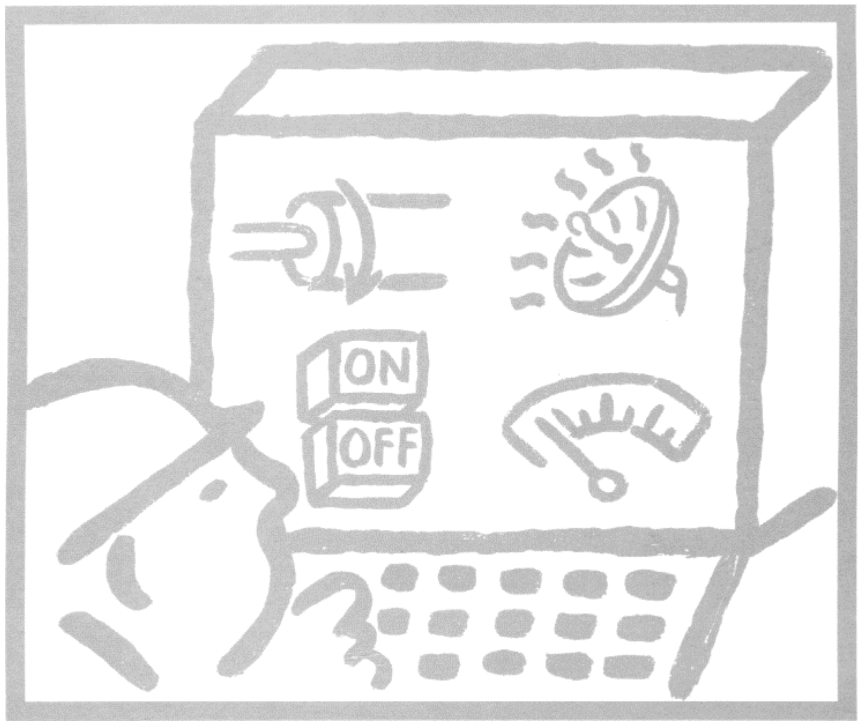
ASIC-200 Version 5.0

integrated industrial control software



Language Reference

Pro-face
XYCOM™

Revision	Description	Date
C	Name change, correct where applicable with document	4/07

Language Reference: 139183(C)

Published by: Pro-face
750 North Maple Road
Saline, MI 48176

ASCI-200 release 5.0 documents include:

Getting Started	137586
User Guide	139837
Language Reference	139183
HMI Guide	139168

Note: the current revisions of each of these documents should be used.

Note: Features available on your system depend on product version
and installed options (toolkits).

Contents

Introduction

This document provides language reference information. It contains general information on identifiers, data types, and symbols, and a description of each function and function block and how to use the functions and function blocks in each programming language.

Identifiers

Identifiers are used for symbol names. Identifiers have the following characteristics:

- Consist of upper and lower case letters (A-Z and a-z), numerals (0-9), and the underscore (_) character.
- Must begin with a letter or single underscore character.
- Case is considered.
- For uniqueness of an identifier, every character position is considered.
- Cannot contain multiple, sequential underscore characters.
- Cannot contain spaces.
- A maximum of 100 characters are allowed.
- Cannot be a system reserved keyword. Refer to Keywords for a list of keywords.
- Must be unique within its scope.

Examples of valid user identifiers are:

_Sym1	
Sym_Two_A	
SYM	(SYM and sym are considered unique identifiers)
Sym	
A	(A and AA are considered unique identifiers)
AA	

Examples of invalid user identifiers are:

1A	(begins with a numeral)

Sym___Two	(multiple sequential underscores)
Sym Two	(contains a space)
Sym&One	(contains an invalid character)
END	(a reserved identifier)

Literals

Literals are used to define or represent data values. For example, literals can be used as inputs to functions or function blocks, can be used to assign values to variables and constants, and used within program statements. There are four types of literals: numeric, character string, time duration, and time-of-day and date.

Numeric Literals

Numeric literals are either integer or real. Integer literals can be decimal, base 2, base 8, or base 16. Examples of numeric literals:

0 456 +34 -7_000	(integer literals)
0.0 0.11	(real literals)
2#1010_1010 (170 decimal)	(base 2 literal)
8#252 (170 decimal)	(base 8 literal)
16#AA (170 decimal)	(base 16 literal)
FALSE 0 TRUE 1	(Boolean literal)

Notes:

1. A numeric literal can contain single underscore (_) characters and do not affect the value of the literal.
2. Real literals contain a decimal point.
3. Decimal based numeric literals can contain a leading + or - sign.
4. The keywords FALSE and TRUE correspond to Boolean 0 and 1, respectively.

Character String Literals

A string literal is a string of 0 or more characters delimited by single quotation marks ('). The $ (dollar sign) character has a special use in a string literal. If it is followed by two hexadecimal digits, it is interpreted as the hexadecimal representation of the eight-bit character code. It also is used in two-character strings to represent the dollar sign ($), single quote (') and specified unprintable characters.

Examples of character string literals:

''	(the empty string)
'XYZ'	(a three-character string)

' '	(a space)
'$41 $42 $43'	(a five-character string 'A B C')
‘$$’	(dollar sign)
‘$’ ’	(single quote)
‘$L’ or ‘$l’	(line feed)
‘$N’ or ‘$n’	(new line)
‘$P’ or ‘$p’	(form feed)
‘$R’ or ‘$r’	(carriage return)
‘$T’ or ‘ $t’	(tab)

Time Duration Literals

Time duration literals are prefixed by the keyword T#, TIME#, t#, or time# and followed by one or more units of time. Examples of time duration literals:

T#1D1H1M1S	(1 day, 1 hour, 1 minute, 1 second)
Time#1d_1h_1m_1s	(same as preceding)
time#25h1ms	(25 hours, 1 millisecond)
t#1m_2.5s	(1 minute, 2.5 seconds)

Notes:

1. The time units can be written in upper or lower case. D, d=days; H, h=hours; M, m=minutes; S, s=seconds; and MS, ms=milliseconds.
2. An underscore (_) can be used to separate the time duration units.
3. The most significant unit of a time duration literal can overflow.
4. The least significant unit of a time duration literal can be written as a real number (with no exponent).

Time of Day and Date Literals

Time of day and date literals are prefixed by one of the following keywords and followed by time of day and date in the appropriate format.

DATE#YYYY-MM-DD	(date only)
D#YYYY:MM:DD	(same as previous)
TIME_OF_DAY#HH:MM:SS.MS	(time only)
TOD#HH:MM:SS.MS	(same as previous)
DATE_AND_TIME#YYYY-MM-DD-HH:MM:SS.MS	(date and time)
DT#YYYY-MM-DD-HH:MM:SS.MS	(same as previous)

Examples of time of day and date literals:

DATE#1998-02-13

D#1998-02-13

TIME_OF_DAY#12:00:00

TOD#12:00:00.01

DATE_AND_TIME#1998-02-13-12:00:00.01

Note: The date and time keywords can be abbreviated. DATE or D; TIME_OF_DAY or TOD; DATE_AND_TIME or DT.

Data Types

Data types must be assigned to symbols (variables and constants). Characteristics of the elementary data types are given in the following paragraphs. Generic data types are described in Error! Reference source not found.. User-defined data types are described in **User-Defined Data Type**.

In the following descriptions:

- Generic type gives the generic types for which this data type can be substituted.
- Size is the amount of memory that one instance of the data type occupies (for example, the single-bit Boolean type is really stored as a byte).
- Range is the range of values an instance of this type can take on.
- Default is the default initial value given to an instance of this type if an initial value is not otherwise specified.

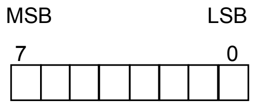
BOOL (Boolean)

A BOOL can have one of two states: 0 or 1, corresponding to FALSE or TRUE.

Generic type	ANY, ANY_BIT.
Size	1 bit.
Range	0 (FALSE), 1 (TRUE).
Default value	0

BYTE

A BYTE is a bit string of length 8.

Generic type	ANY, ANY_BIT.
Size	1 byte.
Range	Not applicable.
Format	
Default value	00000000

DATE

This data type is used to represent a date (only) in the format YYYY-MM-DD.

If you create an expression of DATE data types, all values must be of the same type, and the result must be a date.

Generic type	ANY, ANY_DATE.
Size	4 bytes
Range	-
Default value	D#0001-01-01

DINT (Double Integer)

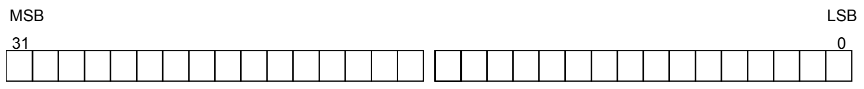
The DINT is a signed integer data type that is composed of one or more of the digits (0-9) and cannot contain a decimal point.

Generic type	ANY, ANY_NUM, ANY_INT.
Size	4 bytes.
Range	-2147483648 to +2147483647.
Default value	0

DWORD (Double WORD)

A DWORD is a bit string of length 32.

Generic type	ANY, ANY_BIT.
Size	4 bytes.
Range	Not applicable.
Format	



Default value	0

INT (Integer)

The INT is a signed integer data type that is composed of one or more of the digits (0-9) and cannot contain a decimal point.

Generic type	ANY, ANY_NUM, ANY_INT.
Size	In an enhancement to the IEC 1131-3 specification, the INT is 4 bytes.
Range	-2147483648 to +2147483647.
Default value	0

REAL

A REAL number data type is a 64-bit value composed of one or more of the digits (0-9), is signed, and contains a decimal point. (When you communicate with the run-time engine with a Fast DDE interface, 64 bit REAL data types are transmitted at 32-bit precision.)

Generic type	ANY, ANY_NUM, ANY_REAL.
Size	8 bytes.
Range	-3.402823 E38 to -1.401298 E-45 (negative), +1.401298 E-45 to +3.402823 E38 (positive).
Default value	0.0

STRING

ASCII character string of variable length.

Generic type	ANY.
Size	64 bytes.
Default value	'' (empty string)
Format	String of ASCII characters in single quotation marks. Example: ‘ This is a valid string. ’

TIME

This data type is used to represent a time duration in the format T#[nD][nH][nM][nS][nMS], where n is the number of Days, Hours, Minutes, Seconds, or Milliseconds.

If you create an expression of TIME data types, all values must be of the same type, and the result must be a time.

Generic type	ANY.
Size	-
Range	-
Default value	T#0S
Format	

TOD (TIME_OF_DAY)

This data type is used to represent the time of day (only) in the format HH:MM:SS.

If you create an expression of TOD data types, all values must be of the same type, and the result must be a TOD.

Generic type	ANY, ANY_DATE.
Size	4 bytes.
Range	00:00:00 to 23:59:59
Default value	TOD#00:00:00
Format	HH:MM:SS (hours:minutes:seconds).

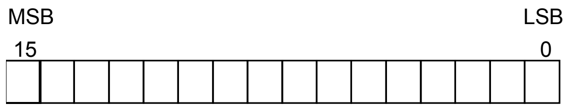
UINT (Unsigned Integer)

A UINT is an unsigned integer data type that is composed of one or more of the digits (0-9) and cannot contain a decimal point.

Generic type	ANY, ANY_NUM, ANY_INT.
Size	2 bytes.
Range	0 to 65535.
Default value	0

WORD

A WORD is a bit string of length 16.

Generic type	ANY, ANY_BIT.
Size	2 bytes.
Range	Not applicable.
Format	
Default value	0

Data Type Overrange/Rollover Conditions

In general, allowance should be made to prevent data value overrange or rollover conditions. What is meant by overrange is, for example, assigning a bit string of WORD type length to a variable of type BYTE or incrementing a variable beyond its range or bit string length (rollover). This could occur implicitly if, for example, adding two BYTE values and assigning the result to another BYTE variable. In this case, whether data type overrange occurs depends on the (run-time) values of the two integers. To avoid overrange in this case, the result could be assigned to a WORD.

Note: There is no compile-time or run-time indication of potential or actual data type overrange/rollover conditions. You must account for this in your program.

BOOL	BOOLs are either 0 if their assigned value is 0 or 1 otherwise.
BYTE	In general, overrange behavior of numeric (ANY_NUM) variables is to assume the value of the low-order bytes of the assignment, up to their own data size. Binary type (ANY_BIT) behavior is obvious. In other cases, the result depends on whether the data type is signed or unsigned, etc.
WORD	
DWORD	
REAL	
DINT	
INT	
SINT	
UDINT	
UINT	
USINT	
STRING	If a STRING data type variable is assigned a string exceeding its length, right-most characters exceeding its string length are truncated.

Type Conversion

The compiler performs implicit data type conversion automatically. The user may perform explicit type conversion using the standard MOVE function. The following sub-sections describe the results of type conversion.

Note: In certain cases, the MOVE function and the assignment statement in structured text do not behave the same. In particular, an attempt to MOVE a REAL to a BOOL results in a compiler error; however, assigning a REAL to a BOOL (BoolVar:=RealVar) in structured text does not result in a compiler error.

BOOL

A variable of type BOOL can be assigned to any numeric variable (ANY_NUM). An Illegal type conversion ERROR is issued by the compiler for any other BOOL type assignments.

For assignments to BOOL variables, refer to the following table.

Bool01 := BoolA;	Bool01 follows BoolA.
Bool01 := ByteA;	Bool01 = 0 if the numeric variable evaluates to 0; otherwise, Bool01 = 1.
Bool01 := WordA;	
Bool01 := DWordA;	
Bool01 := RealA;	
Bool01 := DIntA;	
Bool01 := IntA;	
Bool01 := SIntA;	
Bool01 := UDIntA;	
Bool01 := UIntA;	
Bool01 := USIntA;	
Bool01 := TODA;	Bool01 = 0 if TODA :=TOD#00:00:00; otherwise Bool01 = 1.
Bool01 := DateA;	Bool01 = 0 if DateA := DATE#1970-01-01; otherwise Bool01 = 1.
Bool01 := DTA;	Bool01 = 0 if DTA := DT#1970-01-01-00:00:00; otherwise Bool01 = 1.
Bool01 := TimeA;	Bool01 = 0 if TimeA := T#000.0ms; otherwise Bool01 = 1.
Bool01 := StringA;	Compiler issues *Illegal type conversion* ERROR

BYTE

A variable of type BYTE can be assigned to any numeric variable (ANY_NUM), as conditioned by overrange. A BYTE can also be assigned to a variable of type TIME; the result is entered into the TIME variable as microseconds. An Illegal type conversion ERROR is issued by the compiler for any other BYTE type assignments.

For assignments to BYTE variables, refer to the following table. WORD (DWORD) type assignments are similar with adjustments made for data size.

Byte01 := BoolA;	Byte01 follows BoolA.
Byte01 := ByteA;	Byte01 follows ByteA.
Byte01 := WordA;	Byte01 = the lower byte of the WORD (DWORD, LWORD) variable.
Byte01 := DWordA;	

Byte01 := RealA;	If the ANY_REALvariable is positive, then Byte01 = the value of the lowest, non-decimal byte of the ANY_REAL variable. If the ANY_REAL variable is negative, then Byte01 = the 1's complement value of the lowest, non-decimal byte of the ANY_REAL variable.
Byte01 := DIntA;	If the ANY_INT variable is positive (or unsigned), then Byte01 = the value of the lowest byte of the ANY_INT variable. If the ANY_INT variable is negative, then Byte01 = the 1's complement value of the lowest byte of the ANY_INT variable.
Byte01 := IntA;	
Byte01 := SIntA;	
Byte01 := UDIntA;	
Byte01 := UIntA;	
Byte01 := USIntA;	
Byte01 := TODA;	No meaningful result. At compile time, the compiler will issue a warning.
Byte01 := DateA;	
Byte01 := DTA;	
Byte01 := TimeA;	Byte1 = the number of seconds in the TIME variable, as conditioned by overrange.
Byte01 := StringA;	Compiler issues *Illegal type conversion* ERROR

WORD (DWORD)

A variable of type WORD can be assigned to any numeric variable (ANY_NUM), as conditioned by overrange. A WORD can also be assigned to a variable of type TIME; the result is entered into the TIME variable as microseconds. An Illegal type conversion ERROR is issued by the compiler for any other WORD type assignments.

For assignments to WORD variables, similar rules apply as with BYTE assingments, taking into consideration the WORD length. Refer to BYTE.

REAL (LREAL)

A variable of type REAL can be assigned to any numeric variable (ANY_NUM), as conditioned by overrange. A REAL can also be assigned to a variable of type TIME; the result is entered into the TIME variable as microseconds. An Illegal type conversion ERROR is issued by the compiler for any other REAL type assignments.

For assignments to REAL variables, refer to the following table. LREAL variable assignments are similar with adjustment made for data size.

Real01 := BoolA;	Real01 assumes the value of the numeric variable, conditioned by round off or overrange.
Real01 := ByteA;	
Real01 := WordA;	
Real01 := DWordA;	
Real01 := RealA;	
Real01 := DIntA;	
Real01 := IntA;	
Real01 := SIntA;	
Real01 := UDIntA;	
Real01 := UIntA;	
Real01 := USIntA;	
Real01 := TODA;	No meaningful result. At compile time, the compiler will issue a warning.
Real01 := DateA;	
Real01 := DTA;	(The number of microseconds since 1970-01-01-00:00:00 to the current date/time.)
Real01 := TimeA;	Real01 = the number of seconds in the TIME variable, as conditioned by overrange.
Real01 := StringA;	Compiler issues *Illegal type conversion* ERROR

INT (SINT, DINT)

A variable of type INT (SINT, DINT) can be assigned to any numeric variable (ANY_NUM), as conditioned by overrange if applicable. An INT can also be assigned to a variable of type TIME; the result is entered into the TIME variable as microseconds. An Illegal type conversion ERROR is issued by the compiler for any other INT type assignments.

For assignments to INT variables, refer to the following table. Other integer type assignments are similar with adjustments made for their data size.

Int01 := BoolA;	Int01 = BoolA.
Int01 := ByteA;	Int01 = ByteA.
Int01 := WordA;	Int01 = the lower two bytes of the WORD (DWORD) variable, with the most significant bit the sign bit.
Int01 := DWordA;	
Int01 := RealA;	Int01 = the lower two, non-decimal bytes of the REAL variable, with the most significant bit as the sign bit.
	Int01 = the signed INT variable conditioned by overrange.
Int01 := DIntA;	
Int01 := IntA;	
Int01 := SIntA;	

Int01 := UDIntA;	Int01 = the unsigned INT variable conditioned by overrange, but with the most-significant bit taken as the sign bit.
Int01 := UIntA;	
Int01 := USIntA;	
Int01 := TODA;	No meaningful result. At compile time, the compiler will issue a warning.
Int01 := DateA;	
Int01 := DTA;	
Int01 := TimeA;	Int01 = the number of seconds in the TIME variable, as conditioned by overrange.
Int01 := StringA;	Compiler issues *Illegal type conversion* ERROR

UINT (USINT, DINT)

A variable of type UINT (USINT, UDINT) can be assigned to any numeric variable (ANY_NUM), as conditioned by overrange. A UINT can also be assigned to a variable of type TIME; the result is entered into the TIME variable as microseconds. An Illegal type conversion ERROR is issued by the compiler for any other UINT type assignments.

For assignments to unsigned UINT variables, refer to the following table. Other unsigned integer type assignments are similar with adjustments made for data size.

UInt01 := BoolA;	UInt01 = the BOOL, BYTE, or WORD (DWORD) value, conditioned by overrange.
UInt01 := ByteA;	
UInt01 := WordA;	
UInt01 := DWordA;	
UInt01 := RealA;	If a positive REAL type is assigned to an unsigned integer type, then it assumes the value represented by the whole-number portion least-significant bytes, up to its own byte size. If a negative REAL type is assigned to an unsigned integer type, then it assumes the value represented by the 1's complement of the whole-number portion least-significant bytes, up to its own byte size.
UInt01 := DIntA;	If a positive signed integer type is assigned to an unsigned integer type, then the unsigned integer type variable assumes the value represented by the least-significant bytes of the signed type variable, up to its own byte size. If a negative signed integer type is assigned to an unsigned integer type, then the unsigned integer type variable assumes the value represented by the 1's complement of the least-significant bytes of the signed integer type variable, up to its own byte size.
UInt01 := IntA;	
UInt01 := SIntA;	

UInt01 := UDIntA;	UInt01 = the unsigned integer value, conditioned by overrange.
UInt01 := UIntA;	
UInt01 := USIntA;	
UInt01 := TODA;	No meaningful result. At compile time, the compiler issues a warning.
UInt01 := DateA;	
UInt01 := DTA;	
UInt01 := TimeA;	UInt01 = the number of seconds in the TIME variable, as conditioned by overrange.
UInt01 := StringA;	Compiler issues *Illegal type conversion* ERROR

TOD, DATE, DATE_AND_TIME

Variables of these data types can only be assigned meaningfully to a variable of a like type (i.e., TOD_A := TOD_B, etc.). However a variable of any of these types can also be assigned to any numeric type (BOOL, BYTE, WORD, INT, REAL, etc.). The compiler issues a WARNING for these type assignments.

TIME

A variable of type TIME can be assigned to any numeric variable (RealVar:=TimVar). The result will be the number of seconds in the TIME variable, as conditioned by overrange. If assigning a TIME variable to a BOOL variable, the BOOL variable will equal 0 if the TIME variable equals T#000.0ms; otherwise, the BOOL variable will equal 1. The compiler issues an I*llegal type conversion* error for any other TIME type assignments.

Any numeric variable (ANY_NUM, with the exception of BOOL) can be assigned to a TIME variable (TimeVar:=RealVar), with the time result in seconds.

STRING

A variable of type STRING can only be assigned to another variable of type STRING. The compiler issues an Illegal type conversion error for any other STRING type assignments.

Generic Data Types

The following table shows the data type and generic type hierarchy. The generic types are those prefixed by ANY_ and are used in the function or function block descriptions where applicable (instead of detailing a long list of data types).

For example, if a function input accepts a data type of ANY_NUM, then a symbol having a data type of REAL, any of the integer types (INT, DINT, etc.), or any of the bit string types (WORD, DWORD, etc.) can be assigned to the function input. If a function input accepts a data type of ANY_BIT, then only a symbol having a data type of WORD, DWORD, etc. can be assigned to the function input.

Data Type Hierarchy					
ANY_ OR_ DERIVED	ANY	ANY_NUM	ANY_REAL	REAL	
			ANY_INT_ OR_ BIT	ANY_INT	DINT, INT, UINT
				ANY_BIT	DWORD, WORD, BYTE, BOOL
		STRING			
		ANY_DATE			DATE, TIME_OF_DAY
		TIME			
	Derived				

User-Defined Data Type

For more sophisticated data handling, you can create your own structured data types. A structure can contain several members of different base types or user-defined structured types. Consider a user-defined structure named UserStructure01 having and integer type USInt member, a Boolean type USBool, and a string type USString. The individual members can be accessed in the following manner:

```
UserStructure01.USInt:=101;
UserStructure01.USBool:= TRUE;
UserStructure01.USString:="ABC";
```

User-defined types are valid anyplace that accepts an ANY or USER-DEFINED data type. Refer to Error! Reference source not found. for more information.

Arrays

To access a particular element of an array, enter the symbol name followed by square brackets with the number of the element you wish to access. For example to access the fifth element of an array symbol called Myarray you would type Myarray[5]. This is assuming you have defined the lower bound of the array as 1. You can also index into an array by placing a symbol name of type INT inside the square brackets.

Pointer Symbols

Structured Text has two pointer operators: the pointer reference & operator and the pointer dereference * operator. These operators are used in indirect addressing operations.

In programming languages, data values are typically referred to by symbolic name. This is known as direct addressing. The data value is given a symbolic name and that symbolic name is used to directly access that data value.

```
X := Y;
```

The data value known as X is assigned the value of the Y data value.

Indirect addressing is a common paradigm in programming languages. When using indirect addressing, the symbolic name refers to the location where the data value is stored. These indirect symbols are commonly called pointer symbols. To get the actual data value using indirect addressing, the symbolic name of the pointer symbol is used to obtain the location of the data value, then the location of the data value is used to get the actual data value (an indirect operation).

If pVar1 is a pointer symbol, then in the following assignment, pVar1 is assigned the location of the X data value.

```
pVar1 := & X;
```

If pVar1 is a pointer symbol, then in the following assignment, Y is assigned the value contained in Var1, since pVar1 contains the location of Var1.

```
Y := * pVar1;
```

If pVar1 is a pointer symbol, then in the following assignment,Var1 is assigned the value contained in Y.

```
* pVar1 := Y;
```

Pointer Notes

1. When a pointer symbol is defined, it is defined as a pointer to a symbol of a specific data type (REAL, INT, STRING, etc.). For example, the pointer symbol pVar1 could be assigned the location of any symbol of its data type.
2. As with other symbols, pointer symbols are defined in the Symbol Manager.
3. Pointers to standard data types and user structures can be defined. Pointers to function blocks and system objects cannot be defined.
4. A pointer symbol can be substituted for a symbol of the same base type by prefixing it with the dereference operator *. (*pVarInt1 = VarInt1, provided the assignment pVarInt1:=&VarInt1 has been performed.)
5. A pointer to a structure can be used directly in place of the structure name.

Assume a user structure UserStruct1 is defined and pStruct1 is defined as a pointer to this user structure.

For user structure symbols, the name of the user structure (in this case UserStruct1) is a pointer to the user structure data values. However, the user structure name can never be assigned to a different location, it will always point to the user structure.

```
pStruct := UserStruct1;
```

pStruct is assigned the location of UserStruct1.

```
pStruct.intMember1 := VarInt1;
```

The intMember1 member of UserStruct1 is assigned the value of VarInt1.

6. A pointer that is assigned an address in an array (e.g., pInt:= &intArray[5];) can be used with the array index operator (pInt[Index]) to index into the array. This index starts with 0 (the array element pointed to by the pointer) and continues to the end of the array. For example:

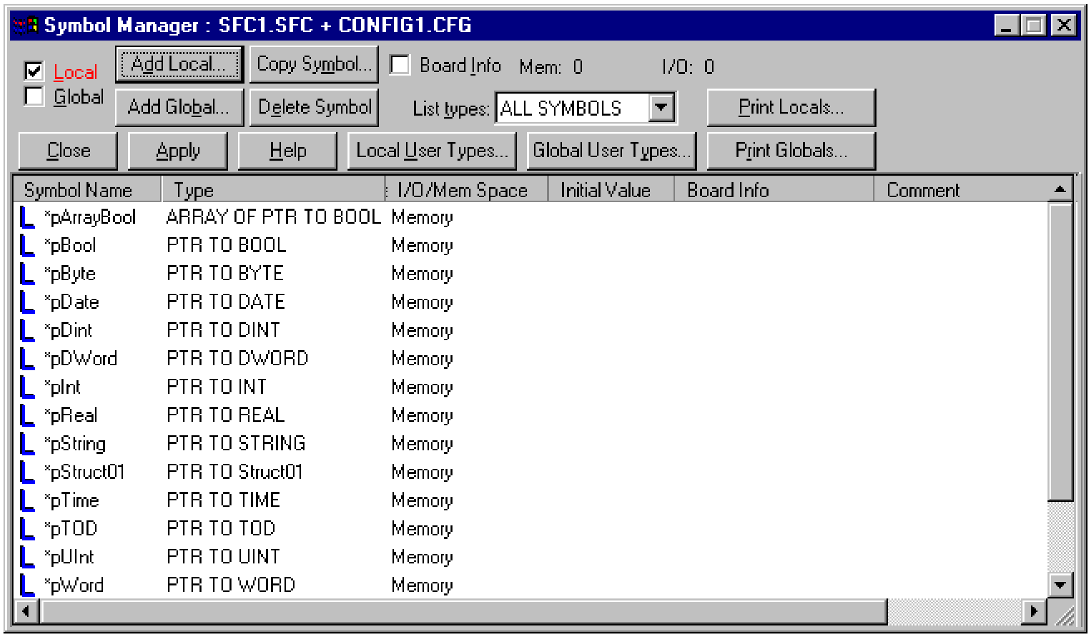
   intArray is defined as an array of ten integers (ARRAY[1..10])
   pInt is defined as a pointer to integer type
   Then:

   ```
   pInt:= &intArray[5];
   pInt[0]:= 0;            (*intArray[5]*)
   pInt[5]:= 5;            (*intArray[10]*)
   ```

7. The location of a pointer symbol can be initialized in a MOVE function block. The output of the MOVE function block is the pointer symbol to be initialized. The input of the MOVE function block is either another pointer symbol or a direct symbol preceded by the pointer reference operator &.
8. Pointers cannot be passed into FILE functions, bit array functions (SHL, AND_BITS, etc.), and STRING_TO_ARRAY functions.

