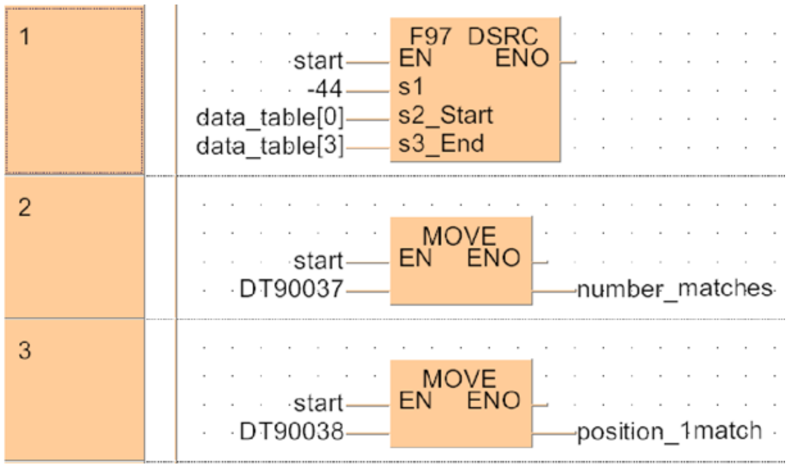
Panasonic®

PROGRAMMABLE CONTROLLER

FP Series

Programming Manual



Panasonic® FP Series Programming Manual

Panasonic Electric Works Europe AG

ACGM0313V2EN 7/2012

BEFORE BEGINNING

Liability and Copyright for the Hardware

This manual and everything described in it are copyrighted. You may not copy this manual, in whole or part, without written consent of Panasonic Electric Works Europe AG (PEWEU).

PEWEU pursues a policy of continuous improvement of the design and performance of its products. Therefore we reserve the right to change the manual/product without notice. In no event will PEWEU be liable for direct, special, incidental, or consequential damage resulting from any defect in the product or its documentation, even if advised of the possibility of such damages.

We invite your comments on this manual. Please e-mail us at:

techdoc.peweu@eu.panasonic.com.

Please direct support matters and technical questions to your local Panasonic representative.

LIMITED WARRANTY

If physical defects caused by distribution are found, PEWEU will replace/repair the product free of charge. Exceptions include:

- When physical defects are due to different usage/treatment of the product other than described in the manual.
- When physical defects are due to defective equipment other than the distributed product.
- When physical defects are due to modifications/repairs by someone other than PEWEU.
- When physical defects are due to natural disasters.

Important symbols

One or more of the following symbols may be used in this documentation:

DANGER!

The warning triangle indicates especially important safety instructions. If they are not adhered to, the results could be fatal or critical injury.

◆ CAUTION

Indicates that you should proceed with caution. Failure to do so may result in injury or significant damage to instruments or their contents, e.g. data.

◆ NOTE

Contains important additional information.

◆ EXAMPLE

Contains an illustrative example of the previous text section.

◆ Procedure

Indicates that a step-by-step procedure follows.

◆ REFERENCE

Indicates where you can find additional information on the subject at hand.

Table of Contents

Part I Basics

Part II IEC instructions

14. Timer instructions 323

Part III FP instructions

15. Arithmetic instructions 341

20. Conversion instructions 621

Part IV Tool instructions

Chapter 1

Basics

1.1 Operands

In FPWIN Pro the following operands are available:

- in- and outputs (X/Y) as well as internal memory areas
- internal relays
- special internal relays
- timers and counters
- data registers
- special data registers
- file registers
- link registers and relays

The number of operands which are available depends on the PLC-type and its configuration. To see how many of the respective operands are available, see your hardware description.

1.1.1 Inputs/Outputs

The amount of inputs/outputs available depends on the PLC and unit type. Each input terminal corresponds to one input **X,** each output terminal corresponds to one output **Y**.

In system register 20 you set whether an output can be used once or more during the program.

☞ **Outputs which do not exist physically can be used like flags. These flags are non-holding, which means their contents will be lost, e.g. after a power failure.**

1.1.2 Internal Relays

Internal Relays are memory areas where you can store interim results. Internal relays are treated like internal outputs.

In system register no. 7 you define which internal relays are supposed to be holding/non-holding. Holding means that its values will be retained even after a power failure.

The number of available internal relays depends on the PLC type (see hardware description of your PLC).

1.1.3 Special Internal Relays

Special internal relays are memory areas which are reserved for special PLC functions. They are automatically set/reset by the PLC and are used:

- to indicate certain system states, e.g. errors
- as an impulse generator
- to initialize the system
- as ON/OFF control flag under certain conditions

such as when some flags get a certain status if data are ready for transmission in a PLC network.

The number of special internal relays available depends on the PLC type (see hardware description of your PLC).

☞ **Special internal relays can only be read.**

1.1.4 Timers and Counters

Timers and Counters use one common memory and address area.

Define in system registers 5 and 6 how the memory area is to be divided between timers and counters and which timers/counters are supposed to be holding or non-holding. Holding means that even after a power failure all data will be saved, which is not the case in non-holding registers.

Entering a number in system register 5 means that the first counter is defined. All smaller numbers define timers.

For example, if you enter zero, you define counters only. If you enter the highest value possible, you define timers only.

In the default setting the holding area is defined by the start address of the counter area. This means all timers are holding and all counters are non-holding. You can of course customize this setting and set a higher value for the holding area, which means some of the timers, or if you prefer, all of them can be defined as holding.

In addition to the timer/counter area, there is a memory area reserved for the set value (SV) and the elapsed value (EV) of each timer/counter contact. The size of both areas is 16 bits (WORD). In the SV and EV area one INTEGER value from 0 to 32,767 can be stored.

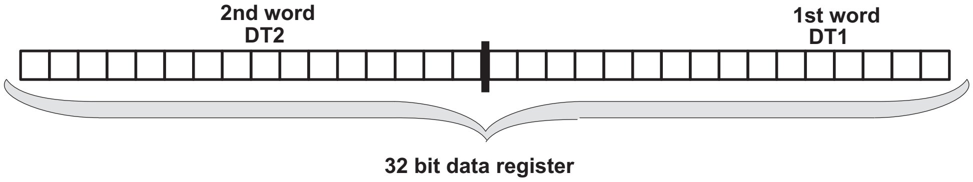
Timer/Counter No.	SV	EV	Relay
TM0	SV0	EV0	T0
. . .	. . .	. . .	. . .
TM99	SV99	EV99	T99
CT100	SV100	EV100	C100
. . .	. . .	. . .	. . .

While a timer or counter is being processed, the respective acual value can be read and under certain conditions be edited.

☞ **After changing the settings in system register 5, do not forget to adjust the addresses of the timers/counters in your PLC program because they correspond to the TM/CT numbers.**

1.1.5 Data Registers (DT)

Data registers have a width of 16 bits. You can use them, for example, to write and read constants/parameters. If an instruction requires 32 bits, two 16-bit data registers are used. If this is the case, enter the address of the first data register with the prefix DDT instead of DT. The next data register (word) will be used automatically (for more information, please refer to addresses (see page 30)).



Data registers can be holding or non-holding. Holding means that even after a power failure all data will be saved. Set the holding/non-holding areas in system register 8 by entering the start address of the holding area.

The amount of data registers available depends on the PLC type (see hardware description).

1.1.6 Special Data Registers (DT)

Special data registers are like the special internal relays reserved for special functions and are in most cases set/reset by the PLC.

The register has a width of 16 bits (data type = WORD). The amount of special data registers available depends on the PLC type (see hardware description).

Most special data registers can only be read. Here some exceptions:

- interrupts and scan time (DT9027, DT9023-DT9024; FP0 T32P DT90027, DT90023 to DT90024)...
- actual values of the high-speed counter (DT9044 and DT9045; for FP0-T32CP DT90044 and DT90045)
- control flag of the high-speed counter DT9052 (DT90053 for FP0-T32CP)
- real-time clock (FP2, FP2SH: DT90054 to DT90058; FP0-T32CP: DT90054 to DT90058)

See also:

Data Transfer to and from Special Data Registers (see page 859)

1.1.7 File Registers (FL)

Some PLC types (see hardware description) provide additional data registers which can be used to increase the number of data registers. File registers are used in the same way as data registers. Set the holding/non-holding area in system register 9. Holding means that even after a power failure all data will be saved.

1.1.8 Link Relays and Registers (L/LD)

Link relays have a width of 1 bit (BOOL). In system registers 10-13 and 40-55, set the:

- transmission area
- amount of link relay words to be sent
- holding/non-holding area

Link registers have a width of 16 bits (WORD). In system registers 10-13 and 40-55, set the:

- transmission area
- amount of link relay words to be sent
- holding/non-holding area

1.2 Addresses

In the List of Global Variables, enter the physical address in the field “Address” for each global variable used in the PLC program.

The operand and the address number are part of the address. In FPWIN Pro you can use either FP and/or IEC addresses. The following abbreviations are used:

Meaning	FP	IEC
Input	X	I
Output	Y	Q
Memory (internal memory area)	R	M0
Timer relay	T	M1
Counter relay	C	M2
Set value	SV	M3
Elapsed value	EV	M4
Data register	DT/DDT	M5
Link relay	L	M6
Link register	LD	M7
File register	FL	M8

You find the register numbers (e.g. DT9000/DT90000) in your hardware description. The next two sections show how FP and IEC addresses are composed.

1.2.1 FP Addresses

An address represents the hardware address of an in-/output, register, or counter.

For example, the hardware address of the 1st input and the 4th output of a PLC is:

- X0 (X = input, 0 = first relay)
- Y3 (Y = output, 3 = fourth relay)

Use the following address abbreviations for the memory areas. You find the register numbers in your hardware description.

Memory Area	Abbr. FP	Example
Memory (internal memory area)	R	R9000: self diagnostic error
Timer relay	T	T200: timer relay no. 200 (settings in system register 5+6)
Counter relay	C	C100: counter relay no. 100 (settings in system register 5+6)
Set value	SV	SV200 (set value for counter relay 200)
Elapsed value	EV	EV100 (elapsed value for timer relay 100)
Data register	DT	DT9001/DT90001 (signals power failure)
Link relay	L	L1270
Link register	LD	LD255
File register	FL	FL8188

1.2.2 IEC Addresses

The composition of an IEC-1131 address depends on:

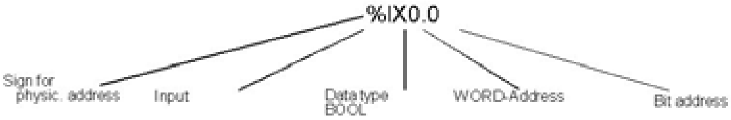
- operand type
- data type
- slot no. of the unit (word address)
- relay no. (bit address)
- PLC type

In- and Outputs are the most important components of a programmable logic controller (PLC). The PLC receives signals from the input relays and processes them in the PLC program. The results can either be stored or sent to the output relays, which means the PLC controls the outputs.

A PLC provides special memory areas, in short “M”, to store interim results, for example.

If you want to read the status of the input 1 of the first module and control the output 4 of the second module, for example, you need the physical address of each in-/output. Physical FPWIN Pro addresses are composed of the per cent sign, an abbreviation for in-/output, an abbreviation for the data type and of the word and bit address:

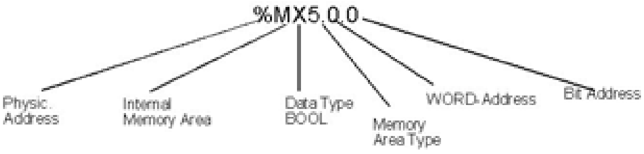
Example **IEC address for an input**



The per cent sign is the indicator of a physical address. “I” means input, “X” means data type BOOL. The first zero represents the word address (slot no.) and the second one the bit address. Note that counting starts with zero and that counting word and bit addresses differs among the PLC types.

Each PLC provides internal memory areas (M) to store interim results, for example. When using internal memory areas such as data registers, do not forget the additional number (here 5) for the memory type:

Example **IEC address for an internal memory area**



Bit addresses do not have to be defined for data registers, counters, timers, or the set and actual values.

According to IEC 1131, abbreviations for **in- and output** are “I” and “Q”, respectively. Abbreviations for the **memory areas** are as follows:

Memory Type	No.	Example
Internal Relay (R)	0	%MX0.900.0 = internal relay R9000
Timer (T)	1	%MX1.200 = counter no. 200
Counter (C)	2	%MX2.100 = counter no. 100
Set Value counters/timers (SV)	3	%MW3.200 = set value of the counter no. 200

Memory Type	No.	Example
Elapsed Value counters/timers (EV)	4	%MW4.100 = elapsed value of the timer no. 100
Data Registers (DT, DDT)	5	%MW5.9001 = data register DT9001 %MD5.90001 = 32-bit data register DDT90001
Link Relay (WL)	7	%MW7.63 = link relay 63
Link Register (LD)	8	%MW8.127 = link register 127
File Register (FL)	9	%MW9.800 = file register 800

☞ **Tables with hardware addresses can be found in the hardware description of your PLC.**

The following data types are available:

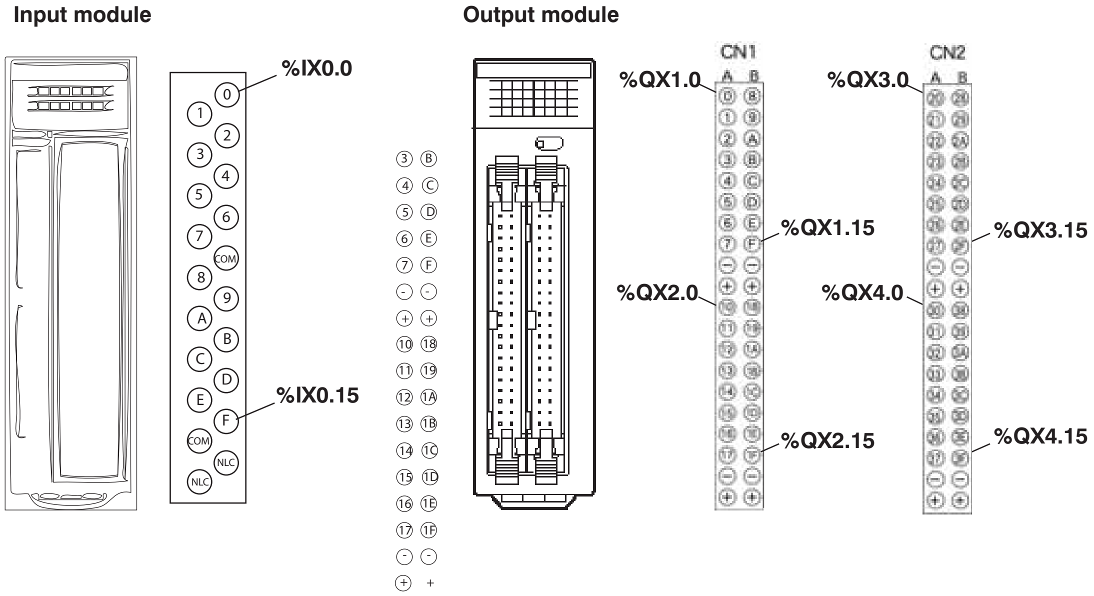
Keyword	Data type	Range	Reserved memory	Initial value
BOOL	Boolean	0 (FALSE) 1 (TRUE)	1 bit	0
WORD	Bit string of length 16	0–65535	16 bits	0
DWORD	Bit string of length 32	0–4294967295	32 bits	0
INT	Integer	-32768–32,767	16 bits	0
DINT	Double integer	-2147483648– 2147483647	32 bits	0
UINT	Unsigned integer	0–65,535	16 bits	0
UDINT	Unsigned double integer	0–4294967295	32 bits	0
REAL	Real number	-3.402823466*E38–-1.175494351*E-38 0.0 +1.175494351*E-38–+3.402823466*E38	32 bits	0.0
TIME	Duration	T#0s–T#327.67s	16 bits [1]	T#0s
		T#0s–T#21474836.47s	32 bits [1]	
DATE_AND_TIME	Date and time	DT#2001-01-01-00:00:00–DT#2099-12-31-23:59:59	32 bits	DT#2001-01-01-00:00:00
DATE	Date	D#2001-01-01–D#2099-12-31	32 bits	D#2001-01-01
TIME_OF_DAY	Time of day	TOD#00:00:00–TOD#23:59:59	32 bits	TOD#00:00:00
STRING	Variable-length character string	1–32767 bytes (ASCII) depending on PLC memory size	2 words for the head + (n+1)/2 words for the characters	''

[1] Depending on PLC type

☞ **Please take into account that not all data types can be used with each IEC command.**

Numbering of in-/output addresses depends on the type of PLC used (see respective hardware description). For FP0, FP-Sigma the addresses **are not serially numbered**. Counting restarts with zero at the first output. Supposing you have one FP1-C24 with 16 inputs and 8 outputs, the resulting addresses are: for the input: %IX0.0 - %IX0.15, and for the output: %QX0.0 - %QX0.7. In other words the counting for the word and bit number begins at zero for the outputs.

In-/Output addresses are **numbered serially**. Supposing the first slot of your PLC contains an input module with 16 inputs and the second slot of your PLC contains an output module with 32 outputs, the input module occupies the addresses: %IX0.0 - %IX0.15, the output module: %QX1.0 - %QX2.15. The physical address depends therefore on the module type (I/Q), the slot number (word address) the module is assigned and the relay number (bit address).


Input module

Output module

This shows how the hexadecimal counting of 0-F for 0-15 is converted. The address assignment can be found in your hardware description.

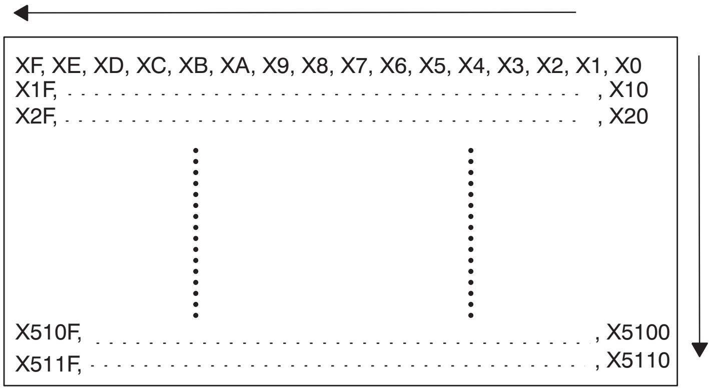
☞

- **Find the tables with all memory areas in your hardware description.**
- **When using timers, counters, set/elapsed values, and data registers, the bit address does not have to be indicated.**
- **You can also enter the register number (R9000, DT9001/90001) or the FP address, e.g. “X0” (input 0), instead of the IEC address.**

1.2.3 Specifying Relay Addresses

External input relay (X), external output relay (Y), internal relay (R), link relay (L) and pulse relay (P)The lowest digit for these relay’s adresses is expressed in hexadecimals and the second and higher digits are expressed in decimals as shown below.

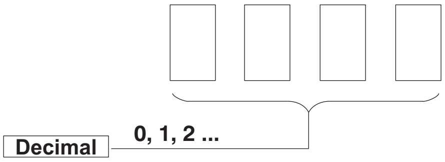
Example **Configuration of external input relay (X)**



XF, XE, XD, XC, XB, XA, X9, X8, X7, X6, X5, X4, X3, X2, X1, X0
X1F, ... , X10
X2F, ... , X20
X510F, ... , X5100
X511F, ... , X5110

1.2.4 Timer Contacts (T) and Counter Contacts (C)

Addresses of timer contacts (T) and counter contacts (C) correspond to the **TM** and **CT** instruction numbers and depend on the PLC type.



e.g. for FP2:
T0, T1 .... T2999
C3000, C3001 .... C3072

Since addresses for timer contacts (T) and counter contacts (C) correspond to the TM and CT instruction numbers, if the TM and CT instruction sharing is changed by system register 5, timer and counter contact sharing is also changed.

1.2.5 Error alarm relays

NOTE

Error alarm relays are only available for FP2SH/FP10SH.

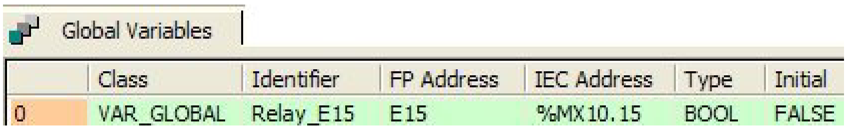
Restrictions of error alarm relays (see page 36)

Error alarm relays are designed to facilitate the analysis of error conditions and to record errors. Therefore in the special data registers a buffer has been defined so that the user has access to information about errors and their occurrence, including the actual number of error relays in the TRUE state, the order they were set to TRUE and the time at which the first error relay was set to TRUE.

When an error relay is set to TRUE by the error alarm program because the corresponding error situation has arisen, the number of error relays in the TRUE state stored in special data register DT90400 increases by one each time an error occurs. Relay numbers will be stored in DT90401 through DT90419 in the order that they were set to TRUE. If at least one of the error alarm relays E0 through E2047 is set to TRUE, R9040 (sys_bIsErrorAlarmRelayOn) will be set to TRUE. The time at which the first error alarm relay was set to TRUE is stored in DT90420 through DT90422.

The diagram below illustrates the internal structure and address assignment in the special data register area of this error buffer.

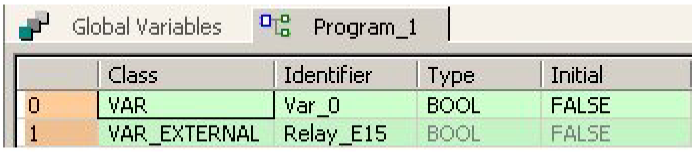
GVL



Global Variables

	Class	Identifier	FP Address	IEC Address	Type	Initial
0	VAR_GLOBAL	Relay_E15	E15	%MX10.15	BOOL	FALSE

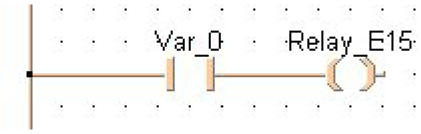
POU Header



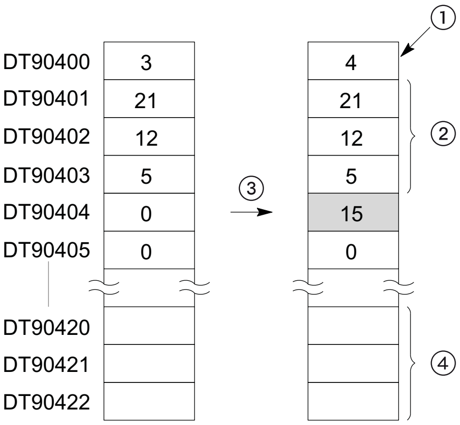
Global Variables | Program_1

	Class	Identifier	Type	Initial
0	VAR	Var_0	BOOL	FALSE
1	VAR_EXTERNAL	Relay_E15	BOOL	FALSE

LD



Error alarm diagram



DT90400	3	4
DT90401	21	21
DT90402	12	12
DT90403	5	5
DT90404	0	15
DT90405	0	0
DT90420		
DT90421		
DT90422		

①	DT90400	Number of alarms that have occurred
②	DT90401–DT90419	Error alarm relays stored in the order they were set to TRUE
③	Error alarm E15 is set to TRUE	
④	Time at which the first error alarm relay was set to TRUE:	
	DT90420	Second and minute data
	DT90421	Hour and day data
	DT90422	Month and year data
	R9040	TRUE when one of the error alarm relays E0–E2047 is TRUE

Because in Control FPWIN Pro all write operations on error relays are internally compiled into SET (see page 508) and RST (see page 508) instructions, all write operations to an error relay affect the special internal relay

R9040 and the special data registers DT90400 to DT90422.

When all error alarm relays are set to FALSE, R9040 will be set to FALSE.

To monitor alarm relays using Control FPWIN Pro: **Monitor** → **Special Relays and Registers** → **Alarm Relays**.

1.2.5.1 Restrictions of error alarm relays

There is no limit to the number of times an error alarm relay can be used in a program. However, if one error alarm relay is used with different error conditions in more than one error alarm program it will not be possible to accurately determine the nature of the error. The CPU does not check for multiple use.

When the power is turned OFF or when switching between PROG. and RUN, the error relays as well as the affected special data registers are held. To reset the error relays and the special data registers, you have to press up the initialize/test switch in PROG. mode.

However, in system register 4, bit 10 (Error alarm relay) can be set to "Clear not" to ensure that no error alarm relays are turned OFF when the initialize/test switch is pressed up. Then only the next download of the program will reset the error relays and the corresponding special data registers.

1.2.6 Pulse relays (P)

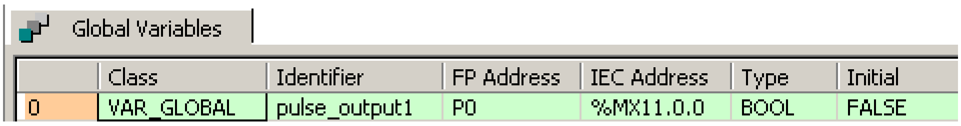
A pulse relay (P) goes ON for one scan only. The ON/OFF state is not externally output and only operates in the program.

A pulse relay only goes on when a rising edge start instruction or a falling edge start instruction is executed.

When used as the trigger, a pulse relay only operates during one scan when a leading edge or trailing edge is detected.

Example: Declared globally

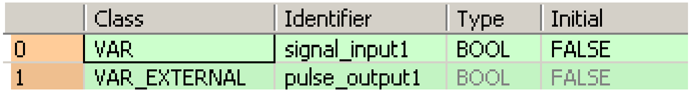
GVL:



Global Variables

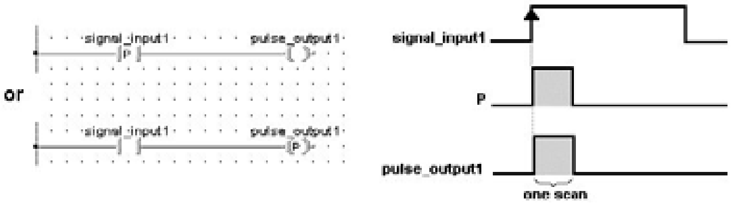
	Class	Identifier	FP Address	IEC Address	Type	Initial
0	VAR_GLOBAL	pulse_output1	P0	%MX11.0.0	BOOL	FALSE

POU Header

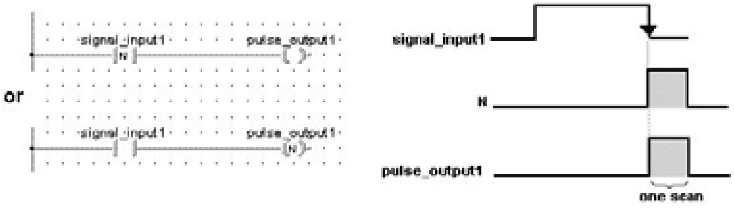


	Class	Identifier	Type	Initial
0	VAR	signal_input1	BOOL	FALSE
1	VAR_EXTERNAL	pulse_output1	BOOL	FALSE

Execution with a rising edge:



Execution with a falling edge:



1.2.6.1 Restrictions of pulse relay (P)

A pulse relay can only be used once in a program as an output destination, i.e. duplicate output is prohibited.

There is no limitation on the number of times a pulse relay can used as a contact.

A pulse relay cannot be specified as an output destination for a **KP**, **SET**, **RST** or **ALT** instruction.

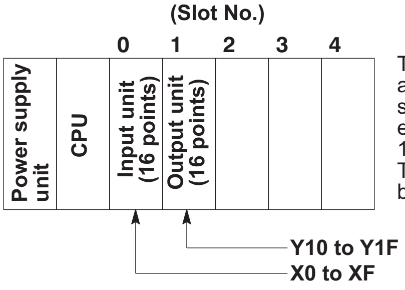
A word unit pulse relay (WP) cannot be specified as a storage location for a high-level instruction.

In Control FPWIN Pro pulse relays can only be used in the above situations or together with a DF or DFN instruction. Although one reason might be to increase the number of relays, there is no special reason to use these pulse relays in Control FPWIN Pro.

1.2.7 External input (X) and output relays (Y)

- The external input relays available are those actually allocated for input use.
- The external output relays actually allocated for output can be used for turning ON or OFF external devices. The other external output relays can be used in the same way as internal relays.
- I/O allocation is based on the combination of I/O and intelligent modules installed.For FP10SH and FP2SH, 8,192 points can be used, including both input and output. For FP2 and FP3, 2048 points can be used.

Example



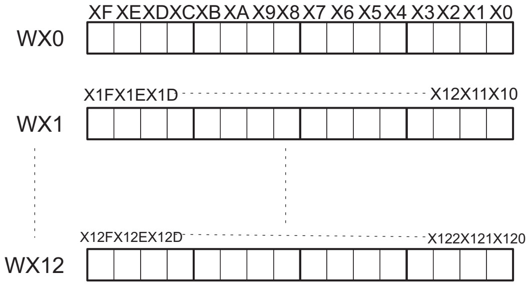
The X0 to XF 16-point external relay is allocated to the 16-point input unit in slot 0, and the Y10 to Y1F 16-point external output is allocated to the 16-point output unit in slot 1.
The sixteen points X10 to X1F cannot be used in this combination.

1.2.8 Word representation of relays (WX, WY, WR, and WL)

The external input relay (X), external output relay (Y), internal relay (R) and link relay (L) can also be expressed in word format. The word format treats 16-bit relay groups as one word. The word expressions for these relays are word external input relay (WX), word external output relay (WY), word internal relay (WR) and word link relay (WL), respectively.

Example:

Configuration of word external input relay (WX)



☞ **Since the contents of the word relay correspond to the state of its relays (components), if some relays are turned ON, the contents of the word change.**

1.3 Constants

A constant represents a fixed value. Depending on the application, a constant can be used as an addend, multiplier, address, in-/output number, set value, etc.

There are 3 types of constants:

- decimal
- hexadecimal
- BCD

1.3.1 Decimal Constants

Decimal constants can have a width of either 16 or 32 bits.

Range 16 bit: -32,768 to 32,768

Range 32 bit: -2,147,483,648 to 2,147,483,648

Constants are internally changed into 16-bit binary numbers including character bit and are processed as such. Simply enter the decimal number in your program.

1.3.2 Hexadecimal Constants

Hexadecimal constants occupy fewer digit positions than binary data. 16 bit constants can be represented by 4-digit, 32-bit constants by 8-digit hecadecimal constants.

Range 16 bit: 8000 to 7FFF

Range 32 bit: 80000000 to 7FFFFFFFF

Enter e.g.: 16#7FFF for the hexadecimal value 7FFF in your program.

1.3.3 BCD Constants

BCD is the abbreviation for Binary Coded Decimal.

Range 16 bit: 0 to 9999

Range 32 bit: 0 to 99999999

Enter BCD constants in the program either as:

binary: 2#0001110011100101 or
hexadecimal: 16#9999

1.4 Data types

In Control FPWIN Pro, variable declarations require a data type. All data types conform to IEC61131-3.

For details, please refer to the Programming Manual or to the online help of Control FPWIN Pro.

1.4.1 Elementary data types

Keyword	Data type	Range	Reserved memory	Initial value
BOOL	Boolean	0 (FALSE) 1 (TRUE)	1 bit	0
WORD	Bit string of length 16	0–65535	16 bits	0
DWORD	Bit string of length 32	0–4294967295	32 bits	0
INT	Integer	-32768–32,767	16 bits	0
DINT	Double integer	-2147483648– 2147483647	32 bits	0
UINT	Unsigned integer	0–65,535	16 bits	0
UDINT	Unsigned double integer	0–4294967295	32 bits	0
REAL	Real number	-3.402823466*E38– -1.175494351*E-38 0.0 +1.175494351*E-38– +3.402823466*E38	32 bits	0.0
TIME	Duration	T#0s–T#327.67s	16 bits [1]	T#0s
		T#0s–T#21474836.47s	32 bits [1]	
DATE_AND_TIME	Date and time	DT#2001-01-01-00:00:00– DT#2099-12-31-23:59:59	32 bits	DT#2001-01-01-00 :00:00
DATE	Date	D#2001-01-01–D#2099-12-31	32 bits	D#2001-01-01
TIME_OF_DAY	Time of day	TOD#00:00:00–TOD#23:59:59	32 bits	TOD#00:00:00
STRING	Variable-length character string	1–32767 bytes (ASCII) depending on PLC memory size	2 words for the head + (n+1)/2 words for the characters	"

[1] Depending on PLC type

1.4.1.1 BOOL

Variables of the data type BOOL are binary variables. They can only have the value 0 or 1, and always have a width of 1 bit.

The condition 0 corresponds to **FALSE** (e.g. initial value in the POU header) and means that the variable is switched off. In this case we also speak of the variable not being set.

The condition 1 corresponds to **TRUE** (e.g. initial value in the POU header) and means that the variable is switched on. In this case we also speak of the variable being set.

The default initial value, e.g. for the variable declaration in the POU header or in the global variable list = 0 (FALSE). In this case the variable is not set during the PLC program start. If this is not the case, the initial value may also be set to TRUE.

1.4.1.2 INT

Variable values of the data type INTEGER are natural numbers without decimal places. The range of values for INTEGER values is from -32768 to 32767.

The default initial value of a variable of this data type is 0.

Numbers can be entered in decimal, hexadecimal or binary format.

Decimal number	Hexadecimal number	Binary number
1234	16#4D2	2#10011010010
-1234	16#FB2E	2#1111101100101110

1.4.1.3 UINT

Variable values of the data type unsigned INTEGER are numerical numbers without decimal places. The range of values for UINT values is from 0–65535.

1.4.1.4 DINT

Variable values of the data type DOUBLE INTEGER are natural numbers without decimal places. The value range for a DOUBLE INTEGER values is from -2147483648 to 2147483647.

The default initial value of a variable of this data type is 0.

Numbers can be entered in decimal, hexadecimal or binary format.

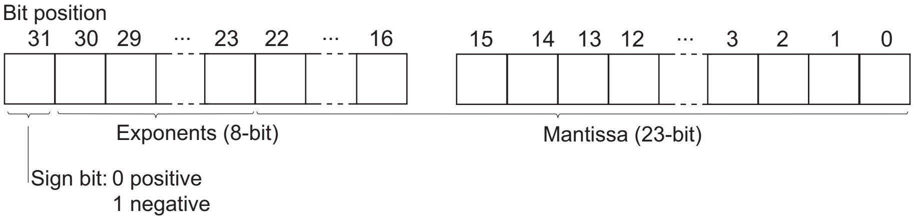
Decimal number	Hexadecimal number	Binary number
123456789	16#75BCD15	2#111010110111100110100010101
-123456789	16#F8A432EB	2#11111000101001000001100101110

1.4.1.5 UDINT

Variable values of the data type unsigned DOUBLE INTEGER are numerical numbers without decimal places. The value range for UDINT values is from 0–4294967295.

1.4.1.6 REAL

Variables of the data type REAL are real 32 bit numbers based on IEEE754. The mantissa is 23 bits and the exponent is 8 bits.



The value range for REAL values is between -3.402823466*E38 to -1.175494351*E-38, 0.0, +1.175494351*E-38 to +3.402823466*E38.

The default for the initial value, e.g. for the variable declaration in the POU header or in the global variable list = 0.0

For FP-e and FP0 only: Do not use REAL instructions in interrupt programs.

You can enter REAL values in the following format:

[+-] Integer.Integer [(Ee) [+-] Integer]

Examples:

5.983e-7

-33.876e12

3.876e3

0.000123

123.0

☞ **The REAL value always has to be entered with a decimal point (e.g. 123.0).**

1.4.1.7 WORD

A variable of the data type WORD consists of 16 binary states. The switching states of 16 in/outputs can be combined as a unity in one word (WORD).

The default initial value of a variable of this data type is 0.

Numbers can be entered in decimal, hexadecimal or binary format.

Decimal number	Hexadecimal number	Binary number
1234	16#4D2	2#10011010010
64302	16#FB2E	2#1111101100101110

1.4.1.8 DWORD

A variable of the data type DOUBLE WORD consists of 32 binary states. The switching states of 32 inputs/outputs can be combined as a unity in one DOUBLE WORD.

The default initial value of a variable of this data type is 0.

Numbers can be entered in decimal, hexadecimal or binary format.

Decimal number	Hexadecimal number	Binary number
123456789	16#75BCD15	2#111010110111100110100010101
4171510507	16#F8A432EB	2#1111100010100100001100101110

1.4.1.9 TIME

TIME (16 Bits): FP3, FP-C, FP5, FP10, FP10S

For variables of the data type TIME (16 bits) you can indicate a duration of 0.01 to 327.67 seconds. The resolution amounts to 10 ms.

TIME (32 Bits): FP-X, FP-Sigma, FP0, FP0R, FP2/2SH, FP10SH

For variables of the data type TIME (32 bits) you can indicate a duration of 0.01 to 21 474 836.47 seconds. The resolution amounts to 10 ms.

Default for 16 and 32 values = T#0 (corresponds to 0 seconds)

☞ ◆**NOTE**

- Duration data must be delimited on the left by the prefix T# or TIME#.
- The units of duration literals can be separated by the character "_".
- Time units, e.g., seconds, milliseconds, etc., can be represented in upper- or lower- case letters.
- "Overflow" of the most significant unit of a duration literal is permitted, e.g., the notation T#25h_15m is permitted.

Description	Examples

Duration literals without underlines:
short prefix
	`T#14ms T#-14ms T#14.7s T#14.7m`
	`T#14.7h T#14.7d t#25h15m`
	`t#5d14h12m18s3.5ms`

long prefix	`TIME#14s TIME#-14s time#14.7s`
Duration literals with underlines: short prefix	`T#25h_15m` `T#5d_14h_12m_18s_3.5ms`
long prefix	`TIME#25h_15m` `time#5d_14h_12m_18s_3.5ms`

1.4.1.10 DATE_AND_TIME

Variable values of the data type DATE_AND_TIME are date and time literals. The range of values for DATE_AND_TIME values is from DT#2001-01-01-00:00:00– DT#2099-12-31-23:59:59.

Description	Examples
Short prefix	DT#2010-06-07-15:36:55 dt#2010-06-07-15:36:55
Long prefix	DATE_AND_TIME#2010-06-07-15:36:55 date_and_time#2010-06-07-15:36:55
Internal representation	Seconds after DT#2001-01-01-00:00:00

Advantages:

- Can be used to set (SET_RTC_DT (see page 294)) or read (GET_RTC_DT (see page 289)) the PLC's real-time clock, for example
- Facilitates all kinds of calculations for date and time
- Well suited for solar tracking applications
- Sun's position, sunrise, sunset
- Conversions between universal time and local time
- Building automation
- Holidays (e.g. Easter holidays), daylight saving time
- Enables better integration and adaptation of POUs created with other manufacturers' IEC 61131-3 programming software, e.g. OSCAT (Open Source Community for Automation Technology)

1.4.1.11 DATE

Variable values of the data type DATE are date literals. The range of values for DATE values is from D#2001-01-01–D#2099-12-31.

Description	Examples
Short prefix	D#2010-06-07 d#2010-06-07
Long prefix	DATE#2010-06-07 date#2010-06-07
Internal representation	Seconds after 2001-01-01

Advantages:

- Facilitates all kinds of calculations for date and time
- Well suited for solar tracking applications
- Sun's position, sunrise, sunset
- Conversions between universal time and local time
- Building automation
- Holidays (e.g. Easter holidays), daylight saving time

1.4.1.12 TIME_OF_DAY

Variable values of the data type TIME_OF_DAY are time of day literals. The range of values for TIME_OF_DAY values is from TOD#00:00:00–TOD#23:59:59.

Description	Examples
Short prefix	TOD#15:36:55 tod#15:36:55
Long prefix	TIME_OF_DAY#15:36:55 time_of_day#15:36:55
Internal representation	Seconds after TOD#00:00:00

Advantages:

- Facilitates all kinds of calculations for date and time
- Well suited for solar tracking applications
- Sun's position, sunrise, sunset
- Conversions between universal time and local time
- Building automation
- Holidays (e.g. Easter holidays), daylight saving time

1.4.1.13 STRING

The data type STRING consists of a series (a string) of up to 32767 ASCII characters. The maximum number of characters depends on the memory size of the PLC. Change the default setting under **Extras** → **Options** → **Compile options** → **Code generation**.

The default initial value, e.g. for variable declarations in the POU header or global variable list, is '', i.e. an empty string.

Declaration

To declare STRING type variables in the POU header use the following syntax:

STRING[n], where **n** = number of characters

The default number of characters for STRING is 32.

Internal memory structure of strings on the PLC

Each character of the string is stored in one byte. A string's memory area consists of a header (two words) and one word for every two characters.

- The first word contains the number of characters reserved for the string.
- The second word contains the actual number of characters in the string.
- Subsequent words contain the ASCII characters (two per word)

To reserve a certain memory area for the string, specify the string length using the following formula: Memory size = 2 words (header) + (n+1)/2 words (characters)

The memory is organized in word units. Therefore, word numbers are always rounded up to the next whole number.

Word x	Number of characters reserved for string	
Word x+1	Actual number of characters in string	
Word x+2	Character 2	Character 1
Word x+3	Character 4	Character 3
Word x+4	Character 6	Character 5
	...	...
Word x+(n+1)/2+1	Character n	Character n-1
	High byte	Low byte

See F159_MTRN (see page 741) for a programming example.

String literals (according to IEC 61131-3)

A character string literal is a sequence of zero or more characters prefixed and terminated by the single quote character (').

Three-character combinations of the dollar sign ($) followed by two hexadecimal digits are to be interpreted as the hexadecimal representation of the eight-bit character code.

