Table 7- 41 Data types for parameters

Parameter	Data type[1]	Description
IN1 and IN2	SInt, Int, DInt, USInt, UInt, UDInt, Constant	Modulo inputs
OUT	SInt, Int, DInt, USInt, UInt, UDInt	Modulo output

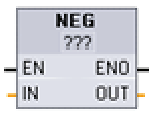
[1] The IN1, IN2, and OUTparameters must be the same data type.

Table 7- 42 ENO values

ENO	Description
1	No error
0	Value IN2 = 0, OUT is assigned the value zero

7.5.4 Negation instruction

Table 7- 43 NEG instruction

LAD / FBD	SCL	Description
	`-(in);`	The NEG instruction inverts the arithmetic sign of the value at parameter IN and stores the result in parameter OUT.

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 44 Data types for parameters

Parameter	Data type[1]	Description
IN	SInt, Int, DInt, Real, LReal, Constant	Math operation input
OUT	SInt, Int, DInt, Real, LReal	Math operation output

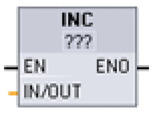
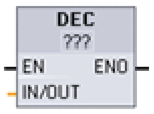
[1] The IN and OUT parameters must be the same data type.

Table 7- 45 ENO status

ENO	Description
1	No error
0	The resulting value is outside the valid number range of the selected data type. Example for SInt: NEG (-128) results in +128 which exceeds the data type maximum.

7.5.5 Increment and decrement instructions

Table 7- 46 INC and DEC instructions

LAD / FBD	SCL	Description
	`in_out := in_out + 1;`	Increments a signed or unsigned integer number value: IN_OUT value +1 = IN_OUT value
	`in_out := in_out - 1;`	Decrements a signed or unsigned integer number value: IN_OUT value - 1 = IN_OUT value

1 For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 47 Data types for parameters

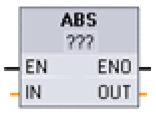
Parameter	Data type	Description
IN/OUT	SInt, Int, DInt, USInt, UInt, UDInt	Math operation input and output

Table 7- 48 ENO status

ENO	Description
1	No error
0	The resulting value is outside the valid number range of the selected data type. Example for SInt: INC (+127) results in +128, which exceeds the data type maximum.

7.5.6 Absolute value instruction

Table 7- 49 ABS instruction

LAD / FBD	SCL	Description
	`out := ABS(in);`	Calculates the absolute value of a signed integer or real number at parameter IN and stores the result in parameter OUT.

1 For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 50 Data types for parameters

Parameter	Data type[1]	Description
IN	SInt, Int, DInt, Real, LReal	Math operation input
OUT	SInt, Int, DInt, Real, LReal	Math operation output

[1] The IN and OUT parameters must be the same data type.

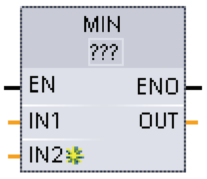
Table 7- 51 ENO status

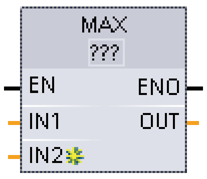
ENO	Description
1	No error
0	The math operation result value is outside the valid number range of the selected data type. Example for SInt: ABS (-128) results in +128 which exceeds the data type maximum.

7.5.7 Minimum and Maximum instructions

Table 7- 52 MIN and MAX instructions

LAD / FBD	SCL	Description

	`out:= MIN(`
	`    in1:=_variant_in_,`
	`    in2:=_variant_in_`
	`   [,...in32]);`
		The MIN instruction compares the value of two parameters IN1 and IN2 and assigns the minimum (lesser) value to parameter OUT.

	`out:= MAX(`
	`    in1:=_variant_in_,`
	`    in2:=_variant_in_`
	`   [,...in32]);`
		The MAX instruction compares the value of two parameters IN1 and IN2 and assigns the maximum (greater) value to parameter OUT.

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 53 Data types for the parameters

Parameter	Data type[1]	Description
IN1, IN2 [...IN32]	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Constant	Math operation inputs (up to 32 inputs)
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Math operation output

[1] The IN1, IN2, and OUT parameters must be the same data type.

To add an input, click the "Create" icon or right-click on an input stub for one of the existing IN parameters and select the "Insert input" command.

To remove an input, right-click on an input stub for one of the existing IN parameters (when there are more than the original two inputs) and select the "Delete" command.

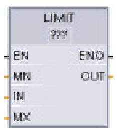
Table 7- 54 ENO status

ENO	Description
1	No error
0	For Real data type only: • One or more inputs is not a real number (NaN). • The resulting OUT is +/- INF (infinity).

7.5.8 Limit instruction

Table 7- 55 LIMIT instruction

LAD / FBD	SCL	Description

	`LIMIT(MN:=_variant_in_,`
	`IN:=_variant_in_,`
	`MX:=_variant_in_,`
	`OUT:=_variant_out_);`
		The Limit instruction tests if the value of parameter IN is inside the value range specified by parameters MIN and MAX and if not, clamps the value at MIN or MAX.

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 56 Data types for the parameters

Parameter	Data type[1]	Description
MN, IN, and MX	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Constant	Math operation inputs
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Math operation output

[1] The MN, IN, MX, and OUTparameters must be the same data type.

If the value of parameter IN is within the specified range, then the value of IN is stored in parameter OUT. If the value of parameter IN is outside of the specified range, then the OUT value is the value of parameter MIN (if the IN value is less than the MIN value) or the value of parameter MAX (if the IN value is greater than the MAX value).

Table 7- 57 ENO status

ENO	Description
1	No error
0	Real: If one or more of the values for MIN, IN and MAX is NaN (Not a Number), then NaN is returned.
0	If MIN is greater than MAX, the value IN is assigned to OUT.

SCL examples:

- MyVal := LIMIT(MN:=10,IN:=53, MX:=40); //Result: MyVal = 40
- MyVal := LIMIT(MN:=10,IN:=37, MX:=40); //Result: MyVal = 37
- MyVal := LIMIT(MN:=10,IN:=8, MX:=40); //Result: MyVal = 10

7.5.9 Floating-point math instructions

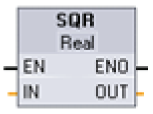
You use the floating point instructions to program mathematical operations using a Real or LReal data type:

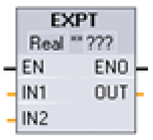
- SQR: Square (IN^2 = OUT)
- SQRT: Square root (√IN = OUT)
- LN: Natural logarithm (LN(IN) = OUT)
- EXP: Natural exponential (e^{IN} =OUT), where base e = 2.71828182845904523536
- EXPT: General exponential ($IN1^{IN2}$ = OUT)

  EXPT parameters IN1 and OUT are always the same data type, for which you must select Real or LReal. You can select the data type for the exponent parameter IN2 from among many data types.
- FRAC: Fraction (fractional part of floating point number IN = OUT)
- SIN: Sine (sin(IN radians) = OUT)
  ASIN: Inverse sine (arcsine(IN) = OUT radians), where the sin(OUT radians) = IN
- COS: Cosine (cos(IN radians) = OUT)
  ACOS: Inverse cosine (arccos(IN) = OUT radians), where the cos(OUT radians) = IN
- TAN: Tangent (tan(IN radians) = OUT)
  ATAN: Inverse tangent (arctan(IN) = OUT radians), where the tan(OUT radians) = IN

Table 7- 58 Examples of floating-point math instructions

LAD / FBD	SCL	Description

	`out := SQR(in);`
	or
	`out := in * in;`
		Square: IN ^{2} = OUT
		For example: If IN = 9, then OUT = 81.

	`out := in1 ** in2;`	General exponential: IN1 ^{IN2} = OUT For example: If IN1 = 3 and IN2 = 2, then OUT = 9.

1 For LAD and FBD: Click the "???" (by the instruction name) and select a data type from the drop-down menu.

2 For SCL: You can also use the basic SCL math operators to create the mathematical expressions.

Table 7- 59 Data types for parameters

Parameter	Data type	Description
IN, IN1	Real, LReal, Constant	Inputs
IN2	SInt, Int, DInt, USInt, UInt,UDInt, Real, LReal, Constant	EXPT exponent input
OUT	Real, LReal	Outputs

Table 7- 60 ENO status

ENO	Instruction	Condition	Result (OUT)
1	All	No error	Valid result
0	SQR	Result exceeds valid Real/LReal range	+INF
		IN is +/- NaN (not a number)	+NaN
	SQRT	IN is negative	-NaN
		IN is +/- INF (infinity) or +/- NaN	+/- INF or +/- NaN
	LN	IN is 0.0, negative, -INF, or -NaN	-NaN
		IN is +INF or +NaN	+INF or +NaN
	EXP	Result exceeds valid Real/LReal range	+INF
		IN is +/- NaN	+/- NaN
	SIN, COS, TAN	IN is +/- INF or +/- NaN	+/- INF or +/- NaN
	ASIN, ACOS	IN is outside valid range of -1.0 to +1.0	+NaN
		IN is +/- NaN	+/- NaN
	ATAN	IN is +/- NaN	+/- NaN
	FRAC	IN is +/- INF or +/- NaN	+NaN
	EXPT	IN1 is +INF and IN2 is not -INF	+INF
		IN1 is negative or -INF	+NaN if IN2 is Real/LReal, -INF otherwise
		IN1 or IN2 is +/- NaN	+NaN

ENO	Instruction	Condition	Result (OUT)
		IN1 is 0.0 and IN2 is Real/LReal (only)	+NaN

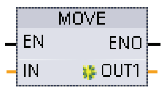
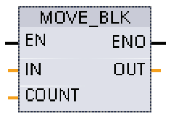
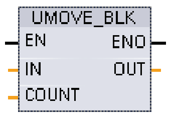
7.6 Move

7.6.1 Move and block move instructions

Use the Move instructions to copy data elements to a new memory address and convert from one data type to another. The source data is not changed by the move process.

- The MOVE instruction copies a single data element from the source address specified by the IN parameter to the destination addresses specified by the OUT parameter.
- The MOVE_BLK and UMOVE_BLK instructions have an additional COUNT parameter. The COUNT specifies how many data elements are copied. The number of bytes per element copied depends on the data type assigned to the IN and OUT parameter tag names in the PLC tag table.

Table 7- 61 MOVE, MOVE_BLK and UMOVE_BLK instructions

LAD / FBD	SCL	Description
	`out1 := in;`	Copies a data element stored at a specified address to a new address or multiple addresses.[1]
	`MOVE_BLK( in:=_variant_in, count:=_uint_in, out=>_variant_out);`	Interruptible move that copies a block of data elements to a new address.
	`UMOVE_BLK( in:=_variant_in, count:=_uint_in, out=>_variant_out);`	Uninterruptible move that copies a block of data elements to a new address.

[1] MOVE instruction: To add another output in LAD or FBD, click the "Create" icon by the output parameter. For SCL, use multiple assignment statements. You might also use one of the loop constructions.

Table 7- 62 Data types for the MOVE instruction

Parameter	Data type	Description
IN	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Char, Array, Struct, DTL, Time	Source address
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Char, Array, Struct, DTL, Time	Destination address

To add MOVE outputs, click the "Create" icon or right-click on an output stub for one of the existing OUT parameters and select the "Insert output" command.

To remove an output, right-click on an output stub for one of the existing OUT parameters (when there are more than the original two outputs) and select the "Delete" command.

Table 7- 63 Data types for the MOVE_BLK and UMOVE_BLK instructions

Parameter	Data type	Description
IN	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal Byte, Word, DWord	Source start address
COUNT	UInt	Number of data elements to copy
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord	Destination start address

Note

Rules for data copy operations

- To copy the Bool data type, use SET_BF, RESET_BF, R, S, or output coil (LAD) (Page 178)
- To copy a single elementary data type, use MOVE
- To copy an array of an elementary data type, use MOVE_BLK or UMOVE_BLK
- To copy a structure, use MOVE
- To copy a string, use S_MOVE (Page 254)
- To copy a single character in a string, use MOVE
- The MOVE_BLK and UMOVE_BLK instructions cannot be used to copy arrays or structures to the I, Q, or M memory areas.

MOVE_BLK and UMOVE_BLK instructions differ in how interrupts are handled:

- Interrupt events are **queued and processed** during MOVE_BLK execution. Use the MOVE_BLK instruction when the data at the move destination address is not used within an interrupt OB subprogram or, if used, the destination data does not have to be consistent. If a MOVE_BLK operation is interrupted, then the last data element moved is complete and consistent at the destination address. The MOVE_BLK operation is resumed after the interrupt OB execution is complete.
- Interrupt events are **queued but not processed** until UMOVE_BLK execution is complete. Use the UMOVE_BLK instruction when the move operation must be completed and the destination data consistent, before the execution of an interrupt OB subprogram. For more information, see the section on data consistency (Page 153).

ENO is always true following execution of the MOVE instruction.

Table 7- 64 ENO status

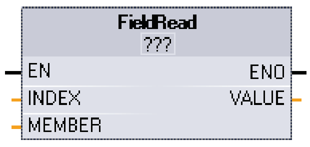
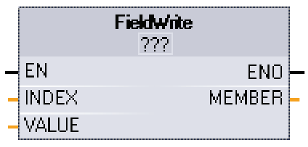
ENO	Condition	Result
1	No error	All COUNT elements were successfully copied.
0	Either the source (IN) range or the destination (OUT) range exceeds the available memory area.	Elements that fit are copied. No partial elements are copied.

7.6.2 FieldRead and FieldWrite instructions

Note

STEP 7 V10.5 **did not support** a variable reference as an array index or multi-dimensional arrays. The FieldRead and FieldWrite instructions were used to provide variable array index operations for a one-dimensional array. STEP 7 V11 **does support** a variable as an array index and multi-dimensional arrays. FieldRead and FieldWrite are included in STEP 7 V11 for backward compatibility with programs that have used these instructions.

Table 7- 65 FieldRead and FieldWrite instructions

LAD / FBD	SCL	Description
	`value := member[index];`	FieldRead reads the array element with the index value INDEX from the array whose first element in specified by the MEMBER parameter. The value of the array element is transferred to the location specified at the VALUE parameter.
	`member[index] := value;`	WriteField transfers the value at the location specified by the VALUE parameter to the array whose first element is specified by the MEMBER parameter. The value is transferred to the array element whose array index is specified by the INDEX parameter.

1 For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 66 Data types for parameters

Parameter and type		Data type	Description
Index	Input	DInt	The index number of the array element to be read or written to
Member [1]	Input	Array element types: Bool, Byte, Word, DWord, Char, SInt, Int, Dint, USInt, UInt, UDInt, Real, LReal	Location of the first element in a one-dimension array defined in a global data block or block interface. For example: If the array index is specified as [-2..4], then the index of the first element is -2 and not 0.
Value [1]	Out	Bool, Byte, Word, DWord, Char, SInt, Int, Dint, USInt, UInt, UDInt, Real, LReal	Location to which the specified array element is copied (FieldRead) Location of the value that is copied to the specified array element (FieldWrite)

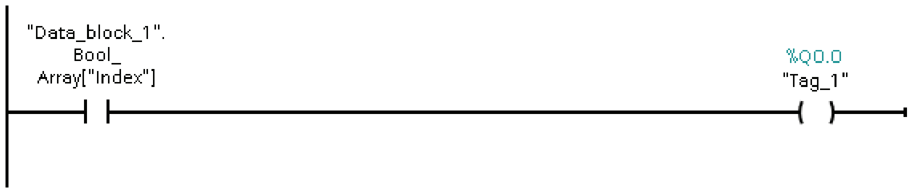
[1] The data type of the array element specified by the MEMBER parameter and the VALUE parameter must have the same data type.

The enable output ENO = 0, if one of the following conditions applies:

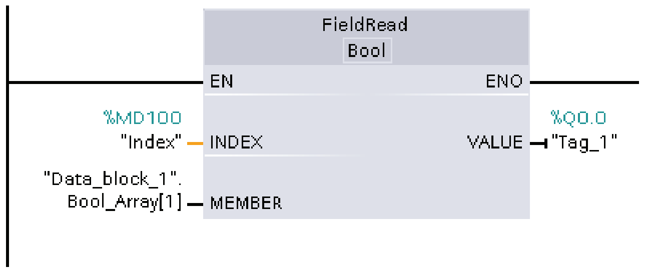
- The EN input has signal state "0"
- The array element specified at the INDEX parameter is not defined in the array referenced at MEMBER parameter
- Errors such as an overflow occur during processing

Accessing data by array indexing

To access elements of an array with a variable, simply use the variable as an array index in your program logic. For example, the network below sets an output based on the Boolean value of an array of Booleans in "Data_block_1" referenced by the PLC tag "Index".



The logic with the variable array index is equivalent to the former method using the FieldRead instruction:



FieldWrite and FieldRead instructions can be replaced with variable array indexing logic.

SCL has no FieldRead or FieldWrite instructions, but supports indirect addressing of an array with a variable:
`#Tag_1 := "Data_block_1".Bool_Array[#Index];`

7.6.3 Fill instructions

Table 7- 67 FILL_BLK and UFILL_BLK instructions

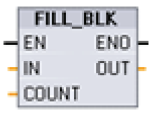
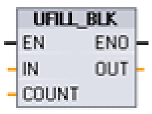
LAD / FBD	SCL	Description
	`FILL_BLK( in:=_variant_in, count:=int, out=>_variant_out);`	Interruptible fill instruction: Fills an address range with copies of a specified data element
	`UFILL_BLK( in:=_variant_in, count:=int out=>_variant_out);`	Uninterruptible fill instruction: Fills an address range with copies of a specified data element

Table 7- 68 Data types for parameters

Parameter	Data type	Description
IN	SInt, Int, DIntT, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord	Data source address
COUNT	USInt, UInt	Number of data elements to copy
OUT	SInt, Int, DIntT, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord	Data destination address

Note

Rules for data fill operations

- To fill with the BOOL data type, use SET_BF, RESET_BF, R, S, or output coil (LAD)
- To fill with a single elementary data type, use MOVE
- To fill an array with an elementary data type, use FILL_BLK or UFILL_BLK
- To fill a single character in a string, use MOVE
- The FILL_BLK and UFILL_BLK instructions cannot be used to fill arrays in the I, Q, or M memory areas.

The FILL_BLK and UFILL_BLK instructions copy the source data element IN to the destination where the initial address is specified by the parameter OUT. The copy process repeats and a block of adjacent addresses is filled until the number of copies is equal to the COUNT parameter.

FILL_BLK and UFILL_BLK instructions differ in how interrupts are handled:

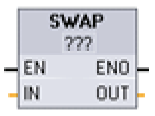
- Interrupt events are **queued and processed** during FILL_BLK execution. Use the FILL_BLK instruction when the data at the move destination address is not used within an interrupt OB subprogram or, if used, the destination data does not have to be consistent.
- Interrupt events are **queued but not processed** until UFILL_BLK execution is complete. Use the UFILL_BLK instruction when the move operation must be completed and the destination data consistent, before the execution of an interrupt OB subprogram.

Table 7- 69 ENO status

ENO	Condition	Result
1	No error	The IN element was successfully copied to all COUNT destinations.
0	The destination (OUT) range exceeds the available memory area	Elements that fit are copied. No partial elements are copied.

7.6.4 Swap instruction

Table 7- 70 SWAP instruction

LAD / FBD	SCL	Description
	`out := SWAP(in);`	Reverses the byte order for two-byte and four-byte data elements. No change is made to the bit order within each byte. ENO is always TRUE following execution of the SWAP instruction.

1 For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 71 Data types for the parameters

Parameter	Data type	Description
IN	Word, DWord	Ordered data bytes IN
OUT	Word, DWord	Reverse ordered data bytes OUT

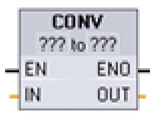
Example 1	Parameter IN = MB0 (before execution)		Parameter OUT = MB4, (after execution)	
Address	MB0	MB1	MB4	MB5
W#16#1234	12	34	34	12
WORD	MSB	LSB	MSB	LSB

Example 2	Parameter IN = MB0 (before execution)				Parameter OUT = MB4, (after execution)			
Address	MB0	MB1	MB2	MB3	MB4	MB5	MB6	MB7
DW#16# 12345678	12	34	56	78	78	56	34	12
DWORD	MSB			LSB	MSB			LSB

7.7 Convert

7.7.1 CONV instruction

Table 7- 72 Convert (CONV) instruction

LAD / FBD	SCL	Description
	`out := <data type in>_TO_<data type out>(in);`	Converts a data element from one data type to another data type.

1 For LAD and FBD: Click the "???" and select the data types from the drop-down menu.

2 For SCL: Construct the conversion instruction by identifying the data type for the input parameter (in) and output parameter (out). For example, DWORD_TO_REAL converts a DWord value to a Real value.

Table 7- 73 Data types for the parameters

Parameter	Data type	Description
IN	Bit string[1], SInt, USInt, Int, UInt, DInt, UDInt, Real, LReal, BCD16, BCD32	Input value
OUT	Bit string[1], SInt, USInt, Int, UInt, DInt, UDInt, Real, LReal, BCD16, BCE32	Input value converted to a new data type

1 The instruction does not allow you to select Bit strings (Byte, Word, DWord). To enter an operand of data type Byte, Word, or DWord for a parameter of the instruction, select an unsigned integer with the same bit length. For example, select USInt for a Byte, UInt for a Word, or UDInt for a DWord.

After you select the (convert from) data type, a list of possible conversions is shown in the (convert to) dropdown list. Conversions from and to BCD16 are restricted to the Int data type. Conversions from and to BCD32 are restricted to the DInt data type.