The following figure shows the definition of several pointer types.

For more examples of using pointers, refer to Structured Text Programming.



Symbol Name	Type	I/O/Mem Space	Initial Value	Board Info	Comment
*pArrayBool	ARRAY OF PTR TO BOOL	Memory			
*pBool	PTR TO BOOL	Memory			
*pByte	PTR TO BYTE	Memory			
*pDate	PTR TO DATE	Memory			
*pDint	PTR TO DINT	Memory			
*pDWord	PTR TO DWORD	Memory			
*pInt	PTR TO INT	Memory			
*pReal	PTR TO REAL	Memory			
*pString	PTR TO STRING	Memory			
*pStruct01	PTR TO Struct01	Memory			
*pTime	PTR TO TIME	Memory			
*pTOD	PTR TO TOD	Memory			
*pUInt	PTR TO UINT	Memory			
*pWord	PTR TO WORD	Memory			

System Symbols

Predefined System Symbols

The system software automatically creates the following symbols that can be used with application programs.

Symbol	Description
TODAY	Contains the current system date. The TODAY system symbol is a DATE data type that contains the current system date and can be used to determine when an event takes place. The following operators can be used with the TODAY symbol: EQ, LT, GT, LE, GE, and NE. Use the assignment statement or MOVE command to define a value for TODAY. Use the ADD command to add a time duration to TODAY.

Symbol	Description
NOW	Contains the current system time. The NOW system symbol is a TOD data type that contains the current system time and can be used to determine when an event takes place. The following operators can be used with the NOW symbol: EQ, LT, GT, LE, GE, and NE. Use the assignment statement or MOVE command to define a value for NOW. Use the ADD command to add a time duration to NOW.
NULL	Used to set a pointer symbol to a null value or to compare a pointer symbol (equal or not equal) to a null value.
TMR Variables	These variables contain status information for the TMR data type.
Counter Variables	These variables contain status information for RLL counters (CTD, CTU, and CTUD). These symbols are Local to the application program.
Timer Variables	These variables contain status information for RLL timers (TOF, TON, and TP). These symbols are Local to the application program.
"stepname".X	Contains the active/inactive status of an SFC step. These symbols are Local to the application program.
"stepname".T	Contains the elapsed execution time of an SFC step. These symbols are Local to the application program.
Motion Control	These variables contain status information for the axis, axis variables group, program control, and spindle
File Control Block	These variables contain status information for file operations.
Program Control	These variables contain the status information for the PRGCB data type.

Run-Time Symbols

The following symbols are automatically created by the system software. These symbols are accessible from the Symbol Manager or the Watch Window and can be used in user application programs.

Symbol Name	Description
RT_ERROR	(INT) Math errors: 0 = no error 1 = divide by zero 2 = negative square root RT_ERROR must be cleared by the user.

Symbol Name	Description
RT_FIRST_SCAN	(BOOL) set to TRUE on the first scan of the first program running in the ASIC run-time engine. After all programs are aborted, RT_FIRST_SCAN will be set again for the first scan of the first program to run.
RT_SCAN_OVERRUN	(BOOL) set to TRUE when I/O scan + logic scan exceed scan rate.
RT_MAX_SCAN	(REAL) duration in milliseconds of maximum run-time engine scan.
RT_LAST_SCAN	(REAL) duration in milliseconds of last runtime engine scan.
RT_AVG_SCAN	(REAL) duration in milliseconds of average runtime engine scan (runtime scan= logic + I/O + overhead). Rolling average calculated over the last 100 scans.
RT_LOGIC_MAX	(REAL) duration in milliseconds of maximum logic scan.
RT_LOGIC_LAST	(REAL) duration in milliseconds of last logic scan.
RT_LOGIC_AVG	(REAL) duration in milliseconds of average logic scan. Rolling average calculated over the last 100 scans.
RT_IO_MAX	(REAL) duration in milliseconds of maximum I/O scan.
RT_IO_LAST	(REAL) duration in milliseconds of last I/O scan.
RT_IO_AVG	(REAL) duration in milliseconds of average I/O scan. Rolling average calculated over the last 100 scans.
RT_MEM_PCT	(REAL) contains remaining percentage of system RAM (heap space) allocated for the programmable control system software.
RT_LOW_BATTERY	(BOOL) low battery signal from UPS.
RT_POWER_FAIL	(BOOL) power fail signal from UPS.
RT_SCAN_RATE	(REAL) configured scan rate of (in milliseconds) as set in the active configuration.

Keywords

The identifiers listed in the following table are reserved system symbols (keywords). Do not create user symbols using these identifiers. Note that all system symbols are in uppercase letters. This list is subject to change on future releases of the product. To avoid future conflicts, user created symbols should be of mixed case or lower case.

ABORT_ALL	ABS	AC
ACCEL	ACOS	ACTION
ADD	ADD_NOFLY	AND_SLOWFLY
AND	AND_BITS	ANDN
ANDT	ANDTN	ANY

ANY_BIT	ANY_DATE	ANY_INT
ANY_NUM	ANY_REAL	APPENDFILE
ARRAY	ARRAY_TO_STRING	AS
ASIN	AT	ATAN
AXIS	AXISGRP	AXSJOG
BCD_TO_INT	BEGIN	BEGIN_IL
BEGIN_RS274	BOOL	BREAK
BY	BYTE	CASE
CAL	CALC	CALCN
CD	CLK	CLOSEFILE
CONCAT	CONFIGURATION	CONSTANT
COPYFILE	COS	CTD
CTU	CTUD	CU
CV	D	DATE
DATE_AND_TIME	DELETE	DELETEFILE
DINT	DIV	DO
DS	DSPMSG	DT
DWORD	ELSE	ELSEIF
EN	END	END_ACTION
END_CASE	END_CONFIGURATION	END_FOR
END_FOR_NOWAIT	END_FUNCTION	END_FUNCTION_BLOCK
END_IF	ENDIF	END_IL
END_RS274	ENO	END_PROGRAM
END_REPEAT	END_RESOURCE	END_STEP
END_STRUCT	END_TRANSITION	END_TYPE
END_VAR	END_WHILE	END_WHILE_NOWAIT
EQ	ESTOP	ET
EXIT	EXP	EXPT
F	FALSE	FB
F_EDGE	FILE	FIND
F_EDGE	FNAME	FOR
FROM	F_TRIG	FTYPE
FUNCTION	FUNCTION_BLOCK	G
GE	GLOBAL	GOTO
GT	H	I
IF	IN	IN1
IN2	INCLUDE	INIT
INITIAL_STEP	INSERT	INT
INTERVAL	INT_TO_BCD	INT_TO_REAL
INT_TO_STRING	IP	J

JMP	JMPC	JMPCN
JOGCONT	JOGDIR	JOGDIST
JOGHOME	JOGINCR	JOGMINUS
JOGPLUS	JOGSPD	JOGTYPE
K	L	LD
LDN	LDT	LE
LEFT	LEN	LIMIT
LINT	LL	LN
LOG	LREAL	LT
LWORD	M	MACROSTEP
MAX	MC	MID
MIN	MOD	MOVE
MOVEAXS	MS	MSGWND
MUL	MULP	MUX
N	NAME	NE
NEWFILE	NIL	NOT
NOW	NT	NULL
		OF
ON	OPENFILE	OR
OR_BITS	ORN	ORT
ORTN	OUT	P
PID	POSTN	POW
PRGCB	PRIORITY	PROGRAM
PT	PV	PW
Q	QU	QD
R	R1	READFILE
READ_ONLY	READ_WRITE	REAL
REAL_TO_STRING	R_EDGE	REPEAT
REPLACE	RESET_ESTOP	RESOURCE
RET	RETC	RETCN
RETAIN	RETURN	REWINDFILE
RIGHT	ROL	ROR
RS	RTC	R_TRIG
RUNG	S1	SCAN
SD	SEL	SEMA
SET_LOADSIZE	SHL	SHR
SIN	SINT	SL
SR	SQRT	ST
STT	STN	STTN
STEP	STOPJOG	STRING

STRING_TO_ARRAY	STRUCT	SUB
T	TAN	TASK
THEN	TIME	TIME_OF_DAY
TMR	TO	TOD
TODAY	TOF	TON
TP	TRANS	TRANSITION
TRUE	TRUNC	TYPE
UDINT	UL	ULINT
UNTIL	USINT	VAR
VAR_ACCESS	VAR_EXTERNAL	VAR_GLOBAL
VAR_INPUT	VAR_IN_OUT	VAR_OUTPUT
VEL	WHILE	WRITEFILE
WITH	WORD	XOR
XOR_BITS	XORN	XTON
XTOF	XTP	ZZZZ

Functions and Function Blocks

Functions are pre-defined algorithms that return a single value. They do not preserve any state information.

Function blocks are pre-defined algorithms that are instantiated (that is, a function block is a type, and each use of the function block must be given an instance name) and that output one or more values. A function block instantiation preserves its state between calls to it. Each function block instantiation must be given a unique name.

The standard functions and function blocks described in this reference document can be used within the Relay Ladder Logic, Structured Text, and Instruction List language editors.

Note: Currently, pointers can not be passed into FILE functions, bit string functions, and STRING_TO_ARRAY functions.

Function Execution Control

All functions have associated Enable (EN) and Enable Output (ENO) variables. EN is implicitly declared as an input variable; ENO is implicitly declared as an output variable. The RLL Editor uses these variables as the rung input and rung output for most functions.

In operation, the following sequence controls execution of a function:

1. When the function is invoked:
   - If EN is FALSE, EN0 is reset FALSE and the function is not executed. The default value for EN is TRUE.
   - If EN is TRUE, ENO is set TRUE and the function is executed. (Since ENO is declared as an output variable, a Boolean value can be assigned to it.)
2. During function execution, if an error is detected, ENO is reset FALSE.

In the RLL language, functions are placed on a rung with their EN and ENO variables. EN is placed toward the left (power) side of the rung and ENO is placed toward the right side of the rung.

RLL Diagrams

When a function block is inserted into an RLL diagram, it is automatically given an instance name, such as CTU17, where CTU is the function block type (an up counter) and 17 is an instance number, incremented for each function block inserted into the diagram. You can give the function block an instance name instead of using the default, but it must be unique. RLL function block instances do not appear in the Symbol Manager. Function block inputs and outputs are referenced by the instance name followed by the variable as given in the description. For example CTU17.R is the reset input for the counter instance CTU17.

The following rules govern the use of C function blocks or functions (which are defined in a DLL) in Relay Ladder Logic:

C Function Block

If the first input and first output are defined as type BOOL, the function block can be used in RLL and the first input and first output are attached to the rung. If the first input or first output are not defined as type BOOL, the function block cannot be used in RLL.

C Function

The rung input to the C function is never seen by the function. If the rung input is FALSE, the function is not called. If the rung input is TRUE, the function is called .If the return type of the C function is BOOL, the return value will be passed as power on the rung to the next RLL element. If the return type of the C function is not BOOL, the power on the rung will always be passed (TRUE) unless a system error has been signaled by the C function.

Structured Text

Before using a function block (or system object - PID, PRGCB, TMR), an instance of the type must be created in the Symbol Manager.

The following is an example of the syntax used for a standard function block in Structured Text (C DLL function blocks use the same type of syntax):

ctu1 and ton1 are created in the Symbol Manager with types CTU and TON respectively.

```
ctu1.PV := 10;
ctu1.EN := TRUE;
ton1.IN := TRUE;
ton1.PT := t#30s;
ton1.IN := TRUE;
ton1.EN := TRUE;
WHILE NOT In2 DO
                    ctu1.CU := In3;
                    ctu1();
                    Out3 := ctu1.Q;
                    ton1();
                    Out2 := ton1.Q;
END_WHILE;
```

Notice that all of the inputs must be explicitly set and the outputs must be explicitly transferred to the output destination. Also, the function block must be continually called using the ctu1(); syntax.

The standard function block types cannot currently be used as arrays (the C DLL function blocks can be used as arrays). For C DLL function blocks a function block array call is made with the following syntax:

```
FOR index := 1 TO maxIndex
        cdllfb1[index]();
END_FOR;
```

Instruction List

The following op codes are for calling function blocks and functions from Instruction List:

CAL	always call the function or function block.
CALC	only call the function if the accumulator is TRUE.
CALCN	only call the function if the accumulator is FALSE.

Note that CAL is used with either a function or function block and CALC and CALCN are used only with a function.

Accumulator Relationships

For function blocks, there is no relationship between the accumulator and the function block. The function block is always called, the accumulator is not passed into the function block, and after the function block returns, the accumulator has the same value as it had before the function block was called.

For functions the accumulator has no effect on the function inputs. When using CALC, the function will not be called if the accumulator is FALSE.

When using CALCN, the function will not be called if the accumulator is TRUE. When the function returns, if the return value is a BOOL, the return value is automatically loaded into the accumulator. If the function return value is a non BOOL, the return value can be saved into a variable using the func1 (OUT:=outvar1) syntax. In this case the return value is only saved into outvar1 if the function is actually called (CAL, CALC with TRUE accumulator, or CALCN with FALSE accumulator). If the function return value is a non-BOOL, the accumulator should be automatically loaded with the inverted value of the BOOL system error symbol (RTERROR). If the function is called and an error is flagged by the function, the accumulator will be loaded with a FALSE value after the function returns. If the function is called and no error is flagged by the function, the accumulator will be loaded with a TRUE value.

The following is an example of the syntax used for a standard function:

```
CALC   EXPT(OUT:=VarReal, AnyNum1, AnyNum2)
```

The following are examples of the syntax used for a standard function block:

```
LD     10
ST     ctu1.PV
LD     TRUE
ST     ctu1.EN
LD     In3
ST     ctu1.CU
CAL    ctu1
LD     ctu1.Q
ST     Out3
```

```
LD     TRUE
ST     ton1.IN
ST     ton1.EN
LD     t#30s
ST     ton1.PT
CAL    ton1
LD     ton1.Q
ST     Out2
```

Bit String

Introduction

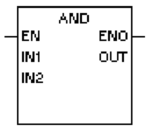
Bit string functions include:

AND	Computes the Boolean or bitwise AND of two variables.
NOT	Computes the Boolean or bitwise complement of a variable.
OR	Computes the Boolean or bitwise OR of two variables.
XOR	Computes the Boolean or bitwise Exclusive OR of two variables.
Rotate Left	Rotates the input left by the number of bits specified by a shift number (circular bit shift).
Rotate Right	Rotates the input right by the number of bits specified by a shift number (circular bit shift).
Shift Left	Shifts the input left by the number of bits specified by a shift number (0-fill on right bit shift).
Shift Right	Shifts the input right by the number of bits specified by a shift number (0-fill on left bit shift).

AND

Description Returns the Boolean or bitwise logical AND of the input values.

RLL



ST Function **AND(***AnyBit1*, *AnyBit2***)**

ST Operator **out :=** *AnyBit1* **AND** *AnyBit2;*
out := *AnyBit1* **&** *AnyBit2;*

IL Function **CALC AND(OUT:=** *VarBit*, *AnyBit1*, *AnyBit2***)**

Where

AnyBit1, AnyBit2 (IN1, IN2)	The values to be ANDed. Data type: ANY_BIT.
return (OUT)	The result of ANDing the inputs. Data type: ANY_BIT.

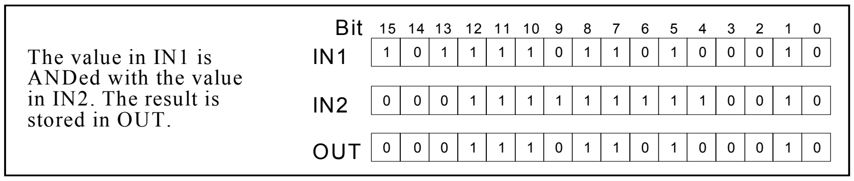
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Truth Table The truth table for the AND is as follows.

IN1	IN2	OUT
0	0	0
0	1	0
1	0	0
1	1	1

Example The result of ANDing bits is shown in the following figure.

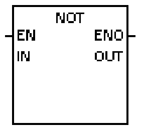


The value in IN1 is ANDed with the value in IN2. The result is stored in OUT.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IN1	1	0	1	1	1	1	0	1	1	0	1	0	0	0	1	0
IN2	0	0	0	1	1	1	1	1	1	1	1	1	0	0	1	0
OUT	0	0	0	1	1	1	0	1	1	0	1	0	0	0	1	0

Description Returns the Boolean or bitwise inversion of the input value.

RLL



ST Function **NOT(***AnyBit***)**

ST Operator **out := NOT** *AnyBit;*
out := ! *AnyBit;*

IL Function **CALC NOT(OUT:=** *VarBit*, *AnyBit***)**

Where

AnyBit (IN)	The value to be NOTed. Data type: ANY_BIT.
return (OUT)	The NOTed value of *AnyBit*. Data type: ANY_BIT.

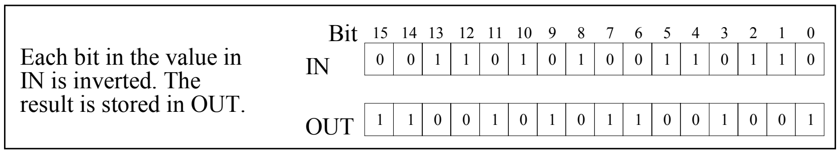
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Truth Table The truth table for the NOT is as follows.

IN	OUT
0	1
1 (non-zero)	0

Example The result of NOTing bits is shown in the following figure.

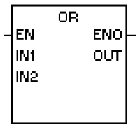

Each bit in the value in IN is inverted. The result is stored in OUT.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IN	0	0	1	1	0	1	0	1	0	0	1	1	0	1	1	0
OUT	1	1	0	0	1	0	1	0	1	1	0	0	1	0	0	1

OR

Description Returns the Boolean or bitwise logical OR of the input values.

RLL



ST Function **OR(***AnyBit1***,** *AnyBit2***)**

ST Operator **out :=** *AnyBit1* **OR** *AnyBit2*

IL Function **CALC OR(OUT:=** *VarBit***,** *AnyBit1***,** *AnyBit2***)**

Where	
AnyBit1, AnyBit2 (IN1, IN2)	The values to be ORed. Data type: ANY_BIT.
return (OUT)	The result of ORing the inputs. Data type: ANY_BIT.

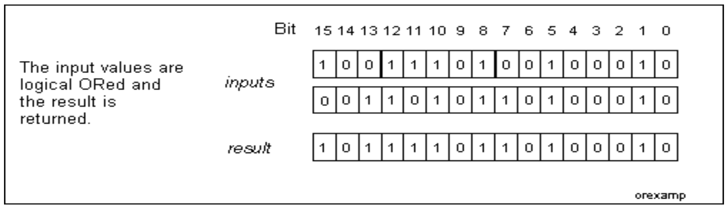
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Truth Table The truth table for the OR is as follows.

IN1	IN2	OUT
0	0	0
0	1	1
1	0	1
1	1	1

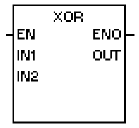
Example The result of ORing bits is shown in the following figure.


The input values are logical ORed and the result is returned.

Exclusive OR (XOR)

Description Returns the Boolean or bitwise logical Exclusive OR of the input values.

RLL RLL



ST Function **XOR(***AnyBit1*, *AnyBit2***)**

ST Operator **out :=** *AnyBit1* **XOR** *AnyBit2***;**

IL Function **CALC XOR(OUT:=** *VarBit*, *AnyBit1*, *AnyBit2***)**

Where

AnyBit1, AnyBit2 (IN1, IN2)	The values to be XORed. Data type: ANY_BIT.
return (OUT)	The result of XORing the inputs. Data type: ANY_BIT.

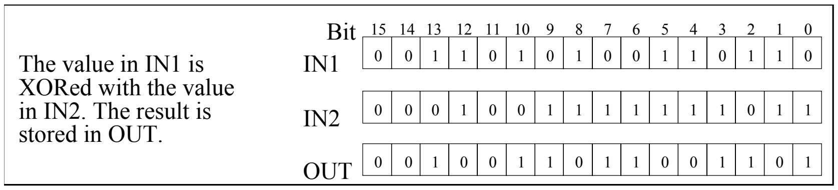
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Truth Table The truth table for the XOR is as follows.

IN1	IN2	OUT
0	0	0
0	1	1
1	0	1
1	1	0

Example The result of XORing bits is shown in the following figure.



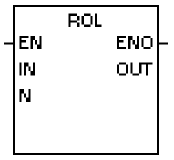
The value in IN1 is XORed with the value in IN2. The result is stored in OUT.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IN1	0	0	1	1	0	1	0	1	0	0	1	1	0	1	1	0
IN2	0	0	0	1	0	0	1	1	1	1	1	1	1	0	1	1
OUT	0	0	1	0	0	1	1	0	1	1	0	0	1	1	0	1

Rotate Left (ROL)

Description Returns a value calculated by circularly shifting the bits of the input value a specified number of positions to the left. Bit values shifted from the most significant bit (MSB) position are rotated to the least significant bit (LSB) position.

RLL



ST Function **ROL(IN :=** *BitString*, **N :=** *RotateNum***)**

IL Function **CALC ROL(OUT:=** *VarBit*, **IN:=** *BitString*, **N:=** *RotateNum***)**

Where

BitString (IN)	The value to be rotated. Data type: ANY_BIT.
RotateNum(N)	Specifies the number of bit positions to rotate *BitString*. Data type: ANY_INT.
return (OUT)	The result of rotating the bits in *BitString*. Data type: ANY_BIT.

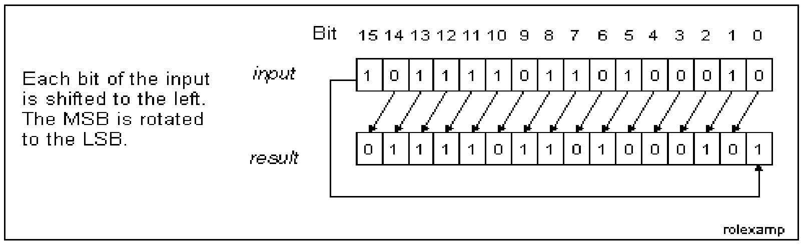
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example 1 byte1 := ROL (IN:= byte2, N:= num);

If byte2 = 10000010, and num = 1, then after execution, byte1 = 00000101.

Example 2 An example of left-rotating the bits in a word is shown in the figure (N:=1).

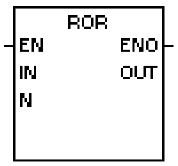


Each bit of the input is shifted to the left. The MSB is rotated to the LSB.
Bit 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
input 1 0 1 1 1 1 0 1 1 0 1 0 0 0 1 0
result 0 1 1 1 1 0 1 1 0 1 0 0 0 1 0 1
rolexamp

Rotate Right (ROR)

Description Returns a value calculated by circularly shifting the bits of the input value a specified number of positions to the right. Bit values shifted from the least significant bit (LSB) position are rotated to the most significant bit (MSB) position.

RLL



ST Function **ROR(IN:=** *BitString*, **N:=** *RotateNum***)**

IL Function **CALC ROR(OUT:=** *VarBit*, **IN:=** *BitString*, **N:=** *RotateNum***)**

Where

BitString (IN)	The value to be rotated. Data type: ANY_BIT.
RotateNum (N)	Specifies the number of bit positions to rotate *BitString*. Data type: ANY_INT.
return (OUT)	The result of rotating the bits in *BitString*. Data type: ANY_BIT.

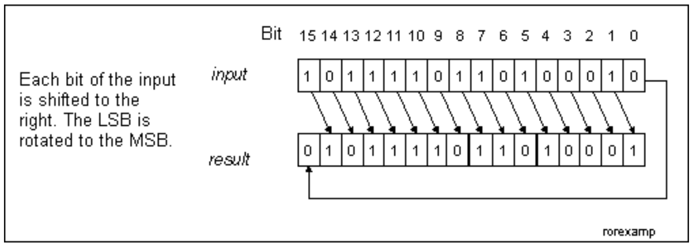
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example 1 byte1 := ROR (IN:= byte2, N:= num);

If byte2 = 00001001, and num = 1, then after the operation, byte1 = 10000100.

Example 2 An example of right-rotating the bits in a word is shown in the figure (N:=1).

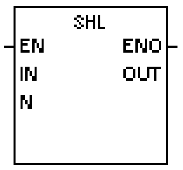

Each bit of the input is shifted to the right. The LSB is rotated to the MSB.

Shift Left (SHL)

Description

Returns a value calculated by shifting the bits of the input value a specified number of positions to the left. Bit values shifted from the most significant bit position are discarded during the shift, and the least significant bit positions are zero-filled.

RLL



ST Function

SHL(IN:=*BitString***, N:=** *ShiftNum***)**

IL Function

CALC SHL(OUT:= *VarBit***, IN:=** *BitString***, N:=** *ShiftNum***)**

Where

BitString (IN)	The value to be shifted. Data type: ANY_BIT.
ShiftNum (N)	Specifies the number of bit positions to shift *BitString*. Data type: INT.
return (OUT)	The result of shifting the bits in *BitString*. Data type: ANY_BIT.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

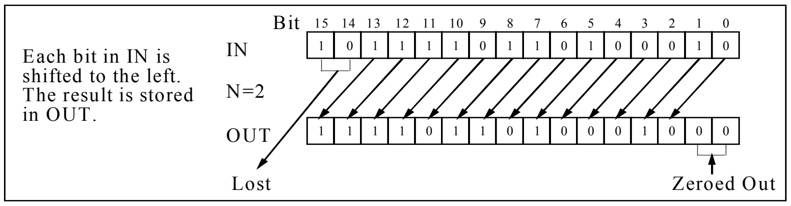
Example 1

byte1 := SHL (IN:= byte2, N:= num);

If byte2 = `10000100` and num = 1, then after the shift operation has completed, num = `00001000`.

Example 2

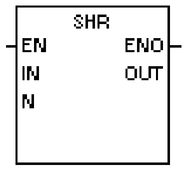
An example of left-shifting the bits in a word is shown in the figure.


