950 support communications with the following Allen-Bradley PLCs.

- **SLC500 family of processors** — both solicited and unsolicited commands.
- **PLC5 family of processors** — solicited commands only (the SC950 can initiate read/write commands, but does not respond to read/write commands initiated by the PLC5).

Other devices supporting Allen Bradley DF1 Serial Communications protocol may be able to communicate with the SC950.

Procedure

To establish Allen-Bradley DF1 communications between the SC950 and another device:

1. The SC950 comm port (J51) must be properly wired to the other device.
2. All the software communication settings on both devices must match. For more detail, see ABCrc, BaudRate, and RuntimeProtocol. In general, the following settings are appropriatey for AB DF1..

	SC950	Other Device
Mode	RunTimeProtocol = 5 *	Full Duplex *
BaudRate	19200	19200
Data Bits	n/a	8 *
Stop Bits	n/a	1 *
Parity	Parity = 0	None
Error Detect	ABCrc = 1 ABCrc = 0	CRC BCC

* This parameter must be set to the value (setting) indicated.

Related Instructions

The 950BASIC language supports Allen-Bradley DF1 communications using the following commands / functions:

ABInfo Block
ReadPLC5Binary
ReadPLC5Float
ReadPLC5Integer
ReadSLC5Binary
ReadSLC5Float
ReadSLC5Integer
WritePLC5Binary
WritePLC5Float
WritePLC5Integer
WriteSLC5Binary
WriteSLC5Float
WriteSLC5Integer

Diagnostic Variables

There are several "diagnostic" counters maintained by the OC950 firmware as it processes Allen-Bradley DF1 messages. They can be helpful in diagnosing problems in setting up or maintaining an Allen-Bradley DF1 application. The variables and a brief expanation are shown below:

Variable	Explanation
ABAcksRcvd	# of message ACKs received
ABAcksSent	# of message ACKssent
ABAckTimeouts	# of messages received without an ACK
ABDupMsgs	# of duplicate messages discarded
ABErrCount	# of times ab_error(..) called
ABMsgsRcvd	# of messages received
ABMsgsSent	# of messagessent
ABNaksRcvd	# of message NAKs received
ABNaksSent	# of message NAKs sent
ABRspTimeouts	# of messages received without a response
ABTXQMax	max # of outbound messages stacked up
ABUnsRsps	# of unsolicited 'response' messagesreceived

ACK = Acknowledgement — received message is valid (correct CRC\BCC and frame).

NAK = Negative Acknowledgement — received message is invalid.

Map Wizard

This wizard creates and/or updates an ABInfo block in your program. The ABInfo block is used to map pre-defined variables or user-defined global variables to specific ABComm elements so an Allen-Bradley DF1device can read or write them. This mapping is only used when the OC950 is processing ABComm messages initiated by another device, not when it is initiating commands.

The wizard allows you to map OC950 variables (in Allen-Bradley DF1 terminology) as Integer file elements (Allen-Bradley pre-defined file # 7), or as Float file elements (Allen-Bradley pre-defined file # 8).

Procedure

To create a mapping of an OC950 variable to an Allen-Bradley DF1 element:

1. Select file type (Integer or Float)
2. Specify the element address
3. Specify the OC950 variable name

You may also specify an optional scale factor (the default=1.0). This scale factor is automatically applied when the Allen-Bradley DF1 element is read or written by the Allen-Bradley DF1 master. This is particularly useful for mapping floating-point OC950 variables into integer Allen-Bradley DF1 elements. It can also be used for mapping integer OC950.

Example

You could map RunSpeed as a 16-bit integer element and specify a scale factor of 10.

```
$ABMapInteger(1,runspeed,10.0)
```

Whenever the Allen-Bradley DF1 master reads integer element 1, the OC950 automatically multiplies the present value of RunSpeed by 10 and returns this value to the master. When the master writes to integer element 1, the OC950 automatically divides the new value by 10.0 before writing it to RunSpeed.

In this case, if the value of RunSpeed was 22.5 the Master reads 225 for integer element 1. Similarly, if the master wrote a value of 307, the RunSpeed is set to 30.7.

SLC500 to OC950 Cable

To establish Allen-Bradley DF1 communications between the SC950 and the SLC500 PLC, the following connections are required:

SC950 (J51) DB9	SLC500 (Channel 0) DB9
2 (RS232 TX)	2 (RS232 RX)
3 (RS232 RX)	3 (RS232 TX)
5 (Common)	5 (Common)

PLC5 to OC950 Cable

To establish Allen-Bradley DF1 communications between the SC950 and the PLC5, the following connections are required:

SC950 (J51) DB9	PLC5 (Channel 0) DB25
2 (RS232 TX)	3 (RS232 RX)
3 (RS232 RX)	2 (RS232 TX)
5 (Common)	7 (Common)

1.13 Cam Profiling

The following functionality applies only to OC950s with** Enhanced Firmware**. Standard OC950s are are not capable of cam profiling

In the 950, a cam is a cyclic, generally non-linear relationship between master encoder position and slave (motor) position. The relationship between slave and master counts is no longer a constant ratio, but changes as a function of master counts. As in electronic gearing, once a cam is active, the program no longer needs to do anything special to maintain it - the motion profile is repeated indefinitely until the cam is deactivated.

In camming terminology, a master is typically an external encoder. The encoder is wired into the SC950 encoder input port (connector J4 pins 21-24). It is also possible to use the SC950's virtual (internal) encoder.

Procedure

To use a cam profile on the SC950, you must:

1. Declare the cam ($DeclareCam).
2. Create the cam profile (CreateCam).
3. Activate the cam profile (ActiveCam).

Related Variables

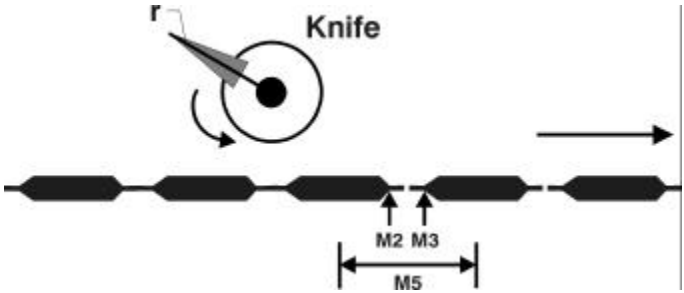
CamMaster Specifies the source of the input to the cam table for cam profiling.

CamCorrectDir Specifies the direction of the correction move that is done when a new cam table is activated (by setting ActiveCam = *n*).

Addpoint( ) Adds the specified "point" (master position and corresponding slave position) to the cam table being created.

Cam Wizard

The Cam Wizard is designed to solve cut to length applications. The picture below shows a typical setup:



In this application, material is being fed beneath a rotary knife. The master encoder measures forward movement of the material under the knife. The slave motor controls rotation of the knife. In order for this to work properly, the slave motor must be controlled (as a function of master encoder counts) so the blade of the rotary knife:

1. Stays out of the way until the proper amount of material has passed,
2. Accelerates so the speed of the knife matches the speed of the material during the cut and,
3. Decelerates back to the original speed until the material is almost in position for the next cut.

The rotary knife either accelerates or decelerates to match the speed of the material in the cut phase, depending on whether or not the circumference of the rotary knife is less than or greater than the length of the piece to be cut. You may need to interchange the terms 'accelerate' and 'decelerate', or simply think of them as signed quantities.

950BASIC's AddPoint statements specify a cam profile as a mapping from master position to slave position. The problem refers to relative velocities and accelerations. It is not always clear how to get from velocity and acceleration to position.

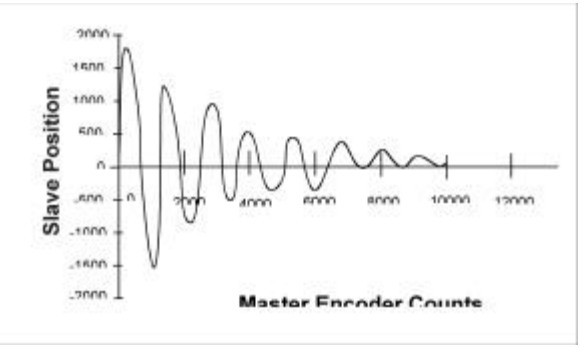
The Cam Wizard was designed to make such applications easy to implement. You provide:

1. the number of master counts corresponding to the length of material to be cut,
2. the number of slave counts corresponding to one complete rotation of the knife and,
3. the ratio of slave counts to master counts during the 'cut' phase of the cycle.

Once you have provided these three pieces of information, the Cam Wizard automatically:

1. generates code to declare a cam table of the correct size,
2. generates a subroutine to create the cam table and,
3. generates a subroutine to activate the cam.

Example You can create a cam to approximate any continuous function, but the Cam Wizard cannot help you with it. The basic technique is to develop a 950BASIC expression (or function)defining the slave position as a function of master position and use it to generate a series of AddPoint statements at appropriate master position intervals, such as the one shown in the next figure.



Program

```
const MC = 10000
‘ master counts in total cycle
const NPOINTS = 501
‘ number of points in cam profile
const pi = 3.1415926535
‘ tuning constants for nice motion
const k = 0.69314718/100
const w = 1/(7.5*pi)
‘_____________________________________________
$declarecam(2,NPOINTS)
‘_____________________________________________
‘ sub ActivateCam_2
sub activatecam_2
   Enable = 1
   EncPosModulo = MC
   PosModulo = MC
   EncPos = 0
   ActiveCam = 2
end sub
‘_____________________________________________
‘ sub CreateCam_2
‘ This code creates a cam whose profile is an exponentially
‘ damped sine wave.
sub CreateCam_2
  dim m,s as float
  dim i as integer
  CreateCam(2)
    for i = 0 to NPOINTS-1
                        ‘ master position
      m = i*(MC/(NPOINTS-1))
                        ‘ computed slave position
      s = (1/exp(1.5*k*i)) * sin(2*pi*w*i)
     addpoint(m,2000*s)
    next i
  end createcam
 end sub
```

Program (*continued*)

```
‘_____________________________________________-
‘ Generate a cam that does exponentially-damped sinusoidal
‘ motion, and activate it.  Please note that since we’re computing
‘ 500 points of slave profile here, several seconds will elapse
‘ during the calculation of the cam table.
main
  enable = 1
  vmdir = 0
  vmrunfreq = 1000
  vmgovel
  print “Creating cam 2"
  call CreateCam_2
  call ActivateCam_2
  print “Cam 2 is active now”
```

Virtual encoder (virtual master)

The virtual encoder is an internal count generator that is used as the input to the cam. It is controlled much like the profile generator used to control the motion of the motor. The pre-defined variables and statements associated with the virtual encoder are listed below:

Move Parameters	vmDir	specifies direction for vmGoVel
	vmIndexDist	specifies distance for vmGoIncr
	vmRunFreq	specifies speed (frequency) for vmGoIncr and vmGoVel
Move Statements	vmGoIncr	executes incremental move
	vmGoVel	executes velocity move
	vmUpdMove	updates move parameters on move in progress
	vmStopMotion	stops motion
Other Variables	vmEncpos	gives the value of the internal counter
	vmMoving	indicates whether a move is in progress

The virtual encoder is used as the input to the cam, either alone (as a virtual master) or in combination with the actual encoder (Encpos), to add an offset to the master position. This functionality is controlled by the variable, CamMaster.

2 QUICK REFERENCE

This section contains functions, parameters, statements and variables for 950BASIC. Below is a summary table of the list of instructions.

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 always retain their power on value at the start of program execution.

Name	Type	Default Value	Page #
$ABMapFloat( )	Statement		3-2
$ABMapInteger( )	Statement		3-3
$DeclareCam( )	Statement		3-4
$Include	Statement		3-5
$MBMapBit( )	Statement		3-6
$MBMap16( )	Statement		3-7
$MBMap32( )	Statement		3-8
$MBMapFloat( )	Statement		3-9
$PACLANAddr	Compiler Directive		3-9
ABCrc	Pre-defined Variable, Integer		3-10
ABErr	Pre-defined Variable, Integer		3-10
ABInfo...End			3-11
AbortMotion	Statement		3-12
Abs( )	Function		3-12
AccelGear	Pre-defined Variable, Integer	16,000,000 rpm/sec	3-13
AccelRate	Pre-defined Variable, Integer	10,000 rpm/sec	3-14
ActiveCam	Pre-defined Variable, Integer		3-15
AddPoint( )	Statement		3-17
ADF0	Pre-defined Variable, Float	1,000 Hz	3-18
ADOffset	Pre-defined Variable, Float	0 volts	3-18
Alias	Statement		3-19
AnalogIn	Pre-defined Variable, Float, Status Variable, Read Only		3-19
AnalogOut1	Pre-defined Variable, Float, Control Variable	0 volts	3-20
AnalogOut2	Pre-defined Variable, Float, Control Variable	0 volts	3-20
And	Operator		3-20
ARF0	Pre-defined Variable, Float, NV Parameter	set up	3-21

Name	Type	Default Value	Page #
ARF1	Pre-defined Variable, Float, NV Parameter	set up	3-21
ARZ0	Pre-defined Variable, Float	0 Hz	3-22
ARZ1	Pre-defined Variable, Float	0 Hz	3-22
Asc( )	Function		3-23
Atan( )	Function		3-23
Autostart	Pre-defined Variable, Integer	0	3-23
AxisAddr	Pre-defined Variable, Integer, Read-Only	255	3-24
Band	Operator		3-24
BaudRate	Pre-defined Variable, Integer	19200	3-25
BDInp1	Pre-defined Variable, Integer, Status Variable, Read Only		3-25
BDInp2-BDInp4	Pre-defined Variable, Integer, Status Variable, Read Only		3-26
BDInp5-BDInp6	Pre-defined Variable, Integer, Status Variable, Read Only		3-27
BDInputs	Pre-defined Variable, Integer, Status Variable, Read-Only		3-27
BDIOMap1	Pre-defined Variables, Integer, NV Parameter		3-28
BDIOMap2	Pre-defined Variables, Integer, NV Parameter		3-29
BDIOMap3	Pre-defined Variables, Integer, NV Parameter		3-30
BDIOMap4	Pre-defined Variables, Integer, NV Parameter		3-31
BDIOMap5	Pre-defined Variables, Integer, NV Parameter		3-32
BDIOMap6	Pre-defined Variables, Integer, NV Parameter		3-33
BDLgcThr	Pre-defined Variable, Integer	0	3-34
BDOut1	Pre-defined Variable, Integer, Control Variable	1	3-34
BDOut2-BDOut4	Pre-defined Variable, Integer, Control Variable	1	3-35
BDOut5-BDOut6	Pre-defined Variable, Integer, Control Variable	1	3-36
BDOutputs	Pre-defined Variable, Integer, Control Variable	63	3-37
Beep	Statement		3-37
BlkType	Pre-defined Variable, Integer	2	3-38
Bnot	Operator		3-38
Bor	Operator		3-39

Name	Type	Default Value	Page #
Brake	Pre-defined Variable, Integer, Mappable Output Function, Read-Only		3-39
Bxor	Operator		3-40
Call	Statement		3-40
CamCorrectDir	Pre-defined Variable, Integer	2	3-41
CamMaster	Pre-defined Variable, Integer	0	3-42
CamMasterPos	Pre-defined Variable, Integer, Read Only		3-42
CamSlaveOffset	Pre-defined Variable, Integer, Read Only		3-43
CCDate	Pre-defined Variable, Status Variable, Read Only	factory	3-43
CCSNum	Pre-defined Variable, Integer, Status Variable, Read Only	factory	3-43
CcwInh	Pre-defined Variable, Integer		3-44
Ccwot	Pre-defined Variable, Integer	0	3-44
Chr$( )	Function		3-44
Cint( )	Function		3-45
Cls	Statement		3-45
CmdGain	Pre-defined Variable, Float	0.5	3-45
CommEnbl	Pre-defined Variable, Integer, Control Variable	1	3-46
CommOff	Pre-defined Variable, Float, NV Parameter	set up	3-46
CommSrc	Pre-defined Variable, Integer	0	3-47
ConfigPLS( )	Statement		3-48
Const	Statement		3-49
Cos( )	Function		3-49
CountsPerRev	Pre-defined Variable, Integer	4096	3-49
CreateCam( )	Statement		3-50
CwInh	Pre-defined Variable		3-51
Cwot	Pre-defined Variable		3-52
DecelGear	Pre-defined Variable, Integer	16,000,000 rpm/sec	3-52
DecelRate	Pre-defined Variable, Integer	10,000 rpm/sec	3-53
Dim	Statement		3-54
Dir	Pre-defined Variable, Integer	0	3-54
DM1F0	Pre-defined Variable, Integer	1,000 Hz	3-55
DM1Gain	Pre-defined Variable, Float	0.6667	3-56
DM1Map	Pre-defined Variable, Integer	9	3-57

Name	Type	Default Value	Page #
DM1Out	Pre-defined Variable, Float, Status Variable, Read-Only		3-58
DM2F0	Pre-defined Variable, Float	1,000 Hz	3-58
DM2Gain	Pre-defined Variable, Float	2.0	3-59
DM2Map	Pre-defined Variable, Integer	1	3-60
DM2Out	Pre-defined Variable, Float, Status Variable, Read-Only		3-61
Enable	Pre-defined Variable, Integer	0	3-61
Enabled	Pre-defined Variable, Integer, Read-Only		3-62
EnablePLS0	Pre-defined Variable, Integer	0	3-62
EnablePLS1-EnablePLS2	Pre-defined Variable, Integer	0	3-63
EnablePLS3-EnablePLS4	Pre-defined Variable, Integer	0	3-64
EnablePLS5-EnablePLS6	Pre-defined Variable, Integer	0	3-65
EnablePLS7	Pre-defined Variable, Integer	0	3-66
EncFreq	Pre-defined Variable, Float, Status Variable, Read-Only		3-66
EncIn	Pre-defined Variable, Integer	1024	3-67
EncInF0	Pre-defined Variable, Float	800,000	3-68
EncMode	Pre-defined Variable, Integer	0	3-69
EncOut	Pre-defined Variable, Integer	1024	3-69
EncPos	Pre-defined Variable, Integer		3-70
EncPosModulo	Pre-defined Variable, Integer	0	3-70
End	Statement		3-71
Err	Pre-defined Variable		3-71
Exit	Statement		3-74
Exp( )	Function		3-74
ExtFault	Pre-defined Variable, Integer, Status Variable		3-75
Fault	Pre-defined Variable, Integer, Mappable Output Function		3-76
FaultCode	Pre-defined Variable, Integer, Status Variable, Read-Only		3-77
FaultReset	Pre-defined Variable, Integer, Mappable Input Function	0	3-78
Fix( )	Function		3-78
For...Next	Statement		3-79

Name	Type	Default Value	Page #
Function	Statement		3-80
FVelErr	Pre-defined Variable, Float, Status Variable, Read-Only		3-81
FwV	Pre-defined Variable, Integer, Status Variable, Read-Only		3-81
GearError	Pre-defined Variable, Integer		3-82
Gearing	Pre-defined Variable, Integer	0	3-83
GearLock	Pre-defined Variable, Float, Read-Only		3-84
GetMotor$( )	Function		3-85
GoAbs	Statement		3-85
GoAbsDir	Pre-defined Variable, Integer	3	3-86
GoHome	Statement		3-87
GoIncr	Statement		3-87
Goto	Statement		3-88
GoVel	Statement		3-88
Hex$( )	Function		3-89
HSTemp	Pre-defined Variable, Float, Status Variable, Read-Only		3-89
HwV	Pre-defined Variable, Integer, Status Variable, Read-Only		3-90
ICmd	Pre-defined Variable, Float, Status Variable, Read-Only		3-90
IFB	Pre-defined Variable, Status Variable, Read-Only		3-90
If...Then...Else	Statement		3-91
ILmtMinus	Pre-defined Variable, Integer, NV Parameter	set up	3-91
ILmtPlus	Pre-defined Variable, Integer, NV Parameter	set up	3-92
IndexDist	Pre-defined Variable, Integer	4096	3-92
Inkey$	String Function		3-93
Inp0-Inp20	Pre-defined Variable, Integer, Read-Only		3-93
InPosition	Pre-defined Variable, Integer, Read-Only		3-94
InPosLimit	Pre-defined Variable	5	3-94
Input	Statement		3-95
Inputs	Pre-defined Variable, Integer, Read-Only		3-95
Insert( )	Function		3-96
Int( )	Function		3-96
Interrupt...End Interrupt	Statement		3-97
Intr{source}	Pre-defined Variable, Integer		3-98

Name	Type	Default Value	Page #
Ipeak	Pre-defined Variable, Float, Status Variable, Read-Only		3-100
ItF0	Pre-defined Variable, Float	0.02 Hz	3-100
ItFilt	Pre-defined Variable, Float, Status Variable, Read-Only		3-101
ItThresh	Pre-defined Variable, Integer, NV Parameter	set up	3-102
ItThreshA	Pre-defined Variable, Float, Status Variable, Read-Only		3-102
I_R	Pre-defined Variable, Float, Status Variable, Read-Only		3-102
I_S	Pre-defined Variable, Float, Status Variable, Read-Only		3-102
I_T	Pre-defined Variable, Float, Status Variable, Read-Only		3-103
Kii	Pre-defined Variable, Float	50 Hz	3-103
Kip	Pre-defined Variable, Float, NV Parameter	set up	3-103
Kpp	Pre-defined Variable, Float, NV Parameter	set up	3-104
Kvff	Pre-defined Variable, Float	0 %	3-104
Kvi	Pre-defined Variable, Float, NV Parameter	set up	3-105
Kvp	Pre-defined Variable, Float, NV Parameter	set up	3-105
LanFLT( )	Pre-defined Array Variable, Float	0.0	3-106
LANint( )	Pre-defined Array Variable, Integer	0	3-106
LANInterrupt[ ]	Statement		3-107
LANIntrArg	Pre-defined Array Variable, Integer		3-107
LANIntrSource	Pre-defined Variable, Integer		3-108
Lcase$( )	Function		3-108
Left$( )	Function		3-109
Len( )	Function		3-109
Log( )	Function		3-109
Log10( )	Function		3-110
Ltrim$( )	Function		3-110
Main	Statement		3-111
MB32WordOrder	Pre-defined Variable	1	3-111
MBErr	Pre-defined Variable, Integer	0	3-112
MBFloatWordOrder	Pre-defined Variable	1	3-113
MBInfo Block…End	Statement		3-114
MBReadBit( )	Pre-defined Function		3-115
MBRead16( )	Pre-defined Function		3-116
MBRead32( )	Pre-defined Function		3-117
MBReadFloat( )	Pre-defined Function		3-118
MBWriteBit( )	Statement		3-119
MBWrite16( )	Statement		3-120

Name	Type	Default Value	Page #
MBWrite32( )	Statement		3-121
MBWriteFloat( )	Statement		3-122
Mid$( )	Function		3-123
Mod	Operator		3-123
Model	Pre-defined Variable, Integer, Status Variable, Read-Only		3-123
ModelExt	Pre-defined Variable, Integer, Status Variable, Read-Only		3-124
ModifyEncPos( )	Statement		3-124
Motor	Pre-defined Variable	sine(1, 162, 758, 483)	3-125
Moving	Pre-defined Variable, Integer, Read-Only	0	3-125
OCDate	Pre-defined Variable, Integer, Status Variable, Read-Only	factory	3-126
OCSNum	Pre-defined Variable, Integer, Status Variable, Read-Only	factory	3-126
Oct$( )	Function		3-126
On Error Goto	Statement		3-127
Or	Operator		3-128
Out0-Out20	Pre-defined Variable, Integer	1	3-128
Outputs	Pre-defined Variable, Integer	2,097,151	3-129
Params...EndParams	Statement		3-129
Pause( )	Statement		3-130
PoleCount	Pre-defined Variable, Integer, NV Parameter	set up	3-130
PosCommand	Pre-defined Variable, Integer		3-131
PosError	Pre-defined Variable, Integer, Status Variable, Read-Only		3-131
PosErrorMax	Pre-defined Variable, Integer	40960	3-132
Position	Pre-defined Variable, Integer, Read-Only		3-132
PosModulo	Pre-defined Variable, Integer	0	3-133
PosPolarity	Pre-defined Variable, Integer	0	3-134
Print	Statement		3-135
PulsesIn	Pre-defined Variable, Integer	1	3-135
PulsesOut	Pre-defined Variable, Integer	1	3-136
Random	Pre-defined Variable, Float, Read-Only		3-137

Name	Type	Default Value	Page #
Randomize	Statement		3-138
Ratio	Pre-defined Variable, Floating point	1.0	3-139
ReadPLC5Binary( )	Pre-defined Function		3-140
ReadPLC5Float( )	Pre-defined Function		3-141
ReadPLC5Integer( )	Pre-defined Function		3-142
ReadSLC5Binary( )	Pre-defined Function		3-143
ReadSLC5Float( )	Pre-defined Function		3-144
ReadSLC5Integer( )	Pre-defined Function		3-145
Reg1HiEncpos	Pre-defined Variable, Integer, Read-Only		3-146
Reg1HiFlag	Pre-defined Variable, Integer	0	3-146
Reg1HiPosition	Pre-defined Variable, Integer, Read-Only		3-147
Reg1LoEncpos	Pre-defined Variable, Integer, Read-Only		3-147
Reg1LoFlag	Pre-defined Variable, Integer	0	3-148
Reg1LoPosition	Pre-defined Variable, Integer, Read-Only		3-148
Reg2HiEncpos	Pre-defined Variable, Integer, Read-Only		3-149
Reg2HiFlag	Pre-defined Variable, Integer	0	3-149
Reg2HiPosition	Pre-defined Variable, Integer, Read-Only		3-150
Reg2LoEncpos	Pre-defined Variable, Integer, Read-Only		3-150
Reg2LoFlag	Pre-defined Variable, Integer	0	3-151
Reg2LoPosition	Pre-defined Variable, Integer, Read-Only		3-151
RegControl	Pre-defined Variable, Integer	0	3-152
RemoteFB	Pre-defined Variable, Integer	0	3-153
ResPos	Pre-defined Variable, Integer, Status Variable, Read-Only		3-154
Restart	Statement		3-154
Right$( )	Function		3-155
Rtrim$( )	Function		3-155
RunSpeed	Pre-defined Variable, Floating Point	1,000	3-155
RuntimeParity	Pre-defined Variable	0	3-156
RuntimeProtocol	Pre-defined Variable	0	3-156
ScurveTime	Pre-defined Variable, Floating Point	0	3-157
Select Case	Statement		3-158
SendLANInterrupt( )[ ]	Pre-defined function		3-159
SetMotor( )	Function		3-161

Name	Type	Default Value	Page #
Sgn( )	Function		3-161
SHL	Left Shift Operator		3-161
SHRA	Arithmetic Right Shift Operator		3-162
SHRL	Logial right Shift Operator		3-162
Sin( )	Function		3-162
Space$( )	Function		3-162
Sqr( )	Function		3-163
Static	Statement		3-163
Status[ ]	Pre-defined Variable		3-164
Stop	Statement		3-164
Str$( )	Function		3-165
String$( )	Function		3-165
Sub...End Sub	Statement		3-166
Swap	Statement		3-167
SysLanWindow1-8	Pre-defined Variable		3-167
Tan( )	Function		3-167
TargetPosition	Pre-defined Variable, Integer	0	3-168
Time	Pre-defined Variable, Float, Status Variable, Read-Only		3-168
Trim$( )	Function		3-169
Ucase$( )	Function		3-169
UpdMove	Statement		3-170
Val( )	Function		3-170
VBus	Pre-defined Variable, Float, Status Variable, Read-Only		3-171
VBusThresh	Pre-defined Variable, Float	-1	3-171
VelCmd	Pre-defined Variable, Float, Status Variable, Read-Only		3-172
VelErr	Pre-defined Variable, Float, Status Variable, Read-Only		3-172
VelFB	Pre-defined Variable, Float, Status Variable, Read-Only		3-172
VelLmtHi	Pre-defined Variable, Float	10,000	3-173
VelLmtLo	Pre-defined Variable, Float	-10,000	3-173
Velocity	Pre-defined Variable, Float, Status Variable, Read-Only		3-174
vmDir	Pre-defined Variable, Integer	0	3-174
vmEncpos	Pre-defined Variable, Integer		3-175
vmGoIncr	Statement		3-176
vmGoVel	Statement		3-177
vmMoving	Pre-defined Variable, Float, Read Only		3-178
vmRunFreq	Pre-defined Variable, Float	10,000	3-179
vmStopMotion	Statement		3-179
vmUpdMove	Statement		3-180
When	Statement		3-181

Name	Type	Default Value	Page #
WhenEncPos	Pre-defined Variable, Integer, Status Variable, Read-Only		3-182
WhenPosCommand	Pre-defined Variable, Integer, Status Variable, Read-Only		3-183
WhenPosition	Pre-defined Variable, Integer, Status Variable, Read-Only		3-183
WhenResPos	Pre-defined Variable, Integer, Status Variable, Read-Only		3-184
WhenTime	Pre-defined Variable, Float, Status Variable, Read-Only		3-184
While...Wend	Statement		3-184
WritePLC5Binary( )	Statement		3-185
WritePLC5Float( )	Statement		3-186
WritePLC5Integer( )	Statement		3-187
WriteSLC5Binary( )	Statement		3-188
WriteSLC5Float( )	Statement		3-189
WriteSLC5Integer( )	Statement		3-190
Xor	Operator		3-191

3 INSTRUCTIONS

This section is an alphabetical reference to 950BASIC instructions:

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

The name and type of each instruction is listed at the top of each 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 commands that are similar to this particular instruction.