Table 7- 74 ENO status

ENO	Description	Result OUT
1	No error	Valid result
0	IN is +/- INF or +/- NaN	+/- INF or +/- NaN
0	Result exceeds valid range for OUT data type	OUT is set to the least-significant bytes of IN

7.7.2 Conversion instructions for SCL

Conversion instructions for SCL

Table 7- 75 Conversion from a Bool, Byte, Word, or DWord

Data type	Instruction	Result
Bool	`BOOL_TO_BYTE, BOOL_TO_WORD, BOOL_TO_DWORD, BOOL_TO_INT, BOOL_TO_DINT`	The value is transferred to the least significant bit of the target data type.
Byte	`BYTE_TO_BOOL`	The lowest bit is transferred into the destination data type.
	`BYTE_TO_WORD, BYTE_TO_DWORD`	The value is transferred to the low byte of the target data type.
	`BYTE_TO_SINT, BYTE_TO_USINT`	The value is transferred to the target data type.
	`BYTE_TO_INT, BYTE_TO_UINT, BYTE_TO_DINT, BYTE_TO_UDINT`	The value is transferred to the least significant byte of the target data type.
Word	`WORD_TO_BOOL`	The lowest bit is transferred into the destination data type.
	`WORD_TO_BYTE`	The low byte of the source value is transferred to the target data type
	`WORD _TO_DWORD`	The value is transferred to the low byte of the target data type.
	`WORD _TO_SINT, WORD _TO_USINT`	The low byte of the source value is transferred to the target data type.
	`WORD _TO_INT, WORD _TO_UINT`	The value is transferred to the target data type.
	`WORD _TO_DINT, WORD _TO_UDINT`	The value is transferred to the low byte of the target data type.
DWord	`DWORD_TO_BOOL`	The lowest bit is transferred into the destination data type.
	`DWORD_TO_BYTE, DWORD_TO_WORD, DWORD_TO_SINT, DWORD_TO_USINT, DWORD_TO_INT, DWORD_TO_UINT`	The low byte of the source value is transferred to the target data type.
	`DWORD_TO_DINT, DWORD_TO_UDINT, DWORD_TO_REAL`	The value is transferred to the target data type.

Table 7- 76 Conversion from a short integer (SInt or USInt)

Data type	Instruction	Result
SInt	`SINT_TO_BOOL`	The lowest bit is transferred into the destination data type.
	`SINT_TO_BYTE`	The value is transferred to the target data type
	`SINT_TO_WORD, SINT_TO_DWORD, SINT_TO_INT, SINT_TO_DINT`	The value is transferred to the low byte of the target data type.
	`SINT_TO_USINT, SINT_TO_UINT, SINT_TO_UDINT, SINT_TO_REAL, SINT_TO_LREAL, SINT_TO_CHAR, SINT_TO_STRING`	The value is converted.
USInt	`USINT_TO_BOOL`	The lowest bit is transferred into the destination data type.
	`USINT_TO_BYTE`	The value is transferred to the target data type
	`USINT_TO_WORD, USINT_TO_DWORD, USINT_TO_INT, USINT_TO_UINT, USINT_TO_DINT, USINT_TO_UDINT`	The value is transferred to the low byte of the target data type.
	`USINT_TO_SINT, USINT_TO_REAL, USINT_TO_LREAL, USINT_TO_CHAR, USINT_TO_STRING`	The value is converted.

Table 7- 77 Conversion from an integer (Int or UInt)

Data type	instruction	Result
Int	`INT_TO_BOOL`	The lowest bit is transferred into the destination data type.
	`INT_TO_BYTE, INT_TO_DWORD, INT_TO_SINT, INT_TO_USINT, INT_TO_UINT, INT_TO_UDINT, INT_TO_REAL, INT_TO_LREAL, INT_TO_CHAR, INT_TO_STRING`	The value is converted.
	`INT_TO_WORD`	The value is transferred to the target data type.
	`INT_TO_DINT`	The value is transferred to the low byte of the target data type.
UInt	`UINT_TO_BOOL`	The lowest bit is transferred into the destination data type.
	`UINT_TO_BYTE, UINT_TO_SINT, UINT_TO_USINT, UINT_TO_INT, UINT_TO_REAL, UINT_TO_LREAL, UINT_TO_CHAR, UINT_TO_STRING`	The value is converted.
	`UINT_TO_WORD, UINT_TO_DATE`	The value is transferred to the target data type.
	`UINT_TO_DWORD, UINT_TO_DINT, UINT_TO_UDINT`	The value is transferred to the low byte of the target data type.

Table 7- 78 Conversion from a double integer (Dint or UDInt)

Data type	Instruction	Result
DInt	`DINT_TO_BOOL`	The lowest bit is transferred into the destination data type.
	`DINT_TO_BYTE, DINT_TO_WORD, DINT_TO_SINT, DINT_TO_USINT, DINT_TO_INT, DINT_TO_UINT, DINT_TO_UDINT, DINT_TO_REAL, DINT_TO_LREAL, DINT_TO_CHAR, DINT_TO_STRING`	The value is converted.
	`DINT_TO_DWORD, DINT_TO_TIME`	The value is transferred to the target data type.
UDInt	`UDINT_TO_BOOL`	The lowest bit is transferred into the destination data type.
	`UDINT_TO_BYTE, UDINT_TO_WORD, UDINT_TO_SINT, UDINT_TO_USINT, UDINT_TO_INT, UDINT_TO_UINT, UDINT_TO_DINT, UDINT_TO_REAL, UDINT_TO_LREAL, UDINT_TO_CHAR, UDINT_TO_STRING`	The value is converted.
	`UDINT_TO_DWORD, UDINT_TO_TOD`	The value is transferred to the target data type.

Table 7- 79 Conversion from a Real number (Real or LReal)

Data type	Instruction	Result
Real	`REAL_TO_DWORD, REAL_TO_LREAL`	The value is transferred to the target data type.
	`REAL_TO_SINT, REAL_TO_USINT, REAL_TO_INT, REAL_TO_UINT, REAL_TO_DINT, REAL_TO_UDINT, REAL_TO_STRING`	The value is converted.
LReal	`LREAL_TO_SINT, LREAL_TO_USINT, LREAL_TO_INT, LREAL_TO_UINT, LREAL_TO_DINT, LREAL_TO_UDINT, LREAL_TO_REAL, LREAL_TO_STRING`	The value is converted.

Table 7- 80 Conversion from Time, DTL, TOD or Date

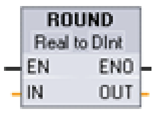
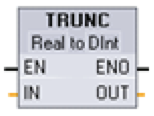
Data type	Instruction	Result
Time	`TIME_TO_DINT`	The value is transferred to the target data type.
DTL	`DTL_TO_DATE, DTL_TO_TOD`	The value is converted.
TOD	`TOD_TO_UDINT`	The value is converted.
Date	`DATE_TO_UINT`	The value is converted.

Table 7- 81 Conversion from a Char or String

Data type	Instruction	Result
Char	`CHAR_TO_SINT, CHAR_TO_USINT, CHAR_TO_INT, CHAR_TO_UINT, CHAR_TO_DINT, CHAR_TO_UDINT`	The value is converted.
	`CHAR_TO_STRING`	The value is transferred to the first character of the string.
String	`STRING_TO_SINT, STRING_TO_USINT, STRING_TO_INT, STRING_TO_UINT, STRING_TO_DINT, STRING_TO_UDINT, STRING_TO_REAL, STRING_TO_LREAL`	The value is converted.
	`STRING_TO_CHAR`	The first character of the string is copied to the Char.

7.7.3 Round and truncate instructions

Table 7- 82 ROUND and TRUNC instructions

LAD / FBD	SCL	Description
	`out := ROUND (in);`	Converts a real number to an integer. The default data type is DINT. When the output is a valid data type other than DINT, it must be declared explicitly; for example, ROUND_REAL or ROUND_LREAL. The real number fraction is rounded to the nearest integer value (IEEE - round to nearest). If the number is exactly one-half the span between two integers (for example, 10.5), then the number is rounded to the even integer. For example: • ROUND (10.5) = 10 • ROUND (11.5) = 12
	`out := TRUNC(in);`	TRUNC converts a real number to an integer. The fractional part of the real number is truncated to zero (IEEE - round to zero).

[1] For LAD and FBD: Click the "???" (by the instruction name) and select a data type from the drop-down menu.

Table 7- 83 Data types for the parameters

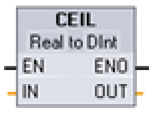
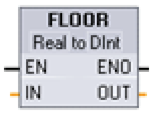
Parameter	Data type	Description
IN	Real, LReal	Floating point input
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Rounded or truncated output

Table 7- 84 ENO status

ENO	Description	Result OUT
1	No error	Valid result
0	IN is +/- INF or +/- NaN	+/- INF or +/- NaN

7.7.4 Ceiling and floor instructions

Table 7- 85 CEIL and FLOOR instructions

LAD / FBD	SCL	Description
	`out := CEIL(in);`	Converts a real number (Real or LReal) to the closest integer greater than or equal to the selected real number (IEEE "round to +infinity").
	`out := FLOOR(in);`	Converts a real number (Real or LReal) to the closest integer smaller than or equal to the selected real number (IEEE "round to -infinity").

1 For LAD and FBD: Click the "???" (by the instruction name) and select a data type from the drop-down menu.

Table 7- 86 Data types for the parameters

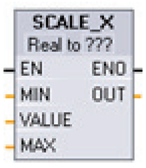
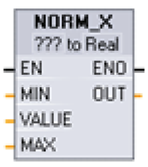
Parameter	Data type	Description
IN	Real, LReal	Floating point input
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Converted output

Table 7- 87 ENO status

ENO	Description	Result OUT
1	No error	Valid result
0	IN is +/- INF or +/- NaN	+/- INF or +/- NaN

7.7.5 Scale and normalize instructions

Table 7- 88 SCALE_X and NORM_X instructions

LAD / FBD	SCL	Description
 SCALE_X Real to ??? EN ENO MIN OUT VALUE MAX	`out :=SCALE_X(min:=_in_, value:=_in_, max:=_in_);`	Scales the normalized real parameter VALUE where (0.0 <= VALUE <= 1.0) in the data type and value range specified by the MIN and MAX parameters: OUT = VALUE (MAX - MIN) + MIN
 NORM_X ??? to Real EN ENO MIN OUT VALUE MAX	`out :=NORM_X(min:=_in_, value:=_in_, max:=_in_);`	Normalizes the parameter VALUE inside the value range specified by the MIN and MAX parameters: OUT = (VALUE - MIN) / (MAX - MIN), where (0.0 <= OUT <= 1.0)

1 For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 89 Data types for the parameters

Parameter	Data type[1]	Description
MIN	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Input minimum value for range
VALUE	SCALE_X: Real, LReal NORM_X: SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Input value to scale or normalize
MAX	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Input maximum value for range
OUT	SCALE_X: SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal NORM_X: Real, LReal	Scaled or normalized output value

1 For SCALE_X: Parameters MIN, MAX, and OUTmust be the same data type.
For NORM_X: Parameters MIN, VALUE, and MAXmust be the same data type.

Note

SCALE_X parameter VALUE should be restricted to (0.0 <= VALUE <= 1.0)

If parameter VALUE is less than 0.0 or greater than 1.0:

- The linear scaling operation can produce OUT values that are less than the parameter MIN value or above the parameter MAX value for OUT values that fit within the value range of the OUT data type. SCALE_X execution sets ENO = TRUE for these cases.
- It is possible to generate scaled numbers that are not within the range of the OUT data type. For these cases, the parameter OUT value is set to an intermediate value equal to the least-significant portion of the scaled real number prior to final conversion to the OUT data type. SCALE_X execution sets ENO = FALSE in this case.

NORM_X parameter VALUE should be restricted to (MIN <= VALUE <= MAX)

If parameter VALUE is less than MIN or greater than MAX, the linear scaling operation can produce normalized OUT values that are less than 0.0 or greater than 1.0. NORM_X execution sets ENO = TRUE in this case.

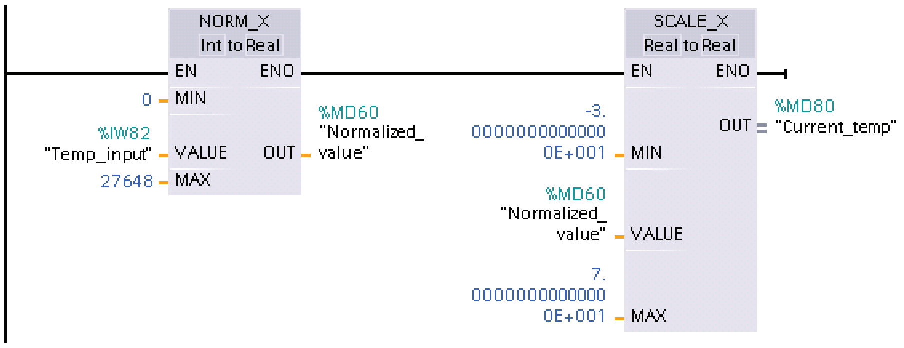
Table 7- 90 ENO status

ENO	Condition	Result OUT
1	No error	Valid result
0	Result exceeds valid range for the OUT data type	Intermediate result: The least-significant portion of a real number prior to final conversion to the OUT data type.
0	Parameters MAX <= MIN	SCALE_X: The least-significant portion of the Real number VALUE to fill up the OUT size. NORM_X: VALUE in VALUE data type extended to fill a double word size.
0	Parameter VALUE = +/- INF or +/- NaN	VALUE is written to OUT

Example (LAD): normalizing and scaling an analog input value

An analog input from an analog signal module or signal board using input in current is in the range 0 to 27648 for valid values. Suppose an analog input represents a temperature where the 0 value of the analog input represents -30.0 degrees C and 27648 represents 70.0 degrees C.

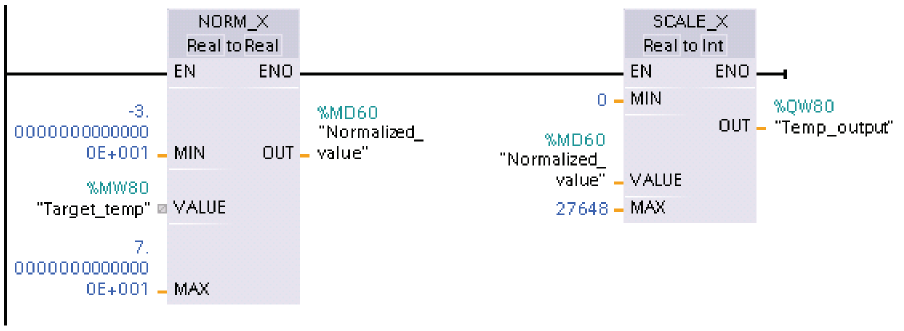
To transform the analog value to the corresponding engineering units, normalize the input to a value between 0.0 and 1.0, and then scale it between -30.0 and 70.0. The resulting value is the temperature represented by the analog input in degrees C:



Note that if the analog input was from an analog signal module or signal board using voltage, the MIN value for the NORM_X instruction would be -27648 instead of 0.

Example (LAD): normalizing and scaling an analog output value

An analog output to be set in an analog signal module or signal board using output in current must be in the range 0 to 27648 for valid values. Suppose an analog output represents a temperature setting where the 0 value of the analog input represents -30.0 degrees C and 27648 represents 70.0 degrees C. To convert a temperature value in memory that is between -30.0 and 70.0 to a value for the analog output in the range 0 to 27648, you must normalize the value in engineering units to a value between 0.0 and 1.0, and then scale it to the range of the analog output, 0 to 27648:



Note that if the analog output was for an analog signal module or signal board using voltage, the MIN value for the SCALE_X instruction would be -27648 instead of 0.

Additional information on analog input representations (Page 770) and analog output representations (Page 771) in both voltage and current can be found in the Technical Specifications.

7.8 Program control

7.8.1 Overview of SCL program control statements

Structured Control Language (SCL) provides three types of program control statements for structuring your user program:

- Selective statements: A selective statement enables you to direct program execution into alternative sequences of statements.
- Loops: You can control loop execution using iteration statements. An iteration statement specifies which parts of a program should be iterated depending on certain conditions.
- Program jumps: A program jump means an immediate jump to a specified jump destination and therefore to a different statement within the same block.

These program control statements use the syntax of the PASCAL programming language.

Table 7- 91 Types of SCL program control statements

Program control statement		Description
Selective	IF-THEN statement (Page 223)	Enables you to direct program execution into one of two alternative branches, depending on a condition being TRUE or FALSE
	CASE statement (Page 224)	Enables the selective execution into 1 of *n* alternative branches, based on the value of a variable
Loop	FOR statement (Page 225)	Repeats a sequence of statements for as long as the control variable remains within the specified value range
	WHILE-DO statement (Page 226)	Repeats a sequence of statements while an execution condition continues to be satisfied
	REPEAT-UNTIL statement (Page 227)	Repeats a sequence of statements until a terminate condition is met
Program jump	CONTINUE statement (Page 227)	Stops the execution of the current loop iteration
	EXIT statement (Page 228)	Exits a loop at any point regardless of whether the terminate condition is satisfied or not
	GOTO statement (Page 229)	Causes the program to jump immediately to a specified label
	IF-THEN statement (Page 223)	Causes the program to exit the block currently being executed and to return to the calling block

See also

RETURN statement (Page 229)

7.8.2 IF-THEN statement

The IF-THEN statement is a conditional statement that controls program flow by executing a group of statements, based on the evaluation of a Bool value of a logical expression. You can also use brackets to nest or structure the execution of multiple IF-THEN statements.

Table 7- 92 Elements of the IF-THEN statement

SCL	Description

`IF "condition" THEN`
`     statement_A;`
`     statement_B;`
`     statement_C;`
`     ;`
	If "condition" is TRUE or 1, then execute the following statements until encountering the END_IF statement.
	If "condition" is FALSE or 0, then skip to END_IF statement (unless the program includes optional ELSIF or ELSE statements).

`[ELSIF "condition-n" THEN`
`    statement_N;`
`    ;]`
	The optional ELSEIF[1] statement provides additional conditions to be evaluated. For example: If "condition" in the IF-THEN statement is FALSE, then the program evaluates "condition-n". If "condition-n" is TRUE, then execute "statement_N".

`[ELSE`
`    statement_X;`
`    ;]`
	The optional ELSE statement provides statements to be executed when the "condition" of the IF-THEN statement is FALSE.

`END_IF;`	The END_IF statement terminates the IF-THEN instruction.

[1] You can include multiple ELSIF statements within one IF-THEN statement.

Table 7- 93 Variables for the IF-THEN statement

Variables	Description
"condition"	Required. The logical expression is either TRUE (1) or FALSE (0).
"statement_A"	Optional. One or more statements to be executed when "condition" is TRUE.
"condition-n"	Optional. The logical expression to be evaluated by the optional ELSIF statement.
"statement_N"	Optional. One or more statements to be executed when "condition-n" of the ELSIF statement is TRUE.
"statement_X"	Optional. One or more statements to be executed when "condition" of the IF-THEN statement is FALSE.

An IF statement is executed according to the following rules:

- The first sequence of statements whose logical expression = TRUE is executed. The remaining sequences of statements are not executed.
- If no Boolean expression = TRUE, the sequence of statements introduced by ELSE is executed (or no sequence of statements if the ELSE branch does not exist).
- Any number of ELSIF statements can exist.

Note

Using one or more ELSIF branches has the advantage that the logical expressions following a valid expression are no longer evaluated in contrast to a sequence of IF statements. The runtime of a program can therefore be reduced.

7.8.3 CASE statement

Table 7- 94 Elements of the CASE statement

SCL	Description

`CASE "Test_Value" OF`
`     "ValueList": Statement[; Statement, ...]`
`     "ValueList": Statement[; Statement, ...]`
`[ELSE`
`Else-statement[; Else-statement, ...]]`
`END_CASE;`
	The CASE statement executes one of several groups of statements, depending on the value of an expression.

Table 7- 95 Parameters

Parameter	Description
"Test_Value"	Required. Any numeric expression of data type Int

"ValueList"	Required. A single value or a comma-separated list of values or ranges of values. (Use two periods to define a range of values: 2..8) The following example illustrates the different variants of the value list:
	1: Statement_A;
	2, 4: Statement _B;
	3, 5..7,9: Statement _C;

Statement	Required. One or more statements that are executed when "Test_Value" matches any value in the value list
Else-statement	Optional. One or more statements that are executed if no match with a value of the "ValueList" stated matches

The CASE statement is executed according to the following rules:

- The selection expression must return a value of the type Int.
- When a CASE statement is processed, the program checks whether the value of the selection expression is contained within a specified list of values. If a match is found, the statement component assigned to the list is executed.
- If no match is found, the program section following ELSE is executed or no statement is executed if the ELSE branch does not exist.

CASE statements can be nested. Each nested case statement must have an associated END_CASE statement.

```
CASE var1 OF
     1 : var2 := "A";
     2 : var2 := "B";
ELSE
     CASE var3 OF
           65..90: var2 := "UpperCase";
           97..122: var2 := "LowerCase";
     ELSE
           var2:= "SpecialCharacter";
```

```
    END_CASE;
END_CASE;
```

7.8.4 FOR statement

Table 7- 96 Elements of the FOR statement

SCL	Description

`FOR "control_variable" := "begin" TO "end"`
`[BY "increment"] DO`
`     statement;`
`     ;`
`END_FOR;`
	A FOR statement is used to repeat a sequence of statements as long as a control variable is within the specified range of values. The definition of a loop with FOR includes the specification of an initial and an end value. Both values must be the same type as the control variable.

Table 7- 97 Parameters

Parameter	Description
"control_variable"	Required. An integer (Int or DInt) that serves as a loop counter
"begin"	Required. Simple expression that specifies the initial value of the control variables
"end"	Required. Simple expression that determines the final value of the control variables
"increment"	Optional. Amount by which a "control variable" is changed after each loop. The "increment" has the same data type as "control variable". If the "increment" value is not specified, then the value of the run tags will be increased by 1 after each loop. You cannot change "increment" during the execution of the FOR statement.

The FOR statement executes as follows:

- At the start of the loop, the control variable is set to the initial value (initial assignment) and each time the loop iterates, it is incremented by the specified increment (positive increment) or decremented (negative increment) until the final value is reached.
- Following each run through of the loop, the condition is checked (final value reached) to establish whether or not it is satisfied. If the condition is satisfied, the sequence of statements is executed, otherwise the loop and with it the sequence of statements is skipped.

Rules for formulating FOR statements:

- The control variable may only be of the data type Int or DInt.
- You can omit the statement BY [increment]. If no increment is specified, it is automatically assumed to be +1.