Each bit in IN is shifted to the left. The result is stored in OUT.
Bit 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
IN 1 0 1 1 1 1 0 1 1 0 1 0 0 0 1 0
N=2
OUT 1 1 1 1 0 1 1 0 1 0 0 0 1 0 0 0
Lost
Zeroed Out

Shift Right (SHR)

Description Returns a value calculated by shifting the bits of the input value a specified number of positions to the right. Bit values shifted from the least significant bit position are discarded during the shift, and most significant bit positions are zero-filled.

RLL



ST Function **SHR(IN:=***BitString***, N:=** *ShiftNum***)**

IL Function **CALC SHR(OUT:=** *VarBit***, IN:=** *BitString***, N:=** *ShiftNum***)**

Where

BitString (IN)	The value to be shifted. Data type: ANY_BIT.
ShiftNum (N)	Specifies the number of bit positions to shift *BitString*. Data type: INT.
return (OUT)	The result of shifting the bits in *BitString*. Data type: ANY_BIT.

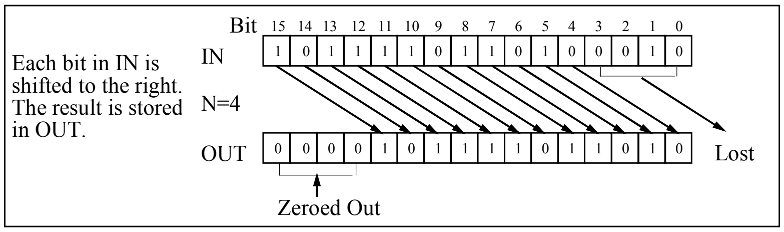
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example 1 byte1 := SHR (IN:= byte2, N:= num);

If byte2 = 10000100 and num = 1, then after the shift operation has completed, byte1 = 01000010.

Example 2 An example of right-shifting the bits in a word is shown in the figure.



Character String

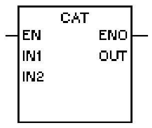
Introduction

Concatenate	Concatenates one or more input strings to the end of an initial string.
Delete	Deletes characters from a string.
Find	Searches for one string within another string.
Insert	Inserts a string into another string.
Left	Returns the leftmost characters of a string.
Length	Returns the length of a string.
Mid	Returns characters from the middle of a string.
Right	Returns the rightmost characters from a string.
Replace	Replaces characters in a string with another string.

Concatenate

Description Returns the result of concatenating two strings (appending one string to the end of another string).

RLL



ST Function **CONCAT(***StringA*, *StringB***)**

IL Function **CALC CONCAT(OUT:=** *StringAB*, **IN1:=** *StringA*, **IN2:=** *StringB***)**

Where	
StringA, *StringB* (IN1, IN2)	Specifies the strings to be concatenated. Data type: STRING (0 to 255 characters).
return (OUT)	The result of the concatenation of the strings. Data type: STRING.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

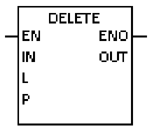
Example fullstring := CONCAT (firststring, laststring);

If firststring = "Control_" and laststring = "Power", then fullstring = "Control_Power".

Delete

Description Returns the result of deleting a specified number of characters from a specified position in the middle of the input string.

RLL



ST Function **DELETE(IN:=** *StringA*, **L:=** *NumChar*, **P:=** *Position***)**

IL Function **CALC DELETE(OUT:=** *StringB*, **IN:=** *StringA*, **L:=** *NumChar*, **P:=** *Position***)**

Where	Type Description
StringA (IN)	The string from which characters are deleted. Data type: STRING (0 to 255 characters).
NumChar (L)	Specifies the number of characters to delete. Data type: INT.
Position (P)	Specifies the position within the string to begin deleting characters. Valid values: INT.
return (OUT)	The string resulting from deleting characters from the input string. Data type: STRING.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example areacode := DELETE (IN := phonenum, L := 9, P := 4);

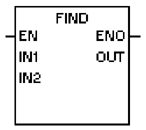
If phonenum = "567 445 9999" then areacode = "567".

Where the first character position in the string is 1.

Find

Description Returns the starting position of one string within a second string. FIND returns 0 if the string is not found.

RLL



ST Function **FIND (IN1:=** *StringA*, **IN2:=** *StringB***)**

IL Function **CALC FIND(OUT:=** *VarInt*, **IN1:=** *StringA*, **IN2:=** *StringB***)**

Where

StringA (IN1)	The string of characters to be searched. Data type: STRING (0 to 255 characters).
StringB (IN2)	The string of characters for which a match is to be found. Data type: STRING (0 to 255 characters).
return (OUT)	The result of the search. Data type: INT.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

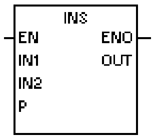
Example position := FIND (IN1:= fullname, IN2:= midname);

If fullname = "ann marie williams" and midname = "marie", then position = 5.

Insert

Description Returns a string formed by inserting one string of characters into another string at a specified position.

RLL



ST Function **INSERT(IN1:=***StringA***, IN2:=** *StringB***, P:=** *Position***)**

IL Function **CALC INSERT(OUT:=** *StringC***, IN1:=** *StringA***, IN2:=** *StringB***, P:=** *Position***)**

Where	
StringA (IN1)	The string of characters into which another string is to be inserted. Data type: STRING.
StringB (IN2)	The string of characters that is to be inserted into S*tringA*. Data type: STRING.
Position (P)	Specifies the character position of S*tringA* at which S*tringB* is inserted. Data type: INT.
return (OUT)	The resulting string. Data type: STRING.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

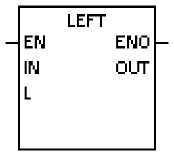
Example fullname := INSERT (IN1:= name, IN2:= midname, P:= position);

If name = "ann williams", midname = "marie", and position = 5, then fullname = "ann marie wiliams".

Left

Description Returns a specified number of the leftmost characters of the input string.

RLL



ST Function **LEFT (IN:=** *StringA*, **L:=** *NumChar***)**

IL Function **CALC LEFT(OUT:=** *StringB*, **IN:=** *StringA*, **L:=** *NumChar***)**

Where	
StringA (IN)	The string from which the characters are copied. Data type: STRING (0 to 255 characters).
NumChar (L)	Specifies the number of characters to copy. Data type: INT.
return (OUT)	The new string of characters. Data type: STRING.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

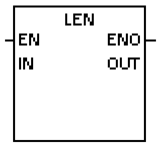
Example firstname := LEFT (IN:= fullname, L:= length);

If fullname = "john williams" and length = 4, then firstname = "john".

Length

Description Returns the length of a character string.

RLL



ST Function **LEN (*StringA*)**

IL Function **CALC LEN(OUT:=** *VarInt*, **IN:=** *StringA***)**

Where

StringA (IN)	The string for which the length is determined. Data type: STRING (0 to 255 characters).
return (OUT)	Contains the integer length of the string. Data type: INT.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example

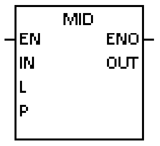
```
namelength := LEN(fullname);
```

If fullname = "john williams", then namelength = 13.

Middle

Description Returns a specified number of characters from the middle (at a specified position) of the input string.

RLL



ST Function **MID(IN:=** *StringA*, **L:=** *NumChar*, **P:=** *Position***)**

IL Function **CALC MID(OUT:=** *StringB*, **IN:=** *StringA*, **L:=** *NumChar*, **P:=** *Position***)**

Where	
StringA (IN)	The string from which characters are copied. Data type: STRING (0 to 255 characters).
NumChar (L)	Specifies the number of characters to copy. Data type: INT.
Position (P)	Specifies the position within the input string to begin copying characters. Data type: INT.
return (OUT)	The new string of characters. Data type: STRING.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

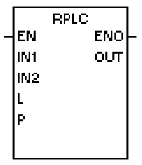
Example midname := MID (IN:= fullname, L:= 5, P:= 5);

If fullname = "ann marie williams", then midname = "marie".

Replace

Description Returns a string formed by replacing characters in one string (at a specified position) with a specified number of characters from another string.

RLL



ST Function **REPLACE(IN1:=** *StringA*, **IN2:=** *StringB*, **L:=** *NumChar*, **P:=** *Position***)**

IL Function **CALC REPLACE(OUT:=** *StringC*, **IN1:=** *StringA*, **IN2:=** *StringB*, **L:=** *NumChar*, **P:=** *Position***)**

Where

StringA (IN1)	The string in which characters are replaced. Data type: STRING (0 to 255 characters).
StringB (IN2)	The string from which the characters are copied. Data type: STRING (0 to 255 characters).
NumChar (L)	Specifies the number of characters to copy. Data type: INT.
Position (P)	Specifies the position within the string to begin replacing characters. Data type: INT.
return (OUT)	The result of the character replacement. Data type: STRING.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

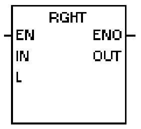
Example newname := REPLACE (IN1:= fullname, IN2:= newmid, L:= 7, L:= 8);

If fullname = "Thomas William Anderson", and newmid = "Alan", then newname = "Thomas Alan Anderson".

Right

Description Returns a specified number of the rightmost characters of the input string.

RLL



ST Function **RIGHT(IN:=** *StringA*, **L:=** *NumChar***)**

IL Function **CALC RIGHT(OUT:=** *StringB*, **IN:=** *StringA*, **L:=** *NumChar***)**

Where

StringA (IN)	The string from which the characters are copied. Data type: STRING (0 to 255 characters).
NumChar (L)	Specifies the number of characters to copy. Data type: INT.
return (OUT)	The new string of characters. Data type: STRING.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example lastname := RIGHT(IN:= fullname, L:= length);

If fullname = "john williams", and length = 8, then lastname = "williams".

Comparison

Introduction

Comparison functions include:

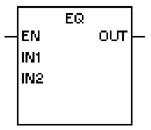
Equal (EQ)	Tests two inputs for equality.
Greater Than or Equal (GE)	Tests if first input is greater than or equal second input.
Greater Than (GT)	Tests if first input is greater than second input.
Less Than or Equal (LE)	Tests if first input is less than or equal second input.
Less Than (LT)	Tests if first input is less than second input.
Not Equal (NE)	Tests two inputs for inequality.

Note: In the RLL Editor, the comparison function's OUT is the rung output, not ENO.

Equal (EQ)

Description Returns a Boolean TRUE if the inputs are equal; otherwise, returns FALSE. Sets the RLL rung output accordingly.

RLL



ST Function **EQ(***Any1***,** *Any2***)**

ST Operator **out :=** *Any1* **=** *Any2***;**

IL Function **CALC EQ(***Any1***,** *Any2***)**

Where

Any1, *Any2* (IN1, IN2)	The values to be compared. Data type: ANY.
return (OUT)	The result of the comparison. The rung output in RLL Data type: BOOL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example

result := EQ(a, b);

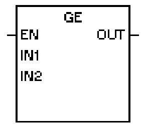
result := a=b;

In either case, if a, b are equal to the same value, result will be TRUE.

Greater Than or Equal (GE)

Description	Returns a Boolean TRUE if the first input is greater than or equal to the second input; otherwise, returns FALSE. Sets the RLL rung output accordingly.

RLL



ST Function **GE(***Any1*, *Any2***)**

ST Operator **out :=** *Any1* >= *Any2***;**

IL Function **CALC GE(***Any1*, *Any2***)**

Where	
Any1, *Any2* (IN1, IN2)	The values to be compared. Data type: ANY.
return (OUT)	The result of the comparison. The rung output in RLL Data type: BOOL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example

result := GE(a, b);

result := a>=b;

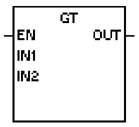
If a=5, b=4, then result will be TRUE.

If a=5, b=5 then result will be TRUE.

Greater Than (GT)

Description Returns a Boolean TRUE if the first input is greater than the second input; otherwise, returns FALSE. Sets the RLL rung output accordingly.

RLL



ST Function **GT(***Any1*, *Any2***)**

ST Operator **out :=** *Any1* > *Any2***;**

IL Function **CALC GT(***Any1*, *Any2***)**

Where

Any1, *Any2* (IN1, IN2)	The values to be compared. Data type: ANY.
return (OUT)	The result of the comparison. The rung output in RLL Data type: BOOL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example

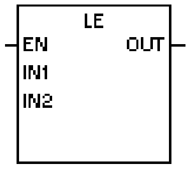
```
result := GT(a, b);

result := a>b;
```

If a=5, b=4, then result will be TRUE.

Less Than or Equal (LE)

Description Returns a Boolean TRUE if the first input is less than or equal to the second input; otherwise, returns FALSE. Sets the RLL rung output accordingly.

RLL



ST Function **LE(***Any1*, *Any2***)**

ST Operator **out :=** *Any1* = *Any2*;

IL Function **CALC LE(***Any1*, *Any***)**

Where	
Any, *Any2* (IN1, IN2)	The values to be compared. Data type: ANY.
return (OUT)	The result of the comparison. The rung output in RLL Data type: BOOL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example

result := LE(a, b);

result := a<=b;

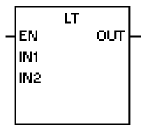
If a=1, b=2, then result will be TRUE.

If a=5, b=5, then result will be TRUE.

Less Than (LT)

Description	Returns a Boolean TRUE if the first input is less than the second input; otherwise, returns FALSE. Sets the RLL rung output accordingly.

RLL



ST Function **LT(***Any1*, *Any2***)**

ST Operator **out :=** *Any1* **<** *Any2***;**

IL Function **CALC LT(***Any1*, *Any2***)**

Where	
Any1, *Any2* (IN1, IN2)	The values to be compared. Data type: ANY.
return (OUT)	The result of the comparison. The rung output in RLL Data type: BOOL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example

result := LT(a, b);

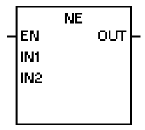
result := a<b);

If a=1, b=2, then result will be TRUE.

Not Equal (NE)

Description Returns a Boolean TRUE if the inputs are not equal; otherwise, returns FALSE. Sets the RLL rung output accordingly.

RLL



ST Function **NE(***Any1*, *Any2***)**

ST Operator **out :=** *Any1* **<>** *Any2***;**

IL Function **CALC NE(***Any1*, *Any2***)**

Where	
Any1, *Any2* (IN1, IN2)	The values to be compared. Data type: ANY.
return (OUT)	The result of the comparison. The rung output in RLL Data type: BOOL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example

```
result := NE(a, b);

result := a<>b;
```

If a=1 and b=5, then result will be TRUE.

Conversion

Introduction

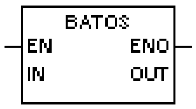
Conversion functions include:

Byte Array to String (BATOS)	Converts a byte array to a STRING value.
Date to String (DateToString)	Converts a DATE to a STRING value.
Integer to String (ITOA)	Converts an INT to a STRING value.
R2INT	Converts a REAL to the rounded INT value.
Real to String (RTOA)	Converts a REAL to a STRING value.
RGB to DWORD	Converts a triplet of red, green and blue values to a DWORD.
String to Byte Array (STOBA)	Converts a STRING to a byte array.
String to Date (StringToDate)	Converts a STRING to DATE.
String to Integer (ATOI)	Converts an ASCII numeric STRING to an integer value.
String to Real (ATOR)	Converts an ASCII numeric STRING to a real value.
String to Time of Day (StringToTOD)	Converts a STRING to a TOD.
Time of Day to String (TODToString)	Converts a TOD to a STRING value.
Trunc	Converts a REAL to the truncated INT value.

Byte Array to String (BATOS)

Description Converts an input array of bytes to ASCII characters and stores them to a string output. The terminating byte must contain zero.

RLL



ST Function **ARRAY_TO_STRING(OUT :=** *StringVariable*, **IN :=** *ByteArray*)

IL Function **CALC ARRAY_TO_STRING(OUT:=***StringVariable***,IN:=***ByteArray***)**

Where

StringVariable (OUT)	Contains the result of the conversion of the array of bytes to a string. Data type: STRING.
ByteArray (IN)	Specifies the array of bytes to be converted. Each byte in the array contains a decimal ASCII character code.
return	None.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example ARRAY_TO_STRING (OUT:=string1, IN := byte_array);

If byte_array consists of four bytes, and byte_array[1] = 65, byte_array[2] = 66, byte_array[3] = 67, and byte_array[4] = 0, then string1 contains ABC.

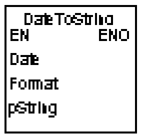
The following is an example of the BATOS function block:

Byte to String Operation:
IN = name_data
OUT = operator_name
If name_data consists of four bytes, and
byte_array[1] = 83
byte_array[2] = 65
byte_array[3] = 77
byte_array[4] = 0
then
operator_name = SAM

Date to String (DateToString)

Description Converts a DATE to a STRING value. A pointer variable is used to point to the string.

RLL



ST Function **DateToString (***Date*, *Format*, *pString***)**

IL Function **CALC DateToString(***Date*,*Format*,*pString***)**

Where	
Date	DATE variable to be converted. Data type: DATE.
Format	Date format: 0 = DD/MM/YY format; 1 = Month DD, Year format. Data type: INT.
pString	Pointer to the result of DATE conversion. Data type: Pointer. **Note:** A STRING symbol prefixed with the pointer reference operator can also be used (e.g., &SymString). If a STRING symbol is entered in the RLL dialog box, the pointer reference operator (&) is automatically prefixed to the symbol when you click OK.
return (OUT)	None.

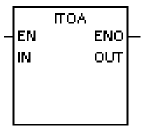
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.
3. Pointer functionality must be supported in the product in order to use this function.

Integer to String (ITOA)

Description Converts an integer to an ASCII character string representation of the value of the integer and returns the result.

RLL



ST Function **INT_TO_STRING (***AnyInt***)**

IL Function **CALC INT_TO_STRING(OUT:=***VarString***,IN:=***AnyInt***)**

Where	
AnyInt (IN)	The integer to be converted. Data type: INT.
return (OUT)	The result of the conversion of the integer to a string. Data type: STRING.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example

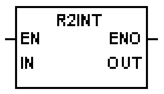
```
intstring := INT_TO_STRING (num);
```

If num = 11, then intstring contains the ASCII character string '11'.

Real to Integer (R2INT)

Description Rounds a REAL value to the nearest integer value and returns the result.

RLL



ST Function **R2INT (*AnyReal*)**

IL Function **CALC R2INT (OUT:=*VarInt*, IN:=*AnyReal*)**

Where	
AnyReal (IN)	The real value to be rounded. Data type: REAL.
return (OUT)	The result of the rounding of the real to an integer. Data type: INT.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

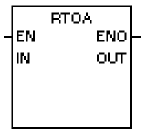
Example

```
IntVar := R2INT (1.6); (* Result is 2 *)

IntVar := R2INT (1.4); (* Result is 1 *)

IntVar := R2INT (1.5); (* Result is 2 *)
```

Real to String (RTOA)

Description Converts a real number to an ASCII character string representation of the value of the number and returns the result.

RLL



ST Function **REAL_TO_STRING (***AnyReal* **)**

IL Function **CALC REAL_TO_STRING(OUT:=***VarString***,IN:=***AnyReal***)**

Where

AnyReal (IN)	The real to be converted. Data type: REAL.
return (OUT)	The result of the conversion of the real to a string. Data type: STRING.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

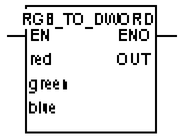
Example realstring := REAL_TO_STRING (num);

If num = 13.2288, then realstring contains the ASCII character string '13.2288'.

RGB to DWORD

Description Converts a triplet of red, green and blue values to a DWORD. Used by the ActiveX control feature of the Operator Interface.

RLL



ST Function **RGB_TO_DWORD(***Red*, *Green*, *Blue***)**

IL Function

CALC RGB_TO_DWORD(OUT:= *DWord*, *Red*, *Green*, *Blue***)**
LD 1

Where

Red, *Green*, *Blue*	Integer values corresponding to the intensity of each color. Data type: INT.
return (OUT)	The result of the conversion of the RGB triplet to DWORD. Data type: DWORD.

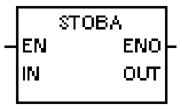
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

String to Byte Array (STOBA)

Description Converts each character in the input string to its decimal value and stores the result in a byte array. A zero is placed in the last byte of the array.

RLL



ST Function **STRING_TO_ARRAY(OUT :=** *ByteArray*, **IN :=** *StringVariable***)**

IL Function **CALC STRING_TO_ARRAY(OUT:=***ByteArray***,IN:=***StringVariable***)**

Where

ByteArray (OUT)	The result of the conversion of the string to an array of bytes. Data type: Any BYTE variable array.
StringVariable (IN)	The string of characters to be converted. Values are decimal codes. Data type: STRING.
return	None.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example 1 STRING_TO_ARRAY (OUT:= byte_array, IN:= string1);

If string1 contains "Sara", then the array consists of five bytes, and byte_array[0] = 83, byte_array[1] = 97, byte_array[2] = 114, byte_array[3] = 97, and byte_array[4] = 0.

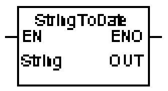
Example 2 The following is an example of the STOBA function block:

String to Byte Operation**:**
IN = operator_name
OUT = name_data
If operator_name = SAM,
then
name_data consists of four bytes, and
byte_array[1] = 83
byte_array[2] = 65
byte_array[3] = 77
byte_array[4] = 0

String to Date (StringToDate)

Description Converts a STRING to a DATE and returns the result.

RLL



ST Function **StringToDate(***String***)**

IL Function **CALC StringToDate(OUT:=***VarDate*, *String***)**

Where

String (IN)	The ASCII string to be converted. The form of the STRING input is DD/MM/YY or DD/MM/YYYY where all of the fields DD, MM and YY are required. Data type: STRING.
return (OUT)	The converted string. Data type: DATE.

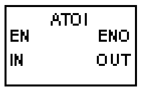
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

String to Integer (ATOI)

Description Converts an ASCII numeric string to its integer equivalent. Non-numeric ASCII characters or numeric characters following a non-numeric character are ignored.

RLL



ST Function **ATOI(***String***)**

IL Function **CALC ATOI(OUT:=** *VarInt*, *String***)**
LD 1

Where

String (IN)	The ASCII string to be converted. Data type: STRING.
return (OUT)	The converted integer value. Data type: INT. If the string begins with a non-numeric character, a zero (0) is returned.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

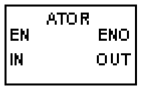
Examples

```
IntVar01 := ATOI("1234");      (*IntVar01 = 1234*)

IntVar02 := ATOI("ABC");       (*IntVar02 = 0*)

IntVar03 := ATOI("1A2B3C");    (*IntVar03 = 1*)
```

String to Real (ATOR)

Description Converts an ASCII numeric string (including decimal point) to its real equivalent. Non-numeric ASCII characters or numeric characters following a non-numeric character are ignored.

RLL



ST Function **ATOR(***String***)**

IL Function **CALC ATOR(OUT:=** *VarReal*, *String***)**
LD 1

Where

String (IN)	The ASCII string to be converted. Data type: STRING.
return (OUT)	The converted real value. Data type: REAL. If the string begins with a non-numeric or non-decimal point character, a zero (0) is returned.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

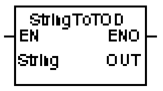
Examples

```
RealVar01 := ATOI("12.34");       (*RealVar01 = 12.34*)

RealVar02 := ATOI("ABC");         (*RealVar02 = 0*)

RealVar03 := ATOI("1.A2B3C");  (*RealVar03 = 1*)
```

String to TOD (StringToTOD)

Description Converts a STRING to a TOD and returns the result.

RLL



ST Function **StringToTOD(***String***)**

IL Function **CALC StringToDate(OUT:=***VarDate***,** *String***)**

Where	
String (IN)	The ASCII string to be converted. The form of the STRING input is HH:MM:SS where MM and SS could be omitted. Data type: STRING.
return (OUT)	The result of the conversion. Data type: TOD.

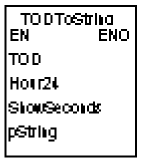
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Time of Day to String (TODToString)

Description Converts a TOD to a STRING value and places the result in the symbol pointed to by pString.

RLL



ST Function **TODToString (***TOD*, *Hour24*, *ShowSeconds*, *pString***)**

IL Function **CALC TODToString(***TOD*,*Hour24*,*ShowSeconds*,*pString***)**

Where

TOD	Time-of-day variable to be converted. Data type: TOD
Hour24	Display time in 24 hour format. Data type: INT.
ShowSeconds	Show seconds in time display. Data type: INT.
pString	A pointer to the symbol containing the string output of TOD conversion. Data type: Pointer. **Note:** A STRING symbol prefixed with the pointer reference operator can also be used (e.g., &SymString). If a STRING symbol is entered in the RLL dialog box, the pointer reference operator (&) is automatically prefixed to the symbol when you click OK.
return (OUT)	None.

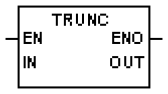
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.
3. Pointer functionality must be supported in the product in order to use this function.

Truncate (TRUNC)

Description Truncates a REAL value to an integer value and returns the result.

RLL



ST Function **TRUNC (***AnyReal***)**

IL Function **CALC TRUNC (OUT:=***VarInt***, IN:=***AnyReal***)**

Where	
AnyReal (IN)	The real value to be truncated. Data type: REAL.
return (OUT)	The result of truncating of the real to an integer. Data type: INT.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example

IntVar := TRUNC (1.6); (* Result is 1 *)

IntVar := TRUNC (1.4); (* Result is 1 *)

IntVar := TRUNC (2.4); (* Result is 2 *)

Counters and Timers

Introduction

Counter function blocks include:

Counter down (CTD)	Counts events by decrementing by one.
Counter up (CTU)	Counts events by incrementing by one.
Counter up/down (CTUD)	Counts events up or down.

Timer function blocks include:

Timer Off Delay (TOF)	Provides off-delay timing of events.
Timer On Delay (TON)	Provides on-delay timing of events.
Timer Pulse (TP)	Activated by a pulse, provides off-delay timing of events.

Also refer to **Extended Timers**.

Using Counter and Timer Function Blocks

Warning
Assigning the same function block name to different counters and/or timers may cause unpredictable operation by the controller, which can result in death or injury to personnel and/or damage to equipment. Always use a unique name for each counter or timer.

When the program is running, you can view the current value of all function block inputs and outputs. To do this, click the Function Block Details icon or double click on the instruction to display a dialog box. You can also open a watch window and enter the counter or timer variables that you want to observe at run-time.

You can use any of the counter or timer inputs and outputs in any expression, contact, or coil instead of a symbol of the same type. However, they are local variables and cannot be used in DDE applications. To reference a counter or timer input or output, enter the function block name followed by a period and the specific input or output suffix.

For example:

- CTD1.CU refers to the count up input of CTD1.
- TOF5.Q refers to the rung output of TOF5

For more information about a function block's inputs and outputs, refer to the explanation of a specific function block.

Setting a Timer Preset

To set the preset time value (PT), you can enter the time directly or use the Define Time Duration dialog box.

To enter the duration directly:

Follow the IEC 1131-3 specification: a keyword, e.g., T#, TIME#, t#, time#, followed by time in days, hours, minutes, seconds. Examples are shown below.

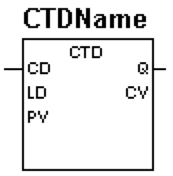
Time	Format	Time	Format
14.7 days	T#14.7d	4 seconds	Time#4s
2 minutes 5 seconds	T#2m5s	1 day 29 minutes	t#1d29m
74 minutes*	time#74m	1 hour 5 seconds 44 milliseconds	T#1h5s44ms

*The IEC 1131-3 specification allows overflow of the most significant unit in a duration.