Two-character combinations beginning with the dollar sign are to be interpreted as shown in the table:

Combination	Interpretation when printed
$$	Dollar sign ($24)
$'	Single quote ($27)
$L or $l	Line feed ($0A)
$N or $n	New line ($0D$0A)
$P or $p	Form feed (page) ($0C)
$R or $r	Carriage return ($0D)

Combination	Interpretation when printed
$T or $t	Tab ($09)

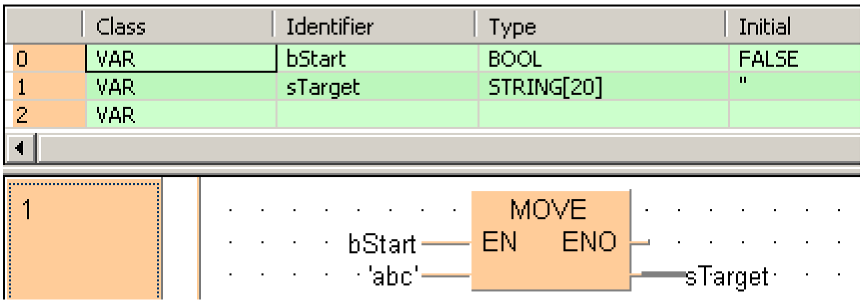
Examples of string literals

Example	Explanation
"	Empty string (length 0)
'A'	String of length 1 containing the single character A
' '	String of length 1 containing the space character
'$"	String of length 1 containing the single quote character
'$R$L'	String of length 2 containing CR and LF characters
'$$1.00'	String of length 5 which would print as "$1.00"
'$02$03'	String of length 2 containing STX and ETX characters

Strings as constants

It is possible to enter values of the data type STRING directly as constants into a function or a function block. The string must be enclosed in single quotes.

Transfer a constant character string 'abc' to the string variable sTarget.



	Class	Identifier	Type	Initial
0	VAR	bStart	BOOL	FALSE
1	VAR	sTarget	STRING[20]	"
2	VAR			

Transferring strings to functions or function blocks

When character strings are transferred, only as many characters that fit into the target string are transferred. Please refer to the following examples in the online help under the keyword 'STRING':

1. Copy a source string to a target string which is shorter.
2. Copy a constant character string to another which is shorter.
3. Generate a message using a string function.

◆ **NOTE**

The conversion functions INT_TO_STRING (see page 217), DINT_TO_STRING (see page 220), REAL_TO_STRING (see page 228), TIME_TO_STRING (see page 230), etc. need many system resources in terms of programming steps and processing time. When you use these functions frequently, create a user-defined function that embeds the conversion function and use the user-defined function in your project. For older PLC types (FP0, FP3, FP5, FP10), this is also true for the CONCAT (see page 269) and FIND (see page 273) instructions.

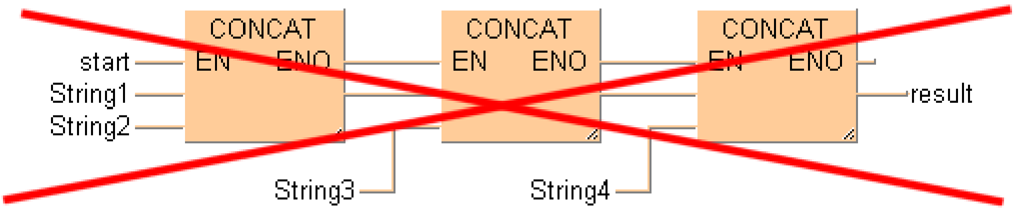
STRING with EN/ENO

Ladder diagram (LD) and function block diagram (FBD)

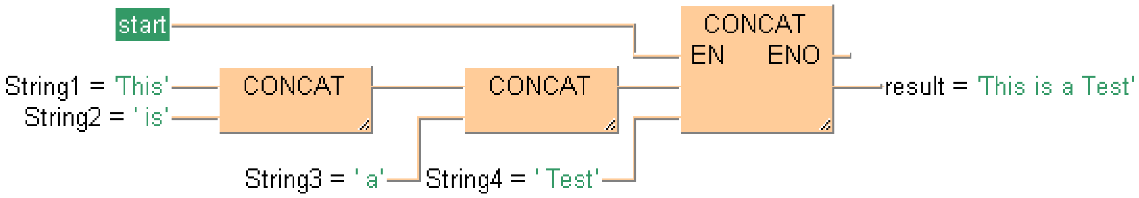
STRING instructions with enable input (EN) and enable output (ENO) contacts may NOT be connected to each other in LD and FBD. First connect the STRING instructions without EN/ENO and then add an instruction with EN/ENO in the final position. The enable input (EN) then controls the output of the overall result.

◆ **EXAMPLE**

This arrangement is not possible:



This arrangement is possible:

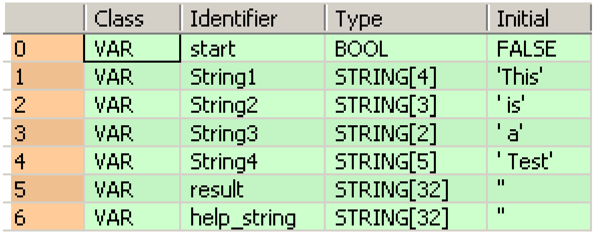


Instruction list (IL)

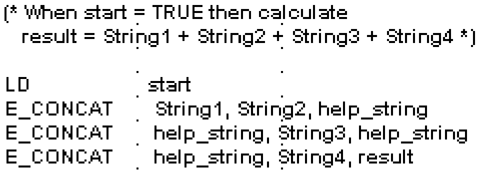
STRING instructions with EN/ENO may be connected to each other in IL. Nevertheless, in order to avoid intermediate variables, it is recommended that you use a conditional jump instead of connecting a series of functions with EN/ENO.

◆ **EXAMPLE**

Program with dummy string

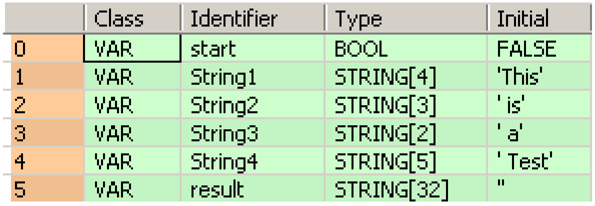


	Class	Identifier	Type	Initial
0	VAR	start	BOOL	FALSE
1	VAR	String1	STRING[4]	'This'
2	VAR	String2	STRING[3]	' is'
3	VAR	String3	STRING[2]	' a'
4	VAR	String4	STRING[5]	' Test'
5	VAR	result	STRING[32]	''
6	VAR	help_string	STRING[32]	''

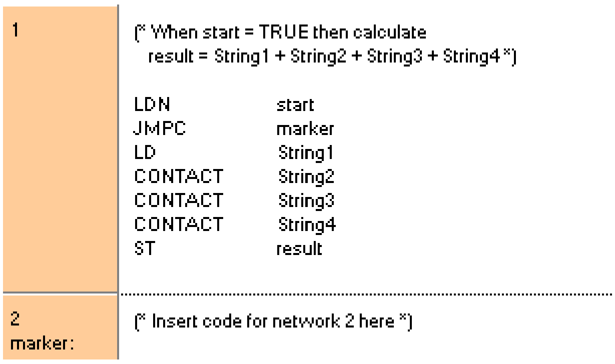


```
(* When start = TRUE then calculate
   result = String1 + String2 + String3 + String4 *)

LD          start
E_CONCAT    String1, String2, help_string
E_CONCAT    help_string, String3, help_string
E_CONCAT    help_string, String4, result
```

Program with conditional jump

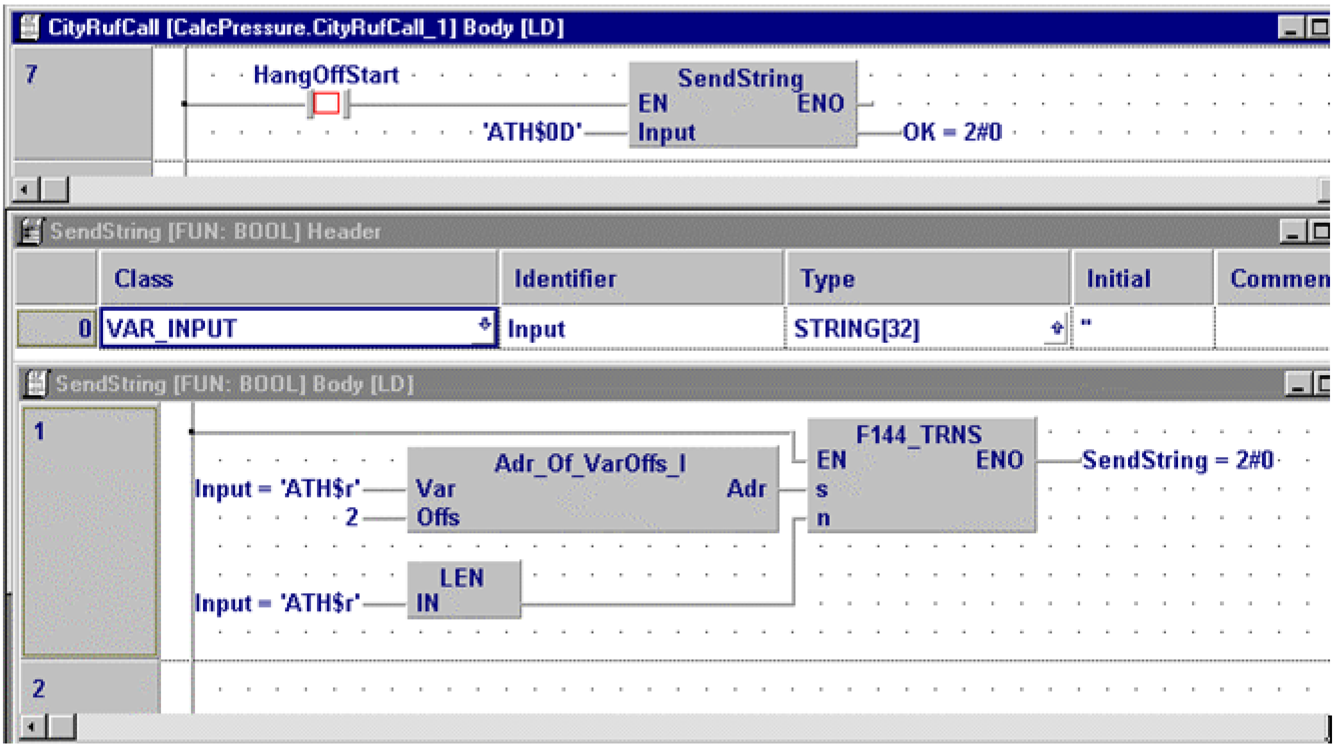


	Class	Identifier	Type	Initial
0	VAR	start	BOOL	FALSE
1	VAR	String1	STRING[4]	'This'
2	VAR	String2	STRING[3]	' is'
3	VAR	String3	STRING[2]	' a'
4	VAR	String4	STRING[5]	' Test'
5	VAR	result	STRING[32]	''



```
1          (* When start = TRUE then calculate
             result = String1 + String2 + String3 + String4 *)

           LDN          start
           JMPC         marker
           LD           String1
           CONTACT      String2
           CONTACT      String3
           CONTACT      String4
           ST           result

2          (* Insert code for network 2 here *)
marker:
```

The difficulty of programming with a dummy string lies in correctly choosing its length. When connecting unconditional string instructions in series, the length is calculated automatically.



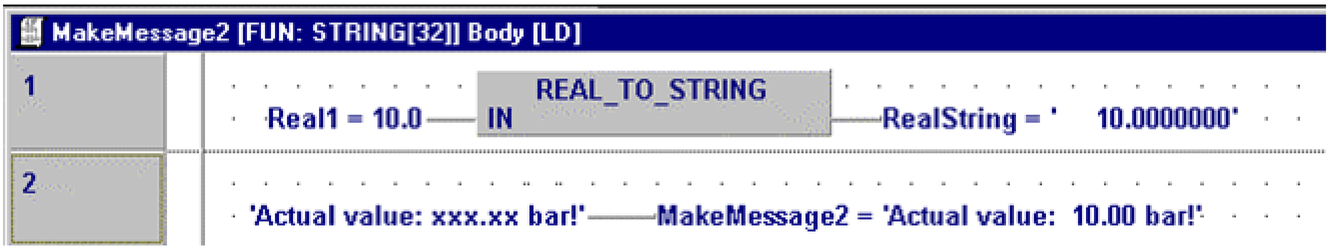
CityRufCall [CalcPressure.CityRufCall_1] Body [LD]

SendString [FUN: BOOL] Header

	Class	Identifier	Type	Initial	Commen
0	VAR_INPUT	Input	STRING[32]	''	

SendString [FUN: BOOL] Body [LD]

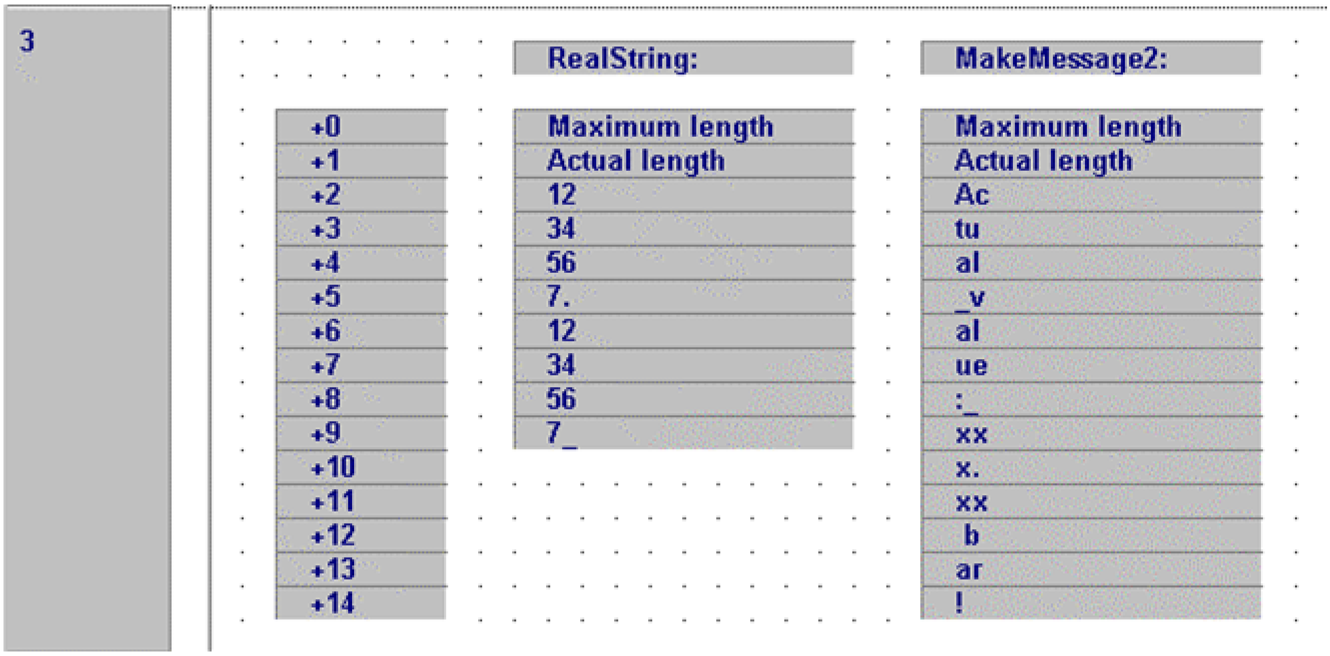
2. MakeMessage2



MakeMessage2 [FUN: STRING[32]] Body [LD]

1
REAL_TO_STRING
Real1 = 10.0
IN
RealString = '   10.0000000'
2
'Actual value: xxx.xx bar!'
MakeMessage2 = 'Actual value: 10.00 bar!'

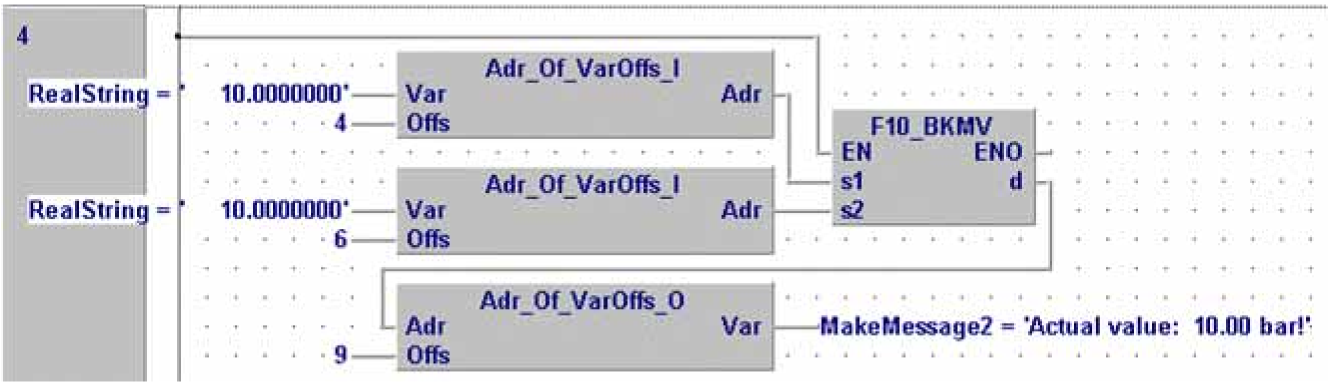
From these commands one gets the following address occupation:



3

	RealString:	MakeMessage2:
+0	Maximum length	Maximum length
+1	Actual length	Actual length
+2	12	Ac
+3	34	tu
+4	56	al
+5	7.	_v
+6	12	al
+7	34	ue
+8	56	:_
+9	7_	xx
+10		x.
+11		xx
+12		b
+13		ar
+14		!

Another use with functions from the FP TOOL Library (Adr_OfVarOffs....):



1.4.2 Generic data types

Generic data types are used internally by system functions and function blocks and cannot be selected in user-defined POUs. Generic data types are identified by the prefix ANY.

◆NOTE

Generic data types are not available in user-defined POUs.

Hierarchy of generic data types

ANY

ANY_NUM	REAL, ANY_INT
ANY_INT	INT, DINT UINT, UDINT
ANY16	WORD INT, UINT
ANY32	DWORD DINT, UDINT DATE, TOD, DT
ANY_BIT	BOOL WORD, DWORD
ANY_DATE	DATE, TOD, DT

1.4.3 DUT

A DUT (data unit type) is composed of several elementary data types which may differ in type.

1.4.3.1 Creating DUTs

1. **Object → New → DUT or**
2. **Enter DUT name**

   If desired, select the check box for DUTs with overlapping elements (see page 53).
3. **[OK]**
4. **Open the new DUT from the "Project" pane**
5. **Declare variables for the DUT**
6. **Object → Check or**

1.4.3.2 Using DUTs in the global variable list

1. **Open "Global variables" from "Project" pane**
2. **Enter a new line with or , if necessary**
3. **Under "Class", select "VAR_GLOBAL"**
4. **Under "Identifier", enter a symbolic name**
5. **Enter FP address or IEC address**

   The first element of the DUT determines the address type: for BOOL type elements, assign a 1-bit address (e.g. R10), for other data types assign a 16-bit address (e.g. WR1). If you assign an address, DUTs with non-overlapping elements must consist entirely of BOOL type elements, or entirely of non-BOOL type elements.
6. **Under "Type", select to open the "Type selection" dialog**
7. **Under "Type Class", select "Data Unit Types"**
8. **Under "Type", select the desired DUT**
9. **[OK]**
10. **Under "Initial", select to open the "Data Unit Initial Values" dialog**

    This dialog shows how the individual variables have been defined in the DUT. You can only change the initial values for one single variable (not for the DUT).
11. **Change initial value for the desired variable, if necessary**
12. **[OK]**
13. **Under "Comment", enter a text, if desired**

14. Object → Save

☞ ◆**NOTE**

A DUT defined in the global variable list can be used in a POU body only when copied into the header of the corresponding POU beforehand.

1.4.3.3 Using DUTs in a POU header

1. **Open POU header from "Project" pane**
2. **Enter a new line with or , if necessary**
3. **Under Class, select "VAR"**
4. **Under "Identifier", enter a symbolic name**
5. **Under "Type", select to open the "Type selection" dialog**
6. **Under "Type Class", select "Data Unit Types"**
7. **Under "Type", select the desired DUT**
8. **[OK]**
9. **Under "Initial", select to open the "Data Unit Initial Values" dialog**

   This dialog shows how the individual variables have been defined in the DUT. You can only change the initial values for one single variable (not for the DUT).
10. **Change initial value for the desired variable, if necessary**
11. **[OK]**
12. **Under "Comment", enter a text, if desired**
13. **Object → Save**

    Now the DUT or a single variable of the DUT can be used in the POU body. The DUT can be assigned with the help of the "Variables" pane (**<F2>)**.

☞ ◆**NOTE**

A DUT defined in the global variable list can be used in a POU body only when copied into the header of the corresponding POU beforehand.

1.4.3.4 DUTs with non-overlapping elements

Using the Properties dialog, you can assign a DUT one of two ways of occupying memory:

1. with overlapping elements (see page 53)
2. with non-overlapping elements

How DUTs with non-overlapping elements occupy memory:

All elements of the data type BOOL are lumped together in a block and allocated one after the other in a memory area reserved for bits, beginning at a 16-bit word address.

All elements of the data type ARRAY OF BOOL are lumped together in a block and allocated in a memory area reserved for bits, beginning at a 16-bit word address.

All other elements are lumped together and allocated one after the other in a block in a memory area reserved for 16-bit words.

For details on working with DUTs and predefined system DUTs, please refer to the online help.

1.4.3.5 DUTs with overlapping elements

How DUTs with overlapping elements occupy memory:

All elements of the same data type (BOOL, WORD, INT, DWORD, DINT, REAL and STRINGs with the same, common string length) are each lumped together and allocated one after the other beginning from a common starting address. Arrays are also allocated to this common starting address.

The following conditions apply to this starting address: If DUT consists of BOOL or ARRAY OF BOOL type elements, it is stored in a memory area reserved for bits; otherwise it is stored in a memory area reserved for 16-bit words.

To avoid ambiguity during initialization no initialization is allowed. The following default initializations are executed:

- BOOL: FALSE
- WORD, INT, DWORD, DINT: 0
- REAL: 0.0
- STRING: '' (i.e. the address occupied by the maximum string length is initialized with the maximum length of the string that is greater or equal to zero. The rest of the string is initialized with zeros.)

Also, all element variables of the data type STRING must be located at the end of the declaration.

◆**NOTE**

- In general, you should pay exact attention to how memory area is occupied by the data types used.
- Especially when using STRINGs, note that their particular way of occupying memory allows them to be repeatedly overwritten with the help of other elements.
- Ensure the maximum string length and the current string length are valid values before using string commands.

For details on working with DUTs and predefined system DUTs, please refer to the online help.

1.4.4 Array

Arrays

An array is a group of variables which all have the **same** elementary data type and that are grouped together, one after the other, in a continuous data block. This variable group itself is a variable and must hence be declared for this reason. In the program you can either use the whole array or individual array elements.

Declaration

To declare ARRAY type variables in the POU header use the following syntax:

ARRAY[A...B,C...D,E...F] OF <data type> where:

A=	first element index	first dimension
B=	last element index	
C=	first element index	second dimension (optional)
D=	last element index	
E=	first element index	third dimension (optional)
F=	last element index	

Arrays can be 1, 2 or 3-dimensional. In each dimension, an array can have several fields. Element indexes are positive or negative integers. The first element must be smaller than the last element.

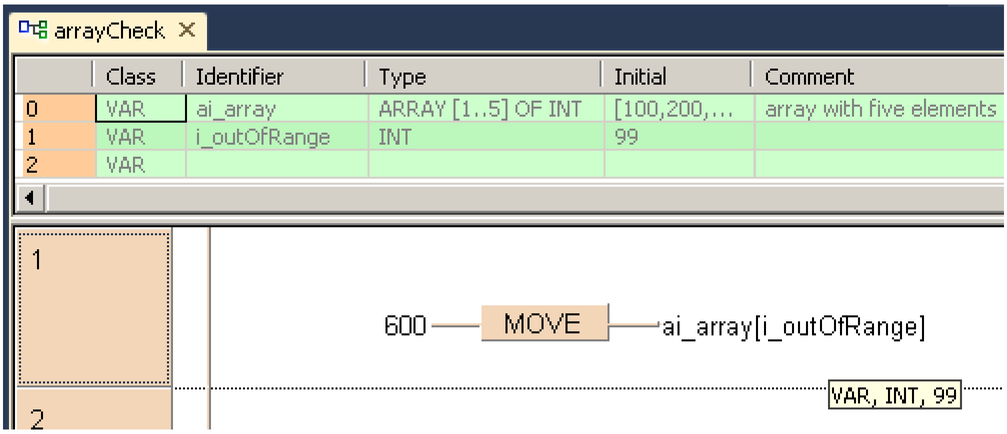
◆**NOTE**

An array cannot be used as a variable by another array.

When accessing an index of an array, Control FPWIN Pro does not check the index against the bounds of the array. Make sure the index lies within the range defined in the POU header.

Example: ARRAY [1..5] OF INT

In this example, ai_array[99] is out of range but does not produce an error message.



arrayCheck

	Class	Identifier	Type	Initial	Comment
0	VAR	ai_array	ARRAY [1..5] OF INT	[100,200,...	array with five elements
1	VAR	i_outOfRange	INT	99	
2	VAR				

Data types valid for arrays are:

- BOOL
- DATE
- DATE_AND_TIME
- DINT
- DWORD
- INT
- REAL
- STRING
- TIME

- TIME_OF_DAY
- UDINT
- UINT
- WORD

Data Unit Type

A **D**ata **U**nit **T**ype (DUT) is a group of variables composed of several **different** elementary data types (BOOL, WORD etc.). These groups are used when tables are edited, such as for data table control, e.g. F174_PulseOutput_DataTable (see page 1069). Define a DUT in the DUT pool first. Then you can use the DUT in the "Type" field of the global variable list or of a POU header similarly to the integer, BOOL etc. data types. In the program you can then use either the whole DUT or individual variables of the DUT.

☞ ◆**NOTE**

A DUT cannot be used as a variable by another DUT.

For details on working with ARRAYs or DUTs, please refer to the online help.

1.4.5 Special data types only available in conversion functions

☞ ◆**NOTE**

- Valid data types are: BOOL16, BOOL32, BOOLS, SDT, SDDT, BCD, IPADDR, ETLANADDR
- These data types are valid for conversion functions to special data types (see page 1335) only.
- These data types cannot be declared in POU headers.

1.4.5.1 BOOL16

Allowed are:

- Arrays with exactly 16 elements of the data type BOOL
  Note:
  These types can lie in the areas X, Y, R, L, T, and C. For failure to make an assignment in the address field of the global variable list or for local variables, they are automatically placed in area R by the compiler.

- All DUTs with exactly 16 members of the data type BOOL
  Note:
  These are automatically placed by the compiler in area R.

1.4.5.2 BOOL32

Allowed are:

- Arrays with exactly 32 elements of the data type BOOL
  Note:
  These types can lie in the areas X, Y, R, L, T, and C. For failure to make an assignment in the address field of the global variable list or for local variables, they are automatically placed in area R

by the compiler.

- All DUTs with exactly 32 members of the data type BOOL
  Note:
  These are automatically placed by the compiler in area R.

1.4.5.3 BCD_WORD

The data type BCD_WORD (binary-coded decimal) only occurs in the conversion functions INT_TO_BCD_WORD (see page 243) and UINT_TO_BCD_WORD (see page 245). These conversion functions use variables of the type WORD, which are interpreted as BCD numbers, e.g. the decimal number 654 is interpreted as the hexadecimal number 16#0654.

1.4.5.4 WORD_BCD

The data type WORD_BCD (binary-coded decimal) only occurs in the conversion functions WORD_BCD_TO_INT (see page 146) and WORD_BCD_TO_UINT (see page 158). These conversion functions use variables of the type WORD, which are interpreted as BCD numbers, e.g. the decimal number 654 is interpreted as the hexadecimal number 16#0654.

1.4.5.5 BCD_DWORD

The data type BCD_DWORD (binary-coded decimal) only occurs in the conversion functions DINT_TO_BCD_DWORD (see page 244) and UDINT_TO_BCD_DWORD (see page 246). These conversion functions use variables of the type DWORD, which are interpreted as BCD numbers, e.g. the decimal number 654 is interpreted as the hexadecimal number 16#0654.

1.4.5.6 DWORD_BCD

The data type DWORD_BCD (binary-coded decimal) only occurs in the conversion functions DWORD_BCD_TO_DINT (see page 169) and DWORD_BCD_TO_UDINT (see page 182). These conversion functions use variables of the type DWORD, which are interpreted as BCD numbers, e.g. the decimal number 654 is interpreted as the hexadecimal number 16#0654.

1.4.5.7 IPADDR

The data type IPADDR only occurs in the following conversion functions:

- IPADDR_TO_STRING (see page 232)
- IPADDR_TO_STRING_NO_LEADING_ZEROS (see page 233)
- STRING_TO_IPADDR (see page 247)
- STRING_TO_IPADDR_STEPSAVER (see page 248)

These conversion functions interpret variables of the type DWORD as strings in IPADDR format. This format consists of four octal numbers (with or without leading zeros) separated by periods in opposite order, i.e. the highest octal number in the IPADDR number will be the lowest octal number in the string.

Example:

Value	Conversion function	Result
16#01020304	IPADDR_TO_STRING	004.003.002.001
	IPADDR_TO_STRING_NO _LEADING_ZEROS	4.3.2.1

☞ ◆NOTE

If you want an interpretation of the DWORD in direct order, use the conversion functions that invoke the data type ETLANADDR.

1.4.5.8 ETLANADDR

The data type ETLANADDR only occurs in the following conversion functions:

- ETLANADDR_TO_STRING (see page 234)
- ETLANADDR_TO_STRING_NO_LEADING_ZEROS (see page 235)
- STRING_TO_ETLANADDR (see page 249)
- STRING_TO_ETLANADDR_STEPSAVER

These conversion functions interpret variables of the type DWORD as strings in ETLANADDR format. This format consists of four octal numbers (with or without leading zeros) seperated by periods in direct order, i.e. the highest octal number in the ETLANADDR number will be the highest octal number in the string.

Example:

Value	Conversion function	Result
16#01020304	ETLANADDR_TO_STRING	001.002.003.004
	ETLANADDR_TO_STRING_ NO_LEADING_ZEROS	1.2.3.4

☞ ◆NOTE

If you want an interpretation of the DWORD in inverse order, use the conversion functions invoking the data type IPADDR.

1.4.5.9 ANY_IN_UNITS_OF_WORDS

Allowed are:

- Data types INT, DINT, WORD, DWORD, REAL, STRING, TIME
- Arrays with data types other than BOOL
- All DUTs that contain elements with data types besides BOOL

  Note:
  These data types can lie in the following areas: WX, DWX, WY, DWY, WR, DWR, WL, DWL, SV, DSV, EV, DEV, DT, DDT, LD, DLD, FL, DFL. For failure to make an assignment in the address field

of the global variable list or for local variables, they are automatically placed in DT, DDT, FL or DFL by the compiler.

- Arrays with the data type BOOL under the condition that the total number of elements can be divided by 16.

  Note:
  These types can lie in the areas X, Y, R, L, T, and C. For failure to make an assignment in the address field of the global variable list or for local variables, they are automatically placed in R by the compiler.

- All DUTs with a number of simple BOOL variables divisible by 16 remain.

  Note:
  These are automatically placed by the compiler in area R.

1.4.5.10 ANY_SIMPLE_NOT_BOOL

Allowed are:

Data types INT, DINT, WORD, DWORD, REAL, STRING, TIME (but not BOOL)

These data types can lie in the following areas:
WX, DWX, WY, DWY, WR, DWR, WL, DWL, SV, DSV, EV, DEV, DT, DDT, LD, DLD, FL, DFL.
For failure to make an assignment in the address field of the global variable list or for local variables, they are automatically placed in DT, DDT, FL or DFL by the compiler.

Chapter 2

Data transfer instructions

MOVE

Move value to specified destination

Description MOVE assigns the unchanged value of the input variable to the output variable.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** MOVE **(see page 1328)**

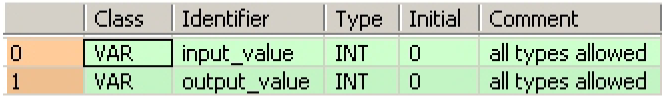
- **When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.**
- **For further information refer to the online help: Upgrade Problems with Data Type STRING**

Data types

Data type	I/O	Function
all data types	input	source
all data types	output as input	destination

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

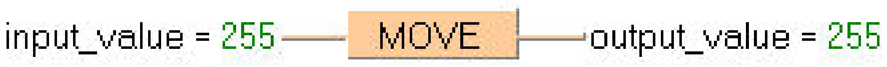


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	INT	0	all types allowed
1	VAR	output_value	INT	0	all types allowed

In this example the input variable **input_value** has been declared. Instead, you may enter a constant directly at the input contact of a function.

Body **Input_value** is assigned to **output_value** without being modified.

LD



ST When programming with structured text, enter the following:

```
output_value:= input_value;
```

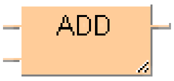
Chapter 3

Arithmetic instructions

ADD

Add

Description This function adds the input variables IN1 + IN2 +... and writes the addition result into the output variable.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of ADD (see page 1335)**

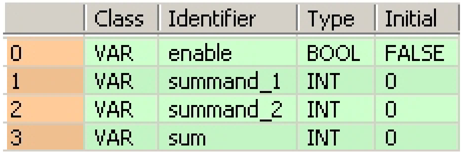
- **All operands must be of the same data type.**
- **This function can be expanded to a maximum of 28 input contacts.**

Data types

Data type	I/O	Function
INT, DINT, REAL	1st input	augend
INT, DINT, REAL	2nd input	addend
INT, DINT, REAL	output as input	sum

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables which are required for programming the function are declared in the POU header.

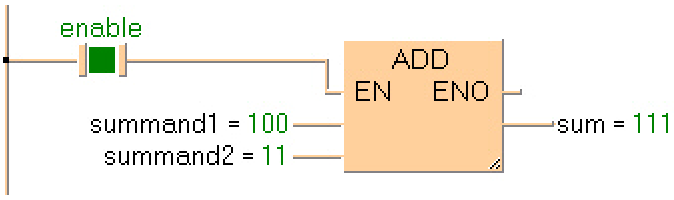


	Class	Identifier	Type	Initial
0	VAR	enable	BOOL	FALSE
1	VAR	summand_1	INT	0
2	VAR	summand_2	INT	0
3	VAR	sum	INT	0

In this example the input variables (**summand_1, summand_2** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body If **enable** is set (TRUE), **summand_1** is added to **summand_2**. The result is written into **sum**.

LD



SUB

Subtract

Description The content of the accumulator is subtracted from the operand defined in the operand field.The result is transferred to the accumulator.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of SUB (see page 1335)**

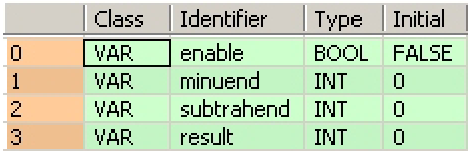
- **All operands must be of the same data type.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

Data types

Data type	I/O	Function
INT, DINT, REAL	1st input	minuend
INT, DINT, REAL	2nd input	subtrahend
INT, DINT, REAL	output as input	result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables which are required for programming the function are declared in the POU header.

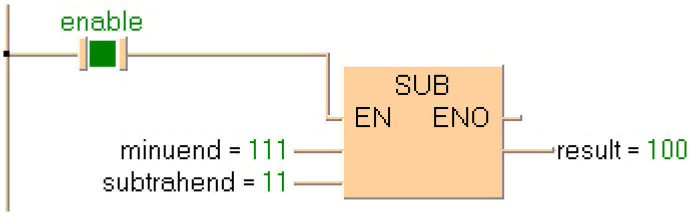


	Class	Identifier	Type	Initial
0	VAR	enable	BOOL	FALSE
1	VAR	minuend	INT	0
2	VAR	subtrahend	INT	0
3	VAR	result	INT	0

In this example the input variables (**minuend, subtrahend** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body If **enable** is set, s**ubtrahend** (data type INT) is subracted from **minuend**. The result will be written into **result** (data type INT).

LD



MUL

Multiply

Description MUL multiplies the values of the input variables with each other and writes the result into the output variable.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types ☞ **Availability of MUL (see page 1335)**

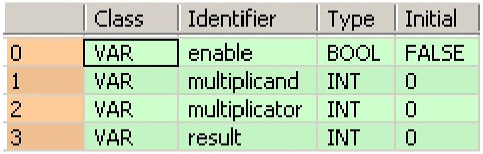
- **All operands must be of the same data type.**
- **This function can be expanded to a maximum of 28 input contacts.**
- **Modifying elements**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

Data types

Data type	I/O	Function
INT, DINT, REAL	1st input	multiplicand
INT, DINT, REAL	2nd input	multiplicator
INT, DINT, REAL	output as input	result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables which are required for programming the function are declared in the POU header.

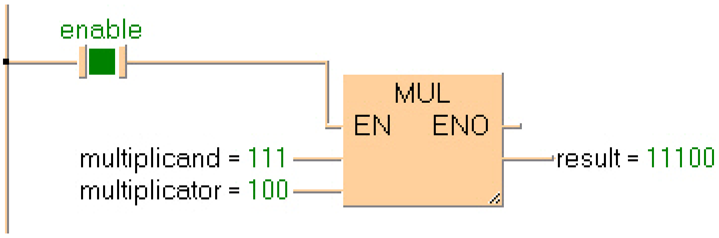


	Class	Identifier	Type	Initial
0	VAR	enable	BOOL	FALSE
1	VAR	multiplicand	INT	0
2	VAR	multiplicator	INT	0
3	VAR	result	INT	0

In this example the input variables (**multiplicand, multiplicator** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body If **enable** is set (TRUE), the **multiplicant** is multiplied with the **multiplicator**. The result will be written into **result**.

LD



DIV

Divide

Description DIV divides the value of the first input variable by the value of the second.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of DIV (see page 1335)**

☞

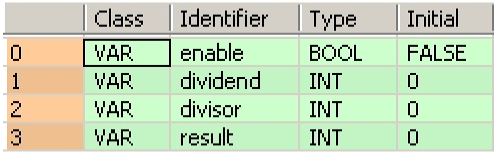
- **Input and output variables must be of one of the noted data types. All operands must be of the same data type.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

Data types

Data type	I/O	Function
INT, DINT, REAL	1st input	dividend
INT, DINT, REAL	2nd input	divisor
INT, DINT, REAL	output as input	result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables which are required for programming the function are declared in the POU header.

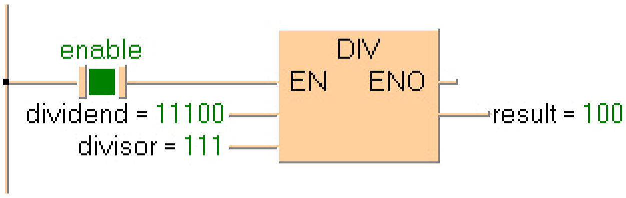


	Class	Identifier	Type	Initial
0	VAR	enable	BOOL	FALSE
1	VAR	dividend	INT	0
2	VAR	divisor	INT	0
3	VAR	result	INT	0

In this example the input variables (**dividend, divisor** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body If **enable** is set (TRUE), **dividend** is divided by **divisor**. The result is written into **result**.

LD



ABS

Absolute Value

Description ABS calculates the value in the accumulator into an absolute value. The result is saved in the output variable.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

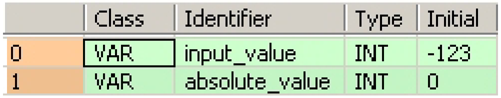
PLC types **Availability of ABS (see page 1318)**

Data types

Data type	I/O	Function
INT, DINT, REAL	input	input data type
INT, DINT, REAL	output as input	absolute value

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

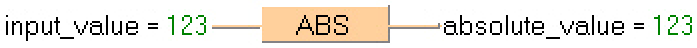


	Class	Identifier	Type	Initial
0	VAR	input_value	INT	-123
1	VAR	absolute_value	INT	0

This example uses variables. You can also use a constant for the input variable.

Body **Input_value** of the data type INTEGER is converted into an absolute value of the data type INTEGER. The converted value is written into **absolute_value**.

LD



ST When programming with structured text, enter the following:

```
absolute_value:=ABS(input_value);
```

MOD

Modular arithmetic division, remainder stored in output variable

Description MOD divides the value of the first input variable by the value of the second. The rest of the integral division (5 : 2 : 2 + rest = 1) is written into the output variable.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

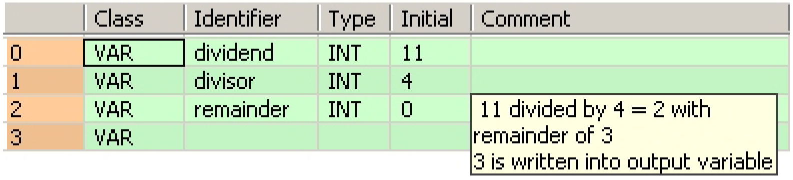
PLC types **Availability of MOD (see page 1328)**

Data types

Data type	I/O	Function
INT, DINT	1st input	dividend
INT, DINT	2nd input	divisor
INT, DINT	output as input	remainder

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

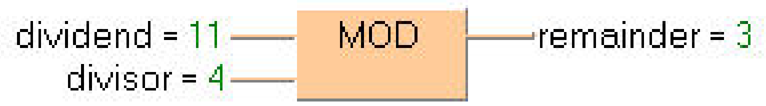
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	dividend	INT	11	
1	VAR	divisor	INT	4	
2	VAR	remainder	INT	0	11 divided by 4 = 2 with remainder of 3 3 is written into output variable
3	VAR				

Body This example uses variables. You may also use constants for the input variables. Dividend (11) is divided by divisor (4). The remainder (3) of the division is written in **remainder**.