To end the loop regardless of the state of the "condition" expression, use the EXIT statement (Page 228). The EXIT statement executes the statement immediately following the END_FOR statement.

Use the CONTINUE statement (Page 227) to skip the subsequent statements of a FOR loop and to continue the loop with the examination of whether the condition is met for termination.

7.8.5 WHILE-DO statement

Table 7- 98 WHILE statement

SCL	Description

`WHILE "condition" DO`
`     Statement;`
`     Statement;`
`     ...;`
`END_WHILE;`
	The WHILE statement performs a series of statements until a given condition is TRUE.
	You can nest WHILE loops. The END_WHILE statement refers to the last executed WHILE instruction.

Table 7- 99 Parameters

Parameter	Description
"condition"	Required. A logical expression that evaluates to TRUE or FALSE. (A "null" condition is interpreted as FALSE.)
Statement	Optional. One or more statements that are executed until the condition evaluates to TRUE.

Note

The WHILE statement evaluates the state of "condition" before executing any of the statements. To execute the statements at least one time regardless of the state of "condition", use the REPEAT statement.

The WHILE statement executes according to the following rules:

- Prior to each iteration of the loop body, the execution condition is evaluated.
- The loop body following DO iterates as long as the execution condition has the value TRUE.
- Once the value FALSE occurs, the loop is skipped and the statement following the loop is executed.

To end the loop regardless of the state of the "condition" expression, use the EXIT statement (Page 228). The EXIT statement executes the statement immediately following the END_WHILE statement

Use the CONTINUE statement to skip the subsequent statements of a WHILE loop and to continue the loop with the examination of whether the condition is met for termination.

7.8.6 REPEAT-UNTIL statement

Table 7- 100 REPEAT instruction

SCL	Description

`REPEAT`
`     Statement;`
`     ;`
`UNTIL "condition"`
`END_REPEAT;`
	The REPEAT statement executes a group of statements until a given condition is TRUE.
	You can nest REPEAT loops. The END_REPEAT statement always refers to the last executed Repeat instruction.

Table 7- 101 Parameters

Parameter	Description
Statement	Optional. One or more statements that are executed until the condition is TRUE.
"condition"	Required. One or more expressions of the two following ways: A numeric expression or string expression that evaluates to TRUE or FALSE. A "null" condition is interpreted as FALSE.

Note

Before evaluating the state of "condition", the REPEAT statement executes the statements during the first iteration of the loop (even if "condition" is FALSE). To review the state of "condition" before executing the statements, use the WHILE statement.

To end the loop regardless of the state of the "condition" expression, use the EXIT statement (Page 228). The EXIT statement executes the statement immediately following the END_REPEAT statement

Use the CONTINUE statement (Page 227) to skip the subsequent statements of a REPEAT loop and to continue the loop with the examination of whether the condition is met for termination.

7.8.7 CONTINUE statement

Table 7- 102 CONTINUE statement

SCL	Description

`CONTINUE`
`     Statement;`
`     ;`
	The CONTINUE statement skips the subsequent statements of a program loop (FOR, WHILE, REPEAT) and continues the loop with the examination of whether the condition is met for termination. If this is not the case, the loop continues.

The CONTINUE statement executes according to the following rules:

- This statement immediately terminates execution of a loop body.
- Depending on whether the condition for repeating the loop is satisfied or not the body is executed again or the iteration statement is exited and the statement immediately following is executed.
- In a FOR statement, the control variable is incremented by the specified increment immediately after a CONTINUE statement.

Use the CONTINUE statement only within a loop. In nested loops CONTINUE always refers to the loop that includes it immediately. CONTINUE is typically used in conjunction with an IF statement.

If the loop is to exit regardless of the termination test, use the EXIT statement.

The following example shows the use of the CONTINUE statement to avoid a division-by-0 error when calculating the percentage of a value:

```
FOR x = 0 TO 10 DO
IF value[i] = 0 THEN CONTINUE; END_IF;
     p := part / value[i] * 100;
     s := INT_TO_STRING(p);
     percent=CONCAT(IN1:=s, IN2:="%");
END_FOR;
```

7.8.8 EXIT statement

Table 7- 103 EXIT instruction

SCL	Description
`EXIT;`	An EXIT statement is used to exit a loop (FOR, WHILE or REPEAT) at any point, regardless of whether the terminate condition is satisfied.

The EXIT statement executes according to the following rules:

- This statement causes the repetition statement immediately surrounding the exit statement to be exited immediately.
- Execution of the program is continued after the end of the loop (for example after END_FOR).

Use the EXIT statement within a loop. In nested loops, the EXIT statement returns the processing to the next higher nesting level.

```
FOR i = 0 TO 10 DO
CASE value[i, 0] OF
 1..10: value [i, 1]:="A";
 11..40: value [i, 1]:="B";
 41..100: value [i, 1]:="C";
ELSE
EXIT;
END_CASE;
END_FOR;
```

7.8.9 GOTO statement

Table 7- 104 GOTO statement

SCL	Description

`GOTO JumpLabel;`
`Statement;`
` ... ;`
`JumpLabel: Statement;`
	The GOTO statement skips over statements by jumping to a label in the same block.
	The jump label ("JumpLabel") and the GOTO statement must be in the same block. The name of a jump label can only be assigned once within a block. Each jump label can be the target of several GOTO statements.

It is not possible to jump to a loop section (FOR, WHILE or REPEAT). It is possible to jump from within a loop.

In the following example: Depending on the value of the "Tag_value" operand, the execution of the program resumes at the point defined by the corresponding jump label. If "Tag_value" equals 2, the program execution resumes at the jump label "MyLabel2" and skips "MyLabel1".

```
CASE "Tag_value" OF
1 : GOTO MyLabel1;
2 : GOTO MyLabel2;
ELSE GOTO MyLabel3;
END_CASE;
MyLabel1: "Tag_1" := 1;
MyLabel2: "Tag_2" := 1;
MyLabel3: "Tag_4" := 1;
```

7.8.10 RETURN statement

Table 7- 105 RETURN instruction

SCL	Description
`RETURN;`	The Return instruction exits the code block being executed without conditions. Program execution returns to the calling block or to the operating system (when exiting an OB).

Example of a RETURN instruction:

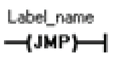
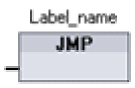
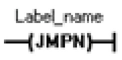
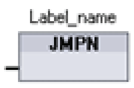
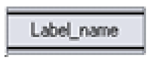
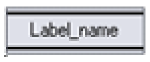
```
IF "Error" <> 0 THEN
RETURN;
END_IF;
```

Note

After executing the last instruction, the code block automatically returns to the calling block. Do not insert a RETURN instruction at the end of the code block.

7.8.11 Jump and label instructions

Table 7- 106 JMP, JMPN, and LABEL instruction

LAD	FBD	SCL	Description
		See the GOTO (Page 229) statement.	If there is power flow to a JMP coil (LAD), or if the JMP box input is true (FBD), then program execution continues with the first instruction following the specified label.
			If there is no power flow to a JMPN coil (LAD), or if the JMPN box input is false (FBD), then program execution continues with the first instruction following the specified label.
			Destination label for a JMP or JMPN jump instruction.

[1] You create your label names by typing in the LABEL instruction directly. Use the parameter helper icon to select the available label names for the JMP and JMPN label name field. You can also type a label name directly into the JMP or JMPN instruction.

Table 7- 107 Data types for the parameters

Parameter	Data type	Description
Label_name	Label identifier	Identifier for Jump instructions and the corresponding jump destination program label

- Each label must be unique within a code block.
- You can jump within a code block, but you cannot jump from one code block to another code block.
- You can jump forward or backward.
- You can jump to the same label from more than one place in the same code block.

7.8.12 JMP_LIST instruction

Table 7- 108 JMP_LIST instruction

LAD / FBD	SCL	Description

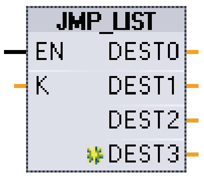
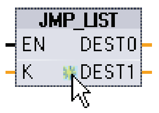
	`CASE k OF`
	`0: GOTO dest0;`
	`1: GOTO dest1;`
	`2: GOTO dest2;`
	`[n: GOTO destn;]`
	`END_CASE;`
		The JMP_LIST instruction acts as a program jump distributor to control the execution of program sections. Depending on the value of the K input, a jump occurs to the corresponding program label. Program execution continues with the program instructions that follow the destination jump label. If the value of the K input exceeds the number of labels - 1, then no jump occurs and processing continues with the next program network.

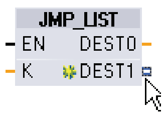
Table 7- 109 Data types for parameters

Parameter	Data type	Description
K	UInt	Jump distributor control value
DEST0, DEST1, .., DESTn.	Program Labels	Jump destination labels corresponding to specific K parameter values: If the value of K equals 0, then a jump occurs to the program label assigned to the DEST0 output. If the value of K equals 1, then a jump occurs to the program label assigned to the DEST1 output, and so on. If the value of the K input exceeds the (number of labels - 1), then no jump occurs and processing continues with the next program network.

For LAD and FBD: The JMP_LIST box is first placed in your program, there are two jump label outputs. You can add or delete jump destinations.



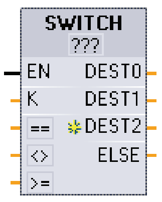
Click the create icon inside the box (on the left of the last DEST parameter) to add new outputs for jump labels.



- Right-click on an output stub and select the "Insert output" command.
- Right-click on an output stub and select the "Delete" command.

7.8.13 SWITCH instruction

Table 7- 110 SWITCH instruction

LAD / FBD	SCL	Description
 SWITCH ??? EN DEST0 K DEST1 == DEST2 <> ELSE >=	`Not available`	The SWITCH instruction acts as a program jump distributor to control the execution of program sections. Depending on the result of comparisons between the value of the K input and the values assigned to the specified comparison inputs, a jump occurs to the program label that corresponds to the first comparison test that is true. If none of the comparisons is true, then a jump to the label assigned to ELSE occurs. Program execution continues with the program instructions that follow the destination jump label.

1 For LAD and FBD: Click below the box name and select a data type from the drop-down menu.

2 For SCL: Use an IF-THEN set of comparisons.

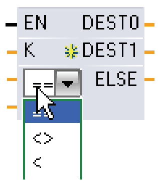
Table 7- 111 Data types for parameters

Parameter	Data type[1]	Description
K	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, TOD, Date	Common comparison value input
==, <>, <, <=, >. >=	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, TOD, Date	Separate comparison value inputs for specific comparison types
DEST0, DEST1, .., DESTn. ELSE	Program Labels	Jump destination labels corresponding to specific comparisons: The comparison input below and next to the K input is processed first and causes a jump to the label assigned to DEST0, if the comparison between the K value and this input is true. The next comparison test uses the next input below and causes a jump to the label assigned to DEST1, if the comparison is true, The remaining comparisons are processed similarly and if none of the comparisons are true, then a jump to the label assigned to the ELSE output occurs.

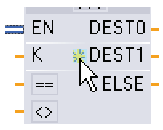
[1] The K input and comparison inputs (==, <>, <, <=, >, >=) must be the same data type.

Adding inputs, deleting inputs, and specifying comparison types

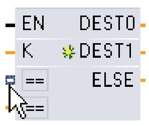
When the LAD or FBD SWITCH box is first placed in your program there are two comparison inputs. You can assign comparison types and add inputs/jump destinations, as shown below.



Click a comparison operator inside the box and select a new operator from the drop-down list.



Click the create icon inside the box (to the left of the last DEST parameter) to add new comparison-destination parameters.



- Right-click on an input stub and select the "Insert input" command.
- Right-click on an input stub and select the "Delete" command.

Table 7- 112 SWITCH box data type selection and allowed comparison operations

Data type	Comparison	Operator syntax
Byte, Word, DWord	Equal	==
	Not equal	<>

Data type	Comparison	Operator syntax
SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Time, TOD, Date	Equal	==
	Not equal	<>
	Greater than or equal	>=
	Less than or equal	<=
	Greater than	>
	Less than	<

SWITCH box placement rules

- No LAD/FBD instruction connection in front of the compare input is allowed.
- There is no ENO output, so only one SWITCH instruction is allowed in a network and the SWITCH instruction must be the last operation in a network.

7.8.14 RET execution control instruction

The optional RET instruction is used to terminate the execution of the current block. If and only if there is power flow to the RET coil (LAD) or if the RET box input is true (FBD), then program execution of the current block will end at that point and instructions beyond the RET instruction will not be executed. If the current block is an OB, the "Return_Value" parameter is ignored. If the current block is a FC or FB, the value of the "Return_Value " parameter is passed back to the calling routine as the ENO value of the called box.

You are not required to use a RET instruction as the last instruction in a block; this is done automatically for you. You can have multiple RET instructions within a single block.

For SCL, see the RETURN (Page 229) statement.

Table 7- 113 Return_Value (RET) execution control instruction

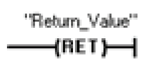
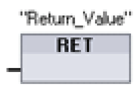
LAD	FBD	SCL	Description
		`RETURN;`	Terminates the execution of the current block

Table 7- 114 Data types for the parameters

Parameter	Data type	Description
Return_Value	Bool	The "Return_value" parameter of the RET instruction is assigned to the ENO output of the block call box in the calling block.

Sample steps for using the RET instruction inside an FC code block:

1. Create a new project and add an FC:
2. Edit the FC:
   - Add instructions from the instruction tree.
   - Add a RET instruction, including one of the following for the "Return_Value" parameter:

     TRUE, FALSE, or a memory location that specifies the required return value.
   - Add more instructions.
3. Call the FC from MAIN [OB1].

The EN input on the FC box in the MAIN code block must be true to begin execution of the FC.

The value specified by the RET instruction in the FC will be present on the ENO output of the FC box in the MAIN code block following execution of the FC for which power flow to the RET instruction is true.

7.8.15 Re-trigger scan cycle watchdog instruction

Table 7- 115 RE_TRIGR instruction

LAD / FBD	SCL	Description
RE_TRIGR EN ENO	`RE_TRIGR();`	RE_TRIGR (Re-trigger scan time watchdog) is used to extend the maximum time allowed before the scan cycle watchdog timer generates an error.

Use the RE_TRIGR instruction to restart the scan cycle monitoring timer during a single scan cycle. This has the effect of extending the allowed maximum scan cycle time by one maximum cycle time period, from the last execution of the RE_TRIGR function.

Note

Prior to S7-1200 CPU firmware version 2.2, RE_TRIGR was restricted to execution from a program cycle OB and could be used to extend the PLC scan time indefinitely. ENO = FALSE and the watchdog timer is not reset when RE_TRIGR was executed from a start up OB, an interrupt OB, or an error OB.

For firmware version 2.2 and later, RE_TRIGR can be executed from any OB (including start up, interrupt, and error OBs). However, the PLC scan can only be extended by a maximum of 10x the configured maximum cycle time.

Setting the PLC maximum cycle time

Configure the value for maximum scan cycle time in the Device configuration for "Cycle time".

Table 7- 116 Cycle time values

Cycle time monitor	Minimum value	Maximum value	Default value
Maximum cycle time	1 ms	6000 ms	150 ms

Watchdog timeout

If the maximum scan cycle timer expires before the scan cycle has been completed, an error is generated. If an error handling code block OB 80 is included in the user program, the CPU executes OB 80 where you may add program logic to create a special reaction. If OB 80 is not included, the first timeout condition is ignored and the CPU goes to STOP.

If a second maximum scan time timeout occurs in the same program scan (2 times the maximum cycle time value), an error is triggered that causes the CPU to transition to STOP mode.

In STOP mode, your program execution stops while CPU system communications and system diagnostics continue.

7.8.16 Stop scan cycle instruction

Table 7- 117 STP instruction

LAD / FBD	SCL	Description
	`STP();`	STP (Stop scan cycle) puts the CPU in STOP mode. When the CPU is in STOP mode, the execution of your program and physical updates from the process image are stopped.

For more information see: Configuring the outputs on a RUN-to-STOP transition (Page 87).

If EN = TRUE, then the CPU goes to STOP mode, the program execution stops, and the ENO state is meaningless. Otherwise, EN = ENO = 0.

7.8.17 Get Error instructions

The get error instructions provide information about program block execution errors. If you add a GetError or GetErrorID instruction to your code block, you can handle program errors within your program block.

GetError

Table 7- 118 GetError instruction

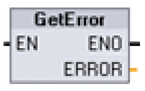
LAD / FBD	SCL	Description
	`GET_ERROR(_out_);`	Indicates that a local program block execution error has occurred and fills a predefined error data structure with detailed error information.

Table 7- 119 Data types for the parameters

Parameter	Data type	Description
ERROR	ErrorStruct	Error data structure: You can rename the structure, but not the members within the structure.

Table 7- 120 Elements of the ErrorStruct data structure

Structure components		Data type	Description
ERROR_ID		Word	Error ID
FLAGS		Byte	Shows if an error occurred during a block call. • 16#01: Error during a block call. • 16#00: No error during a block call.
REACTION		Byte	Default reaction: • 0: Ignore (write error), • 1: Continue with substitute value "0" (read error), • 2: Skip instruction (system error)
CODE_ADDRESS		CREF	Information about the address and type of block
	BLOCK_TYPE	Byte	Type of block where the error occurred: • 1: OB • 2: FC • 3: FB
	CB_NUMBER	UInt	Number of the code block
	OFFSET	UDInt	Reference to the internal memory
MODE		Byte	Access mode: Depending on the type of access, the following information can be output:

Mode	(A)	(B)	(C)	(D)	(E)
0					
1					Offset
2			Area		
3	Location	Scope		Number	
4			Area		Offset
5			Area	DB no.	Offset

Structure components		Data type	Description					
			6	PtrNo. /Acc		Area	DB no.	Offset
			7	PtrNo. / Acc	Slot No. / Scope	Area	DB no.	Offset
OPERAND_NUMBER		UInt	Operand number of the machine command					
POINTER_NUMBER_ LOCATION		UInt	(A) Internal pointer					
SLOT_NUMBER_SCOPE		UInt	(B) Storage area in internal memory					
DATA_ADDRESS		NREF	Information about the address of an operand					
	AREA	Byte	(C) Memory area: • L: 16#40 – 4E, 86, 87, 8E, 8F, C0 – CE • I: 16#81 • Q: 16#82 • M: 16#83 • DB: 16#84, 85, 8A, 8B					
	DB_NUMBER	UInt	(D) Number of the data block					
	OFFSET	UDInt	(E) Relative address of the operand					

GetErrorID

Table 7- 121 GetErrorID instruction

LAD / FBD	SCL	Description
GetErrorID EN ENO ID	`GET_ERR_ID();`	Indicates that a program block execution error has occurred and reports the ID (identifier code) of the error.

Table 7- 122 Data types for the parameters

Parameter	Data type	Description
ID	Word	Error identifier values for the ErrorStruct ERROR_ID member

Table 7- 123 Error_ID values

ERROR_ID Hexadecimal	ERROR_ID Decimal	Program block execution error
0	0	No error
2503	9475	Uninitialized pointer error
2522	9506	Operand out of range read error
2523	9507	Operand out of range write error
2524	9508	Invalid area read error

ERROR_ID Hexadecimal	ERROR_ID Decimal	Program block execution error
2525	9509	Invalid area write error
2528	9512	Data alignment read error (incorrect bit alignment)
2529	9513	Data alignment write error (incorrect bit alignment)
2530	9520	DB write protected
253A	9530	Global DB does not exist
253C	9532	Wrong version or FC does not exist
253D	9533	Instruction does not exist
253E	9534	Wrong version or FB does not exist
253F	9535	Instruction does not exist
2575	9589	Program nesting depth error
2576	9590	Local data allocation error
2942	10562	Physical input point does not exist
2943	10563	Physical output point does not exist

Operation

By default, the CPU responds to a block execution error by logging an error in the diagnostics buffer. However, if you place one or more GetError or GetErrorID instructions within a code block, this block is now set to handle errors within the block. In this case, the CPU does not log an error in the diagnostics buffer. Instead, the error information is reported in the output of the GetError or GetErrorID instruction. You can read the detailed error information with the GetError instruction, or read just the error identifier with GetErrorID instruction. Normally the first error is the most important, with the following errors only consequences of the first error.

The first execution of a GetError or GetErrorID instruction within a block returns the first error detected during block execution. This error could have occurred anywhere between the start of the block and the execution of either GetError or GetErrorID. Subsequent executions of either GetError or GetErrorID return the first error since the previous execution of GetError or GetErrorID. The history of errors is not saved, and execution of either instruction will re-arm the PLC system to catch the next error.

The ErrorStruct data type used by the GetError instruction can be added in the data block editor and block interface editors, so your program logic can access these values. Select ErrorStruct from the data type drop-down list to add this structure. You can create multiple ErrorStruct elements by using unique names. The members of an ErrorStruct cannot be renamed.

Error condition indicated by ENO

If EN = TRUE and GetError or GetErrorID executes, then:

- ENO = TRUE indicates a code block execution error occurred and error data is present
- ENO = FALSE indicates no code block execution error occurred

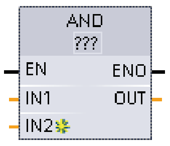
You can connect error reaction program logic to ENO which activates after an error occurs. If an error exists, then the output parameter stores the error data where your program has access to it.

GetError and GetErrorID can be used to send error information from the currently executing block (called block) to a calling block. Place the instruction in the last network of the called block program to report the final execution status of the called block.

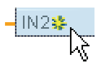
7.9 Word logic operations

7.9.1 AND, OR, and XOR instructions

Table 7- 124 AND, OR, and XOR instruction

LAD / FBD	SCL	Description
	`out := in1 AND in2;`	AND: Logical AND
	`out := in1 OR in2;`	OR: Logical OR
	`out := in1 XOR in2;`	XOR: Logical exclusive OR

1 For LAD and FBD: Click the "???" and select a data type from the drop-down menu.



To add an input, click the "Create" icon or right-click on an input stub for one of the existing IN parameters and select the "Insert input" command.