Count Down (CTD)

Description Counts from a preset value down to zero. The output (Q) goes TRUE when the count equals zero. It can be used, for example, to count recurring events.

RLL



ST Function

CTDName.**CD :=** *CountDown*;	(*Assign inputs*)
CTDName.**LD :=** *Load*;	
CTDName.**PV :=** *PresetVal*;	
CTDName **( );**	(*Call*)
Output **:=** *CTDName*.**Q;**	(*Access outputs*)
CurrentVal **:=** *CTDName*.**CV;**	

Where

CTDName	Unique name for the counter.
CountDown (CD)	Decrements the counter when CD transitions from FALSE to TRUE. Data type: BOOL. Rung input for RLL.
Load (LD)	When TRUE, loads the counter with the preset value (PV). Data type: BOOL.
PresetVal (PV)	The count value at which the CTD begins to count. Data type: WORD (0-65535).
Output (Q)	TRUE when the current value (CV) <= zero. Rung output for RLL.
CurrentVal (CV)	The current count value of the counter. Data type: WORD (0-65535).
Enable (EN)	Enables the counter. Data type: BOOL. The default value is TRUE.
Enable Output (ENO)	Echoes EN. Data type: BOOL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using function blocks with the Instruction List language.
3. Refer to **Structured Text** for information on using function blocks with the Structured Text language.

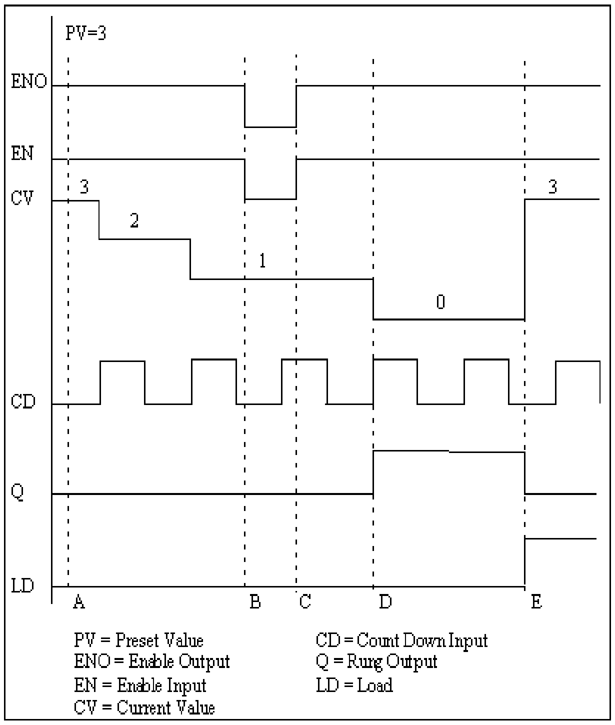
Operation

- The counter is enabled while EN is TRUE.
- While EN is TRUE, the counter decrements CV by one each time CD transitions from FALSE to TRUE.
- When CV <= zero, the rung output Q is set to TRUE.
- When EN is FALSE, the counter freezes Q and CV in their current states until EN is set to TRUE again.
- When EN is TRUE and LD is set to TRUE, the counter sets CV to the preset value PV and sets output Q FALSE.
- ENO echoes the value of EN.

CTD Variables You can reference the CTD counter variables by prefixing the variable with the counter instance name (*CTDName*) as follows:

CTDName.**CD**
CTDName.**LD**
CTDName.**PV**
CTDName.**Q**
CTDName.**CV**
CTDName.**EN**
CTDName.**ENO**

Example An example operation of the CTD is shown in the following timing diagram.



A. PV has been previously loaded with the preset value of 3.
EN has been previously set to TRUE.
CV contains the current value of 3.

B. EN transitions from TRUE to FALSE, disabling the counter.
CV, which had been counting down as CD changed state, holds at 1.

C. EN transitions from FALSE to TRUE, re-enabling the counter.
CV resumes counting down at the next FALSE-to-TRUE transition of CD, and reaches 0.

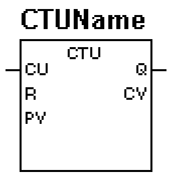
D. Q transitions to TRUE when CV=0.

E. LD transitions from FALSE to TRUE, which loads the preset value of 3 into CV. CV holds at 3 while LD is TRUE.

Count Up (CTU)

Description Counts from zero up to a preset value. The output (Q) goes TRUE when the count equals the preset count. It can be used, for example, to count recurring events.

RLL



ST Function

CTUName.**CU :=** *CountUp*;	(*Assign inputs*)
CTUName.**R :=** *Reset*;	
CTUName.**PV :=** *PresetVal*;	
CTUName **( );**	(*Call*)
Output **:=** *CTUName*.**Q;**	(*Access outputs*)
CurrentVal **:=** *CTUName*.**CV;**	

Where

CTUName	Unique name for the counter.
CountUp (CU)	Increments the counter when CU transitions from FALSE to TRUE. Data type: BOOL. Rung input for RLL.
Reset (R)	When TRUE, resets the counter (resets CV to 0). Data type: BOOL.
PresetVal (PV)	The value up to which the CTU counts. Data type: WORD (0-65535).
Output (Q)	TRUE when CV >= PV. Data type: BOOL. Rung output for RLL.
CurrentValue (CV)	The current count value of the counter. Data type: WORD (0-65535).
Enable (EN)	Enables the counter. Data type: BOOL. Default is TRUE.
Enable Output (ENO)	Echoes EN. Data type: BOOL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using function blocks with the Instruction List language.
3. Refer to **Structured Text** for information on using function blocks with the Structured Text language.

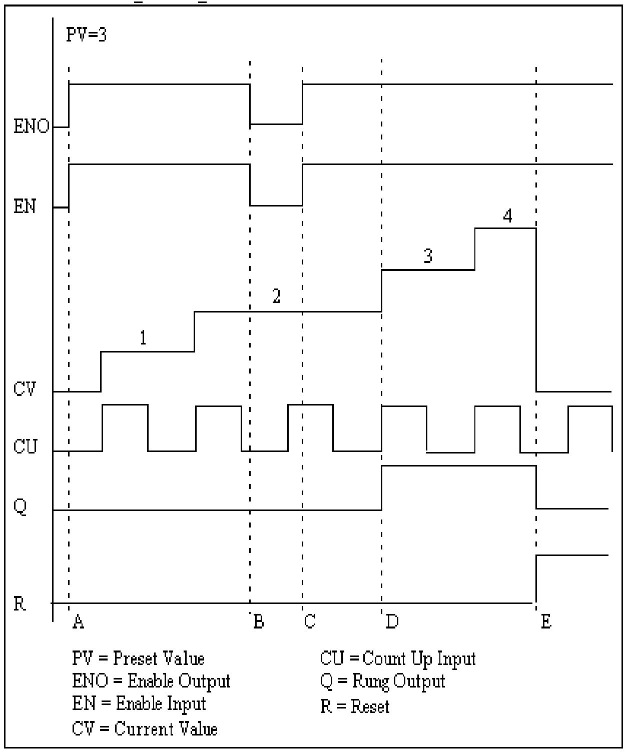
Operation

- The counter is enabled when EN is TRUE.
- While EN is TRUE, the counter increments CV by one each time CU transitions from FALSE to TRUE.
- When CV is >= PV, the counter sets rung output Q to TRUE.
- When EN is FALSE, the counter freezes Q and CV in their current states until EN is set to TRUE again.
- When EN is TRUE and R is TRUE, CV is set to zero and Q is set to FALSE.
- ENO echoes the value of EN.

CTU Variables You can reference the CTU counter variables by prefixing the variable with the counter instance name (*CTUName*) as follows:

CTUName.**CU**
CTUName.**R**
CTUName.**PV**
CTUName.**Q**
CTUName.**CV**
CTUName.**EN**
CTUName.**ENO**

Example An example operation of the CTU is shown in the following timing diagram.



A. PV has been previously loaded with the preset value of 3.
   EN and ENO transition from FALSE to TRUE.
   CV contains the current value of 0.

B. EN transitions from TRUE to FALSE, disabling the counter.
   CV, which had been counting up as CU changed state, holds at 2.

C. EN transitions from FALSE to TRUE, re-enabling the counter.
   CV resumes counting up at the next FALSE-to-TRUE transition of CU, and reaches 3. CV continues counting up until reset by R.

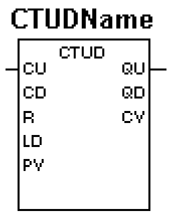
D. Q transitions to TRUE when CV=3.

E. R transitions from FALSE to TRUE, which resets CV to 0. CV holds at 0 while R is TRUE.

Count Up/Down (CTUD)

Description Counts up or down, setting an up-count output when the current count is greater than or equal to the preset count, or a down-count output when the current count is less than or equal to zero.

RLL



ST Function

CTUDName.**CU** **:=** *CountUp*; (*Assign inputs*)
CTUDName.**CD** **:=** *CountDn*;
CTUDName.**R** **:=** *Reset*;
CTUDName.**LD** **:=** *Load*;
CTUDName.**PV** **:=** *PresetVal*;
CTUDName **( )** (*Call*)
CountUpOutput **:=** *CTUDName*.**QU;** (*Access outputs*)
CountDnOutput **:=** *CTUDName*.**QD;**
CurrentValue **:=** *CTUDName*.**CV;**

Where

CTUDName	Unique name for the counter.
CountUp (CU)	Increments current value CV when CU transitions from FALSE to TRUE. Data type: BOOL. Rung input for RLL.
CountDn CD)	Decrements current value CV when CD transitions from FALSE to TRUE. Data type: BOOL.
Reset (R)	Sets current count value CV to zero and sets the count-up output QU to FALSE. Data type: BOOL.
Load (LD)	Sets current count value CV to the preset value PV and sets the count-down output QD to FALSE. Data type: BOOL.
PresetVal (PV)	The value to which the CTUD counts up, or at which the CTUD begins to counts down. Data type: WORD (0-65535).
CountUpOutput (QU)	TRUE when current value CV >= preset value PV. Rung output for RLL.

CountDnOutput (QD)	TRUE when current value CV <= zero. Data type: BOOL.
CurrentValue (CV)	The current count value of the counter. Data type: WORD (0-65535).
Enable (EN)	Enables the counter. Data type: BOOL.
Enable Output (ENO)	Echoes EN. Data type: BOOL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using function blocks with the Instruction List language.
3. Refer to **Structured Text** for information on using function blocks with the Structured Text language.

Operation

Operation for an up-count:

- The counter is enabled when EN is TRUE.
- When EN and R are TRUE, CV is set to zero and QU is set to FALSE. (Note: If PV is 0, then QU is TRUE.)
- While EN is TRUE and R is FALSE, the counter increments CV by one each time the count up input CU transitions from FALSE to TRUE.
- When CV is greater than or equal to PV, the counter sets rung output QU to TRUE. QU is FALSE when CV is less than PV.
- When EN is FALSE, the QU and CV are frozen in their current states until EN is TRUE again. Toggling R does not affect CV or QU while EN is FALSE.
- ENO echoes the value of EN.

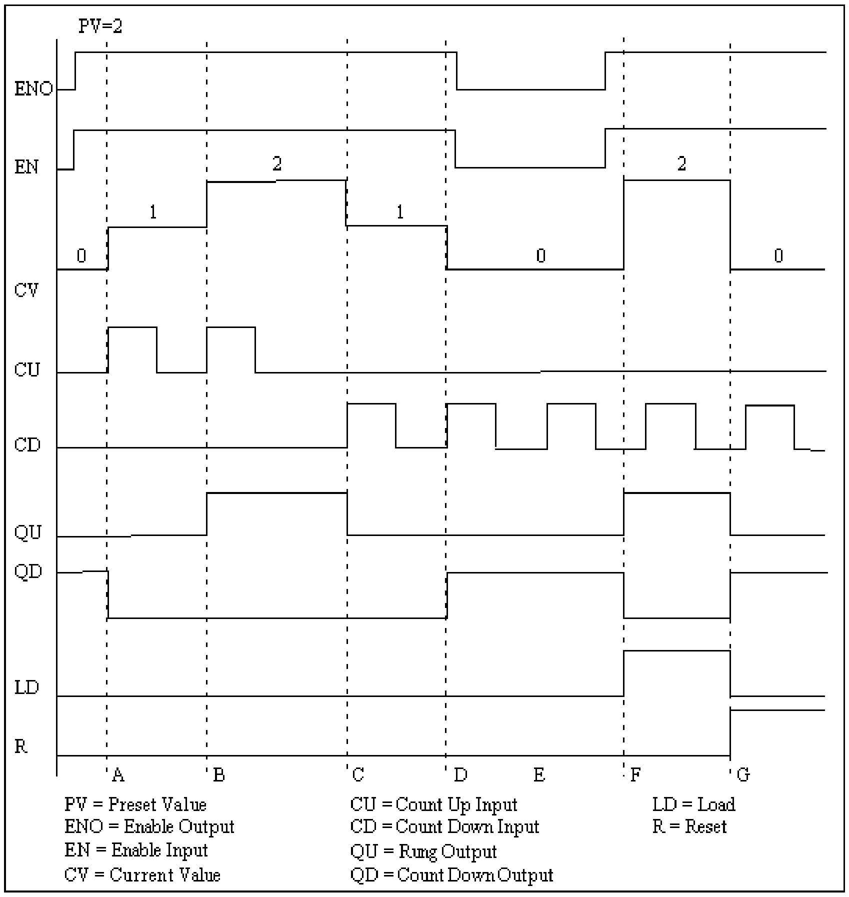
Operation for down-count:

- The counter is enabled when EN is TRUE.
- When EN and LD are TRUE and R is FALSE, the counter sets CV to the preset value PV and sets QD to FALSE. (Note: If PV is 0, then QD is TRUE.)
- While EN is TRUE and LD is FALSE, the counter decrements the current value CV by one each time CD transitions from FALSE to TRUE.
- When CV is less than or equal to zero, the counter sets QD to TRUE. QD is FALSE when CV is greater than zero.
- When EN is FALSE, the counter freezes QD and CV in their current states until EN is set to TRUE again. Toggling LD does not effect CV or QD while EN is FALSE or while R is TRUE.
- ENO echoes the value of EN.

Variables You can reference the CTUD counter variables by prefixing the variable with the counter instance name (*CTUDName*) as follows:

CTUDName.**CU**
CTUDName.**CD**
CTUDName.**R**
CTUDName.**LD**
CTUDName.**PV**
CTUDName.**QU**
CTUDName.**QD**
CTUDName.**CV**
CTUDName.**EN**
CTUDName.**ENO**

Example An example operation of the CTUD is shown in the following timing diagram.



A. PV has been previously loaded with the preset value of 2.
EN and ENO transition from FALSE to TRUE.
QU is FALSE and QD is TRUE because CV is 0.

B. CV, which had been counting up as CU changed state, reaches 2.
QU transitions to TRUE.

C. CV decrements to 1 when CD transitions from FALSE to TRUE.
QU transitions to FALSE.

D. CV decrements to 0 at the next transition of CD from FALSE to TRUE.
QD transitions to TRUE.

E. After point D, EN transitions from TRUE to FALSE, disabling the counter. CV, QU, and QD are frozen in their current states. Prior to point E, EN is set TRUE, enabling the counter.

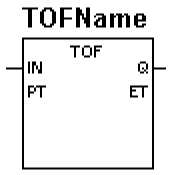
F. LD transitions from FALSE to TRUE.
The preset value of 2 is loaded to CV, QD is set to FALSE, and QU is set to TRUE.

G. R transitions from FALSE to TRUE.
CV is set to 0, QU is set to FALSE, and QD is set to TRUE.

Timer Off Delay (TOF)

Description Turns off an output after a preset time delay.

RLL



ST Function

```
TOFName.IN := Start;              (*Assign inputs*)
TOFName.PT := PresetTime;
TOFName ( );                      (*Call*)
Output := TOFName.Q;              (*Access outputs*)
ElapsedTime := TOFName.ET;
```

Where

TOFName	Unique name for the timer.
Start (IN)	Starts the timer. Data type: BOOL. Rung input in RLL.
PresetTime (PT)	Specifies the timer period. Data type: TIME.
Output (Q)	FALSE when timer times out. Data type: BOOL.. Rung output in RLL.
ElapsedTime (ET)	The current elapsed time. Data type: TIME.
Enable (EN)	Enables the timer. Data type: BOOL.
Enable Output (ENO)	Echoes EN. Data type: BOOL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using function blocks with the Instruction List language.
3. Refer to **Structured Text** for information on using function blocks with the Structured Text language.

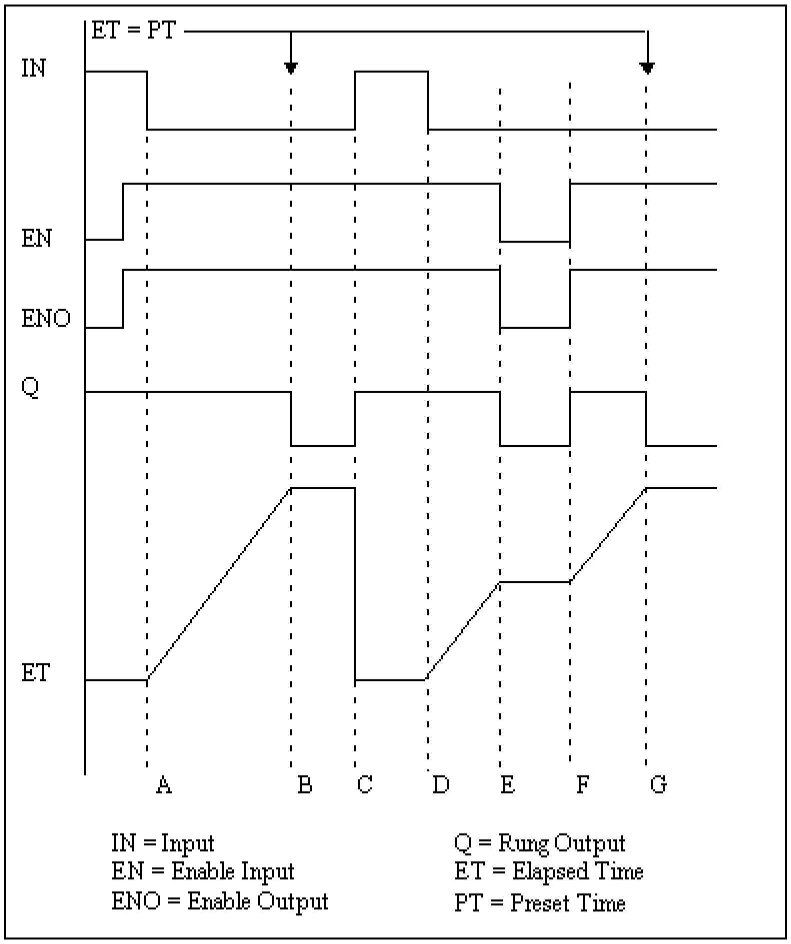
Operation

- When EN is TRUE the timer is enabled.
- When IN is FALSE, the timer is active, storing the elapsed time in ET.
- When the elapsed time ET equals preset time PT, TOF sets the rung output Q to FALSE.
- When EN is set to FALSE, TOF freezes ET in its current state and sets Q to zero. When EN is set TRUE again, Q is restored to its previous value.
- When IN is set TRUE, ET is reset to zero and Q is set to FALSE. (Note: If PT is zero, then Q is TRUE.)
- ENO echoes the value of EN.

TOF Variables You can reference the TOF timer variables by prefixing the variable with the counter instance name (*TOFName*) as follows:

*TOFName***.IN**
*TOFName***.PT**
*TOFName***.Q**
*TOFName***.ET**
*TOFName***.EN**
*TOFName***.ENO**

Example An example operation of the TOF is shown in the following timing diagram.



A. EN and ENO have been previously set to TRUE.

B. IN transitions from TRUE to FALSE and ET indicates that the timer has begun timing. When ET equals PT, Q transitions from TRUE to FALSE.

C. IN transitions from FALSE to TRUE.
Q is set to TRUE and ET is set to zero.

D. IN transitions from TRUE to FALSE and ET indicates that the timer has begun timing.

E. EN transitions from TRUE to FALSE, disabling the timer.
ET, which had been timing, holds at its current value, and Q is set to FALSE.

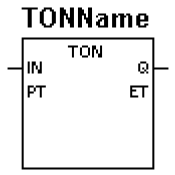
F. EN transitions from FALSE to TRUE, re-enabling the timer.
ET resumes timing, and Q is set to TRUE.

G. When ET equals PT, Q transitions from TRUE to FALSE.

Timer On Delay (TON)

Description Turns on an output after a preset time delay.

RLL



ST Function

TONName.**IN** := *Start*;	(*Assign inputs*)
TONName.**PT** := *PresetTime*;	
TONName **( );**	(*Call*)
Output **:=** *TONName*.**Q;**	(*Access outputs*)
ElapsedTime **:=** *TONName*.**ET;**	

Where

TONName	Unique name for the timer.
Start (IN)	Starts the timer. Data type: BOOL. Rung input in RLL.
PresetTime (PT)	The timer period. Data type: TIME.
Output (Q)	TRUE when timer times out. Rung output in RLL.
ElapsedTime (ET)	The current elapsed time. Data type: TIME.
Enable (EN)	Enables the timer. Data type: BOOL.
Enable Output (ENO)	Echoes EN. Data type: BOOL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.
3. Refer to **Structured Text** for information on using function blocks with the Structured Text language.

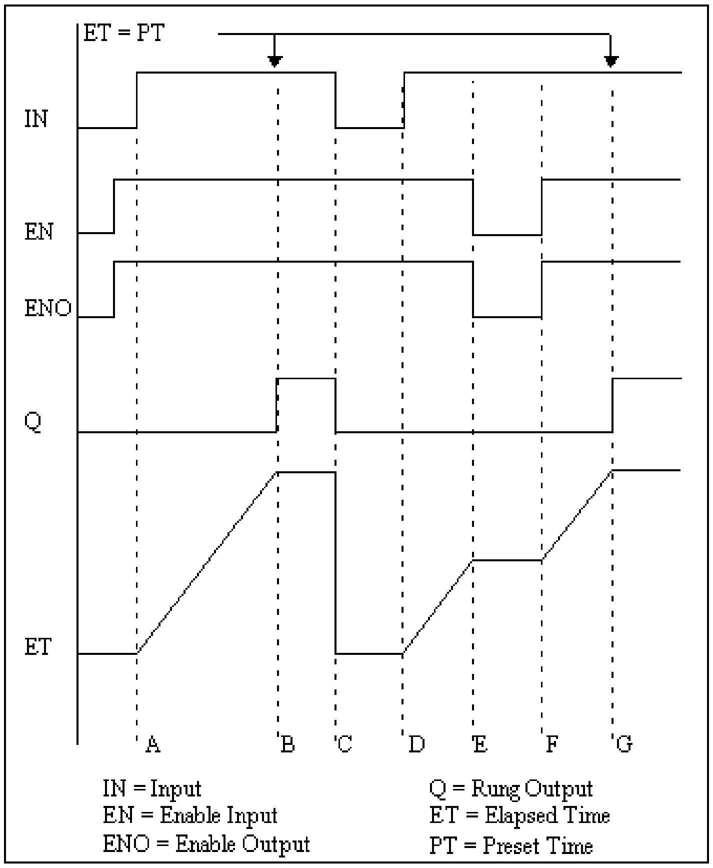
Operation

- When the enable input EN is TRUE, the timer is enabled.
- When input IN transitions to TRUE, the timer is active, storing the elapsed time in output ET.
- When the elapsed time ET equals preset time PT, TON sets the rung output Q to TRUE.
- When EN is set to FALSE, TON freezes ET in its current state and sets Q to zero. When EN is set to TRUE again, Q is restored to its original value.
- If input IN becomes FALSE, the system resets ET to zero and sets output Q to FALSE.
- Enable output ENO echoes the value of EN.

TON Variables You can reference the TON timer variables by prefixing the variable with the counter instance name (*TONName*) as follows:

> *TONName*.**IN**
> *TONName*.**PT**
> *TONName*.**Q**
> *TONName*.**ET**
> *TONName*.**EN**
> *TONName*.**ENO**

Example An example operation of the TON is shown in the following timing diagram.



A. EN and ENO have been previously set to TRUE.

- When IN transitions from FALSE to TRUE, ET indicates that the timer has begun timing.

B. ET equals PT and Q transitions from FALSE to TRUE.

C. When IN transitions from TRUE to FALSE, ET and Q are set to zero.

D. When IN transitions from FALSE to TRUE, ET indicates that the timer has begun timing.

E. When EN transitions from TRUE to FALSE, the timer is disabled.

- ET, which had been timing, holds at its current value, and Q is set to FALSE.

F. When EN transitions from FALSE to TRUE, the timer is re-enabled.

- ET resumes timing.

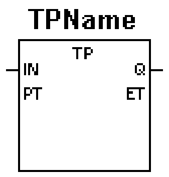
G. ET equals PT, and Q transitions from FALSE to TRUE.

Timer Pulse (TP)

Description

The TP function block times the duration of an event. After its input pulses from off to on, the TP keeps time to the preset interval and sets an output FALSE, which makes the TP an off-delay timer.

RLL



ST Function

```
TPName.IN := Start;               (*Assign inputs*)
TPName.PT := PresetTime;
TPName ( );                       (*Call*)
Output := TPName.Q;               (*Access outputs*)
ElapsedTime := TPName.ET;
```

Where

TPName	Unique name for the timer.
Start (IN)	Starts the timer. Data type: BOOL. Rung input in RLL.
PresetTime (PT)	Specifies the period for which the timer times. Data type: TIME.
Output (Q)	Changes to FALSE when timer times out. Data type: BOOL. Rung output in RLL.
ElapsedTime (ET)	Contains the current elapsed time. Data type: TIME.
Enable (EN)	Enables the timer. Data type: BOOL.
Enable Output (ENO)	Echoes EN. Data type: BOOL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using function blocks with the Instruction List language.
3. Refer to **Structured Text** for information on using function blocks with the Structured Text language.