Programming guidelines: Pertinent information about the instruction and its use.

Example program: Possible use of the instruction in a program.

$ABMAPFLOAT( )
(STATEMENT)

Purpose

$ABMapFloat( ) maps a float variable (pre-defined or user defined) to the SC950 Float File register.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

$ABMapFloat(*x, MyFloat)*
x = register number
MyFloat = Pre-defined or global user-defined float variable.

Guidelines

Only needed when the SLC500 initiates read (write) transactions from (to) the SC950.

Related Instructions

ABInfo

Example

This example maps a predefined variable (RunSpeed) and a global user variable (MyFlt) to SC950 ABComm Float file registers. RunSpeed is mapped to Register 1 of the SC950 Float file. MyFlt is mapped to Register 5 of the SC950 Float file.

```
Dim MyFlt as float
ABInfo
    $ABMapFloat(1, RunSpeed)
    $ABMapFloat(5, MyFlt)
End
```

$ABMAPINTEGER( )
(STATEMENT)

Purpose

$ABMapInteger( ) maps an integer variable (pre-defined or user defined) to the SC950 Integer File register.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

$ABMapInteger(*x, MyVar*)
x = register number.
MyVar = Predefined or global user-defined integer variable.

Guidelines

Only needed when the SLC500 initiates read (write) transactions from (to) the SC950.

Related Instructions

ABInfo

Example

This example maps a pre-defined variable (IndexDist) and a global user variable (MyInt) to SC950 Allen-Bradley Integer file registers. IndexDist is mapped to Register 1 of the SC950 Integer file. MyInt is mapped to Register 27 of the SC950 Integer file.

```
Dim MyInt as integer
ABInfo
    $ABMapInteger(1, IndexDist)
    $ABMapInteger(27, MyInt )
End
```

$DECLARECAM( )
(STATEMENT)

Purpose

$DeclareCam( ) allocates memory for the specified cam table. You must declare a cam table before you can create the cam table. The $DeclareCam( ) statement must be put before the word, MAIN, in your program.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

$DeclareCam(*x, y*)

where *x* is the cam number (1-8) and *y* is the maximum number of points put into the cam table. *y* must be less than 1000.

Guidelines

This statement allocates memory for the cam table. You cannot put in more points than you declare, but you can put in less.

Related Instructions

CreateCam( ), AddPoint( ), ActiveCam

Example

To declare cam #1 with 10 points, the statement is:

```
$DeclareCam(1, 10).
```

The $DeclareCam statement must appear before main.

```
$DeclareCam(1, 10)

main
  . . .
end
```

$INCLUDE
(STATEMENT)

Purpose The $Include statement allows you to textually include multiple separate files in a single source file.

Syntax $Include "*include-file-name*"

Guidelines A file cannot include itself, either directly or indirectly. Include file nesting is allowed to a depth of 16. Relative paths in a nested include file are relative to the directory location of the include file, not the current working directory of the compiler.

Example This example shows two files, myinc.inc and myfile.bas. The file myinc.inc has a sub-procedure for doing and incremental move that is used by the main program in myfile.bas.

```
MyInc.Inc
Sub DoIndexMove(Distance as integer)
        IndexDist = Distance
        GoIncr
        while Moving : wend
End Sub
MyFile.Bas
$Include "myinclude.inc"
Main
while 1
        call DoIndexMove(4096)
        Pause(0.5)
wend
End Main
```

$MBMAPBIT( )
(STATEMENT)

Purpose

$MBMapBit( ) maps a pre-defined variable or a global user variable to a ModBus Bit Register Address (0x reference or 1x reference).

This feature is only available in the Enhanced OC950 Firmware.

Syntax

$MapBit(*ModBus Address, Variable Name*)

Guidelines

This statement is used to map a pre-defined variable or a global user variable to a ModBus address when the 950 is acting as a ModBus slave.

Once a variable has been mapped and the ModBus Slave Protocol has been turned on (RuntimeProtocol=2), the ModBus master can read and/or write to this variable.

The $MBMapBit statement must be located inside an MBInfo block.

Related Instructions

RuntimeProtocol

Example

In the example below, Dir is mapped to ModBus address 1 and Enable is mapped to the ModBus address 10002.

```
MBInfo
        $MBMapBit( 1, Dir)
        $MBMapBit(10002, Enable)
End
```

$MBMAP16( )
(STATEMENT)

Purpose

$MBMap16( ) maps a pre-defined variable or a global user variable to a ModBus 16 Bit Register Address (3x reference or 4x reference).

This feature is only available in the Enhanced OC950 Firmware.

Syntax

$Map16(*ModBus Address, Variable Name*[,*ScaleFactor*])

Guidelines

Once a variable has been mapped and the ModBus Slave Protocol has been turned on (RuntimeProtocol=2), the ModBus master can read and/or write to these variables without any interaction by the user's program.

Related Instructions

$MBMap32( )

Example

In the example below, Faultcode is mapped to ModBus address 30001, RunSpeed is mapped to ModBus address 40001, and Velocity is mapped to ModBus address 40002 with a scale factor of 10.

```
MBInfo
        $MBMap16(30001, Faultcode)
        $MBMap16(40001, RunSpeed)
        $MBMap16(40002, Velocity, 10)
End
```

$MBMAP32( )
(STATEMENT)

Purpose

$MBMap32( ) maps a pre-defined variable or a global user variable to two contiguous ModBus 16 Bit Register Addresses (3x reference or 4x reference) as a 32 bit integer.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

$Map32(*ModBus Address, Variable Name[*,*ScaleFactor*])

Guidelines

Once a variable has been mapped and the ModBus Slave Protocol has been turned on (RuntimeProtocol=2), the ModBus master can read and/or write to these variables without any interaction by the user's program.

Related Instructions

MB32WordOrder , MBFloatWordOrder

Example

```
MBInfo
        $MBMap32( 30001, Position)
        $MBMap32( 30003, PosCommand)
        $MBMap32( 40001, IndexDist)
        $MBMap32( 40003, TargetPos)
End
```

$MBMAPFLOAT( ) *(STATEMENT)*

Purpose $MBMapFloat( ) maps a pre-defined variable or a global user variable to two contiguous ModBus 16 Bit register addresses (0x reference or 1x reference) as a floating point number.

This feature is only available in the Enhanced OC950 Firmware.

Syntax $MapFloat(*ModBus Address, Variable Name,*[*scale factor*])

Guidelines Once a variable is mapped and the ModBus slave protocol is on (RuntimeProtocol=2), the ModBus master reads and/or writes to these variables without user program interaction. The default [*scale factor*] is 1.0.

Related Instructions MB32WordOrder, MBFloatWordOrder

Example

```
MBInfo
        $MBMapFloat( 30001, Velocity, 1.0)
        $MBMapFloat( 30003, Time, 1.0)
        $MBMapFloat( 40001, RunSpeed, 1.0)
End
```

$PACLANADDR( ) *(COMPILER DIRECTIVE)*

Purpose $PACLANAddr( ) specifies the axes to which a program is downloaded. The $PACLANAddr directive must be enclosed in a ProgramInfo block. This is created automatically by the OC950 IDE when you use File|New to create a new program.

Syntax

```
ProgramInfo
$PACLANAddr( axis list )
End ProgramInfo
```

Guidelines Specify the number of axes in the axis list by separating them with commas. Specify a range of addresses using To.

Examples The first example shows a simple $PACLANAddr( ) directive that specifies axis 255. The second, a more complicated PACLANAddr( ) directive, specifies axes 1 - 3 and 6 - 9.

```
ProgramInfo
        $PACLANAddr(255)
End ProgramInfo

ProgramInfo
        $PACLANAddr(1,3, 6 to 9)
End ProgramInfo
```

ABCRC
(PRE-DEFINED VARIABLE, INTEGER)

Purpose ABCrc sets the method by which an Allen-Bradley DF1 message is checked for validity.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

ABCrc = 1	Sets message check method to CRC
ABCrc = 0	Sets message check method to BCC

Guidelines The setting in the SC950 MUST match the setting in the PLC.

Example The following program reads an integer from a SLC500 PLC. It then sets RunSpeed to twice the integer read.

All communication settings on both devices (SC950 and SLC500) must match.

```
main
dim SLC5Speed as integer
   runtimeprotocol = 5   'Allen-Bradley DF1 protocol
   baudrate = 19200      'baudrate must match PLC setting
   abcrc = 1             'Set check to CRC — MUST
   match PLC setting
   SLC5Speed = ReadSLC5Integer(5,7,19)
   RunSpeed = SLC5Speed * 2
end
```

ABERR
(PRE-DEFINED VARIABLE, INTEGER)

Purpose ABErr contains the error code of the last Allen-Bradley DF1 transaction.

This feature is only available in the Enhanced OC950 Firmware.

Syntax x = ABErr

Guidelines

ABErr	*Meaning*
0	No error
1	Response error
2	Response timeout
3	Max number of NAKs received
4	Max number of ENQs (enquiries) sent and still no response
5	SC950 Allen-Bradley DF1 receive buffer is full

ABINFO...END

Purpose

The ABInfo block section of a program is used to map pre-defined variables and/or global user variables to specific SC950 register addresses so that the OC950 can respond to unsolicited messages from a SLC500.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

```
ABInfo
    <$ABMap Statements>
End
```

Guidelines

This ABInfo block is only used when you are configuring the OC950 as an Allen-Bradley DF1device communicating with a SLC500. The ABInfo block is only needed when the SLC500 initiates read/wrtie commands to the SC950. If the SC950 initiates all read/write commands, the ABInfo block is unnecessary.

There can be only one ABInfo block in a program. It should be put before the Main section of the program.

Related Instructions

$ABMapFloat( ), $ABMapInteger( )

Example

This example maps several pre-defined variables and one global user variable (MyFloat) to SC950 Allen-Bradley Df1 file registers. IndexDist is mapped to Register 1 of the SC950 Integer file. Position is mapped to Register 27 of the SC950 Integer file. MyFloat is mapped to Register 9 of the SC950 Float file.

```
ABInfo
    $ABMapInteger(1, IndexDist)
    $ABMapInteger(27, Position )
    $ABMapFloat(9, MyFloat )
End
Dim MyFloat As Float

Main
    RuntimeProtocol = 5
    ...
```

ABORTMOTION
(STATEMENT)

Purpose AbortMotion stops motor motion, while allowing continued program execution.

Deceleration is determined by the motor torque capability in conjunction with the current limit parameters.

Syntax AbortMotion

Example This program segment commands the motor at constant velocity until input 1 goes to a logic 0. Then, the motor is commanded to stop.

```
AccelRate = 12000
'Set acceleration rate equal to 12,000 rpm/sec
RunSpeed = 120
'Set Run speed equal to 120 rpm
GoVel
When Inp1 = 0, AbortMotion
Print "Move Aborted!"
```

ABS( )
(FUNCTION)

Purpose Abs( ) converts the associated value (*x*) to an absolute value (positive value).

Syntax result = Abs(*x*)

Guidelines Enter the argument (*x*) immediately following the term, Abs.

Example

```
for x = -10 to 10
print Abs(x)
next
```

ACCELGEAR
(PRE-DEFINED VARIABLE, INTEGER)

Purpose

AccelGear sets the maximum acceleration commanded on the follower when Gearing is turned ON or the electronic gearing ratio (Ratio, PulsesOut, PulsesIn) is increased. This maximum acceleration limit remains in effect until Gearlock is achieved. Once Gearlock is achieved, the follower follows the master with the required acceleration or deceleration.

AccelGear is independent of DecelGear. Each variable must be set, independently, to the appropriate value for the desired motion.

Syntax

AccelGear = x

Units

rpm/sec

Range

1 to 16,000,000 rpm/sec

Default

16,000,000 rpm/sec

Guidelines

Set AccelGear prior to initiating Gearing.

Related Instructions

DecelGear, GearError, GearLock

Example

This example shows how to use AccelGear to limit acceleration and make up the lost distance.

```
AccelGear = 10000       'set AccelGear
Ratio     = 1.0
Enable    = 1
GearError = 0           'clear GearError
Gearing   = 1
While GearLock = 0
Wend                    'wait for LOCK
IndexDist = GearError
GoIncr
```

ACCELRATE
(PRE-DEFINED VARIABLE, INTEGER)

Purpose AccelRate (acceleration rate) sets the maximum commanded acceleration rate when the speed is increased.

AccelRate is independent of DecelRate. Each variable must be set, independently, to the appropriate value for the desired motion.

Syntax AccelRate = x

Units rpm/sec

Range 1 to 16,000,000 rpm/sec

Default 10,000 rpm/sec

Guidelines Set AccelRate prior to initiating the move. You can update AccelRate during a move by executing an UpdMove statement.

Related Instructions DecelRate

Example This example sets AccelRate to 10,000 rpm/sec and does an incremental move of 10 motor revolutions (assuming CountsPerRev is 4096).

```
RunSpeed = 1000
AccelRate = 10000
DecelRate = 10000
IndexDist = 40960
GoIncr
```

ACTIVECAM
(PRE-DEFINED VARIABLE, INTEGER)

Purpose

ActiveCam activates the specified cam table. The Position Command is calculated according to the Master Position (CamMasterPos) and the points in the specified cam table.

When you activate a new cam, the drive accelerates (at AccelGear) or decelerates (at DecelGear) as necessary to the speed required by the present motion of the Cam Master and the slave position profile defined in the cam table.

When speed synchronization is achieved, GearLock is set to one and a correction move is performed to bring the slave into position lock with the cam table. The direction of this move is controlled by CamCorrectDir. The parameters of this correction move are the same as for any other move (i.e., AccelRate, DecelRate, RunSpeed).

If the master is not moving or if the slave position profile in the cam table does not require cam motion when the cam is activated, the speed synchronization occurs instantly and the correction move is executed as soon as the cam is activated.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

ActiveCam = x

Range

0 to 8

Guidelines

ActiveCam is automatically set to zero (i.e., any cam is disengaged) when the drive is disabled.

To disable the correction move, set CamCorrectDir = 3.

You must declare and create a cam table before you make it active.

If RunSpeed is equal to zero when you set ActiveCam, a run-time error is generated because the correction move cannot be performed.

Related Instructions

CamCorrectDir

Example The following example declares, creates, and activates a cam.

```
$DeclareCam(1, 5)      'allocate space for cam #1, 5
points
main
  CreateCam(1)
'start the cam create block
    AddPoint(0, 0)
    AddPoint(200, 100)
    AddPoint(400, 200)
'add the points
    AddPoint(600, 300)
    AddPoint(800, 400)
   End
'end the cam create block
Enable = 1             'enable the motor
EncPosModulo = 800
'set EncPosModulo to master counts per cycle
PosModulo = 400
'sets PosModulo to slave (SC950) counts per cycle
EncPos = 0             'clear the counter
ActiveCam = 1          'activate cam #1
End
```

ADDPOINT( )
(STATEMENT)

Purpose

Addpoint( ) adds the specified "point" (master position and corresponding slave position) to the cam table being created. This statement is only used inside a CreateCam block.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

AddPoint(*master_position, slave_position*)

Guidelines

You must be inside a CreateCam block to use the Addpoint statement.

The master position for the first Addpoint statement in a CreateCam block must always be zero.

The master position must always increase as you add points to the cam table.

There must be at least three points in your cam table.

Related Instructions

$DeclareCam( )

Example

In the following example, a cam is declared, created, and activated.

```
$DeclareCam(1, 5)     'allocate space for cam #1, 5 points
main
  CreateCam(1)
'start the cam create block
   AddPoint(0, 0)
   AddPoint(200, 100)
   AddPoint(400, 200)
'add the points
   AddPoint(600, 300)
   AddPoint(800, 400)
  End                 'end the cam create block
Enable = 1            'enable the motor
EncPosModulo = 800
'set EncPosModulo to master counts per cycle
PosModulo = 400
'set PosModulo to slave counts per cycle
EncPos = 0            'clear the counter
ActiveCam = 1         'activate cam #1
End
```

ADF0
(PRE-DEFINED VARIABLE, FLOAT)

Purpose ADF0 is the first-order low-pass filter corner frequency for the analog input channel (J4-1 to J4-2).

Syntax ADF0 = x

Units Hertz

Range 0.01 to 4.17e7

Default 1,000 Hertz

Guidelines ADF0 is the corner frequency in Hz of the single-order low-pass filter. The purpose of the filter is to attenuate the high frequency components from the digitized input signal. Decreasing ADF0 lowers the response time to input changes, but also increases the effective resolution of AnalogIn.

	AnalogIn	
ADF0	**Effective Bits**	**LSB Size**
Max	14	1.6 mV
150	16	0.4 mV
10	18	0.1 mV

ADOFFSET
(PRE-DEFINED VARIABLE, FLOAT)

Purpose ADOffset adjusts the steady-state value of the analog command input.

Syntax ADOffset = x

Units Volts

Range -15 to +15

Default 0 volts

Guidelines AnalogIn is equal to the differential voltage between J4-1 and J4-2 plus ADOffset.

ALIAS
(STATEMENT)

Purpose	Alias allows you to define your own names for system resources, such as Input or Output pins.
Syntax	Alias <*name*> = <*expression*>
Guidelines	ALIAS is much more powerful than CONST. Constant expressions are computable at compile-time, whereas an alias has a value that may only be known at the time that it is being used. For this reason aliases should be used with care—-too much aliasing can make it very difficult for you to read your own program.
Related Instructions	Const

Example	Alias CONVEYOR_IS_RUNNING = (inp1=0)
	If CONVEYOR_IS_RUNNING Then
	  print “The conveyor is running”
	End If

ANALOGIN
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ ONLY)

Purpose	AnalogIn (Analog input) contains the digitized value of the analog input channel, which is the differential voltage of J4-1 (+) relative to J4-2 (-) after ADOffset is added and passed through ADFO low-pass filter.
Syntax	x = AnalogIn
Units	Volts
Range	-13.5 to +13.5
Default	None
Guidelines	AnalogIn can be monitored to check the presence and voltage of signals at the analog input terminals.

ANALOGOUT1
(PRE-DEFINED VARIABLE, FLOAT, CONTROL VARIABLE)

Purpose	AnalogOut1 (Analog Output1) sets the voltage level of the DAC Monitor 1 (J4-3) when DM1Map = 0.
Syntax	AnalogOut1 = *x*
Units	Volts
Range	-5.0 to +4.961
Default	0 volts
Guidelines	When DM1Map is not equal to 0, AnalogOut1 is not used.

ANALOGOUT2
(PRE-DEFINED VARIABLE, FLOAT, CONTROL VARIABLE)

Purpose	AnalogOut2 (Analog Output1) sets the voltage level of the DAC Monitor 2 (J4-4) when DM2Map = 0.
Syntax	AnalogOut2 = x
Units	Volts
Range	-5.0 to +4.961
Default	0 volts
Guidelines	When DM2Map is not equal to 0, AnalogOut2 is not used.

AND
(OPERATOR)

Purpose	And performs a logical AND operation on two expressions.
Syntax	result = A and B
Guidelines	The result evaluates to True if, and only if, both expressions are True. Otherwise, the result is False.
Related Instructions	Or, Xor, Band, Bor, Bxor

Example	x = 17
	y = 27
	If (x > 20) And (y > 20) Then
	  print “This won’t get printed”
	End If
	If (x < 20) And (y > 20) Then
	  print “This will get printed”
	End If

ARF0
(PRE-DEFINED VARIABLE, FLOAT, NV PARAMETER)

Purpose	ARF0 is the first velocity loop compensation anti-resonance low-pass filter corner frequency.
Syntax	ARF0 = x
Units	Hertz
Range	0.01 to 10e6 -10e6 to -0.01
Default	Parameter values are specified in the Params...End Params section of the program. The 950 IDE New Program function calculates this value based upon the specified motor and drive.
Guidelines	ARF0 is the corner frequency (in Hz) of one of two single-order low-pass anti-resonant filters or (if < 0) is the under-damped pole pair frequency in Hz and ARF1 is the pole pair Q. The purpose of the anti-resonant filters is to attenuate the velocity loop gain at the mechanical resonant frequency.
Related Instructions	ARF1, ARZ0, ARZ1

ARF1
(PRE-DEFINED VARIABLE, FLOAT, NV PARAMETER)

Purpose	ARF1 is the second velocity loop compensation anti-resonance low-pass filter corner frequency.
Syntax	ARF1 = x
Units	Hertz
Range	0.01 to 10,000,000 1 to 100 (Q)
Default	Parameter values are specified in the Params...End Params section of the program. The 950 IDE New Program function calculates this value based upon the specified motor and drive.
Guidelines	ARF1 is the corner frequency, in Hz, of one of two single-order low-pass anti-resonant filters or if ARF0 is < 0, ARF1 is the Q of the under damped pole pair. The purpose of the anti-resonant filters is to attenuate the velocity gain at the mechanical resonant frequency.
Related Instructions	ARF0, ARZ0, ARZ1

ARZ0
(PRE-DEFINED VARIABLE, FLOAT)

Purpose ARZ0 is the first velocity loop compensation zero.

Syntax ARZ0 = x

Units Hertz

Range 20 to 1e5
-1e5 to -35

Default 0 Hertz

Guidelines For very demanding compensation schemes, ARZ0 is used to add lead compensation or (with ARZ1) to add a notch filter. Otherwise, it is set to 0. ARZ0 positive sets the zero frequency in Hz and if < 0, sets the under damped zero pair frequency in Hz.

Related Instructions ARF0, ARF1, ARZ1

ARZ1
(PRE-DEFINED VARIABLE, FLOAT)

Purpose ARZ1 is the second velocity loop compensation zero.

Syntax ARZ1 = x

Units Hertz

Range 20 to 1e6
-100 to 100 (Q)

Default 0 Hertz

Guidelines For very demanding compensation schemes, ARZ1 is used to add lead compensation or (with ARZ0) to add a notch filter. Otherwise, it is set to 0. ARZ1 sets the zero frequency in Hz unless ARZ0 is set < 0. Then, ARZ1 sets the under damped zero pair Q.

Related Instructions ARF0, ARF1, ARZ0

ASC( )
(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 x = Asc(s$)

Guidelines If the string begins with an uppercase letter, the value of Asc( ) is between 65 and 90.

If the string begins with a lowercase letter, the value of Asc( ) is between 97 and 122.

Values 0 to 9 return 48 to 57.

ATAN( )
(FUNCTION)

Purpose Atan( ) (arc tangent) returns the arctangent of its argument in radians.

Syntax result = atan(*x*)

Guidelines The result is always between - ð/2 and ð/2.
The value of *x* may be any numeric type.
To convert from degrees to radians, multiply by 0.01745329

AUTOSTART
(PRE-DEFINED VARIABLE, INTEGER)

Purpose Autostart specifies whether or not the program in the OC950 starts executing automatically when AC power is applied.

0 = Program does not start automatically

1 = Program starts automatically

Syntax Autostart = x

Units none

Range 0 or 1

Default 0

Guidelines Set Autostart to 0 or 1 in the Variables Window (Compiler Menu, Variables option) of the 950 IDE.

AXISADDR
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose AxisAddr indicates the PacLAN address of the OC950. It is also used as a general configuration parameter, allowing you to have the same program in different drives that behave differently on some of them, depending on the value of the DIP switch.

Syntax x = AxisAddr

Units none

Range 1 to 255

Default Set by Address DIP Switch S1 on OC950.

Guidelines Every OC950 in a PacLAN network must have a unique address.

BAND
(OPERATOR)

Purpose Band performs a bitwise And of two integer expressions.

Syntax result = *x* Band *y*

Guidelines The Band operator performs a bitwise And operation on the two numeric expressions. The expressions are converted to integers (32 bits) before the Band operation takes place.

For each of the 32 bits in the result, the bit is set to 1 if, and only if, the corresponding bit in both of the arguments is 1.

Example

```
x = 45                  '0010 1101 binary
y = 99                  '0110 0011 binary
print x Band y          'prints: 33 (0010 0001)
```

BAUDRATE
(PRE-DEFINED VARIABLE, INTEGER)

Purpose BaudRate specifies the baudrate used on the OC950 Serial Port, either 19200 or 9600 baud.

Syntax BaudRate = x

Range 9600 or 19200

Default 19200

Guidelines When you configure your OC950 to communicate at 9600 baud, it communicates at this baudrate while the program is running and when the program is stopped. Therefore, it is essential that you also configure the 950IDE software on your PC to communicate at the same baudrate.

Once you configure your OC950 to communicate at 9600 baud, this information is retained after cycling power.

See Appendix A, "Operating at 9600 Baud" for additional information.

BDINP1
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ ONLY)

Purpose BDInp1 reads the state of BDIO1, J4-7.

Syntax *x* = BDInp1

Range 0 or 1

Guidelines BDInp1 indicates whether BDIO1 input voltage is above or below the logic threshold selected by the variable BDLgcThr.

BDInp1 = 0 indicates a logic low input

BDInp1 = 1 indicates a logic high input

BDINP2
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ ONLY)

Purpose BDInp2 reads the state of BDIO2, J4-8.

Syntax *x* = BDInp2

Range 0 or 1

Guidelines BDInp2 indicates whether BDIO2 input voltage is above or below the logic threshold selected by the variable BDLgcThr.

BDInp2 = 0 indicates a logic low input

BDInp2 = 1 indicates a logic high input

BDINP3
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ ONLY)

Purpose BDInp3 reads the state of BDIO3, J4-9.

Syntax *x* = BDInp3

Range 0 or 1

Guidelines BDInp3 indicates whether BDIO3 input voltage is above or below the logic threshold selected by the variable BDLgcThr.

BDInp3 = 0 indicates a logic low input

BDInp3 = 1 indicates a logic high input

BDINP4
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ ONLY)

Purpose BDInp4 reads the state of BDIO4, J4-10.

Syntax *x* = BDInp4

Range 0 or 1

Guidelines BDInp4 indicates whether BDIO4 input voltage is above or below the logic threshold selected by the variable BDLgcThr.

BDInp4 = 0 indicates a logic low input

BDInp4 = 1 indicates a logic high input

BDINP5
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ ONLY)

Purpose BDInp5 reads the state of BDIO5, J4-11.

Syntax *x* = BDInp5

Range 0 or 1

Guidelines BDInp5 indicates whether BDIO5 input voltage is above or below the logic threshold selected by the variable BDLgcThr.

BDInp5 = 0 indicates a logic low input
BDInp5 = 1 indicates a logic high input

BDINP6
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ ONLY)

Purpose BDInp6 reads the state of BDIO6, J4-12.

Syntax *x* = BDInp6

Range 0 or 1

Guidelines BDInp6 indicates whether BDIO6 input voltage is above or below the logic threshold selected by the variable BDLgcThr.

BDInp6 = 0 indicates a logic low input
BDInp6 = 1 indicates a logic high input

BDINPUTS
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ-ONLY)

Purpose BDInputs reads the state of the BDIO inputs in parallel. This variable is determined by the voltage levels applied to the BDIO input pins J4-7 to J4-12.

Syntax x = BDInputs

Range 0 to 63 (6 BDIOs)

Guidelines BDInputs = 1*BDIO1 + 2*BDIO2 + 4*BDIO3 + 8*BDIO4 + 16*BDIO5 + 32*BDIO6.

0 = low input
1 = high input.

For example, BDInputs = 12 means that BDIO 1, 2, 5, 6 are low and BDIO 3, 4 are high. See BDInp1-BDInp6 to query inputs individually.

BDIOMAP1
(PRE-DEFINED VARIABLES, INTEGER, NV PARAMETER)

Purpose BDIOMap1 sets the logical function of the BDIOs on J4-7.

Syntax BDIOMap1 = x

Range -2,147,482,648 to 2,147,482,648

Default Parameter value specified in the Params...End Params section of your program. The 950 IDE New Program function assigns BDIOMap1=Fault Reset Input Active Low.

Guidelines To use BDIO1 as a programmable Input/Output, set BDIOMap1 to zero.

Although the value is a 32 bit integer, the value is easily set in the Variables Screen or in the program using the following pre-defined constants for setting BDIOMap1:

Fault_Reset_Inp_Hi	Fault_Out_Hi
Fault_Reset_Inp_Lo	Fault_Out_Lo
CW_Inhibit_Inp_Hi	Enabled_Out_Hi
CW_Inhibit_Inp_Lo	Enabled_Out_Lo
CCW_Inhibit_Inp_Hi	Brake_Out_Hi
CCW_Inhibit_Inp_Lo	Brake_Out_Lo

Related Instructions Input Functions: FaultReset, CwInh, CcwInh
Output Functions: Fault, Enabled, Brake

Example BDIOMap1 = Enabled_Out_Lo maps Enabled as an active low output to J4-7.

BDIOMAP2
(PRE-DEFINED VARIABLES, INTEGER, NV PARAMETER)

Purpose BDIOMap2 sets the logical function ofBDIO on J4-8.