LD



ST When programming with structured text, enter the following:

```
remainder:= dividend MOD divisor;
```

SQRT

Square root

Description SQRT calculates the square root of an input variable of the data type REAL (value ≥ 0.0). The result is written into the output variable.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** SQRT **(see page 1331)**

☞ **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

Data types

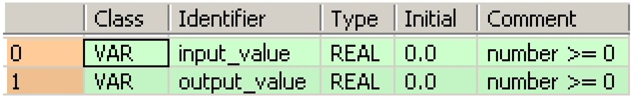
Data type	I/O	Function
REAL	input	input value
REAL	output as input	square root of input value

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input variable does not have the data type REAL or input variable is not ≥ 0.0
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	▪ output variable is zero
R9009	%MX0.900.9	for an instant	▪ processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

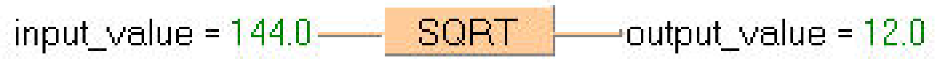


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	number >= 0
1	VAR	output_value	REAL	0.0	number >= 0

This example uses variables. You can also use a constant for the input variable.

Body The square root of **input_value** is calculated and written into **output_value**.

LD



ST When programming with structured text, enter the following:

```
output_value:= SQRT(input_value);
```

SIN

Sine with Radian Input Data

Description SIN calculates the sine of the input variable and writes the result into the output variable. The angle data has to be specified in radians (value < 52707176).

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

- **The accuracy of the calculation decreases as the angle data specified in the input variable increases. Therefore, we recommend to enter angle data in radians ≥ -2π and ≤ 2π.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

PLC types **Availability of SIN (see page 1330)**

Data types

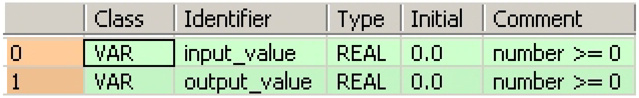
Data type	I/O	Function
REAL	input	input value, angle data in radians
REAL	output as input	SINE of input value

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input variable does not have the data type REAL or input variable ≥ 52707176
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	▪ output variable is zero
R9009	%MX0.900.9	for an instant	▪ processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables which are required for programming the function are declared in the POU header.

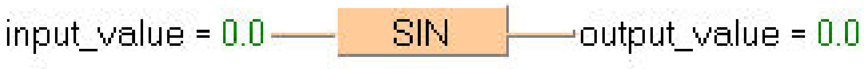


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	number >= 0
1	VAR	output_value	REAL	0.0	number >= 0

This example uses variables. You can also use a constant for the input variable.

Body The sine of **input_value** is calculated and written into **output_value**.

LD



ASIN

Arcsine

Description ASIN calculates the arcsine of the input variable and writes the angle data in radians into the output variable. The function returns a value from - π/2 to π/2.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

☞ **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

PLC types **Availability of** ASIN **(see page 1318)**

Data types

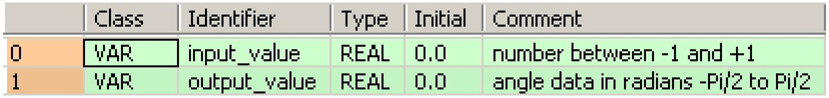
Data type	I/O	Function
REAL	input	input value between -1 and +1
REAL	output as input	arcsine of input value in radians

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input variable does not have the data type REAL or input variable is not ≥ -1.0 and ≤ 1.0
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	▪ output variable is zero
R9009	%MX0.900.9	for an instant	▪ processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

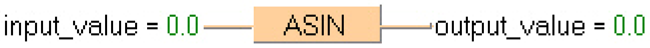


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	number between -1 and +1
1	VAR	output_value	REAL	0.0	angle data in radians -Pi/2 to Pi/2

This example uses variables. You can also use a constant for the input variable.

Body The arc sine of **input_value** is calculated and written into **output_value**.

LD



ST When programming with structured text, enter the following:

```
output_value:=ASIN(input_value);
```

COS

Cosine

Description COS calculates the cosine of the input variable and writes the result into the output variable. The angle data has to be specified in radians (value < 52707176).

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

☞

- **The accuracy of the calculation decreases as the angle data specified in the input variable increases. Therefore, we recommend to enter angle data in radians ≥ -2π and ≤ 2π.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

PLC types **Availability of COS (see page 1318)**

Data types

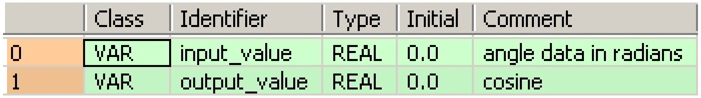
Data type	I/O	Function
REAL	input	input value, angle data in radians
REAL	output as input	cosine of input value

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input variable does not have the data type REAL or input variable ≥ 52707176
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	▪ output variable is zero
R9009	%MX0.900.9	for an instant	▪ processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

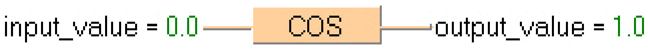


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	angle data in radians
1	VAR	output_value	REAL	0.0	cosine

This example uses variables. You can also use a constant for the input variable.

Body The cosine of **input_value** is calculated and written into **output_value**.

LD



ST When programming with structured text, enter the following:

```
output_value:=COS(input_value);
```

ACOS

Arccosine

Description ACOS calculates the arccosine of the input variable and writes the angle data in radians into the output variable. The function returns a value from 0.0 to π.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.

PLC types **Availability of** ACOS **(see page 1318)**

Data types

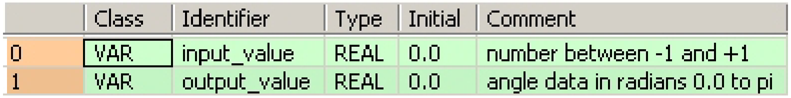
Data type	I/O	Function
REAL	input	input value between -1 and +1
REAL	output as input	arccosine of input value in radians

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input variable does not have the data type REAL or input variable is not ≥ -1.0 and ≤ 1.0
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	▪ output variable is zero
R9009	%MX0.900.9	for an instant	▪ processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

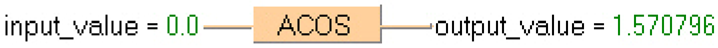


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	number between -1 and +1
1	VAR	output_value	REAL	0.0	angle data in radians 0.0 to pi

This example uses variables. You can also use a constant for the input variable.

Body The arc cosine of **input_value** is calculated and written into **output_value**.

LD



input_value = 0.0 — ACOS — output_value = 1.570796

ST

When programming with structured text, enter the following:

```
output_value:=ACOS(input_value);
```

TAN

Tangent

Description TAN calculates the tangent of the input variable and writes the result into the output variable. The angle data has to be specified in radians (value < 52707176).

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

☞

- **The accuracy of the calculation decreases as the angle data specified in the input variable increases. Therefore, we recommend to enter angle data in radians -2π and 2π.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

PLC types **Availability of** TAN **(see page 1331)**

Data types

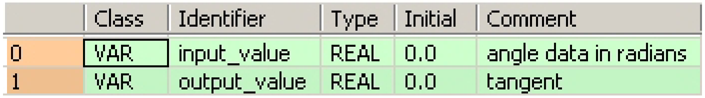
Data type	I/O	Function
REAL	input	input value in radians
REAL	output as input	tangent of input value

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input variable does not have the data type REAL or input variable ≥ 52707176
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	▪ output variable is zero
R9009	%MX0.900.9	for an instant	▪ processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables which are required for programming the function are declared in the POU header.

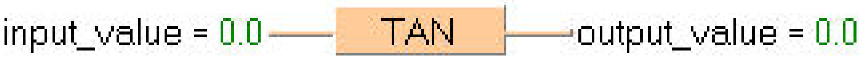


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	angle data in radians
1	VAR	output_value	REAL	0.0	tangent

This example uses variables. You can also use a constant for the input variable.

Body The tangent of **input_value** is calculated and written into **output_value**.

LD



ATAN

Arctangent

Description ATAN calculates the arctangent of the input variable (value ± 52707176) and writes the angle data in radians into the output variable. The function returns a value greater than -π/2 and smaller than π/2.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

☞ **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

PLC types **Availability of** ATAN **(see page 1318)**

Data types

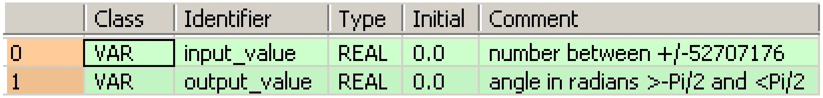
Data type	I/O	Function
REAL	input	input value between -52707176 and +52707176
REAL	output as input	arctangent of input value in radians

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input variable does not have the data type REAL or input variable ≥ 52707176
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	▪ output variable is zero
R9009	%MX0.900.9	for an instant	▪ processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

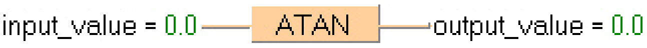


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	number between +/-52707176
1	VAR	output_value	REAL	0.0	angle in radians >-Pi/2 and <Pi/2

This example uses variables. You can also use a constant for the input variable.

Body The arc tangent of **input_value** is calculated and written into **output_value**.

LD



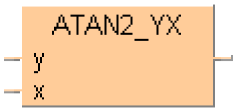
ST When programming with structured text, enter the following:

```
output_value:=ATAN(input_value);
```

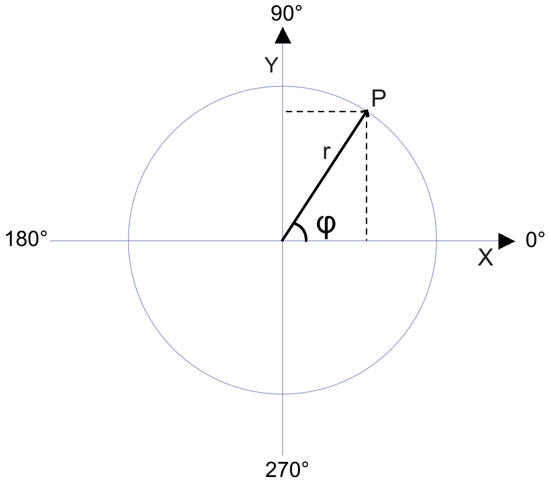
ATAN2_YX

Returns the angle φ of the Cartesian coordinates (x,y)

Description ATAN2_YX returns the angle φ of the Cartesian coordinates (x,y) within the range of -π to +π.



Each position **P** of the two-dimensional coordinates can be defined by Cartesian coordinates P(x,y) or by polar coordinates P(r,φ) (r = radius, φ = angle).



Define ATAN2_YX as follows:

ATAN2_YX(y,x)	x	y
$\arctan\frac{y}{x}$	x > 0	
$\arctan\frac{y}{x}+\pi$	x < 0	y ≥ 0
$\arctan\frac{y}{x}-\pi$		y < 0
$+\frac{\pi}{2}$	x = 0	y > 0
$-\frac{\pi}{2}$		y < 0
0		y = 0

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

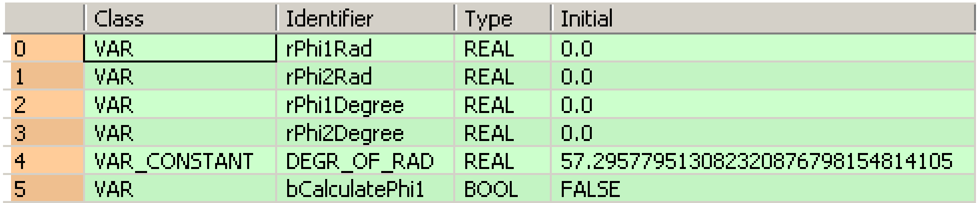
PLC types **Availability of** ATAN2_YX **(see page 1318)**

Data types

Data type	I/O	Function
REAL	y	Cartesian y coordinate
REAL	x	Cartesian x coordinate

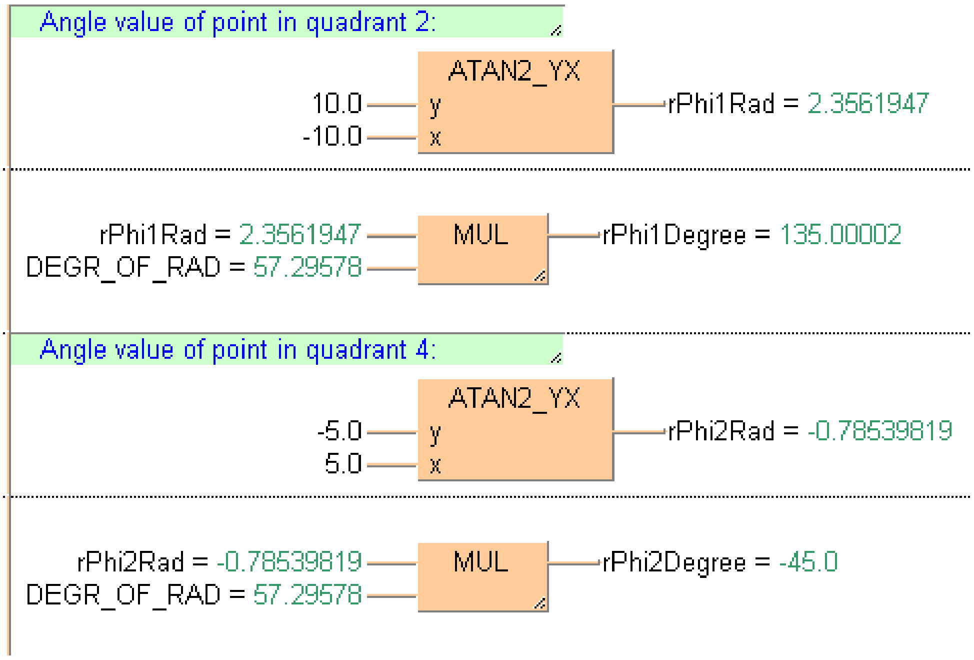
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	rPhi1Rad	REAL	0.0
1	VAR	rPhi2Rad	REAL	0.0
2	VAR	rPhi1Degree	REAL	0.0
3	VAR	rPhi2Degree	REAL	0.0
4	VAR_CONSTANT	DEGR_OF_RAD	REAL	57.295779513082320876798154814105
5	VAR	bCalculatePhi1	BOOL	FALSE

LD



ST When programming with structured text, enter the following:

```
(* Angle value of point in quadrant 2 *)
rPhi1Rad:=ATAN2_YX(y := 10.0, x := -10.0); (* Result: 2.3561947 *)
rPhi1Degree := rPhi1Rad * DEGR_OF_RAD;      (* Result: 135.00002 *)

(* Angle value of point in quadrant 4 *)
rPhi2Rad:=ATAN2_YX(y := -5.0, x :=   5.0); (* Result: -0.78539819 *)
rPhi2Degree := rPhi2Rad * DEGR_OF_RAD;      (* Result: -45.0 *)
```

LN Natural logarithm

Description LN calculates the logarithm of the input variable (value > 0.0) to the base e (Euler's number = 2.7182818) and writes the result into the output variable. This function is the reversion of the EXP (see page 80) function.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

☞ **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

PLC types **Availability of** LN **(see page 1328)**

Data types

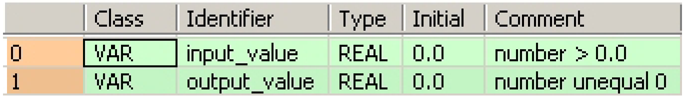
Data type	I/O	Function
REAL	input	input value
REAL	output as input	natural logarithm of input value

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input variable does not have the data type REAL or input variable is not > 0.0
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	▪ output variable is zero
R9009	%MX0.900.9	for an instant	▪ processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

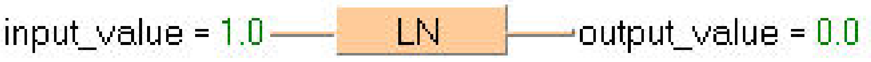


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	number > 0.0
1	VAR	output_value	REAL	0.0	number unequal 0

This example uses variables. You can also use a constant for the input variable.

Body The logarithm of **input_value** is calculated to the base e and written into **output_value**.

LD



ST When programming with structured text, enter the following:

```
output_value:=LN(input_value);
```

LOG

Logarithm to the Base 10

Description **LOG** calculates the logarithm of the input variable (value > 0.0) to the base 10 and writes the result into the output variable.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.

PLC types **Availability of** LOG **(see page 1328)**

Data types

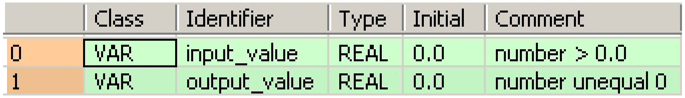
Data type	I/O	Function
REAL	input	input value
REAL	output as input	logarithm of input value

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input variable does not have the data type REAL or input variable is not > 0.0
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	▪ output variable is zero
R9009	%MX0.900.9	for an instant	▪ processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

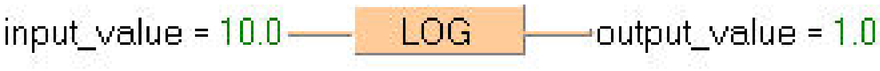


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	number > 0.0
1	VAR	output_value	REAL	0.0	number unequal 0

This example uses variables. You can also use a constant for the input variable.

Body The logarithm of **input_value** is calculated to the base 10 and written into **output_value**.

LD



ST When programming with structured text, enter the following:

```
output_value:=LOG(input_value);
```

EXP

Exponent of input variable to base e

Description EXP calculates the power of the input variable to the base e (Euler's number = 2.7182818) and writes the result into the output variable. The input variable has to be greater than -87.33 and smaller than 88.72. This function is the reversion of the LN (see page 78) function.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

☞ **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

PLC types **Availability of** EXP **(see page 1320)**

Data types

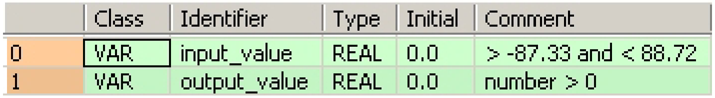
Data type	I/O	Function
REAL	input	input value between -87.33 and +88.72
REAL	output as input	exponent of input variable to base e

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input variable does not have the data type REAL or input variable is not > -87.33 and < 88.72
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	▪ output variable is zero
R9009	%MX0.900.9	for an instant	▪ processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

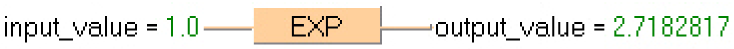


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	> -87.33 and < 88.72
1	VAR	output_value	REAL	0.0	number > 0

This example uses variables. You can also use a constant for the input variable.

Body The power of **input_value** is calculated to the base e and written into **output_value**.

LD



ST When programming with structured text, enter the following:

```
output_value:=EXP(input_value);
```

EXPT

Raises 1st input variable by the power of the 2nd input variable

Description EXPT raises the first input variable to the power of the second input variable (OUT = $IN1^{IN2}$) and writes the result into the output variable. Input variables have to be within the range -1.70141 x 10 E^{38} to 1.70141 x 10 E^{38}.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** EXPT **(see page 1320)**

Data types

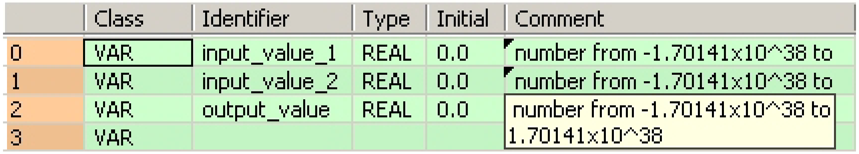
Data type	I/O	Function
REAL	1st input	input value
REAL	2nd input	exponent of the input value
REAL	output as 1st input	result

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ first and the second input variable do not have the data type REAL
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	▪ output variable is zero
R9009	%MX0.900.9	for an instant	▪ processing result overflows the output variable

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

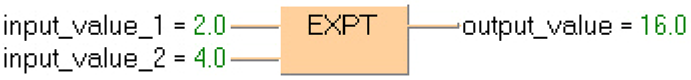


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value_1	REAL	0.0	number from -1.70141x10^38 to
1	VAR	input_value_2	REAL	0.0	number from -1.70141x10^38 to
2	VAR	output_value	REAL	0.0	number from -1.70141x10^38 to 1.70141x10^38
3	VAR				

In this example the input variables (**input_value_1** and **input_value_2**) have been declared. Instead, you may enter constants directly at the input contacts of a function.

Body **Input_value_1** is raised to the power of **input_value_2**. The result is written into **output_value**.

LD



ST

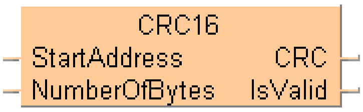
When programming with structured text, enter the following:

```
output_value:=input_value_1**input_value_2;
```

CRC16

Cyclic Redundancy Check

Description This function calculates the CRC16 (**C**yclic **R**edundancy **C**heck) for all PLC types by using 8 bytes (8 bits) specified with the parameter **NumberOfBytes** and the starting address **StartAddress**.



Depending on the PLC type, one of the following two implementations of the function will be used:

- PLCs which support the instruction F70_BCC (see page 411) with the parameter s1=10 to calculate CRC16 (FP-e, FP-Sigma, FP2, FP2SH, FP10SH) use F70_BCC (see page 411) directly.
- For the other PLCs (FP0, FP0R, FP3, FP5, FP10), a sub-program making an explicit CRC16 calculation is called. The following restrictions apply to this sub-program:
  - During the first eight execution scans an internal table is built. During this time, no check sum is calculated, and the output **IsValid** remains FALSE. Starting with the fifth scan, the check sum is calculated, and the output **IsValid** is set to TRUE.
  - **StartAddress** requires an address in the DT or FL area.

☞ **The number of steps can increase up to approx. 200 when CRC16 is used as a sub-program.**

When programming, please be aware that a certain amount of time is needed to build the internal table and to calculate the check sum, especially for large data volumes.

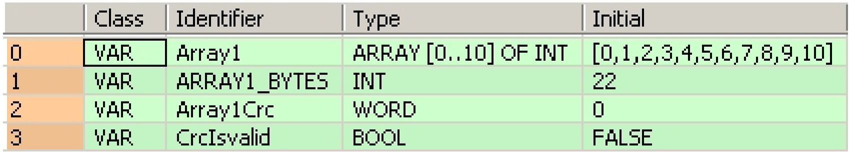
PLC types **Availability of** CRC16 **(see page 1318)**

Data types

Input variables (VAR_INPUT):		
Variable	**Data type**	**Function**
StartAddress	ANY	Starting address for the calculation of the check sum. For PLCs which do not support the instruction F70_BCC (see page 411) with CRC16 calculation (FP0, FP5, FP10), the starting address must be in the DT or FL area.
NumberOfBytes	INT	The number of bytes (8 bits), beginning with **AdrStart**, on which the CRC16 calculation is performed.
Output variables (VAR_OUTPUT):		
CRC	ANY16	The calculated check sum, which is only valid if the flag **IsValid** is set to TRUE.
IsValid	BOOL	Flag indicating whether the calculated check sum is valid or not. For PLCs which do not support the instruction F70_BCC (see page 411) with CRC16 calculation (FP0, FP5, FP10) the CRC is not valid: ▪ during the first eight execution scans when an internal table is built ▪ if the address area of the variable StartAddress is not in the DT or FL area. For PLCs that support the instruction F70_BCC with CRC16 calculation, the CRC is always valid.

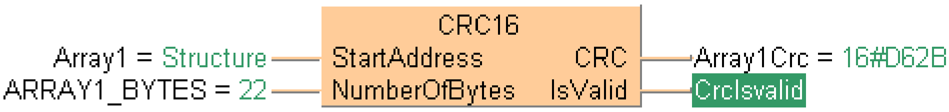
Example In this example, the same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Array1	ARRAY [0..10] OF INT	[0,1,2,3,4,5,6,7,8,9,10]
1	VAR	ARRAY1_BYTES	INT	22
2	VAR	Array1Crc	WORD	0
3	VAR	CrcIsvalid	BOOL	FALSE

LD



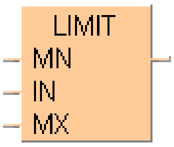
ST When programming with structured text, enter the following:

```
CRC16(StartAddress := Array1,
        NumberOfBytes := ARRAY1_BYTES,
        CRC => Array1Crc,
        IsValid => CrcIsvalid);
```

LIMIT

Limit value for input variable

Description In LIMIT the 1st input variable forms the lower and the 3rd input variable the upper limit value. If the 2nd input variable is within this limit, it will be transferred to the output variable. If it is above this limit, the upper limit value will be transferred; if it is below this limit the lower limit value will be transferred.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

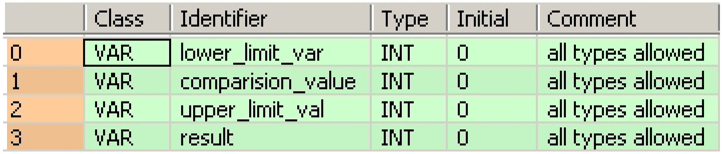
PLC types **Availability of** LIMIT **(see page 1328)**

Data types

Data type	I/O	Function
all data types	1st input	upper limit
all data types	2nd input	value compared to upper and lower limit
all data types	3rd input	lower limit
all data types	output as input	result, 2nd input value if between upper and lower limit, otherwise the upper or lower limit

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

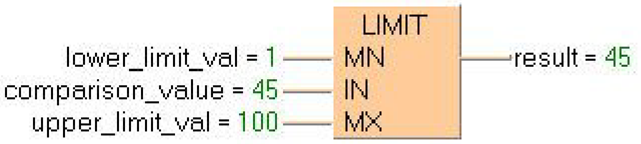


	Class	Identifier	Type	Initial	Comment
0	VAR	lower_limit_var	INT	0	all types allowed
1	VAR	comparision_value	INT	0	all types allowed
2	VAR	upper_limit_val	INT	0	all types allowed
3	VAR	result	INT	0	all types allowed

In this example the input variables (**lower_limit_val, comparison_value** and **upper_val**) have been declared. Instead, you may enter a constant directly at the input contact of a function.

Body **Lower_limit_val** and **upper_limit_val** form the range where the **comparison_value** has to be, if it has to be transferred to **result**. If the **comparison_value** is above the **upper_limit_val**, the value of **upper_limit_val** will be transferred to **result**. If it is below the **lower_limit_val**, the value of **lower_limit_val** will be transferred to **result**.

LD



ST When programming with structured text, enter the following:

```
result:=LIMIT(MN:=lower_limit_val, IN:=comparison_value,
MX:=upper_limit_val);
```

Chapter 4

Bitwise Boolean instructions

AND

Logical AND operation

Description The content of the accumulator is connected with the operand defined in the operand field by a logical AND operation. The result is transferred to the accumulator.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of AND (see page 1335)**

☞

- **All operands must be of the same data type.**
- **This function can be expanded to a maximum of 28 input contacts.**
- **Modifying elements**

Data types

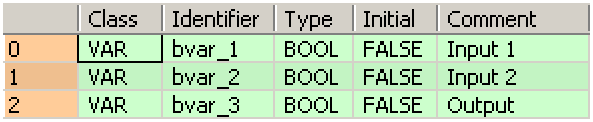
Data type	I/O	Function
BOOL, WORD, DWORD	1st input	element 1 of logical AND operation
BOOL, WORD, DWORD	2nd input	element compared to input 1
BOOL, WORD, DWORD	output as input	result

Truth table:

	Input 1	Input 2	Output
Signal	0	0	0
	0	1	0
	1	0	0
	1	1	1

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

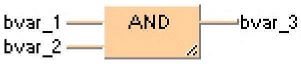
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	bvar_1	BOOL	FALSE	Input 1
1	VAR	bvar_2	BOOL	FALSE	Input 2
2	VAR	bvar_3	BOOL	FALSE	Output

Body **bvar_1** will be logically AND-linked with **bvar_2**. The result will be written into the output variable **bvar_3**.

LD



ST When programming with structured text, enter the following:

```
bvar_3:= bvar_1&bvar_2;
```

OR

Logical OR operation

Description The content of the accumulator is connected with the operand defined in the operand field by a logical OR operation. The result is transferred to the accumulator.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of OR (see page 1335)**

☞

- **All operands must be of the same data type.**
- **This function can be expanded to a maximum of 28 input contacts.**
- **Modifying elements**

Data types

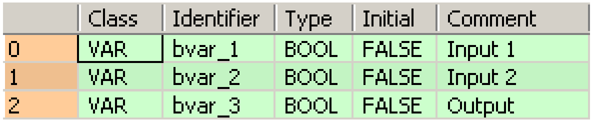
Data type	I/O	Function
BOOL, WORD, DWORD	1st input	element 1 of logical OR operation
BOOL, WORD, DWORD	2nd input	element compared to input 1
BOOL, WORD, DWORD	output as input	result

Truth table:

	Input 1	Input 2	Output
Signal	0	0	0
	1	0	1
	0	1	1
	1	1	1

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

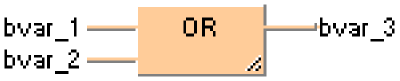
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	bvar_1	BOOL	FALSE	Input 1
1	VAR	bvar_2	BOOL	FALSE	Input 2
2	VAR	bvar_3	BOOL	FALSE	Output

Body **bvar_1** and **bvar_2** are linked with a logical OR. The result will be written in **bvar_3**. This example uses variables. You may also use constants for the input variables.

LD



ST When programming with structured text, enter the following:

```
bvar_3:= var_1 OR bvar_2;
```

XOR

Exclusive OR operation

Description The content of the accumulator is connected with the operand defined in the operand field by a logical XOR operation. The result is transferred to the accumulator.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** XOR **(see page 1335)**

☞

- **All operands must be of the same data type.**
- **This function can be expanded to a maximum of 28 input contacts.**
- **Modifying elements**

Data types

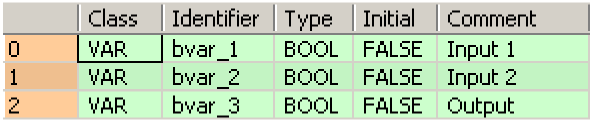
Data type	I/O	Function
BOOL, WORD, DWORD	1st input	element 1 of logical XOR operation
BOOL, WORD, DWORD	2nd input	element compared to input 1
BOOL, WORD, DWORD	output as input	result

Truth table:

	Input 1	Input 2	Output
Signal	0	0	0
	1	0	1
	0	1	1
	1	1	0

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

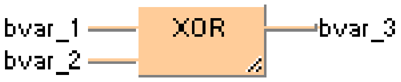
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	bvar_1	BOOL	FALSE	Input 1
1	VAR	bvar_2	BOOL	FALSE	Input 2
2	VAR	bvar_3	BOOL	FALSE	Output

Body The Boolean variables **bvar_1** and **bvar_2** are logically EXCLUSIVE-OR linked and the result is written in **bvar_3.**

LD



ST When programming with structured text, enter the following:

```
var_3:= var_1 XOR var_2;
```

NOT

Bit inversion

Description NOT performs a bit inversion of input variables. The result will be written into the output variable.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** NOT **(see page 1328)**

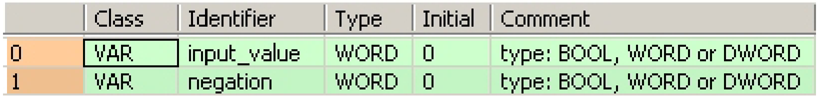
☞ **All operands must be of the same data type.**

Data types

Data type	I/O	Function
BOOL, WORD, DWORD	input	input for NOT operation
BOOL, WORD, DWORD	output as input	result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

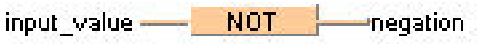


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	WORD	0	type: BOOL, WORD or DWORD
1	VAR	negation	WORD	0	type: BOOL, WORD or DWORD

This example uses variables. You can also use a constant for the input variable.

Body The bits of **input_value** are inversed (0 is inversed to 1 and vice versa). The inversed result is written into **negation.**

LD



ST When programming with structured text, enter the following:

```
negation:= NOT(input_value);
```

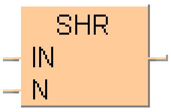
Chapter 5

Bit-shift instructions

SHR

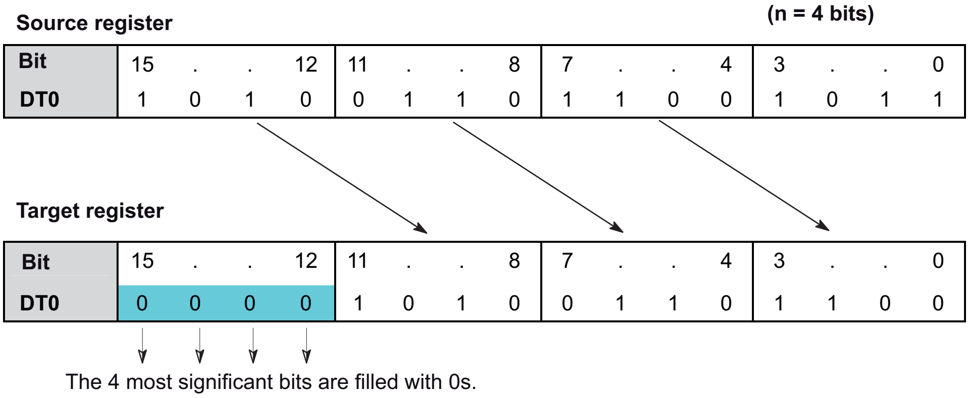
Shift bits to the right

Description SHR shifts a bit value by a defined number of positions (N) to the right and fills the vacant positions with zeros.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Bit shift to the right, zero-filled on left:



Source register

(n = 4 bits)

Bit	15	.	.	12	11	.	.	8	7	.	.	4	3	.	.	0
DT0	1	0	1	0	0	1	1	0	1	1	0	0	1	0	1	1

Target register

Bit	15	.	.	12	11	.	.	8	7	.	.	4	3	.	.	0
DT0	0	0	0	0	1	0	1	0	0	1	1	0	1	1	0	0

The 4 most significant bits are filled with 0s.

PLC types **Availability of** SHR **(see page 1330)**

Data types

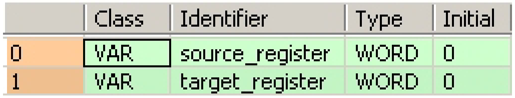
Data type	I/O	Function
ANY_BIT	1st input	input value
	2nd input	number of bits by which the input value is shifted to the right
	output as input	result

☞

- **If the second input variable N (the number of bits to be shifted) is of the data type DWORD, then only the lower 16 bits are taken into account.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

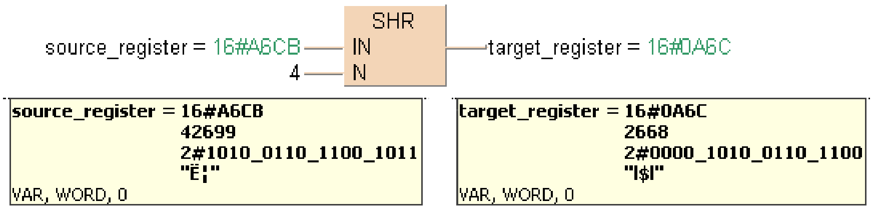


	Class	Identifier	Type	Initial
0	VAR	source_register	WORD	0
1	VAR	target_register	WORD	0

This example uses variables. You can also use a constant for the input variable.

Body The last **N** bits (here 4) of **source_register** are right-shifted. The vacant positions on the left are filled with zeros. The result is written into **target_register**.

LD



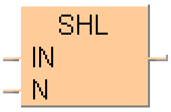
```
source_register = 16#A6CB
                  42699
                  2#1010_0110_1100_1011
                  "Ë¦"
VAR, WORD, 0
```

```
target_register = 16#0A6C
                  2668
                  2#0000_1010_0110_1100
                  "l$l"
VAR, WORD, 0
```

SHL

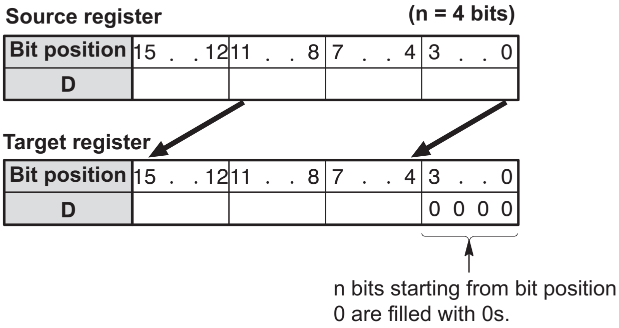
Shift bits to the left

Description SHL shifts a bit value by a defined number of positions (N) to the left and fills the vacant positions with zeros.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Bit shift to the left, zero-filled on right:



PLC types **Availability of** SHL **(see page 1330)**

Data types

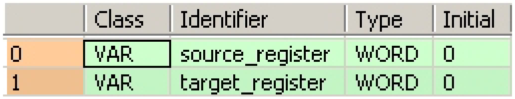
Data type	I/O	Function
ANY_BIT	1st input	input value
	2nd input	number of bits by which the input value is shifted to the left
	output as input	result

☞

- **If the second input variable N (the number of bits to be shifted) is of the data type DWORD, then only the lower 16 bits are taken into account.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

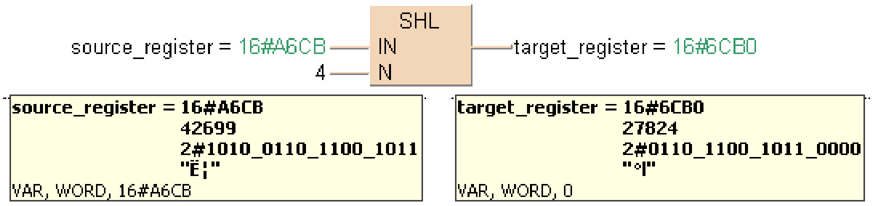


	Class	Identifier	Type	Initial
0	VAR	source_register	WORD	0
1	VAR	target_register	WORD	0

This example uses variables. You can also use a constant for the input variable.

Body The first **N** bits (here 4) of **source_register** are left-shifted, the vacant positions on the right are filled with zeros. The result is written into **target_register**.

LD



```
source_register = 16#A6CB
4
target_register = 16#6CB0
```

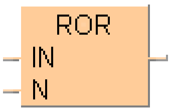
```
source_register = 16#A6CB
                  42699
                  2#1010_0110_1100_1011
                  "Ë¡"
VAR, WORD, 16#A6CB
```

```
target_register = 16#6CB0
                  27824
                  2#0110_1100_1011_0000
                  "°l"
VAR, WORD, 0
```

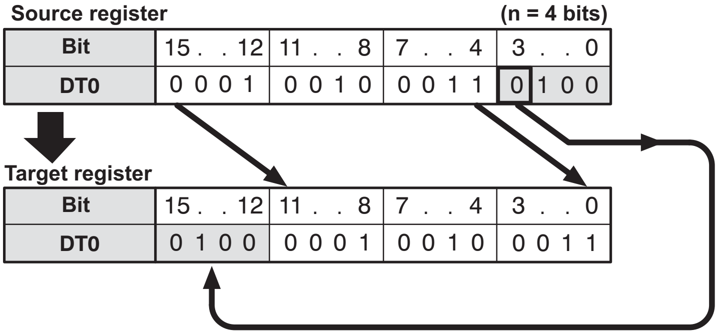
ROR

Rotate N bits the right

Description ROR rotates a defined number (N) of bits to the right.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.



Source register **(n = 4 bits)**

Bit	15 . . 12	11 . . 8	7 . . 4	3 . . 0
DT0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0

Target register

Bit	15 . . 12	11 . . 8	7 . . 4	3 . . 0
DT0	0 1 0 0	0 0 0 1	0 0 1 0	0 0 1 1

PLC types **Availability of** ROR **(see page 1330)**

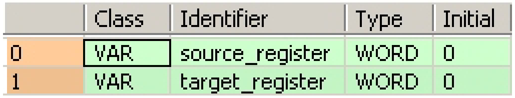
Data types

Data type	**I/O**	**Function**
ANY_BIT	1st input	input value
	2nd input	number of bits by which the input value is rotated to the right
	output as input	result

The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

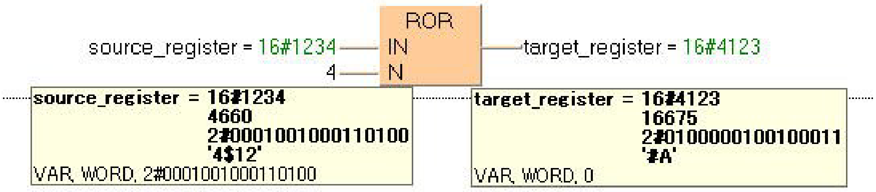


	Class	Identifier	Type	Initial
0	VAR	source_register	WORD	0
1	VAR	target_register	WORD	0

This example uses variables. You can also use a constant for the input variable.

Body The first **N** bits (here N = 4) of **source_register** are right-rotated. The result will be written into **target_register**.

LD

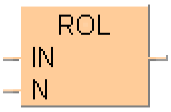


```
source_register = 16#1234                target_register = 16#4123

source_register = 16#1234                target_register = 16#4123
                  4660                                     16675
                  2#0001001000110100                       2#0100000100100011
                  '4$12'                                   '#A'
VAR, WORD, 2#0001001000110100            VAR, WORD, 0
```

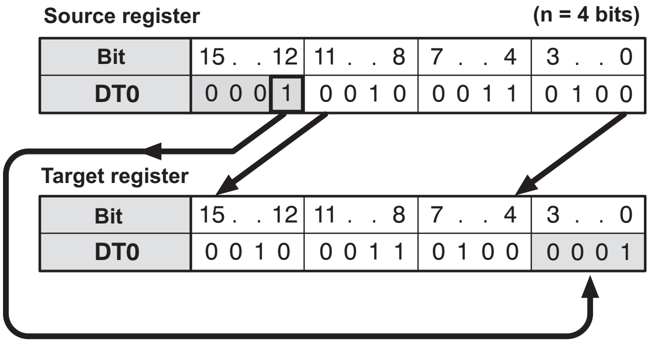
ROL

Rotate N bits to the left

Description ROL rotates a defined number (N) of bits to the left.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.