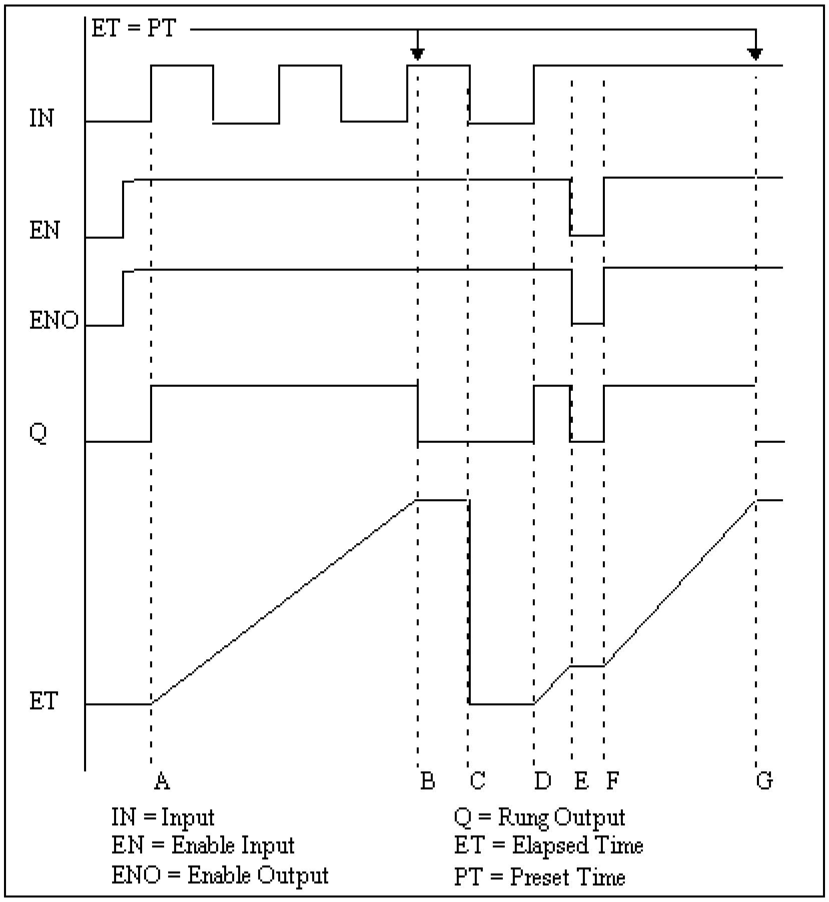
Operation

- When the enable input EN is TRUE the timer is enabled.
- When input IN transitions to TRUE, the timer increments, storing the elapsed time in output ET. The timer continues to time up, even if IN transitions to FALSE. Therefore, a FALSE-to-TRUE pulse can start a timer that is enabled.
- When the elapsed time ET equals preset time PT, TP sets the rung output Q to FALSE.
- If enable input EN becomes FALSE, TP freezes CV in its current state and sets Q to FALSE. When EN becomes TRUE again, Q is restored to its previous value.
- ENO echoes the value of EN.

TP Variables You can reference the TP timer variables by prefixing the variable with the counter instance name (*TPName*) as follows:

TPName.**IN**
TPName.**PT**
TPName.**Q**
TPName.**ET**
TPName.**EN**
TPName.**ENO**

Timing Diagram An example of the TP timing diagram is shown in the following figure.



A. EN and ENO have been previously set to TRUE. IN transitions from FALSE to TRUE, Q is set to TRUE, and ET indicates that the timer has begun timing. While the timer is timing, IN can toggle without affecting ET or Q.

B. ET equals PT and Q transitions from TRUE to FALSE.

C. IN transitions from TRUE to FALSE. ET is set to zero.

D. IN transitions from FALSE to TRUE, Q is set to TRUE, and ET indicates that the timer has begun timing.

E. EN transitions from TRUE to FALSE, disabling the timer. ET, which had been timing, holds at its current value, and Q is set to FALSE.

F. EN transitions from FALSE to TRUE, re-enabling the timer. ET resumes timing, and Q is set to TRUE.

G. ET equals PT and Q transitions from TRUE to FALSE.

Edge Detection

Introduction

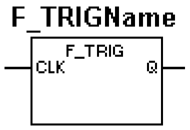
Edge detection function blocks include:

Falling edge trigger (F_TRIG)	Turns on an output when triggered by a falling edge trigger.
Rising edge trigger (R_TRIG)	Turns on an output when triggered by a rising edge trigger.

Falling Edge Trigger (F_TRIG)

Description The F_TRIG function block sets an output to TRUE for one scan when the input to the function block transitions from TRUE to FALSE.

RLL



ST Function

FtrigName.**CLK :=** *Clock*;	(*Assign input*)
FTrigName **( );**	(*Call*)
Output **:=** *FTrigName*.**Q;**	(*Access output*)

Where

FTrigName	Unique name for the function block.
Clock (CLK)*	Enables function input (BOOL) to execute when it transitions from TRUE to FALSE. Data type: BOOL.
Output (Q)*	Output is set to TRUE when CLK transitions from TRUE to FALSE. Data type: BOOL.

* In Relay Ladder Logic, although CLK and Q are Boolean data types, the only way to access them is through the rung.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using function blocks with the Instruction List language.
3. Refer to **Structured Text** for information on using function blocks with the Structured Text language.

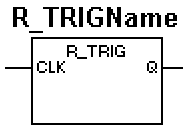
Operation When the input to the FTRIG transitions from TRUE to FALSE, the FTRIG sets output Q to TRUE for one scan.

You can use the FTRIG input and output in any expression, contact or coil instead of a symbol of the same type. However, they are local variables and cannot be used in DDE applications. To reference an FTRIG input or output, enter the function block name followed by a period and the specific input or output suffix. For example, FTRIG1.Q refers to the output of FTRIG1.

Rising Edge Trigger (R_TRIG)

Description The RTRIG function block sets an output to TRUE for one scan when the input to the function block transitions from FALSE to TRUE.

RLL



ST Function

*RTrigName***.CLK :=** *Clock***;**	(*Assign input*)
RTrigName **( );**	(*Call*)
Output **:=** *RTrigName* **.Q;**	(*Access output*)

Where

RTrigName	Unique name for the function block.
Clock (CLK)*	Rung input enables the function block rung input (BOOL) to execute when it transitions from FALSE to TRUE.
Output (Q)*	Rung output is set to TRUE when CLK transitions from FALSE to TRUE.

* In Relay Ladder Logic, although CLK and Q are Boolean data types, the only way to access them is through the rung.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using function blocks with the Instruction List language.
3. Refer to **Structured Text** for information on using function blocks with the Structured Text language.

Operation When the input to the RTRIG transitions from FALSE to TRUE, the RTRIG sets output Q to TRUE for one scan.

You can use the RTRIG input and output in any expression, contact or coil instead of a symbol of the same type. However, they are local variables and cannot be used in DDE applications. To reference an RTRIG input or output, enter the function block name followed by a period and the specific input or output suffix. For example, RTRIG1.Q refers to the output of RTRIG1.

Extended PID

Introduction

There is one extended PID function block:

Extended PID (EX_PID)	Provides automatic closed-loop operation of continuous process control loops.

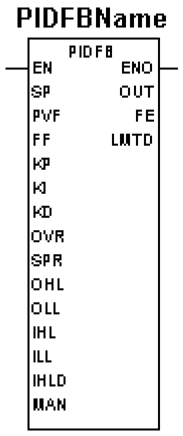
Also refer to the PID system object PID Loop Control (PID).

EX_PID

Description A PID is an instruction providing automatic closed-loop operation of continuous process control loops. For each loop the instruction performs proportional control and optionally integral control, derivative control, or both:

- Proportional control - causes an output signal to change as a direct ratio of the error signal variation.
- Integral control - causes an output signal to change as a function of the integral of error signal over the time duration.
- Derivative control - causes an output signal to change as a function of the rate of change of the error signal.

RLL



ST Function

```
PIDFBName.SP:= SetPoint;                        (*Assign inputs*)
PIDFBName.PVF:= ProcessVariable;
PIDFBName.FF:= FeedForward;
PIDFBName.KP:= PGain;
PIDFBName.KI:= IGain;
PIDFBName.KD:= DGain;
PIDFBName.OVR:= ManOverride;
PIDFBName.SPR:= SPRamping;
PIDFBName.OHL:= HighOutLimit;
PIDFBName.OLL:= LowOutLimit;
PIDFBName.IHL:= HighIntegralLimit;
PIDFBName.ILL:= LowIntegralLimit;
PIDFBName.IHLD:= IntegralHold;
PIDFBName.MAN:= AutoManual;
PIDFBName( );                              (*Call*)
PIDOut:= PIDFBName.OUT;                    (*Access Outputs*)
PIDFE:= PIDFBName.FE;
PIDLimited:= PIDFBName.LMTD;
```

Where

EN	Enable. Data type: BOOL.
SP	Set point. Data type: REAL.
PVF	Process variable (feedback). Data type: REAL.
FF	Output feed forward. Data type: REAL.
KP	Proportional gain. Data type: REAL.
KI	Integral gain. Data type: REAL.
KD	Derivative gain. Data type: REAL.
OVR	Manual override. Data type: REAL.
SPR	Set point ramping. **Note:** If this parameter is zero or negative, set point ramping is turned off. Data type: REAL.
OHL	High output limit. **Note:** If OHL=OLL, the limiting function is ignored. Data type: REAL.
OLL	Low output limit. **Note:** If OHL=OLL, the limiting function is ignored. Data type: REAL.
IHL	High integral limit. Data type: REAL.
ILL	Low integral limit. Data type: REAL.

IHLD	Integral hold. Data type: BOOL.
MAN	Auto/manual control. Data type: BOOL.
ENO	Enable output. Data type: BOOL.
OUT	Output. Data type: REAL.
FE	Following error. Data type: REAL.
LMTD	Output limited. Data type: BOOL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using function blocks with the Instruction List language.
3. Refer to **Structured Text** for information on using function blocks with the Structured Text language.
4. Not all parameters are available within the RLL editor. If used, you must set these parameters by some other means; for example, in structured text.

Operation

A PID symbol is a named local symbol of type PIDFB. The SP input should be connected to the symbol providing the setpoint value for the process. The PVF input should be connected to the symbol providing the feedback input from the process. The OUT value should be connected to the symbol that will be sent to control the process. The rate of change of the internal SP input will be limited by the SPR value.

The KP, KI and KD are the programmable gains for the process. The FF input provides an optional FeedForward value for the PID output (OUT). It can be used to feed forward a value to offset the PID output either dynamically or statically. The PID is evaluated every I/O scan. The output (OUT) of the PID function is approximately equal to:

```
OUT = KP * ( FE + KD * derivative(FE) + Iterm ) + FF
Where  Iterm = integral(KI * FE)
       FE = SP - PVF
```

The PID output is limited by OHL as the high limit and OLL as the lower limit. LMTD output will be set if the computed OUT was greater than OHL or less than IHL, and is reset otherwise.

The contribution of the Integral Term is limited to a maximum of IHL and a minimum of ILL. The accumulation of the Integral Term can be held and its value frozen by making the IHLD input TRUE. The Integral Term is automatically frozen internally whenever the PID output (OUT) reaches the OHL or OLL limits.

The function of the PID can be overridden by placing the PID into manual mode by making the MAN input TRUE. In manual mode the PID output (OUT) will be equal to the OVR input. When the PID mode is switched back to automatic mode by making the MAN input FALSE, the transfer of the PID output will occur bumplessly by internally loading the Integral Term with the amount of the current PID output (OUT) minus the FF value and the amount that would be contributed by the Proportional term (KP) given the current SP and PV: Iterm = (OUT - FF - KP*(SP-PV)) / KP. This will prevent the PID output from making a sudden discontinuous change when the PID mode is switched from manual to automatic.

Operation Notes

1. A symbol of type PID should only be defined as a local variable. A global PID symbol will be solved once per I/O scan per running program. This effect will cause instability and unpredictable results in PID outputs.
2. If SPR < 0, then no setpoint ramping will occur and the PID loop will use the raw SP value.
3. When MAN = FALSE (AUTO mode), the OVR is set equal to OUT each time the PID is evaluated.
4. IHL, ILL and IHLD will affect the bumpless transfer result.
5. FF can be used as either static bias or as a dynamic FeedForward term or both.

Extended Timers

Introduction

Extended timers include:

Extended Timer Off Delay (XTOF)	Provides off-delay timing of events.
Extended Timer On Delay (XTON)	Provides on-delay timing of events.
Extended Timer Pulse (XTP)	Activated by a pulse, provides off-delay timing of events.

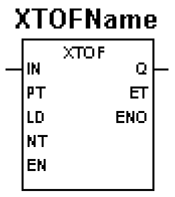
Also see Counter and Timers.

Extended Timer Off Delay (XTOF)

Description

Turns off an output after a preset time delay. The XTOF timer has two additional inputs (*LoadTime* and *NewTime*, described below), but otherwise works the same as the TOF timer.

RLL



ST Function

```
XTOFName.IN := Start;               (*Assign inputs*)
XTOFName.PT := PresetTime;
XTOFName.LD := LoadTime;
XTOFName.NT := NewTime;
XTOFName ( );                       (*Call*)
Output := XTOFName.Q;               (*Access outputs*)
ElapsedTime := XTOFName.ET;
```

Where

XTOFName	Unique name for the timer.
Start (IN)	Starts the timer. Data type: BOOL. Rung input in RLL.
PresetTime (PT)	Specifies the timer period. Data type: TIME.
LoadTime (LD)	A transition from low to high of LD causes NT to be loaded directly into the internal time accumulator of the timer. The NT is immediately reflected in the timer's ET output. Any transition of other timer inputs during the transition from low to high of LD are ignored except for EN. If EN goes low at the same time LD goes high, the LD input is ignored and the timer is disabled. Data type: BOOL.
NewTime (NT)	The time value loaded by *LoadTime*. Data type: WORD.
Output (Q)	FALSE when timer times out. Data type: BOOL. Rung output in RLL.
ElapsedTime (ET)	The current elapsed time. Data type: TIME.
Enable (EN)	Enables the timer. Data type: BOOL.

Enable Output (ENO)	Echoes EN. Data type: BOOL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using function blocks with the Instruction List language.
3. Refer to **Structured Text** for information on using function blocks with the Structured Text language.

Operation

- When EN is TRUE the timer is enabled.
- When IN is FALSE, the timer is active, storing the elapsed time in ET.
- When the elapsed time ET equals preset time PT, TOF sets the rung output Q to FALSE.
- When EN is set to FALSE, TOF freezes ET in its current state and sets Q to zero. When EN is set TRUE again, Q is restored to its previous value.
- When IN is set TRUE, ET is reset to zero and Q is set to FALSE. (Note: If PT is zero, then Q is TRUE.)
- A transition from low to high of LD will cause NT to be loaded directly into the internal time accumulator of the timer. The NT will immediately be reflected in the timer's ET output. Any transition of other timer inputs during the transition from low to high of LD are ignored except for EN. If EN goes low at the same time LD goes high, the LD input is ignored and the timer is disabled.
- ENO echoes the value of EN.

Variables

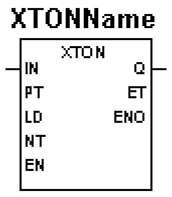
You can reference the XTOF timer variables by prefixing the variable with the counter instance name (*XTOFName*) as follows:

XTOFName.**IN**
XTOFName.**PT**
XTOFName.**LD**
XTOFName.**NT**
XTOFName.**Q**
XTOFName.**ET**
XTOFName.**EN**
XTOFName.**ENO**

Extended Timer On Delay (XTON)

Description Turns on an output after a preset time delay. The XTON timer has two additional inputs (*LoadTime* and *NewTime,* described below), but otherwise works the same as the TON timer.

RLL



ST Function

XTONName.**IN :=** *Start*;	(*Assign inputs*)
XTONName.**PT :=** *PresetTime*;	
XTONName.**LD :=** *LoadTime*;	
XTONName.**NT :=** *NewTime*;	
XTONName **( );**	(*Call*)
Output **:=** *XTONName*.**Q;**	(*Access outputs*)
ElapsedTime **:=** *XTONName*.**ET;**	

Where

XTONName	Unique name for the timer.
Start (IN)	Starts the timer. Data type: BOOL. Rung input in RLL.
PresetTime (PT)	The timer period. Data type: TIME.
LoadTime (LD)	A transition from low to high of LD will cause NT to be loaded directly into the internal time accumulator of the timer. The NT will immediately be reflected in the timer's ET output. Any transition of other timer inputs during the transition from low to high of LD are ignored except for EN. If EN goes low at the same time LD goes high, the LD input is ignored and the timer is disabled. Data type: BOOL.
NewTime (NT)	The time value loaded by *LoadTime*. Data type: WORD.
Output (Q)	TRUE when timer times out. Rung output in RLL.
ElapsedTime (ET)	The current elapsed time. Data type: TIME.
Enable (EN)	Enables the timer. Data type: BOOL.

Enable Output (ENO)	Echoes EN. Data type: BOOL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.
3. Refer to **Structured Text** for information on using function blocks with the Structured Text language.

Operation

- When the enable input EN is TRUE, the timer is enabled.
- When input IN transitions to TRUE, the timer is active, storing the elapsed time in output ET.
- When the elapsed time ET equals preset time PT, TON sets the rung output Q to TRUE.
- When EN is set to FALSE, TON freezes ET in its current state and sets Q to zero. When EN is set to TRUE again, Q is restored to its original value.
- If input IN becomes FALSE, the system resets ET to zero and sets output Q to FALSE.
- A transition from low to high of LD will cause NT to be loaded directly into the internal time accumulator of the timer. The NT will immediately be reflected in the timer's ET output. Any transition of other timer inputs during the transition from low to high of LD are ignored except for EN. If EN goes low at the same time LD goes high, the LD input is ignored and the timer is disabled.
- Enable output ENO echoes the value of EN.

Variables

You can reference the XTON timer variables by prefixing the variable with the counter instance name (X*TONName*) as follows:

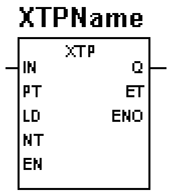
XTONName.**IN**
XTONName.**PT**
XTONName.**LD**
XTONName.**NT**
XTONName.**Q**
XTONName.**ET**
XTONName.**EN**
XTONName.**ENO**

Extended Timer Pulse (XTP)

Description

The XTP function block times the duration of an event. After its input pulses from off to on, the XTP keeps time to the preset interval and sets an output FALSE, which makes the XTP an off-delay timer. The XTP timer has two additional inputs (*LoadTime* and *NewTime*, described below), but otherwise works the same as the TP timer.

RLL



ST Function

XTPName.**IN** := *Start*;	(*Assign inputs*)
XTPName.**PT** := *PresetTime*;	
XTPName.**LD** := *LoadTime*;	
XTPName.**NT** := *NewTime*;	
XTPName **( );**	(*Call*)
Output **:=** *XTPName*.**Q;**	(*Access outputs*)
ElapsedTime **:=** *XTPName*.**ET;**	

Where

XTPName	Unique name for the timer.
Start (IN)	Starts the timer. Data type: BOOL. Rung input in RLL.
PresetTime (PT)	Specifies the period for which the timer times. Data type: TIME.
LoadTime (LD)	A transition from low to high of LD will cause NT to be loaded directly into the internal time accumulator of the timer. The NT will immediately be reflected in the timer's ET output. Any transition of other timer inputs during the transition from low to high of LD are ignored except for EN. If EN goes low at the same time LD goes high, the LD input is ignored and the timer is disabled. Data type: BOOL.
NewTime (NT)	The time value loaded by *LoadTime*. Data type: WORD.
Output (Q)	Changes to FALSE when timer times out. Data type: BOOL. Rung output in RLL.

ElapsedTime (ET)	Contains the current elapsed time. Data type: TIME.
Enable (EN)	Enables the timer. Data type: BOOL.
Enable Output (ENO)	Echoes EN. Data type: BOOL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using function blocks with the Instruction List language.
3. Refer to **Structured Text** for information on using function blocks with the Structured Text language.

Operation

- When the enable input EN is TRUE the timer is enabled.
- When input IN transitions to TRUE, the timer increments, storing the elapsed time in output ET. The timer continues to time up, even if IN transitions to FALSE. Therefore, a FALSE-to-TRUE pulse can start a timer that is enabled.
- When the elapsed time ET equals preset time PT, TP sets the rung output Q to FALSE.
- If enable input EN becomes FALSE, TP freezes CV in its current state and sets Q to FALSE. When EN becomes TRUE again, Q is restored to its previous value.
- A transition from low to high of LD will cause NT to be loaded directly into the internal time accumulator of the timer. The NT will immediately be reflected in the timer's ET output. Any transition of other timer inputs during the transition from low to high of LD are ignored except for EN. If EN goes low at the same time LD goes high, the LD input is ignored and the timer is disabled.
- ENO echoes the value of EN.

XTP Variables

You can reference the XTP timer variables by prefixing the variable with the counter instance name (*XTPName*) as follows:

*XTPName***.IN**
*XTPName***.PT**
*XTPName***.LD**
*XTPName***.NT**
*XTPName***.Q**
*XTPName***.ET**
*XTPName***.EN**
*XTPName***.ENO**

File

Introduction

File functions include:

Append File	Writes data to the end of the file.
Close File	Closes a file that has been opened by the OPENFILE function.
Copy File	Copies an existing file.
Delete File	Deletes an existing file.
New File	Creates a new file.
Open File	Opens a file for operations, such as reading or writing.
Read File	Reads data from a file.
Rewind File	Positions the internal file pointer to the beginning of a file.
Write File	Writes data to a file.

Note: File functions (New File, Open File, etc.) default to the project directory, unless a path is provided in the file name.

File Control Block Variable

File operations are performed using a file control block variable to identify the file. For example, to open a file, the Structured Text statement similar to the following is used:

```
OPENFILE(File01, FILE:= "DataFile");
```

Where "DataFile" is the name of the file to open. Further references to the file are made using the file control variable File01, not "DataFile". The file control block handles access to the file and error conditions. All file functions that operate on the same file must use the same file control block variable.

Note: The file control block variable should not be declared as a variable in the Symbol Manager.

You can perform only one operation for each file control block at a time. When you use file I/O operations within a step in an SFC, subsequent commands in the step cannot be executed, and the step cannot be terminated until the file operation has completed. The system automatically handles this coordination and suspends the execution of the SFC step until the FCBVar.BUSY flag is reset. Any other logic in the SFC program outside of the step containing the file operation executes normally.

File Status Variables

Other file variables provide status information about the file; for example, if the file is open or an error has occurred. By default, the status variables are accessed as local variables in the form: FCBVar.FStatus, where FCBVar is the file control block variable and FStatus is one of the file status variables. In the RLL Editor, status variables can be given explicitly defined symbol names. For example, File01.ERR would contain the error code of the file referenced by the file control variable File01.

You can use any of the file status variables in any expression, contact, or coil instead of a symbol of the same type. They are local variables, and cannot be used within the Watch Window, Operator Interface, or DDE applications.

Status Variable	Description
FCBVar.OPEN (File Open)	A Boolean variable indicating the open/closed status of the file. The system sets the File Open variable to true when the file is open.
FCBVar.BUSY (File Control Busy)	A Boolean variable indicating that the file is being accessed. The system sets the File Control Busy variable to true when the file is being accessed by another file function.
FCBVar.EFLAG (File Error)	A Boolean variable indicating when an error occurs. Each time the system accesses the file, it sets the File Error variable to FALSE. If an error occurs during a file operation, the system sets the File Error variable to TRUE.
FCBVar.ERR (File Error Code)	An integer variable containing the error code if an error occurs. Each time the system accesses the file, it writes a zero to the File Error Code integer. If an error occurs during a file operation, the system writes an error code to the File Error Code integer. For a listing of the error codes, refer to **File Error Codes**.
FCBVar.RDN (File Read Done)	A Boolean variable indicating that a read operation has been completed. The system sets the File Read Done variable to TRUE when the read operation is finished.

Status Variable	Description
FCBVar.WDN (File Write Done)	A Boolean variable indicating that a write operation has been completed. The system sets the File Write Done variable to TRUE when the write operation is finished.
FCBVar.CLSD (File Close)	A Boolean variable indicating that a file has been closed. The system sets the File Close variable to TRUE when it has closed the file.
FCBVar.EOF (End of File)	A Boolean variable indicating the system encountered an End of File. The system sets the End of File variable to TRUE when it encounters the EOF.
Note: FCBVar is the name of the file control block.	

File Error Codes

These error codes are written into the File Error Code variable.

Error Code	Description	Error Code	Description
15	File control block is busy	20	Read failed
16	No file name specified	21	File copy failed
17	File has not been opened	22	Write failed
18	File not found	23	End of line expected
19	Disk full	24	End of file expected

Append File

Description Writes data to the end of a file specified by the file control block.

RLL (not applicable)

ST Function **APPENDFILE (***FCBVar***, IN:=** *Structure***, F:=** *FieldSep***, S:=** *StringSep***, T:=** *EOL***)**

Where	
FCBVar	The name of the file control block that handles operations for this file.
Structure	A user-defined data type containing the data structure to write to the file.
FieldSep	Optional string used to separate fields in the file. The default is the space character.
StringSep	Optional string used to separate strings in the file.
EOL	Optional string used to indicate the end of a line in the file.
return	None.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **File Status Variables** for information on using function blocks with the Structured Text language.

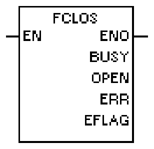
Example APPENDFILE (fcbrpt, IN:= datstruct, F:= “,”, S:= “$””, T:= “$n”);

The system accesses the data structure defined by datstruct and writes the data to the file referenced by the file control block called fcbrpt. The comma is used to separate fields, strings are enclosed within double quotation marks, and an end of line is indicated by $n.

Close File

Description Closes a file that has been opened by the OPENFILE function.

RLL



ST Function **CLOSEFILE (***FCBVar***)**

Where	
FCBVar	The name of the file control block that handles operations for this file.
BUSY, OPEN, ERR, EFLAG	For information about the file status variables, refer to File Status Variables.
return	None.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **File Status Variables** for information on using function blocks with the Structured Text language.

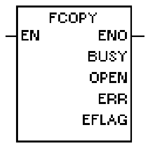
Example CLOSEFILE (datrpt);

The system closes the file associated with the file control block called datrpt.

Copy File

Description Copies the file specified by *FromFilename* to a new file and assigns it the name that is specified by *ToFilename*.

RLL



ST Function **COPYFILE (***FCBVar***, OUT:=** *ToFilename***, IN:=** *FromFilename***)**

Where

FCBVar	The name of the file control block that handles operations for this file
ToFilename (Destination File Name)	The name of the file to which the source is copied. Data type: STRING.
FromFilename (Source File Name)	The name of the file to be copied. Data type: STRING.
BUSY, OPEN, ERR, EFLAG	For information about the file status variables, refer to File Status Variables.
return	None.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **File Status Variables** for information on using function blocks with the Structured Text language.

Example

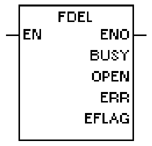
```
COPYFILE (datrpt, OUT:= newdatcopy IN:= olddatcopy);
```

The system makes a copy of the file called olddatcopy and names it newdatcopy.

Delete File

Description Deletes the file specified by *Filename*.

RLL



ST Function **DELETEFILE (***FCBVar***, IN:=** *Filename***)**

Where	
FCBVar	The name of the file control block that handles operations for this file.
Filename	Name of the file to be deleted. Data type: STRING.
BUSY, OPEN, ERR, EFLAG	For information about the file status variables, refer to File Status Variables.
return	None.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **File Status Variables** for information on using function blocks with the Structured Text language.

Example

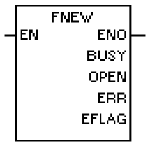
```
DELETEFILE (datrpt, IN:= olddatcopy);
```

The system deletes the file called olddatcopy.

New File

Description Creates the file specified by *Filename* and assigns it a file control block variable *FCBVar*. The file control block handles access to the file and error conditions. All file functions that operate on the same file must use the same file control block name.

RLL



ST Function **NEWFILE (***FCBVar***, FILE:=** *Filename***);**

Where

FCBVar	The name of the file control block that handles operations for this file.
Filename	Name of the file to be created. Data type: STRING.
BUSY, OPEN, ERR, EFLAG	For information about the file status variables, refer to File Status Variables.
return	None.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **File Status Variables** for information on using function blocks with the Structured Text language.

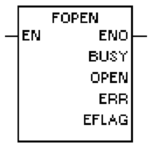
Example NEWFILE (fcbdatrpt, FILE:= datareport);

The system creates a new file called datareport.