Syntax BDIOMap2 = x

Range -2,147,482,648 to 2,147,482,648

Default Parameter value specified in the Params...End Params section of your program. The 950 IDE New Program function assigns BDIOMap2=CW Inhibit Input Active Low.

Guidelines To use BDIO2 as a programmable Input/Output, set BDIOMap2 to zero.

Although the value is a 32 bit integer, the value is easily set in the Variables Screen or in the program using the following pre-defined constants for setting BDIOMap2:

Fault_Reset_Inp_Hi	Fault_Out_Hi
Fault_Reset_Inp_Lo	Fault_Out_Lo
CW_Inhibit_Inp_Hi	Enabled_Out_Hi
CW_Inhibit_Inp_Lo	Enabled_Out_Lo
CCW_Inhibit_Inp_Hi	Brake_Out_Hi
CCW_Inhibit_Inp_Lo	Brake_Out_Lo

Related Instructions Input Functions: FaultReset, CwInh, CcwInh
Output Functions: Fault, Enabled, Brake

Example BDIOMap2 = Enabled_Out_Lo maps Enabled as an active low output to J4-8.

BDIOMAP3
(PRE-DEFINED VARIABLES, INTEGER, NV PARAMETER)

Purpose BDIOMap3 sets the logical function ofBDIO on J4-9.

Syntax BDIOMap3 = x

Range -2,147,482,648 to 2,147,482,648

Default Parameter value specified in the Params...End Params section of your program. The 950 IDE New Program function assigns BDIOMap3=CCW Inhibit Input Active Low.

Guidelines To use BDIO3 as a programmable Input/Output, set BDIOMap3 to zero.

Although the value is a 32 bit integer, the value is easily set in the Variables Screen or in the program using the following pre-defined constants for setting BDIOMap3:

Fault_Reset_Inp_Hi	Fault_Out_Hi
Fault_Reset_Inp_Lo	Fault_Out_Lo
CW_Inhibit_Inp_Hi	Enabled_Out_Hi
CW_Inhibit_Inp_Lo	Enabled_Out_Lo
CCW_Inhibit_Inp_Hi	Brake_Out_Hi
CCW_Inhibit_Inp_Lo	Brake_Out_Lo

Related Instructions Input Functions: FaultReset, CwInh, CcwInh
Output Functions: Fault, Enabled, Brake

Example BDIOMap3 = Enabled_Out_Lo maps Enabled as an active low output to J4-9.

BDIOMAP4

(PRE-DEFINED VARIABLES, INTEGER, NV PARAMETER)

Purpose BDIOMap4 sets the logical function ofBDIO on J4-10.

Syntax BDIOMap4 = x

Range -2,147,482,648 to 2,147,482,648

Default Parameter value specified in the Params...End Params section of your program. The 950 IDE New Program function assigns BDIOMap4=OFF.

Guidelines To use BDIO4 as a programmable Input/Output, set BDIOMap4 to zero.

Although the value is a 32 bit integer, the value is easily set in the Variables Screen or in the program using the following pre-defined constants for setting BDIOMap4:

Fault_Reset_Inp_Hi	Fault_Out_Hi
Fault_Reset_Inp_Lo	Fault_Out_Lo
CW_Inhibit_Inp_Hi	Enabled_Out_Hi
CW_Inhibit_Inp_Lo	Enabled_Out_Lo
CCW_Inhibit_Inp_Hi	Brake_Out_Hi
CCW_Inhibit_Inp_Lo	Brake_Out_Lo

Related Instructions Input Functions: FaultReset, CwInh, CcwInh
Output Functions: Fault, Enabled, Brake

Example BDIOMap4 = Enabled_Out_Lo maps Enabled as an active low output to J4-10.

BDIOMAP5
(PRE-DEFINED VARIABLES, INTEGER, NV PARAMETER)

Purpose BDIOMap5 sets the logical function of BDIO on J4-11.

Syntax BDIOMap5 = x

Range -2,147,482,648 to 2,147,482,648

Default Parameter value specified in the Params...End Params section of your program. The 950 IDE New Program function assigns BDIOMap5=Brake Output Active High.

Guidelines To use BDIO5 as a programmable Input/Output, set BDIOMap5 to zero.

Although the value is a 32 bit integer, the value is easily set in the Variables Screen or in the program using the following pre-defined constants for setting BDIOMap5:

Fault_Reset_Inp_Hi	Fault_Out_Hi
Fault_Reset_Inp_Lo	Fault_Out_Lo
CW_Inhibit_Inp_Hi	Enabled_Out_Hi
CW_Inhibit_Inp_Lo	Enabled_Out_Lo
CCW_Inhibit_Inp_Hi	Brake_Out_Hi
CCW_Inhibit_Inp_Lo	Brake_Out_Lo

Related Instructions Input Functions: FaultReset, CwInh, CcwInh
Output Functions: Fault, Enabled, Brake

Example BDIOMap5 = Enabled_Out_Lo maps Enabled as an active low output to J4-11.

BDIOMAP6
(PRE-DEFINED VARIABLES, INTEGER, NV PARAMETER)

Purpose BDIOMap6 sets the logical function of BDIO on J4-12.

Syntax BDIOMap6 = x

Range -2,147,482,648 to 2,147,482,648

Default Parameter value specified in the Params...End Params section of your program. The 950 IDE New Program function assigns BDIOMap6=Fault Output Active High.

Guidelines To use BDIO6 as a programmable Input/Output, set BDIOMap6 to zero.

Although the value is a 32 bit integer, the value is easily set in the Variables Screen or in the program using the following pre-defined constants for setting BDIOMap6:

Fault_Reset_Inp_Hi	Fault_Out_Hi
Fault_Reset_Inp_Lo	Fault_Out_Lo
CW_Inhibit_Inp_Hi	Enabled_Out_Hi
CW_Inhibit_Inp_Lo	Enabled_Out_Lo
CCW_Inhibit_Inp_Hi	Brake_Out_Hi
CCW_Inhibit_Inp_Lo	Brake_Out_Lo

Related Instructions Input Functions: FaultReset, CwInh, CcwInh
Output Functions: Fault, Enabled, Brake

Example BDIOMap6 = Enabled_Out_Lo maps Enabled as an active low output to J4-12.

BDLGCTHR
(PRE-DEFINED VARIABLE, INTEGER)

Purpose BDLgcThr sets the switching threshold for the Base drive inputs (BDInp1 - BDInp6) and the pull up voltage for the Base drive outputs (BDOut1 - BDOut6).

Syntax BDLgcThr = *x*

Range 0 or 1

Default 0 (5 volt compatible)

Guidelines **0** selects 5 volt logic compatibility
1 selects 24 volt logic compatibility

BDLgcThr	Low (Volts)	High (Volts)	Pull up (Volts)
0	2.1	3.1	5.0
1	4.0	5.0	12.0

BDOUT1
(PRE-DEFINED VARIABLE, INTEGER, CONTROL VARIABLE)

Purpose BDOut1 allows setting the output logic state of BDIO1 not mapped to an output function via BDIOMap1. BDOut1 sets the state of BDIO1, J4-7.

Syntax BDOut1 = *x*

Range 0 or 1

Default 1 (transistor turned off)

Guidelines 0 turns on the pull down transistor
1 turns off the pull down transistor
To use BDIO1 as an input, BDOut1 must be set to 1 (default).

BDOUT2
(PRE-DEFINED VARIABLE, INTEGER, CONTROL VARIABLE)

Purpose	BDOut2 allows setting the output logic state of BDIO2 not mapped to an output function via BDIOMap2. BDOut2 sets the state of BDIO2, J4-8.
Syntax	BDOut2 = x
Range	0 or 1
Default	1 (transistor turned off)
Guidelines	0 turns on the pull down transistor 1 turns off the pull down transistor To use BDIO2 as an input, BDOut2 must be set to 1 (default).

BDOUT3
(PRE-DEFINED VARIABLE, INTEGER, CONTROL VARIABLE)

Purpose	BDOut3 allows setting the output logic state of BDIO3 not mapped to an output function via BDIOMap3. BDOut3 sets the state of BDIO3, J4-9.
Syntax	BDOut3 = x
Range	0 or 1
Default	1 (transistor turned off)
Guidelines	0 turns on the pull down transistor 1 turns off the pull down transistor To use BDIO3 as an input, BDOut3 must be set to 1 (default).

BDOUT4
(PRE-DEFINED VARIABLE, INTEGER, CONTROL VARIABLE)

Purpose	BDOut4 allows setting the output logic state of BDIO4 not mapped to an output function via BDIOMap4. BDOut4 sets the state of BDIO4, J4-10.
Syntax	BDOut4 = x
Range	0 or 1
Default	1 (transistor turned off)
Guidelines	0 turns on the pull down transistor 1 turns off the pull down transistor To use BDIO4 as an input, BDOut4 must be set to 1 (default).

BDOUT5
(PRE-DEFINED VARIABLE, INTEGER, CONTROL VARIABLE)

Purpose BDOut5 allows setting the output logic state of BDIO5 not mapped to an output function via BDIOMap5. BDOut5 sets the state of BDIO5, J4-11.

Syntax BDOut5 = x

Range 0 or 1

Default 1 (transistor turned off)

Guidelines **0** turns on the pull down transistor
1 turns off the pull down transistor
To use BDIO5 as an input, BDOut5 must be set to 1 (default).

BDOUT6
(PRE-DEFINED VARIABLE, INTEGER, CONTROL VARIABLE)

Purpose BDOut6 allows setting the output logic state of BDIO6 not mapped to an output function via BDIOMap6. BDOut6 sets the state of BDIO6, J4-12.

Syntax BDOut6 = x

Range 0 or 1

Default 1 (transistor turned off)

Guidelines **0** turns on the pull down transistor
1 turns off the pull down transistor
To use BDIO6 as an input, BDOut6 must be set to 1 (default).

BDOUTPUTS
(PRE-DEFINED VARIABLE, INTEGER, CONTROL VARIABLE)

Purpose For BDIO outputs not mapped to an output function via BDIOMap, allows setting their output logic state in parallel.

Syntax BDOutputs = x

Range 0 to 63 (6 BDIOs)

Default 63

Guidelines BDOutputs = 1*BDIO1 + 2*BDIO2 + 4*BDIO3 + 8 *BDIO4 + 16*BDIO5 + 32*BDIO6.

0 turns on the corresponding pull down transistor
1 turns off the corresponding pull down transistor.

BDIOs mapped to output functions via their BDIOMap are determined by that function and their value in BDOutputs is ignored.

Example BDInputs = 12 pulls down BDIO 1, 2, 5, 6 and open circuit BDIO 3, 4. See BDOut1-BDOut6 to control outputs individually.

BEEP
(STATEMENT)

Purpose Beep transmits a BEEP character (ASCII 07) to the serial port.

Syntax Beep

Example
```
print “Listen to this...”
pause(0.5)
Beep
```

BLKTYPE
(PRE-DEFINED VARIABLE, INTEGER)

Purpose BlkType specifies configuration as a position, velocity, or torque block.

Syntax BlkType = x

Range 0, 1 or 2

Default 2 (Position Mode)

Guidelines BlkType sets the overall control functionality of the drive. For block diagrams of the drive configurations, refer to the manual (alternative BlkType settings). When used in any of the analog modes, the analog control is the differential voltage applied to the Analog Cmd+ (Analog Command +) and Analog Cmd- (Analog Command -) inputs (J4-1 and J4-2 respectively).

BlkType	Servo Configuration
0	Analog Torque Block
1	Analog Velocity Block
2	Digital Position Block

BNOT
(OPERATOR)

Purpose Bnot performs a bitwise NOT of the integer expression.

Syntax result = Bnot *x*

Guidelines The Bnot operator performs a bitwise NOT operation on a numeric expression. The expression is converted to an integer (32 bits) before the BNOT operation takes place.

For each of the 32 bits in the result, the bit is set to 1 if the corresponding bit in the argument is 0. The bit is set to 0 if the corresponding bit in the argument is 1.

Example
```
x = 45
‘0010 1101 binary
print Bnot x
‘prints: -46
```

BOR
(OPERATOR)

Purpose Bor performs a bitwise OR of two integer expressions.

Syntax result = *x* Bor *y*

Guidelines Bor performs a bitwise OR operation on the two numeric expressions. The expressions are converted to integers (32 bits) before the BOR operation takes place.

For each of the 32 bits in the result, the bit is set to 1 if the corresponding bit in either of the arguments is 1.

Example

```
x = 45                  '0010 1101 binary
y = 99                  '0110 0011 binary
print x Bor y           'prints: 111 (0110 1111)
```

BRAKE
(PRE-DEFINED VARIABLE, INTEGER, MAPPABLE OUTPUT FUNCTION, READ-ONLY)

Purpose Brake indicates when the motor is not powered and a mechanical brake needs to hold the motor.

Syntax x = Brake

Range 0 or 1

Guidelines 0 = the motor is powered and the brake should be off.
1 = the mechanical brake should engage

To insure that a mechanical brake is engaged when a drive's control power is removed, map the active high Brake function to a BDIO pin.

BXOR
(OPERATOR)

Purpose Bxor performs a bitwise XOR of two integer expressions.

Syntax result = *x* Bxor *y*

Guidelines Bxor performs a bitwise XOR operation on the two numeric expressions. The expressions are converted to integers (32 bits) before the BXOR operation takes place.

For each of the 32 bits in the result, the bit is set to 1 if the corresponding bits in the two arguments are different from each other. If the corresponding bits are identical (both 0 or both 1), the bit is set to 0.

Example

```
x = 45                  '0010 1101 binary
y = 99                  '0110 0011 binary
print x Bor y           'prints: 78 (0100 1110)
```

CALL
(STATEMENT)

Purpose Call transfers program control to a subroutine. When the subroutine is complete, control is transferred to the line following the Call. Call statement replaces the GoSub statement (no longer supported).

Syntax Call sub [(*arg1, arg2*, ...)]

Guidelines A subroutine is essentially a function with no return value. Arguments to subroutines are passed by value. This means that the subroutine receives a copy of these arguments. Any assignments to these arguments made by the subroutine have no effect on these variables in the calling function or subroutine.

Related Instructions Sub

Example

```
Call PrintSum(3, 4)
...
Sub PrintSum( i, j as integer)
        print i+j
End Sub
```

CAMCORRECTDIR
(PRE-DEFINED VARIABLE, INTEGER)

Purpose CamCorrectDir specifies the direction of the correction move when a new cam table is activated (set ActiveCam = *n*) or when speed synchronization is achieved.

This feature is only available in the Enhanced OC950 Firmware.

Syntax CamCorrectDir = *x*

Range 0 to 3

Default 2 (shortest distance)

Guidelines CamCorrectDir takes one of the following values:

0 move is done clockwise

1 move is done counter-clockwise

2 move is done in the direction yielding the shortest move (see below)

3 no correction move is performed.

Use AccelRate, DecelRate and RunSpeed for a correction move. Even if CamCorrectDir specifies a clockwise correction move, it only specifies the direction of the superimposed move. If the cam generated speed is the opposite direction and larger than RunSpeed, the slave slows down.

For CamCorrectDir = 2, the direction of the correction is calculated (based upon PosModulo) to yield the shortest distance move. For example, if PosModulo = 10000 and the clockwise correction move is 8000, a counter-clockwise move of 2000 is performed instead.

Related Instructions ActiveCam

Example In the following example, the correction move is in the direction yielding the shortest move distance.

```
....
'The cam table for Cam #1 needs to have been
'already declared and created
'_____________________________________________

CamCorrectDir = 2
ActiveCam = 1
  ....
```

CAMMASTER
(PRE-DEFINED VARIABLE, INTEGER)

Purpose CamMaster is used to specify the source of the input to the cam table for cam profiling.

This feature is only available in the Enhanced OC950 Firmware.

Syntax CamMaster = *x*

Range 0 to 2

Default 0 ("real" encoder + "virtual" encoder)

Guidelines CamMaster takes one of the following values:

0 Encpos + vmEncpos

1 vmEncpos only (Encpos is ignored)

2 Encpos only (vmEncpos is ignored)

Related Instructions CamMasterPos

CAMMASTERPOS
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose CamMasterPos gives the value of the master position presently being used as the input to the cam table. The value of CamMasterPos depends upon Encpos, vmEncpos and CamMaster as follows:

Value of CamMaster	Value of CamMasterPos
0	vmEncpos + Encpos
1	vmEncpos
2	Encpos

This feature is only available in the Enhanced OC950 Firmware.

Syntax *x* = CamMasterPos

Units encoder counts

Range 0 - EncposModulo

Related Instructions CamMaster, Encpos, vmEncpos

CAMSLAVEOFFSET
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose CamSlaveOffset indicates the offset (or difference) between PosCommand and the position command that is calculated from the active cam table based upon the present value of Encpos and/or vmEncpos. This offset is the result of incremental (GoIncr) or velocity (GoVel) moves superimposed (by you) on the cam table.

This feature is only available in the Enhanced OC950 Firmware.

Syntax x = CamSlaveOffset

Units feedback counts

Range 0 - PosModulo

Guidelines If there is no active cam (ActiveCam = 0), the value of this variable is undefined.

CCDATE
(PRE-DEFINED VARIABLE, STATUS VARIABLE, READ ONLY)

Purpose CCDate gives the Control Card date code.

Syntax CCDate = *x*

Range 0 to 231

Default Set at factory

CCSNUM
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ ONLY)

Purpose CCSNum gives the Control Card serial number.

Syntax CCSNum = *x*

Range 0 to 231

Default Set at factory

CCWINH
(PRE-DEFINED VARIABLE, INTEGER)

Purpose CcwInh indicates the current state of the CCWINH (Inhibit -) Input. It can also be used as an interrupt source.

Syntax *x* = CcwInh

Range 0 or 1

Units none

Default none

CCWOT
(PRE-DEFINED VARIABLE, INTEGER)

Purpose Ccwot sets the counter-clockwise software over-travel limit. When the position of the motor becomes more negative than this limit, a counter-clockwise over-travel interrupt occurs if that interrupt is active.

Syntax Ccwot = *x*

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

Units resolver counts

Default 0

CHR$( )
(FUNCTION)

Purpose Chr$( ) returns a one character string whose ASCII value is the argument.

Syntax s$ = Chr$(*x*)

Guidelines The argument to Chr$( ) must be a numeric value in the range 0 to 255.

Example This example prints an uppercase B.

```
dim a$ as string
  a$ = Chr$(66)
  print a$
```

CINT( )
(FUNCTION)

Purpose Cint( ) converts a numeric expression to the closest integer number.

Syntax *x* = Cint(*numeric-expression*)

Related Instructions Int( ), Fix( )

CLS
(STATEMENT)

Purpose Cls transmits 40 line feed characters (ASCII code = 10) to the serial port. Cls clears the display of a terminal.

Syntax Cls

Example
```
print “Take a good look now ...”
pause(2)
cls
```

CMDGAIN
(PRE-DEFINED VARIABLE, FLOAT)

Purpose CmdGain sets the scale factor of the analog input for BlkTypes 0 and 1.

Syntax CmdGain = x.x

Units, Range

BlkType = 0 amperes/volt	$\pm 1010 * I_{PEAK}$
BlkType = 1 krpm/volt	±1010

Default 0.5

Guidelines CmdGain is a floating point variable that sets the command gain on the analog input (voltage from J4-1 to J4-2) for BlkTypes 0 (Analog torque block) and 1 (Analog velocity block).

COMMENBL
(PRE-DEFINED VARIABLE, INTEGER, CONTROL VARIABLE)

Purpose CommEnbl allows/disallows normal commutation.

Syntax CommEnbl = *x*

Range 0 or 1

Default 1

Guidelines 0 (disables commutation. Commutation angle set by CommOff)
1 (enables commutation)

CommEnbl must always be 1 for normal operation. Leaving CommEnbl at 0 can overheat and possibly damage the motor.

COMMOFF
(PRE-DEFINED VARIABLE, FLOAT, NV PARAMETER)

Purpose CommOff sets the origin for the electrical commutation angle.

Syntax CommOff = x.x

Units degrees

Range 0 to 360

Default Parameter value specified in the Params...End Params section of your program. The 950 IDE New Program function sets this value to 0 degrees.

Guidelines The value for standard Danaher Motion's Pacific Scientific motors is 0.

For CommSrc = 1 (incremental encoder commutation) CommOff is set to 0 on every power up, independent of the value in the non-volatile memory. Drive RAM value is always read/write.

COMMSRC
(PRE-DEFINED VARIABLE, INTEGER)

Purpose CommSrc selects resolver or incremental encoder feedback for motor commutation.

Syntax CommSrc = *x*

Range 0 or 1

Default 0 (resolver)

Guidelines **0** selects resolver feedback commutation — PoleCount set to number of motor pole pairs.

1 selects incremental encoder feedback commutation — PoleCount set to number of quadrature encoder counts per motor electrical cycle.

Writing to CommSrc sets Polecount = 0. Therefore, first set CommSrc to the correct value and then set PoleCount.

CONFIGPLS( )
(STATEMENT)

Purpose ConfigPLS( ) configures the functionality of one of the eight Programmable Limit Switches (PLS) on the OC950.

Syntax ConfigPLS(*PLSNumber, StartPosition, Duration, ActiveLevel, Source*)

PLSNumber:	the PLS being configured (0-7)
StartPosition:	the position where the PLS turns on
Duration:	the distance for which the PLS is on
ActiveLevel:	0 - output is set to zero when the PLS is ON
	1 - output is set to one when the PLS is ON
Source:	0 - Resolver Position
	1 - Encoder Position

Guidelines ConfigPLS( ) configures the PLS. You must enable the PLS using the appropriate EnablePLS*x* pre-defined variable before the PLS starts executing.

PLSs are used to generate position based interrupts. The I/O points are bi-directional on the OC950. Therefore, configure an interrupt to occur on the rising/falling edge of the Input (IntrI0Hi) associated with the Output (Out0) that the PLS (PLS0) is controlling.

Related Instructions EnablePLS*x*

Example The statements below configures PLS0 such that Out0 is set to 1 when Position is between 4096 and 4196. Out0 is set to 0 at all other times.

```
ConfigPLS(0, 4096, 100, 1, 0)
EnablePLS0 = 1
```

The example below configures PLS0 to generate an interrupt once during each revolution of the motor.

```
Main
        PosModulo = 4096
        ConfigPLS(0, 2048, 500, 1, 0)
        EnablePLS0 = 1
        Enable = 1
        IntrI0Hi = 1
        Runspeed = 1000
        GoVel
        While 1:wend
End
Interrupt I0Hi
        Print “Interrupt generated on PLS0”
        IntrI0Hi = 1        ‘Re-enable “I0Hi” interrupt on exit”
End Interrupt
```

CONST
(STATEMENT)

Purpose Const declares symbolic constants to be used instead of numeric values.

Syntax Const name = *x*

Guidelines The CONST statement makes your program much more readable and self-documenting. Unlike variables, constants assume only one value in a program.

Related Instructions Alias

Example
```
Const SLEW_SPEED = 2500
Const WORK_SPEED = 100
RunSpeed = SLEW_SPEED : GoVel
Pause(0.5)
RunSpeed = WORK_SPEED : GoVel
```

COS( )
(FUNCTION)

Purpose Cos(*x*) returns the cosine of *x*, where *x* is in radians.

Syntax *y* = Cos(*x*)

Guidelines *x* must be in radians. To convert from degrees to radians, multiply by 0.017453.

COUNTSPERREV
(PRE-DEFINED VARIABLE, INTEGER)

Purpose CountsPerRev specifies the scaling of all position-based pre-defined variables.

Syntax CountsPerRev = *x*

Units Resolver Counts

Range 4096, 8192, 16384, 32768, 65536

Default 4096

Guidelines CountsPerRev specifies the scaling and hence, the resolution, of all position based variables. The default value is 4096 resolver counts per motor revolution (5.27 arc-min).

This variable controls the resolution of position variables. It does not affect accuracy.

CREATECAM( )
(STATEMENT)

Purpose

CreateCam( ) initiates the creation of a cam table. The actual points in the cam table are inserted with a series of AddPoint( ) statements. The CreateCam( ) block must terminated by an End statement.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

```
CreateCam( n )
        AddPoint(0, y1 )
        ....
        AddPoint(xx, yy)
End
```

where *n* is the cam number (1-8) of the cam table that you are creating.

Guidelines

You must declare a cam table before you create the cam table. You can create a cam table as many times as you want. You must create a cam table before you make it active. You cannot create a cam table if it is active. The master position for the first entry must be 0. The master positions must keep increasing as you add points. EncPosModulo must equal the total master distance in you CAM. For a repeating CAM, PosModulo should be set equal to the distance that the slave travels in one CAM cycle.

Related Instructions

$DeclareCam( ), AddPoint( ), ActiveCam

Example In the following example, a cam is declared, created, and activated.

```
$DeclareCam(1, 5)
‘allocate space for cam #1, 5 points
main
  CreateCam(1)
‘start the cam create block
    AddPoint(0, 0)
    AddPoint(200, 100)
    AddPoint(400, 200)
‘add the points
    AddPoint(600, 300)
    AddPoint(800, 400)
   End
‘end the cam create block
Enable = 1                    ‘enable the motor
EncPosModulo = 800            ‘set EncPosModulo to master
 counts/cycle
PosModulo = 400
‘set PosModulo to slave (SC950) counts/cycle
EncPos = 0                    ‘clear the counter
ActiveCam = 1                 ‘activate cam #1
End
```

CWINH
(PRE-DEFINED VARIABLE)

Purpose CwInh indicates the current state of the CWINH (Inhibit +) Input. It can also be used as an interrupt source.

Syntax *x* = CwInh

Range 0 or 1

CWOT
(PRE-DEFINED VARIABLE)

Purpose Cwot sets the clockwise software over-travel limit. When the position of the motor becomes more positive than this limit, a clockwise over-travel interrupt occurs if the interrupt is active.

Syntax Cwot = *x*

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

Units resolver counts

Default 0

DECELGEAR
(PRE-DEFINED VARIABLE, INTEGER)

Purpose DecelGear sets the maximum deceleration commanded on the follower when Gearing is turned ON or the electronic gearing ratio (Ratio, PulsesOut, PulsesIn) is decreased. This maximum acceleration limit remains in effect until Gearlock is achieved. Once Gearlock is achieved, the follower follows the master with whatever acceleration or deceleration is required.

***DecelGear** is independent of **AccelGear**. Each variable must be set, independently, to the appropriate value for the desired motion.*

Syntax DecelGear = *x*

Units rpm/sec

Range 1 to 16,000,000 rpm/sec

Default 16,000,000 rpm/sec

Guidelines Set DecelGear prior to initiating gearing.

Related Instructions AccelGear, GearError, GearLock

DECELRATE
(PRE-DEFINED VARIABLE, INTEGER)

Purpose

DecelRate (deceleration rate) sets the maximum commanded deceleration rate when the speed is decreased.

***DecelRate** is independent of **AccelRate**. Each variable must be set independently to the appropriate value for the desired motion.*

Syntax

DecelRate = *x*

Units

rpm/sec

Range

1 to 16,000,000 rpm/sec

Default

10,000 rpm/sec

Guidelines

Set DecelRate prior to initiating the move. You can update DecelRate during a move by executing an UpdMove statement.

Related Instructions

AccelRate

Example

This example sets DecelRate to 5,000 rpm/sec and does an incremental move of 10 motor revolutions (assuming CountsPerRev is 4096).

```
RunSpeed = 1000
AccelRate = 10000
DecelRate = 5000
IndexDist = 40960
GoIncr
```

DIM
(STATEMENT)

Purpose Dim is used for declaring variables before use. All variables (except pre-defined variables) must be declared before they are used.

Dim also specifies that a global variable is non-volatile. When the controller is power-cycled non-volatile variables retain the value present when the controller was powered down. All other user variables are initialized to zero.

Syntax Dim *var1* [, *var2* [...]] as type [*NV*]

where type is:

INTEGER = 32 bit integer
FLOAT = IEEE single precision float
STRING = default length is 32 characters

Guidelines The default length for strings is overridden by following the STRING type designator with a * (see example). See the examples to use Dim to dimension an array.

Related Instructions Static

Example

```
Dim x, y, z as Integer NV       '3 non-volatile integers
Dim q as float                  '1 floating point
Dim Array1(4,5) as integer      'a 4x5 array
Dim A$ as String*50             'a 50 character string
```

DIR
(PRE-DEFINED VARIABLE, INTEGER)

Purpose Dir specifies the direction the motor turns when a GoVel statement is executed. It has no effect on any other motion statements. If **Dir = 0**, the motor turns in the positive direction. If **Dir = 1**, the motor turns in the negative direction.

Syntax Dir = *x*

Units none

Range 0 or 1

Default 0

Guidelines Positive and negative directions of motor motion are defined by the PosPolarity variable.

Related Instructions GoVel, PosPolarity

DM1F0
(PRE-DEFINED VARIABLE, INTEGER)

Purpose	DM1F0 sets the frequency in Hz of a single pole low-pass filter on the DAC Monitor 1 output (J4-3).
Syntax	DM1F0 = *x*
Units	Hertz
Range	0.01 to 4.17e7
Default	1000 Hertz
Guidelines	DM1F0 is used to attenuate high frequency components from the DM1Map selected signal. Setting DM1F0 to 1 Hz and using DM1Out to examine the filtered value is an easy way to accurately measure the selected signal's DC value.