Source register **(n = 4 bits)**

Bit	15 . . 12	11 . . 8	7 . . 4	3 . . 0
DT0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0

Target register

Bit	15 . . 12	11 . . 8	7 . . 4	3 . . 0
DT0	0 0 1 0	0 0 1 1	0 1 0 0	0 0 0 1

PLC types **Availability of** ROL **(see page 1330)**

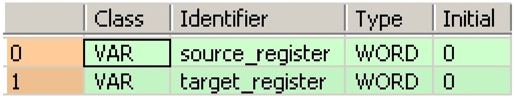
Data types

Data type	I/O	Function
ANY_BIT	1st input	input value
	2nd input	number of bits by which the input value is rotated to the left
	output as input	result

The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

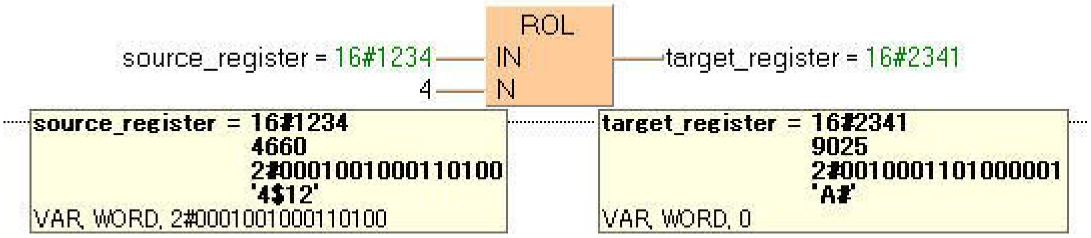


	Class	Identifier	Type	Initial
0	VAR	source_register	WORD	0
1	VAR	target_register	WORD	0

This example uses variables. You can also use a constant for the input variable.

Body The last **N** bits (here 4) of **source_register** are left-rotated. The result will be written in **target_register**.

LD



```
source_register = 16#1234 —— IN   —— target_register = 16#2341
                        4 —— N

source_register = 16#1234
                  4660
                  2#0001001000110100
                  '4$12'
VAR, WORD, 2#0001001000110100

target_register = 16#2341
                  9025
                  2#0010001101000001
                  'A#'
VAR, WORD, 0
```

Chapter 6

Comparison instructions

GT

Greater than

Description The content of the accumulator is compared with the operand defined in the operand field. If the accumulator is greater than the reference value, "TRUE" is stored in the accumulator, otherwise "FALSE".

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** GT **(see page 1335)**

☞

- **Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.**
- **The number of input contacts lies in the range of 2 to 28.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

Data types

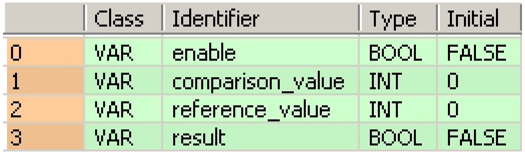
Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if value for comparison is greater than the reference value

The variables that are compared to each other must be of the same data type.

When using more inputs, the first input is compared with the second, the second input is compared with the third input etc. If the first value is greater than the second value AND the second value greater than third etc., TRUE will be written into result, otherwise FALSE.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

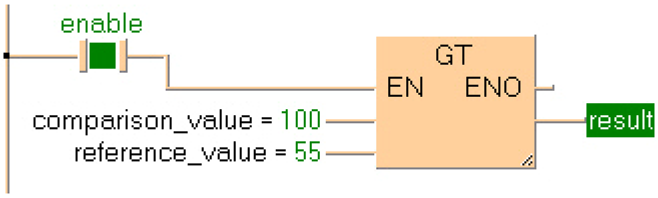


	Class	Identifier	Type	Initial
0	VAR	enable	BOOL	FALSE
1	VAR	comparison_value	INT	0
2	VAR	reference_value	INT	0
3	VAR	result	BOOL	FALSE

In this example the input variables (**comparison_value, reference_value** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body If **enable** is set (TRUE), the **comparison_value** is compared with the **reference_value**. If the **comparison_value** is greater than the **reference_value**, the value TRUE will be written into **result**, otherwise FALSE.

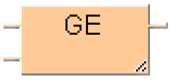
LD


enable
GT
EN
ENO
comparison_value = 100
reference_value = 55
result

GE

Greater than or equal to

Description The content of the accumulator is compared with the operand defined in the operand field. If the accumulator is greater or equal to the reference value, "TRUE" is stored in the accumulator, otherwise "FALSE".



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types ☞ **Availability of** GE **(see page 1335)**

- **Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.**
- **The number of input contacts lies in the range of 2 to 28.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

Data types

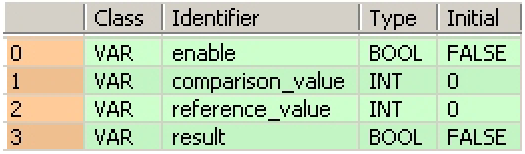
Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if value for comparison is greater than or equal to the reference value

The variables that are compared to each other must be of the same data type.

When using more inputs, the first input is compared with the second, the second input is compared with the third input etc. If the first value is greater than or equal to the second value AND the second value is greater than or equal to the third value etc., TRUE will be written into result, otherwise FALSE.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables which are required for programming the function are declared in the POU header.

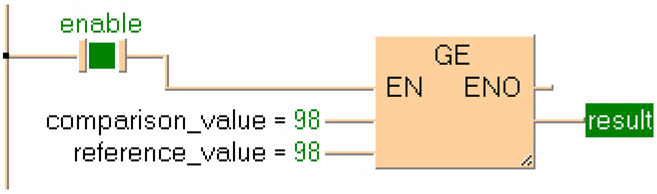


	Class	Identifier	Type	Initial
0	VAR	enable	BOOL	FALSE
1	VAR	comparison_value	INT	0
2	VAR	reference_value	INT	0
3	VAR	result	BOOL	FALSE

In this example the input variables (**comparison_value, reference_value** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body If **enable** is set (TRUE), the **comparison_value** is compared with the **reference_value**. If the **comparison_value** is greater than or equal to the **reference_value**, the value TRUE will be written into **result**, otherwise FALSE.

LD


enable
GE
EN
ENO
comparison_value = 98
reference_value = 98
result

EQ

Equal to

Description The content of the accumulator is compared with the operand defined in the operand field. If both values are equal, "TRUE" is stored in the accumulator, otherwise "FALSE".

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types ☞ **Availability of EQ (see page 1335)**

- **Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.**
- **The number of input contacts lies in the range of 2 to 28.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

Data types

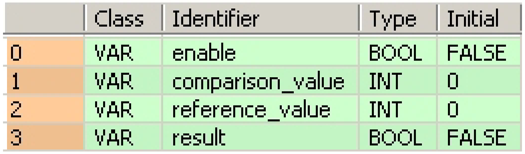
Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if value for comparison is equal to the reference value

The variables that are compared to each other must be of the same data type.

When using more inputs, the first input is compared with the second, the second input is compared with the third input etc. If the first value is equal to the second value AND the second value is equal to the third value etc., TRUE will be written into result, otherwise FALSE.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables which are required for programming the function are declared in the POU header.

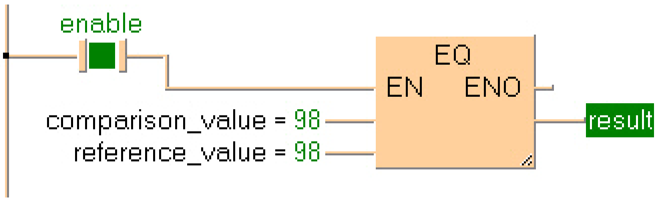


	Class	Identifier	Type	Initial
0	VAR	enable	BOOL	FALSE
1	VAR	comparison_value	INT	0
2	VAR	reference_value	INT	0
3	VAR	result	BOOL	FALSE

In this example the input variables (**comparison_value, reference_value** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body If **enable** is set (TRUE), the variable **comparison_value** is compared with the variable **reference_value**. If the values of the two variables are identical, the value TRUE will be written into **result**, otherwise FALSE.

LD



LE

Less than or equal to

Description The content of the accumulator is compared with the operand defined in the operand field. If the accumulator is less or equal to the reference value, "TRUE" is stored in the accumulator, otherwise "FALSE".

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of LE (see page 1335)**

- **Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.**
- **The number of input contacts lies in the range of 2 to 28.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

Data types

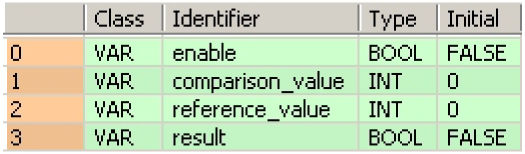
Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if value for comparison is less than or equal to the reference value

The variables that are compared to each other must be of the same data type.

When using more inputs, the first input is compared with the second, the second input is compared with the third input etc. If the first value is less than or equal to the second value AND the second value is less than or equal to the third value etc., TRUE will be written into result, otherwise FALSE.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables which are required for programming the function are declared in the POU header.

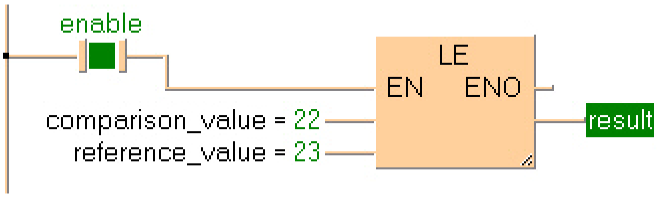


	Class	Identifier	Type	Initial
0	VAR	enable	BOOL	FALSE
1	VAR	comparison_value	INT	0
2	VAR	reference_value	INT	0
3	VAR	result	BOOL	FALSE

In this example the input variables (**comparison_value, reference_value** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body If **enable** is set (TRUE), the **comparison_value** is compared with the variable **reference_value**. If the **comparison_value** is less than or equal to the **reference_value**, TRUE will be written into **result**, otherwise FALSE.

LD



LT

Less than

Description The content of the accumulator is compared with the operand defined in the operand field. If the accumulator is less than the reference value, "TRUE" is stored in the accumulator, otherwise "FALSE".

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types ☞ **Availability of LT (see page 1335)**

- **Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.**
- **The number of input contacts lies in the range of 2 to 28.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

Data types

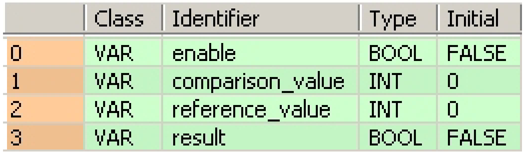
Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if value for comparison is less than the reference value

The variables that are compared to each other must be of the same data type.

When using more inputs, the first input is compared with the second, the second input is compared with the third input etc. If the first value is less than the second value AND the second value is less than the third value etc., TRUE will be written into result, otherwise FALSE.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables which are required for programming the function are declared in the POU header.

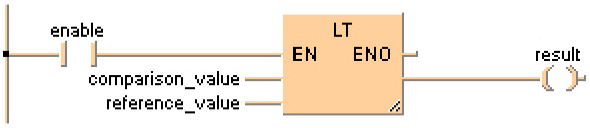


	Class	Identifier	Type	Initial
0	VAR	enable	BOOL	FALSE
1	VAR	comparison_value	INT	0
2	VAR	reference_value	INT	0
3	VAR	result	BOOL	FALSE

In this example the input variables (**comparison_value**, **reference_value** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body If **enable** is set (TRUE), the **comparison_value** is compared with the **reference_value**. If the **comparison_value** is less than the **reference_value**, TRUE will be written into result, otherwise FALSE.

LD



NE

Not equal

Description The content of the accumulator is compared with the operand defined in the operand field. If both values are not equal, "TRUE" is stored in the accumulator, otherwise "FALSE".

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** NE **(see page 1335)**

- **Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

Data types

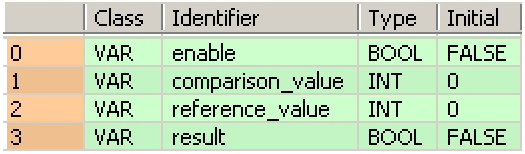
Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if value for comparison is unequal to the reference value, otherwise FALSE

The variables that are compared to each other must be of the same data type.

When using more inputs, the first input is compared with the second, the second input is compared with the third input etc. If the first value is not equal to the second value AND the second value is not equal to the third value etc., TRUE will be written into result, otherwise FALSE.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables which are required for programming the function are declared in the POU header.

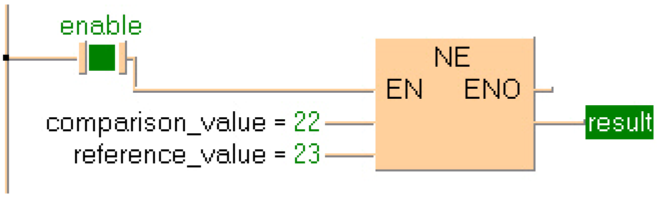


	Class	Identifier	Type	Initial
0	VAR	enable	BOOL	FALSE
1	VAR	comparison_value	INT	0
2	VAR	reference_value	INT	0
3	VAR	result	BOOL	FALSE

In this example the input variables (**comparison_value, reference_value** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

Body If **enable** is set (TRUE), the **comparison_value** is compared with the **reference_value**. If the two values are unequal, TRUE will be written into **result**, otherwise FALSE.

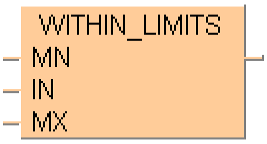
LD



WITHIN_LIMITS

Evaluate if a value is within the limits

Description This instruction evaluates whether the value at the input **IN** is within the limits set at minimum (**MN**) and maximum **MX**.



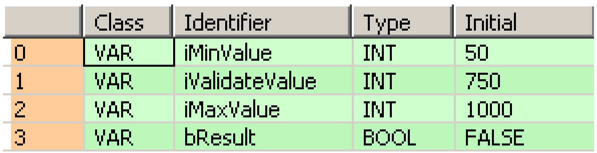
PLC types see see page 1333

Data types

Variable	Data type	Function
MN	ANY_SIMPLE	Minimum limit
IN		Value compared to the limits
MX		Maximum limit
Output variable	BOOL	TRUE if the input value at **IN** falls within the lower and upper limits

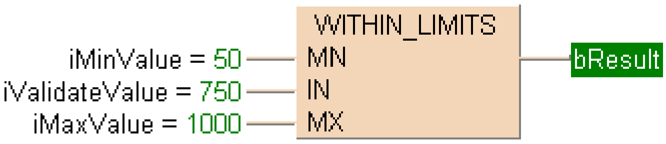
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iMinValue	INT	50
1	VAR	iValidateValue	INT	750
2	VAR	iMaxValue	INT	1000
3	VAR	bResult	BOOL	FALSE

LD



ST
```
bResult := WITHIN_LIMITS(MN := iMinValue, IN := iValidateValue, MX :=
iMaxValue);
```

Chapter 7

Conversion instructions

WORD_TO_BOOL

WORD in BOOL

Description WORD_TO_BOOL converts a value of the data type WORD into a value of the data type BOOL.

WORD_TO_BOOL

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** WORD_TO_BOOL **(see page** 1333**)**

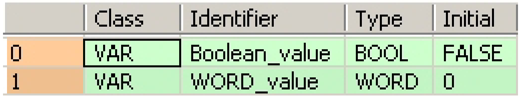
If the input value = 0 (16#0000), the conversion result will be = 0 (FALSE); in any other case, it will be = 1 (TRUE).

Data types

Data type	I/O	Function
WORD	input	input data type
BOOL	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

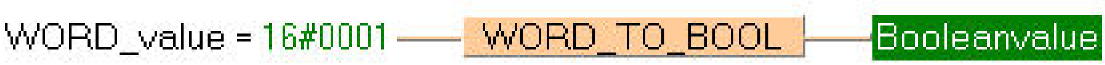


	Class	Identifier	Type	Initial
0	VAR	Boolean_value	BOOL	FALSE
1	VAR	WORD_value	WORD	0

This example uses variables. You can also use a constant for the input variable.

Body **WORD_value** of the data type WORD (16-bit) is converted into a Boolean value (1-bit). The result will be written into **Boolean_value**.

LD



ST When programming with structured text, enter the following:

```
Boolean_value:=WORD_TO_BOOL(WORD_value);
```

DWORD_TO_BOOL

DOUBLE WORD in BOOL

Description DWORD_TO_BOOL converts a value of the data type DOUBLE WORD into a value of the data type BOOL.

DWORD_TO_BOOL

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** DWORD_TO_BOOL **(see page** 1319**)**

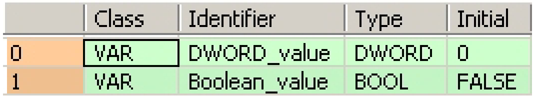
If the input value = 0 (16#0000), the conversion result will be = 0 (FALSE); in any other case, it will be = 1 (TRUE).

Data types

Data type	I/O	Function
DWORD	input	input data type
BOOL	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

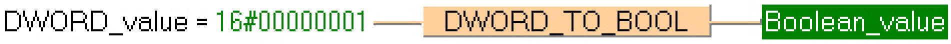


	Class	Identifier	Type	Initial
0	VAR	DWORD_value	DWORD	0
1	VAR	Boolean_value	BOOL	FALSE

This example uses variables. You can also use a constant for the input variable.

Body **DWORD_value** of the data type DOUBLE WORD is converted into a Boolean value (1-bit). the converted value is written into **Boolean_value**.

LD



ST When programming with structured text, enter the following:

```
Boolean_value:=DWORD_TO_BOOL(DWORD_value);
```

INT_TO_BOOL

INTEGER into BOOL

Description INT_TO_BOOL converts a value of the data type INT into a value of the data type BOOL.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** INT_TO_BOOL **(see page** 1327**)**

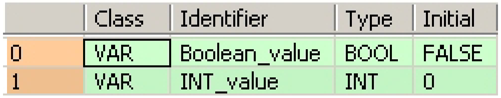
☞ **If the input value = 0 (16#0000), the conversion result will be = 0 (FALSE); in any other case, it will be = 1 (TRUE).**

Data types

Data type	I/O	Function
INT	input	input data type
BOOL	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

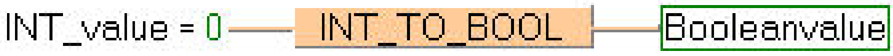


	Class	Identifier	Type	Initial
0	VAR	Boolean_value	BOOL	FALSE
1	VAR	INT_value	INT	0

This example uses variables. You can also use a constant for the input variable.

Body **INT_value** (16-bit) of the data type INTEGER is converted into a Boolean value. The result is written into **Boolean_value.**

LD



ST When programming with structured text, enter the following:

```
Boolean_value:=INT_TO_BOOL(INT_value);
```

☞ **If INT_value has the value 0, the conversion result will be 0 (FALSE), in any other case it will be 1 (TRUE).**

DINT_TO_BOOL

DOUBLE INTEGER into BOOL

Description DINT_TO_BOOL converts a value of the data type DINT into a value of the data type BOOL.

DINT_TO_BOOL

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of DINT_TO_BOOL (see page 1319)**

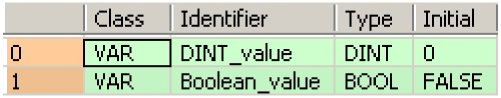
☞ **If the input value = 0 (16#0000), the conversion result will be = 0 (FALSE); in any other case, it will be = 1 (TRUE).**

Data types

Data type	I/O	Function
DINT	input	input data type
BOOL	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

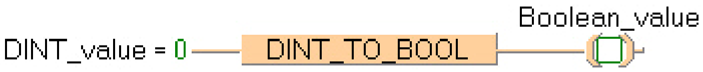


	Class	Identifier	Type	Initial
0	VAR	DINT_value	DINT	0
1	VAR	Boolean_value	BOOL	FALSE

In this example the input variable (**DINT_value**) has been declared. Instead, you may enter a constant directly at the input contact of a function.

Body **DINT_value** of the data type DOUBLE INTEGER is converted into a value of the data type BOOL. The converted value in written into **Boolean_value**.

LD



ST When programming with structured text, enter the following:

```
Boolean_value:=DINT_TO_BOOL(DINT_value);
```

☞ **If the variable DINT_value has the value 0, the conversion result is FALSE, in any other case TRUE.**

UINT_TO_BOOL

Unsigned INTEGER into BOOL

Description UINT_TO_BOOL converts a value of the data type Unsigned INTEGER into a value of the data type BOOL.

UINT_TO_BOOL

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** UINT_TO_BOOL **(see page** 1332**)**

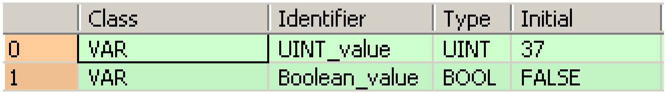
If the input value = 0 (16#0000), the conversion result will be = 0 (FALSE); in any other case, it will be = 1 (TRUE).

Data types

Data type	**I/O**	**Function**
UINT	Input	input data type
BOOL	Output	conversion result

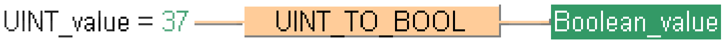
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	37
1	VAR	Boolean_value	BOOL	FALSE

LD



ST
```
Boolean_value:=  UINT_TO_BOOL(UINT_value);
```

UDINT_TO_BOOL

Unsigned DOUBLE INTEGER into BOOL

Description UDINT_TO_BOOL converts a value of the data type Unsigned DOUBLE INTEGER into a value of the data type BOOL.

UDINT_TO_BOOL

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** UDINT_TO_BOOL **(see page 1332)**

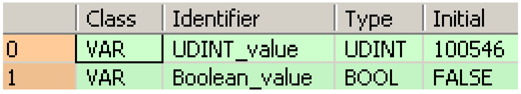
☞ **If the input value = 0 (16#0000), the conversion result will be = 0 (FALSE); in any other case, it will be = 1 (TRUE).**

Data types

Data type	I/O	Function
UDINT	Input	input data type
BOOL	Output	conversion result

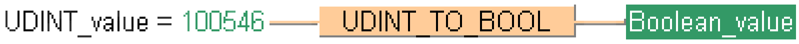
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	100546
1	VAR	Boolean_value	BOOL	FALSE

LD



ST When programming with structured text, enter the following:

```
Boolean_value := UDINT_TO_BOOL(UDINT_value);
```

BOOL_TO_WORD

BOOL into WORD

Description BOOL_TO_WORD converts a value of the data type BOOL into a value of the data type WORD.

BOOL_TO_WORD

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

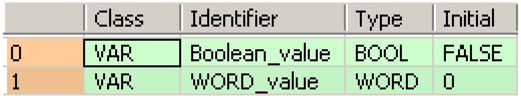
PLC types **Availability of** BOOL_TO_WORD **(see page** 1318**)**

Data types

Data type	I/O	Function
BOOL	input	input data type
WORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

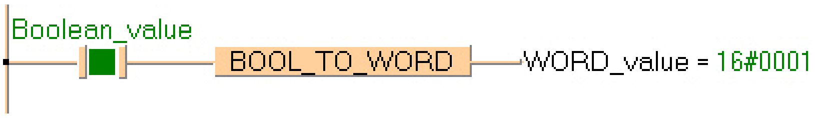


	Class	Identifier	Type	Initial
0	VAR	Boolean_value	BOOL	FALSE
1	VAR	WORD_value	WORD	0

In this example the input variable (**Boolean_value**) has been declared. Instead, you may enter a constant directly at the input contact of a function.

Body The **Boolean_value** of the data type BOOL is converted into a value of the data type WORD. The converted value is written into **WORD_value**.

LD



ST When programming with structured text, enter the following:

```
IF Boolean_value THEN
    WORD_value:=BOOL_TO_WORD(Boolean_value);
END_IF;
```

BOOL16_TO_WORD

BOOL16 to WORD

Description This function copies a variable of the special data type BOOL16 (see page 55) (an array with 16 elements of the data type BOOL or a DUT of 16 members of the data type BOOL) at the input to the data type WORD at the output.

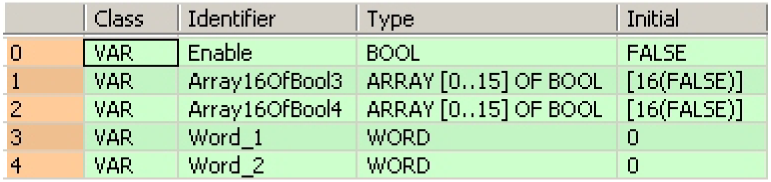
To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** BOOL16_TO_WORD **(see page 1318)**

Data types

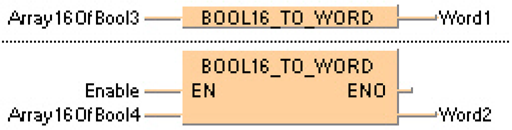
Data type	Comment
ARRAY of BOOL	ARRAY with 16 elements
WORD	output variable

POU header:



	Class	Identifier	Type	Initial
0	VAR	Enable	BOOL	FALSE
1	VAR	Array16OfBool3	ARRAY [0..15] OF BOOL	[16(FALSE)]
2	VAR	Array16OfBool4	ARRAY [0..15] OF BOOL	[16(FALSE)]
3	VAR	Word_1	WORD	0
4	VAR	Word_2	WORD	0

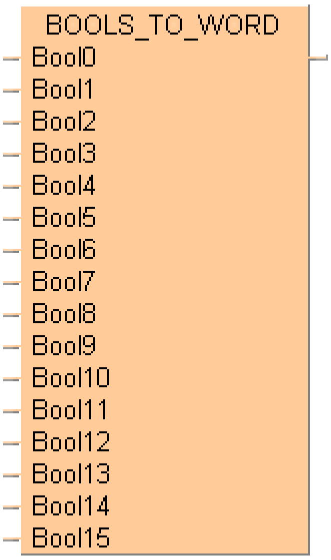
Body with and without EN/ENO:



BOOLS_TO_WORD

16 Variables of the data type BOOL to WORD

Description This function converts 16 values of the data type BOOL bit-wise to a value of the data type WORD.



The inputs Bool0 to Bool15 need not be allocated in LD or FBD, or used explicitly in the ST editor's formal list of parameters. Such unused inputs are assumed to be FALSE. No program code is generated for these inputs (or for any input allocated with the constants TRUE or FALSE). Program code is only generated for inputs to which a variable is allocated.

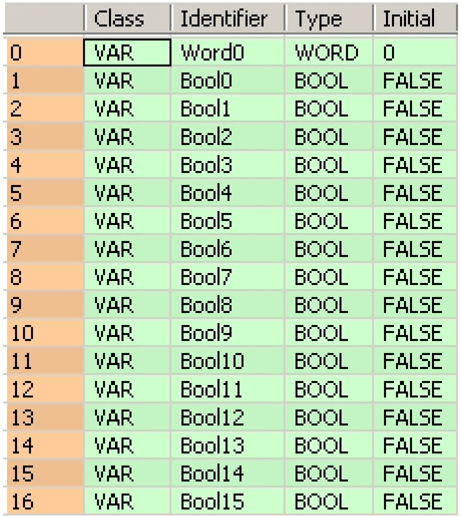
To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** BOOLS_TO_WORD **(see page 1318)**

Data types

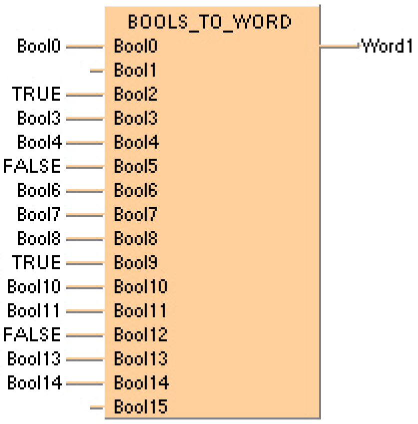
Variable	Data type	Function
BOOL0 ... BOOL15	BOOL	16 input variables of the data type BOOL
	WORD	output variable

POU header:



	Class	Identifier	Type	Initial
0	VAR	Word0	WORD	0
1	VAR	Bool0	BOOL	FALSE
2	VAR	Bool1	BOOL	FALSE
3	VAR	Bool2	BOOL	FALSE
4	VAR	Bool3	BOOL	FALSE
5	VAR	Bool4	BOOL	FALSE
6	VAR	Bool5	BOOL	FALSE
7	VAR	Bool6	BOOL	FALSE
8	VAR	Bool7	BOOL	FALSE
9	VAR	Bool8	BOOL	FALSE
10	VAR	Bool9	BOOL	FALSE
11	VAR	Bool10	BOOL	FALSE
12	VAR	Bool11	BOOL	FALSE
13	VAR	Bool12	BOOL	FALSE
14	VAR	Bool13	BOOL	FALSE
15	VAR	Bool14	BOOL	FALSE
16	VAR	Bool15	BOOL	FALSE

Body with and without EN/ENO:



DWORD_TO_WORD

DOUBLE WORD in WORD

Description DWORD_TO_WORD converts a value of the data type DOUBLE WORD into a value of the data type WORD.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** DWORD_TO_WORD **(see page 1319)**

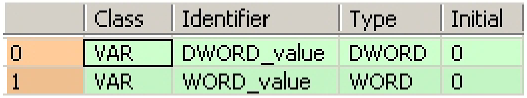
☞ **The first 16 bits of the input variable are assigned to the output variable.**

Data types

Data type	I/O	Function
DWORD	input	input data type
WORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

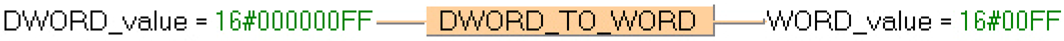


	Class	Identifier	Type	Initial
0	VAR	DWORD_value	DWORD	0
1	VAR	WORD_value	WORD	0

This example uses variables. You can also use a constant for the input variable.

Body **DWORD_value** of the data type DOUBLE WORD (32-bit) is converted into a value of the data type WORD (16-bit). The converted value is written into **WORD_value**.

LD



ST When programming with structured text, enter the following:

```
WORD_value:=DWORD_TO_WORD(DWORD_value);
```

INT_TO_WORD

INTEGER into WORD

Description INT_TO_WORD converts a value of the data type INT into a value of the data type WORD.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** INT_TO_WORD **(see page 1327)**

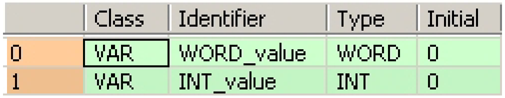
The bit combination of the input variable is assigned to the output variable.

Data types

Data type	I/O	Function
INT	input	input data type
WORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

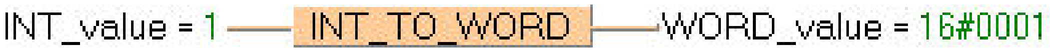


	Class	Identifier	Type	Initial
0	VAR	WORD_value	WORD	0
1	VAR	INT_value	INT	0

This example uses variables. You can also use a constant for the input variable.

Body **INT_value** of the data type INTEGER is converted into a value of the data type WORD. The result is written into **WORD_value**.

LD



ST When programming with structured text, enter the following:

```
WORD_value:=INT_TO_WORD(INT_value);
```

DINT_TO_WORD

DOUBLE INTEGER into WORD

Description DINT_TO_WORD converts a value of the data type DINT into a value of the data type WORD.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** DINT_TO_WORD **(see page 1319)**

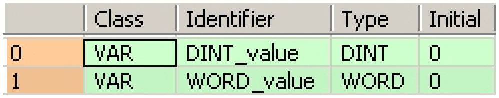
☞ **The first 16 bits of the input variable are assigned to the output variable.**

Data types

Data type	I/O	Function
DINT	input	input data type
WORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

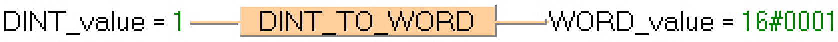


	Class	Identifier	Type	Initial
0	VAR	DINT_value	DINT	0
1	VAR	WORD_value	WORD	0

This example uses variables. You can also use a constant for the input variable.

Body **DINT_value** of the data type DOUBLE INTEGER (32-bit) is converted into a value of the data type WORD (16-bit). The converted value is written into **WORD_value**.

LD



ST When programming with structured text, enter the following:

```
WORD_value:=DINT_TO_WORD(DINT_value);
```

UINT_TO_WORD

Unsigned INTEGER into WORD

Description UINT_TO_WORD converts a value of the data type Unsigned INTEGER into a value of the data type WORD.

UINT_TO_WORD

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** UINT_TO_WORD **(see page 1332)**

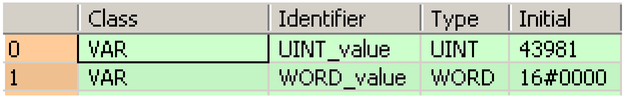
☞ **The first 16 bits of the input variable are assigned to the output variable.**

Data types

Data type	I/O	Function
UINT	Input	input data type
WORD	Output	conversion result

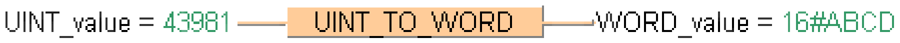
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	43981
1	VAR	WORD_value	WORD	16#0000

LD



ST
```
WORD_value:=UINT_TO_WORD(UINT_value);
```

UDINT_TO_WORD

Unsigned DOUBLE INTEGER into WORD

Description UDINT_TO_WORD converts a value of the data type Unsigned DOUBLE INTEGER into a value of the data type WORD.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** UDINT_TO_WORD **(see page 1332)**

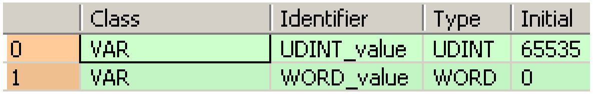
☞ **The first 16 bits of the input variable are assigned to the output variable.**

Data types

Data type	I/O	Function
UDINT	Input	input data type
WORD	Output	conversion result

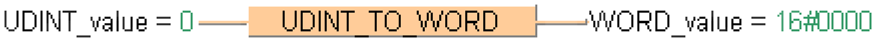
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	65535
1	VAR	WORD_value	WORD	0

LD



ST When programming with structured text, enter the following:

```
WORD_value := UDINT_TO_WORD(UDINT_value);
```

TIME_TO_WORD

TIME into WORD

Description TIME_TO_WORD converts a value of the data type TIME into a value of the data type WORD.

TIME_TO_WORD

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** TIME_TO_WORD **(see page** 1332**)**

Data types

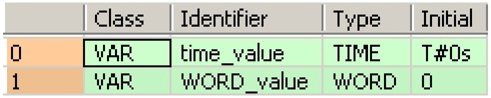
Data type	I/O	Function
TIME	input	input data type
WORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

Examples:

Input variable	Output variable
T#123.4s	1234
T#1.00s	16#0064

POU header All input and output variables used for programming this function have been declared in the POU header.

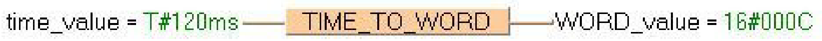


	Class	Identifier	Type	Initial
0	VAR	time_value	TIME	T#0s
1	VAR	WORD_value	WORD	0

This example uses variables. You can also use a constant for the input variable.

Body **Time_value** of the data type TIME is converted into a value of the data type WORD. The result will be written into the output variable **WORD_value**.

LD



ST When programming with structured text, enter the following:

```
WORD_value:=TIME_TO_WORD(time_value);
```

STRING_TO_WORD

STRING (hexadecimal format) to WORD

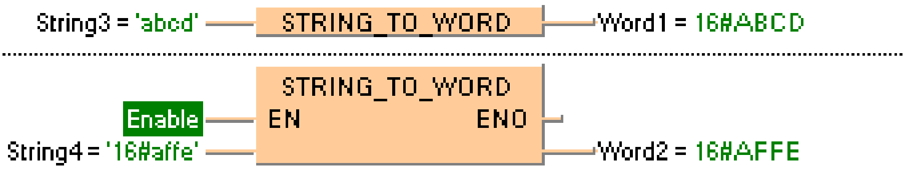
Description This function converts a STRING in hexadecimal format to a value of the data type WORD.

STRING_TO_WORD

Thereby the attached string is first converted to a value of the data type STRING[32]. Finally this is converted to a value of the data type WORD via a sub-program of approx. 270 steps that is also used in the functions STRING_TO_INT, STRING_TO_WORD, STRING_TO_DINT and STRING_TO_DWORD.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Example with and without EN/ENO:



Permissible format:

'[Space][Hexadecimal numbers][Space]' e.g. '   afFE   '

Permissible characters:

Space	All characters except for "+" (plus), "-" (minus) and all hexadecimal numbers
Hexadecimal numbers	Hexadecimal numbers in the ranges "0 - 9", "A - F" or "a - f".

The analysis ends with the first non-hexadecimal number.

PLC types **Availability of** STRING_TO_WORD **(see page** 1331**)**

Data types

Data type	Comment
STRING	input variable
WORD	output variable

STRING_TO_WORD _STEPSAVER

STRING (Hexadecimal Format right-justified) to WORD

Description This function converts the string with the maximum possible number of characters that are right aligned in hexadecimal format to a value of the data type WORD.

STRING_TO_WORD_STEPSAVER

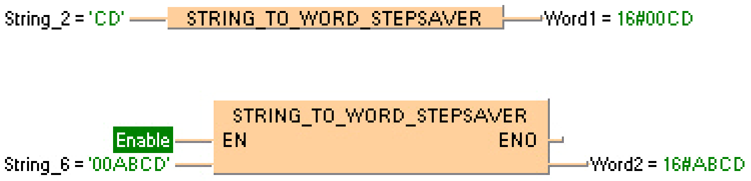
Examples

Input	Defined as	Results in
'D'	STRING[1]	16#D
'CD'	STRING[2]	16#CD
'BCD'	STRING[3]	16#BCD
'ABCD'	STRING[4]	16#ABCD
'0ABCD'	STRING[5]	16#ABCD
'00ABCD'	STRING[6]	16#ABCD

The basic instruction F72_A2HEX (see page 624) is used. The PLC delivers an operation error especially when a character appears that is not a hexadecimal number “0 - 9" or "A-F”.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Example



String_2 = 'CD' STRING_TO_WORD_STEPSAVER Word1 = 16#00CD

Enable EN STRING_TO_WORD_STEPSAVER ENO

String_6 = '00ABCD' Word2 = 16#ABCD

Data types

Data type	Comment
STRING	Input variable
WORD	Output variable

Acceptable Format for STRING[4]:

'Hex1Hex2Hex3Hex4' e.g. perhaps 'AFFE'

Acceptable characters:

Hex1 to Hex4	Hexadecimal numbers in the range "0 - 9“ or "A - F“ (not "a - f").

PLC types **Availability of STRING_TO_WORD_STEPSAVER (see page 1331)**

BOOL_TO_DWORD

BOOL into DOUBLE WORD

Description BOOL_TO_DWORD converts a value of the data type BOOL into a value of the data type DWORD.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

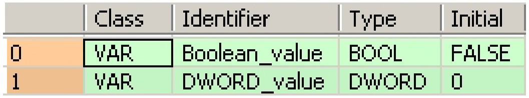
PLC types **Availability of** BOOL_TO_DWORD **(see page 1318)**

Data types

Data type	I/O	Function
BOOL	input	input data type
DWORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

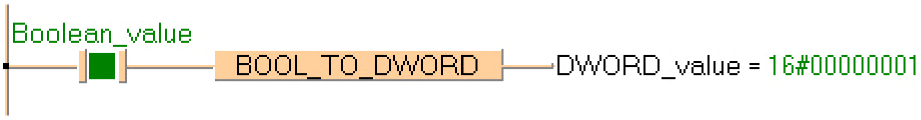


	Class	Identifier	Type	Initial
0	VAR	Boolean_value	BOOL	FALSE
1	VAR	DWORD_value	DWORD	0

In this example the input variable (**Boolean_value**) has been declared. Instead, you may enter a constant directly at the input contact of a function.

Body The **Boolean_value** of the data type BOOL is converted into a value of the data type DOUBLE INTEGER. The converted value is written into **DWORD_value**.

LD



ST When programming with structured text, enter the following:

```
IF Boolean_value THEN
        DWORD_value:=BOOL_TO_DWORD(Boolean_value);
END_IF;
```

BOOL32_TO_DWORD

BOOL32 to DOUBLE WORD

Description This function copies a variable of the special data type BOOL32 (see page 55) (an array with 32 elements of the data type BOOL or a DUT of 32 members of the data type BOOL) at the input to the data type DWORD at the output.

BOOL32_TO_DWORD

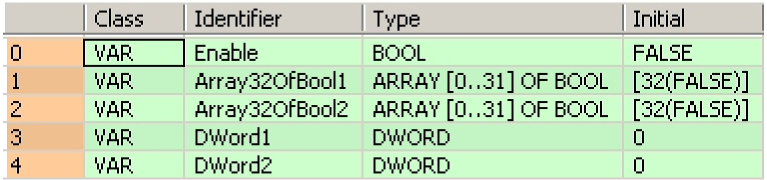
To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** BOOL32_TO_DWORD **(see page 1318)**

Data types

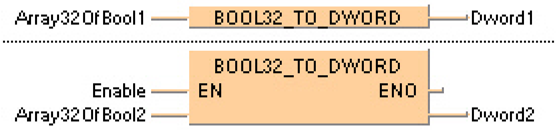
Data type	Comment
ARRAY of BOOL	ARRAY with 32 elements
DWORD	output variable

POU header:



	Class	Identifier	Type	Initial
0	VAR	Enable	BOOL	FALSE
1	VAR	Array32OfBool1	ARRAY [0..31] OF BOOL	[32(FALSE)]
2	VAR	Array32OfBool2	ARRAY [0..31] OF BOOL	[32(FALSE)]
3	VAR	DWord1	DWORD	0
4	VAR	DWord2	DWORD	0

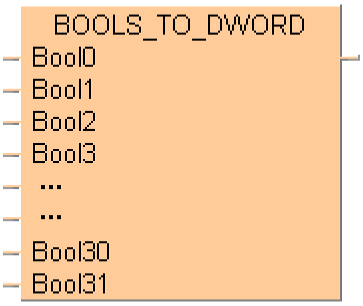
Body with and without EN/ENO:



BOOLS_TO_DWORD

32 Variables of the data type BOOL to DWORD

Description This function converts 32 values of the data type BOOL bit-wise to a value of the data type DWORD.