Open File

Description Opens the file specified by *Filename* and assigns it a file control block variable *FCBVar*. The file control block handles access to the file and error conditions. All file functions that operate on the same file must use the same file control block name. After the file is opened, it is ready for file operations such as reading or writing.

RLL



ST Function **OPENFILE (***FCBVar***, FILE:=** *Filename***);**

Where

FCBVar	The name of the file control block that handles operations for this file.
Filename	Name of the file to be opened. Data type: STRING.
BUSY, OPEN, ERR, EFLAG	For information about the file status variables, refer to File Status Variables.
return	None.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **File Status Variables** for information on using function blocks with the Structured Text language.

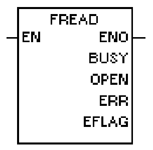
Example OPENFILE (fcbrpt, FILE:= report);

The system opens the file called report.

Read File

Description Reads data from the file specified by *FCBVar* and stores the data in the user defined structure specified by *Structure*. The Open File or the New File function must open the file before the Read File function can read it.

RLL



ST Function **READFILE (***FCBVar*, **OUT:=** *Structure*, **F:=** *FieldSep*, **S:=** *StringSep*, **T:=** *EOL***);**

Where

FCBVar	The name of the file control block that handles operations for this file.
Structure *(Data)*	A user-defined data type containing the data structure appropriate for the data read from the file. Data type: user type.
FieldSep *(Field Separator)*	Optional string used to separate fields in the file. The default is the space character. Data type: STRING (character).
StringSep *(String Delimiter)*	Optional string used to separate strings in the file. Data type: STRING (character).
EOL *(EOL Delimiter)*	Optional string used to indicate the end of a line in the file. Data type: STRING (character).
BUSY, OPEN, ERR, EFLAG	For information about the file status variables, refer to File Status Variables.
return	None.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **File Status Variables** for information on using function blocks with the Structured Text language.

Example READFILE (fcbrpt, OUT:= datstruct, F:= “,”, S:= “$””, T:= “$n”);

The system accesses the file referenced by the file control block called fcbrpt and uses the data structure defined by datstruct. The comma is used to separate fields, strings are enclosed within double quotation marks, and an end of line is indicated by $n.

To read an individual structure element of an array, specify the index (ArrayName[Index]). To read an entire array of structures, specify only the name of the array of structures with no index (ArrayOfStructureElements).

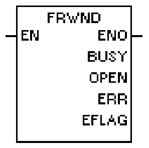
Note: READFILE can read an entire line from a file into a STRING variable in the following manner:

- If the S parameter (string separator) is zero length (''), the function searches for the F parameter (field separator) on either side of the string.
- If both the S and F parameters are zero length, then the function searches for the T parameter (end-of-line) and returns the entire line up to the end-of-line terminator.

Rewind File

Description Sets the file pointer to the beginning of the file specified by *FCBVar*. This lets the next file operation to begin at the start of the file. This operation is done automatically when you open a file with Open File.

RLL



ST Function **REWINDFILE (*FCBVar*)**

Where

FCBVar	The name of the file control block that handles operations for this file.
BUSY, OPEN, ERR, EFLAG	For information about the file status variables, refer to File Status Variables.
return	None.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **File Status Variables** for information on using function blocks with the Structured Text language.

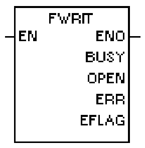
Example REWINDFILE (fcbrpt);

The system rewinds the file referenced by the file control block called fcbrpt.

Write File

Description Writes the data in the structure variable specified by *Structure* to the file specified by *FCBVar*. The Open File or the New File function must open the file before the Write File function can write it.

RLL



ST Function **WRITEFILE (***FCBVar***, IN:=** *Structure***, F:=** *FieldSep***, S:=** *StringSep***, T:=** *EOL***);**

Where

FCBVar	The name of the file control block that handles operations for this file.
Structure *(Data)*	A user-defined data type containing the data structure being written to the file. Data type: user type.
FieldSep *(Field Separator)*	Optional string used to separate fields in the file. The default is the space character. Data type: STRING (character).
StringSep *(String Delimiter)*	Optional string used to separate strings in the file. Data type: STRING (character).
EOL *(EOL Delimiter)*	Optional string used to indicate the end of a line in the file. Data type: STRING (character).
BUSY, OPEN, ERR, EFLAG	For information about the file status variables, refer to File Status Variables.
return	None.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **File Status Variables** for information on using function blocks with the Structured Text language.

Example WRITEFILE (fcbrpt, IN:= datstruct, F:= “,”, S:= “$””, T:= “$n”);

The system accesses the data structure defined by datstruct and writes the data to the file referenced by the file control block called fcbrpt. The comma is used to separate fields, strings are enclosed within double quotation marks, and an end of line is indicated by $n.

To write an individual structure element of an array, specify the index (ArrayName[Index]). To write an entire array of structures, specify only the name of the array of structures with no index (ArrayOfStructureElements).

Mathematical

Introduction

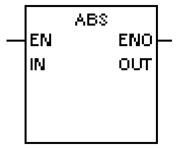
Mathematical functions include:

ABS	Computes the absolute value of a number.
ADD	Adds two numbers.
DIV	Divides one number by another.
EXPT	Raises the first number to the power specified by the second number.
MOD	Divides one number by another and stores the remainder.
MOVE	Copies data from one location to another.
MUL	Multiplies two numbers.
NEG	Negates (changes sign of) the input.
SQRT	Computes the square root of a number.
SUB	Subtracts one number from another.

Absolute Value (ABS)

Description Returns the absolute value of the input.

RLL



ST Function **ABS(***AnyNum***)**

IL Function **CALC ABS(OUT:=***VarNum***,***AnyNum***)**

Where

AnyNum (IN)	The value for which the absolute value is to be calculated. Data type: ANY_NUM.
return (OUT)	The absolute value of the input. Data type: ANY_NUM.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example

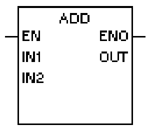
absNum := ABS(num);

If *num* = -99, then *absNum* = 99.

Addition (ADD)

Description Returns the result of summing the inputs.

RLL



ST Function **ADD(***AnyNumOrBit1*, *AnyNumOrBit2***)**

ST Operator **Out :=** *AnyNumOrBit1* **+** *AnyNumOrBit2***;**

IL Function **CALC ADD(OUT:=***VarNumOrBit*, *AnyNumOrAnyBit1*, *AnyNumOrAnyBit2***)**

Where	
AnyNumOrBit1, *AnyNumOrBit2* (IN1, IN2)	Contain the values to be added. Data type: ANY_NUM or ANY_BIT (excluding BOOL).
return (OUT)	The sum of the addition of the inputs. Data type: ANY_NUM or ANY_BIT (excluding BOOL).

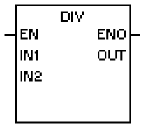
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Division (DIV)

Description Returns the result of dividing the first input value by the second input value.

RLL



ST Function **DIV(***AnyNumOrBit1*, *AnyNumOrBit2***)**

ST Operator **out :=** *AnyNumOrBit1* **/** *AnyNumOrBit2***;**

IL Function **CALC DIV(OUT:=***VarNumOrBit*, *AnyNumOrAnyBit1*, *AnyNumOrAnyBit2***)**

Where	
AnyNumOrBit1 (IN1)	The dividend. Data type: ANY_NUM or ANY_BIT (excluding BOOL).
AnyNumOrBit2 (IN2)	The divisor. Data type: ANY_NUM or ANY_BIT (excluding BOOL).
return (OUT)	The quotient of the division of *AnyNumOrBit1* by *AnyNumOrBit2*. Data type: ANY_NUM or ANY_BIT (excluding BOOL).

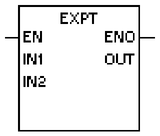
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Exponent (EXPT)

Description Returns the result of raising a value to the power specified by a second value.

RLL



ST Function **EXPT(***AnyNum1*, *AnyNum2***)**

ST Operator **out :=** *AnyNum1* ** *AnyNum2*;

IL Function **CALC EXPT(OUT:=***VarNum*, *AnyNum1*, *AnyNum2***)**

Where

AnyNum1 (IN1)	Contains the value to be raised to the power. Data type: ANY_NUM.
AnyNum2 (IN2)	Contains the value used as the exponent. Data type: ANY_NUM.
return (OUT)	The result of *AnyNum1* raised to the power of *AnyNum2*. Data type: ANY_NUM.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

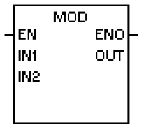
Example numexp := EXPT (vala, valb);

If *vala* = 2.5, and *valb* = 5, then numexp = *97.656250*.

Modulus (MOD)

Description Returns the remainder of dividing the first input by the second input.

RLL



ST Function **MOD(***AnyNumOrBit1*, *AnyNumOrBit2***)**

ST Operator **out :=** *AnyNumOrBit1* **MOD** *AnyNumOrBit2***;**

IL Function **CALC MOD(OUT:=***VarNumOrBit*, *AnyNumOrAnyBit1*, *AnyNumOrAnyBit2***)**

Where

AnyNumOrBit1 (IN1)	Contains the dividend. Data type: ANY_NUM or ANY_BIT (excluding BOOL).
AnyNumOrBit2 (IN2)	Contains the divisor. Data type: ANY_NUM or ANY_BIT (excluding BOOL).
return (OUT)	The modulus of the division of *AnyNumOrBit1* by *AnyNumOrBit2*. Data type: ANY_NUM or ANY_BIT (excluding BOOL).

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

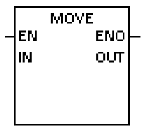
Example realout := (real1 MOD real2);

If real1=11.5 and real2=5, then realout=1.5.

Move (MOVE)

Description Returns the result of converting the input value to the same data type as the output.

RLL



ST Function **MOVE(***Any***)**

IL Function **CALC MOVE(OUT:=***VarAny*, *Any***)**

Where	
Any (IN)	The value to be copied. Data type: ANY.
return (OUT)	The result of the move operation. Data type: ANY.

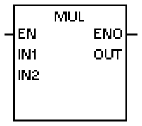
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Multiplication (MUL)

Description Returns the result of multiplying the input values.

RLL



ST Function **MUL(***AnyNumOrBit1***,** *AnyNumOrBit2***)**

ST Operator **out :=** *AnyNumOrBit1* ***** *AnyNumOrBit2***;**

IL Function **CALC MUL(OUT:=***VarNumOrBit***,** *AnyNumOrAnyBit1***,** *AnyNumOrAnyBit2***)**

Where

AnyNumOrBit1, *AnyNumOrBit2* (IN1)	The values to be multiplied. Data type: ANY_NUM or ANY_BIT (excluding BOOL).
return (OUT)	The product of the multiplication of the inputs. Data type: ANY_NUM or ANY_BIT (excluding BOOL).

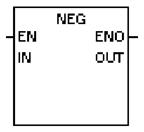
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Negation (NEG)

Description Returns the result of changing the sign of the input value.

RLL



ST Function **NEG(***AnyNum***)**

ST Operator **out := -***AnyNum*;

IL Function **CALC NEG(OUT:=***VarNum*, *AnyNum***)**

Where

AnyNum (IN)	The value to be negated. Data type: ANY_NUM.
return (OUT)	The negated value of IN. Data type: ANY_NUM.

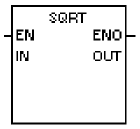
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Square Root (SQRT)

Description Returns the square root of the input value.

RLL



ST Function **SQRT(***AnyNum***)**

IL Function **CALC SQRT(OUT:=***VarNum***,** *AnyNum***)**

Where	
AnyNum (IN)	The value for which the square root is to be calculated. Data type: ANY_NUM.
return (OUT)	The square root of the input. Data type: ANY_NUM.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example

```
sqrtNum := SQRT(num);
```

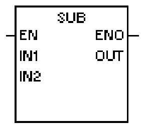
If *num* = 132 (integer), then *sqrtNum* = 11.

If *num* = 132.0 (real), then after the operation, *sqrtNum* = 11.489125.

Subtraction (SUB)

Description Returns the result of subtracting the second input value from the first input value.

RLL



ST Function **SUB(***AnyNumOrBit1***,** *AnyNumOrBit2***)**

ST Operator **out :=** *AnyNumOrBit1* **-** *AnyNumOrBit2***;**

IL Function **CALC SUB(OUT:=***VarNumOrBit***,** *AnyNumOrAnyBit1***,** *AnyNumOrAnyBit2***)**

Where

AnyNumOrBit1 (IN1)	Contains the minuend, the number from which a value is subtracted. Data type: ANY_NUM or ANY_BIT (excluding BOOL).
AnyNumOrBit2 (IN2)	Contains the subtrahend, the number that is subtracted. Data type: ANY_NUM or ANY_BIT (excluding BOOL).
return (OUT)	The result of the subtraction of *AnyNumOrBit2* from *AnyNumOrBit1*. Data type: ANY_NUM or ANY_BIT (excluding BOOL).

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Miscellaneous

Introduction

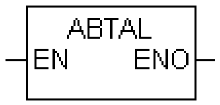
Miscellaneous functions include:

Abort All	Aborts all programs in the runtime subsystem.
Change MMI Screen	Displays an operator interface screen under program control.
Display Message	Displays a message in a Windows message box.
Initialize Array	Initializes all of the elements of an array to a specified value.
Message Window	Displays a message in the Program Editor Output Window.

Abort All

Description Aborts **all** programs in the runtime subsystem.

RLL



ST Function **ABORT_ALL**

IL Function **CALC ABORT_ALL**

Where	
	No parameters.
return	None.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example From Structured Text execute the following statements:

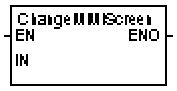
```
IF (In1) THEN
        ABORT_ALL;
END_IF;
```

When the IF statement is evaluated as TRUE the ABORT_ALL function is executed and all programs in the runtime subsystem are aborted.

Change MMI Screen

Description Displays a specified HMI screen from an RLL, SFC, structured text, or instruction list program.

RLL



ST Function **ChangeMMIScreen (***ScreenName***);**

IL Function **CALC ChangeMMIScreen(***ScreenName***)**

Where

ScreenName	Name of the HMI screen to display. Data type: STRING.
return	Always returns TRUE.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Operation If the HMI is not running, the function does nothing. If the HMI is running but the specified screen does not exist, the HMI displays an error message. Otherwise, the specified screen appears.

Example From structured text execute the following statement:

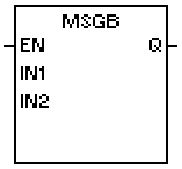
```
ChangeMMIScreen("PID Screen");
```

If the HMI is running and the screen "PID Screen" is in the active HMI configuration, it is displayed.

Display Message

Description Displays a message in a Windows message box. Program execution continues uninterrupted while the message window is displayed. The operator can click on *OK* to dismiss the message.

RLL



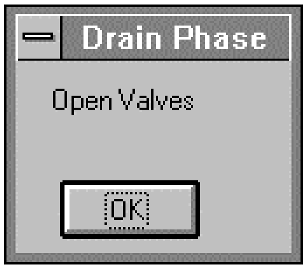
ST Function **DSPMSG (***StringA***,** *StringB***)**

Where	
StringA (IN1)	The message. Data type: STRING.
StringB (IN2)	The message box title. Data type: STRING.
return	None.

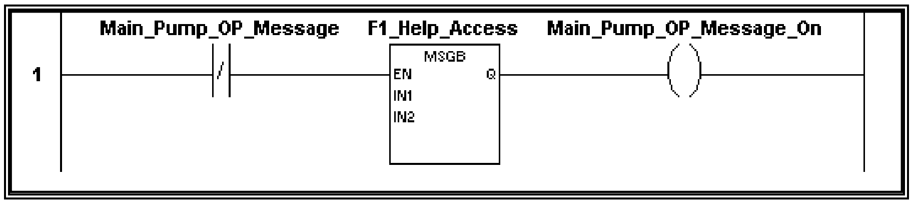
Note Refer to **Function Execution Control** for a description of using the EN input and ENO output.

Example DSPMSG ('Open Valves', Phase);

If variable Phase = "Drain Phase", then the window appears as shown.



The following is an example of the RLL MSGB function:



Initialize Array

Description	Initializes all of the elements of an array to a specified value.
RLL	(not applicable)
ST Function	**INIT(***AnyArray*, *Value***);**

Where

AnyArray	The array to be initialized. Data type: ANY.
Value	The value to which all elements in the array are to be initialized. Data type: ANY (must be the same as the array elements).
return	The value to which the array was initialized. Data type: ANY (must be the same as *Value*).

Note Refer to **Instruction List** for information on using functions with the Instruction List language.

Example

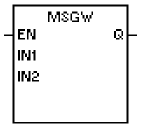
```
Status:= INIT(my_array, 10);
```

When this statement is executed all the elements in the array my_array are set to 10, and the variable Status is set to 10.

Message Window

Description Displays a message in the Program Editor *Output Window*.

RLL



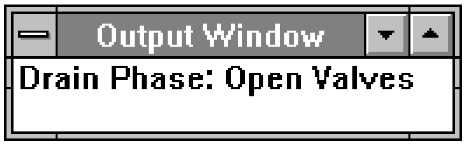
ST Function **MSGWND (***StringA***,** *StringB***)**

Where	
StringA (IN1)	The message. Data type: STRING.
StringB (IN2)	The message title. Data type: STRING.
return	None.

Note Refer to **Function Execution Control** for a description of using the EN input and ENO output.

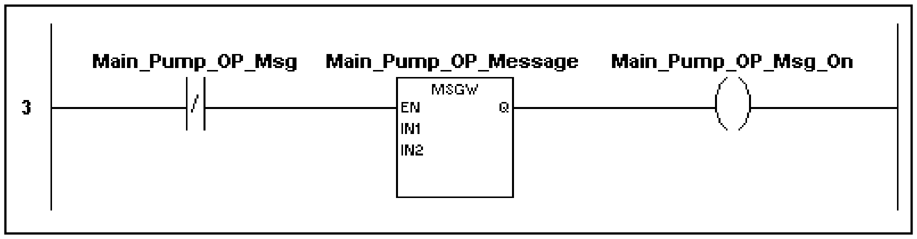
Example MSGWND (‘Open Valves’, Phase);

If the variable Phase = "Drain Phase", then the *Output Window* displays the message as shown below. Multiple message lines can appear in the *Output Window* .



```
Drain Phase: Open Valves
```

The following is an example of the MSGW function block:



PMAC 2 Functions

Introduction

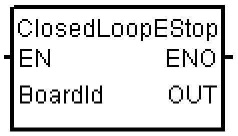
This function is for the PMAC 2 card only. PMAC 2 functions include:

ClosedLoopEStop	Stops axes motion and maintains the closed loop state.

ClosedLoopEStop

Description Stops axes motion at the E-Stop deceleration rate and maintains the closed loop state. This function is for the PMAC 2 driver only.

RLL



ST Function **ClosedLoopEStop(***BoardID***)**

IL Function **CALC ClosedLoopEStop(***BoardID***)**

Where

BoardID	PMAC board identification as defined by the system. Data type: DWORD.
return	An integer value indicating the command result: 0 Success. -1 Fail. -2 Invalid board identification. Data type: Integer.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Selection

Introduction

MAX	Determines the largest value of the inputs and uses that value as the output.
MIN	Determines the minimum value of the inputs and uses that value as the output.

Maximum (MAX)

Description	The MAX extensible function determines the largest value of the inputs and returns that value as the output. The function supports up to 16 inputs.
ST Function	**MAX(***Input1***,** *Input2***, …** *Input16***)**

Input1 through *Input16*	The list of input values to be examined. Data type: Any_Num, Any_Bit, String, Any_Date, Time.
Return	Data from the greatest input. Data type: Any_Num, Any_Bit, String, Any_Date, Time.

Example

```
result := MAX (vala, valb);
```

If vala = 14, and valb = 5.752, then result = 14.

Minimum (MIN)

Description The MIN extensible function determines the minimum value of the inputs and uses that value as the output. The function can have up to 16 inputs.

ST Function **MAX(***Input1*, *Input2*, **…** *Input16***)**

Where

Input1 to *Input16*	The list of input values to be examined. Data type: Any_Num, Any_Bit, String, Any_Date, Time.
Return	Data from the least input. Data type: Any_Num, Any_Bit, String, Any_Date, Time.

Example

```
result := MIN (vala, valb);
```

If vala = 3.7415, and valb = 5.752, then result = 3.7415.

System Objects

Introduction

System objects include:

PID Loop Control (PID)	Providing automatic closed-loop operation of continuous process control loops.
Program control block (PRGCB)	Allows an SFC application program to compile and control the execution of other SFC and RLL or RS-274D application programs.
Timer (TMR)	Implements a timer in the Structured Text language.

PID Loop Control (PID)

Description

A PID is an instruction providing automatic close-loop operation of continuous process control loops. For each loop the instruction performs proportional control and optionally integral control, derivative control, or both:

- Proportional control - causes an output signal to change as a direct ratio of the error signal variation.
- Integral control - causes an output signal to change as a function of the integral of error signal over the time duration.
- Derivative control - causes an output signal to change as a function of the rate of change of the error signal.

Note: Also refer to the PID function block EX_PID.

PID Inputs

Parameter	Variable/Type	Units	Example
Proportional Gain	KP : Real	Output/Input	W/°C
Integral Gain	KI : Real	KP/sec	W/°C/sec
Derivative Gain	KD : Real	KP * sec	W/°C * sec
Process Variable (Feedback)	PVF : Real	Input	°C
SetPoint	SP : Real	Input	°C
SetPoint Ramping	SPR : Real	Input/sec	°C/sec
Output Feed Forward	FF : Real	Output	W
High Output Limit	OHL : Real	Output	W
Low Output Limit	OLL : Real	Output	W
High Integral Limit	IHL : Real	Output	W
Low Integral Limit	ILL : Real	Output	W
Integral Hold	IHLD : BOOL	T/F	F
Auto/Manual Control	MAN : BOOL	T/F	T
Enable	EN : BOOL	T/F	F

	Manual Override	OVR : Real	Output	W

PID Outputs	Parameter	Variable/Type	Units	Example
	Output	OUT : Real	Output	W
	Following Error	FE : Real	Input	°C
	Output Limited	LMTD : BOOL	T/F	F
	Enable Output	ENO : BOOL	T/F	F

Operation

A PID symbol is a named local symbol of type PID. The SP input should be connected to the symbol providing the setpoint value for the process. The PVF input should be connected to the symbol providing the feedback input from the process. The OUT value should be connected to the symbol that will be sent to control the process. The rate of change of the internal SP input will be limited by the SPR value.

The KP, KI and KD are the programmable gains for the process. The FF input provides an optional FeedForward value for the PID output (OUT). It can be used to feed forward a value to offset the PID output either dynamically or statically. The PID is evaluated every I/O scan. The output (OUT) of the PID function is approximately equal to:

```
OUT = KP * ( FE + KD * derivative(FE) + Iterm ) + FF
Where  Iterm = integral(KI * FE)
       FE = SP - PVF
```

The PID output is limited by OHL as the high limit and OLL as the lower limit. LMTD output will be set if the computed OUT was greater than OHL or less than IHL, and is reset otherwise.

The contribution of the Integral Term is limited to a maximum of IHL and a minimum of ILL. The accumulation of the Integral Term can be held and its value frozen by making the IHLD input TRUE. The Integral Term is automatically frozen internally whenever the PID output (OUT) reaches the OHL or OLL limits.

The function of the PID can be overridden by placing the PID into manual mode by making the MAN input TRUE. In manual mode the PID output (OUT) will be equal to the OVR input. When the PID mode is switched back to automatic mode by making the MAN input FALSE, the transfer of the PID output will occur bumplessly by internally loading the Integral Term with the amount of the current PID output (OUT) minus the FF value and the amount that would be contributed by the Proportional term (KP) given the current SP and PV: Iterm = (OUT - FF - KP*(SP-PV)) / KP. This will prevent the PID output from making a sudden discontinuous change when the PID mode is switched from manual to automatic.

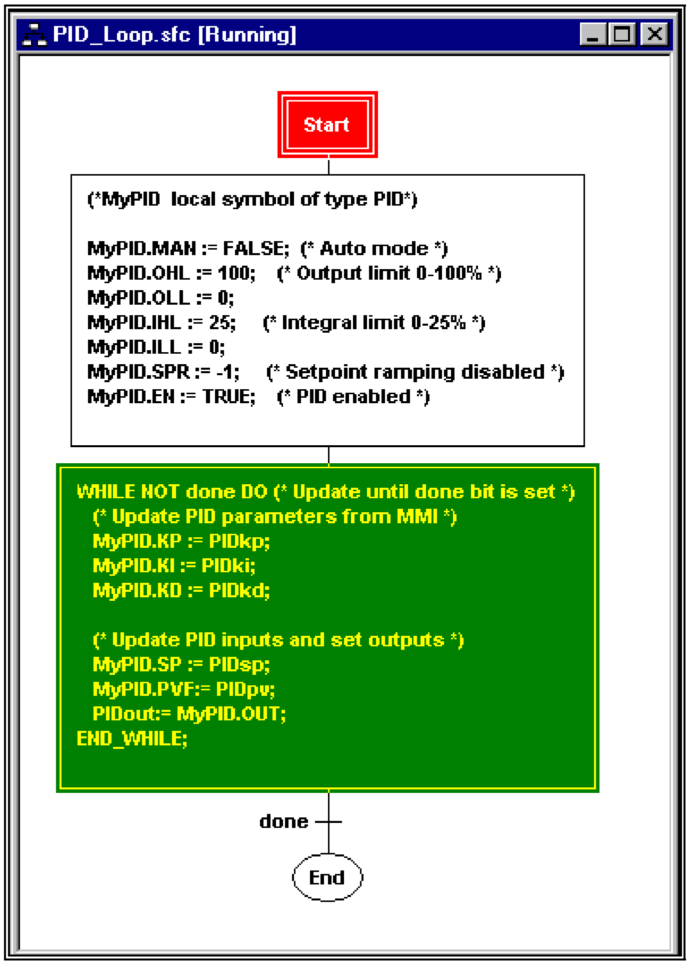
Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. A symbol of type PID should only be defined as a local variable. A global PID symbol will be solved once per I/O scan per running program. This effect will cause instability and unpredictable results in PID outputs.
3. If SPR < 0, then no setpoint ramping will occur and the PID loop will use the raw SP value.
4. When MAN = FALSE (AUTO mode), the OVR is set equal to OUT each time the PID is evaluated.
5. IHL, ILL and IHLD will affect the bumpless transfer result.
6. FF can be used as either static bias or as a dynamic FeedForward term or both.

Example

To use a PID in Structured Text

1. Create a local variable of type PID.
2. Set initial values of KP, KI, KD, OHL, OLL, IHL, ILL, IHLD, SPR, MAN, FF.
3. Create program to constantly update SP, PVF, and FF (optional) and store OUT.
4. Set EN to start the PID updating.



```
(*MyPID  local symbol of type PID*)

MyPID.MAN := FALSE;  (* Auto mode *)
MyPID.OHL := 100;    (* Output limit 0-100% *)
MyPID.OLL := 0;
MyPID.IHL := 25;     (* Integral limit 0-25% *)
MyPID.ILL := 0;
MyPID.SPR := -1;     (* Setpoint ramping disabled *)
MyPID.EN := TRUE;    (* PID enabled *)
```

```
WHILE NOT done DO (* Update until done bit is set *)
  (* Update PID parameters from MMI *)
  MyPID.KP := PIDkp;
  MyPID.KI := PIDki;
  MyPID.KD := PIDkd;

  (* Update PID inputs and set outputs *)
  MyPID.SP := PIDsp;
  MyPID.PVF:= PIDpv;
  PIDout:= MyPID.OUT;
END_WHILE;
```

Program Control Block (PRGCB)

Description The Program Control Block (PRGCB) allows an SFC application program to compile and control the execution of other SFC and RLL or RS-274D application programs.