To remove an input, right-click on an input stub for one of the existing IN parameters (when there are more than the original two inputs) and select the "Delete" command.

Table 7- 125 Data types for the parameters

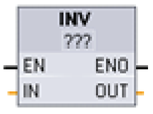
Parameter	Data type	Description
IN1, IN2	Byte, Word, DWord	Logical inputs
OUT	Byte, Word, DWord	Logical output

1 The data type selection sets parameters IN1, IN2, and OUT to the same data type.

The corresponding bit values of IN1 and IN2 are combined to produce a binary logic result at parameter OUT. ENO is always TRUE following the execution of these instructions.

7.9.2 Invert instruction

Table 7- 126 INV instruction

LAD / FBD	SCL	Description
	`Not available`	Calculates the binary one's complement of the parameter IN. The one's complement is formed by inverting each bit value of the IN parameter (changing each 0 to 1 and each 1 to 0). ENO is always TRUE following the execution of this instruction.

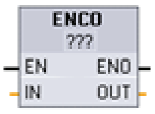
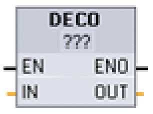
[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 127 Data types for the parameters

Parameter	Data type	Description
IN	SInt, Int, DInt, USInt, UInt, UDInt, Byte, Word, DWord	Data element to invert
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Byte, Word, DWord	Inverted output

7.9.3 Encode and decode instructions

Table 7- 128 ENCO and DECO instruction

LAD / FBD	SCL	Description
	`out := ENCO(_in_);`	Encodes a bit pattern to a binary number The ENCO instruction converts parameter IN to the binary number corresponding to the bit position of the least-significant set bit of parameter IN and returns the result to parameter OUT. If parameter IN is either 0000 0001 or 0000 0000, then a value of 0 is returned to parameter OUT. If the parameter IN value is 0000 0000, then ENO is set to FALSE.
	`out := DECO(_in_);`	Decodes a binary number to a bit pattern The DECO instruction decodes a binary number from parameter IN, by setting the corresponding bit position in parameter OUT to a 1 (all other bits are set to 0). ENO is always TRUE following execution of the DECO instruction. Note: The default data type for the DECO instruction is DWORD. In SCL, change the instruction name to DECO_BYTE or DECO_WORD to decode a byte or word value, and assign to a byte or word tag or address.

[1] For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 129 Data types for the parameters

Parameter	Data type	Description
IN	ENCO: Byte, Word, DWord DECO: UInt	ENCO: Bit pattern to encode DECO: Value to decode
OUT	ENCO: Int DECO: Byte, Word, DWord	ENCO: Encoded value DECO: Decoded bit pattern

Table 7- 130 OUT parameter for ENCO

ENO	Condition	Result (OUT)
1	No error	Valid bit number
0	IN is zero	OUT is set to zero

The DECO parameter OUT data type selection of a Byte, Word, or DWord restricts the useful range of parameter IN. If the value of parameter IN exceeds the useful range, then a modulo operation is performed to extract the least significant bits shown below.

DECO parameter IN range:

- 3 bits (values 0-7) IN are used to set 1 bit position in a Byte OUT
- 4-bits (values 0-15) IN are used to set 1 bit position in a Word OUT
- 5 bits (values 0-31) IN are used to set 1 bit position in a DWord OUT

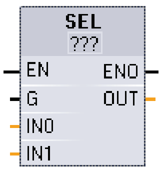
Table 7- 131 Examples

DECO IN value			DECO OUT value (Decode single bit position)
Byte OUT 8 bits	Min. IN	0	00000001
	Max. IN	7	10000000
Word OUT 16 bits	Min. IN	0	0000000000000001
	Max. IN	15	1000000000000000
DWord OUT 32 bits	Min. IN	0	00000000000000000000000000000001
	Max. IN	31	10000000000000000000000000000000

7.9.4 Select, Multiplex, and Demultiplex instructions

Table 7- 132 SEL (select) instruction

LAD / FBD	SCL	Description

	`out := SEL(`
	`    g:=_bool_in,`
	`    in0:-_variant_in,`
	`    in1:=_variant_in);`
		SEL assigns one of two input values to parameter OUT, depending on the parameter G value.

1 For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

Table 7- 133 Data types for the SEL instruction

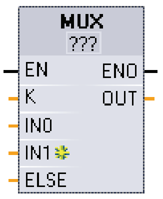
Parameter	Data type [1]	Description
G	Bool	• 0 selects IN0 • 1 selects IN1
IN0, IN1	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, Char	Inputs
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, Char	Output

1 Input variables and the output variable must be of the same data type.

Condition codes: ENO is always TRUE following execution of the SEL instruction.

Table 7- 134 MUX (multiplex) instruction

LAD / FBD	SCL	Description

	`out := MUX(`
	`    k:=_unit_in,`
	`    in1:=variant_in,`
	`    in2:=variant_in,`
	
	`[...in32:=variant_in,]`
	`    inelse:=variant_in);`
		MUX copies one of many input values to parameter OUT, depending on the parameter K value. If the parameter K value exceeds (IN*n* - 1), then the parameter ELSE value is copied to parameter OUT.

1 For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

To add an input, click the "Create" icon or right-click on an input stub for one of the existing IN parameters and select the "Insert input" command.

To remove an input, right-click on an input stub for one of the existing IN parameters (when there are more than the original two inputs) and select the "Delete" command.

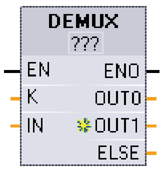
Table 7- 135 Data types for the MUX instruction

Parameter	Data type	Description
K	UInt	• 0 selects IN1 • 1 selects IN2 • *n* selects IN*n*
IN0, IN1, .. INn	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, Char	Inputs
ELSE	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, Char	Input substitute value (optional)
OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, Char	Output

1 Input variables and the output variable must be of the same data type.

Table 7- 136 DEMUX (Demultiplex) instruction

LAD / FBD	SCL	Description

 DEMUX
???
EN
ENO
K
OUT0
IN
OUT1
ELSE
	`DEMUX(`
	`    k:=_unit_in,`
	`    in:=variant_in,`
	`    out1:=variant_in,`
	`    out2:=variant_in,`
	
	`[...out32:=variant_in,]`
	
	`outelse:=variant_in);`
		DEMUX copies the value of the location assigned to parameter IN to one of many outputs. The value of the K parameter selects which output selected as the destination of the IN value. If the value of K is greater than the number (OUT*n* - 1) then the IN value is copied to location assigned to the ELSE parameter.

1 For LAD and FBD: Click the "???" and select a data type from the drop-down menu.

To add an output, click the create icon or right-click on an output stub for one of the existing OUT parameters and select the "Insert output" command. To remove an output, right-click on an output stub for one of the existing OUT parameters (when there are more than the original two outputs) and select the "Delete" command.

To add an output, click the "Create" icon or right-click on an output stub for one of the existing OUT parameters and select the "Insert output" command.

To remove an output, right-click on an output stub for one of the existing OUT parameters (when there are more than the original two outputs) and select the "Delete" command.

Table 7- 137 Data types for the DEMUX instruction

Parameter	Data type [1]	Description
K	UInt	Selector value: • 0 selects OUT1 • 1 selects OUT2 • n selects OUTn
IN	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, Char	Input
OUT0, OUT1, .. OUTn	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, Char	Outputs
ELSE	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal, Byte, Word, DWord, Time, Char	Substitute output when K is greater than (OUTn - 1)

[1] The input variable and the output variables must be of the same data type.

Table 7- 138 ENO status for the MUX and DEMUX instructions

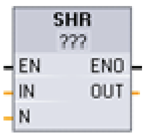
ENO	Condition	Result OUT
1	No error	MUX: Selected IN value is copied to OUT DEMUX: IN value is copied to selected OUT
0	MUX: K is greater than the number of inputs -1	• No ELSE provided: OUT is unchanged, • ELSE provided, ELSE value assigned to OUT
	DEMUX: K is greater than the number of outputs -1	• No ELSE provided: outputs are unchanged, • ELSE provided, IN value copied to ELSE

7.10 Shift and Rotate

7.10.1 Shift instructions

Table 7- 139 SHR and SHL instructions

LAD / FBD	SCL	Description

	`out := SHR(`
	`    in:=_variant_in_,`
	`    n:=_uint_in);`
	`out := SHL(`
	`    in:=_variant_in_,`
	`    n:=_uint_in);`
		Use the shift instructions (SHL and SHR) to shift the bit pattern of parameter IN. The result is assigned to parameter OUT. Parameter N specifies the number of bit positions shifted:
		• SHR: Shift bit pattern right
		• SHL: Shift bit pattern left

[1] For LAD and FBD: Click the "???" and select the data types from the drop-down menu.

Table 7- 140 Data types for the parameters

Parameter	Data type	Description
IN	Byte, Word, DWord	Bit pattern to shift
N	UInt	Number of bit positions to shift
OUT	Byte, Word, DWord	Bit pattern after shift operation

- For N=0, no shift occurs. The IN value is assigned to OUT.
- Zeros are shifted into the bit positions emptied by the shift operation.
- If the number of positions to shift (N) exceeds the number of bits in the target value (8 for Byte, 16 for Word, 32 for DWord), then all original bit values will be shifted out and replaced with zeros (zero is assigned to OUT).
- ENO is always TRUE for the shift operations.

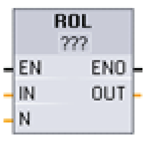
Table 7- 141 SHL example for Word data

Shift the bits of a Word to the left by inserting zeroes from the right (N = 1)			
IN	1110 0010 1010 1101	OUT value before first shift:	1110 0010 1010 1101
		After first shift left:	1100 0101 0101 1010
		After second shift left:	1000 1010 1011 0100
		After third shift left:	0001 0101 0110 1000

7.10.2 Rotate instructions

Table 7- 142 ROR and ROL instructions

LAD / FBD	SCL	Description

	`out := ROL(`
	`    in:=_variant_in_,`
	`    n:=_uint_in);`
	`out := ROR(`
	`    in:=_variant_in_,`
	`    n:=_uint_in);`
		Use the rotate instructions (ROR and ROL) to rotate the bit pattern of parameter IN. The result is assigned to parameter OUT. Parameter N defines the number of bit positions rotated.
		• ROR: Rotate bit pattern right
		• ROL: Rotate bit pattern left

1 For LAD and FBD: Click the "???" and select the data types from the drop-down menu.

Table 7- 143 Data types for the parameters

Parameter	Data type	Description
IN	Byte, Word, DWord	Bit pattern to rotate
N	UInt	Number of bit positions to rotate
OUT	Byte, Word, DWord	Bit pattern after rotate operation

- For N=0, no rotate occurs. The IN value is assigned to OUT.
- Bit data rotated out one side of the target value is rotated into the other side of the target value, so no original bit values are lost.
- If the number of bit positions to rotate (N) exceeds the number of bits in the target value (8 for Byte, 16 for Word, 32 for DWord), then the rotation is still performed.
- ENO is always TRUE following execution of the rotate instructions.

Table 7- 144 ROR example for Word data

Rotate bits out the right -side into the left -side (N = 1)			
IN	0100 0000 0000 0001	OUT value before first rotate:	0100 0000 0000 0001
		After first rotate right:	1010 0000 0000 0000
		After second rotate right:	0101 0000 0000 0000

Extended instructions 8

8.1 Date and time-of-day

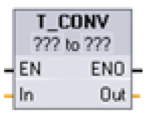
8.1.1 Date and time instructions

Use the date and time instructions to program calendar and time calculations.

- T_CONV converts the data type of a time value: (Time to DInt) or (DInt to Time)
- T_ADD adds Time and DTL values: (Time + Time = Time) or (DTL + Time = DTL)
- T_SUB subtracts Time and DTL values: (Time - Time = Time) or (DTL - Time = DTL)
- T_DIFF provides the difference between two DTL values as a Time value: DTL - DTL = Time
- T_COMBINE combines a Date value and a Time_and_Date value to create a DTL value

For information about the structure of the DTL and Time data, refer to the section on the Time and Date data types (Page 96).

Table 8- 1 T_CONV (Time Convert) instruction

LAD / FBD	SCL	Description
	`out := T_CONV( in:=_variant_in);`	T_CONV converts a Time data type to a DInt data type, or the reverse conversion from DInt data type to Time data type.

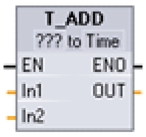
1 For LAD and FBD: Click the "???" and select the data types from the drop-down menu.

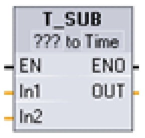
Table 8- 2 Data types for the T_CONV parameters

Parameter and type		Data type	Description
IN	IN	DInt, Time	Input Time value or DInt value
OUT	OUT	DInt, Time	Converted DInt value or Time value

Table 8- 3 T_ADD (Time Add) and T_SUB (Time Subtract) instructions

LAD / FBD	SCL	Description

	`out := T_ADD(`
	`    in1:=_variant_in,`
	`    in2:=_time_in);`
		T_ADD adds the input IN1 value (DTL or Time data types) with the input IN2 Time value. Parameter OUT provides the DTL or Time value result. Two data type operations are possible:
		• Time + Time = Time
		• DTL + Time = DTL

	`out := T_SUB(`
	`    in1:=_variant_in,`
	`    in2:=_time_in);`
		T_SUB subtracts the IN2 Time value from IN1 (DTL or Time value). Parameter OUT provides the difference value as a DTL or Time data type. Two data type operations are possible:
		• Time - Time = Time
		• DTL - Time = DTL

[1] For LAD and FBD: Click the "???" and select the data types from the drop-down menu.

Table 8- 4 Data types for the T_ADD and T_SUB parameters

Parameter and type		Data type	Description
IN1[1]	IN	DTL, Time	DTL or Time value
IN2	IN	Time	Time value to add or subtract
OUT	OUT	DTL, Time	DTL or Time sum or difference

[1] Select the IN1 data type from the drop-down list available below the instruction name. The IN1 data type selection also sets the data type of parameter OUT.

Table 8- 5 T_DIFF (Time Difference) instruction

LAD / FBD	SCL	Description

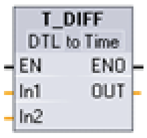
	`out := T_DIFF(`
	`    in1:=_DTL_in,`
	`    in2:=_DTL_in);`
		T_DIFF subtracts the DTL value (IN2) from the DTL value (IN1). Parameter OUT provides the difference value as a Time data type.
		• DTL - DTL = Time

Table 8- 6 Data types for the T_DIFF parameters

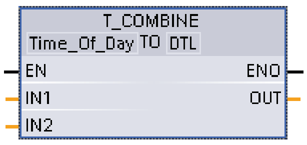
Parameter and type		Data type	Description
IN1	IN	DTL	DTL value
IN2	IN	DTL	DTL value to subtract
OUT	OUT	Time	Time difference

Condition codes: ENO = 1 means no error occurred. ENO = 0 and parameter OUT = 0 errors:

- Invalid DTL value
- Invalid Time value

Table 8- 7 T_COMBINE (combine time values) instruction

LAD / FBD	SCL	Description

 T_COMBINE
Time_Of_Day TO DTL
EN
ENO
IN1
OUT
IN2
	`out := CONCAT_DATE_TOD(`
	`In1 := _date_in,`
	`In2 := _tod_in);`
		T_COMBINE combines a Date value and a Time_of_Day value to create a DTL value.

[1] Note that the T_COMBINEinstruction in the Extended Instructions equates to the CONCAT_DATE_TODfunction in SCL.

Table 8- 8 Data types for the T_COMBINE parameters

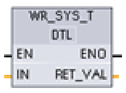
Parameter and type		Data type	Description
IN1	IN	Date	Date value to be combined must be between DATE#1990-01-01 and DATE#2089-12-31
IN2	IN	Time_of_Day	Time_of_Day values to be combined
OUT	OUT	DTL	DTL value

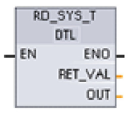
8.1.2 Set and read system clock

Use the clock instructions to set and read the CPU system clock. The data type DTL (Page 96) is used to provide date and time values.

Table 8- 9 System time instructions

LAD / FBD	SCL	Description

	`ret_val := WR_SYS_T(`
	`in:=_DTL_in_);`
		WR_SYS_T (Write System Time) sets the CPU time of day clock with a DTL value at parameter IN. This time value does not include local time zone or daylight saving time offsets.

	`ret_val := RD_SYS_T(`
	`out=>_DTL_out);`
		RD_SYS_T (Read System Time) reads the current system time from the CPU. This time value does not include local time zone or daylight saving time offsets.

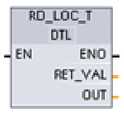
	`ret_val := RD_LOC_T(`
	`out=>_DTL_out);`
		RD_LOC_T (Read Local Time) provides the current local time of the CPU as a DTL data type. This time value reflects the local time zone adjusted appropriately for daylight saving time (if configured).

Table 8- 10 Data types for the parameters

Parameter and type		Data type	Description
IN	IN	DTL	Time of day to set in the CPU system clock
RET_VAL	OUT	Int	Execution condition code
OUT	OUT	DTL	RD_SYS_T: Current CPU system time RD_LOC_T: Current local time. including any adjustment for daylight saving time, if configured

- The local time is calculated by using the time zone and daylight saving time offsets that you set in the device configuration general tab "Time of day" parameters.
- Time zone configuration is an offset to UTC or GMT time.
- Daylight saving time configuration specifies the month, week, day, and hour when daylight saving time begins.
- Standard time configuration also specifies the month, week, day, and hour when standard time begins.
- The time zone offset is always applied to the system time value. The daylight saving time offset is only applied when daylight saving time is in effect.

Note

Daylight saving and standard start time configuration

The "Time of day" properties for "Start for daylight saving time" of the CPU device configuration must be your local time.

Condition codes: ENO = 1 means no error occurred. ENO = 0 means an execution error occurred, and a condition code is provided at the RET_VAL output.

Table 8- 11 Condition codes

RET_VAL (W#16#....)	Description
0000	The current local time is in standard time.
0001	Daylight saving time has been configured, and the current local time is in daylight saving time.
8080	Local time not available
8081	Illegal year value
8082	Illegal month value
8083	Illegal day value
8084	Illegal hour value
8085	Illegal minute value
8086	Illegal second value
8087	Illegal nanosecond value
80B0	The real-time clock has failed.

8.1.3 Run-time meter instruction

Table 8- 12 RTM instruction

LAD / FBD	SCL	Description

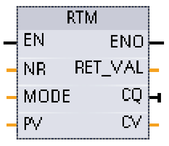
	`RTM(NR:=_uint_in_,`
	`MODE:=_byte_in_,`
	`PV:=_dint_in_,`
	`CQ=>_bool_out_,`
	`CV=>_dint_out_);`
		The RTM (Run Time Meter) instruction can set, start, stop, and read the run-time hour meters in the CPU.

Table 8- 13 Data types for the parameters

Parameter and type		Data type	Description
NR	IN	UInt	Run-time meter number: (possible values: 0..9)
MODE	IN	Byte	RTM Execution mode number: • 0 = Fetch values (the status is then written to CQ and the current value to CV) • 1 = Start (at the last counter value) • 2 = Stop • 4 = Set (to the value specified in PV) • 5 = Set (to the value specified in PV) and then start • 6 = Set (to the value specified in PV) and then stop • 7 = Save all RTM values in the CPU to the MC (Memory Card)
PV	IN	DInt	Preset hours value for the specified run-time meter
RET_VAL	OUT	Int	Function result / error message
CQ	OUT	Bool	Run-time meter status (1 = running)
CV	OUT	DInt	Current run-time hours value for the specified meter

The CPU operates up to 10 run-time hour meters to track the run-time hours of critical control subsystems. You must start the individual hour meters with one RTM execution for each timer. All run-time hour meters are stopped when the CPU makes a run-to-stop transition. You can also stop individual timers with RTM execution mode 2.

When a CPU makes a stop-to-run transition, you must restart the hour timers with one RTM execution for each timer that is started. After a run-time meter value is greater than 2147483647 hours, counting stops and the "Overflow" error is sent. You must execute the RTM instruction once for each timer to reset or modify the timer.

A CPU power failure or power cycle causes a power-down process that saves the current run-time meter values in retentive memory. Upon CPU power-up, the stored run-time meter values are reloaded to the timers and the previous run-time hour totals are not lost. The run-time meters must be restarted to accumulate additional run-time.

Your program can also use RTM execution mode 7 to save the run-time meter values in a memory card. The states of all timers at the instant RTM mode 7 is executed are stored in the memory card. These stored values can become incorrect over time as the hour timers are started and stopped during a program run session. You must periodically update the memory card values to capture important run-time events. The advantage that you get from storing the RTM values in the memory card is that you can insert the memory card in a substitute CPU where your program and saved RTM values will be available. If you did not save the RTM values in the memory card, then the timer values would be lost (in a substitute CPU).

Note

Avoid excessive program calls for memory card write operations

Minimize flash memory card write operations to extend the life of the memory card.

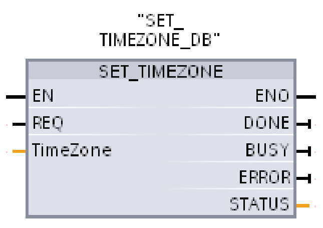
Table 8- 14 Condition codes

RET_VAL (W#16#....)	Description
0	No error
8080	Incorrect run-time meter number
8081	A negative value was passed to the parameter PV
8082	Overflow of the operating hours counter
8091	The input parameter MODE contains an illegal value.
80B1	Value cannot be saved to MC (MODE=7)

8.1.4 SET_TIMEZONE instruction

Table 8- 15 SET_TIMEZONE instruction

LAD / FBD	SCL	Description

 "SET_
TIMEZONE_DB"
SET_TIMEZONE
EN
ENO
REQ
DONE
TimeZone
BUSY
ERROR
STATUS
	`"SET_TIMEZONE_DB"(`
	`REQ:=_bool_in,`
	`Timezone:=_struct_in,`
	`DONE=>_bool_out_,`
	`BUSY=>_bool_out_,`
	`ERROR=>_bool_out_,`
	`STATUS=>_word_out_);`
		Sets the local time zone and daylight saving parameters that are used to transform the CPU system time to local time.