DM1GAIN
(PRE-DEFINED VARIABLE, FLOAT)

Purpose Sets the multiplicative scale factor applied to the DM1Map selected signal before outputting on DAC Monitor 1 (J4-3).

Syntax DM1Gain = x

Default 0.6667

Guidelines Changing DM1Map changes DM1Gain's value unless DM1Map changes to a signal with identical units, such as VelCmdA to VelFB (DM1Map = 1 to 2). Set DM1Gain to keep the signal in the DAC Monitor in the ±5 volt range. Below lists units when DM1Gain = 1.

Monitor #	Scale Factor	Monitor #	Scale Factor
0	No Effect	15	1 V/cycle
1	1 V/krpm	16	1 V/amp
2	1 V/krpm	17	1 V/amp
3	1 V/krpm	18	1 V/amp
4	1 V/krpm	19	1 V/100%
5	1 V/rev	20	1 V/100%
6	1 V/rev	21	1 V/100%
7	1 V/rev	22	1 V/V
8	1 V/amp	23	1 V/rev
9	1 V/amp	24	1 V/amp
10	1 V/V	25	1 V/amp
11	1 V/Hz	26	1 V/100%
12	10 V/4096	27	1 V/100%
13	1 V/100%	28	1 V/krpm
14	1 V/ °C		

Related Instructions DM1Map, DM1F0, and DM1Out.

DM1MAP
(PRE-DEFINED VARIABLE, INTEGER)

Purpose DM1Map selects signal sent to the DAC Monitor 1 output on J4-3.

Syntax DM1Map = *x*

Range 0 to 65,537

Default 9 (IFB, Current Feedback)

Guidelines See Hardware manual for definitions of mnemonics.

Monitor #	Mnemonic	Monitor #	Mnemonic
0	AnalogOut1	16	IR
1	VelFB	17	IS
2	VelCmdA	18	IT
3	VelErr	19	VR
4	FVelErr	20	VS
5	Position*	21	VT
6	PosError*	22	VBus
7	PosCommand*	23	ResPos *
8	ICmd	24	Cmd Non-Torque Current
9	IFB	25	Non-Torque IFB
10	AnalogIn	26	Torque Voltage Duty Cycle
11	EncFreq	27	Non-Torque Voltage Duty Cycle
12	EncPos*	28	VelCmd
13	ItFilt	65536	Clamp Off **
14	HSTemp	65537	Clamp On **
15	Comm Ang *		

*Wraps around when the signal exceeds the output voltage level.
**The value of the selected signal does not change.

Related Instructions DM1Gain, M1F0, and DM1Out

DM1OUT
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose DM1Out indicates the value of the selected, filtered variable output to DAC Monitor 1 (J4-3). The value is reported in the units of the selected variable. For example, DM1Map = 1 selects VelCmdA and the units are rpm.

Syntax x = DM1Out

Range Depends on DM1Map selected signal.

Guidelines With DM1F0 set low (such as 1 Hz), DM1Out's value accurately measures the DM1Map selected signal's DC component.

DM1Out also examines variables that cannot be directly queried, such as motor phase voltage duty cycle, DM1Map = 19, 20 or 21.

DM2F0
(PRE-DEFINED VARIABLE, FLOAT)

Purpose DM2F0 sets the frequency in Hz of a single pole low-pass filter on the DAC Monitor 2 output (J4-4).

Syntax DM2F0 = x

Units Hertz

Range 0.01 to 4.17e7

Default 1000 Hertz

Guidelines DM2F0 is used to attenuate high frequency components from the DM2Map selected signal. Setting DM2F0 to 1 Hz and using DM2Out to examine the filtered value is an easy way to accurately measure the selected signal's DC value.

DM2GAIN
(PRE-DEFINED VARIABLE, FLOAT)

Purpose DM2Gain sets the multiplicative scale factor applied to the DM2Map selected signal before outputting on DAC Monitor 2 (J4-4).

Syntax DM2Gain = *x*

Default 2.0

Guidelines Changing DM2Map changes DM2Gain's value unless DM2Map changes to a signal with identical units, such as VelCmdA to VelFB (DM2Map = 1 to 2). Set DM2Gain to keep the signal in the DAC Monitor in the ±5 volt range. Below lists units when DM2Gain = 1.

Monitor #	Scale Factor	Monitor #	Scale Factor
0	No Effect	15	1 V/cycle
1	1 V/krpm	16	1 V/amp
2	1 V/krpm	17	1 V/amp
3	1 V/krpm	18	1 V/amp
4	1 V/krpm	19	1 V/100%
5	1 V/rev	20	1 V/100%
6	1 V/rev	21	1 V/100%
7	1 V/rev	22	1 V/V
8	1 V/amp	23	1 V/rev
9	1 V/amp	24	1 V/amp
10	1 V/V	25	1 V/amp
11	1 V/Hz	26	1 V/100%
12	10 V/4096	27	1 V/100%
13	1 V/100%	28	1 V/krpm
14	1 V/ °C		

Related Instructions DM2Map, DM2F0, DM2Out.

DM2MAP
(PRE-DEFINED VARIABLE, INTEGER)

Purpose DM2Map selects signal sent to the DAC Monitor 2 output on J4-3.

Syntax DM2Map = *x*

Range 0 to 65,537

Default 1 (VelFB, Velocity Feedback)

Guidelines See Hardware manual for definitions of mnemonics.

Monitor #	Mnemonic	Monitor #	Mnemonic
0	AnalogOut2	16	IR
1	VelFB	17	IS
2	VelCmdA	18	IT
3	VelErr	19	VR
4	FVelErr	20	VS
5	Position*	21	VT
6	PosError*	22	VBus
7	PosCommand*	23	ResPos *
8	ICmd	24	Cmd Non-Torque Current
9	IFB	25	Non-Torque IFB
10	AnalogIn	26	Torque Voltage Duty Cycle
11	EncFreq	27	Non-Torque Voltage Duty Cycle
12	EncPos*	28	VelCmd
13	ItFilt	65536	Clamp Off **
14	HSTemp	65537	Clamp On **
15	Comm Ang *		

*Wraps around when the signal exceeds the output voltage level.
**The value of the selected signal does not change.

Related Instructions DM2Gain, DM2F0, DM2Out

DM2OUT
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose DM2Out indicates the value of the selected, filtered variable output to DAC Monitor 2 (J4-4). The value is reported in the units of the selected variable. For example, DM2Map = 1 selects VelCmdA and the units are rpm.

Syntax x = DM2Out

Range Depends on DM2Map selected signal.

Guidelines With DM2F0 set low (1 Hz), DM1Out's value accurately measures the DM1Map selected signal's DC component.

DM2Out also examines variables that cannot be directly queried, such as motor phase voltage duty cycle, DM2Map = 19, 20 or 21.

ENABLE
(PRE-DEFINED VARIABLE, INTEGER)

Purpose Enable controls whether or not power can flow to the motor, (drive is enabled).

- **0** (disables the drive)
- **1** (enables the drive)

Syntax Enable = x

Units none

Range 0 or 1

Default 0

Guidelines Before power can flow to the motor, the following must all be true:

1. Drive is not faulted.
2. Enable* input (J4-6) is connected to I/O RTN.
3. Enable pre-defined variable is set to 1.

Related Instructions Enabled

ENABLED
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose Enabled indicates whether or not power can flow to the motor, (drive is enabled).

Syntax x = Enabled

Units none

Range 0 or 1

Default none

Guidelines Before power can flow to the motor, the following must all be true:

1. Drive is not faulted.
2. Enable* input (J4-6) is connected to I/O RTN.
3. Enable pre-defined variable is set to 1.

Related Instructions Enable

Example

```
If (Enabled = 1) Then
        print "Drive is Enabled!"
Else
        print "Drive is NOT Enabled"
End If
```

ENABLEPLS0
(PRE-DEFINED VARIABLE, INTEGER)

Purpose EnablePLS0 is a pre-defined variable for PLS0. It is used to enable or disable Out0.

Syntax EnablePLS0 = *x*

Range 0 or 1

Default 0

Guidelines Use EnablePLS0 = 1 to enable a Programmable Limit Switch. Use ConfigPLS( ) to configure the Programmable Limit Switch.

Related Instructions ConfigPLS( )

Example The statements below configure PLS0 so Out0 is set to 1 when Position is between 4096 and 4196. Out0 is set to 0 at all other times.

```
ConfigPLS(0, 4096, 100, 1)
EnablePLS0 = 1
```

ENABLEPLS1
(PRE-DEFINED VARIABLE, INTEGER)

Purpose EnablePLS1 is a pre-defined variable for PLS1. It is used to enable or disable Out1.

Syntax EnablePLS1 = *x*

Range 0 or 1

Default 0

Guidelines Use EnablePLS1 = 1 to enable a Programmable Limit Switch. Use ConfigPLS( ) to configure the Programmable Limit Switch.

Related Instructions ConfigPLS( )

Example The statements below configure PLS1 so Out1 is set to 1 when Position is between 4096 and 4196. Otherwise, set Out1 to 0.

```
ConfigPLS(0, 4096, 100, 1)
EnablePLS1 = 1
```

ENABLEPLS2
(PRE-DEFINED VARIABLE, INTEGER)

Purpose EnablePLS2 is a pre-defined variable for PLS2. It is used to enable or disable Out2.

Syntax EnablePLS2 = *x*

Range 0 or 1

Default 0

Guidelines Use EnablePLS2 = 1 to enable a Programmable Limit Switch. Use ConfigPLS( ) to configure the Programmable Limit Switch.

Related Instructions ConfigPLS( )

Example The statements below configure PLS2 so Out2 is set to 1 when Position is between 4096 and 4196. Otherwise, set Out2 to 0.

```
ConfigPLS(0, 4096, 100, 1)
EnablePLS2 = 1
```

ENABLEPLS3
(PRE-DEFINED VARIABLE, INTEGER)

Purpose EnablePLS3 is a pre-defined variable for PLS3. It is used to enable or disable Out3.

Syntax EnablePLS3 = *x*

Range 0 or 1

Default 0

Guidelines Use EnablePLS3 = 1 to enable a Programmable Limit Switch. Use ConfigPLS( ) to configure the Programmable Limit Switch.

Related Instructions ConfigPLS( )

Example The statements below configure PLS3 so Out3 is set to 1 when Position is between 4096 and 4196. Otherwise, set Out3 to 0.

```
ConfigPLS(0, 4096, 100, 1)
EnablePLS3 = 1
```

ENABLEPLS4
(PRE-DEFINED VARIABLE, INTEGER)

Purpose EnablePLS4 is a pre-defined variable for PLS4. It is used to enable or disable Out4.

Syntax EnablePLS4 = *x*

Range 0 or 1

Default 0

Guidelines Use EnablePLS4 = 1 to enable a Programmable Limit Switch. Use ConfigPLS( ) to configure the Programmable Limit Switch.

Related Instructions ConfigPLS( )

Example The statements below configure PLS4 so Out4 is set to 1 when Position is between 4096 and 4196. Otherwise, set Out4 to 0.

```
ConfigPLS(0, 4096, 100, 1)
EnablePLS4 = 1
```

ENABLEPLS5
(PRE-DEFINED VARIABLE, INTEGER)

Purpose EnablePLS5 is a pre-defined variable for PLS5. It is used to enable or disable Out5.

Syntax EnablePLS5 = *x*

Range 0 or 1

Default 0

Guidelines Use EnablePLS5 = 1 to enable a Programmable Limit Switch. Use ConfigPLS( ) to configure the Programmable Limit Switch.

Related Instructions ConfigPLS( )

Example The statements below configure PLS5 so Out5 is set to 1 when Position is between 4096 and 4196. Otherwise, set Out5 to 0.

```
ConfigPLS(0, 4096, 100, 1)
EnablePLS5 = 1
```

ENABLEPLS6
(PRE-DEFINED VARIABLE, INTEGER)

Purpose EnablePLS6 is a pre-defined variable for PLS6. It is used to enable or disable Out6.

Syntax EnablePLS2 = *x*

Range 0 or 1

Default 0

Guidelines Use EnablePLS6 = 1 to enable a Programmable Limit Switch. Use ConfigPLS( ) to configure the Programmable Limit Switch.

Related Instructions ConfigPLS( )

Example The statements below configure PLS6 so Out6 is set to 1 when Position is between 4096 and 4196. Otherwise, set Out6 to 0.

```
ConfigPLS(0, 4096, 100, 1)
EnablePLS6 = 1
```

ENABLEPLS7
(PRE-DEFINED VARIABLE, INTEGER)

Purpose EnablePLS7 is a pre-defined variable for PLS7. It is used to enable or disable Out7.

Syntax EnablePLS7 = *x*

Range 0 or 1

Default 0

Guidelines Use EnablePLS7 = 1 to enable a Programmable Limit Switch. Use ConfigPLS( ) to configure the Programmable Limit Switch.

Related Instructions ConfigPLS( )

Example The statements below configure PLS7 so Out7 is set to 1 when Position is between 4096 and 4196. Otherwise, set Out7 to 0.

```
ConfigPLS(0, 4096, 100, 1)
EnablePLS7 = 1
```

ENCFREQ
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose EncFreq (Encoder Frequency) is the frequency in quadrature pulses per second of the external encoder, (or steps per second if step-and-direction format is used).

Syntax x = EncFreq

Units Quadrature encoder counts per second (EncMode = 0)
Steps per second (EncMode = 1)

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

Guidelines Calculated from delta EncPos at position loop update rate. Although the values returned do not have fractional parts, this variable is communicated as a floating point quantity. See EncInF0 for recommended maximum count frequencies.

ENCIN
(PRE-DEFINED VARIABLE, INTEGER)

Purpose	EncIn specifies the line count of the encoder being used, (or one-fourth the steps/revolution if step-and-direction input format is used).
Syntax	EncIn = x
Units	Encoder line count (EncMode = 0) Steps per quarter-revolution (EncMode = 1)
Range	1 to 65535
Default	1024
Guidelines	EncIn ensures proper units in KPP, KVP, VelFB when using an encoder for servo feedback (RemoteFB = 1 or 2). EncIn is also used when using the encoder input port for electronic gearing and using the Ratio variable to specify the electronic gearing ratio.

ENCINF0
(PRE-DEFINED VARIABLE, FLOAT)

Purpose EncInF0 selects digital low pass filter frequency on the incremental encoder input connected to J4-21 through J4-24.

Syntax EncInF0 = x

Units Hertz

Range 4 values depending on EncMode:

EncInF0 (Hz)	Max Hardware Quad Count Limit (Hz)	Min Hardware Pulse Width (micro second)
1,600,000	3,333,333	0.6
800,000	952,400	2.1
400,000	476,200	4.2
200,000	238,100	8.4

EncInF0 (Hz)	Max Hardware Quad Count Limit (Hz)	Min Hardware Pulse Width (micro second)
800,000	833,333	0.6
200,000	238,000	2.1
100,000	119,000	4.2
50,000	59,500	8.4

Default 800,000

Guidelines EncInF0 is the maximum recommended count frequency for reliable operation. If the maximum input frequency is <EncInF0, lowering it gives better noise rejection.

The maximum hardware count limits require ideal timing with exact 50% duty cycle, perfect quadrature symmetry, etc. The recommended EncInF0 count takes real world signal tolerances into account. With the SC900's emulated encoder out wired to another SC900's encoder in, and EncInF0 = 1,600,000 Hz, the count frequency works reliably up to 2,000,000 Hz.

ENCMODE
(PRE-DEFINED VARIABLE, INTEGER)

Purpose EncMode specifies the type of digital command expected at the incremental position command port.

Syntax EncMode = x

Range 0, 1, 2, or 3

Default 0 (quadrature)

Guidelines EncMode replaces StepDir.

Value of EncMode	Description
0	Selects quadrature encoder pulses
1	Selects step and direction input signals
2	Selects up/down input signals
3	Ignores input signal, EncPos value held

ENCOUT
(PRE-DEFINED VARIABLE, INTEGER)

Purpose EncOut selects the resolution of the incremental shaft position output port (J4-14, J4-15, J4-16, J4-17, and J4-19, J4-20).

Syntax EncOut = x

Units Emulated encoder line count

Range 0, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 125, 250, 500, 1000, 2000, 4000, 8000, 16000

Default 1024

Guidelines EncOut = 0 cross-connects the Encoder input (J4-21, J4-22 and J4-23, J4-24) to the Encoder output to provide buffering. CH Z out (J4-19, J4-20) is held fixed for EncOut = 0.

ENCPOS
(PRE-DEFINED VARIABLE, INTEGER)

Purpose EncPos indicates the position of the external encoder. For example, with a 1024 line-count encoder, each increment of EncPos is equal to 1/4096 of a revolution of the encoder shaft.

If Encoder Position Modulo functionality is active (EncPosModulo □ 0), EncPos is automatically reset to zero every time it reaches the modulo value.

Syntax *x* = Encpos ***or*** Encpos = *x*

Units encoder counts

Range -2,147,483,648 to -2,147,483,648 or
0 to EncPosModulo-1

Default none

Guidelines EncPos is not affected by the value of EncIn. EncMode must be set to the appropriate value for the type of encoder input you are using.

Related Instructions EncIn, EncInF0, EncMode, EncPosModulo

ENCPOSMODULO
(PRE-DEFINED VARIABLE, INTEGER)

Purpose EncPosModulo specifies the encoder modulo value. The encoder modulo value is the value of EncPos where EncPos is automatically reset to zero.

Syntax EncPosModulo = x

Units encoder counts

Range 0 to 2,147,483,647

Default 0

Guidelines Setting EncPosModulo to 0 turns off the Encoder Position Modulo function and EncPos is never automatically reset (default).

Related Instructions EncPos, PosModulo

END
(STATEMENT)

Purpose End is used to mark the end of a program, a subroutine, a function, an If...Then...Else block, a Select Case block, an Interrupt service routine or a Params section.

Syntax End {[*Main*|*Sub*|*Function*|*If*|*Select*|*Interrupt*| *Params*]}

Guidelines Once the End statement is encountered the block structure is terminated.

Related Instructions Main, Sub, Function, Select Case, Interrupt, Params

ERR
(PRE-DEFINED VARIABLE)

Purpose Err indicates what caused the most recent Runtime Error. The table below shows what each value of Err means.

Value of Err	Error Caused by
1	Division by zero in arithmetic
2	Stack is full.
3 - 5	(not used)
6	Out of Memoryr
7 - 10	(not used)
11	Attempt to use Feature not available in this firmware
12	Internal Firmware Error
13	Invalid Predefined Variable ID Number
14	Attempt to write to a Read-Only Variable
15	DSP Read Error
16	DSP Write Error
17	DSP Command Error
18 - 21	(not used)
22	No Interrupt Handler defined
23	(not used)
24	PACLAN Transmit Error

Value of Err	Error Caused by
25	PACLAN Response Timeout
26	PACLAN Response Error
27	Interrupt Error
28	Maximum String Length Exceeded
29	String Overflow
30	Array Index Bounds Error
31	Invalid Axis in PACLAN Message
32	No LAN Interrupt Handler
33	LAN Interrupt Queue is full
34	LAN Interrupt is not available
35	LAN Interrupt: Destination is busy
36	ModBus: Attempt to do nested master functions
37	ModBus: Attempt to use master without setting RuntimeProtocol
38	ModBus: Illegal Slave Address (255)
39	AB DF1: Invalid PLC Address (0-255)
40	AB DF1: Invalid PLC File Number Specified
41	AB DF1: Invalid PLC Element Number Specified
42	AB DF1: too many unresolved messages outstanding
43	AB DF1: Attempt to use AB DF1 without setting RunTimeProtocol
44	AB DF1: Transmit queue overflow
45	$DeclareCam: Invalid Cam Number specified
46	$DeclareCam: Too many points specified.
47	CreateCam: Tried to create a new cam before finished creating the first one.
48	CreateCam: Tried to create cam without declaring it.
49	Addpoint: Tried to add more points than declared.
50	Addpoint: Starting Master position is non-zero.
51	AddPoint: Used AddPoint outside a CreateCam block.
52	CreateCam: EndList without Create

Value of Err	Error Caused by
53	CreateCam: Tried to create a cam with less than three points.
54	AddPoint: Used the same master position for two points or master position was negative
55	CreateCam: Tried to create the ActiveCam.
56	ActiveCam: Tried to activate a cam that was not created.
57	ActiveCam: Tried to activate a cam while it is being created.
58	ActiveCam: Tried to activate a cam while RunSpeed =0.
59	ActiveCam: Tried to activate a cam with master position outside the cam table.

Runtime errors are caused by the program running on the OC950 trying to do something that is not allowed. For example, runtime errors occur when you attempt to write a value that is too high or too low to a particular variable. We try to catch as many errors as possible when the program is compiled, but some errors are only detected when the program is running.

Determine the particular problem causing Runtime Error (F4 Fault) by looking at the value of the Err variable. Use the Variables Window to find the value of Err.

EXIT
(STATEMENT)

Purpose	Exit is used to exit from a subroutine, a function, an interrupt, a For...Next or a While...Wend.
Syntax	Exit {{*Sub*|*Function*|*Interrupt*|*For*|*While*}]
Guidelines	Do not confuse Exit with End. Exit causes program control to pass to the end of the block structure. End defines the end of the structure.
Related Instructions	Sub, Function, Interrupt, For...Next, While...Wend

EXP( )
(FUNCTION)

Purpose	Exp( ) returns e (the base of natural logarithms) raised to a power.
Syntax	result = Exp(x)
Guidelines	Exp( ) complements Log( ).
Related Instructions	Log( ), Log10( )

EXTFAULT
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE)

Purpose ExtFault provides additional information on fault codes blinking 1 (1) or E (14) and alternating F 3 (243). Otherwise, the value is 0.

Range 0 to 16

Guidelines In the variables window, poll the value of ExtFault for additional fault information.. Values listed below:

LED Display	Value of ExtFault	Description
1	1	|VelFB| < 21038
1	2	|VelFB| < 1.5*max(|VelLmtxx|)
E	0	No ExtFault information
E	1	Resolver calibration data corrupted
E	2	Excessive DC offset in current feedback sensor
E	3	DSP incompletely reset by line power dip
E	6	Excessive DC offset in Analog Command A/D
E	7	Unable to determine option card type
E	8	DSP stack overflow
E	10	Firmware and control card ASIC incompatible
E	11	Actual Model does not match value in non-volatile memory
E	12	Unable to determine power stage
E	13	Control card non-volatile parameters corrupt
E	14	Option card non-volatile parameters corrupt
F3	15	RAM failure
F3	16	Calibration RAM failure

FAULT
(PRE-DEFINED VARIABLE, INTEGER, MAPPABLE OUTPUT FUNCTION)

Purpose	Fault indicates whether the drive has faulted and is disabled.
Syntax	x = Fault
Range	0 or 1
Guidelines	**0** = not faulted, normal operation. **1** = faulted, no power flow to the motor.
Related Instructions	FaultCode, ExtFault

FAULTCODE
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ-ONLY)

Purpose FaultCode indicates a fault has occurred. When the status display is not a 0 or an 8, a fault has occurred. Reset the drive by asserting the fault reset signal or by cycling drive AC power.

Syntax x = FaultCode

Range 0 to 255

Guidelines 0 means the drive is not faulted and not enabled, while 8 means the drive is not faulted and enabled. Alternating 8 ⌐ means actively inhibiting CW motion and alternating 8 ¬ means actively inhibiting CCW motion.

Status LED	Value	Fault Meaning
(Blinking) ***1***	1	Velocity feedback (VelFB) over speed
(Blinking) ***2***	2	Motor Over-Temp
(Blinking) ***3***	3	Drive Over-Temp
(Blinking) ***4***	4	Drive I*t
(Blinking) ***5***	5	l-*n* Fault (9x3)
(Blinking) ***6***	6	Control ±12 V supply under voltage
(Blinking) ***7***	7	Output over current or bus over voltage
(Blinking) ***9***	9	Shunt regulator overload
(Blinking) ***A***	10	Bus OV detected by DSP
(Blinking) ***b***	11	Auxiliary +5V Low
(Blinking) ***C***	12	Not assigned
(Blinking) ***d***	13	Not assigned
(Solid) ***E****	14	Processor throughput fault
(Blinking) ***E****	14	Power Up Self Test Failure
(Alternating) ***E1***	225	Bus UV, Bus Voltage VBusThresh
(Alternating) ***E2***	226	Ambient Temp Too Low
(Alternating) ***E3***	227	Encoder commutation align failed (Only CommSrc=1)
(Alternating) ***E4***	228	Drive software incompatible with NV memory version
(Alternating) ***E5****	229	Control Card hardware not compatible with drive software version

Status LED	Value	Fault Meaning
(Alternating) ***E6***	230	Drive transition from unconfigured to configured while enabled
(Alternating) ***E7***	231	Two AInNull events too close together
(Alternating) ***F1***	241	Excessive Position Following Error
(Alternating) ***F3***	243	Parameter Checksum Error (Memory Error)
(Alternating) ***F4***		Run-time Error.

*FaultReset cannot reset these faults.

See ExtFault for further information on Blinking E, Blinking 1 and Alternating F3. See Err for Alternating F4.

FAULTRESET
(PRE-DEFINED VARIABLE, INTEGER, MAPPABLE INPUT FUNCTION)

Purpose FaultReset resets drive faults.

Syntax FaultReset = x

Range 0 or 1

Default 0 at power up if not mapped

Guidelines FaultReset active automatically disables the drive. When not mapped to a BDIO, setting FaultReset to 1 via the serial port resets the latched function.

If the fault persists when FaultReset is active, the drive remains faulted. If the Fault condition does not persist, setting FaultReset to 1 clears the latched fault and returning FaultReset to 0 resumes normal operation.

FIX( )
(FUNCTION)

Purpose Fix( ) returns the truncated integer part of *x*.

Syntax result = Fix(*x*)

Guidelines Fix( ) does not round off numbers, it simply eliminates the decimal point and all digits to the right of the decimal point.

Related Instructions Abs( ), Cint( ), Int( )

FOR...NEXT *(STATEMENT)*

Purpose

For...Next allows a series of statements to be executed in a loop a specified number of times.

Syntax

For loop_counter = Start_Value To End_Value [*Step increment*]

...*statements*...

Next

Guidelines

You can exit from a For...Next loop using the Exit For. If step increment is omitted then increment defaults to 1.

The loop_counter is floating point or integer.

The Step increment is positive or negative, integer or floating point.

Related Instructions

While...Wend, Exit

Example

```
Dim x as integer
For x = 1 to 100 Step 2
        Print x
'print 2 to 100 in 2's
Next

dim x as float
for x = 0.5 to 1.2 step 0.1
        print x
'print 0.5 to 1.2 in 0.1 increments
next
```

FUNCTION
(STATEMENT)

Purpose

Function declares and defines the name, arguments and type of a user defined function. The code for the function immediately follows the function statement and must be terminated by End Function.

Syntax

Function *function-name* [(*argument-list*)] as *function-type*
...*statements*...
End Function

Guidelines

On entry to the function, all local variables are initialized to zero including all elements of local arrays. All local string variables are initialized to the null string ("").

If a function takes no arguments then the argument-list (including the parentheses) must be omitted, both when declaring the function and when using the function.

The return value for the function is specified by making an assignment to the function name. See the example below.

Arguments, including array arguments, are passed by value. Arrays cannot be returned from functions.

Related Instructions

Dim, Static, End, Exit, Sub

Example

This example declares a function that calculates the cube of a floating point number.

```
Main
    dim LocalFloat as float
    LocalFloat = 1.234
    LocalFloat = cube(LocalFloat)
    print LocalFloat
End Main

Function cube(x as float) as float
    cube = x^3
End Function
```

FVELERR
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose	FVelErr is commanded velocity - measured velocity (VelCmdA - VelFB) after being processed by the velocity loop compensation anti-resonant filter section.
Syntax	x = FVelErr
Units	rpm
Range	-48,000 to +48,000
Related Instructions	ARF0, ARF1, ARZ0, ARZ1

FWV
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ-ONLY)

Purpose	FwV indicates the 950 firmware version number. For example, FwV = 1100 is version 1.1.
Syntax	x = FwV
Range	1000 to 65535

GEARERROR
(PRE-DEFINED VARIABLE, INTEGER)

Purpose GearError indicates the amount of position deviation that has accumulated on the slave axis (in an electronic gearing application) as a result of the slave axis limiting its acceleration or deceleration while achieving velocity synchronization.

Syntax x = GearError

Units resolver counts

Guidelines GearError is never automatically set to zero. It accumulates position deviation each time acceleration limiting is activated. Typically, set GearError to zero before doing something that activates acceleration limiting.

The slave axis' acceleration or deceleration is limited to AccelGear or DecelGear whenever:

1. Gearing is turned on or turned off.
2. Ratio is changed.
3. PulsesIn or PulsesOut is changed.

Related Instructions AccelGear, DecelGear, GearLock

Example

```
AccelGear = 10000
PulsesIn  = 1
PulsesOut = 1
GearError = 0
Gearing = 1
While GearLock = 0 : Wend
'catch up the position lost while acceleration was being limited
IndexDist = GearError
GoIncr
```

GEARING
(PRE-DEFINED VARIABLE, INTEGER)

Purpose Gearing controls the electronic gearing functionality. Turns electronic gearing on or off and sets the allowed direction of motion for electronic gearing.