PRGCB Inputs	Variable Name	Type	Description	Read/ Write	System	User
	prcgb.NAME	STRING	Program name to control	R/W	-	S
	prgcb.RUN	BOOL	Executes program	R/W	C	S
	prgcb.ABORT	BOOL	Cancels program execution	R/W	C	S
	prgcb.GEN	BOOL	Generates .SFC and .SST files from .NC text files	R/W	C	S
	prgcb.REWIND	BOOL	On rise, rewind program	R/W	C	S
	prgcb.SINGLE	BOOL	Activates single block execution	R/W	-	S/C
	prgcb.BLKDEL	BOOL	Activates block delete	R/W	-	S/C

PRGCB Outputs	Variable Name	Type	Description	Read/ Write	System	User
	prgcb.BLKNO	INT	Active block number of RS-274 program	R	S	-
	prgcb.STATUS	INT	Program status	R	S	-
	prgcb.INCYCLE	BOOL	Indicates program is running	R	S/C	-

Key:
R - Read Capability W - Write Capacity
C - Cleared (Reset) by S - Set by

Example **To use the PRGCB**

1. Create an SFC program and define a local variable (e.g. prgcb) of type PRGCB.
2. Store a STRING containing the path and filename of the program to control in prgcb.NAME.
3. Use the PRGCB Boolean inputs to control the program.
4. Use the PRGCB outputs to obtain information about the program.

Controlling the Flow of RLL and Structured Text Application Programs

The Program Control Block (PRGCB) lets an SFC application program to compile and control the execution of other SFC and RLL or RS-274D application programs.

Using the PRGCB Status Code

PRGCB.STATUS allow users to monitor the various states of control program execution. This unique variable returns an integer value that can be used both from within the control program itself and displayed as an indicator on the Operator Interface screen. Prgcb.STATUS indicates the following:

0 - Initialized	8 - Program Complete
1 - File Not Found	9 - Program Aborting
2 - File Opened	10 - Program Aborted
3 - File Parsing	11 - Program Faulted
4 - Parse Error Occurred	12 - Program At Breakpoint
5 - Parse Complete	13 - Program Suspended
6 - Program Running	14- Program Rewound (REWIND functionality)
7 - Program Stopped	15 - Program Not Rewound (REWIND functionality)

Using the PRGCB Rewind Function

After a control program (not ending in M30) has run through to completion, it will not run again until the user has set the local program control variable prgcb.REWIND, thus effectively rewinding the program.

A program's rewind functionality can be controlled by two MCodes: M30 and M02. The incorporation of an M30 MCode in the last RS-274D block of your control program will automatically rewind the program upon completion without the need to set prgcb.REWIND. The incorporation of the

M02 M Code in the last RS-274D block explicitly states for the program to "not rewind". Again, M02 (not rewound) is the default for every program.

A program's rewind status can be monitored via the prgcb.STATUS variable.

Timer (TMR)

Description

Implements a timer in the Structured Text language.

Timer Symbols

The TMR has three system symbols, which are identified by the timer name plus an extension. A TMR symbol is valid in any instruction or function block that accepts an ANY or TMR data type.

Symbol	Description
Tmr_name.PT	Contains the preset time value. The preset value can be a REAL data value, which is interpreted as a value in seconds, or a TIME data value: a T# or t# followed by a sequence of one or more numbers and time unit specifiers.
Tmr_name.EN	Starts/stops the TMR and is a BOOLEAN data type.
Tmr_name.ET	Contains the elapsed time of the TMR in seconds.

Operation

- When tmr_name.EN transitions from FALSE to TRUE, tmr_name.ET is set to zero and the timer begins to time.
- When tmr_name.ET equals the preset, then tmr_name.EN is automatically reset. You can design the program to reset tmr_name.EN earlier than the preset time.
- If tmr_name.EN is set to FALSE, tmr_name.ET is frozen at its last value.
- The elapsed time tmr_name.ET can be read at any time.

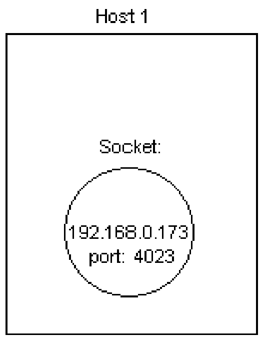
TCP/IP Sockets

Introduction

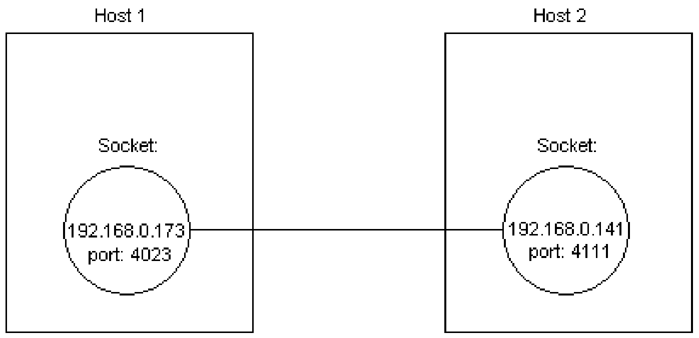
Notes:

1. All descriptions in this document refer specifically to how the control system TCP/IP sockets operate.
2. Refer to the **TWO_CLIENT_ONE_SERVER_DEMO.cfg** in the samples folder for an example of TCP/IP sockets communications.
3. To set or determine the IP address and information on port numbers, refer to IP Address.

A TCP/IP socket is a connection end point in a TCP/IP host. A socket is uniquely identified by its IP address and port number:

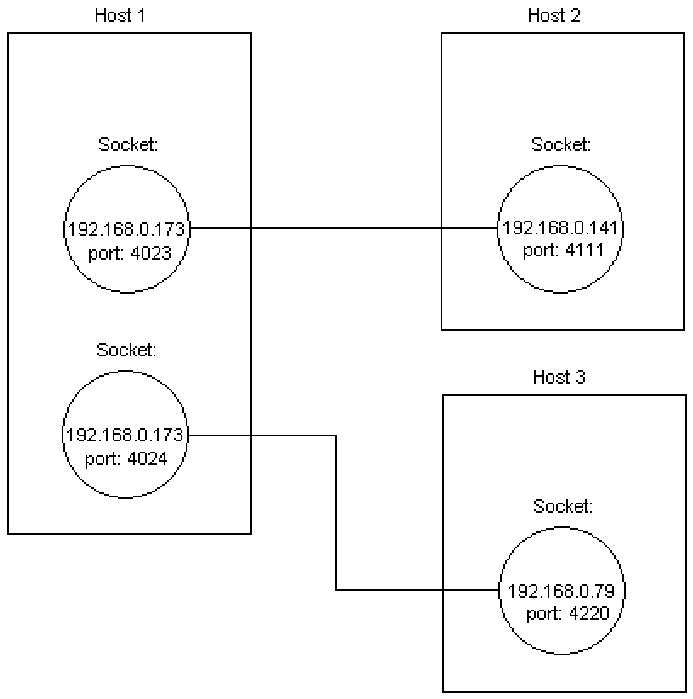


A connection can be made between two sockets, allowing hosts to exchange data through the connection:



Most socket applications are client/server applications. The client end of a connection actively connects to the remote socket. The server end of a connection waits for the client to connect. TCP/IP Sockets require that one end of a connection to be defined as a client socket. The other end must be a server socket. Once connected, the host can send/receive application specific data to/from the remote host through the local socket.

TCP/IP Sockets can support multiple socket connections. Note that a host creates one socket for each connection.



Socket Addressing

TCP/IP Sockets are uniquely identified by its IP address/port number pair. Socket addresses are ASCII strings of the following form:

a.b.c.d:p

Where *a.b.c.d* is the IP address; and *p* is the port number. Each socket on a host must have a unique socket address.

Socket Types

TCP/IP Sockets support two socket types: Stream and Datagram.

Stream Socket	Provides a reliable byte stream connection. It uses TCP (transport) for data delivery, which ensures reliable communications.
Datagram Socket	Provides a packet-based connection where datagrams are exchanged through the socket. Datagrams are buffers of a fixed (typically small) maximum length. The maximum length is determined by the physical network (ethernet, token-ring, etc.). Because UDP is used for data delivery, the connection is unreliable. The application must be able to handle lost or duplicate packets.

Socket Buffers

Associated with each TCP/IP Socket is an internal input and output buffer. The size of each buffer is configurable. API functions allow the application to build and send messages from the output buffer; and to receive messages into the input buffer and to extract data from each message.

TCP/IP Sockets API Summary

Function Type	**Functions**
TCP/IP Initialization	TCP_START_SOCKET_SERVICE WAIT_TCP_START_SOCKET_SERVICE
Socket Creation and Connection	TCP_CREATE WAIT_TCP_CREATE TCP_CONNECT WAIT_TCP_CONNECT
Socket Message Transmission	TCP_CLEAR_SEND_BUFFER TCP_APPEND_*datatype* TCP_SEND_BUFFER

Function Type	Functions
	WAIT_TCP_SEND_BUFFER
Socket Message Reception	TCP_RECV_BUFFER WAIT_TCP_RECV_BUFFER TCP_EXTRACT_*datatype* TCP_GET_EXTRACT_ERROR TCP_RESET_RECV_BUFFER
Socket Closure	TCP_CLOSE WAIT_TCP_CLOSE

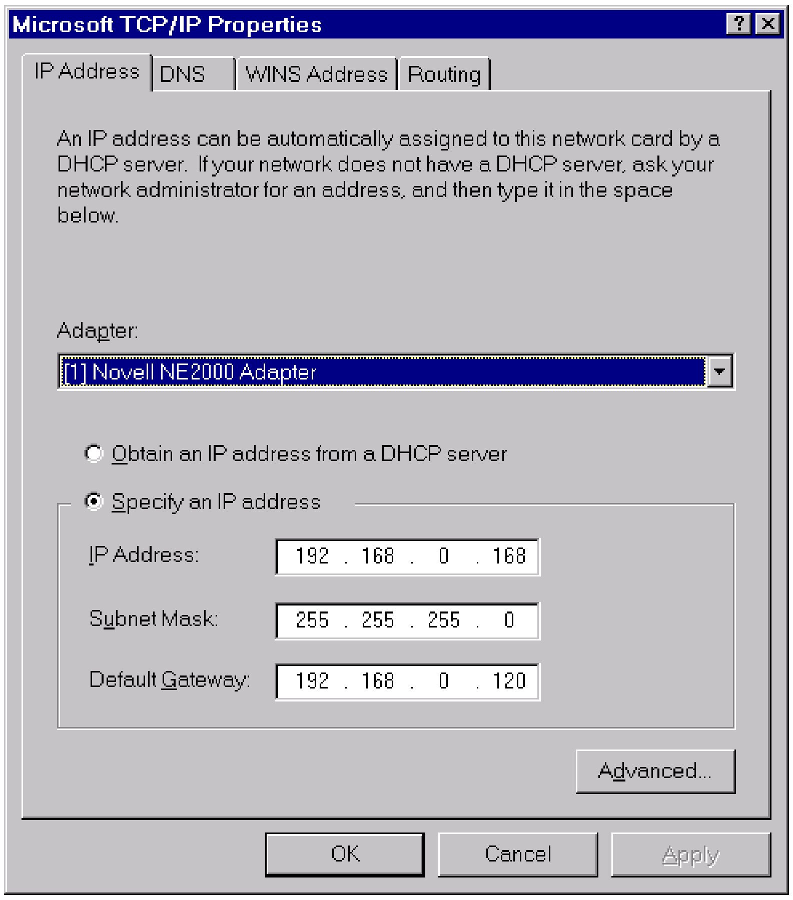
IP Address

In Windows NT4.0, the IP address is available from the *Microsoft TCP/IP Properties* windows. This window is accessed as follows:

1. Open the *Control Panel* from the *Settings* menu on the Windows *Start* menu.
2. Next open the *Network* application.
3. Select the *Protocols* tab
4. Select the *TCP/IP Protocol* and choose *Properties*. The *Microsoft TCP/IP Properties* dialog box appears as shown in the figure.

You can specify or observe the PC's IP address.

Regarding the port number, typically port numbers 4000 and above are available. However, you may need to consult with your network administrator if you run into problems.



An IP address can be automatically assigned to this network card by a DHCP server. If your network does not have a DHCP server, ask your network administrator for an address, and then type it in the space below.

Adapter: [1] Novell NE2000 Adapter

○ Obtain an IP address from a DHCP server

● Specify an IP address

IP Address: 192 . 168 . 0 . 168

Subnet Mask: 255 . 255 . 255 . 0

Default Gateway: 192 . 168 . 0 . 120

TCP_APPEND_*datatype*

Functions INT TCP_APPEND_INT (INT handle, INT val, BOOL conversion)

INT TCP_APPEND_DINT (INT handle, INT val, BOOL conversion)

INT TCP_APPEND_REAL (INT handle, REAL val)

INT TCP_APPEND_LREAL (INT handle, REAL val)

INT TCP_APPEND_BYTE (INT handle, BYTE val)

INT TCP_APPEND_WORD (INT handle, WORD val, BOOL conversion)

INT TCP_APPEND_DWORD (INT handle, DWORD val, BOOL conversion)

Description Appends the specified data item to the end of the message in the output buffer. The valid data types are:

TYPE	SIGN	SIZE
INT	Signed	2 bytes
DINT	Signed	4 bytes
REAL	Signed	4 bytes
LREAL	Signed	8 bytes
BYTE	Unsigned	1 byte
WORD	Unsigned	2 bytes
DWORD	Unsigned	4 bytes

Parameters

handle:	socket handle returned from the `TCP_CREATE` command.
val:	Data to append to end of output message.
conversion:	Data conversion switch:
	FALSE = no conversion
	TRUE = convert to network byte order

Returns	ReturnVal < `-1`	`TCP_APPEND_`*datatype* command failed. See error codes below for the specific error.
	ReturnVal = `0`	`TCP_APPEND_`*datatype* command completed successfully.

Notes

1. Prior to building a new output message, call `TCP_CLEAR_SEND_BUFFER` to empty the output buffer.

2. Use the append functions to sequentially build an output message, starting with the beginning of the message.

3. The socket must be connected, or this command will fail.

4. A string must not exceed 255 characters in length (not including the NULL) or this function will fail. This function appends a NULL character (00h) to the end of the string in the output buffer.

Error codes

-1003	TCPINVALIDHANDLE	Invalid handle.
-1006	TCPNOTCONNECTED	Socket not connected.
-1007	TCPOVERRUN	Data could not be appended to message – no more space in output buffer.
-1008	TCPBUSY	A command is already in progress for this socket.
-1022	TCPSTARTING	Can't perform specified operation – socket service startup in progress.
-1026	TCPINVSTRINGLENGTH	Invalid string length. Length is limited to 0 to 255 characters.

TCP_CLEAR_SEND_BUFFER

Function	INT TCP_CLEAR_SEND_BUFFER(INT handle)
Description	Clears the output buffer for the specified socket.
Parameters	handle: socket handle returned from the `TCP_CREATE` command.
Returns	ReturnVal < `-1` `TCP_CLEAR_SEND_BUFFER` command failed. See error codes below for the specific error. ReturnVal = `0` `TCP_CLEAR_SEND_BUFFER` command completed successfully.
Notes	1. Prior to building a new output message, call this function to empty the output buffer. 2. The socket must be connected, or this command will fail.

Error codes

-1003	TCPINVALIDHANDLE	Invalid handle.
-1006	TCPNOTCONNECTED	Socket not connected.
-1008	TCPBUSY	A command is already in progress for this socket.
-1022	TCPSTARTING	Can’t perform specified operation – socket service startup in progress.

TCP_CLOSE

Function	INT TCP_CLOSE(INT handle)
Description	Disconnects and closes the specified socket.
Parameters	handle: socket handle returned from the `TCP_CREATE` command.
Returns	ReturnVal < `-1` `TCP_CLOSE` command failed. See error codes below for the specific error. ReturnVal = `0` TCP/IP Socket service is closing this socket. Call `WAIT_TCP_CLOSE` to wait for the `TCP_CLOSE` function call to complete.
Notes	The application **MUST** call `WAIT_TCP_CLOSE` to wait for the `TCP_CLOSE` command to complete. If the application attempts to create the same socket before the socket closure completes, the `TCP_CREATE` function call will fail.

Error codes

-1003	TCPINVALIDHANDLE	Invalid handle.
-1022	TCPSTARTING	Can't perform specified operation – socket service startup in progress.

TCP_CONNECT

Function	INT TCP_CONNECT(INT handle, STRING remote_sock_addr,BYTE mode, DWORD timeout)	
Description	Creates a connection between the local socket and the specified remote socket. Once connected, the application can send/receive data through the connection.	
Parameters	handle:	socket handle returned from the `TCP_CREATE` command.
	remote_sock_addr:	The remote socket address of the form: "`a.b.c.d:p`"
	mode:	0 = Client Application. The `TCP_CONNECT` command will actively attempt to connect to the specified remote socket. 1 = Server Application. The `TCP_CONNECT` command will wait for a connection request from the specified remote socket.
	timeout:	Timeout, in seconds. 0 means infinite. Specifies how long to wait for the socket to connect before returning an error.

Returns

ReturnVal < `-1` `TCP_CONNECT` failed. See the error codes below for the specific error.

ReturnVal = `0` Socket is connecting. Call `WAIT_TCP_CONNECT` to wait for the `TCP_CONNECT` command to complete.

Notes

1. Call `WAIT_TCP_CONNECT` to wait for the `TCP_CONNECT` command to complete.

2. To create a connection between two sockets, one socket must connect as a Client -- the other as a Server. Client sockets actively initiate connections. Server sockets wait for connection requests. If both sockets are configured as a Client socket, or both as a Server Socket, then the sockets will **not** connect.

3. The application **must** connect the socket before it can send/receive data.

Error codes	-1000	TCPINVALIDADDRESS	Invalid socket address.
	-1003	TCPINVALIDHANDLE	Invalid handle.
	-1004	TCPINVALIDMODE	Invalid connect mode.
	-1005	TCPALREADYCONN	Socket already connected.
	-1017	TCPNOTCREATED	Socket not created yet, or is being closed.
	-1022	TCPSTARTING	Can’t perform specified operation – socket service startup in progress.

TCP_CREATE

Function	INT TCP_CREATE(STRING local_sock_addr, BYTE sock_type,WORD in_size, WORD out_size)	
Description	Creates an ASIC-200 or ASIC-300 TCP/IP Socket.	
Parameters	`local_sock_addr:`	The local socket address of the form: "`a.b.c.d:p`"
	sock_type:	0 = Datagram
		1 = Stream
	in_size:	Size of the input buffer, in bytes. Range 1 – 65535 bytes. Default is 1024 bytes.
	out_size:	Size of the output buffer, in bytes. Range 1 – 65535 bytes. Default is 1024 bytes.

Returns

ReturnVal < `-1` `TCP_CREATE` failed. See error codes below for the specific error.

ReturnVal > `0` Socket is being created. Value returned is the socket handle. Call `WAIT_TCP_CREATE` to wait for the command to complete.

Notes

1. Call `WAIT_TCP_CREATE` to wait for the command to complete.

2. **Datagram Socket**: Provides a packet-based connection where datagrams are exchanged through the socket. Datagrams are buffers of a fixed (typically small) maximum length. The maximum length is determined by the physical network (ethernet, token-ring, etc.). Because UDP is used for data delivery, the connection is unreliable. The application must be able to handle lost or duplicate packets.

3. **Streams Socket**: Provides a reliable byte stream connection. It uses TCP (transport) for data delivery, which ensures reliable communications. For ASIC to ASIC communications, always use Streams Sockets.

Error codes	-1000	TCPINVALIDADDRESS	Invalid socket address.
	-1001	TCPINVALIDSOCKTYPE	Invalid socket type.
	-1002	TCPINVALIDBUFSIZE	Invalid buffer size.
	-1013	TCPNOHANDLES	No more API handles available.
	-1014	TCPNOTHREAD	Could not create worker thread for socket.
	-1015	TCPDUPADDRESS	Local socket address already in use.
	-1022	TCPSTARTING	Can't perform specified operation – socket service startup in progress.

TCP_EXTRACT_*datatype*

Functions

INT TCP_EXTRACT_INT (INT handle, BOOL conversion)

INT TCP_EXTRACT_DINT (INT handle, BOOL conversion)

REAL TCP_EXTRACT_REAL (INT handle)

REAL TCP_EXTRACT_LREAL (INT handle)

BYTE TCP_EXTRACT_BYTE (INT handle)

WORD TCP_EXTRACT_WORD (INT handle, BOOL conversion)

DWORD TCP_EXTRACT_DWORD (INT handle, BOOL conversion)

Description

Extracts the next data item from the message in the input buffer. A subsequent extract function call will extract the data item following this item. The valid data types are:

TYPE	SIGN	SIZE
INT	Signed	2 bytes
DINT	Signed	4 bytes
REAL	Signed	4 bytes
LREAL	Signed	8 bytes
BYTE	Unsigned	1 byte
WORD	Unsigned	2 bytes
DWORD	Unsigned	4 bytes

Parameters	`Handle:`	socket handle returned from the `TCP_CREATE` command.
	`Conversion:`	Data conversion switch: FALSE = no conversion TRUE = convert from network byte order

Returns

If successful, returns the requested data item.

On error, a value of zero is returned. Call `TCP_GET_EXTRACT_ERROR` to get the specific error. See error codes below.

Notes

1. Use these extraction functions to sequentially extract each data item from a received message, starting at the beginning of the message. After a new message is received via a `TCP_RECV_BUFFER` function call, data extraction begins at the beginning of the message. The contents of the input buffer remain intact until the next receive function call. After data is extracted from the buffer, the application can call `TCP_RESET_RECV_BUFFER` to reset extraction to the beginning of the message.

2. The socket must be connected, or this command will fail.

3. The received string MUST be NULL terminated (00h) and 0 to 255 characters in length or the EXTRACT function will fail.

Error codes

-1006	TCPNOTCONNECTED	Socket not connected.
-1008	TCPBUSY	A command is already in progress for this socket.
-1009	TCPOUTOFDATA	Not enough input data to fulfill this request.
-1026	TCPINVSTRINGLENGTH	A received string must not exceed 255 characters in length

TCP_GET_EXTRACT_ERROR

Function INT TCP_GET_EXTRACT_ERROR(INT handle)

Description Returns the error code associated with the last `TCP_EXTRACT` operation on the specified socket.

Parameters handle: socket handle returned from the `TCP_CREATE` command.

Returns ReturnVal < `-1` The last extraction command did not complete successfully. The value returned is the specific error code.

ReturnVal = `0` The last extraction command completed without errors.

Error codes

-1006	TCPNOTCONNECTED	Socket not connected.
-1008	TCPBUSY	A command is already in progress for this socket.
-1009	TCPOUTOFDATA	Not enough input data to fulfill this request.
-1026	TCPINVSTRINGLENGTH	A received string must not exceed 255 characters in length

TCP_RECV_BUFFER

Function INT TCP_RECV_BUFFER(INT handle, WORD count, DWORD timeout)

Description Receives message data into the input buffer.

Parameters

handle:	socket handle returned from the `TCP_CREATE` command.
count:	Number of bytes to receive.
	Ignored for Datagram sockets.
timeout:	Timeout, in seconds. 0 means infinite. Specifies how long to wait for the requested data.

Returns ReturnVal < `-1` `TCP_RECV_BUFFER` failed. See the error codes below for the specific error.

ReturnVal = `0` Socket is attempting to receive a message. Call `WAIT_TCP_RECV_BUFFER` to wait for the `TCP_RECV_BUFFER` command to complete.

Notes 1. Call `WAIT_TCP_RECV_BUFFER` to wait for the `TCP_RECV_BUFFER` command to complete.

2. Data in the input buffer is lost – replaced by any newly received data.

3. For Datagram sockets, a message is sent over the physical link as a single packet. This same packet is also received as an integral unit. Therefore, `TCP_RECV_BUFFER` returns the entire packet in the input buffer. If the datagram message is too large to fit into the input buffer, the recv will fail. The `count` parameter is ignored for Datagram sockets.

4. For Datagram sockets, messages are sent using UDP. This protocol does not guarantee delivery of messages. It is the responsibility of the application to handle lost packets. For Datagram sockets, you should specify a timeout period to recover from a lost packet error during receive.

5. Use the `TCP_EXTRACT_`*datatype* function calls to extract each data item from the newly received message.

6. The socket must be connected, or this command will fail.

Error codes			
	-1003	TCPINVALIDHANDLE	Invalid handle.
	-1006	TCPNOTCONNECTED	Socket not connected.
	-1008	TCPBUSY	A command is already in progress for this socket.
	-1010	TCPINVALIDCOUNT	Read count invalid or larger then the input buffer size.
	-1022	TCPSTARTING	Can’t perform specified operation – socket service startup in progress.

TCP_RESET_RECV_BUFFER

Function INT TCP_RESET_RECV_BUFFER(INT handle)

Description Resets the pointer to the beginning of the input buffer for the specified socket. The next `TCP_EXTRACT_`*datatype* function call will extract the first data item from the message in the input buffer.

Parameters handle: socket handle returned from the `TCP_CREATE` command.

Returns ReturnVal < `-1` `TCP_RESET_RECV_BUFFER` command failed. See error codes below for the specific error.

ReturnVal = `0` `TCP_RESET_RECV_BUFFER` command completed successfully.

Notes 1. The application can process a received message repeatedly using this reset function in conjunction with the data extraction functions. The contents of the input buffer remain intact until the next recv function is issued.

2. The socket must be connected, or this command will fail.

Error codes

-1003	TCPINVALIDHANDLE	Invalid handle.
-1006	TCPNOTCONNECTED	Socket not connected.
-1008	TCPBUSY	A command is already in progress for this socket.
-1022	TCPSTARTING	Can't perform specified operation – socket service startup in progress.

TCP_SEND_BUFFER

Function	INT TCP_SEND_BUFFER(INT handle)
Description	Sends the message contained in the output buffer.
Parameters	handle: socket handle returned from the `TCP_CREATE` command.
Returns	ReturnVal < `-1` `TCP_SEND_BUFFER` failed. See the error codes below for the specific error. ReturnVal = `0` Socket is sending the message. Call `WAIT_TCP_SEND_BUFFER` to wait for the `TCP_SEND_BUFFER` command to complete.
Notes	1. Call `WAIT_TCP_SEND_BUFFER` to wait for the `TCP_SEND_BUFFER` command to complete. 2. For Datagram sockets, the message is sent over the physical link as a single packet. The message length should not exceed the maximum datagram size for the physical link (ethernet, token-ring, etc) or the send may fail. FYI: maximum amount of data that can be transferred on Ethernet is 1500 bytes. 3. The socket must be connected, or this command will fail.

Error codes

-1003	TCPINVALIDHANDLE	Invalid handle.
-1006	TCPNOTCONNECTED	Socket not connected.
-1008	TCPBUSY	A command is already in progress for this socket.
-1019	TCPNODATA	No data to send.
-1022	TCPSTARTING	Can’t perform specified operation – socket service startup in progress.