[1] In the SCL example, "SET_TIMEZONE_DB" is the name of the instance DB.

Table 8- 16 Data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	REQ=1: execute function
Timezone	IN	TimeTransformationRule	Rules for the transformation from system time to local time
DONE	OUT	Bool	Function complete
BUSY	OUT	Bool	Function busy
ERROR	OUT	Bool	Error detected
STATUS	OUT	Word	Function result / error message

To manually configure the time zone parameters for the CPU, use the "Time of day" properties of the "General" tab of the device configuration.

Use the SET_TIMEZONE instruction to set the local time configuration programmatically. The parameters of the "TimeTransformationRule" structure specify the local time zone and timing for automatic switching between standard time and daylight saving time.

Table 8- 17 "TimeTransformationRule" structure

Parameter	Data type	Description
Bias	Int	Time difference between UTC and local time [min]
DaylightBias	Int	Time difference between winter and summer time [min]
DaylightStartMonth	USInt	Month of daylight saving time
DaylightStartWeek	USInt	Week of daylight saving time: • 1 = First occurrence of the weekday in the month • ... • 5 = Last occurrence of the weekday of the month
DaylightStartWeekday	USInt	Weekday of daylight saving time: • 1 = Sunday • ... • 7 = Saturday
DaylightStartHour	USInt	Hour of daylight saving time
StandardStartMonth	USInt	Month of switching to winter time
StandardStartWeek	USInt	Week of the changeover to winter time: • 1 = First occurrence of the weekday in the month • ... • 5 = last occurrence of the weekday of the month
StandardStartWeekday	USInt	Weekday of winter time: • 1 = Sunday • ... • 7 = Saturday
StandardStartHour	USInt	Hour of the winter time
Time Zone Name	STRING [80]	Name of the zone: (GMT +01:00) Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna

8.2 String and character

8.2.1 String data overview

String data type

String data is stored as a 2-byte header followed by up to 254 character bytes of ASCII character codes. A String header contains two lengths. The first byte is the maximum length that is given in square brackets when you initialize a string, or 254 by default. The second header byte is the current length that is the number of valid characters in the string. The current length must be smaller than or equal to the maximum length. The number of stored bytes occupied by the String format is 2 bytes greater than the maximum length.

Initialize your String data

String input and output data must be initialized as valid strings in memory, before execution of any string instructions.

Valid String data

A valid string has a maximum length that must be greater than zero but less than 255. The current length must be less than or equal to the maximum length.

Strings cannot be assigned to I or Q memory areas.

For more information see: Format of the String data type (Page 97).

8.2.2 S_MOVE instruction

Table 8- 18 String move instruction

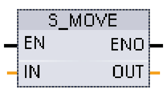
LAD / FBD	SCL	Description
	`out := in;`	Copy the source IN string to the OUT location. S_MOVE execution does not affect the contents of the source string.

Table 8- 19 Data types for the parameters

Parameter	Data type	Description
IN	String	Source string
OUT	String	Target address

If the actual length of the string at the input IN exceeds the maximum length of a string stored at output OUT, then the part of the IN string which can fit in the OUT string is copied.

8.2.3 String conversion instructions

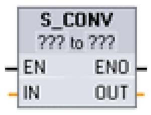
8.2.3.1 String to value and value to string conversions

You can convert number character strings to number values or number values to number character strings with these instructions:

- S_CONV converts (number string to a number value) or (number value to a number string)
- STRG_VAL converts a number string to a number value with format options
- VAL_STRG converts a number value to a number string with format options

S_CONV (String to value conversions)

Table 8- 20 String conversion instruction

LAD / FBD	SCL	Description
	`out := <Type>_TO_<Type>(in);`	Converts a character string to the corresponding value, or a value to the corresponding character string. The S_CONV instruction has no output formatting options. This makes the S_CONV instruction simpler, but less flexible than the STRG_VAL and VAL_STRG instructions.

1 For LAD / FBD: Click the "???" and select the data type from the drop-down list.

2 For SCL: Select S_CONV from the Extended Instructions, and answer the prompts for the data types for the conversion. STEP 7 then provides the appropriate conversion instruction.

Table 8- 21 Data types (string to value)

Parameter and type		Data type	Description
IN	IN	String	Input character string
OUT	OUT	String, Char, SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Output number value

Conversion of the string parameter IN starts at the first character and continues until the end of the string, or until the first character is encountered that is not "0" through "9", "+", "-", or ".". The result value is provided at the location specified in parameter OUT. If the output number value does not fit in the range of the OUT data type, then parameter OUT is set to 0 and ENO is set to FALSE. Otherwise, parameter OUT contains a valid result and ENO is set to TRUE.

Input String format rules:

- If a decimal point is used in the IN string, you must use the "." character.
- Comma characters "," used as a thousands separator to the left of the decimal point are allowed and ignored.
- Leading spaces are ignored.

S_CONV (Value to string conversions)

Table 8- 22 Data types (value to string)

Parameter and type		Data type	Description
IN	IN	String, Char, SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Input number value
OUT	OUT	String	Output character string

An integer, unsigned integer, or floating point value IN is converted to the corresponding character string at OUT. The parameter OUT must reference a valid string before the conversion is executed. A valid string consists of a maximum string length in the first byte, the current string length in the second byte, and the current string characters in the next bytes. The converted string replaces characters in the OUT string starting at the first character and adjusts the current length byte of the OUT string. The maximum length byte of the OUT string is not changed.

How many characters are replaced depends on the parameter IN data type and number value. The number of characters replaced must fit within the parameter OUT string length. The maximum string length (first byte) of the OUT string should be greater than or equal to the maximum expected number of converted characters. The following table shows the maximum possible string lengths required for each supported data type.

Table 8- 23 Maximum string lengths for each data type

IN data type	Maximum number of converted characters in OUT string	Example	Total string length including maximum and current length bytes
USInt	3	255	5
SInt	4	-128	6
UInt	5	65535	7
Int	6	-32768	8
UDInt	10	4294967295	12
DInt	11	-2147483648	13

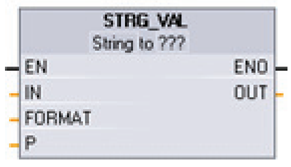
Output String format rules:

- Values written to parameter OUT do not use a leading "+" sign.
- Fixed-point representation is used (no exponential notation).
- The period character "." is used to represent the decimal point when parameter IN is the Real data type.

STRG_VAL instruction

Table 8- 24 String-to-value instruction

LAD / FBD	SCL	Description

 STRG_VAL
String to ???
EN
ENO
IN
OUT
FORMAT
P
	`"STRG_VAL"(`
	`in:=_string_in,`
	`format:=_word_in,`
	`p:=uint_in,`
	`out=>_variant_out);`
		Converts a number character string to the corresponding integer or floating point representation.

1 For LAD / FBD: Click the "???" and select the data type from the drop-down list.

Table 8- 25 Data types for the STRG_VAL instruction

Parameter and type		Data type	Description
IN	IN	String	The ASCII character string to convert
FORMAT	IN	Word	Output format options
P	IN	UInt, Byte, USInt	IN: Index to the first character to be converted (first character = 1)
OUT	OUT	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Converted number value

Conversion begins in the string IN at character offset P and continues until the end of the string, or until the first character is encountered that is not "+", "-", ".", ",", "e", "E", or "0" to "9". The result is placed at the location specified in parameter OUT.

String data must be initialized before execution as a valid string in memory.

The FORMAT parameter for the STRG_VAL instruction is defined below. The unused bit positions must be set to zero.

Table 8- 26 Format of the STRG_VAL instruction

Bit 16							Bit 8	Bit 7							Bit 0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	f	r

f = Notation format	1= Exponential notation 0 = Fixed point notation
r = Decimal point format	1 = "," (comma character) 0 = "." (period character)

Table 8- 27 Values of the FORMAT parameter

FORMAT (W#16#)	Notation format	Decimal point representation
0000 (default)	Fixed point	"."
0001		","
0002	Exponential	"."
0003		","
0004 to FFFF	Illegal values	

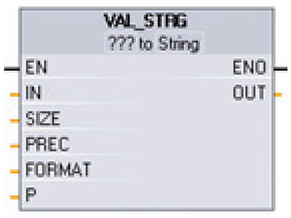
Rules for STRG_VAL conversion:

- If the period character "." is used for the decimal point, then commas "," to the left of the decimal point are interpreted as thousands separator characters. The comma characters are allowed and ignored.
- If the comma character "," is used for the decimal point, then periods "." to the left of the decimal point are interpreted as thousands separator characters. These period characters are allowed and ignored.
- Leading spaces are ignored.

VAL_STRG instruction

Table 8- 28 Value-to-string operation

LAD / FBD	SCL	Description

 VAL_STRG
??? to String
EN
ENO
IN
OUT
SIZE
PREC
FORMAT
P
	`"VAL_STRG"(`
	`    in:=_variant_in,`
	`    size:=_usint_in,`
	`    prec:=_usint_in,`
	`    format:=_word_in,`
	`    p:=uint_in,`
	`    out=>_string_out);`
		Converts an integer, unsigned integer, or floating point value to the corresponding character string representation.

1 For LAD / FBD: Click the "???" and select the data type from the drop-down list.

Table 8- 29 Data types for the VAL_STRG instruction

Parameter and type		Data type	Description
IN	IN	SInt, Int, DInt, USInt, UInt, UDInt, Real, LReal	Value to convert
SIZE	IN	USInt	Number of characters to be written to the OUT string
PREC	IN	USInt	The precision or size of the fractional portion. This does not include the decimal point.
FORMAT	IN	Word	Output format options
P	IN	UInt, Byte, USInt	IN: Index to the first OUT string character to be replaced (first character = 1)
OUT	OUT	String	The converted string

The value represented by parameter IN is converted to a string referenced by parameter OUT. The parameter OUT must be a valid string before the conversion is executed.

The converted string will replace characters in the OUT string starting at character offset count P to the number of characters specified by parameter SIZE. The number of characters in SIZE must fit within the OUT string length, counting from character position P. This instruction is useful for embedding number characters into a text string. For example, you can put the numbers "120" into the string "Pump pressure = 120 psi".

Parameter PREC specifies the precision or number of digits for the fractional part of the string. If the parameter IN value is an integer, then PREC specifies the location of the decimal point. For example, if the data value is 123 and PREC = 1, then the result is "12.3". The maximum supported precision for the Real data type is 7 digits.

If parameter P is greater than the current size of the OUT string, then spaces are added, up to position P, and the result is appended to the end of the string. The conversion ends if the maximum OUT string length is reached.

The FORMAT parameter for the VAL_STRG instruction is defined below. The unused bit positions must be set to zero.

Table 8- 30 Format of the VAL_STRG instruction

Bit 16							Bit 8	Bit 7							Bit 0
0	0	0	0	0	0	0	0	0	0	0	0	0	s	f	r

s = Number sign character	1= use sign character "+" and "-" 0 = use sign character "-" only
f = Notation format	1= Exponential notation 0 = Fixed point notation
r = Decimal point format	1 = "," (comma character) 0 = "." (period character)

Table 8- 31 Values of the FORMAT parameter

FORMAT (WORD)	Number sign character	Notation format	Decimal point representation
W#16#0000	"-" only	Fixed point	"."
W#16#0001			","
W#16#0002		Exponential	"."
W#16#0003			","
W#16#0004	"+" and "-"	Fixed Point	"."
W#16#0005			","
W#16#0006		Exponential	"."
W#16#0007			","
W#16#0008 to W#16#FFFF	Illegal values		

Parameter OUT string format rules:

- Leading space characters are added to the leftmost part of the string when the converted string is smaller than the specified size.
- When the FORMAT parameter sign bit is FALSE, unsigned and signed integer data type values are written to the output buffer without the leading "+" sign. The "-" sign is used if required.
  <leading spaces><digits without leading zeroes>'.'<PREC digits>
- When the sign bit is TRUE, unsigned and signed integer data type values are written to the output buffer always with a leading sign character.

  <leading spaces><sign><digits without leading zeroes>'.'<PREC digits>
- When the FORMAT is set to exponential notation, Real data type values are written to the output buffer as:

  <leading spaces><sign><digit> '.' <PREC digits>'E' <sign><digits without leading zero>
- When the FORMAT is set to fixed point notation, integer, unsigned integer, and real data type values are written to the output buffer as:

  <leading spaces><sign><digits without leading zeroes>'.'<PREC digits>
- Leading zeros to the left of the decimal point (except the digit adjacent to the decimal point) are suppressed.
- Values to the right of the decimal point are rounded to fit in the number of digits to the right of the decimal point specified by the PREC parameter.
- The size of the output string must be a minimum of three bytes more than the number of digits to the right of the decimal point.
- Values are right-justified in the output string.

Conditions reported by ENO

When an error is encountered during the conversion operation, the following results will be returned:

- ENO is set to 0.
- OUT is set to 0, or as shown in the examples for string to value conversion.
- OUT is unchanged, or as shown in the examples when OUT is a string.

Table 8- 32 ENO status

ENO	Description
1	No error
0	Illegal or invalid parameter; for example, an access to a DB that does not exist
0	Illegal string where the maximum length of the string is 0 or 255
0	Illegal string where the current length is greater than the maximum length
0	The converted number value is too large for the specified OUT data type.
0	The OUT parameter maximum string size must be large enough to accept the number of characters specified by parameter SIZE, starting at the character position parameter P.

ENO	Description
0	Illegal P value where P=0 or P is greater than the current string length
0	Parameter SIZE must be greater than parameter PREC.

Table 8- 33 Examples of S_CONV string to value conversion

IN string	OUT data type	OUT value	ENO
"123"	Int or DInt	123	TRUE
"-00456"	Int or DInt	-456	TRUE
"123.45"	Int or DInt	123	TRUE
"+2345"	Int or DInt	2345	TRUE
"00123AB"	Int or DInt	123	TRUE
"123"	Real	123.0	TRUE
"123.45"	Real	123.45	TRUE
"1.23e-4"	Real	1.23	TRUE
"1.23E-4"	Real	1.23	TRUE
"12,345.67"	Real	12345.67	TRUE
"3.4e39"	Real	3.4	TRUE
"-3.4e39"	Real	-3.4	TRUE
"1.17549e-38"	Real	1.17549	TRUE
"12345"	SInt	0	FALSE
"A123"	N/A	0	FALSE
""	N/A	0	FALSE
"++123"	N/A	0	FALSE
"+-123"	N/A	0	FALSE

Table 8- 34 Examples of S_CONV value to string conversion

Data type	IN value	OUT string	ENO
UInt	123	"123"	TRUE
UInt	0	"0"	TRUE
UDInt	12345678	"12345678"	TRUE
Real	-INF	"INF"	FALSE
Real	+INF	"INF"	FALSE
Real	NaN	"NaN"	FALSE

Table 8- 35 Examples of STRG_VAL conversion

IN string	FORMAT (W#16#....)	OUT data type	OUT value	ENO
"123"	0000	Int or DInt	123	TRUE
"-00456"	0000	Int or DInt	-456	TRUE

IN string	FORMAT (W#16#....)	OUT data type	OUT value	ENO
"123.45"	0000	Int or DInt	123	TRUE
"+2345"	0000	Int or DInt	2345	TRUE
"00123AB"	0000	Int or DInt	123	TRUE
"123"	0000	Real	123.0	TRUE
"-00456"	0001	Real	-456.0	TRUE
"+00456"	0001	Real	456.0	TRUE
"123.45"	0000	Real	123.45	TRUE
"123.45"	0001	Real	12345.0	TRUE
"123,45"	0000	Real	12345.0	TRUE
"123,45"	0001	Real	123.45	TRUE
".00123AB"	0001	Real	123.0	TRUE
"1.23e-4"	0000	Real	1.23	TRUE
"1.23E-4"	0000	Real	1.23	TRUE
"1.23E-4"	0002	Real	1.23E-4	TRUE
"12,345.67"	0000	Real	12345.67	TRUE
"12,345.67"	0001	Real	12.345	TRUE
"3.4e39"	0002	Real	+INF	TRUE
"-3.4e39"	0002	Real	-INF	TRUE
"1.1754943e-38" (and smaller)	0002	Real	0.0	TRUE
"12345"	N/A	SInt	0	FALSE
"A123"	N/A	N/A	0	FALSE
""	N/A	N/A	0	FALSE
"++123"	N/A	N/A	0	FALSE
"+-123"	N/A	N/A	0	FALSE

The following examples of VAL_STRG conversions are based on an OUT string initialized as follows:

"`Current Temp = xxxxxxxxxx C`"

where the "`x`" character represents space characters allocated for the converted value.

Table 8- 36 Examples of VAL_STRG conversion

Data type	IN value	P	SIZE	FORMAT (W#16#....)	PREC	OUT string	ENO
UInt	123	16	10	0000	0	`Current Temp = xxxxxxx123 C`	TRUE
UInt	0	16	10	0000	2	`Current Temp = xxxxxx0.00 C`	TRUE
UDInt	12345678	16	10	0000	3	`Current Temp = x12345.678 C`	TRUE
UDInt	12345678	16	10	0001	3	`Current Temp = x12345,678 C`	TRUE
Int	123	16	10	0004	0	`Current Temp = xxxxxx+123 C`	TRUE
Int	-123	16	10	0004	0	`Current Temp = xxxxxx-123 C`	TRUE
Real	-0.00123	16	10	0004	4	`Current Temp = xxx-0.0012 C`	TRUE
Real	-0.00123	16	10	0006	4	`Current Temp = -1.2300E-3 C`	TRUE
Real	-INF	16	10	N/A	4	`Current Temp = xxxxxx-INF C`	FALSE
Real	+INF	16	10	N/A	4	`Current Temp = xxxxxx+INF C`	FALSE
Real	NaN	16	10	N/A	4	`Current Temp = xxxxxxxNaN C`	FALSE
UDInt	12345678	16	6	N/A	3	`Current Temp = xxxxxxxxxx C`	FALSE

8.2.3.2 String-to-characters and characters-to-string conversions

Chars_TO_Strg copies an array of ASCII character bytes into a character string.

Strg_TO_Chars copies an ASCII character string into an array of character bytes.

Note

Only the zero based array types (Array [0..n] of Char) or (Array [0..n] of Byte) are allowed as the input parameter Chars for the Chars_TO_Strg instruction, or as the IN_OUT parameter Chars for the Strg_TO_Chars instruction.

Table 8- 37 Chars_TO_Strg instruction

LAD / FBD	SCL	Description

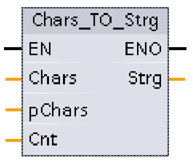
 Chars_TO_Strg
EN
ENO
Chars
Strg
pChars
Cnt
	`Chars_TO_Strg(`
	`    Chars:=_variant_in_,`
	`    pChars:=_dint_in_,`
	`    Cnt:=_uint_in_,`
	`    Strg=>_string_out_);`
		All or part of an array of characters is copied to a string.
		The output string must be declared before Chars_TO_Strg is executed. The string is then overwritten by the Chars_TO_Strg operation.
		Strings of all supported maximum lengths (1..254) may be used.
		The string maximum length value is not changed by Chars_TO_Strg operation. Copying from array to string stops when the maximum string length is reached.
		A nul character '$00' or 16#00 value in the character array works as a delimiter and ends copying of characters into the string.

Table 8- 38 Data types for the parameters (Chars_TO_Strg)

Parameter and type		Data type	Description
Chars	IN	Variant	The Chars parameter is a pointer to zero based array [0..n] of characters to be converted into a string. The array can be declared in a DB or as local variables in the block interface. Example: "DB1".MyArray points to MyArray [0..10] of Char element values in DB1.
pChars	IN	Dint	Element number for the first character in the array to copy. Array element [0] is the default value.
Cnt	IN	UInt	Count of characters to copy: 0 means all
Strg	OUT	String	Target string

Table 8- 39 Strg_TO_Chars instruction

LAD / FBD	SCL	Description

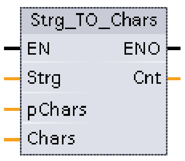
 Strg_TO_Chars
EN
ENO
Strg
Cnt
pChars
Chars
	`Strg_TO_Chars(`
	`Strg:=_string_in_,`
	`pChars:=_dint_in_,`
	`Cnt=>_uint_out_,`
	`Chars:=_variant_inout_);`
		The complete input string Strg is copied to an array of characters at IN_OUT parameter Chars.
		The operation overwrites bytes starting at array element number specified by the pChars parameter.
		Strings of all supported max lengths (1..254) may be used.
		An end delimiter is not written; this is your responsibility. To set an end delimiter just after the last written array character, use the next array element number [pChars+Cnt].

Table 8- 40 Data types for the parameters (Strg_TO_Chars)

Parameter and type		Data type	Description
Strg	IN	String	Source string
pChars	IN	DInt	Array element number for the first string character written to the target array
Chars	IN_OUT	Variant	The Chars parameter is a pointer to zero based array [0..n] of characters copied from the input string. The array can be declared in a DB or as local variables in the block interface. Example: "DB1".MyArray points to MyArray [0..10] of Char element values in DB1.
Cnt	OUT	UInt	Count of characters copied

Table 8- 41 ENO status

ENO	Description
1	No error
0	Chars_TO_Strg: Attempt to copy more character bytes to the output string than allowed by the maximum length byte in the string declaration

ENO	Description
0	Chars_TO_Strg: The nul character (16#00) value was found in the input character byte array.
0	Strg_TO_Chars: Attempt to copy more character bytes to the output array than are allowed by the element number limit

8.2.3.3 ASCII to Hex and Hex to ASCII conversions

Use the ATH (ASCII to hexadecimal) and HTA (hexadecimal to ASCII) instructions for conversions between ASCII character bytes (characters 0 to 9 and uppercase A to F only) and the corresponding 4-bit hexadecimal nibbles.

Table 8- 42 ATH instruction

LAD / FBD	SCL	Description

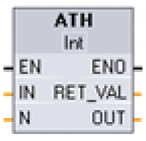
	`ret_val := ATH(`
	`    in:=_variant_in_,`
	`    n:=_int_in_,`
	`    out=>_variant_out_);`
		Converts ASCII characters into packed hexadecimal digits.