The inputs Bool0 to Bool31 need not be allocated in LD or FBD, or used explicitly in the ST editor's formal list of parameters. Such unused inputs are assumed to be FALSE. No program code is generated for these inputs (or for any input allocated with the constants TRUE or FALSE). Program code is only generated for inputs to which a variable is allocated.

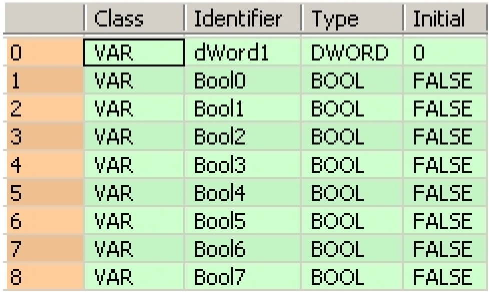
To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** BOOLS_TO_DWORD **(see page 1318)**

Data types

Variable	Data type	Function
BOOL0 ... BOOL31	BOOL	32 input variables of the data type BOOL
	DWORD	output variable

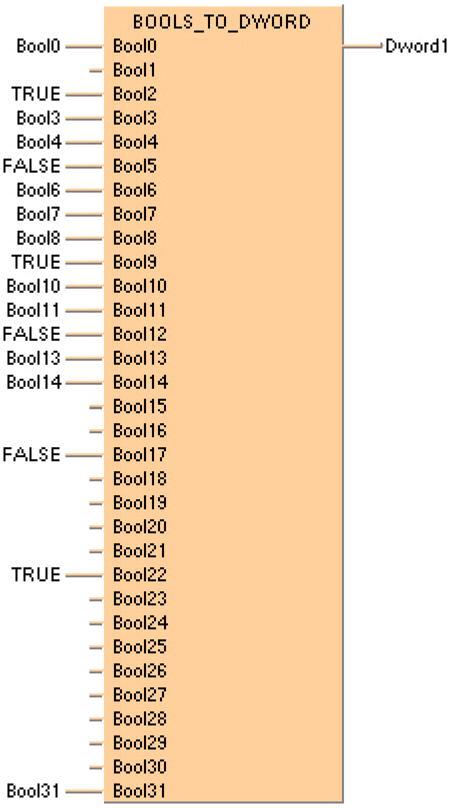
POU header:



	Class	Identifier	Type	Initial
0	VAR	dWord1	DWORD	0
1	VAR	Bool0	BOOL	FALSE
2	VAR	Bool1	BOOL	FALSE
3	VAR	Bool2	BOOL	FALSE
4	VAR	Bool3	BOOL	FALSE
5	VAR	Bool4	BOOL	FALSE
6	VAR	Bool5	BOOL	FALSE
7	VAR	Bool6	BOOL	FALSE
8	VAR	Bool7	BOOL	FALSE

etc. to Bool31

Body with and without EN/ENO:



WORD_TO_DWORD

WORD in DOUBLE WORD

Description WORD_TO_DWORD converts a value of the data type WORD into a value of the data type DWORD.

WORD_TO_DWORD

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** WORD_TO_DWORD **(see page** 1333**)**

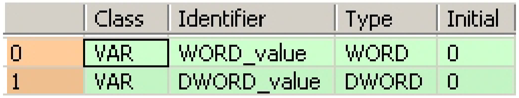
☞ **The bit combination of WORD_value is assigned to DWORD_value.**

Data types

Data type	I/O	Function
WORD	input	input data type
DWORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

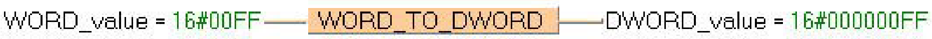


	Class	Identifier	Type	Initial
0	VAR	WORD_value	WORD	0
1	VAR	DWORD_value	DWORD	0

This example uses variables. You can also use a constant for the input variable.

Body **WORD_value** of the data type WORD is converted into a value of the data type DOUBLE WORD. The result will be written into **DWORD_value**.

LD



ST When programming with structured text, enter the following:

```
DWORD_value:=WORD_TO_DWORD(WORD_value);
```

INT_TO_DWORD

INTEGER into DOUBLE WORD

Description INT_TO_DWORD converts a value of the data type INT into a value of the data type DWORD.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

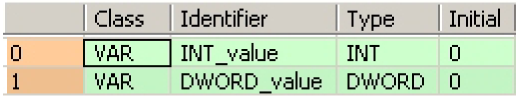
PLC types **Availability of** INT_TO_DWORD **(see page 1327)**

Data types

Data type	I/O	Function
INT	input	input data type
DWORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

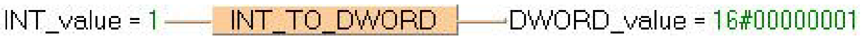


	Class	Identifier	Type	Initial
0	VAR	INT_value	INT	0
1	VAR	DWORD_value	DWORD	0

This example uses variables. You can also use a constant for the input variable.

Body **INT_value** of the data type INTEGER is converted into a value of the data type DOUBLE WORD (32-bit). The result is written into **DWORD_value**.

LD



ST When programming with structured text, enter the following:

```
DWORD_value:=INT_TO_DWORD(INT_value);
```

DINT_TO_DWORD

DOUBLE INTEGER into DOUBLE WORD

Description DINT_TO_DWORD converts a value of the data type DINT into a value of the data type DWORD.

DINT_TO_DWORD

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** DINT_TO_DWORD **(see page** 1319**)**

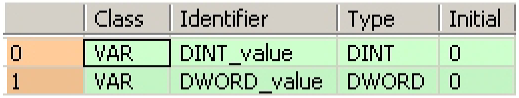
☞ **The bit combination of the input variable is assigned to the output variable.**

Data types

Data type	I/O	Function
DINT	input	input data type
DWORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

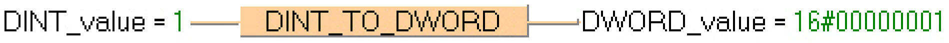


	Class	Identifier	Type	Initial
0	VAR	DINT_value	DINT	0
1	VAR	DWORD_value	DWORD	0

This example uses variables. You can also use a constant for the input variable.

Body **DINT_value** of the data type DOUBLE INTEGER is converted into a value of the data type DOUBLE WORD. The converted value is written into **DWORD_value**.

LD



ST When programming with structured text, enter the following:

```
DWORD_value:=DINT_TO_DWORD(DINT_value);
```

UINT_TO_DWORD

Unsigned INTEGER into DOUBLE WORD

Description UINT_TO_DWORD converts a value of the data type Unsigned INTEGER into a value of the data type DWORD.

UINT_TO_DWORD

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

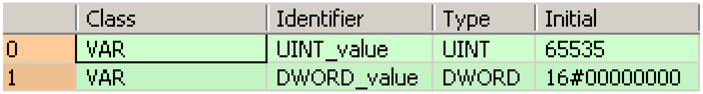
PLC types **Availability of** UINT_TO_DWORD **(see page** 1332**)**

Data types

Data type	I/O	Function
UINT	Input	input data type
DWORD	Output	conversion result

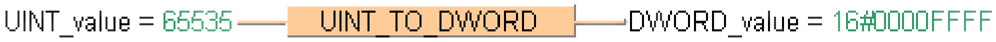
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	65535
1	VAR	DWORD_value	DWORD	16#00000000

LD



UINT_value = 65535 — UINT_TO_DWORD — DWORD_value = 16#0000FFFF

ST
```
DWORD_value:=  UINT_TO_DWORD(UINT_value);
```

UDINT_TO_DWORD

Unsigned DOUBLE INTEGER into DOUBLE WORD

Description UDINT_TO_DWORD converts a value of the data type Unsigned DOUBLE INTEGER into a value of the data type DWORD.

UDINT_TO_DWORD

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

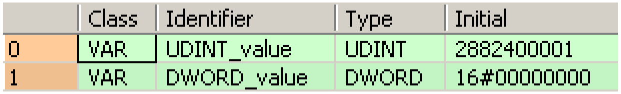
PLC types **Availability of** UDINT_TO_DWORD **(see page** 1332**)**

Data types

Data type	I/O	Function
UDINT	Input	input data type
DWORD	Output	conversion result

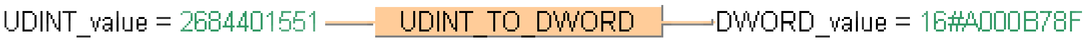
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	2882400001
1	VAR	DWORD_value	DWORD	16#00000000

LD



ST When programming with structured text, enter the following:

```
DWORD_value := UDINT_TO_DWORD(UDINT_value);
```

REAL_TO_DWORD REAL into DOUBLE WORD

Description REAL_TO_DWORD moves bitset information of a REAL variable to a DWORD variable. The same functionality can be obtained using DWORD_OVERLAPPING_DUT.

REAL_TO_DWORD

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

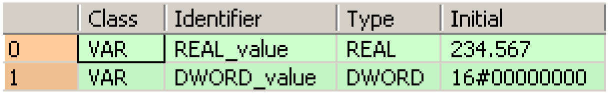
PLC types **Availability of REAL_TO_DWORD (see page 1330)**

Data types

Data type	I/O	Function
REAL	Input	input data type
DWORD	Output	conversion result

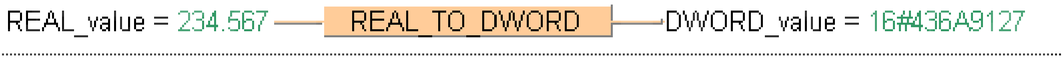
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	REAL_value	REAL	234.567
1	VAR	DWORD_value	DWORD	16#00000000

LD


REAL_value = 234.567 — REAL_TO_DWORD — DWORD_value = 16#436A9127

ST When programming with structured text, enter the following:

```
DWORD_value := REAL_TO_DWORD(REAL_value);
```

TIME_TO_DWORD

TIME into DOUBLE WORD

Description TIME_TO_DWORD converts a value of the data type TIME into a value of the data type DWORD. The time 10ms corresponds to the value 1, e.g. an input value of T#1s is converted to the value 100 (16#64).

TIME_TO_DWORD

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

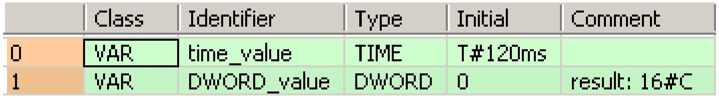
PLC types **Availability of** TIME_TO_DWORD **(see page** 1331**)**

Data types

Data type	**I/O**	**Function**
TIME	input	input data type
DWORD	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

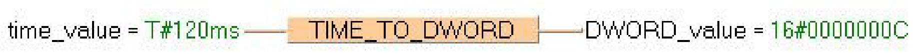


	Class	Identifier	Type	Initial	Comment
0	VAR	time_value	TIME	T#120ms	
1	VAR	DWORD_value	DWORD	0	result: 16#C

This example uses variables. You can also use a constant for the input variable.

Body **time_value** of the data type TIME is converted to value of the data type DWORD and written into the output variable **DWORD_value**.

LD



time_value = T#120ms TIME_TO_DWORD DWORD_value = 16#0000000C

ST When programming with structured text, enter the following:

```
DWORD_value:=TIME_TO_DWORD(time_value);
```

STRING_TO_DWORD

STRING (Hexadecimal Format) to DOUBLE WORD

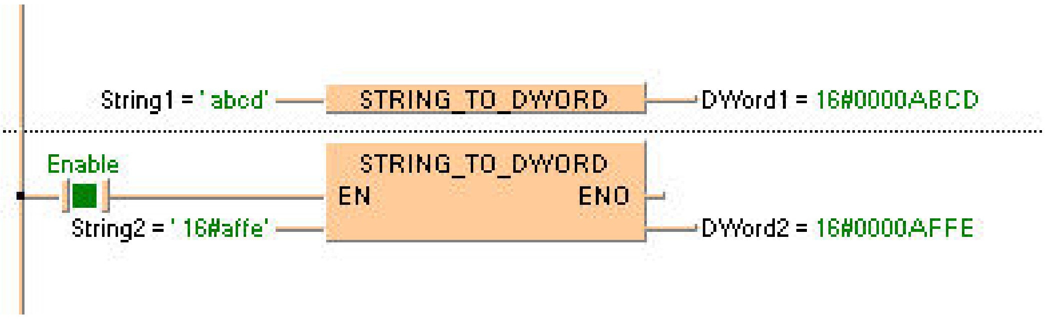
Description This function converts a string in hexadecimal formal to a value of the data type DWORD.

At first the string is converted to a value of the data type STRING[32]. Finally this is converted to a value of the data type DWORD in a subprogram of approximately 270 steps, which is also used by the functions STRING_TO_INT, STRING_TO_WORD, STRING_TO_DINT and STRING_TO_DWORD.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

See also: STRING_TO_DWORD_STEPSAVER

Example with and without EN/ENO:



Acceptable Format:

_'[Space][Hexadecimal number][Space]' e.g. perhaps '   **afFE**   '

Acceptable characters:

Space	Space " "
Signs	Plus "+" and minus "-"
Hexadecimal numbers	Hexadecimal numbers in the range "0 - 9" or "A - F" or "a - f".

The analysis ends with the first non-hexadecimal number.

PLC types **Availability of** STRING_TO_DWORD **(see page 1331)**

Data types

Data type	Comment
STRING	Input variable
DWORD	Output variable

STRING_TO_DWORD _STEPSAVER

STRING (Hexadecimal Format right-justified) to DOUBLE WORD

Description This function converts the string with the maximum possible number of characters that are right aligned in hexadecimal format to a value of the data type DWORD.

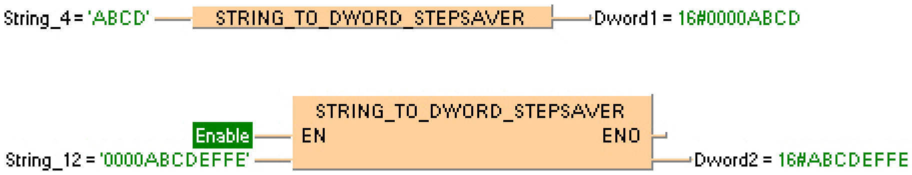
Explanation

Input	Defined as	Results in
'FE'	STRING[2]	16#FE
'EFFE'	STRING[4]	16#EFFE
'CDEFFE'	STRING[6]	16#CDEFFE
'ABCDEFFE'	STRING[8]	16#ABCDEFFE
'00ABCDEFFE'	STRING[10]	16#ABCDEFFE

The basic instruction **F72_A2HEX** (see page 624) is used. The PLC delivers an operation error especially when a character appears that is not a hexadecimal number “0 - 9" or "A - F”.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Example


String_4 = 'ABCD'
STRING_TO_DWORD_STEPSAVER
Dword1 = 16#0000ABCD
Enable
STRING_TO_DWORD_STEPSAVER
EN
ENO
String_12 = '0000ABCDEFFE'
Dword2 = 16#ABCDEFFE

Data types

Data type	Comment
STRING	Input variable
DWORD	Output variable

Acceptable Format for STRING[8]:

'Hex1Hex2Hex3Hex4Hex5Hex6Hex7Hex8' e.g. perhaps '001AAFFE'

Acceptable characters:

Hex1 to Hex8	Hexadecimal numbers in the range "0 - 9“ or "A - F“ (not "a - f").

PLC types **Availability of** STRING_TO_DWORD_STEPSAVER **(see page 1331)**

BOOL_TO_INT

BOOL into INTEGER

Description BOOL_TO_INT converts a value of the data type BOOL into a value of the data type INT.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

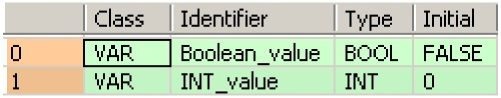
PLC types **Availability of** BOOL_TO_INT **(see page 1318)**

Data types

Data type	I/O	Function
BOOL	input	input data type
INT	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

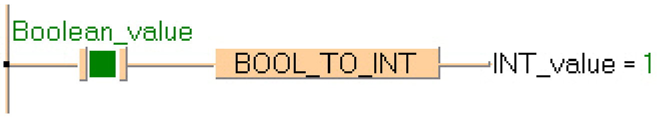


	Class	Identifier	Type	Initial
0	VAR	Boolean_value	BOOL	FALSE
1	VAR	INT_value	INT	0

In this example the input variable (**Boolean_value**) has been declared. Instead, you may enter a constant directly at the input contact of a function.

Body The **Boolean_value** of the data type BOOL is converted into a value of the data type INTEGER. The converted value is written into **INT_value**.

LD



ST When programming with structured text, enter the following:

```
IF Boolean_value THEN
    INT_value:=BOOL_TO_INT(Boolean_value);
END_IF;
```

WORD_TO_INT

WORD value in INTEGER

Description WORD_TO_INT converts a value of the data type WORD into a value of the data type INT.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** WORD_TO_INT **(see page 1333)**

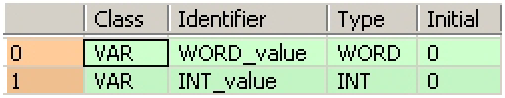
☞ **The bit combination of WORD_value is assigned to INT_value.**

Data types

Data type	I/O	Function
WORD	input	input data type
INT	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

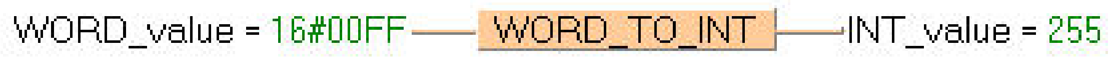


	Class	Identifier	Type	Initial
0	VAR	WORD_value	WORD	0
1	VAR	INT_value	INT	0

This example uses variables. You can also use a constant for the input variable.

Body **WORD_value** of the data type WORD is converted into a value of the data type INTEGER. The result will be written into **INT_value**.

LD



ST When programming with structured text, enter the following:

```
INT_value:=WORD_TO_INT(WORD_value);
```

WORD_BCD_TO_INT

Binary WORD value into INTEGER

Description WORD_BCD_TO_INT converts a binary coded BCD value of WORD into binary values of type INT.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

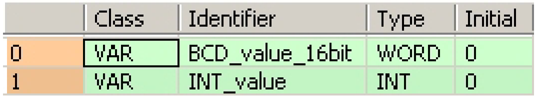
PLC types **Availability of** WORD_BCD_TO_INT **(see page 1333)**

Data types

Data type	I/O	Function
WORD_BCD	Input	input data type
INT	Output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	BCD_value_16bit	WORD	0
1	VAR	INT_value	INT	0

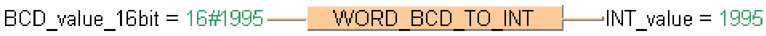
This example uses variables. You can also use a constant for the input variable.

BCD constants can be expressed in Control FPWIN Pro as follows:

2#0001100110010101 or
16#1995

Body **BCD_value_16bit** of the data type WORD is converted into an INTEGER value. The converted value is written into output variable **INT_value**.

LD



ST When programming with structured text, enter the following:

```
INT_value:=WORD_BCD_TO_INT(BCD_value_16bit);
```

DWORD_TO_INT

DOUBLE WORD in INTEGER

Description DWORD_TO_INT converts a value of the data type DWORD into a value of the data type INT.

DWORD_TO_INT

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** DWORD_TO_INT **(see page** 1319**)**

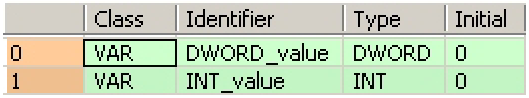
☞ **The first 16 bits of the input variable are assigned to the output variable.**

Data types

Data type	I/O	Function
DWORD	input	input data type
INT	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

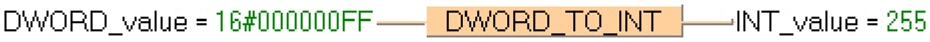


	Class	Identifier	Type	Initial
0	VAR	DWORD_value	DWORD	0
1	VAR	INT_value	INT	0

In this example the input variable (**DWORD _value**) has been declared. Instead, you may enter a constant directly at the input contact of a function.

Body **DWORD_value** of the data type DOUBLE WORD (32-bit) is converted into an INTEGER value (16-bit). The converted value is written into **INT_value**.

LD



ST When programming with structured text, enter the following:

```
INT_value:=DWORD_TO_INT(DWORD_value);
```

DINT_TO_INT

DOUBLE INTEGER into INTEGER

Description DINT_TO_INT converts a value of the data type DINT into a value of the data type INT.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** DINT_TO_INT **(see page 1319)**

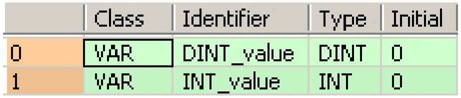
☞ **The value of the input variable should be between -32768 and 32767.**

Data types

Data type	I/O	Function
DINT	input	input data type
INT	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

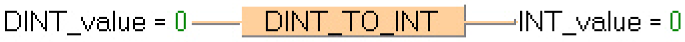


	Class	Identifier	Type	Initial
0	VAR	DINT_value	DINT	0
1	VAR	INT_value	INT	0

This example uses variables. You can also use a constant for the input variable.

Body **DINT_value** of the data type DOUBLE INTEGER (32-bit) is converted into a value of the data type INTEGER (16-bit). The converted value is written into **INT_value**.

LD



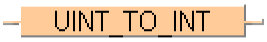
ST When programming with structured text, enter the following:

```
INT_value:=DINT_TO_INT(DINT_value);
```

UINT_TO_INT

Unsigned DOUBLE INTEGER into INTEGER

Description UINT_TO_INT converts a value of the data type Unsigned INTEGER into a value of the data type INT.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

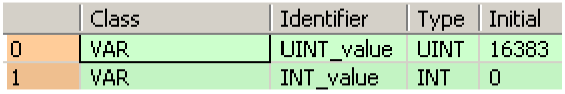
PLC types **Availability of** UINT_TO_INT **(see page** 1332**)**

Data types

Data type	I/O	Function
UINT	Input	input data type
INT	Output	conversion result

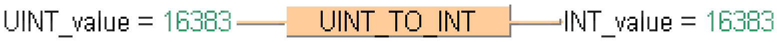
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	16383
1	VAR	INT_value	INT	0

LD



ST
```
INT_value:=  UINT_TO_INT(UINT_value);
```

UDINT_TO_INT

Unsigned DOUBLE INTEGER into INTEGER

Description UDINT_TO_INT converts a value of the data type Unsigned DOUBLE INTEGER into a value of the data type INT.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

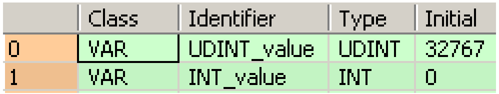
PLC types **Availability of UDINT_TO_INT (see page 1332)**

Data types

Data type	I/O	Function
UDINT	Input	input data type
INT	Output	conversion result

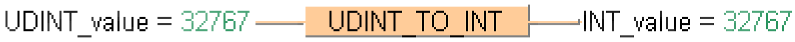
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	32767
1	VAR	INT_value	INT	0

LD



```
UDINT_value = 32767 ——UDINT_TO_INT——INT_value = 32767
```

ST When programming with structured text, enter the following:

```
INT_value := UDINT_TO_INT(UDINT_value);
```

REAL_TO_INT

REAL into INTEGER

Description REAL_TO_INT converts a value of the data type REAL into a value of the data type INTEGER.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

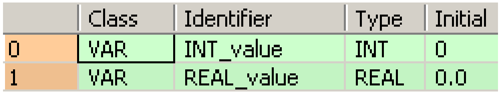
PLC types **Availability of** REAL_TO_INT **(see page 1330)**

Data types

Data type	I/O	Function
REAL	input	input data type
INT	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

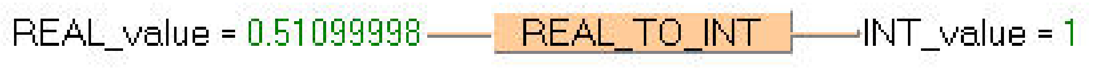


	Class	Identifier	Type	Initial
0	VAR	INT_value	INT	0
1	VAR	REAL_value	REAL	0.0

This example uses variables. You can also use a constant for the input variable.

Body **REAL_value** of the data type REAL is converted into a value of the data type INTEGER. The converted value is stored in **INT_value**.

LD



ST When programming with structured text, enter the following:

```
INT_value:= REAL_TO_INT(REAL_value);
```

TRUNC_TO_INT

Truncate (cut off) decimal digits of REAL input variable, convert to INTEGER

Description TRUNC_TO_INT cuts off the decimal digits of a REAL number and delivers an output variable of the data type INTEGER.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** TRUNC_TO_INT **(see page 1332)**

☞

- **If the decimal digits are cut off, positive numbers will be decreased towards zero and negative numbers will be increased towards zero.**
- **The first 16 bits of the input variable are assigned to the output variable.**

Data types

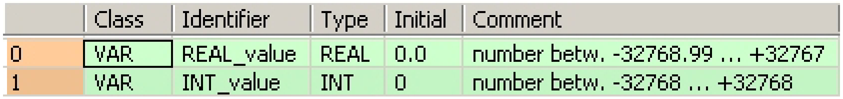
Data type	I/O	Function
REAL	input	input data type
INT	output	conversion result

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input variable does not have the data type REAL
R9008	%MX0.900.8	for an instant	▪ output variable is greater than a 16-bit INTEGER
R9009	%MX0.900.9	for an instant	▪ output variable is zero

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

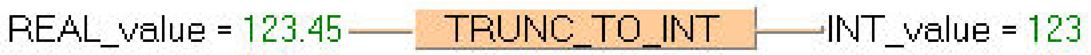


	Class	Identifier	Type	Initial	Comment
0	VAR	REAL_value	REAL	0.0	number betw. -32768.99 ... +32767
1	VAR	INT_value	INT	0	number betw. -32768 ... +32768

This example uses variables. You can also use a constant for the input variable.

Body The decimal digits of **REAL_value** are cut off. The result is stored as a 16-bit INTEGER in **INT_value**.

LD



ST When programming with structured text, enter the following:

```
INT_value:=TRUNC_TO_INT(REAL_value);
```

TIME_TO_INT

TIME into INTEGER

Description TIME_TO_INT converts a value of the data type TIME into a value of the data type INT.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

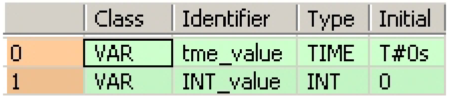
PLC types **Availability of** TIME_TO_INT **(see page 1331)**

Data types

Data type	I/O	Function
TIME	input	input data type
INT	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

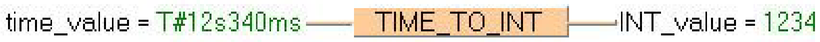


	Class	Identifier	Type	Initial
0	VAR	tme_value	TIME	T#0s
1	VAR	INT_value	INT	0

This example uses variables. You can also use a constant for the input variable.

Body **Time_value** of the data type TIME is converted into a value of the data type INTEGER. The result will be written into the output variable **INT_value**.

LD



ST When programming with structured text, enter the following:

```
INT_value:=TIME_TO_INT(time_value);
```

STRING_TO_INT

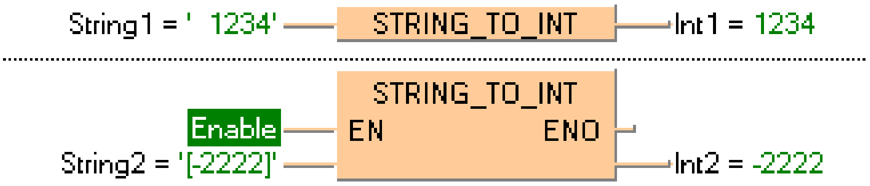
STRING (decimal format) to INTEGER

Description This function converts a STRING in decimal format to a value of the data type INT.

Thereby the attached string is first converted to a value of the data type STRING[32]. Finally this is converted to a value of the data type INT via a sub-programm of approx. 270 steps that is also used in the functions STRING_TO_INT, STRING_TO_WORD, STRING_TO_DINT and STRING_TO_DWORD.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Example



Permissible format:

'[Space][Sign][Decimal numbers][Space]' e.g. '   123456   '

Permissible characters:

Space	All characters except for "+“ (plus), "-" (minus) and all decimal numbers
Sign	"+“ (plus), "-" (minus)
Decimal numbers	Decimal numbers "0 - 9“

The analysis ends with the first non-decimal number.

PLC types **Availability of** STRING_TO_INT **(see page 1331)**

Data types

Data type	Comment
STRING	input variable
INT	output variable

STRING_TO_INT_STEPSAVER

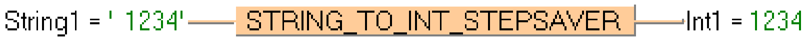
STRING (Decimal Format right-justified) to INTEGER

Description This function converts a right-justifed decimal number in a string to a value of the data type INT.

The basic instruction F76_A2BIN (see page 637) with approx. 7 steps is used. The PLC delivers an operation error especially when a character appears that is not a decimal number “0 - 9”, not a “+” or “-“ or not a space.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Example



Acceptable Format:

'[Space][Sign][Decimal number]' e.g. '␣123456'

Acceptable characters:

Space	Space “␣“
Signs	Plus "+“ and minus "-"
Decimal Number	Decimal numbers "0“ - "9“

PLC types **Availability of** STRING_TO_INT_STEPSAVER **(see page 1331)**

Data types

Data type	Comment
STRING	Input variable
INT	Output variable

BOOL_TO_UINT

BOOL into Unsigned INTEGER

Description BOOL_TO_UINT converts a value of the data type BOOL into a value of the data type Unsigned INTEGER.

BOOL_TO_UINT

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

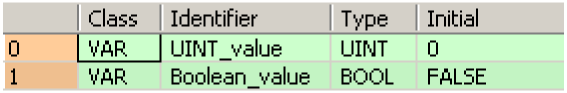
PLC types **Availability of** BOOL_TO_UINT **(see page 1318)**

Data types

Data type	I/O	Function
BOOL	Input	input data type
UINT	Output	conversion result

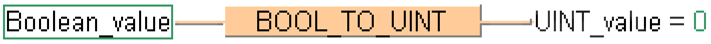
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	0
1	VAR	Boolean_value	BOOL	FALSE

LD



```
ST UINT_value:=  BOOL_TO_UINT(Boolean_value);
```

WORD_TO_UINT

WORD to Unisgned INTEGER

Description WORD_TO_UINT converts a value of the data type WORD into a value of the data type Unsigned INTEGER.

WORD_TO_UINT

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

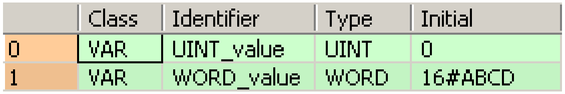
PLC types **Availability of** WORD_TO_UINT **(see page 1333)**

Data types

Data type	I/O	Function
WORD	Input	input data type
UDINT	Output	conversion result

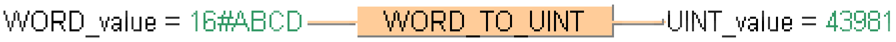
Example In this example the function is programmed in ladder diagram (LD).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	0
1	VAR	WORD_value	WORD	16#ABCD

LD



WORD_value = 16#ABCD — WORD_TO_UINT — UINT_value = 43981

WORD_BCD_TO_UINT

Binary coded WORD value in Unsigned INTEGER

Description WORD_BCD_TO_UINT converts a binary coded value of the data type WORD into a value of the data type Unsigned INTEGER.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

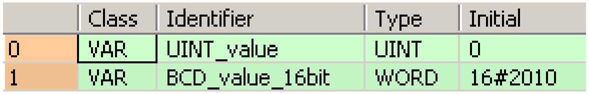
PLC types **Availability of** WORD_BCD_TO_UINT **(see page** 1333**)**

Data types

Data type	I/O	Function
WORD_BCD	Input	input data type
UINT	Output	conversion result

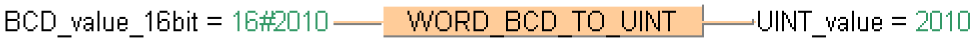
Example In this example the function is programmed in ladder diagram (LD).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	0
1	VAR	BCD_value_16bit	WORD	16#2010

LD



DWORD_TO_UINT

DOUBLE WORD into Unsigned INTEGER

Description DWORD_TO_UINT converts a value of the data type DWORD into a value of the data type Unsigned INTEGER.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

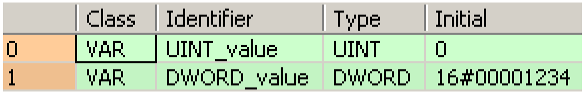
PLC types **Availability of** DWORD_TO_UINT **(see page 1319)**

Data types

Data type	I/O	Function
DWORD	Input	input data type
UINT	Output	conversion result

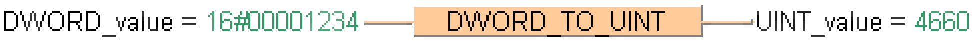
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	0
1	VAR	DWORD_value	DWORD	16#00001234

LD



ST
```
UINT_value:=  DWORD_TO_UINT(DWORD_value);
```

INT_TO_UINT

INTEGER to Unsigned INTEGER

Description INT_TO_UINT converts a value of the data type INT into a value of the data type Unsigned INTEGER.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

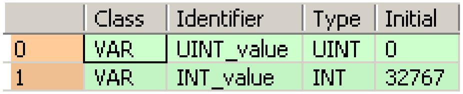
PLC types **Availability of** INT_TO_UINT **(see page 1327)**

Data types

Data type	I/O	Function
INT	Input	input data type
UINT	Output	conversion result

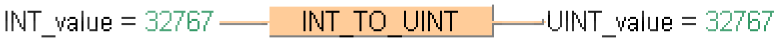
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	0
1	VAR	INT_value	INT	32767

LD



```
INT_value = 32767 ── INT_TO_UINT ── UINT_value = 32767
```

ST
```
UINT_value:=  INT_TO_UINT(INT_value);
```

DINT_TO_UINT

DOUBLE INTEGER into Unsigned INTEGER

Description DINT_TO_UINT converts a value of the data type DINT into a value of the data type Unsigned INTEGER.

DINT_TO_UINT

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

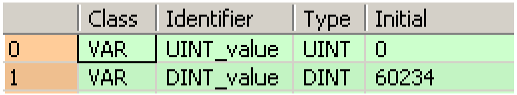
PLC types **Availability of** DINT_TO_UINT **(see page 1319)**

Data types

Data type	I/O	Function
DINT	Input	input data type
UINT	Output	conversion result

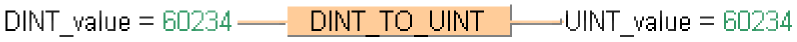
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	0
1	VAR	DINT_value	DINT	60234

LD



ST
```
UINT_value:=  DINT_TO_UINT(DINT_value);
```

UDINT_TO_UINT

Unsigned DOUBLE INTEGER into Unsigned INTEGER

Description UDINT_TO_UINT converts a value of the data type Unsigned DOUBLE INTEGER into a value of the data type Unsigned INTEGER.

UDINT_TO_UINT

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

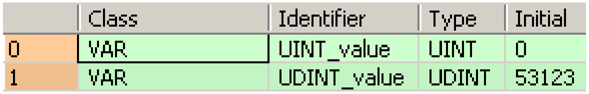
PLC types **Availability of** UDINT_TO_UINT **(see page 1332)**

Data types

Data type	I/O	Function
UDINT	Input	input data type
UINT	Output	conversion result

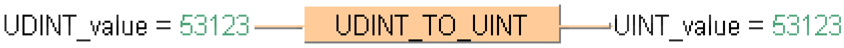
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	0
1	VAR	UDINT_value	UDINT	53123

LD



ST
```
UINT_value:=  UDINT_TO_UINT(UDINT_value);
```

REAL_TO_UINT

REAL into Unsigned INTEGER

Description REAL_TO_UINT converts a value of the data type REAL into a value of the data type Unsigned INTEGER.

REAL_TO_UINT

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

See also: TRUNC_TO_UINT (see page 164)

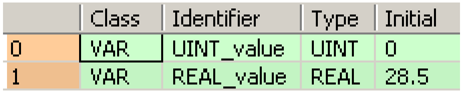
PLC types **Availability of** REAL_TO_UINT **(see page** 1330**)**

Data types

Data type	I/O	Function
REAL	Input	input data type
UINT	Output	conversion result

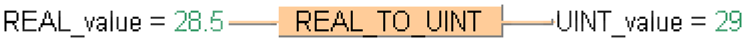
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	0
1	VAR	REAL_value	REAL	28.5

LD



ST
```
UINT_value:=  REAL_TO_UINT(REAL_value);
```

TRUNC_TO_UINT

Truncate (cut off) decimal digits of REAL input variable, convert to UNSIGNED INTEGER

Description TRUNC_TO_UINT cuts off any digits following the decimal of a REAL number and delivers an output variable of the data type Unsigned INTEGER.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** TRUNC_TO_UINT **(see page 1332)**

- **If the decimal digits are cut off, positive numbers will be decreased towards zero and negative numbers will be increased towards zero.**

Data types

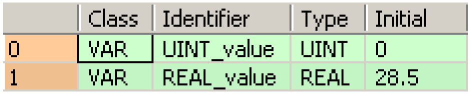
Data type	I/O	Function
REAL	Input	input data type
INT	Output	conversion result

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the input variable is not of the data type REAL
R9008	%MX0.900.8	for an instant	▪ the output variable is greater than a 16-bit INTEGER
R9009	%MX0.900.9	for an instant	▪ the output variable is zero

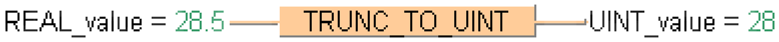
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	0
1	VAR	REAL_value	REAL	28.5

LD



ST
```
UINT_value:=  TRUNC_TO_UINT(REAL_value);
```

STRING_TO_UINT

STRING (decimal format) to Unsigned INTEGER

Description STRING_TO_UINT converts a string in decimal formal to a value of the data type Unsigned INTEGER.

See also: STRING_TO_UINT_STEPSAVER (see page 166)

First, the string is converted to a value of the data type STRING[32], which is subsequently converted to a value of the data type UINT in a subprogram with approximately 270 steps. The subprogram is also used by the functions STRING_TO_INT, STRING_TO_WORD, STRING_TO_UDINT and STRING_TO_DWORD.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Acceptable Format:

'[Space][Sign][Decimal number][Space]', e.g. '   123456   '

Acceptable characters:

Space	Space “ “
Signs	Plus “+” and minus “-“
Decimal numbers	Decimal numbers "0“ - "9“

The analysis ends with the first non-decimal number.

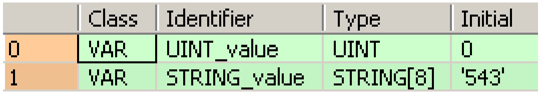
PLC types **Availability of** STRING_TO_UINT **(see page 1331)**

Data types

Data type	Comment
STRING	Input
UINT	Output

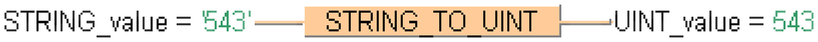
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	0
1	VAR	STRING_value	STRING[8]	'543'

LD



STRING_value = '543' — STRING_TO_UINT — UINT_value = 543

ST
```
UINT_value:=  STRING_TO_UINT(STRING_value);
```

STRING_TO_UINT_ STEPSAVER

STRING (Decimal Format right-justified) to Unsigned INTEGER

Description STRING_TO_UINT_STEPSAVER converts a right-justifed decimal number in a string to a value of the data type Unsigned INTEGER.

STRING_TO_UINT_STEPSAVER

The basic instruction F76_A2BIN (see page 637) with approx. 7 steps is used. The PLC issues an operation error especially if anything other than acceptable characters are used (see the following table "Acceptable characters").

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Acceptable Format:

'[Space][Sign][Decimal number]', e.g. '␣123456'

Acceptable characters:

Space	Space "␣"
Signs	Plus "+" and minus "-"
Decimal Number	Decimal numbers "0" - "9"

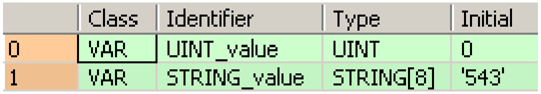
PLC types **Availability of** STRING_TO_UINT_STEPSAVER **(see page 1331)**

Data types

Data type	Comment
STRING	Input
UINT	Output

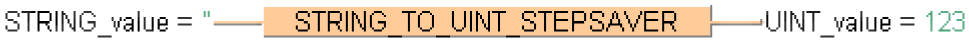
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	0
1	VAR	STRING_value	STRING[8]	'543'

LD


STRING_value = '' STRING_TO_UINT_STEPSAVER UINT_value = 123

ST
```
UINT_value:=  STRING_TO_UINT_STEPSAVER(STRING_value);
```

BOOL_TO_DINT

BOOL into DOUBLE INTEGER

Description BOOL_TO_DINT converts a value of the data type BOOL into a value of the data type DINT.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

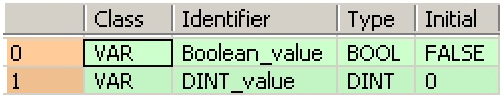
PLC types **Availability of** BOOL_TO_DINT **(see page** 1318**)**

Data types

Data type	I/O	Function
BOOL	input	input data type
DINT	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

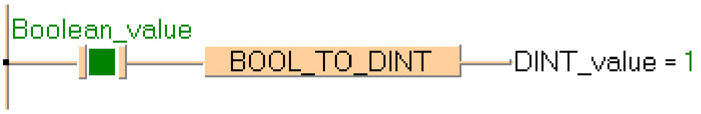


	Class	Identifier	Type	Initial
0	VAR	Boolean_value	BOOL	FALSE
1	VAR	DINT_value	DINT	0

In this example the input variable (**Boolean_value**) has been declared. Instead, you may enter a constant directly at the input contact of a function.