TCP_START_SOCKET_SERVICE

Function INT TCP_START_SOCKET_SERVICE()

Description Initializes the TCP/IP Socket service. The application MUST call this function prior to calling any of the other API functions.

Call `WAIT_TCP_START_SOCKET_SERVICE` to wait for the `TCP_START_SOCKET_SERVICE` function call to complete.

Parameters None

Returns None

Notes 1. Call WAIT_TCP_START_SOCKET_SERVICE to wait for the TCP_START_SOCKET_SERVICE function call to complete.

2. This function closes all open sockets. Calling this function at the beginning of the application ensures that all sockets that were opened during a previous execution of the application are closed. For example: when a program is aborted, then restarted.

3. Failure to call this function may cause the API functions to fail due to the Windows Socket being opened already (error –1015).

4. Call this function only ONCE during the startup of your application.

WAIT_TCP_CONNECT

Function INT WAIT_TCP_CONNECT(INT handle)

Description Waits for the `TCP_CONNECT` function call to complete.

Parameters handle: socket handle returned from the `TCP_CREATE` command.

Returns ReturnVal < `-1` `TCP_CONNECT` command failed. See error codes below for the specific error.

ReturnVal = `-1` `TCP_CONNECT` command still in progress. Continue to call `WAIT_TCP_CONNECT` to determine when the command has completed.

ReturnVal = `0` `TCP_CONNECT` command completed successfully.

Notes After the `TCP_CONNECT` command has successfully completed, the application can send/receive data through the connection.

Error codes

-1003	TCPINVALIDHANDLE	Invalid handle.
-1012	TCPTIMEOUT	Timeout.
-1018	TCPNOCONNECT	Attempt to wait for connect failed – socket is not being connected.
-1022	TCPSTARTING	Can't perform specified operation – socket service startup in progress.
-10014	WSAEFAULT	The Windows Sockets implementation was unable to allocate needed resources for its internal operations.
-10024	WSAEMFILE	The queue is nonempty upon entry to accept and there are no descriptors available.
-10049	WSAEADDRNOTAVAIL	The specified address is not available from the local machine.
-10050	WSAENETDOWN	The network subsystem has failed.

-10051	WSAENETUNREACH	The network cannot be reached from this host at this time.
-10055	WSAENOBUFS	No buffer space is available. The socket cannot be connected.
-10056	WSAEISCONN	The socket is already connected (connection-oriented sockets only).
-10061	WSAECONNREFUSED	The attempt to connect was forcefully rejected.

WAIT_TCP_CLOSE

Function	INT WAIT_TCP_CLOSE(INT handle)
Description	Waits for the `TCP_CLOSE` function call to complete.
Parameters	handle: socket handle returned from the `TCP_CREATE` command.
Returns	ReturnVal < `-1` `TCP_CLOSE` command failed. See error codes below for the specific error.
	ReturnVal = `-1` `TCP_CLOSE` command still in progress. Continue to call `WAIT_TCP_CLOSE` to determine when the command has completed.
	ReturnVal = `0` `TCP_CLOSE` command completed successfully. The specified `handle` is now invalid.

Error codes

-1003	TCPINVALIDHANDLE	Invalid handle.
-1022	TCPSTARTING	Can't perform specified operation – socket service startup in progress.
-1023	TCPNOCLOSE	Attempt to wait for socket closure failed – not closing this socket.

WAIT_TCP_CREATE

Function INT WAIT_TCP_CREATE(INT handle)

Description Waits for the `TCP_CREATE` function call to complete.

Parameters `handle:` socket handle returned from `TCP_CREATE`.

Returns ReturnVal < `-1` `TCP_CREATE` command failed. See error codes below. The `handle` should be closed by calling `TCP_CLOSE`.

ReturnVal = `-1` `TCP_CREATE` command still in progress. Continue to call `WAIT_TCP_CREATE` to determine when the command has completed.

ReturnVal = `0` `TCP_CREATE` command complete.

Notes Create a socket first. Then use `TCP_CONNECT` to connect the socket to a remote host.

Error codes

-1003	TCPINVALIDHANDLE	Invalid handle.
-1016	TCPNOCREATE	Attempt to wait for create failed – socket is not being created.
-1022	TCPSTARTING	Can't perform specified operation – socket service startup in progress.
-1024	TCPOUTOFMEMORY	Out of memory.
-10024	WSAEMFILE	No more socket descriptors are available.
-10050	WSAENETDOWN	The network subsystem or the associated service provider has failed.
-10055	WSAENOBUFS	No buffer space is available. The socket cannot be created.

WAIT_TCP_RECV_BUFFER

Function INT WAIT_TCP_RECV_BUFFER(INT handle)

Description Waits for the `TCP_RECV_BUFFER` function call to complete.

Parameters handle: socket handle returned from the `TCP_CREATE` command.

Returns ReturnVal < `-1` `TCP_RECV_BUFFER` command failed. See error codes below for the specific error.

ReturnVal = `-1` `TCP_RECV_BUFFER` command still in progress. Continue to call `WAIT_TCP_RECV_BUFFER` to determine when the command has completed.

ReturnVal >= `0` `TCP_RECV_BUFFER` command completed successfully. The value returned is the number of bytes read into the input buffer. If the receive command timed-out, the number of bytes returned may be less than the requested amount of data.

Notes The socket must be connected, or this command will fail.

Error codes

-1003	TCPINVALIDHANDLE	Invalid handle.
-1006	TCPNOTCONNECTED	Socket not connected.
-1021	TCPNORECV	Attempt to wait for receive failed – not receiving data on this socket.
-1022	TCPSTARTING	Can't perform specified operation – socket service startup in progress.
-1025	TCPCONNCLOSED	Connection closed.
-10040	WSAEMSGSIZE	The message was too large to fit into the specified buffer and was discarded.
-10050	WSAENETDOWN	The network subsystem has failed.
-10052	WSAENETRESET	The connection has been broken due to the remote host resetting.

-10053	WSAECONNABORTED	The virtual circuit was terminated due to a time-out or other failure. The application should close the socket as it is no longer usable.
-10054	WSAECONNRESET	The virtual circuit was reset by the remote side executing a “hard” or “abortive” close. The application should close the socket as it is no longer usable. On a UDP datagram socket this error would indicate that a previous send operation resulted in an ICMP "Port Unreachable" message.
-10057	WSAENOTCONN	The socket is not connected.
-10060	WSAETIMEDOUT	The connection has been dropped because of a network failure or because the peer system failed to respond.

WAIT_TCP_SEND_BUFFER

Function INT WAIT_TCP_SEND_BUFFER(INT handle)

Description Waits for the `TCP_SEND_BUFFER` function call to complete.

Parameters handle: socket handle returned from the `TCP_CREATE` command.

Returns ReturnVal < `-1` `TCP_SEND_BUFFER` command failed. See error codes below for the specific error.

ReturnVal = `-1` `TCP_SEND_BUFFER` command still in progress. Continue to call `WAIT_TCP_SEND_BUFFER` to determine when the command has completed.

ReturnVal = `0` `TCP_SEND_BUFFER` command completed successfully.

Notes The socket must be connected, or this command will fail.

Error codes

-1003	TCPINVALIDHANDLE	Invalid handle.
-1017	TCPNOTCREATED	Socket not created yet, or is being closed.
-1020	TCPNOSEND	Attempt to wait for send failed – not sending data on this socket.
-1022	TCPSTARTING	Can’t perform specified operation – socket service startup in progress.
-10040	WSAEMSGSIZE	The socket is message oriented, and the message is larger than the maximum supported by the underlying transport.
-10050	WSAENETDOWN	The network subsystem has failed.
-10052	WSAENETRESET	The connection has been broken due to the remote host resetting.
-10053	WSAECONNABORTED	The virtual circuit was terminated due to a time-out or other failure. The application should close the socket as it is no longer usable.

-10054	WSAECONNRESET	The virtual circuit was reset by the remote side executing a “hard” or “abortive” close. For UPD sockets, the remote host was unable to deliver a previously sent UDP datagram and responded with a "Port Unreachable" ICMP packet. The application should close the socket as it is no longer usable.
-10055	WSAENOBUFS	No buffer space is available.
-10057	WSAENOTCONN	The socket is not connected.
-10060	WSAETIMEDOUT	The connection has been dropped, because of a network failure or because the system on the other end went down without notice.
-10065	WSAEHOSTUNREACH	The remote host cannot be reached from this host at this time.

TCP/IP Sockets API Error Codes

-1000	TCPINVALIDADDRESS	Invalid socket address.
-1001	TCPINVALIDSOCKTYPE	Invalid socket type.
-1002	TCPINVALIDBUFSIZE	Invalid buffer size.
-1003	TCPINVALIDHANDLE	Invalid handle.
-1004	TCPINVALIDMODE	Invalid connect mode.
-1005	TCPALREADYCONN	Socket already connected.
-1006	TCPNOTCONNECTED	Socket not connected.
-1007	TCPOVERRUN	Data could not be appended to message – no more space in output buffer.
-1008	TCPBUSY	A command is already in progress for this socket.
-1009	TCPOUTOFDATA	Not enough input data to fulfill this request.
-1010	TCPINVALIDCOUNT	Read count invalid or larger then the input buffer size.
-1011	TCPOVERFLOW	Message received is too large to fit into the input buffer – packet discarded.
-1012	TCPTIMEOUT	Timeout.
-1013	TCPNOHANDLES	No more API handles available.
-1014	TCPNOTHREAD	Could not create worker thread for socket.
-1015	TCPDUPADDRESS	Local socket address already in use.
-1016	TCPNOCREATE	Attempt to wait for create failed – socket is not being created.
-1017	TCPNOTCREATED	Socket not created yet, or is being closed.
-1018	TCPNOCONNECT	Attempt to wait for connect failed – socket is not being connected.
-1019	TCPNODATA	No data to send.
-1020	TCPNOSEND	Attempt to wait for send failed – not sending data on this socket.
-1021	TCPNORECV	Attempt to wait for receive failed – not receiving data on this socket.
-1022	TCPSTARTING	Can’t perform specified operation – socket service startup in progress.
-1023	TCPNOCLOSE	Attempt to wait for socket closure failed – not closing this socket.
-1024	TCPOUTOFMEMORY	Out of memory.
-1025	TCPCONNCLOSED	Connection closed.
-1026	TCPINVSTRINGLENGTH	A string must not exceed 255 characters in length.

Windows Sockets 2.0 Error Codes

The following information is copyright Microsoft Corporation.

-10004	WSAEINTR	*Interrupted function call.*	A blocking operation was interrupted by a call to WSACancelBlockingCall.
-10013	WSAEACCES	*Permission denied*.	An attempt was made to access a socket in a way forbidden by its access permissions. An example is using a broadcast address for sendto without broadcast permission being set using setsockopt(SO_BROADCAST).
-10014	WSAEFAULT	*Bad address*.	The system detected an invalid pointer address in attempting to use a pointer argument of a call. This error occurs if an application passes an invalid pointer value, or if the length of the buffer is too small. For instance, if the length of an argument which is a struct sockaddr is smaller than sizeof(struct sockaddr).
-10022	WSAEINVAL	*Invalid argument.*	Some invalid argument was supplied (for example, specifying an invalid level to the setsockopt function). In some instances, it also refers to the current state of the socket - for instance, calling accept on a socket that is not listening.
-10024	WSAEMFILE	*Too many open files.*	Too many open sockets. Each implementation may have a maximum number of socket handles available, either globally, per process or per thread.
-10035	WSAEWOULD BLOCK	*Resource temporarily unavailable*.	This error is returned from operations on non-blocking sockets that cannot be completed immediately, for example recv when no data is queued to be read from the socket. It is a non-fatal error, and the operation should be retried later. It is normal for WSAEWOULDBLOCK to be reported as the result from calling connect on a non-blocking SOCK_STREAM socket, since some time must elapse for the connection to be established.

-10036	WSAEINPROGRESS	*Operation now in progress*.	A blocking operation is currently executing. Windows Sockets only allows a single blocking operation to be outstanding per task (or thread), and if any other function call is made (whether or not it references that or any other socket) the function fails with the WSAEINPROGRESS error.
-10037	WSAEALREADY	*Operation already in progress*.	An operation was attempted on a non-blocking socket that already had an operation in progress - i.e. calling connect a second time on a non-blocking socket that is already connecting, or canceling an asynchronous request (WSAAsyncGetXbyY) that has already been canceled or completed.
-10038	WSAENOTSOCK	*Socket operation on non-socket.*	An operation was attempted on something that is not a socket. Either the socket handle parameter did not reference a valid socket, or for select, a member of an fd_set was not valid.
-10039	WSAEDESTADDR REQ	*Destination address required*.	A required address was omitted from an operation on a socket. For example, this error will be returned if sendto is called with the remote address of ADDR_ANY.
-10040	WSAEMSGSIZE	*Message too long*.	A message sent on a datagram socket was larger than the internal message buffer or some other network limit, or the buffer used to receive a datagram into was smaller than the datagram itself.
-10041	WSAEPROTOTYPE	*Protocol wrong type for socket*.	A protocol was specified in the socket function call that does not support the semantics of the socket type requested. For example, the ARPA Internet UDP protocol cannot be specified with a socket type of SOCK_STREAM.
-10042	WSAENOPROTOOPT	*Bad protocol option.*	An unknown, invalid or unsupported option or level was specified in a getsockopt or setsockopt call.
-10043	WSAEPROTONOSUPPORT	*Protocol not supported*.	The requested protocol has not been configured into the system, or no implementation for it exists. For example, a socket call requests a SOCK_DGRAM socket, but specifies a stream protocol.

-10044	WSAESOCKTNOSUPPORT	*Socket type not supported.*	The support for the specified socket type does not exist in this address family. For example, the optional type SOCK_RAW might be selected in a socket call, and the implementation does not support SOCK_RAW sockets at all.
-10045	WSAEOPNOTSUPP	*Operation not supported*.	The attempted operation is not supported for the type of object referenced. Usually this occurs when a socket descriptor to a socket that cannot support this operation, for example, trying to accept a connection on a datagram socket.
-10046	WSAEPFNO SUPPORT	*Protocol family not supported*.	The protocol family has not been configured into the system or no implementation for it exists. Has a slightly different meaning to WSAEAFNOSUPPORT, but is interchangeable in most cases, and all Windows Sockets functions that return one of these specify WSAEAFNOSUPPORT.
-10047	WSAEAFNO SUPPORT	*Address family not supported by protocol family*.	An address incompatible with the requested protocol was used. All sockets are created with an associated "address family" (i.e. AF_INET for Internet Protocols) and a generic protocol type (i.e. SOCK_STREAM). This error will be returned if an incorrect protocol is explicitly requested in the socket call, or if an address of the wrong family is used for a socket, e.g. in sendto.
-10048	WSAEADDRINUSE	*Address already in use*.	Only one usage of each socket address (protocol/IP address/port) is normally permitted. This error occurs if an application attempts to bind a socket to an IP address/port that has already been used for an existing socket, or a socket that wasn't closed properly, or one that is still in the process of closing. For server applications that need to bind multiple sockets to the same port number, consider using setsockopt(SO_REUSEADDR). Client applications usually need not call bind at all - connect will choose an unused port automatically.

-10049	WSAEADDRNOT AVAIL	*Cannot assign requested address*.	The requested address is not valid in its context. Normally results from an attempt to bind to an address that is not valid for the local machine, or connect/sendto an address or port that is not valid for a remote machine (e.g. port 0).
-10050	WSAENETDOWN	*Network is down*.	A socket operation encountered a dead network. This could indicate a serious failure of the network system (i.e. the protocol stack that the WinSock DLL runs over), the network interface, or the local network itself.
-10051	WSAENETUNREACH	*Network is unreachable*.	A socket operation was attempted to an unreachable network. This usually means the local software knows no route to reach the remote host.
-10052	WSAENETRESET	*Network dropped connection on reset*.	The host you were connected to crashed and rebooted. May also be returned by setsockopt if an attempt is made to set SO_KEEPALIVE on a connection that has already failed.
-10053	WSAECONN ABORTED	*Software caused connection abort*.	An established connection was aborted by the software in your host machine, possibly due to a data transmission timeout or protocol error.
-10054	WSAECONNRESET	*Connection reset by peer*.	A existing connection was forcibly closed by the remote host. This normally results if the peer application on the remote host is suddenly stopped, the host is rebooted, or the remote host used a "hard close" (see setsockopt for more information on the SO_LINGER option on the remote socket.)
-10055	WSAENOBUFS	*No buffer space available.*	An operation on a socket could not be performed because the system lacked sufficient buffer space or because a queue was full.
-10056	WSAEISCONN	*Socket is already connected.*	A connect request was made on an already connected socket. Some implementations also return this error if sendto is called on a connected SOCK_DGRAM socket (For SOCK_STREAM sockets, the *to* parameter in sendto is ignored), although other implementations treat this as a legal occurrence.

-10057	WSAENOTCONN	*Socket is not connected*.	A request to send or receive data was disallowed because the socket is not connected and (when sending on a datagram socket using sendto) no address was supplied. Any other type of operation might also return this error - for example, setsockopt setting SO_KEEPALIVE if the connection has been reset.
-10058	WSAESHUTDOWN	*Cannot send after socket shutdown*.	A request to send or receive data was disallowed because the socket had already been shut down in that direction with a previous shutdown call. By calling shutdown a partial close of a socket is requested, which is a signal that sending or receiving or both has been discontinued.
-10060	WSAETIMEDOUT	*Connection timed out.*	A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond.
-10061	WSAECONN REFUSED	*Connection refused*.	No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the foreign host - i.e. one with no server application running.
-10064	WSAEHOSTDOWN	*Host is down*.	A socket operation failed because the destination host was down. A socket operation encountered a dead host. Networking activity on the local host has not been initiated. These conditions are more likely to be indicated by the error WSAETIMEDOUT.
-10065	WSAEHOSTUN REACH	*No route to host*.	A socket operation was attempted to an unreachable host. See WSAENETUNREACH
-10067	WSAEPROCLIM	*Too many processes.*	A Windows Sockets implementation may have a limit on the number of applications that may use it simultaneously. WSAStartup may fail with this error if the limit has been reached.

-10091	WSASYSNOTREADY	*Network subsystem is unavailable.*	This error is returned by WSAStartup if the Windows Sockets implementation cannot function at this time because the underlying system it uses to provide network services is currently unavailable. Users should check: • that the appropriate Windows Sockets DLL file is in the current path, • that they are not trying to use more than one Windows Sockets implementation simultaneously. If there is more than one WINSOCK DLL on your system, be sure the first one in the path is appropriate for the network subsystem currently loaded. • the Windows Sockets implementation documentation to be sure all necessary components are currently installed and configured correctly.
-10092	WSAVERNOT SUPPORTED	*WINSOCK.DLL version out of range.*	The current Windows Sockets implementation does not support the Windows Sockets specification version requested by the application. Check that no old Windows Sockets DLL files are being accessed.
-10093	WSANOT INITIALISED	*Successful WSAStartup not yet performed.*	Either the application hasn't called WSAStartup or WSAStartup failed. The application may be accessing a socket which the current active task does not own (i.e. trying to share a socket between tasks), or WSACleanup has been called too many times.
-10094	WSAEDISCON	*Graceful shutdown in progress.*	Returned by recv, WSARecv to indicate the remote party has initiated a graceful shutdown sequence.
-11001	WSAHOST_NOT_ FOUND	*Host not found.*	No such host is known. The name is not an official hostname or alias, or it cannot be found in the database(s) being queried. This error may also be returned for protocol and service queries, and means the specified name could not be found in the relevant database.

-11002	WSATRY_AGAIN	*Non-authoritative host not found.*	This is usually a temporary error during hostname resolution and means that the local server did not receive a response from an authoritative server. A retry at some time later may be successful.
-11003	WSANO_RECOVERY	*This is a non-recoverable error.*	This indicates some sort of non-recoverable error occurred during a database lookup. This may be because the database files (e.g. BSD-compatible HOSTS, SERVICES or PROTOCOLS files) could not be found, or a DNS request was returned by the server with a severe error.
-11004	WSANO_DATA	*Valid name, no data record of requested type.*	The requested name is valid and was found in the database, but it does not have the correct associated data being resolved for. The usual example for this is a hostname -> address translation attempt (using gethostbyname or WSAAsyncGetHostByName) which uses the DNS (Domain Name Server), and an MX record is returned but no A record - indicating the host itself exists, but is not directly reachable.

Trigonometric and Logarithmic

Introduction

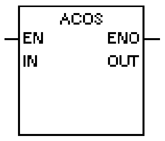
Trigonometric and logarithmic functions include:

Arc Cosine (ACOS)	Computes the arc cosine of a number.
Arc Sine (ASIN)	Computes the arc sine of a number.
Arc Tangent(ATAN)	Computes the arc tangent of a number
Cosine (COS)	Computes the cosine of a number.
Exponent (EXP)	Computes the natural log exponentiation of a number.
Natural Logarithm (LN)	Computes the natural log of a number.
Logarithm (LOG)	Computes the log (base 10) of a number.
Sine (SIN)	Computes the sine of a number.
Tangent (TAN)	Computes the tangent of a number.

Arc Cosine (ACOS)

Description Calculates the arc cosine of the input. The result is in radians.

RLL



ST Function **ACOS(***AnyReal***)**

IL Function **CALC ACOS(OUT:=** *VarReal*, *AnyReal***)**

Where

AnyReal (IN)	Contains the value for which the arc cosine is calculated. If *AnyReal* is out of range for the selected data type, ENO is set FALSE. Data type: ANY_REAL (-1.0 to + 1.0).
return (OUT)	The arc cosine of *AnyReal*. Data type: ANY_REAL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

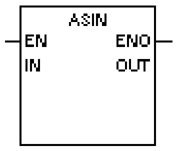
Example angle := ACOS(num);

If num = 0.7071067, then angle = 0.7853975.

Arc Sine (ASIN)

Description Calculates the arc sine of the input. The result is in radians.

RLL



ST Function **ASIN(***AnyReal***)**

IL Function **CALC ASIN(OUT:=** *VarReal***,** *AnyReal***)**

Where	
AnyReal (IN)	Contains the value for which the arc sine is calculated. If *AnyReal* is out of range for the selected data type, ENO is set FALSE. Data type: ANY_REAL (-1.0 to + 1.0).
return (OUT)	The arc sine of *AnyReal*. Data type: ANY_REAL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

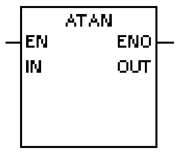
Example angle := ASIN(num);

If num = 0.5, then angle = 0.5235983.

ARC Tangent (ATAN)

Description Calculates the arc tangent of the input. The result is in radians.

RLL



ST Function **ATAN(***AnyReal***)**

IL Function **CALC ATAN(OUT:=** *VarReal*, *AnyReal***)**

Where

AnyReal (IN)	Contains the value for which the arc tangent is calculated. If *AnyReal* is out of range for the selected data type, ENO is set FALSE. Data type: ANY_REAL.
return (OUT)	The arc tangent of *AnyReal*. Data type: ANY_REAL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

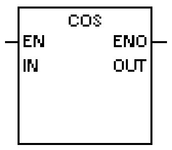
Example angle := ATAN(num);

If num = 1.0, then angle = 0.78539816.

Cosine (COS)

Description Calculates the cosine of the input, which must be in radians.

RLL



ST Function **COS(***AnyReal***)**

IL Function **CALC COS(OUT:=** *VarReal*, *AnyReal***)**

Where	
AnyReal (IN)	Contains the value in radians for which the cosine is calculated. If *AnyReal* is out of range for the selected data type, ENO is set FALSE. Data type: ANY_REAL.
return (OUT)	The cosine of *AnyReal*. Data type: ANY_REAL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example

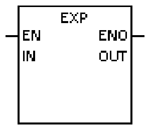
```
cosangle := COS(angle);
```

If angle = 6.0, then cosangle = 0.96017029.

Exponential (EXP)

Description Calculates the natural log exponentiation of the input value (raises *e* to the power of the input).

RLL



ST Function **EXP (***AnyReal***)**

IL Function **CALC EXP(OUT:=** *VarReal*, *AnyReal***)**

Where	
AnyReal (IN)	Contains the value used as the exponent for e. Data type: ANY_REAL.
return (OUT)	The result of e raised to the power of *AnyReal*. Data type: ANY_REAL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

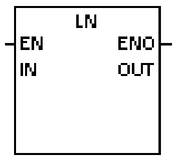
Example newval := EXP (val);

If val = 2.5 then newval = 12.182494.

Natural Log (LN)

Description Calculates the natural logarithm of a value.

RLL



ST Function **LN (***AnyReal***)**

IL Function **CALC LN(OUT:=** *VarReal*, *AnyReal***)**

Where

AnyReal (IN)	Contains the value for which the natural logarithm is calculated. If *AnyReal* is out of range for the selected data type, ENO is set FALSE. Data type: ANY_REAL.
return (OUT)	The natural logarithm of *AnyReal* Data type: ANY_REAL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

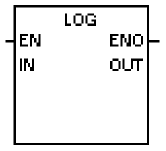
Example ln_num := LN (num);

If num = 674.3, then ln_num = 6.51368.

Logarithm (LOG)

Description Calculates the base 10 logarithm of a value.

RLL



ST Function **LOG (***AnyReal***)**

IL Function **CALC LOG(OUT:=** *VarReal*, *AnyReal***)**

Where

AnyReal (IN)	Contains the value for which the logarithm is calculated. If *AnyReal* is out of range for the selected data type, ENO is set FALSE. Data type: ANY_REAL.
return (OUT)	The logarithm of *AnyReal*. Data type: ANY_REAL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example

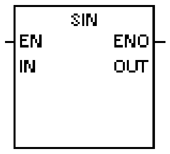
```
lognum := LOG (num);
```

If num = 14.0, then lognum = 1.146128.

Sine (SIN)

Description Calculates the sine of the input, which must be in radians.

RLL



ST Function **SIN(***AnyReal***)**

IL Function **CALC SIN(OUT:=** *VarReal*, *AnyReal***)**

Where

AnyReal (IN)	Contains the value in radians for which the sine is calculated. If *AnyReal* is out of range for the selected data type, ENO is set FALSE. Data type: ANY_REAL.
return (OUT)	The sine of *AnyReal*. Data type: ANY_REAL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

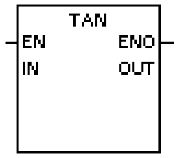
Example sinangle := SIN(angle);

If angle = 6.0, then sinangle = -0.27941550.

Tangent (TAN)

Description Calculates the tangent of the input, which must be in radians.

RLL



ST Function **TAN(***AnyReal***)**

IL Function **CALC TAN(OUT:=** *VarReal*, *AnyReal***)**

Where

AnyReal (IN)	Contains the value in radians for which the tangent is calculated. If *AnyReal* is out of range for the selected data type, ENO is set FALSE. Data type: ANY_REAL.
return (OUT)	The tangent of *AnyReal*. Data type: ANY_REAL.

Notes

1. Refer to **Function Execution Control** for a description of using the EN input and ENO output.
2. Refer to **Instruction List** for information on using functions with the Instruction List language.

Example tanngle := TAN(angle);

If angle = 6.0, then tanngle = -0.2910062.

Index

139183(C)

Xycom Automation, LLC.
750 North Maple Road
Saline, MI 48176–129

734-429-4971

Fax: 734-429-1010

http://www.profaceamerica.com