Table 8- 43 Data types for the ATH instruction

Parameter type		Data Type	Description
IN	IN	Variant	Pointer to ASCII character byte array
N	IN	UInt	Number of ASCII character bytes to convert
RET_VAL	OUT	Word	Execution condition code
OUT	OUT	Variant	Pointer to the converted hexadecimal byte array

Conversion begins at the location specified by parameter IN and continues for N bytes. The result is placed at the location specified by OUT. Only valid ASCII characters 0 to 9 and uppercase A to F can be converted. Any other character will be converted to zero.

8-bit ASCII coded characters are converted to 4-bit hexadecimal nibbles. Two ASCII characters can be stored in a single byte.

The IN and OUT parameters specify byte arrays and not hexadecimal String data. ASCII characters are converted and placed in the hexadecimal output in the same order as they are read. If there are an odd number of ASCII characters, then zeros are put in the right-most nibble of the last converted hexadecimal digit.

Table 8- 44 Examples of ASCII-to-hexadecimal (ATH) conversion

IN character bytes	N	OUT value	ENO
'0123'	4	W#16#0123	TRUE
'123AFx1a23'	10	16#123AF01023	FALSE
'a23'	3	W#16#A230	TRUE

Table 8- 45 HTA instruction

LAD / FBD	SCL	Description

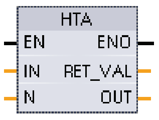
	`ret_val := HTA(`
	`    in:=_variant_in_,`
	`    n:=_uint_in_,`
	`    out=>_variant_out_);`
		Converts packed hexadecimal digits to their corresponding ASCII character bytes.

Table 8- 46 Data types for the HTA instruction

Parameter and type		Data Type	Description
IN	IN	Variant	Pointer to input byte array
N	IN	UInt	Number of bytes to convert (each input byte has two 4-bit nibbles and produces 2N ASCII characters)
RET_VAL	OUT	Word	Execution condition code
OUT	OUT	Variant	Pointer to ASCII character byte array

Conversion begins at the location specified by parameter IN and continues for N bytes. Each 4-bit nibble converts to a single 8-bit ASCII character and produces 2N ASCII character bytes of output. All 2N bytes of the output are written as ASCII characters 0 to 9 through uppercase A to F. The parameter OUT specifies a byte array and not a string.

Each nibble of the hexadecimal byte is converted into a character in the same order as they are read in (left-most nibble of a hexadecimal digit is converted first, followed by the right-most nibble of that same byte).

Table 8- 47 Examples of hexadecimal -to- ASCII (HTA) conversion

IN value	N	OUT character bytes	ENO (ENO always TRUE after HTA execution)
W#16#0123	2	'0123'	TRUE
DW#16#123AF012	4	'123AF012'	TRUE

Table 8- 48 ATH and HTA condition codes

RET_VAL (W#16#....)	Description	ENO
0000	No error	TRUE
0007	Invalid ATH input character: A character was found that was not an ASCII character 0-9, lowercase a-f, or uppercase A-F	FALSE
8101	Illegal or invalid input pointer, for example, an access to a DB that does not exist.	FALSE
8120	Input string is an invalid format, i.e., max= 0, max=255, current>max, or grant length in pointer < max	FALSE
8182	Input buffer is too small for N	FALSE
8151	Data type not allowed for input buffer	FALSE
8301	Illegal or invalid output pointer, for example, an access to a DB that does not exist.	FALSE

RET_VAL (W#16#....)	Description	ENO
8320	Output string is an invalid format, i.e., max= 0, max=255, current>max, or grant length in pointer < max	FALSE
8382	Output buffer is too small for N	FALSE
8351	Data type not allowed for output buffer	FALSE

8.2.4 String operation instructions

Your control program can use the following string and character instructions to create messages for operator display and process logs.

8.2.4.1 LEN

Table 8- 49 Length instruction

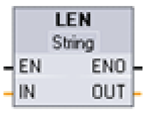
LAD / FBD	SCL	Description
	`out := LEN(in);`	LEN (length) provides the current length of the string IN at output OUT. An empty string has a length of zero.

Table 8- 50 Data types for the parameters

Parameter and type		Data type	Description
IN	IN	String	Input string
OUT	OUT	Int, DInt, Real, LReal	Number of valid characters of IN string

Table 8- 51 ENO status

ENO	Condition	OUT
1	No invalid string condition	Valid string length
0	Current length of IN exceeds maximum length of IN	Current length is set to 0
	Maximum length of IN does not fit within allocated memory range	
	Maximum length of IN is 255 (illegal length)	

8.2.4.2 CONCAT

Table 8- 52 Concatenate strings instruction

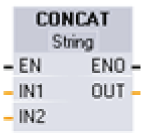
LAD / FBD	SCL	Description
	`out := CONCAT(in1, in2);`	CONCAT (concatenate strings) joins string parameters IN1 and IN2 to form one string provided at OUT. After concatenation, String IN1 is the left part and String IN2 is the right part of the combined string.

Table 8- 53 Data types for the parameters

Parameter and type		Data type	Description
IN1	IN	String	Input string 1
IN2	IN	String	Input string 2
OUT	OUT	String	Combined string (string 1 + string 2)

Table 8- 54 ENO status

ENO	Condition	OUT
1	No errors detected	Valid characters
0	Resulting string after concatenation is larger than maximum length of OUT string	Resulting string characters are copied until the maximum length of the OUT is reached
	Current length of IN1 exceeds maximum length of IN1, current length of IN2 exceeds maximum length of IN2, or current length of OUT exceeds maximum length of OUT (invalid string)	Current length is set to 0
	Maximum length of IN1, IN2 or OUT does not fit within allocated memory range	
	Maximum length of IN1 or IN2 is 255, or the maximum length of OUT is 0 or 255	

8.2.4.3 LEFT, RIGHT, and MID

Table 8- 55 Left, right and middle substring operations

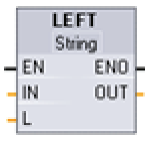
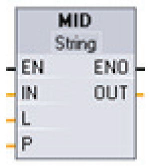
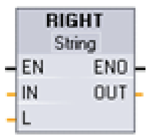
LAD / FBD	SCL	Description
	`out := LEFT(in, L);`	LEFT (Left substring) provides a substring made of the first L characters of string parameter IN. • If L is greater than the current length of the IN string, then the entire IN string is returned in OUT. • If an empty string is the input, then an empty string is returned in OUT.
	`out := MID(in, L, p);`	MID (Middle substring) provides the middle part of a string. The middle substring is L characters long and starts at character position P (inclusive). If the sum of L and P exceeds the current length of the string parameter IN, then a substring is returned that starts at character position P and continues to the end of the IN string.
	`out := RIGHT(in, L);`	RIGHT (Right substring) provides the last L characters of a string. • If L is greater than the current length of the IN string, then the entire IN string is returned in parameter OUT. • If an empty string is the input, then an empty string is returned in OUT.

Table 8- 56 Data types for the parameters

Parameter and type		Data type	Description
IN	IN	String	Input string
L	IN	Int	Length of the substring to be created: • LEFT uses the left-most characters number of characters in the string • RIGHT uses the right-most number of characters in the string • MID uses the number of characters starting at position P within the string
P	IN	Int	MID only: Position of first substring character to be copied P= 1, for the initial character position of the IN string
OUT	OUT	String	Output string

Table 8- 57 ENO status

ENO	Condition	OUT
1	No errors detected	Valid characters
0	• L or P is less than or equal to 0 • P is greater than maximum length of IN • Current length of IN exceeds maximum length of IN, or current length of OUT exceeds maximum length of OUT • Maximum length of IN or OUT does not fit within allocated memory • Maximum length of IN or OUT is 0 or 255	Current length is set to 0
	Substring length (L) to be copied is larger than maximum length of OUT string.	Characters are copied until the maximum length of OUT is reached
	MID only: L or P is less than or equal to 0	Current length is set to 0
	MID only: P is greater than maximum length of IN	
	Current length of IN1 exceeds maximum length of IN1, or current length of IN2 exceeds maximum length of IN2 (invalid string)	Current length is set to 0
	Maximum length of IN1, IN2 or OUT does not fit within allocated memory range	
	Maximum length of IN1, IN2 or OUT is 0 or 255 (illegal length)	

8.2.4.4 DELETE

Table 8- 58 Delete substring instruction

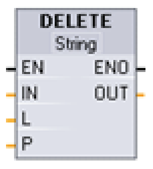
LAD / FBD	SCL	Description
	`out := DELETE(in, L, p);`	Deletes L characters from string IN. Character deletion starts at character position P (inclusive), and the remaining substring is provided at parameter OUT. • If L is equal to zero, then the input string is returned in OUT. • If the sum of L and P is greater than the length of the input string, then the string is deleted to the end.

Table 8- 59 Data types for the parameters

Parameter and type		Data type	Description
IN	IN	String	Input string
L	IN	Int	Number of characters to be deleted
P	IN	Int	Position of the first character to be deleted: The first character of the IN string is position number 1
OUT	OUT	String	Output string

Table 8- 60 ENO status

ENO	Condition	OUT
1	No errors detected	Valid characters
0	P is greater than current length of IN	IN is copied to OUT with no characters deleted
	Resulting string after characters are deleted is larger than maximum length of OUT string	Resulting string characters are copied until the maximum length of OUT is reached
	L is less than 0, or P is less than or equal to 0	Current length is set to 0
	Current length of IN exceeds maximum length of IN, or current length of OUT exceeds maximum length of OUT	
	Maximum length of IN or OUT does not fit within allocated memory	
	Maximum length of IN or OUT is 0 or 255	

8.2.4.5 INSERT

Table 8- 61 Insert substring instruction

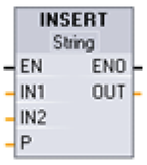
LAD / FBD	SCL	Description
	`out := INSERT(in1, in2, p);`	Inserts string IN2 into string IN1. Insertion begins after the character at position P.

Table 8- 62 Data types for the parameters

Parameter and type		Data type	Description
IN1	IN	String	Input string 1
IN2	IN	String	Input string 2
P	IN	Int	Last character position in string IN1 before the insertion point for string IN2 The first character of string IN1 is position number 1.
OUT	OUT	String	Result string

Table 8- 63 ENO status

ENO	Condition	OUT
1	No errors detected	Valid characters
0	P is greater than length of IN1	IN2 is concatenated with IN1 immediately following the last IN1 character
	P is less than 0	Current length is set to 0

ENO	Condition	OUT
	Resulting string after insertion is larger than maximum length of OUT string	Resulting string characters are copied until the maximum length of OUT is reached
	Current length of IN1 exceeds maximum length of IN1, current length of IN2 exceeds maximum length of IN2, or current length of OUT exceeds maximum length of OUT (invalid string)	Current length is set to 0
	Maximum length of IN1, IN2 or OUT does not fit within allocated memory range	
	Maximum length of IN1 or IN2 is 255, or maximum length of OUT is 0 or 255	

8.2.4.6 REPLACE

Table 8- 64 Replace substring instruction

LAD / FBD	SCL	Description

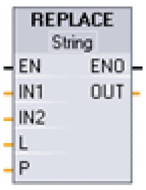
 REPLACE
String
EN
ENO
IN1
OUT
IN2
L
P
	`out := REPLACE(`
	`     in1:=_string_in_,`
	`     in2:=_string_in_,`
	`     L:=_int_in_,`
	`     p:=_int_in);`
		Replaces L characters in the string parameter IN1. Replacement starts at string IN1 character position P (inclusive), with replacement characters coming from the string parameter IN2.

Table 8- 65 Data types for the parameters

Parameter and type		Data type	Description
IN1	IN	String	Input string
IN2	IN	String	String of replacement characters
L	IN	Int	Number of characters to replace
P	IN	Int	Position of first character to be replaced
OUT	OUT	String	Result string

If parameter L is equal to zero, then the string IN2 is inserted at position P of string IN1 without deleting any characters from string IN1.

If P is equal to one, then the first L characters of string IN1 are replaced with string IN2 characters.

Table 8- 66 ENO status

ENO	Condition	OUT
1	No errors detected	Valid characters
0	P is greater than length of IN1	IN2 is concatenated with IN1 immediately following the last IN1 character
	P points within IN1, but fewer than L characters remain in IN1	IN2 replaces the end characters of IN1 beginning at position P
	Resulting string after replacement is larger than maximum length of OUT string	Resulting string characters are copied until the maximum length of OUT is reached
	Maximum length of IN1 is 0	IN2 characters are copied to OUT
	L is less than 0, or P is less than or equal to 0	Current length is set to 0
	Current length of IN1 exceeds maximum length of IN1, current length of IN2 exceeds maximum length of IN2, or current length of OUT exceeds maximum length of OUT	
	Maximum length of IN1, IN2 or OUT does not fit within allocated memory range	
	Maximum length of IN1 or IN2 is 255, or maximum length of OUT is 0 or 255	

8.2.4.7 FIND

Table 8- 67 Find substring instruction

LAD / FBD	SCL	Description

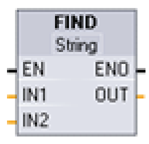
	`out := FIND(`
	`     in1:=_string_in_,`
	`     in2:=_string_in);`
		Provides the character position of the substring specified by IN2 within the string IN1. The search starts on the left. The character position of the first occurrence of IN2 string is returned at OUT. If the string IN2 is not found in the string IN1, then zero is returned.

Table 8- 68 Data types for the parameters

Parameter and type		Data type	Description
IN1	IN	String	Search inside this string
IN2	IN	String	Search for this string
OUT	OUT	Int	Character position in string IN1 of the first search match

Table 8- 69 ENO status

ENO	Condition	OUT
1	No errors detected	Valid character position
0	IN2 is larger than IN1	Character position is set to 0
	Current length of IN1 exceeds maximum length of IN1, or current length of IN2 exceeds maximum length of IN2 (invalid string)	
	Maximum length of IN1 or IN2 does not fit within allocated memory range	
	Maximum length of IN1 or IN2 is 255	

8.3 Distributed I/O (PROFINET, PROFIBUS, or AS-i)

8.3.1 Distributed I/O Instructions

The following Distributed I/O instructions can be used with PROFINET, PROFIBUS, or AS-i:

- RDREC instruction (Page 275): You can read a data record with the number INDEX from a module or device.
- WRREC instruction (Page 275): You can transfer a data record with the number INDEX to a module or device defined by ID.
- RALRM instruction (Page 278): You can receive an interrupt with all corresponding information from a module or device and supply this information to its output parameters.
- DPRD_DAT instruction (Page 284): You must read consistent data areas greater than 64 bytes from a module or device with the DPRD_DAT instruction.
- DPWR_DAT instruction (Page 284): You must write consistent data areas greater than 64 bytes to a module or device with the DPWR_DAT instruction.

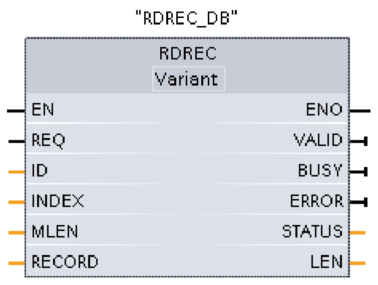
The DPNRM_DG instruction (Page 286) can only be used with PROFIBUS. You can read the current diagnostic data of a DP slave in the format specified by EN 50 170 Volume 2, PROFIBUS.

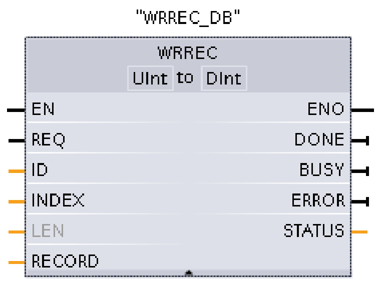
8.3.2 RDREC and WRREC

You can use the RDREC (Read record) and WRREC (Write record) instructions with PROFINET, PROFIBUS, and AS-i.

Table 8- 70 RDREC and WRREC instructions

LAD / FBD	SCL	Description

 "RDREC_DB"
RDREC
Variant
EN
ENO
REQ
VALID
ID
BUSY
INDEX
ERROR
MLEN
STATUS
RECORD
LEN
	`"RDREC_DB"(`
	`    req:=_bool_in_,`
	`    ID:=_word_in_,`
	`    index:=_dint_in_,`
	`    mlen:=_uint_in_,`
	`    valid=>_bool_out_,`
	`    busy=>_bool_out_,`
	`    error=>_bool_out_,`
	`    status=>_dword_out_,`
	`    len=>_uint_out_,`
	`    record:=_variant_inout_);`
		Use the RDREC instruction to read a data record with the number INDEX from the component addressed by the ID, such as a central rack or a distributed component (PROFIBUS DP or PROFINET IO). Assign the maximum number of bytes to read in MLEN. The selected length of the target area RECORD should have at least the length of MLEN bytes.

 "WRREC_DB"
WRREC
UInt to DInt
EN
ENO
REQ
DONE
ID
BUSY
INDEX
ERROR
LEN
STATUS
RECORD
	`"WRREC_DB"(`
	`    req:=_bool_in_,`
	`    ID:=_word_in_,`
	`    index:=_dint_in_,`
	`    len:=_uint_in_,`
	`    done=>_bool_out_,`
	`    busy=>_bool_out_,`
	`    error=>_bool_out_,`
	`    status=>_dword_out_,`
	`    record:=_variant_inout_);`
		Use the WRREC instruction to transfer a data RECORD with the record number INDEX to a DP slave/PROFINET IO device component addressed by ID, such as a module in the central rack or a distributed component (PROFIBUS DP or PROFINET IO).
		Assign the byte length of the data record to be transmitted. The selected length of the source area RECORD should, therefore, have at least the length of LEN bytes.

1 STEP 7 automatically creates the DB when you insert the instruction.

2 In the SCL examples, "RDREC_DB" and "WRREC_DB" are the names of the instance DBs.

Table 8- 71 RDREC and WRREC data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	REQ = 1: Transfer data record
ID	IN	HW_IO (Word)	Logical address of the DP slave/PROFINET IO component (module or submodule): • For an output module, bit 15 must be set (for example, for address 5: ID:= DW#16#8005). • For a combination module, the smaller of the two addresses should be specified. **Note**: The device ID can be determined in one of two ways: • By making the following "Network view" selections: – Device (gray box) – "Properties" of the device – "Hardware identifier" **Note**: Not all devices display their Hardware identifiers, however. • By making the following "Project tree" menu selections: – PLC tags – Default tag table – System constants tab All configured device Hardware identifiers are displayed.
INDEX	IN	Byte, Word, USInt, UInt, SInt, Int, DInt	Data record number
MLEN	IN	Byte, USInt, UInt	Maximum length in bytes of the data record information to be fetched (RDREC)
VALID	OUT	Bool	New data record was received and valid (RDREC). The VALID bit is TRUE for one scan, after the last request was completed with no error.
DONE	OUT	Bool	Data record was transferred (WRREC). The DONE bit is TRUE for one scan, after the last request was completed with no error.
BUSY	OUT	Bool	• BUSY = 1: The read (RDREC) or write (WRREC) process is not yet terminated. • BUSY = 0: Data record transmission is completed.
ERROR	OUT	Bool	ERROR = 1: A read (RDREC) or write (WRREC) error has occurred. The ERROR bit is TRUE for one scan, after the last request was terminated with an error. The error code value at the STATUS parameter is valid only during the single scan where ERROR = TRUE.
STATUS	OUT	DWord	Block status or error information
LEN	OUT (RDREC) IN (WRREC)	UInt	• Length of the fetched data record information (RDREC) • Maximum byte length of the data record to be transferred (WRREC)
RECORD	IN_OUT	Variant	• Target area for the fetched data record (RDREC) • Data record (WRREC)

The RDREC and WRREC instructions operate asynchronously, that is, processing covers multiple instruction calls. Start the job by calling RDREC or WRREC with REQ = 1.

The job status is displayed via output parameter BUSY and the two central bytes of output parameter STATUS. The transfer of the data record is complete when the output parameter BUSY has the value FALSE

TRUE (only for one scan) on the output parameter VALID (RDREC) or DONE (WRREC) verifies that the data record has been successfully transferred into the target area RECORD (RDREC) or to the target device (WRREC). In the case of the RDREC, the output parameter LEN contains the length of the fetched data in bytes.

The output parameter ERROR (only for one scan when ERROR = TRUE) indicates that a data record transmission error has occurred. In this case, the output parameter STATUS (only for the one scan when ERROR = TRUE) contains the error information.

Data records are defined by the hardware device manufacturer. Refer to the hardware manufacturer's device documentation for details about a data record.

Note

If a DPV1 slave is configured via GSD file (GSD rev. 3 and higher) and the DP interface of the DP master is set to "S7 compatible", then you may not read any data records from the I/O modules in the user program with "RDREC" or write to the I/O modules with "WRREC". In this case, the DP master addresses the wrong slot (configured slot + 3).

Remedy: set the interface of the DP master to "DPV1".

Note

The interfaces of the "RDREC" and "WRREC" instructions are identical to the "RDREC" and "WRREC" FBs defined in "PROFIBUS Guideline PROFIBUS Communication and Proxy Function Blocks according to IEC 61131-3".

Note

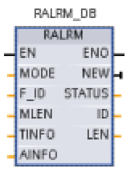
If you use "RDREC" or "WRREC" to read or write a data record for PROFINET IO, then negative values in the INDEX, MLEN, and LEN parameters will be interpreted as an unsigned 16-bit integer.

8.3.3 RALRM

You can use the RALRM (Read alarm) instruction with PROFINET and PROFIBUS.