Body The **Boolean_value** of the data type BOOL is converted into a DOUBLE INTEGER value. The converted value is written into **DINT_value**.

LD



ST When programming with structured text, enter the following:

```
IF Boolean_value THEN
    DINT_value:=BOOL_TO_DINT(Boolean_value);
END_IF;
```

WORD_TO_DINT

WORD in DOUBLE INTEGER

Description WORD_TO_DINT converts a value of the data type WORD into a value of the data type DINT.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

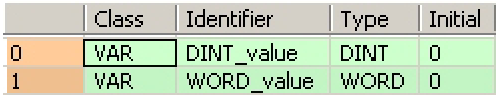
PLC types **Availability of** WORD_TO_DINT **(see page 1333)**

Data types

Data type	I/O	Function
WORD	input	input data type
DINT	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

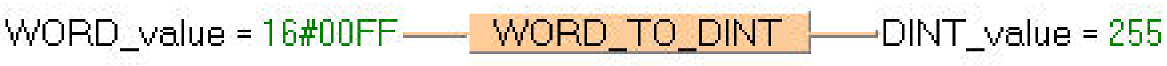


	Class	Identifier	Type	Initial
0	VAR	DINT_value	DINT	0
1	VAR	WORD_value	WORD	0

This example uses variables. You can also use a constant for the input variable.

Body **WORD_value** of the data type WORD is converted into a value of the data type INTEGER. The result will be written into **DINT_value**.

LD



ST When programming with structured text, enter the following:

```
DINT_value:=WORD_TO_DINT(WORD_value);
```

DWORD_BCD_TO_DINT

Binary coded DWORD value into DOUBLE INTEGER

Description DWORD_BCD_TO_DINT converts a binary coded value of the data type DWORD into a binary value of the data type DINT in order to be able to process a BCD value in double word format.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

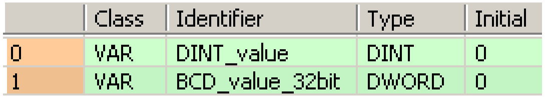
PLC types **Availability of DWORD_BCD_TO_DINT (see page 1319)**

Data types

Data type	I/O	Function
DWORD_BCD	Input	input data type
DINT	Output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.



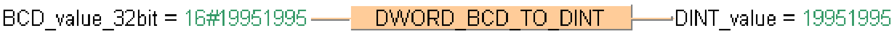
	Class	Identifier	Type	Initial
0	VAR	DINT_value	DINT	0
1	VAR	BCD_value_32bit	DWORD	0

This example uses variables. You can also use a constant for the input variable.

BCD constants can be indicated in Control FPWIN Pro as follows:
2#00011001100101010001100110010101 or
16#19951995

Body **BCD_value_32bit** of the data type DOUBLE WORD is converted into a DOUBLE INTEGER value. The converted value is written into **DINT_value**.

LD



ST When programming with structured text, enter the following:

```
DINT_value:=DWORD_BCD_TO_DINT(BCD_value_32bit);
```

DWORD_TO_DINT

DOUBLE WORD in DOUBLE INTEGER

Description DWORD_TO_DINT converts a value of the data type DOUBLE WORD into a value of the data type DOUBLE INTEGER.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of DWORD_TO_DINT (see page 1319)**

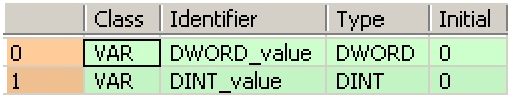
☞ **The bit combination of the input variable is assigned to the output variable.**

Data types

Data type	I/O	Function
DWORD	input	input data type
DINT	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

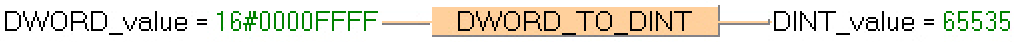


	Class	Identifier	Type	Initial
0	VAR	DWORD_value	DWORD	0
1	VAR	DINT_value	DINT	0

This example uses variables. You can also use a constant for the input variable.

Body **DWORD_value** of the data type DOUBLE WORD is converted into a DOUBLE INTEGER value. The converted value is written into **DINT_value**.

LD



ST When programming with structured text, enter the following:

```
DINT_value:=DWORD_TO_DINT(DWORD_value);
```

INT_TO_DINT

INTEGER into DOUBLE INTEGER

Description INT_TO_DINT converts a value of the data type INT into a value of the data type DINT.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

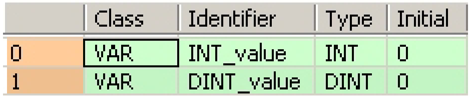
PLC types **Availability of** INT_TO_DINT **(see page** 1327**)**

Data types

Data type	**I/O**	**Function**
INT	input	input data type
DINT	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

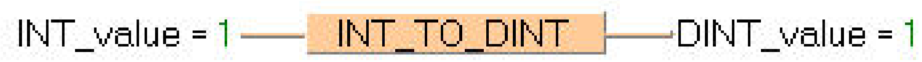


	Class	Identifier	Type	Initial
0	VAR	INT_value	INT	0
1	VAR	DINT_value	DINT	0

This example uses variables. You can also use a constant for the input variable.

Body **INT_value** of the data type INTEGER is converted into a value of the data type DOUBLE INTEGER. The result will be written into **DINT_value**.

LD



ST When programming with structured text, enter the following:

```
DINT_value:=INT_TO_DINT(INT_value);
```

UINT_TO_DINT

Unsigned INTEGER into DOUBLE INTEGER

Description UINT_TO_DINT converts a value of the data type Unsigned INTEGER into a value of the data type DINT.

UINT_TO_DINT

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

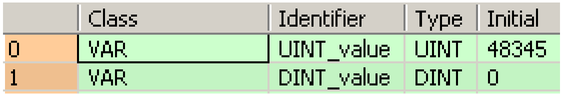
PLC types **Availability of** UINT_TO_DINT **(see page 1332)**

Data types

Data type	I/O	Function
UINT	Input	input data type
DINT	Output	conversion result

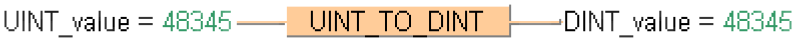
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	48345
1	VAR	DINT_value	DINT	0

LD



UINT_value = 48345 — UINT_TO_DINT — DINT_value = 48345

ST
```
DINT_value:=  UINT_TO_DINT(UINT_value);
```

UDINT_TO_DINT

Unsigned DOUBLE INTEGER into DOUBLE INTEGER

Description UDINT_TO_DINT converts a value of the data type Unsigned DOUBLE INTEGER into a value of the data type DINT.

UDINT_TO_DINT

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

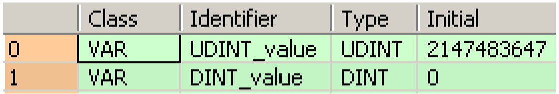
PLC types **Availability of UDINT_TO_DINT (see page 1332)**

Data types

Data type	I/O	Function
UDINT	Input	input data type
DINT	Output	conversion result

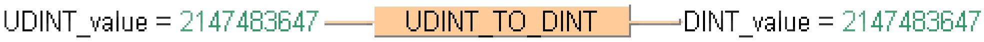
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	2147483647
1	VAR	DINT_value	DINT	0

LD


UDINT_value = 2147483647 UDINT_TO_DINT DINT_value = 2147483647

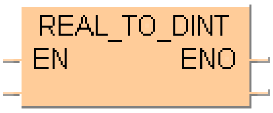
ST When programming with structured text, enter the following:

```
DINT_value := UDINT_TO_DINT(UDINT_value);
```

REAL_TO_DINT

REAL into DOUBLE INTEGER

Description REAL_TO_DINT converts a value of the data type REAL into a value of the data type DOUBLE INTEGER. The result is rounded off to the nearest whole number for the conversion.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** REAL_TO_DINT **(see page 1330)**

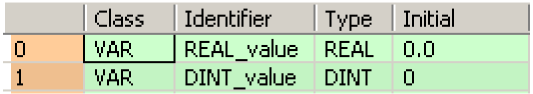
Since REAL numbers only have a resolution of about 7 digits, information for large numbers will be lost.

Data types

Data type	I/O	Function
REAL	input	input data type
DINT	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

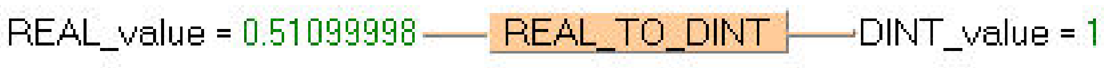


	Class	Identifier	Type	Initial
0	VAR	REAL_value	REAL	0.0
1	VAR	DINT_value	DINT	0

This example uses variables. You can also use a constant for the input variable.

Body **REAL_value** of the data type REAL is converted into a value of the data type DOUBLE INTEGER. The converted value is stored in **DINT_value**.

LD



ST When programming with structured text, enter the following:

```
DINT_value:= REAL_TO_DINT(REAL_value);
```

TRUNC_TO_DINT

Truncate (cut off) decimal digits of REAL input variable, convert to DOUBLE INTEGER

Description TRUNC_TO_DINT cuts off the decimal digits of a REAL number and delivers an output variable of the data type DOUBLE INTEGER.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** TRUNC_TO_DINT **(see page 1332)**

☞

- If the decimal digits are cut off, positive numbers will be decreased towards zero and negative numbers will be increased towards zero.
- Since REAL numbers only have a resolution of about 7 digits, information for large numbers will be lost.

Data types

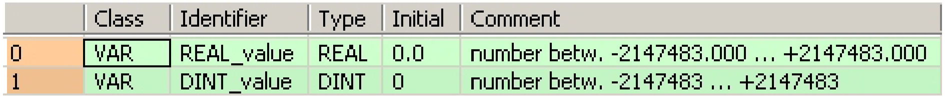
Data type	I/O	Function
REAL	input	input data type
DINT	output	conversion result

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input variable does not have the data type REAL
R9008	%MX0.900.8	for an instant	▪ output variable is greater than a 32-bit DOUBLE INTEGER
R9009	%MX0.900.9	for an instant	▪ output variable is zero

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

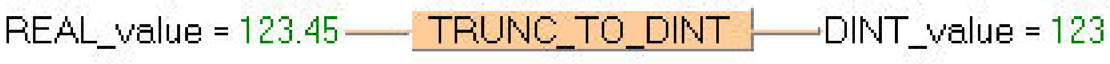


	Class	Identifier	Type	Initial	Comment
0	VAR	REAL_value	REAL	0.0	number betw. -2147483.000 ... +2147483.000
1	VAR	DINT_value	DINT	0	number betw. -2147483 ... +2147483

This example uses variables. You can also use a constant for the input variable.

Body The decimal digits of **REAL_value** are cut off. The result is stored as a 32-bit DOUBLE INTEGER in **DINT_value**.

LD



ST When programming with structured text, enter the following:

```
DINT_value:=TRUNC_TO_DINT(REAL_value);
```

TIME_TO_DINT

TIME into DOUBLE INTEGER

Description TIME_TO_DINT converts a value of the data type TIME into a value of the data type DINT. The time 10ms corresponds to the value 1, e.g. an input value of T#1m0s is converted to the value 6000.

TIME_TO_DINT

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

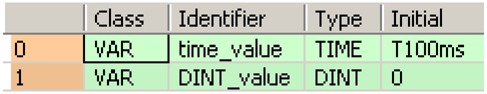
PLC types **Availability of** TIME_TO_DINT **(see page 1331)**

Data types

Data type	I/O	Function
TIME	input	input data type
DINT	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

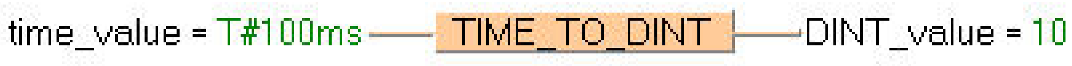


	Class	Identifier	Type	Initial
0	VAR	time_value	TIME	T100ms
1	VAR	DINT_value	DINT	0

This example uses variables. You can also use a constant for the input variable.

Body **time_value** of the data type TIME is converted to value of the data type DOUBLE INTEGER. The result is written into the output variable **DINT_value**.

LD



ST When programming with structured text, enter the following:

```
DINT_value:=TIME_TO_DINT(time_value);
```

STRING_TO_DINT

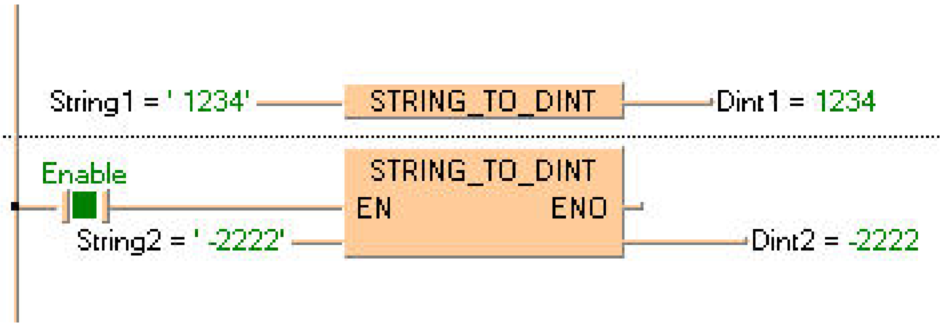
STRING (Decimal Format) to DOUBLE INTEGER

Description This function converts a string in decimal formal to a value of the data type DINT.

At first the string is converted to a value of the data type STRING[32]. Finally this is converted to a value of the data type DINT in a subprogram of approximately 270 steps, which is also used by the functions STRING_TO_INT, STRING_TO_WORD, STRING_TO_DINT and STRING_TO_DWORD.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Example with and without EN/ENO:



Acceptable Format:

'[Space][Sign][Decimal number][Space]' e.g. '   123456   '

Acceptable characters:

Space	Space “ “
Signs	Plus “+” and minus “-“
Decimal Numbers	Decimal numbers "0“ - "9“

The analysis ends with the first non-decimal number.

PLC types **Availability of** STRING_TO_DINT **(see page 1331)**

Data types

Data type	**Comment**
STRING	Input variable
DINT	Output variable

STRING_TO_DINT_ STEPSAVER

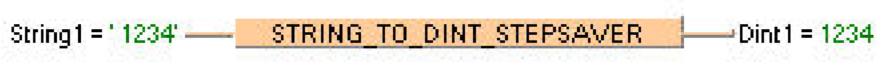
STRING (Decimal Format right-justified) to DOUBLE INTEGER

Description This function converts a right-justifed decimal number in a string to a value of the data type DINT.

The basic instruction F78_DA2BIN (see page 643) with approx. 11 steps is used. The PLC delivers an operation error especially when a character appears that is not a decimal number “0 - 9”, not a “+” or “-“ or not a space.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Example



Acceptable Format:

'[Space][Sign][Decimal number]' e.g. '   123456'

Acceptable characters:

Space	Space “␣“
Signs	Plus "+“ and minus "-"
Decimal Numbers	Decimal numbers "0“ - "9“

PLC types **Availability of** STRING_TO_DINT_STEPSAVER **(see page 1331)**

Data types

Data type	Comment
STRING	Input variable
DINT	Output variable

BOOL_TO_UDINT

BOOL into Unsigned DOUBLE INTEGER

Description BOOL_TO_UDINT converts a value of the data type BOOL into a value of the data type Unsigned DOUBLE INTEGER.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

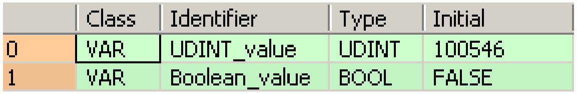
PLC types **Availability of** BOOL_TO_UDINT **(see page 1318)**

Data types

Data type	I/O	Function
BOOL	Input	input data type
UDINT	Output	conversion result

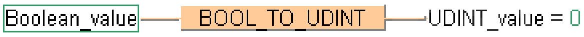
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	100546
1	VAR	Boolean_value	BOOL	FALSE

LD



ST When programming with structured text, enter the following:

```
UDINT_value := BOOL_TO_UDINT(Boolean_value);
```

WORD_TO_UDINT

WORD in Unsigned DOUBLE INTEGER

Description WORD_TO_UDINT converts a value of the data type WORD into a value of the data type Unsigned DOUBLE INTEGER.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

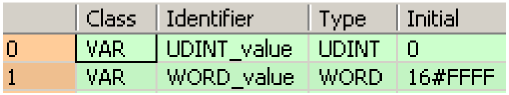
PLC types **Availability of** WORD_TO_UDINT **(see page 1333)**

Data types

Data type	I/O	Function
WORD	Input	input data type
UDINT	Output	conversion result

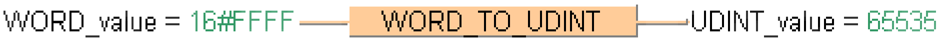
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	0
1	VAR	WORD_value	WORD	16#FFFF

LD



WORD_value = 16#FFFF WORD_TO_UDINT UDINT_value = 65535

ST When programming with structured text, enter the following:

```
UDINT_value :=WORD_TO_UDINT(WORD_value);
```

DWORD_TO_UDINT

DOUBLE WORD in Unsigned DOUBLE INTEGER

Description DWORD_TO_UDINT converts a value of the data type DWORD into a value of the data type Unsigned DOUBLE INTEGER.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

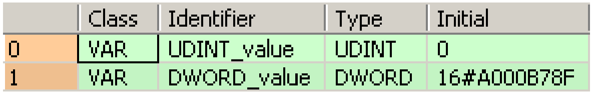
PLC types **Availability of** DWORD_TO_UDINT **(see page 1319)**

Data types

Data type	I/O	Function
DWORD	Input	input data type
UDINT	Output	conversion result

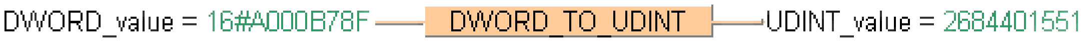
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	0
1	VAR	DWORD_value	DWORD	16#A000B78F

LD



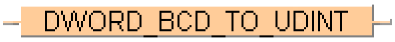
ST When programming with structured text, enter the following:

```
UDINT_value := DWORD_TO_UDINT(DWORD_value);
```

DWORD_BCD_TO_UDINT

Binary value of DOUBLE WORD in Unsigned INTEGER

Description DWORD_BCD_TO_UDINT converts a binary value of the data type DWORD into a value of the data type Unsigned DOUBLE INTEGER.


DWORD_BCD_TO_UDINT

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

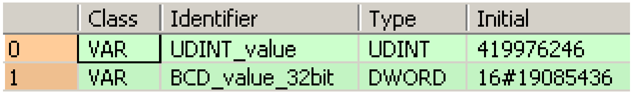
PLC types **Availability of** DWORD_BCD_TO_UDINT **(see page** 1319**)**

Data types

Data type	I/O	Function
DWORD_BCD	Input	input data type
UDINT	Output	conversion result

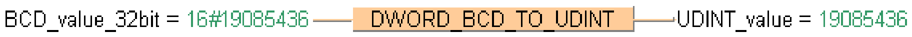
Example In this example the function is programmed in ladder diagram (LD).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	419976246
1	VAR	BCD_value_32bit	DWORD	16#19085436

LD


BCD_value_32bit = 16#19085436 — DWORD_BCD_TO_UDINT — UDINT_value = 19085436

INT_TO_UDINT

INTEGER into Unsigned DOUBLE INTEGER

Description INT_TO_UDINT converts a value of the data type INT into a value of the data type Unsigned DOUBLE INTEGER.

INT_TO_UDINT

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

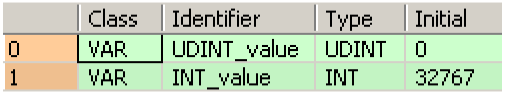
PLC types **Availability of** INT_TO_UDINT **(see page 1327)**

Data types

Data type	I/O	Function
INT	Input	input data type
UDINT	Output	conversion result

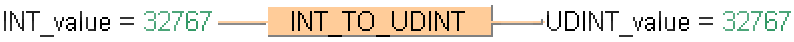
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	0
1	VAR	INT_value	INT	32767

LD



ST When programming with structured text, enter the following:

```
UDINT_value := INT_TO_UDINT(INT_value);
```

UINT_TO_UDINT

Unsigned INTEGER to Unsigned DOUBLE INTEGER

Description UINT_TO_UDINT converts a value of the data type Unsigned INTEGER into a value of the data type Unsigned DOUBLE INTEGER.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

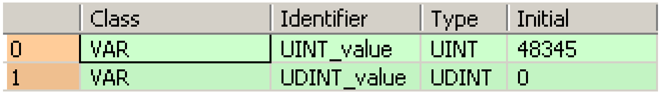
PLC types **Availability of** UINT_TO_UDINT **(see page 1332)**

Data types

Data type	I/O	Function
UINT	Input	input data type
UDINT	Output	conversion result

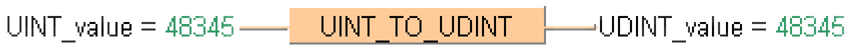
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	48345
1	VAR	UDINT_value	UDINT	0

LD



ST

```
UDINT_value:=UINT_TO_UDINT(UINT_value);
```

DINT_TO_UDINT

DOUBLE INTEGER into Unsigned DOUBLE INTEGER

Description DINT_TO_UDINT converts a value of the data type DINT into a value of the data type Unsigned DOUBLE INTEGER.

DINT_TO_UDINT

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

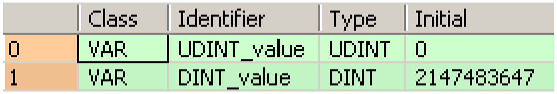
PLC types **Availability of** DINT_TO_UDINT **(see page 1319)**

Data types

Data type	I/O	Function
DINT	Input	input data type
UDINT	Output	conversion result

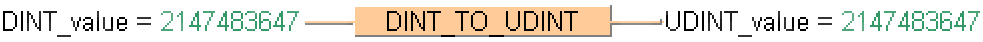
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	0
1	VAR	DINT_value	DINT	2147483647

LD



DINT_value = 2147483647 — DINT_TO_UDINT — UDINT_value = 2147483647

ST When programming with structured text, enter the following:

```
UDINT_value := DINT_TO_UDINT(DINT_value);
```

REAL_TO_UDINT

REAL into unsigned DOUBLE INTEGER

Description REAL_TO_UDINT converts a value of the data type REAL into a value of the data type Unsigned DOUBLE INTEGER. The result is rounded off to the nearest whole number for the conversion.

See also: TRUNC_TO_UDINT (see page 187)

REAL_TO_UDINT

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** REAL_TO_UDINT **(see page 1330)**

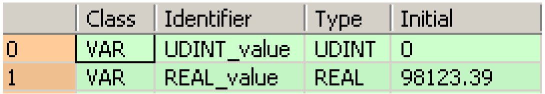
Since REAL numbers only have a resolution of about 7 digits, information for large numbers will be lost.

Data types

Data type	I/O	Function
REAL	Input	input data type
UDINT	Output	conversion result

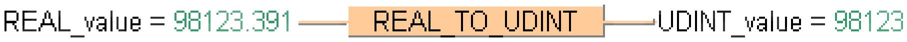
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	0
1	VAR	REAL_value	REAL	98123.39

LD


REAL_value = 98123.391 — REAL_TO_UDINT — UDINT_value = 98123

ST When programming with structured text, enter the following:

```
UDINT_value := REAL_TO_UDINT(REAL_value);
```

TRUNC_TO_UDINT Truncate (cut off) decimal digits of REAL input variable, convert to Unsigned DOUBLE INTEGER

Description TRUNC_TO_UDINT cuts off the digits following the decimal of a REAL number and delivers an output variable of the data type Unsigned DOUBLE INTEGER.

TRUNC_TO_UDINT

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** TRUNC_TO_UDINT **(see page** 1332**)**

☞

- If the decimal digits are cut off, positive numbers will be decreased towards zero and negative numbers will be increased towards zero.
- Since REAL numbers only have a resolution of about 7 digits, information for large numbers will be lost.

Data types

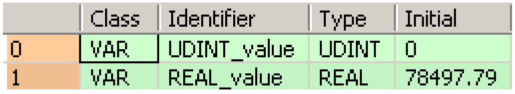
Data type	I/O	Function
REAL	Input	input data type
UDINT	Output	conversion result

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the input variable is not of the data type REAL
R9008	%MX0.900.8	for an instant	▪ the output variable is greater than a 32-bit DOUBLE INTEGER
R9009	%MX0.900.9	for an instant	▪ the output variable is zero

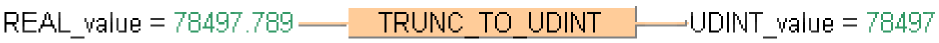
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	0
1	VAR	REAL_value	REAL	78497.79

LD



ST When programming with structured text, enter the following:

```
UDINT_value := TRUNC_TO_UDINT(REAL_value);
```

STRING_TO_UDINT

STRING (Decimal Format) into Unsigned DOUBLE INTEGER

Description STRING_TO_UDINT converts a string in decimal format to a value of the data type Unsigned DOUBLE INTEGER.

STRING_TO_UDINT

First, the string is converted to a value of the data type STRING[32], which is subsequently converted to a value of the data type UDINT in a subprogram with approximately 270 steps. This subprogram is also used by the functions STRING_TO_INT, STRING_TO_WORD, STRING_TO_UDINT and STRING_TO_DWORD.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Acceptable Format:

'[Space][Sign][Decimal number][Space]', e.g. '   123456   '

Acceptable characters:

Space	Space “ “
Signs	Plus “+” and minus “-“
Decimal numbers	Decimal numbers "0“ - "9“

The analysis ends with the first non-decimal number.

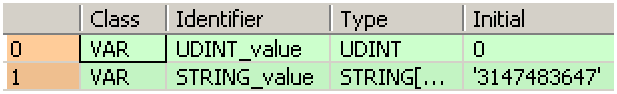
PLC types **Availability of** STRING_TO_UDINT **(see page 1331)**

Data types

Data type	Comment
STRING	Input
UDINT	Output

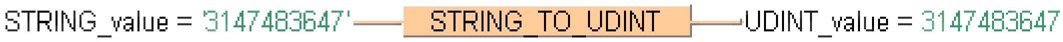
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	0
1	VAR	STRING_value	STRING[...	'3147483647'

LD



ST When programming with structured text, enter the following:

```
UDINT_value := STRING_TO_UDINT(STRING_value);
```

DATE_TO_UDINT

DATE into Unsigned DOUBLE INTEGER

Description DATE_TO_UDINT converts a value of the data type DATE into a value of the data type Unsigned DOUBLE INTEGER according to its internal format, which is seconds elapsed since "2001-01-01".

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

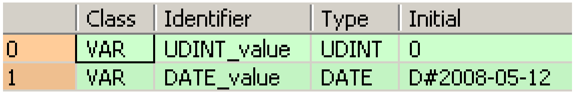
PLC types **Availability of** DATE_TO_UDINT **(see page 1319)**

Data types

Data type	I/O	Function
DATE	Input	input data type
UDINT	Output	conversion result

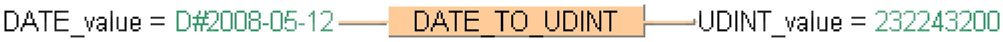
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	0
1	VAR	DATE_value	DATE	D#2008-05-12

LD



ST When programming with structured text, enter the following:

```
UDINT_value := DATE_TO_UDINT(DATE_value);
```

DT_TO_UDINT

DATE_AND_TIME into Unsigned DOUBLE INTEGER

Description DT_TO_UDINT converts a value of the data type DATE_AND_TIME into a value of the data type Unsigned DOUBLE INTEGER according to its internal format, which is seconds elapsed since "2001-01-01-00:00:00".

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

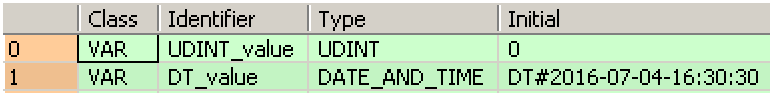
PLC types **Availability of** DT_TO_UDINT **(see page 1319)**

Data types

Data type	I/O	Function
DT	Input	input data type
UDINT	Output	conversion result

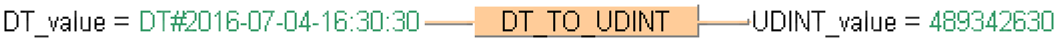
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	0
1	VAR	DT_value	DATE_AND_TIME	DT#2016-07-04-16:30:30

LD



DT_value = DT#2016-07-04-16:30:30 — DT_TO_UDINT — UDINT_value = 489342630

ST When programming with structured text, enter the following:

```
UDINT_value := DT_TO_UDINT(DT_value);
```

TOD_TO_UDINT

TIME_OF_DAY into Unsigned DOUBLE INTEGER

Description TOD_TO_UDINT converts a value of the data type TIME_OF_DAY into a value of the data type Unsigned DOUBLE INTEGER according to its internal format, which is seconds elapsed since "00:00:00".

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

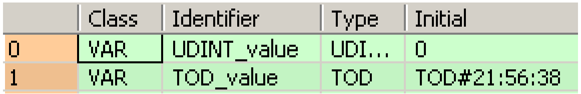
PLC types **Availability of** TOD_TO_UDINT **(see page 1332)**

Data types

Data type	I/O	Function
TOD	Input	input data type
UDINT	Output	conversion result

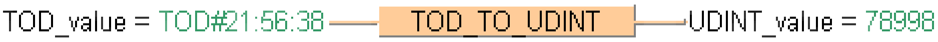
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDI...	0
1	VAR	TOD_value	TOD	TOD#21:56:38

LD



TOD_value = TOD#21:56:38 — TOD_TO_UDINT — UDINT_value = 78998

ST When programming with structured text, enter the following:

```
UDINT_value := TOD_TO_UDINT(TOD_value);
```

DWORD_TO_REAL

DWORD into REAL

Description DWORD_TO_REAL moves the bitset information of a DWORD variable to a REAL variable. The same functionality can be obtained using DWORD_OVERLAPPING_DUT.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

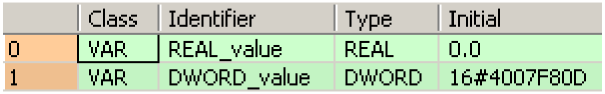
PLC types **Availability of DWORD_TO_REAL (see page 1319)**

Data types

Data type	I/O	Function
DWORD	Input	input data type
REAL	Output	conversion result

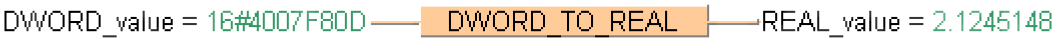
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	REAL_value	REAL	0.0
1	VAR	DWORD_value	DWORD	16#4007F80D

LD



DWORD_value = 16#4007F80D — DWORD_TO_REAL — REAL_value = 2.1245148

ST When programming with structured text, enter the following:

```
REAL_value := DWORD_TO_REAL(DWORD_value);
```

INT_TO_REAL

INTEGER into REAL

Description INT_TO_REAL converts a value of the data type INTEGER into a value of the data type REAL.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

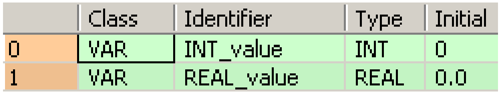
PLC types **Availability of** INT_TO_REAL **(see page 1327)**

Data types

Data type	I/O	Function
INT	input	input data type
REAL	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

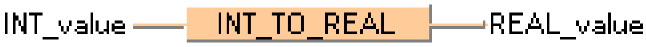


	Class	Identifier	Type	Initial
0	VAR	INT_value	INT	0
1	VAR	REAL_value	REAL	0.0

In this example the input variable (**INT_value**) has been declared. Instead, you may enter a constant directly at the input contact of a function.

Body **INT_value** of the data type INTEGER is converted into a value of the data type REAL.The converted value is stored in **REAL_value**.

LD



ST When programming with structured text, enter the following:

```
REAL_value:=INT_TO_REAL(INT_value);
```

DINT_TO_REAL

DOUBLE INTEGER into REAL

Description DINT_TO_REAL converts a value of the data type DOUBLE INTEGER into a value of the data type REAL.

DINT_TO_REAL

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

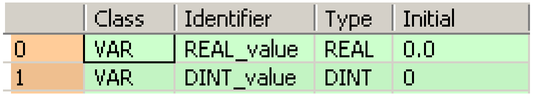
PLC types **Availability of DINT_TO_REAL (see page 1319)**

Data types

Data type	I/O	Function
DINT	input	input data type
REAL	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

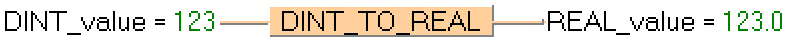


	Class	Identifier	Type	Initial
0	VAR	REAL_value	REAL	0.0
1	VAR	DINT_value	DINT	0

This example uses variables. You may also use a constant for the input variable

Body **DINT_value** of the data type DOUBLE INTEGER is converted into a value of the data type REAL. The converted value is stored in **REAL_value**.

LD



ST When programming with structured text, enter the following:

```
REAL_value:=DINT_TO_REAL(DINT_value);
```

UINT_TO_REAL

Unsigned INTEGER into REAL

Description UINT_TO_REAL converts a value of the data type Unsigned INTEGER into a value of the data type REAL.

UINT_TO_REAL

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** UINT_TO_REAL **(see page** 1332**)**

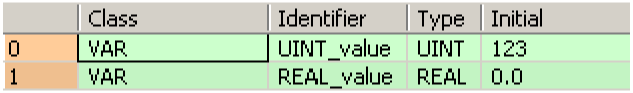
☞ **Since REAL numbers only have a resolution of about 7 digits, information for large numbers will be lost.**

Data types

Data type	I/O	Function
UINT	Input	input data type
REAL	Output	conversion result

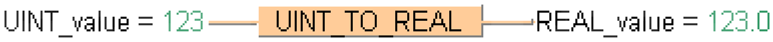
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	123
1	VAR	REAL_value	REAL	0.0

LD



ST
```
REAL_value:=  UINT_TO_REAL(UINT_value);
```

UDINT_TO_REAL Unsigned DOUBLE INTEGER into REAL

Description UDINT_TO_REAL converts a value of the data type Unsigned DOUBLE INTEGER into a value of the data type REAL.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of UDINT_TO_REAL (see page 1332)**

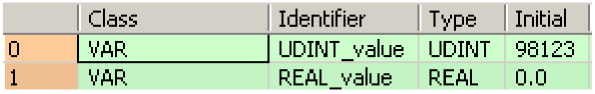
☞ **Since REAL numbers only have a resolution of about 7 digits, information for large numbers will be lost.**

Data types

Data type	I/O	Function
UDINT	Input	input data type
REAL	Output	conversion result

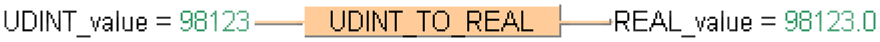
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	98123
1	VAR	REAL_value	REAL	0.0

LD



ST When programming with structured text, enter the following:

```
REAL_value := UDINT_TO_REAL(UDINT_value);
```

TIME_TO_REAL

TIME into REAL

Description TIME_TO_REAL converts a value of the data type TIME to a value of the data type REAL. 10ms of the data type TIME correspond to 1.0 REAL unit, e.g. when TIME = 10ms, REAL = 1.0; when TIME = 1s, REAL = 100.0. The resolution amounts to 10ms.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

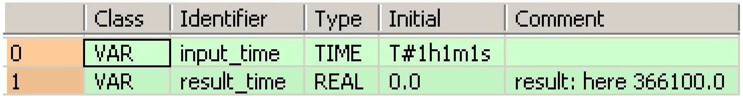
PLC types **Availability of** TIME_TO_REAL **(see page 1331)**

Data types

Data type	I/O	Function
TIME	input	input data type
REAL	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

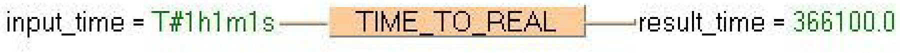
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	input_time	TIME	T#1h1m1s	
1	VAR	result_time	REAL	0.0	result: here 366100.0

This example uses variables. You can also use a constant for the input variable.

LD



ST When programming with structured text, enter the following:

```
result_real:=TIME_TO_REAL(input_time);
```

STRING_TO_REAL

STRING to REAL

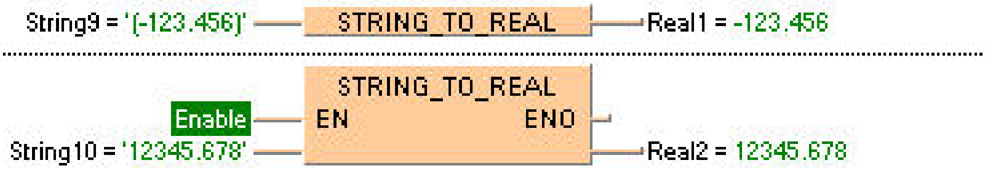
Description function converts a STRING in floating-point format into a value of the data type REAL.

STRING_TO_REAL

Thereby the attached string is first converted to a value of the data type STRING[32]. Finally this is converted to a value of the data type REAL via a sub-program that requires approximately 290 steps.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Example with and without EN/ENO:



Permissible format:

'[Space][Sign][Decimal numbers].[Decimal numbers][Space]' e.g. '   -123.456   '

Permissible characters:

Space	All characters except for "+“ (plus), "-" (minus) and all decimal numbers
Decimal numbers	Decimal numbers "0“-"9“

The analysis ends with the first non-decimal number.

PLC types **Availability of STRING_TO_REAL (see page 1331)**

Data types

Data type	Comment
STRING	input variable
REAL	output variable

WORD_TO_TIME

WORD in TIME

Description WORD_TO_TIME converts a value of the data type WORD into a value of the data type TIME.

WORD_TO_TIME

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** WORD_TO_TIME **(see page 1333)**

Data types

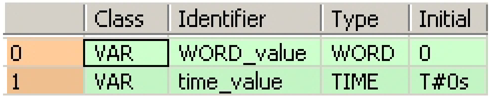
Data type	I/O	Function
WORD	input	input data type
TIME	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

Examples:

Input variable	Output variable
12345	T#123.45s
16#0012	T#180.00ms

POU header All input and output variables used for programming this function have been declared in the POU header.

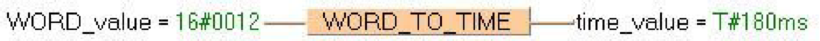


	Class	Identifier	Type	Initial
0	VAR	WORD_value	WORD	0
1	VAR	time_value	TIME	T#0s

This example uses variables. You can also use a constant for the input variable.

Body **WORD_value** of the data type WORD (16-bit) is converted into a value of the data type TIME (16-bit). The result will be written into the output variable **time_value.**

LD



ST When programming with structured text, enter the following:

```
time_value:=WORD_TO_TIME(WORD_value);
```

DWORD_TO_TIME DOUBLE WORD in TIME

Description DWORD_TO_TIME converts a value of the data type DWORD into a value of the data type TIME. A value of 1 corresponds to a time of 10ms, e.g. the input value 12345 (16#3039) is converted to a TIME T#2m3s450.00ms.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

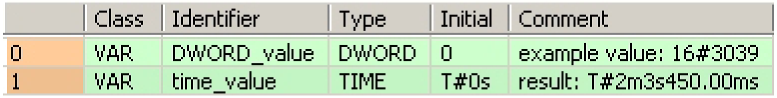
PLC types **Availability of** DWORD_TO_TIME **(see page 1319)**

Data types

Data type	I/O	Function
DWORD	input	input data type
TIME	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

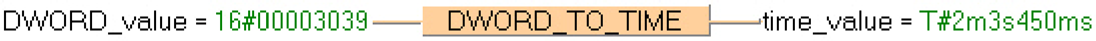


	Class	Identifier	Type	Initial	Comment
0	VAR	DWORD_value	DWORD	0	example value: 16#3039
1	VAR	time_value	TIME	T#0s	result: T#2m3s450.00ms

This example uses variables. You can also use a constant for the input variable.

Body **DWORD_value** of the data type DWORD (32-bit) is converted to value of the data type TIME (16-bit). The result is written into the output variable **time_value**.

LD



ST When programming with structured text, enter the following:

```
time_value:=DWORD_TO_TIME(DWORD_value);
```

INT_TO_TIME

INTEGER into TIME

Description INT_TO_TIME converts a value of the data type INT into a value of the data type TIME. The resolution is 10ms, e.g. when the INT value = 350, the TIME value = 3s500ms.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

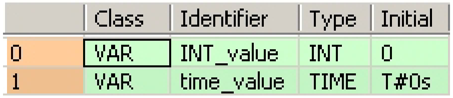
PLC types **Availability of** INT_TO_TIME **(see page 1327)**

Data types

Data type	I/O	Function
INT	input	input data type
TIME	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

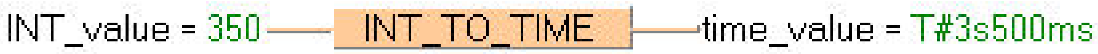


	Class	Identifier	Type	Initial
0	VAR	INT_value	INT	0
1	VAR	time_value	TIME	T#0s

This example uses variables. You can also use a constant for the input variable.

Body **INT_value** of the data type INTEGER is converted into a value of the data type TIME. The result will be written into the output variable **time_value**.