Value	Description
0	Off. No electronic gearing.
1	On. Motor motion allowed in either direction/
2	On. Motor motion allowed only in the positive direction.
3	On. Motor motion allowed only in the negative direction.

Syntax Gearing = x

Units none

Range 0, 1, 2, 3

Default 0

Guidelines Moving does not recognize motor motion caused by electronic gearing.

When unidirectional gearing is used (Gearing = 2 or 3), motion in the allowed direction occurs only when the master encoder returns to the point at which it originally reversed direction. Other motion commands (GoVel or GoIncr) cause motor motion in the disabled gearing direction.

Other motion commands (GoVel or GoIncr), may be executed while gearing is active. These moves are superimposed (added to) on the electronic gearing motion.

Related Instructions PulsesIn, PulsesOut, EncIn

GEARLOCK
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose GearLock indicates when the slave axis (follower axis) in an electronic gearing application has achieved velocity synchronization with the electronic gearing master. GearError contains the amount of position deviation accumulated while the slave axis was limiting its acceleration or deceleration.

Syntax *x* = GearLock

where:

x = 0 indicates that the slave has not achieved velocity synchronization.

x = 1 indicates that the slave has achieved velocity synchronization.

Range 0 or 1

Guidelines The slave axis' acceleration or deceleration is limited to AccelGear or DecelGear whenever:

1. Gearing is turned on or turned off.
2. Ratio is changed.
3. PulsesIn or PulsesOut is changed.

Related Instructions AccelGear, DecelGear, GearError

Example

```
AccelGear = 10000
PulsesIn  = 1
PulsesOut = 1
GearError = 0
Gearing = 1
While GearLock = 0 : Wend
'catch up the position lost while acceleration was being limited
IndexDist = GearError
GoIncr
```

GETMOTOR$( )
(FUNCTION)

Purpose	GetMotor$( ) returns a string indicating the motor name specified with the last SetMotor( ) function.
Syntax	A$ = GetMotor$
Guidelines	GetMotor$( ) returns the motor name in upper-case, even if you specified the name with lower-case letters.
Related Instructions	SetMotor( )

GOABS
(STATEMENT)

Purpose	GoAbs (Go to Absolute Position) causes the motor to move to the position specified by TargetPos. This is an absolute position referenced to the position where PosCommand = 0.
Syntax	GoAbs
Guidelines	Program execution continues with the line immediately following the GoAbs statement as soon as the move is initiated. Program execution does not wait until the move is complete.
Related Instructions	AbortMotion, GoHome, GoIncr, GoVel

GOABSDIR
(PRE-DEFINED VARIABLE, INTEGER)

Purpose GoAbsDir determines the direction of rotation when PosModulo (or EncposModulo) is used and an absolute move (GoAbs) is commanded.

GoAbsDir	Direction
0	Clockwise (CW)
1	Counter-Clockwise (CCW)
2	Shortest Distance (CW or CCW)
3	None

Syntax GoAbsDir = x

Units none

Range 0, 1, 2, 3

Default 3

Guidelines Set GoAbsDir before GoAbs.

Example The following program illustrates GoAbsDir. Assume Position = 550.

```
Enable = 1
PosModulo = 1000
AccelRate = 1000
DecelRate = 1000
RunSpeed = 5000
TargetPos = 850
GoAbsDir = 0
GoAbs          'The motor travels CW 300 counts.
GoAbsDir = 1
GoAbs          'The motor travels CCW 700 counts
GoAbsDir = 2
GoAbs          'The motor travels 300 counts CW
GoAbsDir = 3
GoAbs          'The motor travels CW 300 counts
```

GOHOME (STATEMENT)

Purpose

GoHome causes the motor to move to the position specified where PosCommand = 0. GoHome is identical to GoAbs with TargetPos = 0.

The motor speed follows a velocity profile as specified by AccelRate, DecelRate, and RunSpeed. This profile may be modified during the move using UpdMove.

Syntax

GoHome

Guidelines

Program execution continues with the line immediately following the GoHome statement as soon as the move is initiated. Program execution does not wait until the move is complete.

The drive must be enabled in order for any motion to take place.

Related Instructions

AbortMotion, GoAbs, GoIncr, GoVel

GOINCR (STATEMENT)

Purpose

GoIncr (Go Incremental) causes the motor to move a distance specified by IndexDist.

The motor speed follows a velocity profile as specified by AccelRate, DecelRate, and RunSpeed. This profile may be modified during the move using UpdMove.

Syntax

GoIncr

Guidelines

Program execution continues with the line immediately following the GoIncr statement as soon as the move is initiated. Program execution does not wait until the move is complete.

The drive must be enabled in order for any motion to take place.

Related Instructions

AbortMotion, GoAbs, GoHome, GoVel

GOTO
(STATEMENT)

Purpose GoTo causes the software to jump to the specified label and continue executing from there.

Syntax Goto *Label*

Guidelines GOTO is NOT RECOMMENDED as a looping technique. Excessive use of GOTO statements lead to disorganized and confusing programs. Preferred looping techniques are:

For...Next
If...Then...Else
While...Wend

Related Instructions On Error Goto

GOVEL
(STATEMENT)

Purpose GoVel (Go at Velocity) moves the motor at a constant speed specified by RunSpeed and direction specified by Dir.

The motor speed follows a velocity profile as specified by AccelRate, DecelRate, and RunSpeed. This profile may be modified during the move using UpdMove.

Syntax GoVel

Guidelines Program execution continues with the line immediately following GoVel as soon as the move is initiated. Program execution does not wait until the move is complete.

The drive must be enabled in order for any motion to take place.

Related Instructions AbortMotion, GoAbs, GoHome, GoIncr

HEX$( )
(FUNCTION)

Purpose	Hex$( ) converts an integer number to its equivalent hexadecimal ASCII string.
Syntax	result$ = Hex$(x)
Guidelines	Hexadecimal numbers are numbers to the base 16 (rather than base 10). The argument to Hex$( ) is rounded to an integer before Hex$(x) is evaluated.
Related Instructions	Oct$( ), Str$( )
Example	dim x,y as integer dim result1$, result2$ as string x = 20 y = &H6A result1$ = Hex$(x) result2$ = Hex$(y) print result1$, result2$ Prints: 14 6A

HSTEMP
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose	HSTemp indicates the drive heatsink temperature.
Syntax	x = HSTemp
Units	Degrees Centigrade
Range	-10 to +150
Guidelines	The drive heat sink temperature is monitored to determine if the drive is within a safe operating region for the power electronics. This variable is used to see how much thermal margin remains for a given application.
Related Instructions	ItThresh

HwV
(Pre-defined Variable, Integer, Status Variable, Read-Only)

Purpose HwV indicates the drive's control electronics hardware version number.
Syntax x = HwV
Range Greater than 0
Guidelines 12 = first production control card version

ICmd
(Pre-defined Variable, Float, Status Variable, Read-Only)

Purpose ICmd indicates the commanded motor torque current. ***ILmtMinus** and **ILmtPlus** limit the range of this variable.*
Syntax x = ICmd
Units Amperes
Range - Ipeak to + Ipeak

IFB
(Pre-defined Variable, Status Variable, Read-Only)

Purpose IFB indicates the measured motor torque current value.
Syntax x = IFB
Units Amperes
Range - Ipeak to + Ipeak
Guidelines IFB can be monitored to observe the actual torque current flowing in the motor. IFB should equal ICmd.

IF...THEN...ELSE
(STATEMENT)

Purpose If...Then...Else controls program execution based on the evaluation of numeric or string expressions

Syntax
IF *condition1* THEN
    ...statement block1...
[ELSEIF *condition2* THEN
    ...statement block2...]
[ELSE
    ...statement block3...]
END IF

Guidelines If *condition1* is True, *statement block1* is executed. If *condition2* is True, *statement block2* is executed. If the original *IF* condition is False and all *ELSEIF* conditions are False, the *ELSE* statement block (*statement block3*) is executed.

Related Instructions Select Case, While...Wend, Exit

ILMTMINUS
(PRE-DEFINED VARIABLE, INTEGER, NV PARAMETER)

Purpose ILmtMinus (Counter-Clockwise Current Limit) sets the maximum allowable torque current amplitude in the counter-clockwise direction. This is a percentage of the drive's peak current rating (I_{PEAK}).

Syntax ILmtMinus = x

Units % (Percentage) of peak current rating of drive.

Range 0 to 100

Default Parameter value specified in the Params...End Params section of your program. The 950 IDE New Program function calculates this value based upon the specified motor and drive.

Guidelines Only integer values may be entered (no fractions).

If ILmtMinus*0.01*I_{PEAK} > twice the motor's continuous current rating, the motor's over temperature sensor is not guaranteed to always respond fast enough to prevent motor winding damage.

ILMTPLUS
(PRE-DEFINED VARIABLE, INTEGER, NV PARAMETER)

Purpose ILmtPlus (Clockwise Current Limit) sets the maximum allowable torque current amplitude in the clockwise direction. This is a percentage of the drive's peak current rating (I_{PEAK}).

Syntax ILmtPlus = x

Units % (Percentage) of peak current rating of drive.

Range 0 to 100

Default Parameter value specified in the Params...End Params section of your program. The 950 IDE New Program function calculates this value based upon the specified motor and drive.

Guidelines Only integer values may be entered (no fractions).

If ILmtPlus*0.01*I_{PEAK} twice the motor's continuous current rating, the motor's over temperature sensor is not guaranteed to always respond fast enough to prevent motor winding damage.

INDEXDIST
(PRE-DEFINED VARIABLE, INTEGER)

Purpose IndexDist specifies the distance the motor turns during an incremental move (GoIncr).

Syntax IndexDist = x

Units resolver counts

Default 4096

Guidelines Specify IndexDist before initiating GoIncr.

Related Instructions AccelRate, DecelRate, RunSpeed, GoIncr

Example This example sets IndexDist to 40,960 (10 motor revolutions, assuming CountsPerRev is 4096) and does an incremental move.

```
RunSpeed = 1000
AccelRate = 10000
DecelRate = 5000
IndexDist = 40960
GoIncr
```

INKEY$
(STRING FUNCTION)

Purpose Inkey$ returns a 1-character string corresponding to the character in the serial port receive buffer. If there is no character waiting, Inkey$ is a Null string (“”). If several characters are pending, only the first one is returned.

Syntax x$ = Inkey$

Guidelines Assigning a string from Inkey$ removes the character from the serial port’s receive buffer.

Related Instructions Character Interrupt

Example The following program lines removes all characters from the receive buffer and puts them into A$.

```
new$ = Inkey$
while new$  “”
    A$ = A$ + new$
    new$ = Inkey$
wend
```

INP0-INP20
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose Inp0-Inp20 reports the value of one of the discrete digital inputs on the OC950.
0 - indicates a logic low level
1 - indicates a logic high level

Syntax x = Inpn

Units none

Range 0 or 1

Default none

Guidelines Each of the 21 inputs can be used to trigger an interrupt on either or both its high-to-low and/or low-to-high transition(s).

Related Instructions Inputs

Example

```
Wait for Inp0=0 and Inp1=1 before starting...
While (Inp0 = 1) OR (Inp1 = 0) : Wend
Print “Starting”
```

INPOSITION
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose	InPosition indicates whether or not the motor has achieved commanded position. InPosition is useful to monitor move commands to ensure that the desired motion has been completed. InPosition is always 0 (False) or 1 (True).
Syntax	x = InPosition
Units	none
Range	0 or 1
Default	none
Guidelines	InPosition is 1 (True) only if all the following are true: - Moving = 0 - Position Error less than InPosLimit
Related Instructions	InPosLimit, Moving

INPOSLIMIT
(PRE-DEFINED VARIABLE)

Purpose	InPosLimit specifies the tolerance of Position Error (PosError) within which the InPosition flag is set to 1 (True).
Syntax	InPosLimit = x
Units	resolver counts
Default	5
Guidelines	Set InPosLimit before using InPosition.
Related Instructions	InPosition

INPUT
(STATEMENT)

Purpose	The Input statement reads a character string received from the serial port, terminated by a carriage-return.
Syntax	Input [*prompt-string*] [, | ;] *input-variable*
Guidelines	The input variable can be integer, floating-point or a string. As an option, the prompt-string is transmitted when the Input statement is encountered. This prompt-string is either a string constant or string variable. If the prompt-string is followed by a semi-colon, a question mark is printed at the end of the prompt-string. If the prompt-string is followed by a comma, no question mark is printed.
Related Instructions	Inkey$
Example	dim YourName$ as string input “What’s your name”; YourName$ print “Hello ”;YourName$;", I’m leaving..."

INPUTS
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose	Inputs reports the status of the 21 bi-directional I/O points on the OC950 as a parallel word. For each bit in Inputs: 0 - corresponds to a low logic level 1 - corresponds to a high logic level
Syntax	x = Inputs
Units	none
Range	0 - 21,757,952
Default	none
Guidelines	Use Inp0 through Inp20 to look at inputs individually.
Related Instructions	Inpn, BDInputs, Outputs, BDOutputs

INSTR( )
(FUNCTION)

Purpose Instr( ) returns the starting location of a substring within a string.

Syntax result = Instr([*n*], *x$*, *y$*)
x$= string
y$ = substring
n optionally sets the start of the search

Guidelines *n* must be in the range 1 to 255
Instr( ) returns 0 if:
n Len(*x$*)
y$* cannot be found in *x$
If *y$* is null (empty, “”), Instr( ) returns n)

Related Instructions Len( )

INT( )
(FUNCTION)

Purpose Int( ) (convert to largest integer) truncates an expression to a whole number.

Syntax result = Int(*x*)

Guidelines Int( ) behaves the same as Fix( ) for positive numbers. They behave differently for negative numbers.

Related Instructions Cint( ), Fix( )

Example
```
Print Int(12.34)      ‘ prints the value 12
Print Int(-12.34)     ‘ prints the value -13
```

INTERRUPT...END INTERRUPT (STATEMENT)

Purpose

The Interrupt statement marks the beginning of an Interrupt Service Routine. The Interrupt Service Routine is defined by a program structure resembling a subroutine. The interrupt feature permits execution of a user-defined subroutine upon receipt of a hardware interrupt signal or a pre-defined interrupt event.

Syntax

Interrupt {*Interrupt-Source-Name*}
...*program statements*...
End Interrupt

Guidelines

Interrupts are triggered by pre-defined events or external hardware sources. The *interrupt-source-name* and interrupt enable flag are unique for each interrupt source.

Receiving an interrupt suspends program execution and the interrupt service routine is executed. Program execution resumes at the point at which it was interrupted.

Interrupts are enabled (or disabled) by setting (or clearing) the associated interrupt enable flag. Interrupts are disabled until explicitly enabled. After an interrupt is triggered it is automatically disabled until it is enabled again.

Related Instructions

Intr{*source*}, Sub...Endsub, Restart

Example

```
main
    Time = 0
    IntrI0Lo = 1
    while 1
        pause(0.5)
        Out0=0 : Pause(0.005) : Out0=1
    wend
end main

Interrupt I0Lo
    print "I'm awake"
    If Time > 10 then
        print "OK. That's it."
    else
        IntrI0Lo = 1
    end if
End Interrupt
```

INTR{*SOURCE*}
(PRE-DEFINED VARIABLE, INTEGER)

Purpose Intr{*source*} is used to enable or disable interrupts from the specified source. If you enable a given interrupt then there must be an Interrupt Service Routine for that interrupt source in your program.

Syntax Intr{*source*} = x

Units none

Range 0 (disabled) or 1 (enabled)

Default 0 (disabled)

Guidelines

IntrCcwinh	when CCWInh goes True.
IntrCcwot	when Position < CcwOt.
IntrCwinh	when CWInh goes True.
IntrChar	when a character is received.
IntrCwot	when Position > CwOt.
IntrDisable	when the drive gets disabled.
IntrFault	when the drive faults.
IntrI0Hi	when Inp0 goes from 0 to 1
IntrI0Lo	when Inp0 goes from 1 to 0
IntrI1Hi	when Inp1 goes from 0 to 1
IntrI1Lo	when Inp1 goes from 1 to 0
IntrI2Hi	when Inp2 goes from 0 to 1
IntrI2Lo	when Inp2 goes from 1 to 0
IntrI3Hi	when Inp3 goes from 0 to 1
IntrI3Lo	when Inp3 goes from 1 to 0
IntrI4Hi	when Inp4 goes from 0 to 1
IntrI4Lo	when Inp4 goes from 1 to 0
IntrI5Hi	when Inp5 goes from 0 to 1
IntrI5Lo	when Inp5 goes from 1 to 0
IntrI6Hi	when Inp6 goes from 0 to 1
IntrI6Lo	when Inp6 goes from 1 to 0
IntrI7Hi	when Inp7 goes from 0 to 1
IntrI7Lo	when Inp7 goes from 1 to 0
IntrI8Hi	when Inp8 goes from 0 to 1
IntrI8Lo	when Inp8 goes from 1 to 0
IntrI9Hi	when Inp9 goes from 0 to 1
IntrI9Lo	when Inp9 goes from 1 to 0
IntrI10Hi	when Inp10 goes from 0 to 1

IntrI10Lo	when Inp10 goes from 1 to 0
IntrI11Hi	when Inp11 goes from 0 to 1
IntrI11Lo	when Inp11 goes from 1 to 0
IntrI12Hi	when Inp12 goes from 0 to 1
IntrI12Lo	when Inp12 goes from 1 to 0
IntrI13Hi	when Inp13 goes from 0 to 1
IntrI13Lo	when Inp13 goes from 1 to 0
IntrI14Hi	when Inp14 goes from 0 to 1
IntrI14Lo	when Inp14 goes from 1 to 0
IntrI15Hi	when Inp15 goes from 0 to 1
IntrI15Lo	when Inp15 goes from 1 to 0
IntrI16Hi	when Inp16 goes from 0 to 1
IntrI16Lo	when Inp16 goes from 1 to 0
IntrI17Hi	when Inp17 goes from 0 to 1
IntrI17Lo	when Inp17 goes from 1 to 0
IntrI18Hi	when Inp18 goes from 0 to 1
IntrI18Lo	when Inp18 goes from 1 to 0
IntrI19Hi	when Inp19 goes from 0 to 1
IntrI19Lo	when Inp19 goes from 1 to 0
IntrI20Hi	when Inp20 goes from 0 to 1
IntrI20Lo	when Inp20 goes from 1 to 0
IntrPACLAN	when a PACLAN interrupt is received.
IntrPosError	When a Position Error Fault would have occurred.

Related Instructions Interrupt...End Interrupt

Example

```
IntrI0Lo = 1
    while 1
        pause(0.5)
        Out0 = 0
        pause(0.005)
'toggle I/O point 0
        Out0 = 1
    wend
End Main
Interrupt I0Lo
    print "Interrupt"
    IntrI0Lo = 1
End Interrupt
```

I_{PEAK}
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose I_{PEAK} is the drive's maximum 0-peak current rating.
Syntax $x = I_{PEAK}$
Units Amperes
Range single value (see Default)
Default

Model Number	I_{PEAK}
952	7.5
953	15.0
954	30.0
955	60.0

ITF0
(PRE-DEFINED VARIABLE, FLOAT)

Purpose ItF0 specifies the corner frequency of the low-pass filters implementing the I*t drive thermal protection circuit.
Syntax ItF0 = x
Units Hertz
Range Lower limit set by Model
Upper limit > 10
Default 0.02 Hertz
Guideline ItF0 with ItThresh specifies the thermal protection circuit for the drive. ItF0 is the corner frequency of a low-pass filter, which processes an estimate of the drive's power dissipation. Increasing ItF0 makes the response more sensitive to over-current conditions.

The minimum frequency for ItF0 (slowest to fault) is limited to protect the drive's power electronics.

ITFILT
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose ItFilt is the drive's output current amplitude low pass filtered by ItF0 and normalized by I_{PEAK} to a percentage. ItFilt is the input to the drive's I*t thermal protection fault.

Syntax x = ItFilt

Units % (percentage) of drive peak current (I_{PEAK}).

Range 0 to 100

Guidelines ItFilt provides a means of evaluating the I*t protection circuit. When ItFilt exceeds the threshold specified by ItThreshA, the drive faults with Faultcode 4.

ItFilt = ItF0 low pass filter of (|IR| + |IS| + |IT|)*(50/I_{PEAK})

ITTHRESH
(PRE-DEFINED VARIABLE, INTEGER, NV PARAMETER)

Purpose ItThresh sets the maximum continuous output current, as a percentage of I_{PEAK}, before the I*t thermal protection faults the drive.

Syntax ItThresh = x

Units % (percentage) of drive peak current

Range 0 to 100 (actual upper limit is model-dependent)

Default Parameter value specified in the Params...End Params section of your program. The 950 IDE New Program function calculates this value based upon the specified motor and drive.

Guidelines ItThresh with ItF0 specifies the thermal protection fault for the drive. The actual I*t fault threshold may be lowered if the heat sink temperature (HSTemp) gets too high.

The maximum value for ItThresh is limited to protect the drive's power electronics.

Related Instructions ItThreshA

ITTHRESHA
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose ItThreshA is the maximum continuous output current, as a percentage of I_{PEAK}, trip level for the I*t thermal protection fault.

Syntax x = ItThreshA

Units percent

Range 0 to 100

Default none

Guidelines ItThresh, sets the desired value for ItThreshA and the two are equal for lower heat sink temperatures (HsTemps). At higher HSTemps, ItThreshA is lowered to protect the power stage. When ItFilt exceeds ItThreshA, the drive I*t faults. When doing a worst-case motion profile, examining ItThreshA, ItFilt, and HSTemp indicate how much drive thermal margin remains.

I_R
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose I_R is the measured current flowing in Motor Phase R, J2-4.

Syntax x = I_R

Units Amps

I_S
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose I_S is the measured current flowing in Motor Phase S, J2-3.

Syntax x = I_S

Units Amps

I_T
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose	I_T is the measured current flowing in Motor Phase T, J2-2.
Syntax	x = I_T
Units	Amps

KII
(PRE-DEFINED VARIABLE, FLOAT)

Purpose	Kii sets the integral gain of the current loops.
Syntax	Kii = x
Units	Hertz
Range	0 to 2546
Default	50 Hertz
Guidelines	Kii is the current loop's integral gain. It defines the frequency where the current loop compensation transitions from predominantly integral characteristics (gain decreasing with frequency) to predominantly proportional characteristics (constant gain with frequency). This value should typically be less than 10% of the current loop's bandwidth.
Related Instructions	Kip

KIP
(PRE-DEFINED VARIABLE, FLOAT, NV PARAMETER)

Purpose	Kip sets the proportional gain of the current loop.
Syntax	Kip = x
Units	Volts/Ampere
Range	0 to 2.15e5/I_{PEAK}
Default	Parameter value specified in the Params...End Params section of your program. The 950 IDE New Program function calculates this value based upon the specified motor and drive.
Guidelines	Current loop bandwidth in rad/sec is Kip/*L*, where *L* is the motor's line-to-line inductance (in henries). Recommended bandwidth is 2ð * 1000 rad/sec. Maximum bandwidth is 2ð * 1500 rad/sec.

KPP
(PRE-DEFINED VARIABLE, FLOAT, NV PARAMETER)

Purpose Kpp sets the proportional gain of the position loop.

Syntax Kpp = x

Units Hertz

Range 0.0 to 159.4

Default Parameter value specified in the Params...End Params section of your program. The 950 IDE New Program function calculates this value based upon the specified motor and drive.

Guidelines Kpp is defined by the following relationship:

KVFF
(PRE-DEFINED VARIABLE, FLOAT)

Purpose Kvff sets the proportion of velocity feed-forward signal added to the velocity command from differentiated position command.

Syntax Kvff = x

Units % (Percentage)

Range 0 to 199.9

Default 0 %

Guidelines Kvff is functional only for positioning modes (BlkType = 2).

When Kvff = 0, the net velocity command in positioning mode results entirely from PosError. There is a static nonzero PosError when commanding a constant shaft speed, know as the following error. Velocity feed forward adds a term to VelCMd proportional to delta PosCommand at the position loop update rate, which decreases following error.

Increasing Kvff reduces steady state following error and gives faster response time. However, if Kvff is too large, it causes overshoot. Typically, Kvff should not be set larger than 80% for smooth dynamics and acceptable overshoot, but should be set to 100% for minimum following error (necessary in electronic gearing applications).

KVI
(PRE-DEFINED VARIABLE, FLOAT, NV PARAMETER)

Purpose	Kvi sets the integral gain of the velocity loop.
Syntax	Kvi = x
Units	Hertz
Range	0.0 to 636.6
Default	Parameter value specified in the Params...End Params section of your program. The 950 IDE New Program function calculates this value based upon the specified motor and drive.
Guidelines	Kvi is the velocity loop integral gain. It defines the frequency where the velocity loop compensation transitions from predominantly integral characteristics (gain decreasing with frequency) to predominantly proportional characteristics (constant gain with frequency). This value should typically be less than 10% of the velocity loop bandwidth.
Related Instructions	Kvp

KVP
(PRE-DEFINED VARIABLE, FLOAT, NV PARAMETER)

Purpose	Kvp sets the proportional gain of the velocity loop.
Syntax	Kvp = x
Units	Amps/(Radians/Second)
Range	0 to Ipeak*12.6
Default	Parameter value specified in the Params...End Params section of your program. The 950 IDE New Program function calculates this value based upon the specified motor and drive.
Guidelines	Kvp is defined by the following relationship: where commanded motor current has units of (amperes) and Velocity Error has units of (radians/second). Kvp must be adjusted for total load inertia and motor torque constant.  ***Idealized velocity loop bandwidth (rad/sec) = Kvp*(Kt/J(lb-in-sec^2))(rad/sec^2/amp)*** ***Maximum recommended bandwidth = 2ð * 400 rad/sec.***

LANFLT( )
(PRE-DEFINED ARRAY VARIABLE, FLOAT)

Purpose LANFlt( ) is an array of 32 floating-point variables globally accessible over PACLAN. Each OC950 has its own LANFlt( ) array.

Syntax LANFlt(*n*)[*y*] = *z* *or*,
z = LANFlt(*n*)[*y*]
where (*n)* is the array index (1-32) and [*y*] is the axis address of the OC950 whose LANFlt array being used.

Units none

Default 0.0 for all entries

Guidelines Omit the [*axis #*] designation when reading or writing your own LANFlt(*n*) variables.

Related Instructions LANInt( )

LANINT( )
(PRE-DEFINED ARRAY VARIABLE, INTEGER)

Purpose LANInt( ) is an array of 32 integer variables globally accessible over PACLAN. Each OC950 has its own LANInt( ) array.

Syntax LANInt(*n*)[*y*] = *z* *or*,
z = LANInt(*n*)[*y*]
where (*n*) is the array index (1-32) and [*y*] is the axis address of the OC950 whose LANInt array being used.

Default 0 for all entries

Guidelines Omit the [*axis #*] designation when reading or writing your own LANFlt(*n*) variables.

Related Instructions LANFlt

LANINTERRUPT[ ] *(STATEMENT)*

Purpose LANInterrupt[*n*] invokes the PACLAN interrupt on axis [*n*].

Syntax LANInterrupt[*n*]

where [*n*] identifies the address of the interrupt's destination.

Guidelines Before issuing this statement, ensure that the destination axis is connected to the PACLAN and is running a program. Otherwise, a runtime error is generated on the source axis.

Related Instructions LANIntrSource, Interrupt, SendLANInterrupt( )[ ]

LANINTRARG *(PRE-DEFINED ARRAY VARIABLE, INTEGER)*

Purpose LANIntrArg contains an integer value specified by the source axis of the PACLAN interrupt when that axis invokes a PACLAN interrupt. LANIntrArg is used in the PACLAN interrupt handler for any purpose.

Syntax x = LANIntrArg

Default 0

Related Instructions LANIntrSource, SendLANInterrupt( )[ ]

LANINTRSOURCE *(PRE-DEFINED VARIABLE, INTEGER)*

Purpose LANIntrSource indicates the axis address of the source of a PACLAN interrupt.

Syntax x = LANIntrSource

Range 1 - 255

Default none

Guidelines LANIntrSource is set automatically by the firmware when it processes and dispatches a PACLAN interrupt. You can use it in your PACLAN interrupt handler to do different things, depending on the interrupt sent.

Related Instructions LANIntrArg, SendLANInterrupt()[ ]

LCASE$( )
(FUNCTION)

Purpose Lcase$( ) converts a string expression to lowercase characters.

Syntax result$ = Lcase$(*string-expression*)

Guidelines Lcase$( ) affects only letters in the string expression. Other characters (numbers) are unchanged.

Related Instructions Ucase$( )

Example
```
dim x$ as string
x$ = “U.S.A”
print Lcase$(x$)
‘prints:  u.s.a
```

LEFT$( )
(FUNCTION)

Purpose Left$ () returns a string of the *n* leftmost characters in a string expression.

Syntax result$ = Left$(x$,*n*)

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

Related Instructions Len( ), Mid$( ), Right$( )

Example
```
a$ = “Mississippi”
print Left$(a$, 5)          ‘prints: Missi
```

LEN( )
(FUNCTION)

Purpose Len( ) returns the number of characters in a string expression.