Table 8- 72 RALRM instruction

LAD / FBD	SCL	Description

 RALRM_DB
RALRM
EN
ENO
MODE
NEW
F_ID
STATUS
MLEN
ID
TINFO
LEN
AINFO
	`"RALRM_DB"(`
	`    mode:=_int_in_,`
	`    f_ID:=_word_in_,`
	`    mlen:=_uint_in_,`
	`    new=>_bool_out_,`
	`    status=>_dword_out_,`
	`    ID=>_word_out_,`
	`    len=>_uint_out_,`
	`    tinfo:=_variant_inout_,`
	`    ainfo:=_variant_inout_);`
		Use the RALRM (read alarm) instruction to read diagnostic interrupt information from PROFIBUS or PROFINET I/O modules/devices.
		The information in the output parameters contains the start information of the called OB as well as information of the interrupt source.
		Call RALRM in an interrupt OB to return information regarding the event(s) that caused the interrupt. In the S7-1200, only diagnostic interrupts (OB82) are supported.

1 STEP 7 automatically creates the DB when you insert the instruction.

2 In the SCL example, "RALRM_DB" is the name of the instance DB.

Table 8- 73 Data types for the parameters

Parameter and type		Data type	Description
MODE	IN	Byte, USInt, SInt, Int	Operating mode
F_ID	IN	HW_IO (Word)	Logical start address of the component (module) from which interrupts are to be received **Note**: The device ID can be determined in one of two ways: • By making the following "Network view" selections: – Device (gray box) – "Properties" of the device – "Hardware identifier" **Note**: Not all devices display their Hardware identifiers. • By making the following "Project tree" menu selections: – PLC tags – Default tag table – System constants tab – All configured device Hardware identifiers are displayed.
MLEN	IN	Byte, USInt, UInt	Maximum length in bytes of the data interrupt information to be received. MLEN of 0 will allow receipt of as much data interrupt information as is available in the AINFO Target Area.
NEW	OUT	Bool	A new interrupt was received.
STATUS	OUT	DWord	Status of the RALRM instruction. Refer to "STATUS parameter for RDREC, WRREC, and RALRM" (Page 280) for more information.
ID	OUT	HW_IO (Word)	Hardware identifier of the I/O module that caused the diagnostic interrupt **Note**: Refer to the F_ID parameter for an explanation of how to determine the device ID.

Parameter and type		Data type	Description
LEN	OUT	DWord, UInt, UDInt, DInt, Real, LReal	Length of the received AINFO interrupt information
TINFO	IN_OUT	Variant	Task information: Target range for OB start and management information. The TINFO length is always 32 bytes.
AINFO	IN_OUT	Variant	Interrupt information: Target area for header information and additional interrupt information. For AINFO, provide a length of at least the MLEN bytes, if MLEN is greater than 0. The AINFO length is variable.

Note

If you call "RALRM" in an OB whose start event is not an I/O interrupt, the instruction will provide correspondingly reduced information in its outputs.

Make sure to use different instance DBs when you call "RALRM" in different OBs. If you evaluate data resulting from a "RALRM" call outside of the associated interrupt OB, you should use a separate instance DB per OB start event.

Note

The interface of the "RALRM" instruction is identical to the "RALRM" FB defined in "PROFIBUS Guideline PROFIBUS Communication and Proxy Function Blocks according to IEC 61131-3".

Calling RALRM

You can call the RALRM instruction in three different operating modes (MODE).

Table 8- 74 RALRM instruction operating modes

MODE	Description
0	• ID contains the hardware identifier of the I/O module that triggered the interrupt. • Output parameter NEW is set to TRUE. • LEN produces an output of 0. • AINFO and TINFO are not updated with any information.
1	• ID contains the hardware identifier of the I/O module that triggered the interrupt. • Output parameter NEW is set to TRUE. • LEN produces an output of the amount in bytes of AINFO data that is returned. • AINFO and TINFO are updated with interrupt-related information.
2	If the hardware identifier assigned to input parameter F_ID has triggered the interrupt then: • ID contains the hardware identifier of the I/O module that triggered the interrupt. Should be the same as the value at F_ID. • Output parameter NEW is set to TRUE. • LEN produces an output of the amount in bytes of AINFO data that is returned. • AINFO and TINFO are updated with interrupt-related information.

Note

If you assign a destination area for TINFO or AINFO that is too short, RALRM cannot return the full information.

MLEN can limit the amount of AINFO data that is returned.

Refer to the AINFO parameters and TINFO parameters of the online information system of STEP 7 for information on how to interpret the TINFO and AINFO data.

8.3.4 STATUS parameter for RDREC, WRREC, and RALRM

The output parameter STATUS contains error information that is interpreted as ARRAY[1...4] OF BYTE, with the following structure:

Table 8- 75 STATUS output array

Array element	Name	Description
STATUS[1]	Function_Num	• B#16#00, if no error • Function ID from DPV1-PDU: If an error occurs, B#16#80 is OR'ed (for read data record: B#16#DE; for write data record: B#16#DF). If no DPV1 protocol element is used, then B#16#C0 will be output.
STATUS[2]	Error_Decode	Location of the error ID
STATUS[3]	Error_Code_1	Error ID
STATUS[4]	Error_Code_2	Manufacturer-specific error ID expansion

Table 8- 76 STATUS[2] values

Error_decode (B#16#....)	Source	Description
00 to 7F	CPU	No error or no warning
80	DPV1	Error according to IEC 61158-6
81 to 8F	CPU	B#16#8x shows an error in the "xth" call parameter of the instruction.
FE, FF	DP Profile	Profile-specific error

Table 8- 77 STATUS[3] values

Error_decode (B#16#....)	Error_code_1 (B#16#....)	Explanation (DVP1)	Description
00	00		No error, no warning
70	00	Reserved, reject	Initial call; no active data record transfer
	01	Reserved, reject	Initial call; data record transfer has started

Error_decode (B#16#....)	Error_code_1 (B#16#....)	Explanation (DVP1)	Description
	02	Reserved, reject	Intermediate call; data record transfer already active
80	90	Reserved, pass	Invalid logical start address
	92	Reserved, pass	Illegal type for Variant pointer
	93	Reserved, pass	The DP component addressed via ID or F_ID is not configured.
	96		The "RALRM (Page 278)" cannot supply the OB start information, management information, header information, or additional interrupt information. For OBs 4x, 55, 56, 57, 82, and 83, you can use the "DPNRM_DG (Page 286)" instruction to read the current diagnostics message frame of the relevant DP slave asynchronously (address information from OB start information).
	A0	Read error	Negative acknowledgement while reading from the module
	A1	Write error	Negative acknowledgement while writing to the module
	A2	Module failure	DP protocol error at layer 2 (for example, slave failure or bus problems)
	A3	Reserved, pass	• PROFIBUS DP: DP protocol error with Direct-Data-Link-Mapper or User-Interface/User • PROFINET IO: General CM error
	A4	Reserved, pass	Communication on the communication bus disrupted
	A5	Reserved, pass	-
	A7	Reserved, pass	DP slave or modules is occupied (temporary error).
	A8	Version conflict	DP slave or module reports non-compatible versions.
	A9	Feature not supported	Feature not supported by DP slave or module
	AA to AF	User specific	DP slave or module reports a manufacturer-specific error in its application. Please check the documentation from the manufacturer of the DP slave or module.
	B0	Invalid index	Data record not known in module; illegal data record number ≥ 256
	B1	Write length error	The length information in the RECORD parameter is incorrect. • With "RALRM": Length error in AINFO **Note**: Refer to the online information system of STEP 7 for immediate access to information on how to interpret the "AINFO" returned buffers. • With "RDREC (Page 275)" and "WRREC (Page 275)": Length error in "MLEN"
	B2	Invalid slot	The configured slot is not occupied.
	B3	Type conflict	Actual module type does not match specified module type.
	B4	Invalid area	DP slave or module reports access to an invalid area.
	B5	Status conflict	DP slave or module not ready
	B6	Access denied	DP slave or module denies access.

Error_decode (B#16#....)	Error_code_1 (B#16#....)	Explanation (DVP1)	Description
	B7	Invalid range	DP slave or module reports an invalid range for a parameter or value.
	B8	Invalid parameter	DP slave or module reports an invalid parameter.
	B9	Invalid type	DP slave or module reports an invalid type: • With "RDREC (Page 275)": Buffer too small (subsets cannot be read) • With "WRREC (Page 275)": Buffer too small (subsets cannot be written)
	BA to BF	User specific	DP slave or module reports a manufacturer-specific error when accessing. Please check the documentation from the manufacturer of the DP slave or module.
	C0	Read constraint conflict	• With "WRREC (Page 275)": The data can only be written when the CPU is in STOP mode. **Note**: This means that data cannot be written by the user program. You can only write the data online with a PG/PC. • With "RDREC (Page 275)": The module routes the data record, but either no data is present or the data can only be read when the CPU is in STOP mode. **Note**: If data can only be read when the CPU is in STOP mode, no evaluation by the user program is possible. In this case, you can only read the data online with a PG/PC.
	C1	Write constraint conflict	The data of the previous write request to the module for the same data record has not yet been processed by the module.
	C2	Resource busy	The module is currently processing the maximum possible number of jobs for a CPU.
	C3	Resource unavailable	The required operating resources are currently occupied.
	C4		Internal temporary error. Job could not be carried out. Repeat the job. If this error occurs often, check your installation for sources of electrical interference.
	C5		DP slave or module not available
	C6		Data record transfer was cancelled due to priority class cancellation.
	C7		Job aborted due to warm or cold restart on the DP master.
	C8 to CF		DP slave or module reports a manufacturer-specific resource error. Please check the documentation from the manufacturer of the DP slave or module.
	Dx	User specific	DP Slave specific. Refer to the description of the DP Slave.
81	00 to FF		Error in the initial call parameter (with "RALRM (Page 278)": MODE)
	00		Illegal operating mode
82	00 to FF		Error in the second call parameter

Error_decode (B#16#....)	Error_code_1 (B#16#....)	Explanation (DVP1)	Description
88	00 to FF		Error in the eighth call parameter (with "RALRM (Page 278)": TINFO) **Note**: Refer to the online information system of STEP 7 for immediate access to information on how to interpret the "TINFO" returned buffers.
	01		Wrong syntax ID
	23		Quantity structure exceeded or destination area too small
	24		Wrong range ID
	32		DB/DI number out of user range
	3A		DB/DI number is NULL for area ID DB/DI, or specified DB/DI does not exist.
89	00 to FF		Error in the ninth call parameter (with "RALRM (Page 278)": AINFO) **Note**: Refer to the online information system of STEP 7 for immediate access to information on how to interpret the "AINFO" returned buffers.
	01		Wrong syntax ID
	23		Quantity structure exceeded or destination area too small
	24		Wrong range ID
	32		DB/DI number out of user range
	3A		DB/DI number is NULL for area ID DB/DI, or specified DB/DI does not exist.
8A	00 to FF		Error in the 10th call parameter
8F	00 to FF		Error in the 15th call parameter
FE, FF	00 to FF		Profile-specific error

Array element STATUS[4]

With DPV1 errors, the DP Master passes on STATUS[4] to the CPU and to the instruction. Without a DPV1 error, this value is set to 0, with the following exceptions for the RDREC:

- STATUS[4] contains the target area length from RECORD, if MLEN > the destination area length from RECORD.
- STATUS[4]=MLEN, if the actual data record length < MLEN < the destination area length from RECORD.
- STATUS[4]=0, if STATUS[4] > 255; would have to be set

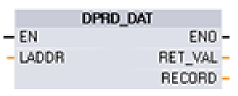
In PROFINET IO, STATUS[4] has the value 0.

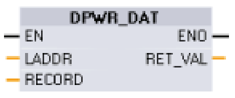
8.3.5 DPRD_DAT and DPWR_DAT

You can use the DPRD_DAT (Read consistent data) and DPWR_DAT (Write consistent data) instructions with PROFINET and PROFIBUS.

Table 8- 78 DPRD_DAT and DPWR_DAT instructions

LAD / FBD	SCL	Description

	`ret_val := DPRD_DAT(`
	`laddr:=_word_in_,`
	`record=>_variant_out_);`
		Use the DPRD_DAT instruction to read the consistent data of a DP standard slave/PROFINET IO device. If no errors occur during the data transfer, the data read is entered into the target area set up by the RECORD parameter. The target area must have the same length as you configured with STEP 7 for the selected module. When you call the DPRD_DAT instruction, you can only access the data of one module / DP identification under the configured start address.

	`ret_val := DPWR_DAT(`
	`laddr:=_word_in_,`
	`record:=_variant_in_);`
		Use the DPWR_DAT instruction to transfer the data in RECORD consistently to the addressed DP standard slave/PROFINET IO device. The source area must have the same length as you configured with STEP 7 for the selected module.

The CPU supports up to 64 bytes of consistent data. For consistent data areas greater than 64 bytes, the DPRD_DAT and DPWR_DAT instructions must be used. If required, these instructions can be used for data areas of 1 byte or greater. If access is rejected, error code W#16#8090 will result.

Note

If you are using the DPRD_DAT and DPWR_DAT instructions with consistent data, you must remove this consistent data from the process-image automatic update. Refer to "PLC concepts: Execution of the user program" (Page 67) for more information.

Table 8- 79 Data types for the parameters

Parameter and type		Data type	Description
LADDR	IN	HW_IO (Word)	• Configured start address from the "I" area of the module from which the data will be read (DPRD_DAT) • Configured start address from the process image output area of the module to which the data will be written (DPWR_DAT) Addresses have to be entered in hexadecimal format (for example, an input or output address of 100 means: LADDR:=W#16#64).
RECORD	OUT	Variant	Destination area for the user data that were read (DPRD_DAT) or source area for the user data to be written (DPWR_DAT). This must be exactly as large as you configured for the selected module with STEP 7. Only the data type Byte is permitted.
RET_VAL	OUT	Int	If an error occurs while the function is active, the return value contains an error code.

DPRD_DAT operations

The destination area must have the same length as configured for the selected module with STEP 7. If no error occurs during the data transfer, the data that have been read are entered into the destination area identified by RECORD.

If you read from a DP standard slave with a modular design or with several DP identifiers, you can only access the data of one module/DP identifier for each DPRD_DAT instruction call, specifying the configured start address.

DPWR_DAT operations

You transfer the data in RECORD consistently to the addressed DP standard slave/PROFINET IO. The data is transferred synchronously, that is, the write process is completed when the instruction is completed.

The source area must have the same length as you configured for the selected module with STEP 7.

If the DP standard slave has a modular design, you can only access one module of the DP slave.

Table 8- 80 DPRD_DAT and DPWR_DAT error codes

Error code	Description
0000	No error occurred
808x	System error with external DP interface module
8090	One of the following cases apply: • You have not configured a module for the specified logical base address. • You have ignored the restriction concerning the length of consistent data. • You have not entered the start address in the LADDR parameter in hexadecimal format.
8092	A type other than Byte is specified in the Any reference.
8093	No DP module/PROFINET IO device from which you can read (DPRD_DAT) or to which you can write (DPWR_DAT) consistent data exists at the logical address specified in LADDR.
80A0	Access error detected while the I/O devices were being accessed (DPRD_DAT).
80A1	Access error detected while the I/O devices were being accessed (DPWR_DAT).
80B0	Slave failure on external DP interface module
80B1	The length of the specified destination (DPRD_DAT) or source (DPWR_DAT) area is not identical to the user data length configured with STEP 7 Basic.
80B2, 80B3, 80C2, 80Fx	System error with external DP interface module (DPRD_DAT) and (DPWR_DAT)
87xy, 808x	System error with external DP interface module (DPRD_DAT)
85xy	System error with external DP interface module (DPWR_DAT)
80C0	The data have not yet been read by the module (DPRD_DAT).
80C1	The data of the previous write job on the module have not yet been processed by the module (DPWR_DAT).
8xyy[1]	General error information

Refer to "Extended instructions, Distributed I/O: Error information for RDREC, WRREC, and RALRM" (Page 280) for more information on general error codes.

Note

If you access DPV1 slaves, error information from these slaves can be forwarded from the DP master to the instruction.

8.3.6 DPNRM_DG

You can use the DPNRM_DG (Read diagnostic data) instruction with PROFIBUS.

Table 8- 81 DPNRM_DG instruction

LAD / FBD	SCL	Description

DPNRM_DG
EN
ENO
REQ
RECORD
LADDR
BUSY
RET_VAL
	`ret_val := DPNRM_DG(`
	`    req:=_bool_in_,`
	`    laddr:=_word_in_,`
	`    record=>_variant_out_,`
	`    busy=>_bool_out_);`
		Use the DPNRM_DG instruction to read the current diagnostic data of a DP slave in the format specified by EN 50 170 Volume 2, PROFIBUS. The data that has been read is entered in the destination area indicated by RECORD following error-free data transfer.

Table 8- 82 DPNRM_DG instruction data types for the parameters

Parameter and type		Data type	Description
REQ	IN	Bool	REQ=1: Read request
LADDR	IN	HW_DPSLAVE	Configured diagnostic address of the DP slave: Must be the address of the station and not for the I/O device. Select the station (and not the image of the device) in the "Network" view of the "Device configuration" to determine the diagnostic address. Enter the addresses in hexadecimal format. For example, diagnostic address 1022 means LADDR:=W#16#3FE.
RET_VAL	OUT	Int	If an error occurs while the function is active, the return value contains an error code. If no error occurs, the length of the data actually transferred is entered in RET_VAL.
RECORD	OUT	Variant	Destination area for the diagnostic data that were read. Only the Byte data type is permitted. The minimum length of the data record to be read or the destination area is 6. The maximum length of the data record to be sent is 240. Standard slaves can provide more than 240 bytes of diagnostic data up to a maximum of 244 bytes. In this case, the first 240 bytes are transferred to the destination area, and the overflow bit is set in the data.
BUSY	OUT	Bool	BUSY=1: The read job is not yet completed

You start the read job by assigning 1 to the input parameter REQ in the DPNRM_DG instruction call. The read job is executed asynchronously, in other words, it requires several DPNRM_DG instruction calls. The status of the job is indicated by the output parameters RET_VAL and BUSY.

Table 8- 83 Slave diagnostic data structure

Byte	Description
0	Station status 1
1	Station status 2
2	Station status 3
3	Master station number
4	Vendor ID (high byte)
5	Vendor ID (low byte)
6 ...	Additional slave-specific diagnostic information

Table 8- 84 DPNRM_DG instruction error codes

Error code	Description	Restriction
0000	No error	-
7000	First call with REQ=0: No data transfer active; BUSY has the value 0.	-
7001	First call with REQ =1: No data transfer active; BUSY has the value 1.	Distributed I/Os
7002	Interim call (REQ irrelevant): Data transfer already active; BUSY has the value 1.	Distributed I/Os
8090	Specified logical base address invalid: There is no base address.	-
8092	The type specified in the Any reference is not Byte.	-
8093	• This instruction is not permitted for the module specified by LADDR (S7-DP modules for S7-1200 are permitted). • LADDR specifies the I/O device instead of specifying the station. Select the station (and not the image of the device) in the "Network" view of the "Device configuration" to determine the diagnostic address for LADDR.	-
80A2	• DP protocol error at layer 2 (for example, slave failure or bus problems) • For ET200S, data record cannot be read in DPV0 mode.	Distributed I/Os
80A3	DP protocol error with user interface/user	Distributed I/Os
80A4	Communication problem on the communication bus	The error occurs between the CPU and the external DP interface module.
80B0	• The instruction is not possible for module type. • The module does not recognize the data record. • Data record number 241 is not permitted.	-
80B1	The length specified in the RECORD parameter is incorrect.	Specified length > record length
80B2	The configured slot is not occupied.	-
80B3	Actual module type does not match the required module type.	-
80C0	There is no diagnostic information.	-
80C1	The data of the previous write job for the same data record on the module have not yet been processed by the module.	-

Error code	Description	Restriction
80C2	The module is currently processing the maximum possible number of jobs for a CPU.	-
80C3	The required resources (memory, etc.) are currently occupied.	-
80C4	Internal temporary error. The job could not be processed. Repeat the job. If this error occurs frequently, check your system for electrical disturbance sources.	-
80C5	Distributed I/Os not available	Distributed I/Os
80C6	Data record transfer was stopped due to a priority class abort (restart or background)	Distributed I/Os
8xyy[1]	General error codes	

Refer to "Extended instructions, Distributed I/O: Error information for RDREC, WRREC, and RALRM" (Page 280) for more information on general error codes.

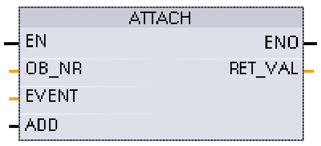
8.4 Interrupts

8.4.1 Attach and detach instructions

You can activate and deactivate interrupt event-driven subprograms with the ATTACH and DETACH instructions.

Table 8- 85 ATTACH and DETACH instructions

LAD / FBD	SCL	Description

	`ret_val := ATTACH(`
	`ob_nr:=_int_in_,`
	`event:=_event_att_in_,`
	`add:=_bool_in_);`
		ATTACH enables interrupt OB subprogram execution for a hardware interrupt event.

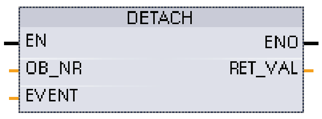
	`ret_val := DETACH(`
	`ob_nr:=_int_in_,`
	`event:=_event_att_ in);`
		DETACH disables interrupt OB subprogram execution for a hardware interrupt event.

Table 8- 86 Data types for the parameters

Parameter and type		Data type	Description
OB_NR	IN	OB_ATT	Organization block identifier: Select from the available hardware interrupt OBs that were created using the "Add new block" feature. Double-click on the parameter field, then click on the helper icon to see the available OBs.
EVENT	IN	EVENT_ATT	Event identifier: Select from the available hardware interrupt events that were enabled in PLC device configuration for digital inputs or high-speed counters. Double-click on the parameter field, then click on the helper icon to see the available events.
ADD (ATTACH only)	IN	Bool	• ADD = 0 (default): This event replaces all previous event attachments for this OB. • ADD = 1: This event is added to previous event attachments for this OB.
RET_VAL	OUT	Int	Execution condition code

Hardware interrupt events

The following hardware interrupt events are supported by the CPU:

- Rising edge events (all built-in CPU digital inputs and SB digital inputs)
  - A rising edge occurs when the digital input transitions from OFF to ON as a response to a change in the signal from a field device connected to the input.
- Falling edge events (all built-in CPU digital inputs and SB digital inputs)
  - A falling edge occurs when the digital input transitions from ON to OFF.
- High-speed counter (HSC) current value = reference value (CV = RV) events (HSC 1 through 6)
  - A CV = RV interrupt for a HSC is generated when the current count transitions from an adjacent value to the value that exactly matches a reference value that was previously established.
- HSC direction changed events (HSC 1 through 6)
  - A direction changed event occurs when the HSC is detected to change from increasing to decreasing, or from decreasing to increasing.
- HSC external reset events (HSC 1 through 6)
  - Certain HSC modes allow the assignment of a digital input as an external reset that is used to reset the HSC count value to zero. An external reset event occurs for such a HSC, when this input transitions from OFF to ON.