LD



ST When programming with structured text, enter the following:

```
time_value:=INT_TO_TIME(INT_value);
```

DINT_TO_TIME

DOUBLE INTEGER into TIME

Description DINT_TO_TIME converts a value of the data type DINT into a value of the data type TIME. A value of 1 corresponds to a time of 10ms, e.g. an input value of 123 is converted to a TIME T#1s230.00ms.

DINT_TO_TIME

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

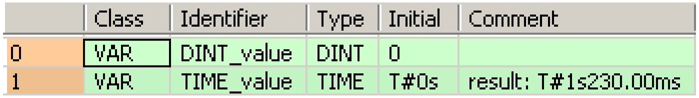
PLC types **Availability of** DINT_TO_TIME **(see page 1319)**

Data types

Data type	I/O	Function
DINT	input	input data type
TIME	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

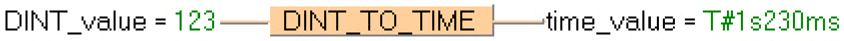


	Class	Identifier	Type	Initial	Comment
0	VAR	DINT_value	DINT	0	
1	VAR	TIME_value	TIME	T#0s	result: T#1s230.00ms

This example uses variables. You can also use a constant for the input variable.

Body **DINT_value** of the data type DOUBLE INTEGER is converted to value of the data type TIME. The result is written into the output variable **time_value**.

LD



ST When programming with structured text, enter the following:

```
time_value:=DINT_TO_TIME(DINT_value);
```

REAL_TO_TIME

REAL into TIME

Description REAL_TO_TIME converts a value of the data type REAL to a value of the data time TIME. 10ms of the data type TIME correspond to 1.0 REAL unit, e.g. when REAL = 1.0, TIME = 10ms; when REAL = 100.0, TIME = 1s. The value of the data type real is rounded off to the nearest whole number for the conversion.

REAL_TO_TIME

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

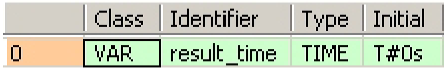
PLC types **Availability of** REAL_TO_TIME **(see page 1330)**

Data types

Data type	I/O	Function
REAL	input	input data type
TIME	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help. Since constants are entered directly at the function's input contact pins, only the output variable need be declared in the header.

POU header All input and output variables used for programming this function have been declared in the POU header.

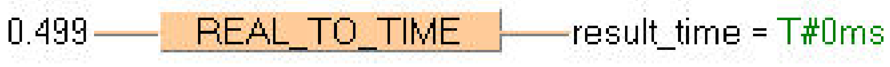


	Class	Identifier	Type	Initial
0	VAR	result_time	TIME	T#0s

Body

By clicking on the monitor icon while in the online mode, you can see the result 0.00ms immediately. Since the value at the REAL input contact is less than 0.5, it is rounded down to 0.0.

LD



ST When programming with structured text, enter the following:

```
result_time:= REAL_TO_TIME(0.499);
```

UDINT_TO_DT

Unsigned DOUBLE INTEGER into DATE_AND_TIME

Description UDINT_TO_DT converts a value of the data type Unsigned DOUBLE INTEGER into a value of the data type DATE_AND_TIME.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

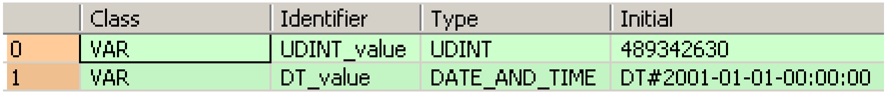
PLC types **Availability of UDINT_TO_DT (see page 1332)**

Data types

Data type	I/O	Function
UDINT	Input	input data type
DATE_AND_TIME	Output	conversion result

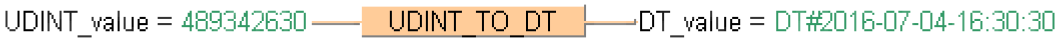
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	489342630
1	VAR	DT_value	DATE_AND_TIME	DT#2001-01-01-00:00:00

LD



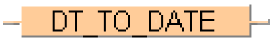
ST When programming with structured text, enter the following:

```
DT_value := UDINT_TO_DT(UDINT_value);
```

DT_TO_DATE

DATE_AND_TIME to DATE

Description DT_TO_DATE converts a value of the data type DATE_AND_TIME to a value of the data type DATE.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

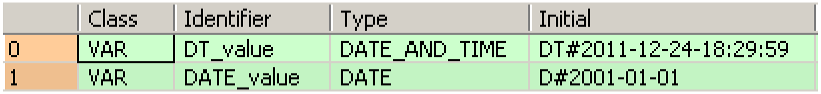
PLC types **Availability of DT_TO_DATE (see page 1319)**

Data types

Data type	I/O	Function
DATE_AND_TIME	input	date and time
DATE	output	date

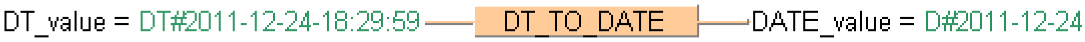
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	DT_value	DATE_AND_TIME	DT#2011-12-24-18:29:59
1	VAR	DATE_value	DATE	D#2001-01-01

LD



ST When programming with structured text, enter the following:

```
DATE_value := DT_TO_DATE(DT_value);
```

UDINT_TO_DATE

Unsigned DOUBLE INTEGER into DATE

Description UDINT_TO_DATE converts a value of the data type Unsigned DOUBLE INTEGER into a value of the data type DATE.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

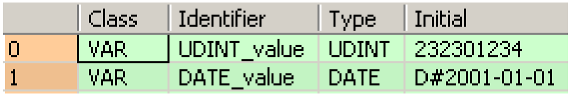
PLC types **Availability of UDINT_TO_DATE (see page 1332)**

Data types

Data type	I/O	Function
UDINT	Input	input data type
DATE	Output	conversion result

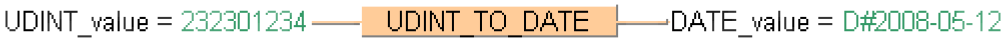
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	232301234
1	VAR	DATE_value	DATE	D#2001-01-01

LD



UDINT_value = 232301234 — UDINT_TO_DATE — DATE_value = D#2008-05-12

ST When programming with structured text, enter the following:

```
DATE_value := UDINT_TO_DATE(UDINT_value);
```

DT_TO_TOD

DATE_AND_TIME to TIME_OF_DAY

Description DT_TO_TOD converts a value of the data type DATE_AND_TIME to a value of the data type TIME_OF_DAY.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

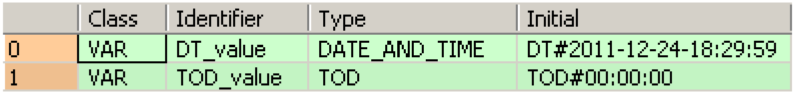
PLC types **Availability of DT_TO_TOD (see page 1319)**

Data types

Data type	I/O	Function
DATE_AND_TIME	input	input data type
TIME_OF_DAY	output	conversion result

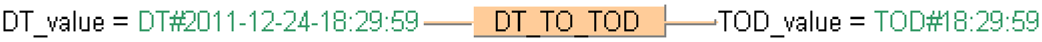
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	DT_value	DATE_AND_TIME	DT#2011-12-24-18:29:59
1	VAR	TOD_value	TOD	TOD#00:00:00

LD



```
DT_value = DT#2011-12-24-18:29:59 ── DT_TO_TOD ──TOD_value = TOD#18:29:59
```

ST When programming with structured text, enter the following:

```
TOD_value := DT_TO_TOD(DT_value);
```

UDINT_TO_TOD

Unsigned DOUBLE INTEGER into TIME_OF_DAY

Description UDINT_TO_TOD converts a value of the data type Unsigned DOUBLE INTEGER into a value of the data type TIME_OF_DAY.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

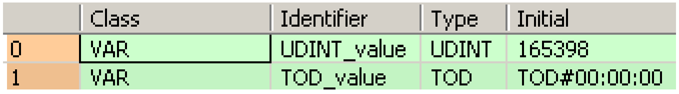
PLC types **Availability of UDINT_TO_TOD (see page 1332)**

Data types

Data type	I/O	Function
UDINT	Input	input data type
TIME_OF_DAY	Output	conversion result

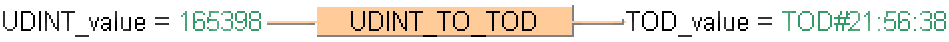
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	165398
1	VAR	TOD_value	TOD	TOD#00:00:00

LD



UDINT_value = 165398 UDINT_TO_TOD TOD_value = TOD#21:56:38

ST When programming with structured text, enter the following:

```
TOD_value := UDINT_TO_TOD(UDINT_value);
```

BOOL_TO_STRING

BOOL into STRING

Description The function BOOL_TO_STRING converts a value of the data type BOOL to a value of the data type STRING[2]. The resulting string is represented by ' 0' or ' 1'.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** BOOL_TO_STRING **(see page** 1318**)**

☞

- **When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.**
- **For further information refer to the online help: Upgrade Problems with Data Type STRING**

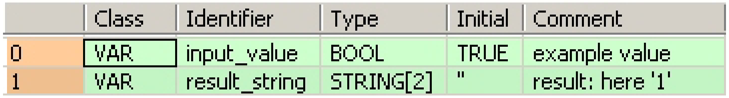
Data types

Data type	I/O	Function
BOOL	input	input data type
STRING	output	conversion result

Example 1 In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

Result string = ' 1' or ' 0'

POU header All input and output variables used for programming this function have been declared in the POU header.

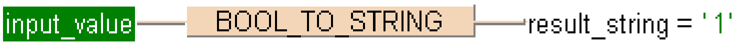


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	BOOL	TRUE	example value
1	VAR	result_string	STRING[2]	''	result: here '1'

The input variable **input_value** of the data type BOOL is intialized by the value TRUE. The output variable **result_string** is of the data type STRING[2]. It can store a maximum of two characters. You can declare a character string that has more than one character, e.g. STRING[5]. From the 5 characters reserved, only 2 are used.
Instead of using the variable **input_value**, you can write the constants TRUE or FALSE directly to the function's input contact in the body.

Body The **input_value** of the data type BOOL is converted into STRING[2]. The converted value is written to **result_string**. When the variable **input_value** = TRUE, **result_string** shows ' 1'.

LD



ST When programming with structured text, enter the following:

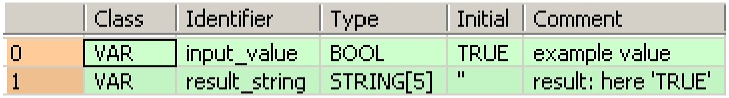
```
IF Boolean_value THEN
    output_value:=BOOL_TO_STRING(input_value);
END_IF;
```

Example 2
Result string = 'TRUE' or 'FALSE'

If you wish to have the result 'TRUE' or 'FALSE' instead of ' 0' or ' 1', you cannot use the function BOOL_TO_STRING. This example illustrates how you create a STRING[5] that contains the characters 'TRUE' or 'FALSE' from an input value of the data type BOOL.

The same POU header is used for all programming languages.

POU header



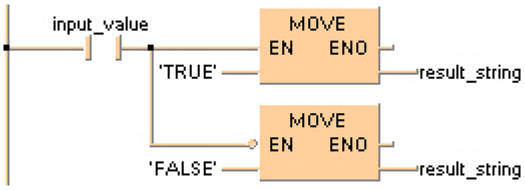
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	BOOL	TRUE	example value
1	VAR	result_string	STRING[5]	"	result: here 'TRUE'

In this example, both an input variable **input_value** of the data type BOOL and an output variable **result_string** of the data type STRING[5] are declared.

Body

In order to realize the intended operation, the standard function MOVE is used. It assigns the value of its input to its output unchanged. At the input, the STRING constant 'TRUE' or 'FALSE' is attached. In essence a "BOOL to STRING" conversion occurs, since the Boolian variable **input_variable** at the enable input (EN) contact decides the output of STRING.

LD



WORD_TO_STRING

WORD into STRING

Description The function WORD_TO_STRING converts a value of the data type WORD to a value of the data type STRING.

Generates a result string in right-aligned decimal representation, filled with leading spaces up to the predefined maximum number of characters.

WORD_TO_STRING

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Explanation

Input	**Output defined as**	**Results in**
16#ABCD	STRING[1]	'D'
	STRING[2]	'CD'
	STRING[3]	'BCD'
	STRING[4]	'ABCD'
	STRING[5]	'0ABCD'
	STRING[6]	'00ABCD'
	and so on...	

PLC types **Availability of** WORD_TO_STRING **(see page 1333)**

☞

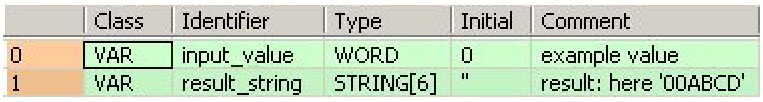
- **When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.**
- **For further information refer to the online help: Upgrade Problems with Data Type STRING**

Data types

Data type	**I/O**	**Function**
WORD	input	input data type
STRING	output	conversion result

Example 1 In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

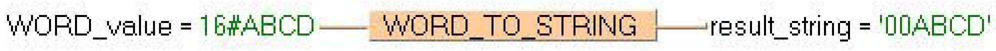


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	WORD	0	example value
1	VAR	result_string	STRING[6]	''	result: here '00ABCD'

The input variable **input_value** of the data type WORD is intialized by the value 16#ABCD. The output variable **result_string** is of the data type STRING[6]. It can store a maximum of 6 characters. Instead of using the variable **input_value**, you can enter a constant directly at the function's input contact in the body.

Body The **input_value** of the data type WORD is converted into STRING[6]. The converted value is written to **result_string**. When the variable **input_value** = 16#ABCD, **result_string** shows '00ABCD'.

LD

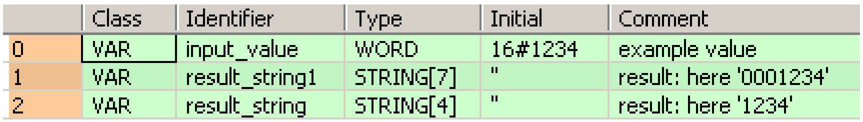


ST When programming with structured text, enter the following:

```
result_string:=WORD_TO_STRING(input_value);
```

Example 2 This example illustrates how you create STRING[4] out of the data type WORD in which the leading part of the string '16#' is cut out.
The example is programmed in LD and IL. The same POU header is used for both programming languages.

POU header



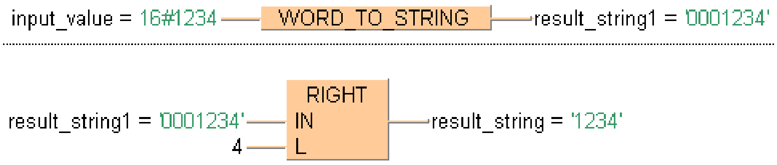
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	WORD	16#1234	example value
1	VAR	result_string1	STRING[7]	''	result: here '0001234'
2	VAR	result_string	STRING[4]	''	result: here '1234'

In this example, both an input variable **input_value** of the data type WORD and an output variable **result_string** of the data type STRING[4] are declared.

Body In carrying out the operation in question, the standard function RIGHT is attached to the function WORD_TO_STRING. RIGHT creates a right-justified character string of length **L**.

In the example, the output string of WORD_TO_STRING function is added at the input of the RIGHT function. At the **L** input of RIGHT, the INT constant 4 determines the length of the STRING to be replaced. Out of the variable **input_value** = 0001234, the **result_string** 1234 results from the data type conversion and the RIGHT function.

LD



DWORD_TO_STRING

DOUBLE WORD into STRING

Description The function DWORD_TO_STRING converts a value of the data type DWORD to a value of the data type STRING.

Generates a result string in right-aligned decimal representation, filled with leading spaces up to the predefined maximum number of characters.

DWORD_TO_STRING

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Explanation

Input	Output defined as	Results in
16#ABCDEFFE	STRING[2]	'FE'
	STRING[4]	'EFFE'
	STRING[6]	'CDEFFE'
	STRING[8]	'ABCDEFFE'
	STRING[10]	'00ABCDEFFE'
	STRING[12]	'0000ABCDEFFE'
	and so on...	

PLC types **Availability of** DWORD_TO_STRING **(see page 1319)**

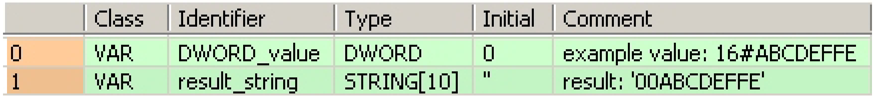
- **When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.**
- **For further information refer to the online help: Upgrade Problems with Data Type STRING**

Data types

Data type	I/O	Function
DWORD	input	input data type
STRING	output	conversion result

Example 1 In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

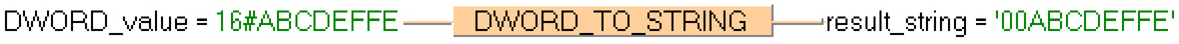


	Class	Identifier	Type	Initial	Comment
0	VAR	DWORD_value	DWORD	0	example value: 16#ABCDEFFE
1	VAR	result_string	STRING[10]	''	result: '00ABCDEFFE'

The input variable **DWORD_value** of the data type DWORD is intialized by the value 16#ABCDEFFE. The output variable **result_string** is of the data type STRING[10]. It can store a maximum of 10 characters. Instead of using the variable **DWORD_value**, you can enter a constant directly at the function's input contact in the body.

Body The **DWORD_value** of the data type DWORD is converted into STRING[10]. The converted value is written to **result_string**. When the variable **DWORD_value** = 16#ABCDEFFE, **result_string** shows '00ABCDEFFE'.

LD

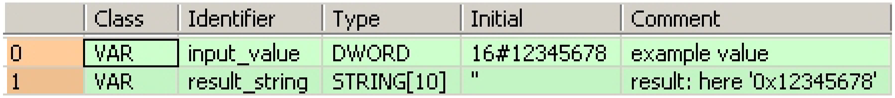


ST When programming with structured text, enter the following:

```
result_string:=DWORD_TO_STRING(input_value);
```

Example 2 This example illustrates how you create STRING[10] out of the data type DWORD in which the leading part of the string '16#' is replaced by the string '0x'.
The example is programmed in LD and IL. The same POU header is used for both programming languages.

POU header



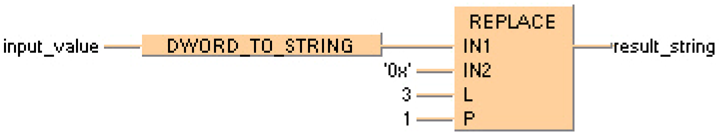
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	DWORD	16#12345678	example value
1	VAR	result_string	STRING[10]	''	result: here '0x12345678'

In this example the input variables **input_value** of the data type DWORD and an output variable **result_string** of the data type STRING[10] are declared.

Body In carrying out the operation in question, the standard function REPLACE is attached to the function DWORD_TO_STRING. REPLACE replaces one section of a character string with another.

In the example, the output string of DWORD_TO_STRING function is added at input IN1 of the REPLACE function. At input IN2, the STRING constant '0x' is added as the replacement STRING. At the L input of REPLACE, the INT constant 3 determines the length of the STRING to be replaced. The P input determines the position at which the replacement begins. In this case it is the INT number 1. From the variable **input_value** = 16#12345678, the **result_string** = '0x12345678' results after undergoing the data type conversion and REPLACE function.

LD



DATE_TO_STRING

DATE into STRING

Description DATE_TO_STRING converts a value of the data type DATE into a value of the data type STRING[10].

The range for the input date is from D#2001-01-01 to D#2099-12-31.

DATE_TO_STRING

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of DATE_TO_STRING (see page 1319)**

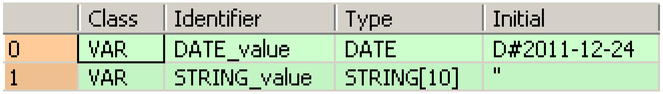
☞ **All character spaces in the result string will be filled.**

Data types

Data type	I/O	Function
DATE	input	input data type
STRING	output	conversion result STRING[10]

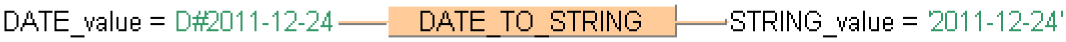
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	DATE_value	DATE	D#2011-12-24
1	VAR	STRING_value	STRING[10]	''

LD



ST When programming with structured text, enter the following:

```
STRING_value := DATE_TO_STRING(DATE_value);
```

DT_TO_STRING

DATE_AND_TIME into STRING

Description DT_TO_STRING converts a value of the data type DATE_AND_TIME into a value of the data type STRING[19].

The range for the input date is from DT#2001-01-01-00:00:00 to DT#2099-12-31-23:59:59.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of DT_TO_STRING (see page 1319)**

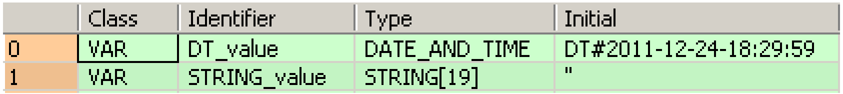
All character spaces in the result string will be filled.

Data types

Data type	I/O	Function
DATE_AND_TIME	input	input data type
STRING	output	conversion result STRING[19]

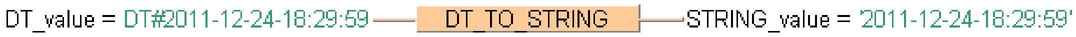
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	DT_value	DATE_AND_TIME	DT#2011-12-24-18:29:59
1	VAR	STRING_value	STRING[19]	''

LD



DT_value = DT#2011-12-24-18:29:59 — DT_TO_STRING — STRING_value = '2011-12-24-18:29:59'

ST When programming with structured text, enter the following:

```
STRING_value := DT_TO_STRING(DT_value);
```

INT_TO_STRING

INTEGER into STRING

Description The function INT_TO_STRING converts a value of the data type INT to a value of the data type STRING.

Generates a result string in right-aligned decimal representation, filled with leading spaces up to the predefined maximum number of characters.

INT_TO_STRING

Explanation

Function used	String1 defined as	Result
String1:=INT_TO_STRING(-12345)	STRING[1]	'5'
	STRING[2]	'45'
	STRING[3]	'345'
	STRING[4]	'2345'
	STRING[5]	'12345'
	STRING[6]	'-12345'
	STRING[7]	'␣-12345'
	STRING[8]	'␣␣-12345'
	and so on...	

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** INT_TO_STRING **(see page 1327)**

☞

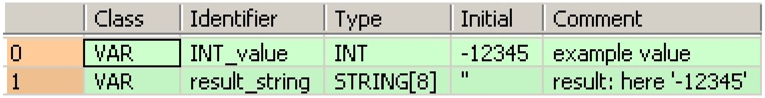
- **When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.**
- **For further information refer to the online help: Upgrade Problems with Data Type STRING**

Data types

Data type	I/O	Function
INT	input	input data type
STRING	output	conversion result

Example 1 In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header In the POU header, input and output variables are declared that are used in the function.

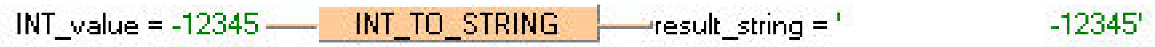


	Class	Identifier	Type	Initial	Comment
0	VAR	INT_value	INT	-12345	example value
1	VAR	result_string	STRING[8]	''	result: here '-12345'

The input variable **INT_value** of the data type INT is intialized by the value -12345. The output variable **result_string** is of the data type STRING[8]. It can store a maximum of 8 characters. Instead of using the variable **INT_value**, you can enter a constant directly at the function's input contact in the body.

Body The **INT_value** of the data type INT is converted into STRING[8]. The converted value is written to **result_string**. When the variable **INT_value** = -12345, **result_string** shows '␣␣-12345'.

LD



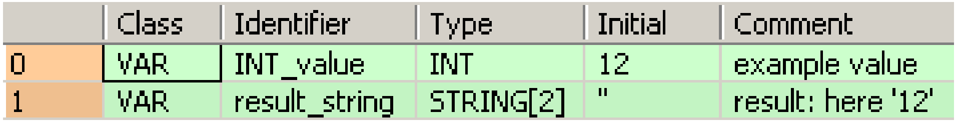
INT_value = -12345 — INT_TO_STRING — result_string = '␣␣-12345'

ST When programming with structured text, enter the following:

```
result_string:= INT_TO_STRING(input_value);
```

Example 2 This example illustrates how you create a STRING[2] that appears right justified out of the data type INT.

POU header

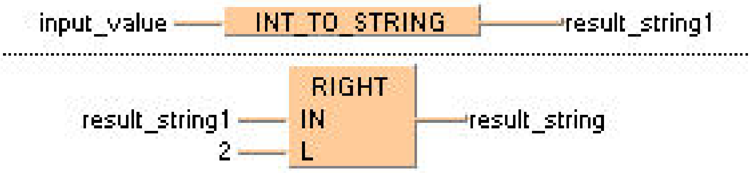


	Class	Identifier	Type	Initial	Comment
0	VAR	INT_value	INT	12	example value
1	VAR	result_string	STRING[2]	''	result: here '12'

In this example, both an input variable **INT_value** of the data type INT and an output variable **result_string** of the data type STRING[2] are declared.

Body In carrying out the operation in question, the standard function RIGHT (see page 265) is attached to the function INT_TO_STRING. RIGHT creates a right-justified character string with the length **L**. In the example, the variable **INT_variable** = 12 is converted by INT_TO_STRING to the dummy string '␣␣12'. The function RIGHT then creates the **result_string** '12'.

LD



ST When programming with structured text, enter the following:

```
result_string:=RIGHT(IN:=INT_TO_STRING(input_value), L:=2);
```

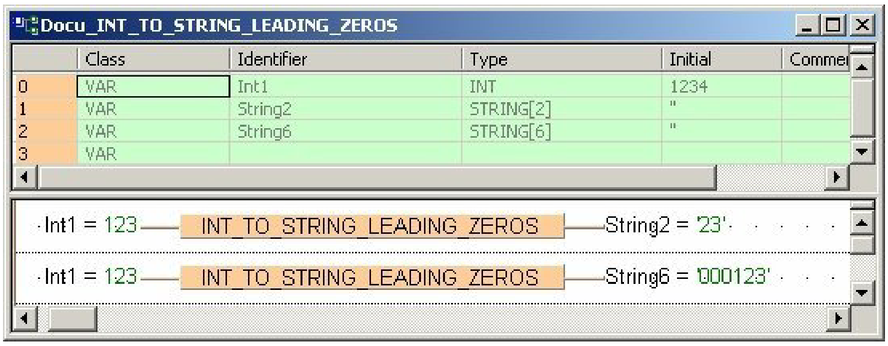
INT_TO_STRING_ LEADING_ZEROS

INTEGER into STRING

Description The function INT_TO_STRING_LEADING_ZEROS converts a value of the data type INT (positive values) to a value of the data type STRING.

Generates a result string in right-aligned decimal representation, filled with leading spaces up to the predefined maximum number of characters.

Example:



Docu_INT_TO_STRING_LEADING_ZEROS

	Class	Identifier	Type	Initial	Commer
0	VAR	Int1	INT	1234	
1	VAR	String2	STRING[2]	''	
2	VAR	String6	STRING[6]	''	
3	VAR				

```
Int1 = 123 —— INT_TO_STRING_LEADING_ZEROS —— String2 = '23'
Int1 = 123 —— INT_TO_STRING_LEADING_ZEROS —— String6 = '000123'
```

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Data types

Data type	I/O	Function
INT	input	input data type
STRING	output	conversion result

Explanation

Function used	String1 defined as	Result
String1:=INT_TO_STRING(25)	STRING[1]	'5'
	STRING[2]	'25'
	STRING[3]	'025'
	STRING[4]	'0025'
	STRING[5]	'00025'
	STRING[6]	'000025'
	STRING[7]	'0000025'
	STRING[8]	'00000025'
	and so on...	

PLC types **Availability of** INT_TO_STRING_LEADING_ZEROS **(see page 1327)**

☞

- **When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.**
- **For further information refer to the online help: Upgrade Problems with Data Type STRING**

DINT_TO_STRING

DOUBLE INTEGER into STRING

Description The function DINT_TO_STRING converts a value of the data type DINT to a value of the data type STRING.

Generates a result string in right-aligned decimal representation, filled with leading spaces up to the predefined maximum number of characters.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Explanation

Function used	String1 defined as	Result
String1:=DINT_TO_STRING(-12345678)	STRING[2]	'78'
	STRING[4]	'5678'
	STRING[6]	'345678'
	STRING[8]	'12345678'
	STRING[10]	'␣-12345678'
	STRING[12]	'␣␣␣-12345678'
	and so on...	

PLC types **Availability of** DINT_TO_STRING **(see page** 1319**)**

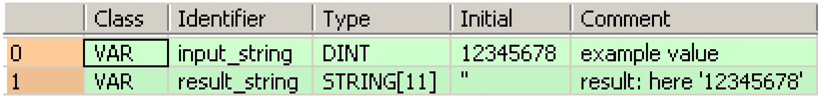
- **When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.**
- **For further information refer to the online help: Upgrade Problems with Data Type STRING**

Data types

Data type	I/O	Function
DINT	input	input data type
STRING	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header In the POU header, input and output variables are declared that are used in the function.

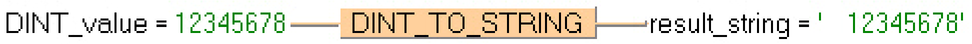


	Class	Identifier	Type	Initial	Comment
0	VAR	input_string	DINT	12345678	example value
1	VAR	result_string	STRING[11]	''	result: here '12345678'

The input variable **input_value** of the data type DINT is intialized by the value 12345678. The output variable **result_string** is of the data type STRING[11]. It can store a maximum of 11 characters. Instead of using the variable **input_value,** you can enter a constant directly at the function's input contact in the body.

Body The **input_value** of the data type DINT is converted into STRING[11]. The converted value is written to **result_string**. When the variable **input_value** = 12345678, **result_string** shows '␣␣␣12345678'.

LD


DINT_value = 12345678 — DINT_TO_STRING — result_string = '   12345678'

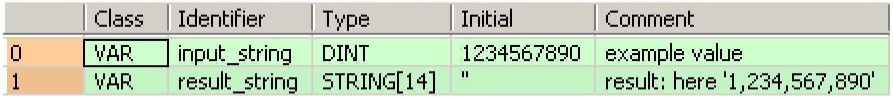
ST When programming with structured text, enter the following:

```
result_string:=DINT_TO_STRING(input_value);
```

Example 2 This example illustrates how you create, from an input value of the data type DINT, a STRING[14] that contains a DINT number representation with commas after every three significant figures.

The example is programmed in LD and IL. The same POU header is used for both programming languages.

POU header



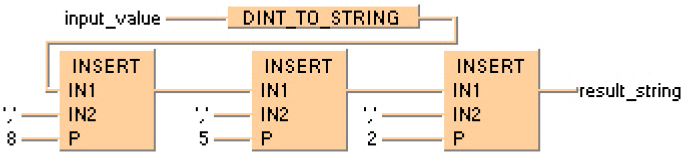
	Class	Identifier	Type	Initial	Comment
0	VAR	input_string	DINT	1234567890	example value
1	VAR	result_string	STRING[14]	''	result: here '1,234,567,890'

In this example, both an input variable **input_value** of the data type DINT and an output variable **result_string** of the data type STRING[14] are declared.

Body In carrying out the operation in question, three standard functions INSERT are attached successively to the function DINT_TO_STRING. Each INSERT function inserts the attached character string at input IN2 into the character string at input IN1. The position at which the character string is to be introduced is determined by INT value at input P.

In the example all three INSERT functions insert the assigned STRING constant ',' after each three significant figures at input IN2. The correct position of each comma is determined by an INT constant at each respective P input. Out of the variable **input_value** = 1234567890, the **result_string** 1,234,567,890 results from the data type conversion and the three INSERT functions.

LD



UDINT_TO_STRING Unsigned DOUBLE INTEGER into STRING

Description The function UDINT_TO_STRING converts a value of the data type UDINT to a value of the data type STRING.

Generates a result string in right-aligned decimal representation, filled with leading spaces up to the predefined maximum number of characters.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Explanation

Function used	String1 defined as	Result
String1:=UDINT_TO_STRING(-12345678)	STRING[2]	'78'
	STRING[4]	'5678'
	STRING[6]	'345678'
	STRING[8]	'12345678'
	STRING[10]	'␣-12345678'
	STRING[12]	'␣␣␣-12345678'
	and so on...	

PLC types **Availability of** UDINT_TO_STRING

☞

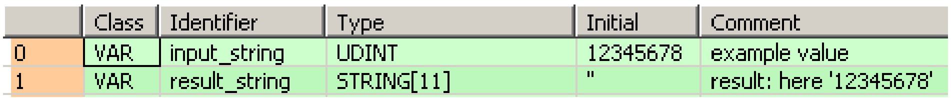
- **When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.**
- **For further information refer to the online help: Upgrade Problems with Data Type STRING**

Data types

Data type	I/O	Function
UDINT	Input	Input data type
STRING	Output	Conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

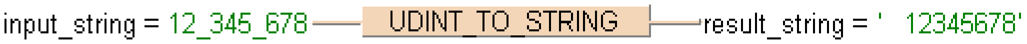


	Class	Identifier	Type	Initial	Comment
0	VAR	input_string	UDINT	12345678	example value
1	VAR	result_string	STRING[11]	''	result: here '12345678'

The input variable **input_value** of the data type UDINT is intialized by the value 12345678. The output variable **result_string** is of the data type STRING[11]. It can store a maximum of 11 characters. Instead of using the variable **input_value,** you can enter a constant directly at the function's input contact in the body.

Body The **input_value** of the data type DINT is converted into STRING[11]. The converted value is written to **result_string**. When the variable **input_value** = 12345678, **result_string** shows '␣␣␣ 12345678'.

LD



ST When programming with structured text, enter the following:

```
result_string:=UDINT_TO_STRING(input_value);
```

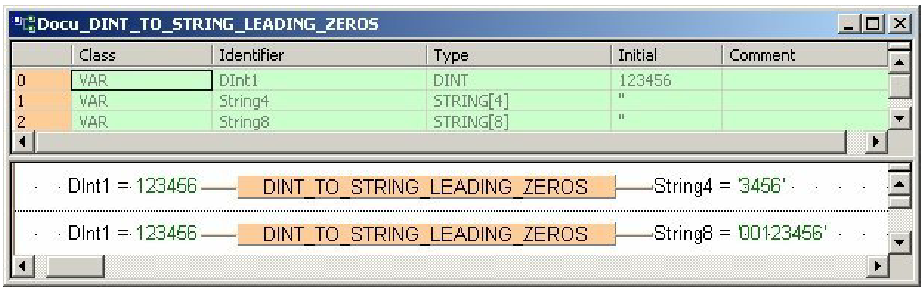
DINT_TO_STRING_LEADING_ZEROS

DOUBLE INTEGER into STRING

Description This function converts a value of the data type DINT (positive value) to a value of the data type STRING. It generates a result string in decimal representation that is right aligned. It is filled with leading zeros up to the maximum number of characters defined for the string.

DINT_TO_STRING_LEADING_ZEROS

Example



Docu_DINT_TO_STRING_LEADING_ZEROS

	Class	Identifier	Type	Initial	Comment
0	VAR	DInt1	DINT	123456	
1	VAR	String4	STRING[4]	''	
2	VAR	String8	STRING[8]	''	

DInt1 = 123456 — DINT_TO_STRING_LEADING_ZEROS — String4 = '3456'

DInt1 = 123456 — DINT_TO_STRING_LEADING_ZEROS — String8 = '00123456'

Explanation

Function used	String1 defined as	Result
String1:=DINT_TO_STRING(12345678)	STRING[2]	'78'
	STRING[4]	'5678'
	STRING[6]	'345678'
	STRING[8]	'12345678'
	STRING[10]	'0012345678'
	STRING[12]	'000012345678'
	and so on...	

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** DINT_TO_STRING_LEADING_ZEROS **(see page 1319)**

- **When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.**
- **For further information refer to the online help: Upgrade Problems with Data Type STRING**

Data types

Data type	I/O	Function
DINT	input	input data type
STRING	output	conversion result

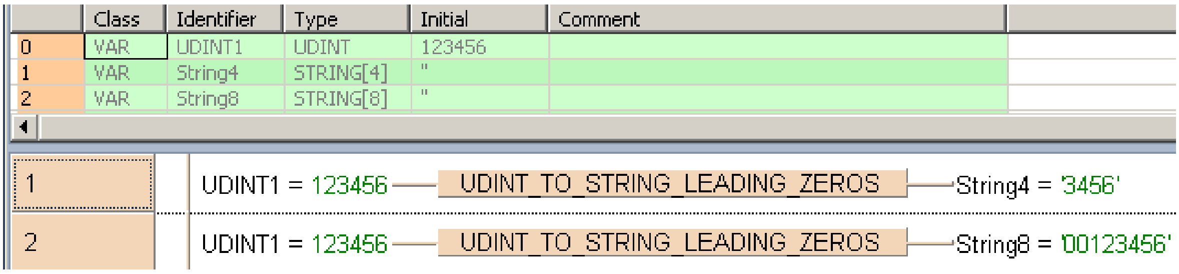
UDINT_TO_STRING _LEADING_ZEROS

Unsigned DOUBLE INTEGER into STRING

Description This function converts a value of the data type UDINT (positive value) to a value of the data type STRING. It generates a result string in decimal representation that is right aligned. It is filled with leading zeros up to the maximum number of characters defined for the string.

UDINT_TO_STRING_LEADING_ZEROS

Example



	Class	Identifier	Type	Initial	Comment
0	VAR	UDINT1	UDINT	123456	
1	VAR	String4	STRING[4]	''	
2	VAR	String8	STRING[8]	''	

1 UDINT1 = 123456 — UDINT_TO_STRING_LEADING_ZEROS — String4 = '3456'

2 UDINT1 = 123456 — UDINT_TO_STRING_LEADING_ZEROS — String8 = '00123456'

Explanation

Function used	String1 defined as	Result
String1:=UDINT_TO_STRING(12345678)	STRING[2]	'78'
	STRING[4]	'5678'
	STRING[6]	'345678'
	STRING[8]	'12345678'
	STRING[10]	'0012345678'
	STRING[12]	'000012345678'
	and so on...	

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** UDINT_TO_STRING_LEADING_ZEROS **(see page 1319)**

- **When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.**
- **For further information refer to the online help: Upgrade Problems with Data Type STRING**

Data types

Data type	I/O	Function
UDINT	input	input data type
STRING	output	conversion result

UINT_TO_STRING

Unsigned INTEGER into STRING

Description UINT_TO_STRING converts a value of the data type Unsigned INTEGER into a value of the data type STRING.

Generates a result string in right-aligned decimal representation, filled with leading spaces up to the predefined maximum number of characters.

See also: **UINT_TO_STRING_LEADING_ZEROS** (see page 227)

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** UINT_TO_STRING **(see page 1332)**

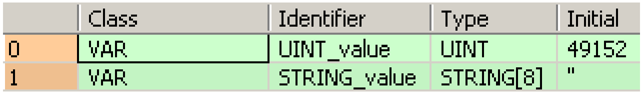
The result is not specified when the range of the input values does not match the range of the output values.

Data types

Data type	**I/O**	**Function**
UINT	Input	input data type
STRING	Output	conversion result

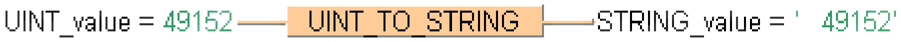
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	49152
1	VAR	STRING_value	STRING[8]	''

LD



UINT_value = 49152 — UINT_TO_STRING — STRING_value = '   49152'

ST
```
STRING_value:= UINT_TO_STRING(UINT_value);
```

UINT_TO_STRING_ LEADING_ZEROS

Unsigned INTEGER into STRING

Description UINT_TO_STRING_LEADING_ZEROS converts a value of the data type Unsigned INTEGER into a value of the data type STRING.

Generates a result string in right-aligned decimal representation, filled with leading spaces up to the predefined maximum number of characters.

UINT_TO_STRING_LEADING_ZEROS

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** UINT_TO_STRING_LEADING_ZEROS **(see page** 1332**)**

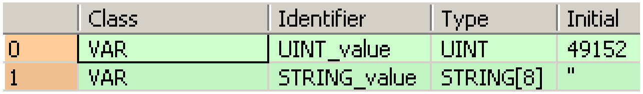
☞ **The result is not specified when the range of the input values does not match the range of the output values.**

Data types

Data type	I/O	Function
UINT	Input	input data type
STRING	Output	conversion result

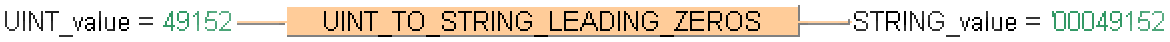
Example In this example the function is programmed in ladder diagram (LD).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	49152
1	VAR	STRING_value	STRING[8]	''

LD

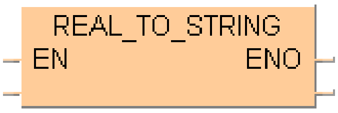


ST
```
STRING_value:=UINT_TO_STRING_LEADING_ZEROS(UINT_value);
```

REAL_TO_STRING

REAL into STRING

Description The function REAL_TO_STRING converts a value from the data type REAL into a value of the data type STRING[15], which has 7 spaces both before and after the decimal point. The resulting string is right justified within the range '-999999.0000000' to '9999999.0000000'. The plus sign is omitted in the positive range. Leading zeros are filled with empty spaces (e.g. out of -12.0 the STRING '␣␣ -12.0').