Syntax result = Len(x$)

Guidelines Non-printing characters and blanks are included.

Example
```
x$ = “New York, New York”
Print Len( x$ )          ‘prints: 18
```

LOG( )
(FUNCTION)

Purpose	Log( ) returns the natural logarithm of a numeric expression.
Syntax	result = Log(x)
Guidelines	*x* must be greater than 0.
Related Instructions	Exp( ), Log10( )
Example	Print Log(45.0 / 7.0) ‘prints: 1.860752 Print Log(1) ‘prints: 0

LOG10( )
(FUNCTION)

Purpose	Log10( ) returns the base 10 logarithm of a numeric expression.
Syntax	result = Log10(x)
Guidelines	*x* must be greater than 0.
Related Instructions	Exp( ), Log( )
Example	Print Log10(100) ‘prints: 2 Print Log10(1) ‘prints: 0

LTRIM$( )
(FUNCTION)

Purpose	Ltrim$( ) returns a copy of the original string with leading blanks removed.
Syntax	result$ = Ltrim$(*x$*)
Guidelines	*x$* is any string-expression
Related Instructions	Rtrim$( ), Trim$( )
Example	x$ = “ Hello “ print “(“+ Ltrim$(x$) +“)” ‘prints (Hello)

MAIN
(STATEMENT)

Purpose Main is used to indicate the start of a program. Every program begins with Main and ends with End Main. This program structure is automatically created for you when you use the New Program function on the File menu.

Syntax
```
Main
...your main program...
End Main
```

Guidelines Only one Main and End Main is allowed in any program.

Related Instructions Sub, Function, Interrupt

Example
```
Main
    print "This is all there is to it."
End Main
```

MB32WORDORDER
(PRE-DEFINED VARIABLE)

Purpose MB32WordOrder specifies the word order for 32 bit (double register) ModBus register accesses. This affects 32 bit integers. The word order for floating point variables is specified by MBFloatWordOrder. The setting for MB32WordOrder affects both master and slave operations.

This feature is only available in the Enhanced OC950 Firmware.

Syntax MB32WordOrder = x

Range 0 or 1
where:
0 - least significant word first, most significant word second
1 - most significant word first, least significant word second

Default 1

MBERR
(PRE-DEFINED VARIABLE, INTEGER)

Purpose

MBErr indicates when and which error occurred when you execute a ModBus master statement or function. MBErr is set to zero only when the program starts executing. After that, it has a "sticky" functionality in that anytime an error occurs, MBErr is updated so you can do multiple ModBus master transactions and verify that MBErr is zero to make sure all were successful.

Value	Description
0	no Error
-1	No Response from Slave (time-out)
-2	Invalid Slave Address Specified (must be 0-254)
-3	Invalid Bit Address Specified (must be 1-9999 or 10001-19999)
-4	Invalid Register Address Specified (must be 30001-39999 or 40001-49999)

This feature is only available in the Enhanced OC950 Firmware.

Syntax

MBErr = 0
x = MBErr

Range

0 to - 4

Default

0

Guidelines

Set MBErr to zero before each block of ModBus master transactions you execute. Refer to *Using an OC950 as a ModBus Master*.

Example This example sets MBErr to 0 and performs two ModBus master transactions. First, it reads a new value for RunSpeed and writes 1 to bit 1 on the ModBus slave. If either transaction fails, it calls HandleModBusError to set Out19 and stop the program.

```
RuntimeProtocol  = 3              'ModBus Master
MBFloatWordOrder = 0              'LS word first
MBErr = 0                         'initialize MBErr to zero
RunSpeed = MBReadFloat(5, 40001 )
MBWriteBit(5, 1, 1)
If MBErr <> 0 Then
  Call HandleModBusError
  ...
  Sub HandleModBusError
    NV_MBErr = MBErr              'save MBErr to NV variable
    Out19 = 0                     'indicate fault
    Stop                          'stop the program
End
'HandleModBusError
```

MBFLOATWORDORDER
(PRE-DEFINED VARIABLE)

Purpose MBFloatWordOrder specifies the word order for floating point (double register) ModBus register accesses. This affects 32 bit integers. The word order for long integer variables is specified by MB32WordOrder. The setting for MBFloatWordOrder affects both master and slave operations.

This feature is only available in the Enhanced OC950 Firmware.

Syntax MBFloatWordOrder = x

Range 0 or 1

where:
0 - least significant word first, most significant word second
1 - most significant word first, least significant word second

Default 1

MBINFO BLOCK...END (STATEMENT)

Purpose

The MBInfo block of a program is used to map pre-defined variables and/or global user variables to specific ModBus register addresses so the OC950 operates as a ModBus slave.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

```
MBInfo
    <statements>
End
```

Guidelines

MBInfo is only used when you are configuring the OC950 as a ModBus Slave. There can be only one MBInfo block in a program. It should be put before the Main section of the program. Refer to *Using an OC950 as a ModBus Slave*.

Related Instructions

$MBMapBit, $MBMap16, $MBMap32, $MBMapFloat

Example

This example maps several pre-defined variables and one global user variable (MyFloat) to ModBus registers.

```
MBInfo
    $MBMapBit(1, Dir)
    $MBMap16(40001, IndexDist)
    $MBMap32(40002, Position )
    $MBMap32(40004, MyFloat )
    $MBMapFloat( 40006, RunSpeed )
End

Dim MyFloat As Float

Main
RuntimeProtocol = 2
...
```

MBREADBIT( )
(PRE-DEFINED FUNCTION)

Purpose

This function reads a bit value (0x or 1x reference) from the specified ModBus slave and returns the value read. If any error occurs, this function returns zero and sets MBErr to indicate the source of the error.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

x = MBReadBit(*SlaveAddress, RegisterAddress*)

Guidelines

This is a ModBus master function. Set RuntimeProtocol to 3 before using this function or a runtime error is received.

ModBus master statements and functions cannot be nested. If you get an interrupt while waiting for a response to a ModBus master statement or function, you cannot initiate another ModBus master statement or function in the interrupt handler. If you do, you get runtime error 36. Refer to *Using an OC950 as a ModBus Master*.

Related Instructions

MBReadBit, MBRead16, MBRead32 , MBReadFloat, MBWriteBit, MBWrite16, MBWrite32, MBWriteFloat, MB32WordOrder, MBFloatWordOrder, MBErr

Example

This example reads a bit value from register 10005 on the ModBus slave at address 5 and puts the value in IndexDist.

```
RuntimeProtocol  = 3                    'ModBus Master
RunSpeed = MBRead32(5, 10005 )
```

MBREAD16( )
(PRE-DEFINED FUNCTION)

Purpose

This function reads an integer value from the specified ModBus slave and returns the value read. If any error occurs, this function returns zero and sets MBErr to indicate the source of the error.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

x = MBRead16(*SlaveAddress, RegisterAddress*)

Guidelines

This is a ModBus master function. Set RuntimeProtocol to 3 before using this function or a runtime error is received.

ModBus master statements and functions cannot be nested. If you get an interrupt while waiting for a response to a ModBus master statement or function, you cannot initiate another ModBus master statement or function in the interrupt handler. If you do, you get runtime error 36. Refer to *Using an OC950 as a ModBus Master*.

Related Instructions

MBReadBit, MBRead16, MBRead32 , MBReadFloat, MBWriteBit, MBWrite16, MBWrite32, MBWriteFloat, MB32WordOrder, MBFloatWordOrder ,MBErr

Example

This example reads an integer value from register 40005 on the ModBus slave at address 5 and puts the value in IndexDist.

```
RuntimeProtocol  = 3                    'ModBus Master
RunSpeed = MBRead32(5, 40005 )
```

MBREAD32( )
(PRE-DEFINED FUNCTION)

Purpose

This function reads a long integer (32 bits) value from the specified ModBus slave and returns the value read. If any error occurs, this function returns zero and sets MBErr to indicate the source of the error. The register address passed to this function is the first register address of the 32 bit integer value.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

x = MBRead32(*SlaveAddress, RegisterAddress*)

Guidelines

This is a ModBus master function. Set RuntimeProtocol to 3 before using this function or a runtime error is received.

ModBus master statements and functions cannot be nested. If you get an interrupt while waiting for a response to a ModBus master statement or function, you cannot initiate another ModBus master statement or function in the interrupt handler. If you do, you get runtime error 36.

There is not complete standardization on the format of long integer (32 bit) numbers among all ModBus devices. You may need to set MB32WordOrder to 0 (its default value is 1) in order to properly receive long integer (32 bit) numbers from a ModBus slave. Refer to *Using an OC950 as a ModBus Master*.

Related Instructions

MBReadBit, MBRead16, MBRead32 , MBReadFloat, MBWriteBit, MBWrite16, MBWrite32, MBWriteFloat, MB32WordOrder, MBFloatWordOrder ,MBErr

Example

This example reads a long integer value from registers 40003 (and 40004) on the ModBus slave at address 5 and puts the value in IndexDist. In this example, the ModBus slave sends long integer data low word first, so MB32WordOrder is set to 0 to properly receive this data.

```
RuntimeProtocol  = 3              'ModBus Master
MB32WordOrder = 0                 'LS word first
RunSpeed = MBRead32(5, 40003 )
```

MBREADFLOAT( )
(PRE-DEFINED FUNCTION)

Purpose

This function reads a floating-point value from the specified ModBus slave and returns the value read. If any error occurs, this function returns zero and sets MBErr to indicate the source of the error. The register address passed to this function is the first register address of the 32 bit floating point value.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

x = MBReadFloat(*SlaveAddress, RegisterAddress*)

Guidelines

This is a ModBus master function. Set RuntimeProtocol to 3 before using this function or a runtime error is received.

ModBus master statements and functions cannot be nested. If you get an interrupt while waiting for a response to a ModBus master statement or function, you cannot initiate another ModBus master statement or function in the interrupt handler. If you do, you get runtime error 36.

There is not complete standardization on the format of floating-point numbers among all ModBus devices. You may need to set MBFloatWordOrder to 0 (its default value is 1) in order to properly receive floating point numbers from a ModBus slave. Refer to *Using an OC950 as a ModBus Master*.

Related Instructions

MBReadBit, MBRead16, MBRead32 , MBReadFloat, MBWriteBit, MBWrite16, MBWrite32, MBWriteFloat, MB32WordOrder, MBFloatWordOrder ,MBErr

Example

This example reads a floating point value from registers 40001 and 40002 on the ModBus slave at address 5 and puts the value in RunSpeed. In this example, the ModBus slave sends floating point data low word first, so MBFloatWordOrder is set to 0 to receive this data properly.

```
RuntimeProtocol  = 3                    'ModBus Master
MBFloatWordOrder = 0                    'LS word first
RunSpeed = MBReadFloat(5, 40001)
```

MBWRITEBIT( )
(STATEMENT)

Purpose

This statement writes a bit value to a 1x reference register the specified ModBus slave. If any error occurs, this function sets MBErr to indicate the source of the error.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

MBWriteBit(*SlaveAddress,RegisterAddress,IntegerValue*)

Guidelines

This is a ModBus master statement. Set RuntimeProtocol to 3 before using it or a runtime error is received.

ModBus master statements and functions cannot be nested. If you get an interrupt while waiting for a response to a ModBus master statement or function, you cannot initiate another ModBus master statement or function in the interrupt handler. If you do, you get runtime error 36. Refer to *Using an OC950 as a ModBus Master*.

Related Instructions

MBReadBit, MBRead16, MBRead32 , MBReadFloat, MBWriteBit, MBWrite16, MBWrite32, MBWriteFloat, MB32WordOrder, MBFloatWordOrder, MBErr

Example

This example writes the integer value of Inp0 to registers 1 on the ModBus slave at address 5.

```
RuntimeProtocol  = 3                    'ModBus Master
MBWriteBit(5, 1, Inp0 )
```

MBWRITE16( )
(STATEMENT)

Purpose

MBWrite16( ) writes an integer (16 bits) value to the specified ModBus slave. If an error occurs, this function sets MBErr to indicate the source of the error.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

MBWrite16(*SlaveAddress,RegisterAddress,IntegerValue*)

Guidelines

This is a ModBus master statement. Set RuntimeProtocol to 3 before using it or a runtime error is received.

ModBus master statements and functions cannot be nested. If you get an interrupt while waiting for a response to a ModBus master statement or function, you cannot initiate another ModBus master statement or function in the interrupt handler. If you do, you get runtime error 36. Refer to *Using an OC950 as a ModBus Master*.

Related Instructions

MBReadBit, MBRead16, MBRead32, MBReadFloat, MBWriteBit, MBWrite16, MBWrite32, MBWriteFloat, MB32WordOrder, MBFloatWordOrder, MBErr

Example

This example writes the integer value of IndexDist to registers 40001 on the ModBus slave at address 5.

```
RuntimeProtocol  = 3                ‘ModBus Master
MBWrite16(5, 40001, IndexDist )
```

MBWRITE32( )
(STATEMENT)

Purpose

This statement writes a long integer (32 bits) value to the specified ModBus slave. If any error occurs, this function sets MBErr to indicate the source of the error. The register address passed to this function is the first register address of the 32 bit long integer value.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

MBWrite32(*SlaveAddress, RegisterAddress, LongIntegerValue*)

Guidelines

This is a ModBus master statement. Set RuntimeProtocol to 3 before using it or a runtime error is received.

ModBus master statements and functions cannot be nested. If you get an interrupt while waiting for a response to a ModBus master statement or function, you cannot initiate another ModBus master statement or function in the interrupt handler. If you do, you get runtime error 36.

There is not complete standardization on the format of long integer numbers among all ModBus devices. Set MB32WordOrder to 0 to properly write floating-point numbers to a ModBus slave. Refer to *Using an OC950 as a ModBus Master*.

Related Instructions

MBReadBit, MBRead16, MBRead32, MBReadFloat, MBWriteBit, MBWrite16, MBWrite32, MBWriteFloat, MB32WordOrder, MBFloatWordOrder, MBErr

Example

This example writes the long integer value of TargetPos to registers 40001 (and 40002) on the ModBus slave at address 5. In this example, the ModBus slave accepts long integer data low word first, so MB32WordOrder is set to 0 so the slave receives this data properly.

```
RuntimeProtocol  = 3              'ModBus Master
MB32WordOrder = 0                 'LS word first
MBWrite32(5, 40001, TargetPos )
```

MBWRITEFLOAT( )
(STATEMENT)

Purpose

This statement writes a floating-point value to the specified ModBus slave. If any error occurs, this function sets MBErr to indicate the source of the error. The register address passed to this function is the first register address of the 32 bit floating point value.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

MBWriteFloat(*SlaveAddress,RegisterAddress,FloatValue*)

Guidelines

This is a ModBus master statement. Set RuntimeProtocol to 3 before using it or a runtime error is received.

ModBus master statements and functions cannot be nested. If you get an interrupt while waiting for a response to a ModBus master statement or function, you cannot initiate another ModBus master statement or function in the interrupt handler. If you do, you get runtime error 36.

There is not complete standardization on the format of floating-point numbers among all ModBus devices. Set MBFloatWordOrder to 0 to properly write floating point numbers to a ModBus slave. Refer to *Using an OC950 as a ModBus Master*.

Related Instructions

MBReadBit, MBRead16, MBRead32, MBReadFloat, MBWriteBit, MBWrite16, MBWrite32, MBWriteFloat, MB32WordOrder, MBFloatWordOrder, MBErr

Example

This example writes the floating point value 1.5 to registers 40001 (and 40002) on the ModBus slave at address 5. In this example, the ModBus slave accepts floating point data low word first, so MBFloatWordOrder is set to 0 so the slave receives this data properly.

```
RuntimeProtocol  = 3              'ModBus Master
MBFloatWordOrder = 0              'LS word first
MBWriteFloat(5, 40001, 1.5 )
```

MID$( )
(FUNCTION)

Purpose Mid$( ) returns a substring of the original string that begins at the specified offset location and is of the specified (optional) length.

Syntax result = Mid$(*x$*, *start* [,*length*])

Guidelines *Start* and *length* must both be numeric expressions. If *length* is omitted, Mid$( ) returns a substring that begins at *start* and goes to the end of *x$*.

Related Instructions Instr( ), Left$( ), Len( ), Right$( )

Example

```
x$ = "abcdefghi"
  print Mid$(x$, 1, 5)        'prints: abcde
  print Mid$(x$, 6)           'prints: fghi
```

MOD
(OPERATOR)

Purpose This is the modulus or remainder. It divides one number by another and returns the remainder.

Syntax x = y MOD z

Guidelines This MOD operator is only used in numeric expressions. There is a Position Modulo value (PosModulo) and an encoder position modulo value (EncPosModulo). These are separate pre-defined variables and are not related directly to the MOD operator.

Example

```
print 19 MOD 5                'prints: 4
```

MODEL
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ-ONLY)

Purpose Model indicates the drive model number (power level).

Syntax Model = x

Range 952, 953, 954, 955

MODELEXT
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ-ONLY)

Purpose ModelExt gives information about the OC950.

Syntax x = ModelExt

Range

Model #	Explanation
501	32K
502	128K
503	32K with PACLAN
504	128K with PACLAN
601	32K with Enhanced Features
602	128K with Enhanced Features
603	32K with PACLAN and Enhanced Features
604	128K with PACLAN and Enhanced Features

Related Instructions Model

MODIFYENCPOS( )
STATEMENT

Purpose ModifyEncPos( ) translates EncPos (encoder position) from *old_value* to *new_value*.

Syntax ModifyEncPos(*old_value, new_value*)

Guidelines Use ModifyEncPos( ) to zero out the encoder position (EncPos) before starting a cam.

Related Instructions EncPos, ActiveCam

Example The following program illustrates ModifyEncPos( ). The encoder position captured by BDIO5 (Reg2 is the zero position).

```
When Reg2HiFlag, Continue
        ModifyEncPos(Reg2HiEncPos,0)
        PosCommand = 0
        ActiveCam = 1
```

MOTOR
(PRE-DEFINED VARIABLE)

Purpose Motor indicates the first 4 characters of the motor part number used to determine the Signature Series current wave shape used to eliminate torque constant ripple.

Syntax x = Motor

Range Up to any 4 ASCII characters.

Default Sine(1,162,768,483)

MOVING
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose Moving indicates whether or not the commanded motion profile is complete.

0 - commanded motion complete

1 - move in progress

Syntax x = Moving

Range 0 or 1

Default 0

Guidelines Moving only indicates whether or not the commanded motion profile is complete. Even when the commanded motion profile is completed (Moving = 0), there may still be motor motion as the result of settling time and/or electronic gearing.

Related Instructions InPosition, InPosLimit

Example

```
IndexDist = 10000
GoIncr
While Moving : Wend
Pause(0.5)
IndexDist = -IndexDist
GoIncr
```

OCDATE
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ-ONLY)

Purpose	OCDate gives the Option Card date code.
Syntax	x = OCDate
Range	0 to 231
Default	Set at factory

OCSNUM
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ-ONLY)

Purpose	OCSNum gives the Option Card serial number.
Syntax	x = OCSNum
Range	0 to 231
Default	Set at factory

OCT$( )
(FUNCTION)

Purpose	Oct$( ) converts an integer number to its equivalent octal ASCII string.
Syntax	result$ = Oct$(*x*)
Guidelines	Octal numbers are numbers to the base 8 (rather than base 10). The argument to Oct$( ) is rounded to an integer before Oct$(*x*) is evaluated.
Related Instructions	Hex$( ), Str$( )
Example	dim x,y as integer dim result1$, result2$ as string x = 20 y = &H6A result1$ = Oct$(x) result2$ = Oct$(y) print result1$, result2$ ‘Prints: 24 152

ON ERROR GOTO
(STATEMENT)

Purpose On Error GoTo allows you to define a run-time error handler to prevent run-time errors from halting program execution. Different error handlers are defined for different parts of the program. An error handler is active from when the On Error GoTo statement is executed until another one is executed.

Syntax On Error Goto Error-*Handler-Name* or
On Errror Goto 0

Guidelines An error handler has the same structure as a subroutine (must end with a Restart), disables any user-defined run-time error handler and reinstalls the default handler. Any subsequent run-time error prints an error message and halts the program.

Errors occurring within the error handler are handled by the default error handler. This means they halt program execution.

Related Instructions Restart

Example

```
dim Count as integer
main
    dim y as integer
    if Count < 10 then
        on error goto MyHandler
    else
        on error goto 0
    end if
    y = 0
    pause(0.5)
    y = 1/y
    print “I’ll never get here”
end main

Sub MyHandler
    Count=Count+1
    print Count
    restart
End Sub
```

OR
(OPERATOR)

Purpose Or performs a logical OR operation on two expressions.

Syntax result = A or B

Guidelines The result evaluates to True if either of the expressions is True. Otherwise, the result is False.

Related Instructions Or, Xor, Band, Bor, Bxor

Example
```
x = 17
y = 27
if (x > 20) or (y >20) then
    print “This will get printed”
end if

if (x < 20) or (y > 20) then
    print “...so will this.”
end if
```

OUT0-OUT20
(PRE-DEFINED VARIABLE, INTEGER)

Purpose Out*n* (Out0 - Out20) sets the state of the individual discrete outputs.

Syntax Out*n* = x

Units none

Range 0 or 1

Default 1

Guidelines **0** turns the output transistor on, output is pulled down.
1 turns the output transistor off, output is pulled up.

Related Instructions Outputs, BDOut*n*, BDOutputs

Example
```
while 1
Out1 = 1
pause(0.5)
Out1 = 0
pause(0.5)
wend
```

OUTPUTS
(PRE-DEFINED VARIABLE, INTEGER)

Purpose Outputs allows setting outputs in parallel.

Syntax Outputs = x

Units none

Range 0 to 2,097,151

Default 2,097,151 (all outputs are 1)

Guidelines For each bit in Outputs:
0 turns the output transistor on, output is pulled down.
1 turns the output transistor off, output is pulled up.

Related Instructions BDOut*n*, BDOutputs, Out*n*

Example
```
while 1
    Outputs = &h155555        'alternate outputs
    pause(0.5)
    Outputs = &h0AAAAA        'alternate again
    pause(0.5)
wend
```

PARAMS...END PARAMS
(STATEMENT)

Purpose Params...End Params specifies the values for the non-volatile parameters. This section is automatically created for you when you use the New Program selection on the File menu.

Syntax
```
Params
    parameter1 = parameter-value
    parameter2 = parameter-value
    ...
End Params
```

Guidelines The values assigned to the parameters are automatically written to these parameters the next time that you power up the drive - before the program is executed.

Even if Autostart is not set and the program does not run automatically, these values get initialized to the specified values. All other pre-defined variables get initialized to default values.

Related Instructions ARF0, ARF1, CommOff, PoleCount, Kip, ILmtMinus, ILmtPlus, ItThresh, Kpp, Kvi, Kvp

PAUSE( )
(STATEMENT)

Purpose	Pause( ) pauses program execution for a specified amount of time. The motion of the motor is unaffected.
Syntax	Pause(*x*)
Guidelines	Interrupts are active during a Pause( ) statement.
Related Instructions	Time
Example	for x = 0.1 to 2.0 step 0.1 Out0 = 1 Pause(x) Out0 = 0 Pause(x) next

POLECOUNT
(PRE-DEFINED VARIABLE, INTEGER, NV PARAMETER)

Purpose	PoleCount matches the drive for the appropriate motor pole count or encoder quadrature counts per motor cycle.
Syntax	PoleCount = x
Units	motor poles
Range	2 to 65534 (even numbers only) 1 to 65535 Encoder Counts per electrical cycle.
Default	Parameter value specified in the Params...End Params section of your program. The 950 IDE New Program function calculates this value based upon the specified motor and drive.
Guidelines	For CommSrc = 0, PoleCount sets the number of motor poles For CommSrc = 1, PoleCount sets the number of encoder quadrature counts per motor electrical cycle.

When the PoleCount set does not match the actual pole count, the motor's operation is erratic.

Set CommSrc before writng to PoleCount.

POSCOMMAND
(PRE-DEFINED VARIABLE, INTEGER)

Purpose PosCommand (Position Command) contains the current position command. The value of PosCommand is affected by PosModulo and PosPolarity.

Syntax PosCommand = x

Units resolver counts

Range -134,217,728 to +134,217,727

Guidelines PosCommand can be used to determine the position being commanded. You can write to PosCommand at any time; this establishes a new electrical home position (where PosCommand = 0). Writing to PosCommand does not affect motor motion.

Related Instructions Position, PosModulo, PosPolarity

Example The following program lines set electrical home position when Inp0 goes to a 0.

```
Dir = 0 : RunSpeed = 100 : GoVel
When Inp0 = 0, Continue
AbortMotion
While Moving : Wend
PosCommand = 0
```

POSERROR
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ-ONLY)

Purpose PosError (Actual Position Error) is equal to the difference between the position command (PosCommand) and the actual position (Position).

Syntax x = PosError

Units Counts (same units as position feedback)

Range -134,217,728 to +134,217,727

Guidelines This variable only makes sense for position control blocks, BlkType = 2.

POSERRORMAX
(PRE-DEFINED VARIABLE, INTEGER)

Purpose	PosErrorMax sets the maximum value in position feed back counts for the position loop following error fault.
Syntax	PosErrorMax = x
Units	Counts (same units as position feedback).
Range	0 to 294,912,000 (4500 revs)
Default	40960
Guidelines	The following error fault compares PosError with the PosError predicted from EncFreq and Kvff. If the magnitude of the difference is larger than PosErrorMax continuously for longer than 1 second or statistically larger over half the time, the drive generates a following error fault, F 1.

POSITION
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose	Position indicates the motor's actual position. This is a read-only variable and cannot be set directly by the software. The value of Position is affected by PosModulo and PosPolarity.
Syntax	x = Position
Units	resolver counts
Range	-134,217,728 to +134,217,727
Default	Set equal to ResPos on power up
Guidelines	If you write a new value to PosCommand, Position is automatically changed such that PosError (the difference between them) is unchanged.
Related Instructions	PosCommand, PosModulo, PosPolarity
Example	print Position, PosCommand PosCommand = 0 print Position, PosCommand

POSMODULO
(PRE-DEFINED VARIABLE, INTEGER)

Purpose PosModulo specifies the position modulo value. The position modulo value is the value of Position where Position is automatically reset to zero. If PosModulo is zero (the default value), position modulo is not used.

Syntax PosModulo = x

Units resolver counts

Range 0 to 134,217,727

Default 0 (turned off)

Guidelines PosModulo is useful for rotary motion applications.

Related Instructions EncPosModulo

POSPOLARITY
(PRE-DEFINED VARIABLE, INTEGER)

Purpose PosPolarity specifies the connection between motor shaft rotation direction (clockwise or counter-clockwise) and position variables' direction as:

0: clockwise is positive, counter-clockwise is negative

1: clockwise is negative, counter-clockwise is positive

After you change PosPolarity, all commanded motion is reversed from what it was.

Syntax PosPolarity = x

Range 0 or 1

Default 0

Guidelines The drive must be disabled to change PosPolarity. When the drive is enabled, PosPolarity is read-only. PosPolarity is used for reversing direction for an entire program.

Related Instructions PosModulo

Example

```
Enable = 0
PosPolarity = 1
Enable = 1
IndexDist = 4096                ‘goes counter-clockwise
GoIncr
while Moving : wend
pause(1)
Dir = 0 : GoVel                 ‘goes counter-clockwise
```

PRINT
(STATEMENT)

Purpose Print displays formatted output through the serial port while the program is running.

Syntax Print *expression1* [[,;] *expression2*] [;]

Guidelines 950BASIC defines zones of 13 characters used to produce output in columns. If a list of expressions is separated by commas (,), each subsequent expression is printed in the next zone. If a list of expressions is separated by semi-colons (;), the zones are ignored and consecutive expressions are printed in the next available character space. If a Print statement ends in a comma or semi-colon, the carriage-return/line-feed at the end of serial output is suppressed.

Example
```
Print “Hello” , “Goodbye”
Print “Hello” ; “Goodbye”
Print “Hello” , “Goodbye”;
Print “...The End.”
```

PULSESIN
(PRE-DEFINED VARIABLE, INTEGER)

Purpose PulsesIn specifies the number of encoder counts used when specifying an exact electronic gearing ratio. PulsesIn is the number of encoder counts required to increase PosCommand by PulsesOut resolver counts when using exact gearing.

Syntax PulsesIn = x

Units encoder counts

Range 1 to 32767

Default 1

Guidelines PulsesIn or PulsesOut must be set more recently than Ratio in order to use exact electronic gearing.

Related Instructions Gearing, PulsesOut, Ratio

PULSESOUT
(PRE-DEFINED VARIABLE, INTEGER)

Purpose	PulsesOut specifies the number of resolver counts used in an exact electronic gearing ratio. PulsesOut is the number of resolver counts the motor moves for each PulsesIn number of encoder counts.
Syntax	PulsesOut = x
Units	resolver counts
Range	-CountsPerRev/2 to CountsPerRev / 2
Default	1
Guidelines	PulsesIn or PulsesOut must be set more recently than Ratio in order to use exact electronic gearing.
Related Instructions	Gearing, PulsesIn, Ratio

RANDOM
(PRE-DEFINED VARIABLE, FLOAT, READ-ONLY)

Purpose Random returns a pseudo random number from a uniform distribution between 0.0 and 1.0 (inclusive).