Enabling hardware interrupt events in the device configuration

Hardware interrupts must be enabled during the device configuration. You must check the enable-event box in the device configuration for a digital input channel or a HSC, if you want to attach this event during configuration or run time.

Check box options within the PLC device configuration:

- Digital input
  - Enable rising edge detection
  - Enable falling edge detection
- High-speed counter (HSC)
  - Enable this high-speed counter for use
  - Generate interrupt for counter value equals reference value count
  - Generate interrupt for external reset event
  - Generate interrupt for direction change event

Adding new hardware interrupt OB code blocks to your program

By default, no OB is attached to an event when the event is first enabled. This is indicated by the "HW interrupt:" device configuration "<not connected>" label. Only hardware-interrupt OBs can be attached to a hardware interrupt event. All existing hardware-interrupt OBs appear in the "HW interrupt:" drop-down list. If no OB is listed, then you must create an OB of type "Hardware interrupt" as follows. Under the project tree "Program blocks" branch:

1. Double-click "Add new block", select "Organization block (OB)" and choose "Hardware interrupt".
2. Optionally, you can rename the OB, select the programming language (LAD or FBD), and select the block number (switch to manual and choose a different block number than that suggested).
3. Edit the OB and add the programmed reaction that you want to execute when the event occurs. You can call FCs and FBs from this OB, to a nesting depth of four.

OB_NR parameter

All existing hardware-interrupt OB names appear in the device configuration "HW interrupt:" drop-down list and in the ATTACH / DETACH parameter OB_NR drop-list.

EVENT parameter

When a hardware interrupt event is enabled, a unique default event name is assigned to this particular event. You can change this event name by editing the "Event name:" edit box, but it must be a unique name. These event names become tag names in the "Constants" tag table, and appear on the EVENT parameter drop-down list for the ATTACH and DETACH instruction boxes. The value of the tag is an internal number used to identify the event.

General operation

Each hardware event can be attached to a hardware-interrupt OB which will be queued for execution when the hardware interrupt event occurs. The OB-event attachment can occur at configuration time or at run time.

You have the option to attach or detach an OB to an enabled event at configuration time. To attach an OB to an event at configuration time, you must use the "HW interrupt:" drop-down list (click on the down arrow on the right) and select an OB from the list of available hardware-interrupt OBs. Select the appropriate OB name from this list, or select "<not connected>" to remove the attachment.

You can also attach or detach an enabled hardware interrupt event during run time. Use the ATTACH or DETACH program instructions during run time (multiple times if you wish) to attach or detach an enabled interrupt event to the appropriate OB. If no OB is currently attached (either from a "<not connected>" selection in device configuration, or as a result of executing a DETACH instruction), the enabled hardware interrupt event is ignored.

DETACH operation

Use the DETACH instruction to detach either a particular event or all events from a particular OB. If an EVENT is specified, then only this one event is detached from the specified OB_NR; any other events currently attached to this OB_NR will remain attached. If no EVENT is specified, then all events currently attached to OB_NR will be detached.

Condition codes

Table 8- 87 Condition codes

RET_VAL (W#16#....)	ENO	Description
0000	1	No error
0001	1	Nothing to Detach (DETACH only)
8090	0	OB does not exist
8091	0	OB is wrong type
8093	0	Event does not exist

8.4.2 Cyclic interrupts

8.4.2.1 SET_CINT (Set cyclic interrupt)

Table 8- 88 SET_CINT (Set cyclic interrupt instruction)

LAD / FBD	SCL	Description

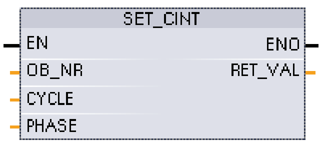
	`ret_val := SET_CINT(`
	`ob_nr:=_int_in_,`
	`cycle:=_udint_in_,`
	`phase:=_udint_in_);`
		Set the specified interrupt OB to begin cyclic execution that interrupts the program scan.

Table 8- 89 Data types for the parameters

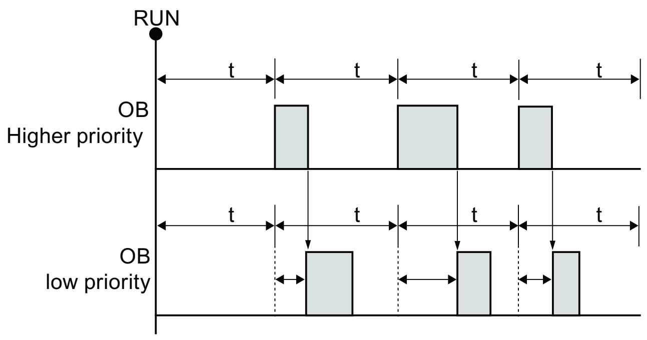
Parameter and type		Data type	Description
OB_NR	IN	OB_CYCLIC	OB number (accepts symbolic name)
CYCLE	IN	UDInt	Time interval, in microseconds
PHASE	IN	UDInt	Phase shift, in microseconds
RET_VAL	OUT	Int	Execution condition code

Time parameter examples:

- If the CYCLE time = 100 us, then the interrupt OB referenced by OB_NR interrupts the cyclic program scan every 100 us. The interrupt OB executes and then returns execution control to the program scan, at the point of interruption.
- If the CYCLE time = 0, then the interrupt event is deactivated and the interrupt OB is not executed.
- The PHASE (phase shift) time is a specified delay time that occurs before the CYCLE time interval begins. You can use the phase shift to control the execution timing of lower priority OBs.

If lower and higher priority OBs are called in the same time interval, the lower priority OB is only called after the higher priority OB has finished processing. The execution start time for the low priority OB can shift depending on the processing time of higher priority OBs.

OB call without phase shift



If you want to start the execution of a lower priority OB on a fixed time cycle, then phase shift time should be greater then the processing time of higher priority OBs.

OB-call with phase shift

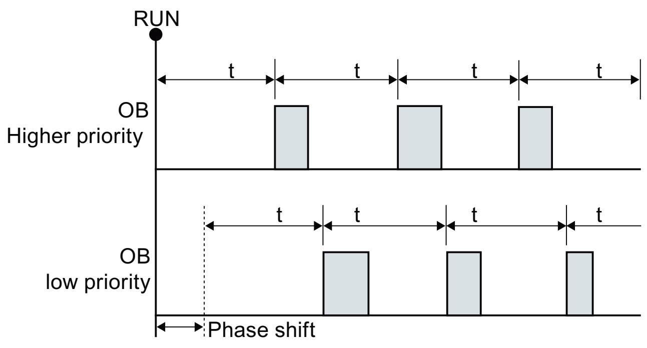


Table 8- 90 Condition codes

RET_VAL (W#16#....)	Description
0000	No error
8090	OB does not exist or is of wrong type
8091	Invalid cycle time
8092	Invalid phase shift time
80B2	OB has no attached event

8.4.2.2 QRY_CINT (Query cyclic interrupt)

Table 8- 91 QRY_CINT (Query cyclic interrupt)

LAD / FBD	SCL	Description

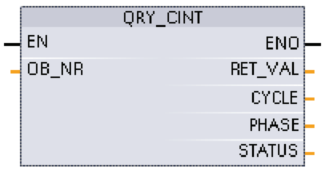
 QRY_CINT
EN
ENO
OB_NR
RET_VAL
CYCLE
PHASE
STATUS
	`ret_val := QRY_CINT(`
	`ob_nr:=_int_in_,`
	`cycle=>_udint_out_,`
	`phase=>_udint_out__,`
	`status=>_word_out_);`
		Get parameter and execution status from a cyclic interrupt OB. The values that are returned existed at the time QRY_CINT was executed.

Table 8- 92 Data types for the parameters

Parameter and type		Data type	Description
OB_NR	IN	OB_CYCLIC	OB number (accepts symbolic name like OB_MyOBName)
RET_VAL	OUT	Int	Execution condition code
CYCLE	OUT	UDInt	Time interval, in microseconds
PHASE	OUT	UDInt	Phase shift, in microseconds
STATUS	OUT	Word	Cyclic interrupt status code: • Bits 0 to 4, see the STATUS table below • Other bits, always 0

Table 8- 93 STATUS parameter

Bit	Value	Description
0	0	During CPU RUN
	1	During startup
1	0	The interrupt is enabled.
	1	Interrupt is disabled via the DIS_IRT instruction.
2	0	The interrupt is not active or has elapsed.
	1	The interrupt is active.
4	0	The OB identified by OB_NR does not exist.
	1	The OB identified by OB_NR exists.

Bit	Value	Description
Other Bits		Always 0

If an error occurs, RET_VAL displays the appropriate error code and the parameter STATUS = 0.

Table 8- 94 RET_VAL parameter

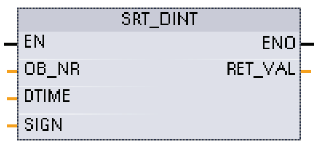
RET_VAL (W#16#....)	Description
0000	No error
8090	OB does not exist or is of wrong type.
80B2	OB has no attached event.

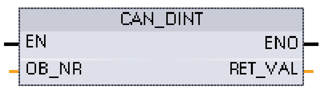
8.4.3 Time delay interrupts

You can start and cancel time delay interrupt processing with the SRT_DINT and CAN_DINT instructions, or query the interrupt status with the QRY_DINT instruction. Each time delay interrupt is a one-time event that occurs after the specified delay time. If the time delay event is cancelled before the time delay expires, the program interrupt does not occur.

Table 8- 95 SRT_DINT, CAN_DINT, and QRY_DINT instructions

LAD / FBD	SCL	Description

	`ret_val := SRT_DINT(`
	`    ob_nr:=_int_in_,`
	`    dtime:=_time_in_,`
	`    sign:=_word_in_);`
		SRT_DINT starts a time delay interrupt that executes an OB when the delay time specified by parameter DTIME has elapsed.

	`ret_val := CAN_DINT(`
	`    ob_nr:=_int_in_);`
		CAN_DINT cancels a time delay interrupt that has already started. The time delay interrupt OB is not executed in this case.

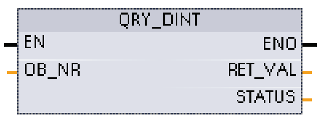
	`ret_val := QRY_DINT(`
	`    ob_nr:=_int_in_,`
	`    status=>_word_out_);`
		QRY_DINT queries the status of the time delay interrupt specified by the OB_NR parameter.

Table 8- 96 Data types for the parameters

Parameter and type		Data type	Description
OB_NR	IN	OB_DELAY	Organization block (OB) to be started after a time-delay: Select from the available time-delay interrupt OBs that were created using the "Add new block" project tree feature. Double-click on the parameter field, then click on the helper icon to see the available OBs.
DTIME [1]	IN	Time	Time delay value (1 to 60000 ms)

Parameter and type		Data type	Description
SIGN [1]	IN	Word	Not used by the S7-1200: Any value is accepted. A value must be assigned to prevent errors.
RET_VAL	OUT	Int	Execution condition code
STATUS	OUT	Word	QRY_DINT instruction: Status of the specified time-delay interrupt OB, see the table below

[1] Only for SRT_DINT

Operation

The SRT_DINT instruction specifies a time delay, starts the internal time delay timer, and associates a time delay interrupt OB subprogram with the time delay timeout event. When the specified time delay has elapsed, a program interrupt is generated that triggers the execution of the associated time delay interrupt OB. You can cancel an in-process time delay interrupt before the specified time delay occurs by executing the CAN_DINT instruction. The total number of active time delay and cyclic interrupt events must not exceed four.

Adding time delay interrupt OB subprograms to your project

Only time delay interrupt OBs can be assigned to the SRT_DINT and CAN_DINT instructions. No time delay interrupt OB exists in a new project. You must add time delay interrupt OBs to your project. To create a time-delay interrupt OB, follow these steps:

1. Double-click the "Add new block" item in the "Program blocks" branch of the project tree, select "Organization block (OB)", and choose "Time delay interrupt".
2. You have the option to rename the OB, select the programming language, or select the block number. Switch to manual numbering if you want to assign a different block number than the number that was assigned automatically.
3. Edit the time delay interrupt OB subprogram and create programmed reaction that you want to execute when the time delay timeout event occurs. You can call other FC and FB code blocks from the time delay interrupt OB, with a maximum nesting depth of four.
4. The newly assigned time delay interrupt OB names will be available when you edit the OB_NR parameter of the SRT_DINT and CAN_DINT instructions.

QRY_DINT parameter STATUS

Table 8- 97 If there is an error (REL_VAL <> 0), then STATUS = 0.

Bit	Value	Description
0	0	In RUN
	1	In startup
1	0	The interrupt is enabled.
	1	The interrupt is disabled.
2	0	The interrupt is not active or has elapsed.
	1	The interrupt is active.

Bit	Value	Description
4	0	An OB with an OB number given in OB_NR does not exist.
	1	An OB with an OB number given in OB_NR exists.
Other bits		Always 0

Condition codes

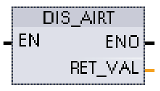
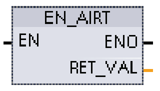
Table 8- 98 Condition codes for SRT_DINT, CAN_DINT, and QRY_DINT

RET_VAL (W#16#...)	Description
0000	No error occurred
8090	Incorrect parameter OB_NR
8091	Incorrect parameter DTIME
80A0	Time delay interrupt has not started.

8.4.4 Asynchronous event interrupts

Use the DIS_AIRT and EN_AIRT instructions to disable and enable alarm interrupt processing.

Table 8- 99 DIS_AIRT and EN_AIRT instructions

LAD / FBD	SCL	Description
	`DIS_AIRT();`	DIS_AIRT delays the processing of new interrupt events. You can execute DIS_AIRT more than once in an OB.
	`EN_AIRT();`	EN_AIRT enables the processing of interrupt events that you previously disabled with the DIS_AIRT instruction. Each DIS_AIRT execution must be cancelled by an EN_AIRT execution. The EN_AIRT executions must occur within the same OB, or any FC or FB called from the same OB, before interrupts are enabled again for this OB.

WARNING

If the filter time for a digital input channel is changed from a previous setting, a new "0" level input value may need to be presented for up to 20.0 ms accumulated duration before the filter becomes fully responsive to new inputs. During this time, short "0" pulse events of duration less than 20.0 ms may not be detected or counted.

This changing of filter times can result in unexpected machine or process operation, which may cause death or serious injury to personnel, and/or damage to equipment.

To ensure that a new filter time goes immediately into effect, a power cycle of the CPU must be applied.

Table 8- 100 Data types for the parameters

Parameter and type		Data type	Description
RET_VAL	OUT	Int	Number of delays = number of DIS_AIRT executions in the queue.

The DIS_AIRT executions are counted by the operating system. Each of these remains in effect until it is cancelled again specifically by an EN_AIRT instruction, or until the current OB has been completely processed. For example: if you disabled interrupts five times with five DIS_AIRT executions, you must cancel these with five EN_AIRT executions before interrupts become enabled again.

After the interrupt events are enabled again, the interrupts that occurred while DIS_AIRT was in effect are processed, or the interrupts are processed as soon as the current OB has been executed.

Parameter RET_VAL indicates the number of times that interrupt processing was disabled, which is the number of queued DIS_AIRT executions. Interrupt processing is only enabled again when parameter RET_VAL = 0.

8.5 Diagnostics (PROFINET or PROFIBUS)

8.5.1 Diagnostic instructions

The following diagnostic instructions can be used with either PROFINET or PROFIBUS:

- GET_DIAG instruction (Page 302): You can read the diagnostic information from a specified device.
- DeviceStates instruction (Page 299): You can retrieve the operational states for a distributed I/O device within an I/O subsystem.
- ModuleStates instruction (Page 301): You can retrieve the operational states for the modules in a distributed I/O device.
- LED instruction (Page 298): You can read the state of the LEDs for a distributed I/O device.

8.5.2 Diagnostic events for distributed I/O

Note

With a PROFIBUS IO system, after a download or power cycle, the CPU will go to RUN mode unless the hardware compatibility is set to allow acceptable substitute modules (Page 123) and one or more modules is missing or is not an acceptable substitute for the configured module.

As shown in the following table, the CPU supports diagnostics that can be configured for the components of the distributed I/O system. Each of these errors generates a log entry in the diagnostic buffer.

Table 8- 101 Handling of diagnostic events for PROFINET and PROFIBUS

Type of error	Diagnostic information for the station?	Entry in the diagnostic buffer?	CPU operating mode
Diagnostic error	Yes	Yes	Stays in RUN mode
Rack or station failure	Yes	Yes	Stays in RUN mode
I/O access error [1]	No	Yes	Stays in RUN mode
Peripheral access error [2]	No	Yes	Stays in RUN mode
Pull / plug event	Yes	Yes	Stays in RUN mode

[1] I/O access error example cause: A module that has been removed.

[2] Peripheral access error example cause: Acyclic communication to a submodule that is not communicating.

Use the GET_DIAG instruction (Page 302) for each station to obtain the diagnostic information. This will allow you to programmatically handle the errors encountered on the device and if desired take the CPU to STOP mode. This method requires you to specify the hardware device from which to read the status information.

The GET_DIAG instruction uses the "L address" (LADDR) of the station to obtain the health of the entire station. This L Address can be found within the Network Configuration view and by selecting the entire station rack (entire gray area), the L Address is shown in the Properties Tab of the station. You can find the LADDR for each individual module either in the properties for the module (in the device configuration) or in the default tag table for the CPU.

8.5.3 LED instruction

Table 8- 102 LED instruction

LAD / FBD	SCL	Description

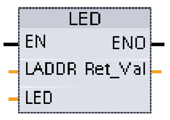
	`ret_val := LED(`
	`laddr:=_word_in_,`
	`LED:=_uint_in_);`
		Use the LED instruction to read the state of the LEDs on a CPU or interface. The specified LED state is returned by the RET_VAL output.

Table 8- 103 Data types for the parameters

Parameter and type		Data type	Description		
LADDR	IN	HW_IO	Identification number of the CPU or interface[1]		
LED	IN	UInt	LED identifier number		
			1	RUN/STOP	Color 1 = green, color 2 = yellow
			2	Error	Color 1 = red

Parameter and type		Data type	Description		
			3	Maintenance	Color 1 = yellow
			4	Redundancy	Not applicable
			5	Link	Color 1 = green
			6	Tx/Rx	Color 1 = yellow
RET_VAL	OUT	Int	Status of the LED		

[1] For example, you can select the CPU (such as "PLC_1") or the PROFINET interface from the drop-down list of the parameter.

Table 8- 104 Status of RET_VAL

RET_VAL (W#16#...)	Description	
0 to 9 LED state	0	LED does not exist
	1	Off
	2	Color 1 On (solid)
	3	Color 2 On (Solid)
	4	Color 1 flashing at 2 Hz
	5	Color 2 flashing 2 Hz
	6	Color 1 & 2 flashing alternatively at 2 Hz
	7	Color 1 on (Tx/Rx)
	8	Color 2 on (Tx/Rx)
	9	State of the LED is not available
8091	Device identified by LADDR does not exist	
8092	Device identified by LADDR does not support LEDs	
8093	LED identifier not defined	
80Bx	CPU identified by LADDR does not support the LED instruction	

8.5.4 DeviceStates instruction

Table 8- 105 DeviceStates instruction

LAD / FBD	SCL	Description

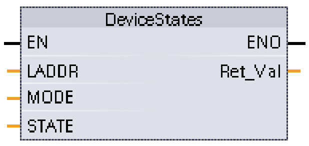
 DeviceStates
EN
ENO
LADDR
Ret_Val
MODE
STATE
	`ret_val := DeviceStates(`
	`    laddr:=hw_io_in_,`
	`    mode:=_uint_in_,`
	`    state:=_variant_inout_);`
		DeviceStates retrieves the I/O device operational states of an I/O subsystem. After execution, the STATE parameter contains the error state of each I/O device in a bit list (for the assigned LADDR and MODE). This information corresponds with the device status seen in the STEP 7 diagnostics view.

Table 8- 106 Data types for the parameters

Parameter and type		Data type	Description
LADDR	IN	HW_IOSYSTEM	Logical address: (Identifier for the I/O system)
MODE	IN	UInt	Status type: • 1: Configuration of device is active or not yet complete. • 2: Device defective • 3: Device disabled • 4: Device exists
RET_VAL	OUT	Int	Execution condition code
STATE[1]	InOut	Variant	Buffer that receives the error status of each device: The data type that you choose for the STATE parameter can be any bit type (Bool, Byte, Word, or DWord) or an array of a bit type • Summary bit: Bit 0 =1, if one of the state bits of the I/O devices is 1 • State bit: State of I/O device with station number n according to the selected MODE. For example, MODE = 2 and bit 3 = 1 means station 3 is faulty.

[1] For PROFIBUS-DP, the length of the status information is 128 bits. For PROFINET I/O, the length is 1024 bits.

After execution, the STATE parameter contains the error state of each I/O device as a bit list (for the assigned LADDR and MODE).

Table 8- 107 Condition codes

RET_VAL (W#16#...)	Description
0	No error
8091	LADDR does not exist.
8092	LADDR does not address an I/O system.
8093	Invalid data type assigned for STATE parameter: Valid data types are (Bool, Byte, Word, or Dword), or an array of (Bools, Bytes, Words, or Dwords)
80Bx	DeviceStates instruction not supported by the CPU for this LADDR.
8452	The complete state data is too large for the assigned STATE parameter. The STATE buffer contains a partial result.