Syntax x = Random

Range 0.0 to 1.0

Guidelines Seed the random number generator with Randomize.

Related Instructions Randomize

Example This program prints two identical random number sequences, followed by a different random number sequence (uses default value of Randomize to seed the random number generator with the current time).

```
main
dim i as integer
randomize(1)
   For i = 1 to 5
       print random;
   Next i
        print
randomize(1)
   For i = 1 to 5
      print random;
   Next i
        print
randomize
   For i = 1 to 5
      print random;
   Next i
end
```

RANDOMIZE
(STATEMENT)

Purpose Randomize[(*x*)] initializes the random number generator. It has an optional floating-point argument, to specify the initial seed. If the optional argument is not present, the system uses the current time as the seed. Given the same initial seed, any two sequences of random numbers are identical.

Syntax Randomize[(*x*)]

Guidelines Use Random to get a random number.

Related Instructions Random

Example This example prints two identical random number sequences followed by a different random number sequence (uses the default value of Randomize to seed the random number generator with the current time).

```
main
dim i as integer
randomize(1)
   For i = 1 to 5
       print random;
   Next i
      print
randomize(1)
   For i = 1 to 5
      print random;
   Next i
      print
randomize
   For i = 1 to 5
      print random;
   Next i
end
```

RATIO
(PRE-DEFINED VARIABLE, FLOATING POINT)

Purpose	Ratio sets the electronic gearing ratio (rev to rev) between the encoder shaft (master) and the motor shaft (slave).
Syntax	Ratio = x
Units	Motor revolutions / Encoder Revolution
Range	-2,000 to 2,000
Default	1.0
Guidelines	Ratio must be set more recently than PulsesIn or PulsesOut in order to use Ratio to control electronic gearing.
Related Instructions	EncIn

READPLC5BINARY( )
(PRE-DEFINED FUNCTION)

Purpose

ReadPLC5Binary( ) reads the specified (16 bit) element from the specified binary file on the specified PLC5.

When this function is encountered in the OC950 program, the OC950 sends the appropriate message to the PLC5 connected to the OC950's serial port and waits for the response. If there is a valid response, the OC950 puts the data in the appropriate variable (variable on the left-hand-side of the equal sign). If there is no valid response, the OC950 sets ABErr.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

x = ReadPLC5Binary(*node address, file number, element number*)

Guidelines

Set RuntimeProtocol to 5 (Allen-Bradley DF1 Protocol) before using this function. Other communication parameters (baudrate and ABCrc) on the SC950 must match the corresponding parameters on the PLC.

Related Instructions

ReadSLC5Integer( ), ReadPLC5Float( ), WritePLC5Integer( ), WritePLC5Binary( ), WritePLC5Float( )

Example

The following program reads an integer from a PLC5 binary file and sets RunSpeed to twice the integer read from the PLC5.

All communication settings on both devices (SC950 and PLC5) must match.

```
main
dim PLC5Speed as integer
runtimeprotocol = 5          'Allen-Bradley DF1 Protocol
baudrate = 19200
'baudrate MUST match PLC setting
abcrc = 1
'Set check to CRC - MUST match PLC setting
PLC5Speed = ReadPLC5Binary(5,3,19)

'PLC5 File 3 = Binary File
RunSpeed = PLC5Speed * 2
end
```

READPLC5FLOAT( )
(PRE-DEFINED FUNCTION)

Purpose

ReadPLC5Float( ) reads the specified (32 bit) element from the specified float file on the specified PLC5.

When this function is encountered in the OC950 program, the OC950 sends the appropriate message to the PLC5 connected to the OC950's serial port and waits for the response. If there is a valid response, the OC950 puts the data in the appropriate variable (variable on the left-hand-side of the equals sign). If there is no valid response, the OC950 sets ABErr.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

x = ReadPLC5Float(*node address, file number, element number*)

Guidelines

You must first set RuntimeProtocol to 5 (Allen-Bradley DF1 Protocol) before using this function. Other communication parameters (baudrate and ABCrc) on the SC950 must match the corresponding parameters on the PLC.

Related Instructions

ReadSLC5Integer( ), ReadPLC5Binary( ), WritePLC5Integer( ), WritePLC5Binary( ), WritePLC5Float( )

Example

The following program reads a float from a PLC5 binary file. It then sets RunSpeed to 3.45 times the value read from the PLC5.

All communication settings on both devices (SC950 and PLC5) must match.

```
main
dim PLC5Speed as float

runtimeprotocol = 5     'Allen-Bradley DF1 Protocol
baudrate = 19200        'baudrate MUST match PLC setting
abcrc = 1
'Set check to CRC — MUST match PLC setting
PLC5Speed = ReadPLC5Float(5, 8, 1)

'PLC5 File 8 = Float File
RunSpeed = PLC5Speed * 3.45
end
```

READPLC5INTEGER( )
(PRE-DEFINED FUNCTION)

Purpose

ReadPLC5Integer( ) reads the specified (16 bit) element from the specified integer file on the specified PLC5.

When this function is encountered in the OC950 program, the OC950 sends the appropriate message to the PLC5 connected to the OC950's serial port and waits for the response. If there is a valid response, the OC950 puts the data in the appropriate variable (variable on the left-hand-side of the equals sign). If there is no valid response, the OC950 sets ABErr.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

x = ReadPLC5Integer(*node address, file number, element number*)

Guidelines

You must first set RuntimeProtocol to 5 (Allen-Bradley DF1 Protocol) before using this function. Other communication parameters (baudrate and ABCrc) on the SC950 must match the corresponding parameters on the PLC.

Related Instructions

ReadPLC5Binary( ), ReadPLC5Float( ),WritePLC5Integer( ), WritePLC5Binary( ), WritePLC5Float( )

Example

The following program reads an integer from a PLC5. It then sets RunSpeed to twice the integer read from the PLC5.

All communication settings on both devices (SC950 and PLC5) must match.

```
main
dim PLC5Speed as integer

runtimeprotocol = 5          'Allen-Bradley DF1 Protocol
baudrate = 19200
'baudrate MUST match PLC setting
abcrc = 1
'Set check to CRC — MUST  match PLC setting
PLC5Speed = ReadPLC5Integer(5, 7, 19)

'PLC5 File 7 = Integer File
RunSpeed = PLC5Speed * 2
end
```

READSLC5BINARY( )
(PRE-DEFINED FUNCTION)

Purpose

ReadSLC5Binary( ) reads the specified element (16 bits) from the specified binary file on the specified SLC500.

When this function is encountered in the OC950 program, the OC950 sends the appropriate message to the SLC500 connected to the OC950's serial port and waits for the response. If there is a valid response, the OC950 puts the data in the appropriate variable (variable on the left-hand-side of the equals sign). If there is no valid response, the OC950 sets ABErr.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

x = ReadSLC5Binary(*SLC500 address, file number, element number*)

Guidelines

Set RuntimeProtocol to 5 (Allen-Bradley DF1 Protocol) before using this function. Other communication parameters (baudrate and ABCrc) on the SC950 must match the corresponding parameters on the PLC.

Related Instructions

ReadSLC5Integer( ), ReadSLC5Float( ), WriteSLC5Integer( ), WriteSLC5Integer( ), WriteSLC5Float( )

Example

The following program reads an integer from a SLC500 PLC binary file and sets IndexDist to twice the value read from the SLC500.

All communication settings on both devices (SC950 and SLC500) must match.

```
main
dim SLC5Dist as integer

runtimeprotocol = 5          'Allen-Bradley DF1 Protocol
baudrate = 19200
'baudrate MUST match PLC setting
abcrc = 1
'Set check to CRC — MUST match PLC setting
SLC5Speed = ReadSLC5Binary(5, 3, 19)

'SLC500 File 3 = Binary File
IndexDist = SLC5Dist * 2
end
```

READSLC5FLOAT( )
(PRE-DEFINED FUNCTION)

Purpose

ReadSLC5Float( ) reads the specified element (32 bits) from the specified Floating file on the specified SLC500.

When this function is encountered in the OC950 program, the OC950 sends the appropriate message to the SLC500 connected to the OC950's serial port and waits for the response. If there is a valid response, the OC950 puts the data in the appropriate variable (variable on the left-hand-side of the equals sign). If there is no valid response, the OC950 sets ABErr.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

x = ReadSLC5Float(*SLC500 address, file number, element number*)

Guidelines

Set RuntimeProtocol to 5 (Allen-Bradley DF1 Protocol) before using this function. Other communication parameters (baudrate and ABCrc) on the SC950 must match the corresponding parameters on the PLC.

Related Instructions

ReadSLC5Integer( ), ReadSLC5Binary( ), WriteSLC5Integer( ), WriteSLC5Integer( ), WriteSLC5Binary( )

Example

The following program reads a float from a SLC500 PLC and sets RunSpeed to 2.55 * value read from the SLC500.

All communication settings on both devices (SC950 and SLC500) must match.

```
main
dim SLC5Speed as float
runtimeprotocol = 5          'Allen-Bradley DF1 Protocol
baudrate = 19200
'baudrate MUST match PLC setting
abcrc = 1
'Set check to CRC — MUST match PLC setting
SLC5Speed = ReadSLC5Float(5, 8, 19)

'SLC500 File 8 = Float File
RunSpeed = SLC5Speed * 2.55
end
```

READSLC5INTEGER( )
(PRE-DEFINED FUNCTION)

Purpose

ReadSLC5Integer( ) reads the specified (16 bit) element from the specified integer file on the specified SLC500.

When this function is encountered in the OC950 program, the OC950 sends the appropriate message to the SLC500 connected to the OC950's serial port and waits for the response. If there is a valid response, the OC950 puts the data in the appropriate variable (variable on the left-hand-side of the equals sign). If there is no valid response, the OC950 sets ABErr.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

x = ReadSLC5Integer(*SLC500 address, file number, element number*)

Guidelines

Set RuntimeProtocol to 5 (Allen-Bradley DF1 Protocol) before using this function. Other communication parameters (baudrate and ABCrc) on the SC950 must match the corresponding parameters on the PLC.

Related Instructions

ReadSLC5Binary( ), ReadSLC5Float( ), WriteSLC5Binary( ), WriteSLC5Integer( ), WriteSLC5Float( )

Example

The following program reads an integer from a SLC500 PLC and sets RunSpeed to twice the integer read from the SLC500.

All communication settings on both devices (SC950 and SLC500) must match.

```
main
dim SLC5Speed as integer
runtimeprotocol = 5          'Allen-Bradley DF1 Protocol
baudrate = 19200
'baudrate MUST match PLC setting
abcrc = 1
'Set check to CRC — MUST match PLC setting
SLC5Speed = ReadSLC5Integer(5, 7, 19)

'SLC500 File 7 = Integer File
RunSpeed = SLC5Speed * 2
end
```

REG1HIENCPOS
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose Reg1HiEncpos contains the latched value of the encoder counter (EncPos) when the Reg1 input (J4-10) captured its last low-to-high registration event.

Set RegControl 60 0 to latch Reg1HiEncpos.

Syntax x = Reg1HiEncpos

Units encoder counts

Guidelines Set Reg1HiFlag to 0 to arm the registration latch.

Related Instructions RegControl, Reg1HiFlag, Reg1LoEncpos

REG1HIFLAG
(PRE-DEFINED VARIABLE, INTEGER)

Purpose Reg1HiFlag arms and monitors the Reg1Hi registration data latches.

Set Reg1HiFlag to zero to arm the latches (prepare them to capture data at a registration transition). This flag is automatically set to one when the hardware detects a low-to-high transition on Reg1 (J4-10).

Syntax Reg1HiFlag = x

Range 0 or 1

Default 0

Guidelines RegControl determines what data gets latched on a Reg1 transition.

Related Instructions RegControl

REG1HIPOSITION
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose	Reg1HiPosition contains the latched value of the motor position (Position) when the Reg1 input (J4-10) captured its last low-to-high registration event.
Syntax	x = Reg1HiPosition
Units	resolver counts
Guidelines	Set Reg1HiFlag to 0 to arm the registration latch.
Related Instructions	RegControl

REG1LOENCPOS
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose	Reg1LoEncpos contains the latched value of the encoder counter (EncPos) when the Reg1 input (J4-10) captured its last high-to-low registration event.  ***To latch Reg1LoEncpos, RegControl must be set to 0.***
Syntax	x = Reg1LoEncpos
Units	encoder counts
Guidelines	Set Reg1HiFlag to 0 to arm the registration latch.
Related Instructions	RegControl

REG1LOFLAG
(PRE-DEFINED VARIABLE, INTEGER)

Purpose Reg1LoFlag arms and monitors the Reg1Lo registration data latches.

Set Reg1LoFlag to zero to arm the latches (prepare to capture data at a registration transition). This flag automatically sets to one when the hardware detects a high-to-low transition on Reg1 (J4-10).

Syntax Reg1LoFlag = x

Range 0 or 1

Default 0

Guidelines RegControl determines what data gets latched on a Reg1 transition.

Related Instructions RegControl

REG1LOPOSITION
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose Reg1LoPosition contains the latched value of the motor position when the Reg1 input (J4-10) captured its last high-to-low registration event.

Syntax x = Reg1LoPosition

Units resolver counts

Guidelines Set Reg1LoFlag to 0 to arm the registration latch.

Related Instructions RegControl

REG2HIENCPOS
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose Reg2HiEncpos contains the latched value of the encoder counter (EncPos) when the Reg2 input (J4-11) captured its last low-to-high registration event.

To latch Reg2HiEncpos, RegControl must be set to 1.

Syntax x = Reg2HiEncpos

Units encoder counts

Guidelines Set Reg2HiFlag to 0 to arm the registration latch.

Related Instructions RegControl

REG2HIFLAG
(PRE-DEFINED VARIABLE, INTEGER)

Purpose Reg2HiFlag arms and monitors the Reg2Hi registration data latches.

Set Reg2HiFlag to zero to arm the latches (prepare to capture data at a registration transition). This flag automatically sets to one when the hardware detects a low-to-high transition on Reg2 (J4-11).

Syntax Reg2HiFlag = x

Units none

Range 0 or 1

Default 0

Guidelines RegControl determines what data gets latched on a Reg2 transition.

Related Instructions RegControl

REG2HIPOSITION
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose Reg2HiPosition contains the latched value of the motor position (Position) when the Reg2 input (J4-11) captured its last low-to-high registration event.

To latch Reg2HiPosition, RegControl must be set to 2.

Syntax x = Reg2HiPosition

Units resolver counts

Guidelines Set Reg2HiFlag to 0 to arm the registration latch.

Related Instructions RegControl

REG2LOENCPOS
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose Reg2LoEncpos contains the latched value of the encoder counter (EncPos) when the Reg2 input (J4-11) captured its last high-to-low registration event.

To latch Reg2LoEncpos, RegControl must be set to 1.

Syntax x = Reg2LoEncpos

Units encoder counts

Guidelines Set Reg2LoFlag to 0 to arm the registration latch.

Related Instructions RegControl

REG2LOFLAG
(PRE-DEFINED VARIABLE, INTEGER)

Purpose	Reg2LoFlag arms and monitors the Reg2Lo registration data latches. Set Reg2LoFlag to zero to arm the latches (prepare to capture data at a registration transition). This flag automatically sets to one when the hardware detects a high-to-low transition on Reg1 (J4-11).
Syntax	Reg2LoFlag = x
Range	0 or 1
Default	0
Guidelines	RegControl determines what data gets latched on a Reg2 transition.
Related Instructions	RegControl

REG2LOPOSITION
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose	Reg2LoPosition contains the latched value of the motor position (Position) when the Reg2 input (J4-11) captured its last high-to-low registration event.  ***To latch Reg2LoPosition, RegControl must be set to 2.***
Syntax	x = Reg2LoPosition
Units	resolver counts
Guidelines	Set Reg2LoFlag to 0 to arm the registration latch.
Related Instructions	RegControl

REGCONTROL
(PRE-DEFINED VARIABLE, INTEGER)

Purpose RegControl controls what data (EncPos or Position) gets latched into the registration latches. Functionality is:

Value of RegControl	Functionality
0	Reg1 transitions capture Position and EncPos Reg2 transitions are ignored
1	Reg1 transitions capture Position Reg2 transitions capture EncPos
2	Reg1 transitions capture Position Reg2 transitions capture Position

Syntax RegControl = x

Range 0, 1, 2

Default 0

Guidelines Set RegControl to the desired value before capturing any registration data.

BDIOMap4 must be set to 0 (off) if Reg1 is being used.
BDIOMap5 must be set to 0 (off) if Reg2 is being used.

Related Instructions Reg1HiFlag, Reg1LoFlag, Reg2HiFlag, Reg2LoFlag

REMOTEFB
(PRE-DEFINED VARIABLE, INTEGER)

Purpose RemoteFB selects the source of the feedback signal for the loops.

Syntax RemoteFB = x

Units When RemoteFB is not equal to 0, the units on the following variables change as shown:

Variable Name	Units (RemoteFB = 1 or 2)	Units (RemoteFB = 0)
PosCommand	encoder counts	resolver counts
RunSpeed	encoder counts/sec	rpm
AccelRate	encoder counts/sec^2	rpm/sec
DecelRate	encoder counts/sec^2	rpm/sec

Range 0, 1, or 2

Default 0 (all loops closed around resolver)

Guidelines

0 Resolver velocity and resolver position feedback
1 Resolver velocity and encoder position feedback
2 Encoder velocity and encoder position feedback

When RemoteFB is not equal to 0, EncIn must be set to the proper value so the scaling of KPP, KVP, and VelFB is in default units.

When RemoteFB is equal to 1 or 2, Encpos becomes Read-Only and Position becomes Read-Write. Use PosCommand to change the value of Encpos in this configuration.

RemoteFB is Read-Only when the drive is enabled. If you attempt to change the value of RemoteFB with the drive enabled, it is ignored.

RESPOS
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ-ONLY)

Purpose ResPos (Resolver Position) is the absolute mechanical orientation of the resolver relative to the motor housing.

Syntax x = ResPos

Units Resolver Counts (1 Resolver count = 1/65536 rev)

Range 0 to 65535

Guidelines Respos varies from zero to maximum range and then back to zero as the motor rotates positive through one complete revolution.

Related Instructions PosPolarity

RESTART
(STATEMENT)

Purpose Restart causes program execution to begin again from the beginning of the program. Restart is the only way to exit from an Error Handler routine. Any interrupts, When statements or loops in progress are aborted.

***Restart** does not clear the user program variables orchange any program variables, any pre-defined variables or has any effect on motor motion.*

Syntax Restart

Guidelines If Restart is used to exit from a user error handle, an infinite loop occurs if the error condition is not cleared.

Related Instructions AbortMotion, On Error Goto

RIGHT$( ) (FUNCTION)

Purpose Right$( ) returns a string of the *n* rightmost characters in a string expression.

Syntax result$ = Right$(*x$, n*)

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

Related Instructions Len( ), Mid$( ), Left$( )

Example
```
a$ = "Mississippi"
print Right$(a$, 5)                    'prints: sippi
```

RTRIM$( ) (FUNCTION)

Purpose Returns a copy of the original string without trailing blanks.

Syntax result$ = Rtrim$(x$)

Guidelines x$ is any string-expression

Related Instructions Ltrim$( ), Trim$( )

Example
```
x$ = "     Hello     "
print "(" + Rtrim$(x$) + ")"          'prints: (     Hello)
```

RUNSPEED (PRE-DEFINED VARIABLE, FLOATING POINT)

Purpose RunSpeed sets the maximum speed allowed during an incremental (GoIncr) or absolute (GoAbs) move, and sets the commanded speed during a velocity move (GoVel).

Syntax RunSpeed = x

Units rpm

Range 0 to 20,000 (actual maximum is set by motor and drive)

Default 1000

Guidelines Specify RunSpeed before initiating any move commands.

Related Instructions GoAbs, GoHome, GoIncr, GoVel, UpdMove

RUNTIMEPARITY
(PRE-DEFINED VARIABLE)

Purpose Specifies the Runtime Parity. Valid values are:

Value	Explanation
0	none (no parity)
1	odd parity
2	even parity

Syntax RuntimeParity = x

Range 0, 1, 2

Default 0

RUNTIMEPROTOCOL
(PRE-DEFINED VARIABLE)

Purpose Specifies runtime protocol. RuntimeProtocol valid values are:

Value	Explanation
0	none
1	user-defined binary
2	ModBus Slave
3	ModBus Master
4	OC950 Protocol (allows communication with IDE)
5	Allen-Bradley DF1 Communications Protocol

ModBus functionality (RuntimeProtocol = 2 or 3) and Allen-Bradley DF1 functionality (RuntimeProtocol = 5) are only available in the enhanced OC950 firmware.

When you set RuntimeProtocol to any value other than zero, Inp20 is automatically used to stop the program. When Inp20 is brought low (0), the program stops because when a run-time protocol is in use, it is impossible to stop the program over the serial port. This means that if you use RuntimeProtocol, neither Inp20 nor Out20 are used only to stop the program.

Syntax RuntimeProtocol = x

Range 0, 1, 2, 3, 4

Default 0

SCURVETIME
(PRE-DEFINED VARIABLE, FLOATING POINT)

Purpose ScurveTime sets the amount of S-curve smoothing applied to all velocity profiles. The greater the value of ScurveTime, the smoother (lower jerk) the profile.

Syntax ScurveTime = x

Units seconds

Range 0.000 to 0.256 seconds
(0.002, 0.004, 0.008, 0.016, 0.032, 0.064, 0.128, 0.256)

Default 0 (trapezoidal profile)

Guidelines Specifying a non-zero value for ScurveTime increases move time by ScurveTime. For example, a trapezoidal move (ScurveTime = 0) that takes 0.500 seconds to complete, takes 0.756 seconds to complete if ScurveTime is set to 0.256.

Change ScurveTime only when the motor is not moving (Moving = 0). If you attempt to change ScurveTime while the motor is moving, the command is ignored.

Related Instructions AccelRate, DecelRate

Example

```
main
Enable = 1
AccelRate = 10000
Decel Rate = 10000
RunSpeed = 1000
IndexDist = 40960

'time the move without S-curve
ScurveTime = 0
Time = 0
GoIncr
While Moving : Wend
Print Time

'now time the move with S-curve
ScurveTime = 0.256
Time = 0
GoIncr
While Moving : Wend
Print Time
end main
```

SELECT CASE
(STATEMENT)

Purpose

Select Case executes one of several statement blocks depending upon the value of an expression.

Syntax

```
Select Case test-expression
        Case expression-list1
        ...statement block1...
        Case expression-list2
        ...statement block1...
        Case expression-list3
        ...statement block1...
        Case Else
        ...else block...
End Select
```

Guidelines

The *test-expression* must evaluate to a numeric or floating-point value.

There can be unlimited Cases in the Select Case statement, but only one Case Else and it must be the last case in the sequence. The Case Else statement block is executed if all other tests fail.

Select Case statements where the *expression-lists* are integer constants are executed more quickly at run-time.

Related Instructions

If...Then...Else

Example

This example prints out some interesting information about the numbers between 1 and 20.

```
main
dim x as integer
for x = 1 to 20
    print x;" is “;
    select case x
        case 1, 3, 5, 7, 9
            print ”Odd"
        case 4, 8
            print “4 or 8"
        case 12 to 18
            print ”between 12 and 18"
        case else
            print “other”
    end select
next
end main
```

SENDLANINTERRUPT( )[ ]
(PRE-DEFINED FUNCTION)

Purpose SendLANInterrupt(*x*)[*n*] invokes PACLAN interrupt on axis *n*. The value of *x* is passed to the destination of the PACLAN interrupt and is automatically placed in the axis' LANIntrArg pre-defined variable.

Syntax result = SendLANInterrupt(*arg*)[*axis*]

where *n* identifies the address of the destination of the interrupt. The possible value returned in result is:

0 destination received and accepted the interrupt (success!)

1 PACLAN transmit failure

2 transmit OK but no response

3 destination's LANInterrupt queue is full

4 destination doesn't have a PACLAN interrupt defined

5 destination is not running a program

6 destination is busy downloading a program

Guidelines Before issuing this statement, ensure that the destination axis is connected to the PACLAN and running a program or a runtime error is generated on the source axis.

The SendLANInterrupt( )[ ] function differs from LANInterrupt[ ] in two ways: it always returns a value indicating whether or not the signal was received by the destination axis, and the LANInterrupt statement faults the drive if the destination cannot accept the signal.

SendLANInterrupt( )[ ] sends a specific argument along with the interrupt signal. For LANInterrupt[ ], the argument value is always 0.

Related Instructions LANIntrArg, LANIntrSource, Interrupt, Status

Example

The following example shows two main programs — one for axis 128, and one for axis 255. The program on axis 255 repeatedly sends a LAN interrupt signal to axis 128 with a sequence count as the argument. The program on axis 128 prints the count, the argument received and the address of the sending axis and the increments its count.

```
‘————————— axis 255 ————————————-
main
dim count as integer
while 1
      print SendLANInterrupt(count)[128]
      count=count+1
      pause(0.5)
wend
end main
‘————————— axis 128 ————————————-
main
IntrPACLAN = 1
while 1 : wend
end main

Interrupt PACLAN
      static count as integer
      print “Count:”,count
      print “Arg:”, LANIntrArg
      print “Source:”,LANIntrSource
      print “^^^^^^^^^^^^^^^^^^^^^^^^”
end interrupt
```

SETMOTOR( )
FUNCTION

Purpose	SetMotor( ) specifies the motor back EMF waveshaping to be used by the OC950.
Syntax	SetMotor(*string-expression*)
Guidelines	When you specify a motor name with SetMotor( ), the OC950 looks up that name to see if it has a custom waveshape for that motor. If it does, it uses this back EMF waveshape for signature-series waveshaping. If it does not find the motor name, it uses a sine-wave for back EMF waveshaping.
Related Instructions	GetMotor$
Example	SetMotor("R32G") Print GetMotor$

SGN( )
(FUNCTION)

Purpose	Sgn( ) returns the sign of a numeric expression.
Syntax	result = Sgn(x) if: x < 0 returns -1 x = 0 returns 0 x > 0 returns 1
Guidelines	*x* is any numeric expression.
Example	print sgn(-33) 'prints -1 print sgn(0) 'prints 0 print sgn(45.77) 'prints 1

SHL
(LEFT SHIFT OPERATOR)

Purpose	Left Shift Operator
Syntax	result = operand1 SHL operand2
Guidelines	This operator performs a left shift by operand2 places of operand1. This is equivalent to multiplying operand1 by 2 operand2 number of times.

SHRA
(ARITHMETIC RIGHT SHIFT OPERATOR)

Purpose Arithmetic Right Shift Operator

Syntax result = operand1 SHRA operand2

Guidelines This operator performs an arithmetic right shift of operand1 by operand2 number of places. This is equivalent to dividing operand1 by 2 operand2 number of times.

SHRL
(LOGICAL RIGHT SHIFT OPERATOR)

Purpose Logical Right Shift Shift Operator

Syntax result = operand1 SHRL operand2

Guidelines This operator performs a logical right shift of operand1 by operand2 number of places. In a logical right shift zeros are shifted in from the left.

SIN( )
(FUNCTION)

Purpose Sin(x) returns the sine of *x*, where *x* is in radians.

Syntax y = Sin(*x*)

Guidelines *x* must be in radians. To convert from degrees to radians, multiply by 0.017453.

SPACE$( )
(FUNCTION)

Purpose Space$( ) returns a string of *n* spaces.

Syntax result$ = Space$(*n*)
n is 0 to 255

Guidelines *n* is rounded to an integer before Space$( ) is evaluated.

Related Instructions String$( )

Example
```
x$ = "(" + Space$(2) + "hello" + Space$(6) + ")"
print x$
prints: (  hello      )
```

SQR( )
(FUNCTION)

Purpose Sqr( ) returns the square root of a numeric expression.

Syntax result = Sqr(*x*)

Guidelines *x* must be greater than or equal to zero.

Example
```
x = 10
print Sqr(x)          'prints: 3.162278
```

STATIC
(STATEMENT)

Purpose Used for declaring variables before use. All variables (except pre-defined variables) must be declared before they are used.

Static is used in a function, sub or interrupt to specify that the specified variable's value be remembered even when the function or sub is finished. The next time that the function, sub or interrupt is executed, the value is available.

Syntax Static *var1* [, *var2* [...]] as type

where type is:

INTEGER	32 bit integer
FLOAT	IEEE single precision float
STRING	default length is 32 characters

Guidelines The default length for strings is 32 characters and is overridden by following the STRING type designator with a *.

Related Instructions Dim, Sub, Function, Interrupt

Example The difference between Dim and Static in a Sub procedure is shown below. *x* is reset to zero, while *y* is incremented.

```
main
    while 1
        call MySub
        pause(1)
    wend
end main

sub MySub
    dim x as integer        'value is forgotten
    static y as integer     'value is remembered
        x = x + 1
        y = y + 1
        print x,y
end sub
```

STATUS[ ]
(PRE-DEFINED VARIABLE)

Purpose Status[*axis*] is used over PACLAN to determine if a particular axis is connected to the PACLAN and whether or not that axis is presently running a program.

Syntax x = Status[*n*]

where *n* is the address of the axis that you are interested in.

Status returns the following values:

0 axis is not connected to PACLAN

1 axis is connected but not running a program

3 axis is connected and is running a program

Example This example checks all 255 possible axis addresses and prints out a message for every axis that is connected to the PACLAN.

```
main
dim x as integer
for x = 1 to 255
    if Status[x] = 1 then
        print “Axis”;x;" is connected."
    elseif Status[x] = 3 then
        print “Axis”;x;" is running a program."
    endif
next
end main
```

STOP
(STATEMENT)

Purpose Stops execution of the user program.

Syntax Stop

Guidelines The program stops the OC950, goes back to message mode, and waits for a command over the communications link.

Related Instructions AbortMotion

STR$( )
(FUNCTION)

Purpose Str$( ) returns a string representing the value of a numeric expression.

Syntax result$ = Str$(*x*)

Related Instructions Hex$( ), Oct$( )

Example
```
x = 45.2 / 7
print str$(x)          ' prints 6.457
```

STRING$( )
(FUNCTION)

Purpose String$( ) returns a string containing the specified number of occurrences of the specified character.

Syntax
x$ = String$(*n, a$*) [1] *or*
x$ = String$(*n, m*) [2]

Guidelines *n* is the number of occurrences of the desired character (the length of the returned string).

In [1], the returned string consists of the first character in *a$*.

In [2], the returned string consists of the ASCII value of *m*.

Related Instructions Space$( )

Example
```
Print String$(5, 45)       'prints: ----
Print String$(5, "A")      'prints: AAAAA
```

SUB…END SUB
(STATEMENT)

Purpose Sub declares a sub procedure and defines the format.

Syntax Sub [*argument-list*]
...body of the sub-procedure...
End Sub

Guidelines A sub procedure is invoked with Call. A sub-procedure accepts arguments like a function, but does not return a value. If the sub-procedure does not take arguments, it is illegal to provide an empty *argument-list* (“”).

Related Instructions Call, Function, Exit, End

Example This example defines a sub-procedure that takes an integer argument.

```
main
    dim x as integer
    for x = 1 to 10
        call MySub(x)
        pause(1)
    next
end main

sub MySub(a as integer)
    print a;"—–> “;
    if a < = 5 then
        print a * 0.5
    else
        print a * 2.0
    end if
end sub
```

SWAP
(STATEMENT)

Purpose Swap exchanges the value of two variables.

Syntax Swap *x, y*

Guidelines The two variables must both be either numeric (floating point of integer) or strings.

Example

```
dim A$, B$ as string
  A$ = “Hello”
  B$ = “Good-bye”
    print A$, B$
Swap A$, B$
    print A$, B$
```

SYSLANWINDOW1-8
(PRE-DEFINED VARIABLE)

Purpose This variable provides advanced troubleshooting information about the ARCNET network.

SysLanWindowX	Description
SysLanWindow1	Number of Messages initiated by this node.
SysLanWindow2	Number of messages processed by this node.
SysLanWindow3	Number of broadcast messages initiated by this node.
SysLanWindow4	Number of broadcast messages processed by this node.
SysLanWindow5	Number of times a response could not be sent to a message.
SysLanWindow6	Number of unexpected response we have received.
SysLanWindow7	Number of messages lost due to receiver overflow.
SysLanWindow8	Number of network reconfigurations.

TAN( )
(FUNCTION)

Purpose Tan(*x*) returns the tangent of x, where x is in radians.

Syntax y = Tan(*x*)

Guidelines *x* must be in radians. To convert from degrees to radians, multiply by 0.017453.

TARGETPOS
(PRE-DEFINED VARIABLE, INTEGER)

Purpose TargetPos specifies the target position for an absolute (GoAbs) move. TargetPos is an absolute position referenced to the electrical home position (the position where PosCommand = 0).

Syntax TargetPos = x

Units resolver counts

Default 0

Guidelines Set TargetPos before initiating a GoAbs.

Related Instructions GoAbs

TIME
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose Time contains the value of the free-running 32 bit timer that is maintained by the internal firmware on the OC950. The resolution on this timer is 1 ms.

Syntax Time = x

Units seconds

Range 0 to -2,147,483 (~24.8 days)

Guidelines Time is set to zero when the SC950 is powered on.

Related Instructions WhenTime

TRIM$( )
(FUNCTION)

Purpose Trim$( ) returns a copy of the original string with leading and trailing blanks removed.

Syntax result$ = Trim$(*x$*)

Guidelines *x$* is any string-expression

Related Instructions Ltrim$( ), Rtrim$( )

Example
```
x$ = "     Hello     "
print "("+ Trim$(x$) +")"          'prints: (Hello)
```

UCASE$( )
(FUNCTION)

Purpose Ucase$( ) converts a string expression to uppercase characters.

Syntax result$ = Ucase$(*string-expression*)

Guidelines Ucase$( ) affects only letters in the string expression. Other characters (numbers) are unchanged.

Related Instructions Lcase$( )

Example
```
dim x$ as string
x$ = "u.s.a"
print Ucase$(x$)          'prints:  U.S.A
```

UPDMOVE
(STATEMENT)

Purpose	UpdMove (Update Move parameters) updates a move in progress with new move parameters. This allows you to change motion on-the-fly. UpdMove updates AccelRate, DecelRate, Dir, and RunSpeed.
Syntax	UpdMove
Guidelines	Program execution continues with the line immediately following the UpdMove statement as soon as the move is initiated. Program execution does not wait until the move is complete. The drive must be enabled in order for any motion to take place. UpdMove does not initiate motion if there is no move in progress, the UpdMove statement is ignored.
Related Instructions	AbortMotion, GoAbs, GoHome, GoIncr

VAL( )
(FUNCTION)

Purpose	Val( ) returns the numerical value of a string.
Syntax	result = Val(*a$*)
Guidelines	If the first character of *a$* is not numeric, Val( ) returns 0.
Related Instructions	Str$( )

VBUS
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose	VBus is the voltage of the high voltage DC supply, rectified from the AC line, used to power the motor.
Syntax	x = VBus
Units	Volts
Range	0 to 1,000
Guidelines	Monitor this variable to detect the presence of the AC line power for the motor DC supply. For 115 VAC line power the Bus is nominally 160 VDC. For 240 VAC line power the Bus is nominally 330 VDC. For 480 VAC line power the Bus is nominally 670 VDC.

VBUSTHRESH
(PRE-DEFINED VARIABLE, FLOAT)

Purpose	VBusThresh is an adjustable parameter that allows the drive to fault if the AC line power for the motor DC supply is low.
Syntax	VBusThresh = x
Units	Volts
Range	-1 to +1000
Default	-1 (fault is disabled).
Guidelines	When VBus < VBusThresh, the drive faults and displays a blinking **E1**. This functionality allows the drive to have an interlock so it does not move the motor unless there is sufficient motor bus voltage. VBusThresh = 255 is a good value to detect a 230 VAC line more than 15% low. ***A value of -1 disables the Bus Under Voltage Fault (E 1).***

VELCMD
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose	VelCMd is the net desired velocity loop command input.
Syntax	x = VelCmd
Units	rpm
Range	VelLmtLo to VelLmtHi (-21,000 to +21,000)
Related Instructions	VelLmtHi, VelLmtLo

VELERR
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose	VelErr is commanded velocity - measured velocity (VelCmd - VelFB).
Syntax	x = VelErr
Units	rpm
Range	-48,000 to +48,000

VELFB
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose	VelFB is the instantaneous value of the velocity feedback.
Syntax	x = VelFB
Units	rpm
Range	-48,000 to +48,000 for resolver -30,000 to +30,000 for encoder
Guidelines	For normal operation, RemoteFB = 0 or 1, VelFB is the resolver velocity. For RemoteFB = 2, VelFB is based on delta EncPos at a position loop update rate.

VELLMTHI
(PRE-DEFINED VARIABLE, FLOAT)

Purpose	VelLmtHi sets the highest VelCmd value allowed and a VelFB overspeed fault threshold.
Syntax	VelLmtHi = x
Units	rpm
Range	-21,039 to +21,039
Default	10,000
Guidelines	For BlkTypes that have a velocity loop (BlkType = 1, 2), VelCmd and VelCmd2 are clamped to be less than VelLmtHi. In torque control, BlkType (0), VelLmtHi has no clamping function. If VelLmtHi is reduced to below the current value of VelCmd2 or VelCmd, then VelCmd2 and/or VelCmd are reduced to VelLmtLo. For all BlkTypes, a fault with FaultCode = 1 occurs if |VelFb|>1.5*max of (|VelLmtLo|, |VelLmtHi|)
Related Instructions	VelLmtLo

VELLMTLO
(PRE-DEFINED VARIABLE, FLOAT)

Purpose VelLmtLo sets the smallest VelCmd value allowed and a VelFB overspeed fault threshold.

Syntax VelLmtLo = x

Units rpm

Range -21,039 to +21,039

Default -10,000

Guidelines For BlkTypes with a velocity loop (BlkType = 1, 2), VelCmd and VelCmd2 are clamped to be greater than VelLmtLo. In torque control, BlkType (0), VelLmtLo has no clamping function. If VelLmtLo is increased to above the current value of VelCmd2 or VelCmd, VelCmd2 and/or VelCmd are increased to VelLmtLo. For all BlkTypes, a fault with FaultCode = 1 occur if

|VelFb|>1.5*max of (|VelLmtLo|, |VelLmtHi|)

Related Instructions VelLmtHi

VELOCITY
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose Velocity is VelFB passed through a 3.5 Hz low pass filter.

Syntax x = Velocity

Units rpm

Range -30,000 to +30,000

Guidelines When the measured velocity exceeds Velocity's range, Velocity's value is incorrect. See VelFB for and instantaneous indication of measured velocity accurate to higher speeds.

VMDIR
(PRE-DEFINED VARIABLE, INTEGER)

Purpose

vmDir specifies the direction the virutal encoder gos when vmGoVel is executed. It also sets the direction of the virtual encoder when vmUpdMove is executed if the virtual encoder is performing a velocity move.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

vmDir = x

Range

0, 1

Default

0

Guidelines

0 is positive
1 is negative

Related Instructions

vmRunFreq, vmGoVel

Example

```
'This runs the virtual encoder forward at 20,000 counts/sec
vmRunFreq = 20000
vmDir = 0
vmGovel
pause(5)

'This runs the virtual encoder backwards at 40,000 counts/sec
vmRunFreq = 40000
vmDir = 1
vmGoVel
```

VMENCPOS
(PRE-DEFINED VARIABLE, INTEGER)

Purpose vmEncpos contains the current value of the virtual encoder counter. Control the virtual encoder using vmGoVel and vmGoIncr.

This feature is only available in the Enhanced OC950 Firmware.

Syntax vmEncpos = x

Units counts

Range 0 to (EncposModulo-1)

Guidelines EncPosModulo is used as the modulo value for vmEncpos.

Related Instructions vmGoIncr, vmGoVel, vmMoving

Example This example shows how vmEncpos is updated during a vmGoIncr move.

```
vmRunFreq  = 10000
vmIndexDist = 100000
Time = 0
EncposModulo = 200000
vmEncpos = 0
vmGoIncr
while Time < 12
 Print "Time = ";Time,"vmEncpos = ";vmEncpos,"vmMoving = ";vmMoving
 Pause(1)
wend
```

VMGOINCR (STATEMENT)

Purpose

vmGoIncr (Go Incremental) causes the virtual master to move a distance specified by vmIndexDist.

The virtual master runs at the frequency specified by vmRunFreq. Use vmUpdMove to modify this frequency during the move.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

vmGoIncr

Guidelines

Program execution continues with the line immediately following the vmGoIncr statement as soon as the move is initiated. Program execution does not wait until the move is complete. The drive does not need to be enabled in order for to use the virtual master.

Related Instructions

vmGoVel, vmStopMotion, vmUpdMove

Example

This example moves the virtual encoder 100,000 counts at a frequency of 20,000 counts/second. This move will take abount 5 seconds.

```
'set up vmEncpos and virtual move parameters
vmEncpos    = 0
vmRunFreq   = 20000
vmIndexDist = 100000          'initiate the move

time = 0            'set time to zero just for measurement

vmGoIncr

'wait for the move to be complete
while vmMoving = 1 : wend

'print the results
print "vmEncpos = ";vmEncpos
print "time     = ";time
```

VMGOVEL
(STATEMENT)

Purpose

vmGoVel (Go at Velocity) causes the virtual master to move continuously at the frequency specified by vmRunFreq in the direction (positive or negative) specified by vmDir. The frequency is modified during the move using vmUpdMove.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

vmGoVel

Guidelines

Program execution continues with the line immediately following vmGoVel as soon as the move is initiated. Program execution does not wait until the move is complete. Stop a velocity move on the virtual encoder using vmStopMotion.

Executing vmGoIncr after vmGoVel and before vmStopMotion causes the virtual encoder to switch to an incremental move that terminates when vmIndexDist encoder counts have been put out. The drive does not need to be enabled to use the virtual master.

Related Instructions

vmGoIncr , vmStopMotion, vmUpdMove

Example

This runs the virtual encoder forward at 20,000 counts/sec

```
vmRunFreq = 20000
vmDir = 0
vmGovel
```

VMMOVING
(PRE-DEFINED VARIABLE, INTEGER, READ-ONLY)

Purpose vmMoving indicates if the virtual encoder is moving.
0 - virtual encoder is not moving
1 - virtual encoder is moving

This feature is only available in the Enhanced OC950 Firmware.

Syntax x = vmMoving

Range 0, 1

Related Instructions vmGoVel, vmGoIncr

Example
```
'Start an incremental move on the virtual encoder
    vmRunFreq   = 10000
    vmIndexDist = 123456
    vmGoIncr

    time = 0
    while vmMoving : wend
    print time
```

VMRUNFREQ
(PRE-DEFINED VARIABLE, FLOATING POINT)

Purpose vmRunFreq sets the maximum frequency allowed during an incremental (vmGoIncr) move, and sets the commanded speed during a velocity move (vmGoVel).

This feature is only available in the Enhanced OC950 Firmware.

Syntax vmRunFreq = x

Units encoder counts/second

Range 0 - 1,000,000

Default 10,000

Guidelines The resolution of vmRunFreq is 1,000 counts/second

Related Instructions vmGoVel, vmDir, vmGoIncr, vmIndexDist

Example
```
'This runs the virtual encoder forward at 20,000 counts/sec
    vmRunFreq = 20000
    vmDir     = 0
    vmGovel
```

VMSTOPMOTION
(STATEMENT)

Purpose

vmStopMotion stops the virtual encoder. vmEncpos stays at its present value.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

vmStopMotion

Guidelines

Program execution continues with the line immediately following vmStopMotion as soon as the move is initiated. Program execution does not wait until the move is complete.

Related Instructions

vmGoIncr, vmGoVel, vmUpdMove

Example

This runs the virtual encoder forward at 20,000 counts/sec for 5 seconds and then stops it.

```
vmRunFreq = 20000
vmDir = 0
vmGovel
pause(5)
vmStopMotion
```

VMUPDMOVE (STATEMENT)

Purpose

vmUpdMove (Update Virtual Encoder Move paramaters) updates a move in progress with new move parameters. This allows you to change motion on-the-fly without having to stop motion and initiate a new move. vmUpdMove updates vmDir and vmRunFreq.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

vmUpdMove

Guidelines

Program execution continues with the line immediately following vmUpdMove as soon as the move is initiated. Program execution does not wait until the move is complete.

vmUpdMove does not initiate motion if there is no move in progress. The vmUpdMove statement is ignored.

Related Instructions

vmGoIncr, vmGoVel

Example

This example initiates an incremental move of 100,000 counts at 50,000 counts/sec. After 1 second, it changes the move speed to 10,000 counts/sec and updates the move parameters.

```
' set up the initial parameters and initiate the move
vmRunFreq = 50000
vmIndexDist = 100000

time = 0
vmGoIncr

'pause 1 second and then update the frequency
pause(1)
vmRunFreq = 10000
vmUpdMove

'wait for the move to be complete and print out the elapsed
time
while vmMoving : wend
print time
```

WHEN
(STATEMENT)

Purpose

The WHEN statement is used for very fast response to certain input conditions. Upon encountering and executing the WHEN statement, program execution waits until the specified condition is satisfied. When the condition is satisfied, the *when-action* is executed immediately and the program continues at the next line after the WHEN statement.

Interrupts are active and are serviced during the execution of WHEN statements. The execution of an interrupt service routine does not affect how quickly the *when-action* is executed after the *when-condition* is satisfied.

Syntax

When *when-condition* , *when-action*

when-conditions:

- INP0 - INP20 = 0,1
- BDINP1 - BDINP6 = 0,1
- Position < value
- Position > value
- EncPos < value
- EncPos > value
- PosCommand < value
- PosCommand > value
- Time > value
- Reg1HiFlag
- Reg1LoFlag
- Reg2HiFlag
- Reg2LoFlag

when-actions:

- AbortMotion
- Continue
- GoAbs
- GoHome
- GoIncr
- GoVel
- Out0 - Out20 = 0,1
- Ratio = value
- UpdMove

Guidelines The *when-condition* is checked every 1 millisecond. At the instant (within 1 msec) that the when-condition is satisfied, the values of the following variables are strobed into special when variables:

- Encpos—WhenEncPos
- PosCommand—WhenPosCommand
- Position—WhenPosition
- ResPos—WhenResPos
- Time—WhenTime

Related Instructions WhenEncPos, WhenPosCommand, WhenPosition, WhenResPos, WhenTime

Example

```
When Inp0 = 1, continue
    ...
When EncPos > 10000, Out3=1
    ...
When Time > 5.6, Ratio = -2.2
```

WHENENCPOS
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ-ONLY)

Purpose WhenEncPos records the value of EncPos when the *when-condition* is satisfied.

Syntax x = WhenEncPos

Units encoder counts

Range -2,147,483,648 to 2,147,483,647

Related Instructions When, EncPos

WHENPOSCOMMAND
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ-ONLY)

Purpose	Records the value of PosCommand when the *when-condition* is satisfied.
Syntax	x = WhenPosCommand
Units	resolver counts
Range	-134,217,728 to 134,217,727
Guidelines	The *when-condition* is checked once per millisecond.
Related Instructions	When, PosCommand

WHENPOSITION
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ-ONLY)

Purpose	Records the value of Position when the *when-condition* is satisfied.
Syntax	x = WhenPosition
Units	resolver counts
Range	-134,217,728 to 134,217,727
Guidelines	The *when-condition* is checked once per millisecond.
Related Instructions	When, Position

WHENRESPOS
(PRE-DEFINED VARIABLE, INTEGER, STATUS VARIABLE, READ-ONLY)

Purpose	Records the value of Respos when the *when-condition* is satisfied.
Syntax	x = WhenRespos
Units	resolver counts
Range	0 - CountsPerRev
Guidelines	The *when-condition* is checked once per millisecond.
Related Instructions	When, Respos

WHENTIME
(PRE-DEFINED VARIABLE, FLOAT, STATUS VARIABLE, READ-ONLY)

Purpose	Records the value of Time when the *when-condition* is satisfied.
Syntax	x = WhenTime
Units	seconds
Range	0 - 2,147,483 (~24.8 days)
Guidelines	The *when-condition* is checked once per millisecond.
Related Instructions	When, Time

WHILE...WEND
(STATEMENT)

Purpose	Executes a series of statements for as long as the condition after the WHILE is True.
Syntax	While condition *...statement block...* Wend
Guidelines	While...Wend statements may be nested. Each Wend is matched to the most recent While. Unmatched While or Wend statements cause compile time errors.
Related Instructions	Exit, For...Next
Example	Time = 0 While Time < 5 Dir = Inp0 : GoVel Wend AbortMotion

WRITEPLC5BINARY( )
(STATEMENT)

Purpose

WritePLC5Binary( ) writes the specified (16 bit) element to the specified binary file on the specified PLC5.

When this function is encountered, the OC950 sends the appropriate message to the SLC500 connected to the OC950's serial port and waits for an acknowledgement (ACK). If there is no valid response, the OC950 sets ABErr.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

WritePLC5Binary(*node address, file number, element number, value*)

Guidelines

Set RuntimeProtocol to 5 (Allen-Bradley DF1 Protocol) before using this function. Other communication parameters (baudrate and ABCrc) on the SC950 must match the corresponding parameters on the PLC.

Related Instructions

ReadPLC5Integer( ), ReadPLC5Binary( ), ReadPLC5Float( ), WritePLC5Integer( ), WriteSLC5Float( )

Example

This example writes an integer to the PLC5 binary file.

All communication settings on both devices (SC950 and PLC5) must match.

```
main
dim PLC5Speed as integer

runtimeprotocol = 5     'Allen-Bradley DF1 protocol
baudrate = 19200        'baudrate MUST match PLC setting
abcrc = 1
'Set check to CRC — MUST match PLC setting
PLC5Speed = 1234
WritePLC5Binary(5, 3, 19, PLC5Speed)

'PLC5 File 3 = Binary File
end
```

WRITEPLC5FLOAT( ) (STATEMENT)

Purpose

WritePLC5Float( ) writes the specified (32 bit) element to the specified float file on the specified PLC5.

When this function is encountered, the OC950 sends the appropriate message to the PLC5 connected to the OC950's serial port and waits for an acknowledgement (ACK). If there is no valid response, the OC950 sets ABErr.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

WritePLC5Float(*node address, file number, element number, value*)

Guidelines

Set RuntimeProtocol to 5 (Allen-Bradley DF1 Protocol) before using this function. Other communication parameters (baudrate and ABCrc) on the SC950 must match the corresponding parameters on the PLC.

Related Instructions

ReadPLC5Integer( ), ReadPLC5Binary( ), ReadPLC5Float( ), WritePLC5Integer( ), WritePLC5Binary( )

Example

This program writes a float to the PLC5 binary file.

All communication settings on both devices (SC950 and PLC5) must match.

```
main
dim PLC5Speed as float

runtimeprotocol = 5        'Allen-Bradley DF1 protocol
baudrate = 19200           'baudrate MUST match PLC setting
abcrc = 1
'Set check to CRC — MUST match PLC setting
PLC5Speed = 345.678
WritePLC5Float(5, 8, 19, PLC5Speed)

'PLC5 File 8 = Float File
end
```

WRITEPLC5INTEGER( )
(STATEMENT)

Purpose

WritePLC5Integer( ) writes the specified (16 bit) element to the specified integer file on the specified PLC5.

When this function is encountered, the OC950 sends the appropriate message to the PLC5 connected to the OC950's serial port and waits for an acknowledgement (ACK). If there is no valid response, the OC950 sets ABErr.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

WritePLC5Integer(*node address, file number, element number, value*)

Guidelines

Set RuntimeProtocol to 5 (Allen-Bradley DF1Protocol) before using this function. Other communication parameters (baudrate and ABCrc) on the SC950 must match the corresponding parameters on the PLC.

Related Instructions

ReadPLC5Integer( ), ReadPLC5Binary( ), ReadPLC5Float( ), WritePLC5Binary( ), WriteSLC5Float( )

Example

The following program writes an integer to the PLC5.

All communication settings on both devices (SC950 and PLC5) must match.

```
main
dim PLC5Speed as integer

runtimeprotocol = 5        'Allen-Bradley DF1 protocol
baudrate = 19200           'baudrate MUST match PLC setting
abcrc = 1
'Set check to CRC — MUST match PLC setting
PLC5Speed = 1234
WritePLC5Integer(5, 7, 19, PLC5Speed)

'PLC5 File 7 = Integer File
end
```

WRITESLC5BINARY( ) (STATEMENT)

Purpose

WriteSLC5Binary( ) writes the specified (16 bit) element to the specified binary file on the specified SLC500.

When this function is encountered, the OC950 sends the appropriate message to the SLC500 connected to the OC950's serial port and waits for an acknowledgement (ACK). If there is no valid response, the OC950 sets ABErr.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

WriteSLC5Binary(*node address, file number, element number, value*)

Guidelines

Set RuntimeProtocol to 5 (Allen-Bradley DF1Protocol) before using this function. Other communication parameters (baudrate and ABCrc) on the SC950 must match the corresponding parameters on the PLC.

Related Instructions

ReadSLC5Binary( ), ReadSLC5Float( ), WriteSLC5Integer( ), ReadSLC5Integer( ), WriteSLC5Float( )

Example

This example writes an integer to the SLC500 PLC binary file.

All communication settings on both devices (SC950 and SLC500) must match.

```
main
dim SLC5Speed as integer

runtimeprotocol = 5        'Allen-Bradley DF1 protocol
baudrate = 19200           'baudrate MUST match PLC setting
abcrc = 1
'Set check to CRC — MUST match PLC setting
SLC5Speed = 1234
WriteSLC5Binary(5, 3, 19, SLC5Speed)

'SLC500 File 3 = Binary File
end
```

WRITESLC5FLOAT( )
(STATEMENT)

Purpose

WriteSLC5Float( ) writes the specified (32 bit) element to the specified float file on the specified SLC500.

When this function is encountered, the OC950 sends the appropriate message to the SLC500 connected to the OC950's serial port and waits for an acknowledgement (ACK). If there is no valid response, the OC950 sets ABErr.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

WriteSLC5Float(*node address, file number, element number, value*)

Guidelines

Set RuntimeProtocol to 5 (Allen-Bradley DF1 Protocol) before using this function. Other communication parameters (baudrate and ABCrc) on the SC950 must match the corresponding parameters on the PLC.

Related Instructions

ReadSLC5Binary( ), ReadSLC5Float( ), WriteSLC5Integer( ), ReadSLC5Integer( ), WriteSLC5Binary( )

Example

This program writes a float to the SLC500 PLC float file.

All communication settings on both devices (SC950 and SLC500) must match.

```
main
dim SLC5Speed as float

runtimeprotocol = 5        'Allen-Bradley DF1 protocol
baudrate = 19200           'baudrate MUST match PLC setting
abcrc = 1
'Set check to CRC — MUST match PLC setting
SLC5Speed = 456.789
WriteSLC5Float(5, 8, 19, SLC5Speed)

'SLC500 File 8 = Float File
end
```

WRITESLC5INTEGER( )
(STATEMENT)

Purpose

WriteSLC5Integer( ) writes the specified (16 bit) element to the specified integer file on the specified SLC500.

When this function is encountered, the OC950 sends the appropriate message to the SLC500 connected to the OC950's serial port and waits for an acknowledgement (ACK). If there is no valid response, the OC950 sets ABErr.

This feature is only available in the Enhanced OC950 Firmware.

Syntax

WriteSLC5Integer(*node address, file number, element number, value*)

Guidelines

Set RuntimeProtocol to 5 (Allen-Bradley DF1Protocol) before using this function. Other communication parameters (baudrate and ABCrc) on the SC950 must match the corresponding parameters on the PLC.

Related Instructions

ReadSLC5Binary( ), ReadSLC5Float( ), WriteSLC5Binary( ), ReadSLC5Integer( ), WriteSLC5Float( )

Example

This example writes an integer to the SLC500 PLC.

All communication settings on both devices (SC950 and SLC500) must match.

```
main
dim SLC5Speed as integer

runtimeprotocol = 5        'Allen-Bradley DF1 protocol
baudrate = 19200           'baudrate MUST match PLC setting
abcrc = 1
'Set check to CRC — MUST match PLC setting
SLC5Speed = 1234
WriteSLC5Integer(5, 7, 19, SLC5Speed)

'SLC500 File 7 = Integer File
end
```

XOR
(OPERATOR)

Purpose Xor performs a logical XOR operation on two expressions.

Syntax result = A xor B

Guidelines The result evaluates to True if, and only if, one of the boolean expressions is True and the other boolean expression is False. Otherwise, the result is False.

Related Instructions Or, Xor, Band, Bor, Bxor

Example

```
x = 17
y = 27
If (x > 20) Xor (y > 20) Then
    print “This will get printed.”
End If

If (x < 20) And (y > 20) Then
    print “This won’t get printed.”
End If
```

APPENDIX A

Operating at 9600 Baud

To set up your OC950 to operate at 9600 Baud:

1. Verify the Firmware version (must be 1.2 or greater). Select **Variables** in the Compile menu.

2. Type **FWV** in the Variables/expression box and press **<Enter>**. The current value should be **1200** or greater.

3. Establish communications with the OC950 at 19200 baud. Type **BaudRate** in the **Variables/Expresion** box and press **<Enter>**. The current value should be **19200**.

4. **<Tab>** to the **New Value** box, type **9600** and press **<Enter>**. A warning message appears indicating that the Target (the OC950) is not responding.

5. Click **<OK>** to clear this error window.

6. Close the **Variables** Window.

7. Select **Communications** in the **Options Menu**.

8. In the **Communications Options** window, select **9600 baud** and click **<OK>**.

9. Return to the **Variables** window by selecting **Variables** in the **Compile Menu**, and verify that **BaudRate** is set to **9600**.

The OC950 and the 950IDE now both communicate at the new baud rate.

Contact Information

Danaher Motion Customer Support
Kollmorgen, Pacific Superior, IDC, Inland Motor, Micron, and NDC products
Phone: (815) 226-2222
Email: customer.service@DanaherMotion.com
Web: www.DanaherMotion.com

INDEX

$

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X