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

☞

- **The function requires approximately 160 steps of program memory. For repeated use you should integrate it into a user function that is only stored once in the memory.**
- **When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.**
- **For further information refer to the online help: Upgrade Problems with Data Type STRING**

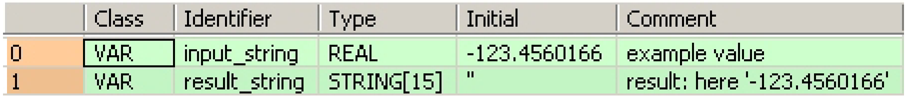
PLC types **Availability of** REAL_TO_STRING **(see page** 1330**)**

Data types

Data type	I/O	Function
REAL	input	input data type
STRING	output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header In the POU header, input and output variables are declared that are used in the function.

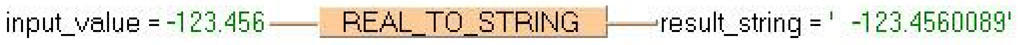


	Class	Identifier	Type	Initial	Comment
0	VAR	input_string	REAL	-123.4560166	example value
1	VAR	result_string	STRING[15]	''	result: here '-123.4560166'

The input variable **input_value** of the data type REAL is intialized by the value -123.4560166. The output variable **result_string** is of the data type STRING[15]. It can store a maximum of 15 characters. Instead of using the variable **input_value,** you can enter a constant directly at the function's input contact in the body.

Body The **input_value** of the data type REAL is converted into STRING[15]. The converted value is written to **result_string**. When the variable **input_value** = 123.4560166, **result_string** shows ' -123.4560165'.

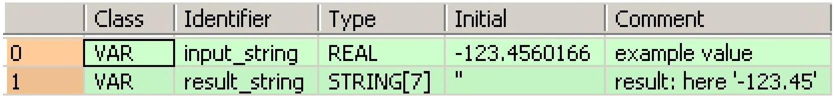
LD



Example 2

This example illustrates how you create a STRING[7] with 4 positions before and 2 positions after the decimal point out of the data type REAL.
The example is programmed in LD and IL. The same POU header is used for both programming languages.

POU header



	Class	Identifier	Type	Initial	Comment
0	VAR	input_string	REAL	-123.4560166	example value
1	VAR	result_string	STRING[7]	''	result: here '-123.45'

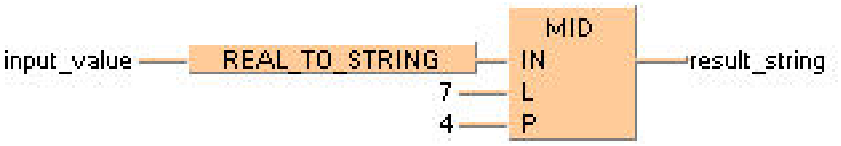
In this example, both an input variable **input_value** of the data type REAL and an output variable **result_string** of the data type STRING[7] are declared.

Body

In carrying out the operation in question, the standard function MID is attached to the function REAL_TO_STRING. MID creates a central sector in the character string from position P (INT value) with L (INT value) characters.

In the example, the INT constant 7 is entered at the L input of MID, which determines the length of the result string. The INT constant 4 at input P determines the position at which the central sector begins. Out of the variable **input_value** = -123.4560166, the STRING '   -123.4560166' results from the data type conversion. The MID function cuts off the STRING at position 4 and yields the **result_string** '-123.45'.

LD



TIME_TO_STRING

TIME into STRING

Description The function TIME_TO_STRING converts a value of the data type TIME to a value of the data type STRING[20]. In accordance with IEC-1131, the result string is displayed with a short time prefix and without underlines. Possible values for the result string's range are from 'T#000d00h00m00s000ms' to 'T#248d13h13m56s470ms'.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

☞ **When using the data type STRING with small PLCs like FP1 or FP-M, make sure that the length of the result string is equal to or greater than the length of the source string.**

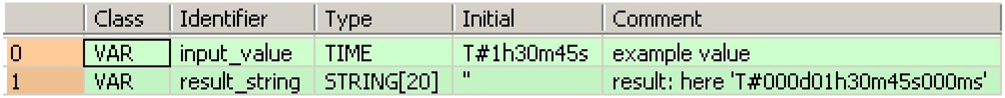
PLC types **Availability of** TIME_TO_STRING **(see page** 1331**)**

Data types

Data type	I/O	Function
TIME	input	input data type
STRING	output	conversion result

Example 1 In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header In the POU header, input and output variables are declared that are used in the function.

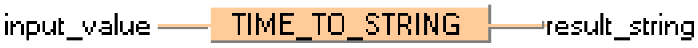


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	TIME	T#1h30m45s	example value
1	VAR	result_string	STRING[20]	''	result: here 'T#000d01h30m45s000ms'

The input variable **input_value** of the data type TIME is intialized by the value T#1h30m45s. The output variable **result_string** is of the data type STRING[20]. It can store a maximum of 20 characters. Instead of using the variable **input_value,** you can enter a constant directly at the function's input contact in the body.

Body The **input_value** of the data type TIME is converted into STRING[20]. The converted value is written to **result_string**. When the variable **input_value** = T#1h30m45s, **result_string** shows 'T#000d01h30m45s000ms'.

LD



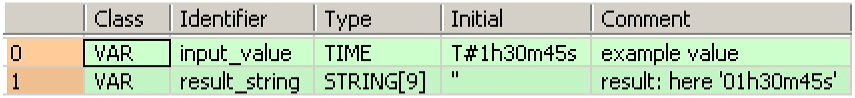
ST When programming with structured text, enter the following:

```
result_string:=TIME_TO_STRING(input_value);
```

Example 2

This example shows how, from an input value of the data type TIME, a TIME STRING[9] with the format 'xxhxxmxxs' is created (only hours, minutes and seconds are output).
The example is programmed in LD and IL. The same POU header is used for both programming languages.

POU header



	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	TIME	T#1h30m45s	example value
1	VAR	result_string	STRING[9]	''	result: here '01h30m45s'

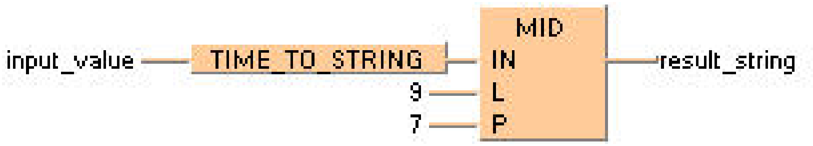
In this example, both an input variable **input_value** of the data type TIME and an output variable **result_string** of the data type STRING[9] are declared.

Body

In carrying out the operation in question, the standard function MID is attached to the function TIME_TO_STRING. MID creates a central sector in the character string from position P (INT value) with L (INT value) characters.

In the example, the INT constant 9 is entered at the L input of MID, which determines the length of the result string. The INT constant 7 at input P determines the position at which the central sector begins. Out of the variable **input_value** = T#1h30m45s, the STRING 'T#000d01h30m45s000ms' results from the data type conversion. The MID function cuts off the STRING at position 7 and yields the **result_string** '01h30m45s'.

LD

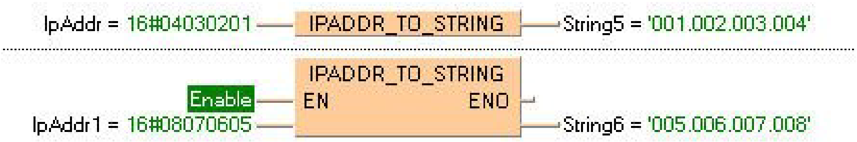


IPADDR_TO_STRING

IP Address to STRING

Description This function converts a binary IP address of the data type DWORD into a STRING in IP address format.

Example



Permissible format:

'Octet1.Octet2.Octet3.Octet4', e.g.: '192.168.206.004'

Permissible characters:

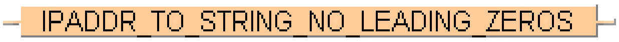
Octets 1-4	Decimal numbers "0“-"9“, maximal 3 positions, without leading zeros in the range 0-255

The conversion is such that the highest byte of the ET-LAN address represents the fourth octet and lowest byte of the IP address the first octet. The format of the IP address corresponds to the standard format as used in "Standard Socket Application Interfaces", for example.

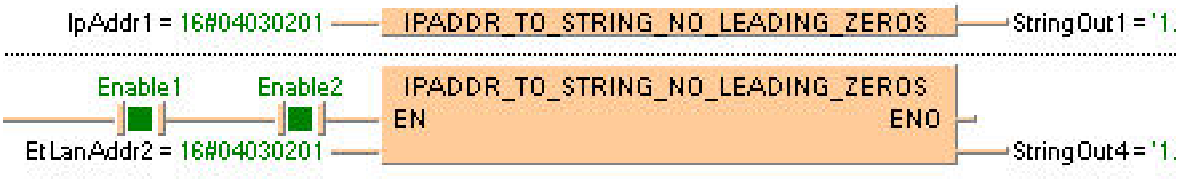
IPADDR_TO_STRING_ NO_LEADING_ZEROS

IP Address to STRING

Description This function converts a binary IP address of the data type DWORD into a STRING in IP address format.



Example



Permissible format:

'Octet1.Octet2.Octet3.Octet4', e.g.: '192.168.206.4'

Permissible characters:

Octets 1-4	Decimal numbers "0“-"9“, maximal 3 positions, without leading zeros in the range 0-255

The conversion is such that the highest byte of the ET-LAN address represents the fourth octet and lowest byte of the IP address the first octet. The format of the IP address corresponds to the standard format as used in "Standard Socket Application Interfaces", for example.

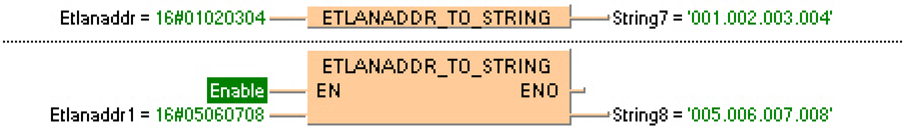
ETLANADDR_TO_STRING

ETLAN Address to STRING

Description This function converts a binary ETLAN address of the data type DWORD into a STRING in ETLAN address format.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Example



Permissible format:

'Octet1.Octet2.Octet3.Octet4', e.g.: '192.168.206.004'

Permissible characters:

Octets 1-4	Decimal numbers "0“-"9“, maximal 3 positions, with leading zeros in the range 0-255

The conversion is such that the highest byte of the ET-LAN address represents the first octet and lowest byte of the IP address the fourth octet. This format for ET-LAN addresses is used, for example, by the FP Serie's ET-LAN modules.

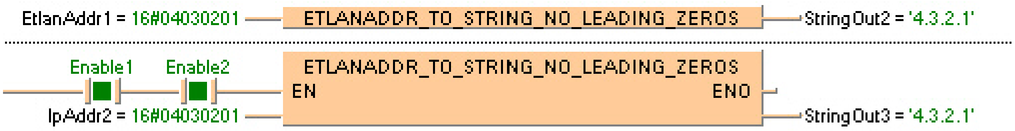
ETLANADDR_TO_STRING _NO_LEADING_ZEROS

ETLAN Address to STRING

Description This function converts a binary ETLAN address of the data type DWORD into a STRING in ETLAN address format.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Example



Permissible format:

'Octet1.Octet2.Octet3.Octet4', e.g.: '192.168.206.4'

Permissible characters:

Octets 1-4	Decimal numbers "0“-"9“, maximal 3 positions, without leading zeros in the range 0-255

The conversion is such that the highest byte of the ET-LAN address represents the first octet and lowest byte of the IP address the fourth octet. This format for ET-LAN addresses is used, for example, by the FP Serie's ET-LAN modules.

TOD_TO_STRING

TIME_OF_DAY into STRING

Description

TOD_TO_STRING converts a value of the data type TIME_OF_DAY into a value of the data type STRING[8].

The range for the input time of day is from TOD#00:00:00 to TOD#23:59:59.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** TOD_TO_STRING **(see page 1332)**

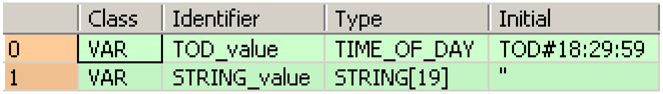
All character spaces in the result string will be filled.

Data types

Data type	I/O	Function
TIME_OF_DAY	input	input data type
STRING	output	conversion result STRING[8]

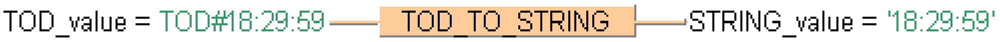
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	TOD_value	TIME_OF_DAY	TOD#18:29:59
1	VAR	STRING_value	STRING[19]	''

LD



ST When programming with structured text, enter the following:

```
STRING_value := TOD_TO_STRING(TOD_value);
```

WORD_TO_BOOL16

WORD to BOOL16

Description This function copies data of the data type WORD at the input to an array with 16 elements of the data type BOOL at the output.

WORD_TO_BOOL16

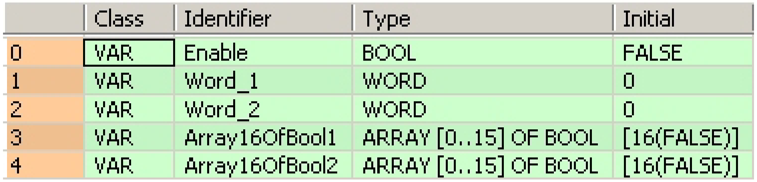
To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** WORD_TO_BOOL16 **(see page 1333)**

Data types

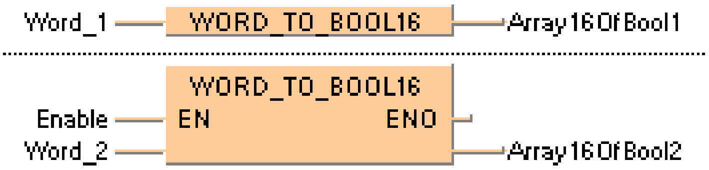
Data type	Comment
WORD	input variable
ARRAY of BOOL	ARRAY with 16 elements

POU header:



	Class	Identifier	Type	Initial
0	VAR	Enable	BOOL	FALSE
1	VAR	Word_1	WORD	0
2	VAR	Word_2	WORD	0
3	VAR	Array16OfBool1	ARRAY [0..15] OF BOOL	[16(FALSE)]
4	VAR	Array16OfBool2	ARRAY [0..15] OF BOOL	[16(FALSE)]

Body with and without EN/ENO:



DWORD_TO_BOOL32

DOUBLE WORD to BOOL32

Description This function copies data of the data type DWORD at the input to an array with 32 elements of the data type BOOL at the output.

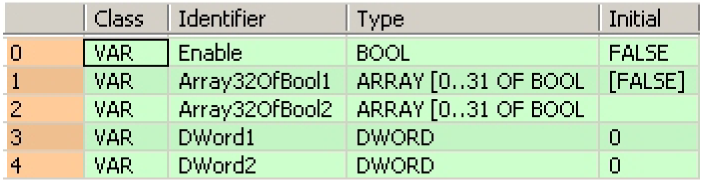
To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** DWORD_TO_BOOL32 **(see page 1319)**

Data types

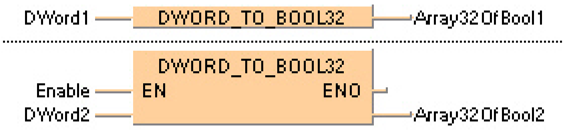
Data type	Comment
DWORD	input variable
ARRAY of BOOL	ARRAY with 32 elements

POU header:



	Class	Identifier	Type	Initial
0	VAR	Enable	BOOL	FALSE
1	VAR	Array32OfBool1	ARRAY [0..31 OF BOOL	[FALSE]
2	VAR	Array32OfBool2	ARRAY [0..31 OF BOOL	
3	VAR	DWord1	DWORD	0
4	VAR	DWord2	DWORD	0

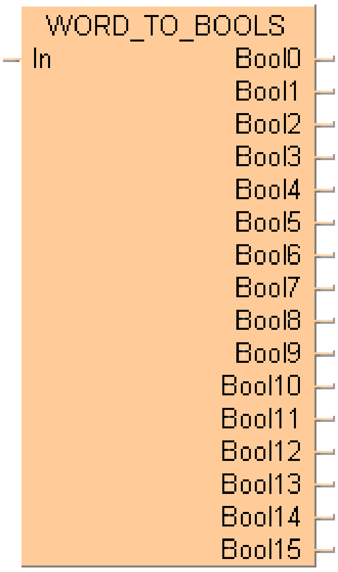
Body with and without EN/ENO:



WORD_TO_BOOLS

WORD to 16 variables of the data type BOOL

Description This function converts a value of the data type WORD bit-wise to 16 values of the data type BOOL.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

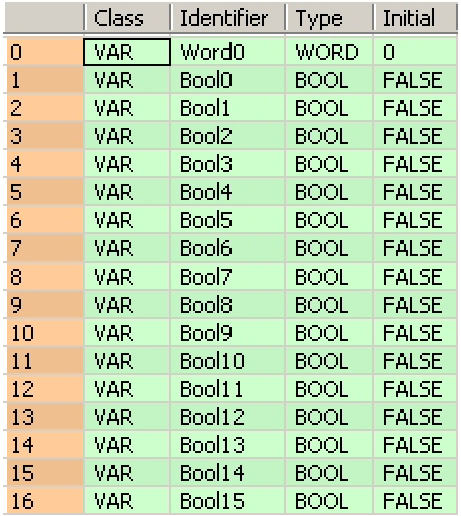
The outputs Bool0 to Bool15 need not be allocated in LD or FBD, or used explicitly in the ST editor's formal list of parameters. Program code is only generated for those outputs that are truly used.

PLC types **Availability of** WORD_TO_BOOLS **(see page 1333)**

Data types

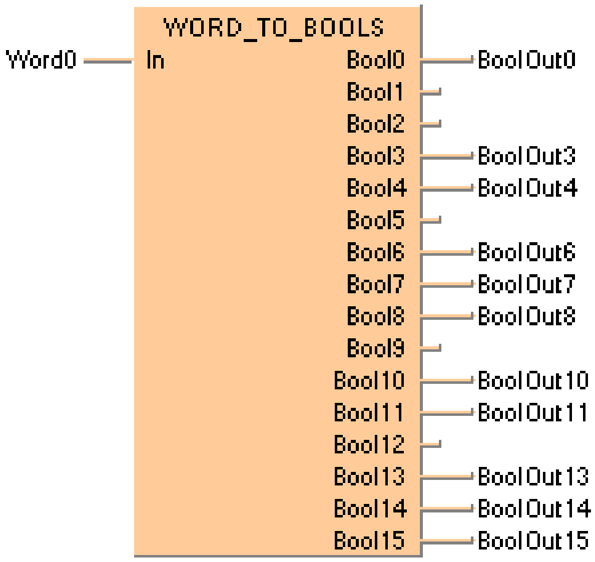
Variable	Data type	Function
In	WORD	input variable
BOOL0 ... **BOOL15**	BOOL	16 output variables of the data type BOOL

POU header:



	Class	Identifier	Type	Initial
0	VAR	Word0	WORD	0
1	VAR	Bool0	BOOL	FALSE
2	VAR	Bool1	BOOL	FALSE
3	VAR	Bool2	BOOL	FALSE
4	VAR	Bool3	BOOL	FALSE
5	VAR	Bool4	BOOL	FALSE
6	VAR	Bool5	BOOL	FALSE
7	VAR	Bool6	BOOL	FALSE
8	VAR	Bool7	BOOL	FALSE
9	VAR	Bool8	BOOL	FALSE
10	VAR	Bool9	BOOL	FALSE
11	VAR	Bool10	BOOL	FALSE
12	VAR	Bool11	BOOL	FALSE
13	VAR	Bool12	BOOL	FALSE
14	VAR	Bool13	BOOL	FALSE
15	VAR	Bool14	BOOL	FALSE
16	VAR	Bool15	BOOL	FALSE

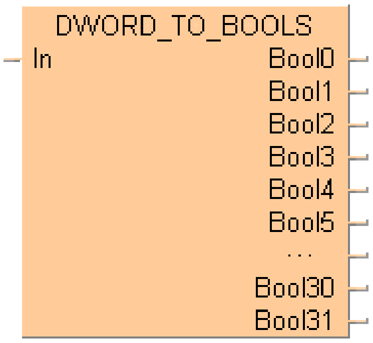
Body:


WORD_TO_BOOLS
Word0
In
Bool0
BoolOut0
Bool1
Bool2
Bool3
BoolOut3
Bool4
BoolOut4
Bool5
Bool6
BoolOut6
Bool7
BoolOut7
Bool8
BoolOut8
Bool9
Bool10
BoolOut10
Bool11
BoolOut11
Bool12
Bool13
BoolOut13
Bool14
BoolOut14
Bool15
BoolOut15

DWORD_TO_BOOLS

DOUBLE WORD to 32 variables of the data type BOOL

Description This function converts a values of the data type DWORD bit-wise to 32 values of the data type BOOL.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

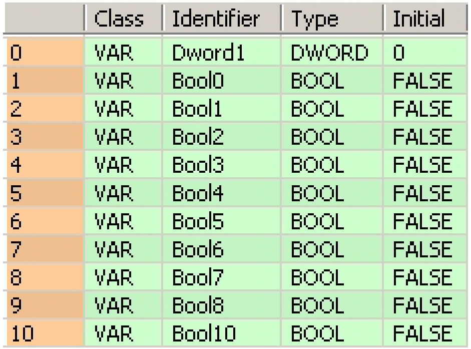
The outputs Bool0 to Bool31 need not be allocated in LD or FBD, or used explicitly in the ST editor's formal list of parameters. Program code is only generated for those outputs that are truly used.

PLC types **Availability of** DWORD_TO_BOOLS **(see page 1319)**

Data types

Variable	Data type	Function
In	DWORD	input variable
BOOL0 ... **BOOL31**	BOOL	32 output variables of the data type BOOL

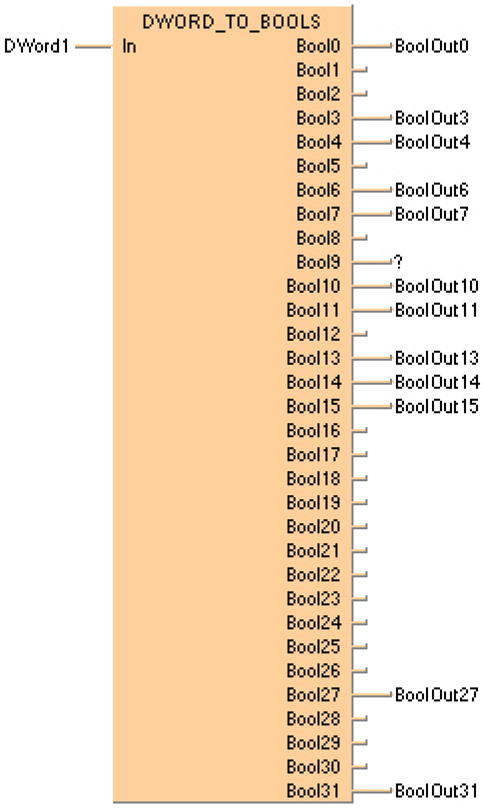
POU header:



	Class	Identifier	Type	Initial
0	VAR	Dword1	DWORD	0
1	VAR	Bool0	BOOL	FALSE
2	VAR	Bool1	BOOL	FALSE
3	VAR	Bool2	BOOL	FALSE
4	VAR	Bool3	BOOL	FALSE
5	VAR	Bool4	BOOL	FALSE
6	VAR	Bool5	BOOL	FALSE
7	VAR	Bool6	BOOL	FALSE
8	VAR	Bool7	BOOL	FALSE
9	VAR	Bool8	BOOL	FALSE
10	VAR	Bool10	BOOL	FALSE

etc. to Bool31

Body:


DWORD_TO_BOOLS
DWord1
In
Bool0
BoolOut0
Bool1
Bool2
Bool3
BoolOut3
Bool4
BoolOut4
Bool5
Bool6
BoolOut6
Bool7
BoolOut7
Bool8
Bool9
?
Bool10
BoolOut10
Bool11
BoolOut11
Bool12
Bool13
BoolOut13
Bool14
BoolOut14
Bool15
BoolOut15
Bool16
Bool17
Bool18
Bool19
Bool20
Bool21
Bool22
Bool23
Bool24
Bool25
Bool26
Bool27
BoolOut27
Bool28
Bool29
Bool30
Bool31
BoolOut31

INT_TO_BCD_WORD

INTEGER into BCD value of WORD

Description INT_TO_BCD_WORD converts a binary value of the data type INT into a binary coded decimal integer (BCD) value of the type WORD in order to be able to output BCD values in word format.

INT_TO_BCD_WORD

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** INT_TO_BCD_WORD **(see page** 1327**)**

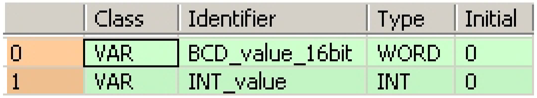
Since the output variable is of the type WORD and is therefore comprised of 16 bits, the value for the input variable is limited to 4 digits and must be between 0 and 9999.

Data types

Data type	**I/O**	**Function**
INT	Input	input data type
BCD_WORD	Output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

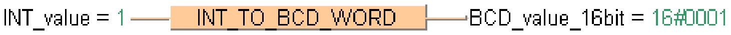


	Class	Identifier	Type	Initial
0	VAR	BCD_value_16bit	WORD	0
1	VAR	INT_value	INT	0

This example uses variables. You can also use a constant for the input variable.

Body **INT_value** of the data type INTEGER is converted into a BCD value of the data type WORD. The converted value is written into **BCD_value_16bit**.

LD



INT_value = 1 — INT_TO_BCD_WORD — BCD_value_16bit = 16#0001

ST When programming with structured text, enter the following:

```
BCD_value_16bit:=INT_TO_BCD_WORD(INT_value);
```

DINT_TO_BCD_DWORD

DOUBLE INTEGER into BCD DOUBLE WORD

Description DINT_TO_BCD_DWORD converts a value of the data type DINT into a BCD value of the data type DWORD.

DINT_TO_BCD_DWORD

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** DINT_TO_BCD_DWORD **(see page 1319)**

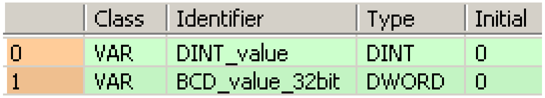
☞ **The value for the input variable should be between 0 and 999,999,999.**

Data types

Data type	I/O	Function
DINT	Input	input data type
BCD_DWORD	Output	conversion result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

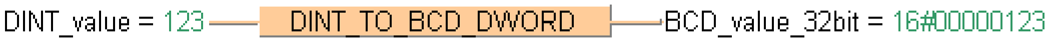


	Class	Identifier	Type	Initial
0	VAR	DINT_value	DINT	0
1	VAR	BCD_value_32bit	DWORD	0

This example uses variables. You can also use a constant for the input variable.

Body **DINT_value** of the data type DOUBLE INTEGER is converted into a BCD value of the data type DOUBLE WORD. The converted value is written to **BCD_value_32bit**.

LD



ST When programming with structured text, enter the following:

```
BCD_value_32bit:=DINT_TO_BCD_DWORD(DINT_value);
```

UINT_TO_BCD_WORD

Unsigned INTEGER into BCD value of WORD

Description UINT_TO_BCD_WORD converts a value of the data type Unsigned INTEGER into a BCD value of the data type WORD.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

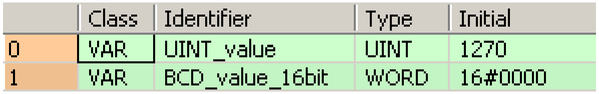
PLC types **Availability of** UINT_TO_BCD_WORD **(see page** 1332**)**

Data types

Data type	I/O	Function
UINT	Input	input data type
BCD_WORD	Output	conversion result

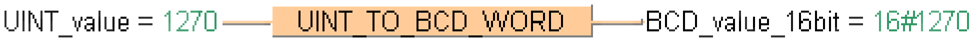
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UINT_value	UINT	1270
1	VAR	BCD_value_16bit	WORD	16#0000

LD



UINT_value = 1270 UINT_TO_BCD_WORD BCD_value_16bit = 16#1270

ST
```
BCD_value_16bit:=UINT_TO_BCD_WORD(UINT_value);
```

UDINT_TO_BCD_DWORD

Unsigned DOUBLE INTEGER into BCD DOUBLE WORD

Description UDINT_TO_BCD_DWORD converts a value of the data type Unsigned DOUBLE INTEGER into a BCD value of the data type D WORD.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

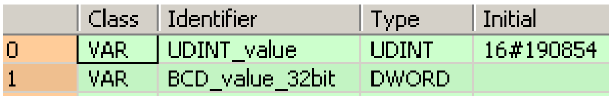
PLC types **Availability of** UDINT_TO_BCD_DWORD **(see page** 1332**)**

Data types

Data type	I/O	Function
UDINT	Input	input data type
BCD_DWORD	Output	conversion result

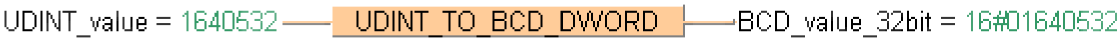
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	UDINT_value	UDINT	16#190854
1	VAR	BCD_value_32bit	DWORD	

LD



ST When programming with structured text, enter the following:

```
BCD_value_32bit := UDINT_TO_BCD_DWORD(UDINT_value);
```

STRING_TO_IPADDR

STRING to IP Address

Description This function converts a STRING in IP address format into a value of the data type DWORD.

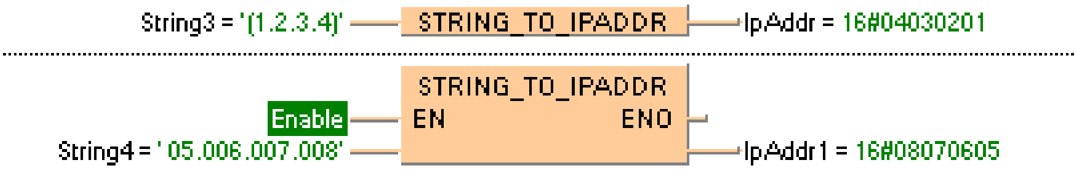
STRING_TO_IPADDR

Thereby the attached string is first converted to a value of the data type STRING[32]. Finally this is converted to a value of the data type DWORD via a sub-programm of approx. 330 steps that is also used in the functions STRING_TO_IPADDR and STRING_TO_ETLANADDR.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

See also: STRING_TO_IPADDR_STEPSAVER (see page 248)

Example:



Permissible format:

'[Space]Octet1.Octet2.Octet3.Octet4[Space]', e.g.: '   [192.168.206.4]   '

Permissible characters:

Space	All characters except for decimal numbers
Octets 1-4	Decimal numbers "0“-"9“, maximal 3 positions, with or without leading zeros in the range 0-255

PLC types **Availability of** STRING_TO_IPADDR **(see page 1331)**

- **The analysis ends with the first non-decimal number after the 4th octet or in case of a format error.**
- **If the format is wrong the result is 0.**
- **The conversion is such that the first octet represents the lowest byte of the IP address and the fourth octet the highest byte of the ET-LAN address. The format corresponds to the standard format as used in "Standard Socket Application Interfaces", for example.**

Data types

Data type	Comment
STRING	input variable
DWORD	output variable

STRING_TO_IPADDR _STEPSAVER

STRING (IP-Address Format 00a.0bb.0cc.ddd) to DWORD

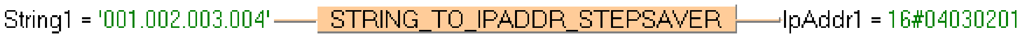
Description This function converts a STRING in IP address format into a value of the data type DWORD.

STRING_TO_IPADDR_STEPSAVER

The function uses for approx. 50 steps of generated code the basic instruction F76_A2BIN (see page 637). The instruction expects that each octet consists of three characters with leading zeros. Otherwise the PLC delivers an operation error.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Example:


String1 = '001.002.003.004' STRING_TO_IPADDR_STEPSAVER IpAddr1 = 16#04030201

Permissible format:

'Octet1.Octet2.Octet3.Octet4[Space]', e.g.: '   [192.168.206.4]   '

Permissible characters:

Octets 1-4	Decimal numbers "0“-"9“, maximal 3 positions, with or without leading zeros in the range 0-255

PLC types **Availability of** STRING_TO_IPADDR_STEPSAVER **(see page 1331)**

☞

- **If the format is wrong the result is 0.**
- **The conversion is such that the first octet represents the lowest byte of the IP address and the fourth octet the highest byte of the ET-LAN address. The format corresponds to the standard format as used in "Standard Socket Application Interfaces", for example.**

Data types

Data type	Comment
STRING	input variable
DWORD	output variable

STRING_TO_ETLAN ADDR

STRING to ETLAN Address

Description This function converts a STRING in IP address format into a value of the data type DWORD.

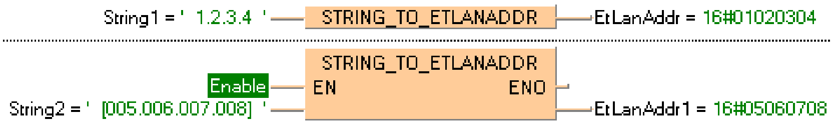
STRING_TO_ETLANADDR

Thereby the attached string is first converted to a value of the data type STRING[32]. Finally this is converted to a value of the data type DWORD via a sub-programm of approx. 330 steps that is also used in the functions STRING_TO_IPADDR and STRING_TO_ETLANADDR.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

See also: **STRING_TO_ETLANADDR_STEPSAVER**

Example with and without EN/ENO:



Permissible format:

'[Space]Octet1.Octet2.Octet3.Octet4[Space]', e.g.: '   [192.168.206.4]   '

Permissible characters:

Space	All characters except for decimal numbers
Octets 1-4	Decimal numbers "0“-"9“, maximal 3 positions, with or without leading zeros in the range 0-255

☞

- **The analysis ends with the first non-decimal number after the 4th octet or in case of a format error.**
- **If the format is wrong the result is 0.**
- **The conversion is such that the highest byte of the ET-LAN address represents the first octet and lowest byte of the IP address the fourth octet. This format for ET-LAN addresses is used, for example, by the FP Serie's ET-LAN modules.**

STRING_TO_ETLAN ADDR_STEPSAVER

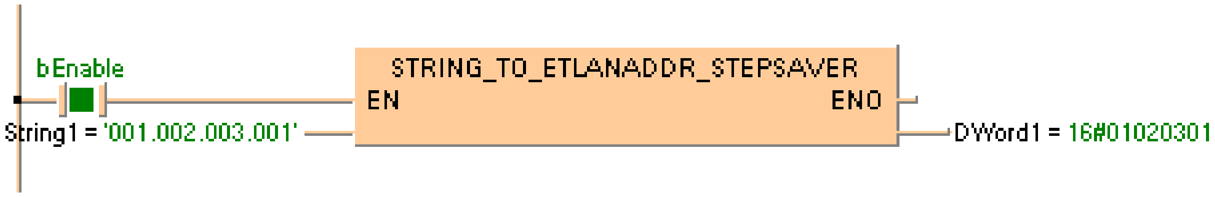
STRING (IP-address format 00a.0bb.0cc.ddd) to ETLAN Address

Description This function converts a STRING in IP address format into a value of the data type DWORD.

The function uses for approx. 50 steps of generated code the basic instruction F76_A2BIN (see page 637). The instruction expects that each octet consists of three characters with leading zeros. Otherwise the PLC delivers an operation error.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Example:



Permissible format:

'Octet1.Octet2.Octet3.Octet4[Space]', e.g.: '   [192.168.206.4]   '

Permissible characters:

Octets 1-4	Decimal numbers "0“-"9“, maximal 3 positions, with or without leading zeros in the range 0-255

☞ **If the format is wrong the result is 0.**

The conversion is such that the highest byte of the ET-LAN address represents the first octet and lowest byte of the IP address the fourth octet. This format for ET-LAN addresses is used, for example, by the FP Serie's ET-LAN modules.

Chapter 8

Selection instructions

MAX

Maximum value

Description MAX determines the input variable with the highest value.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** MAX **(see page 1328)**

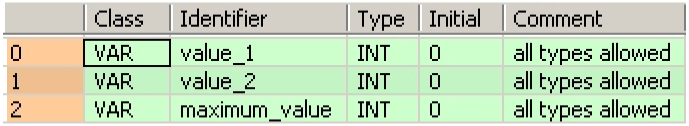
The number of input contacts lies in the range of 2 to 28.

Data types

Data type	I/O	Function
all except STRING	1st input	value 1
all except STRING	2nd input	value 2
all except STRING	output as input	result, whichever input variable's value is greater

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

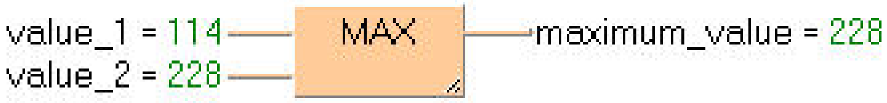


	Class	Identifier	Type	Initial	Comment
0	VAR	value_1	INT	0	all types allowed
1	VAR	value_2	INT	0	all types allowed
2	VAR	maximum_value	INT	0	all types allowed

In this example the input variables (**value_1** and **value_2**) have been declared. Instead, you may enter a constant directly at the input contact of a function.

Body **Value_1** and **value_2** are compared with each other. The maximum value of all input variables is written in **maximum_value**.

LD



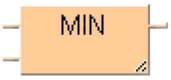
ST When programming with structured text, enter the following:

```
maximum_value:=MAX(value_1, value_2);
```

MIN

Minimum value

Description MIN detects the input variable with the lowest value.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** MIN **(see page 1328)**

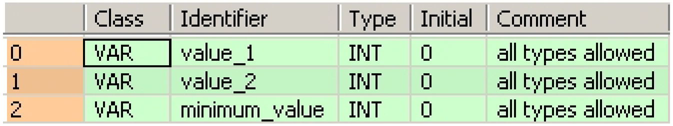
☞ **The number of input contacts lies in the range of 2 to 28.**

Data types

Data type	I/O	Function
all except STRING	1st input	value 1
all except STRING	2nd input	value 2
all except STRING	output as input	result, whichever input variable's value is smallest

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

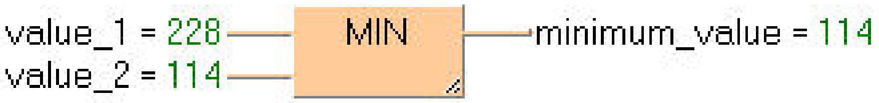


	Class	Identifier	Type	Initial	Comment
0	VAR	value_1	INT	0	all types allowed
1	VAR	value_2	INT	0	all types allowed
2	VAR	minimum_value	INT	0	all types allowed

In this example the input variables (**value_1** and **value_2**) have been declared. Instead, you may enter a constant directly at the input contact of a function.

Body **Value_1** and **value_2** are compared with each other. The lower value of the two is written into **minimum_value**.

LD



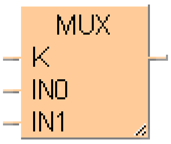
ST When programming with structured text, enter the following:

```
minimum_value:=MIN(value_1, value_2);
```

MUX

Select value from multiple channels

Description The function Multiplexer selects an input variable and writes its value into the output variable. The 1st input variable determines which input variable (IN1or IN2 ...) is to be written into the output variable. The function MUX can be configured for any desired number of inputs.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** MUX **(see page 1328)**

- **When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.**
- **For further information refer to the online help: Upgrade Problems with Data Type STRING**
- **The difference between the functions MUX and SEL (see page 257) is that in MUX with an integer value you can select between plural channels, and in SEL with a Boolean value only between two channels.**
- **The number of input contacts lies in the range of 2 to 28.**

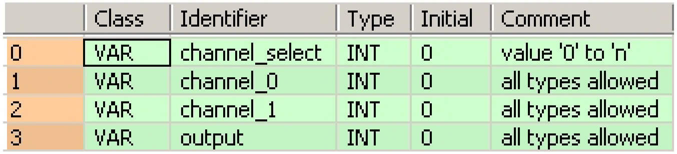
Data types

Data type	I/O	Function
INT	1st input	selects channel for 2nd or 3rd input value to be written to
all data types	2nd input	value 1
all data types	3rd input	value 2
all data types	output as 2nd and 3rd input	result

The 2nd and 3rd input variables must be of the same data type.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

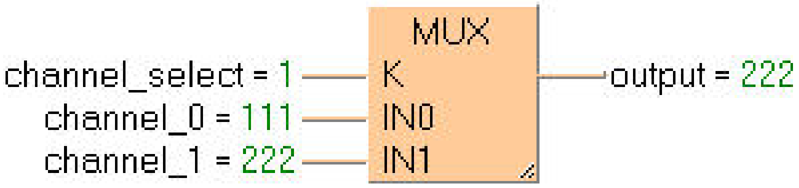


	Class	Identifier	Type	Initial	Comment
0	VAR	channel_select	INT	0	value '0' to 'n'
1	VAR	channel_0	INT	0	all types allowed
2	VAR	channel_1	INT	0	all types allowed
3	VAR	output	INT	0	all types allowed

In this example the input variables (**channel_select, channel_0** and **channel_1**) have been declared. Instead, you may enter a constant directly at the input contact of a function.

Body In **channel_select** you find the integer value (0, 1...n) for the selection of **channel_0** or **channel_1**. The result will be written into **output**.

LD



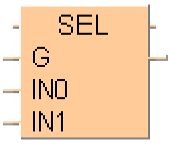
ST When programming with structured text, enter the following:

```
output:=MUX( K:= channel_select , IN0:= channel_0 ,
        IN1:= channel_1 );
```

SEL

Select value from one of two channels

Description With the first input variable (data type BOOL) of SEL you define which input variable is to be written into the output variable. If the Boolean value = 0 (FALSE), the input variable **IN0** will be written into the output variable, otherwise **IN1**.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** SEL **(see page 1330)**

☞

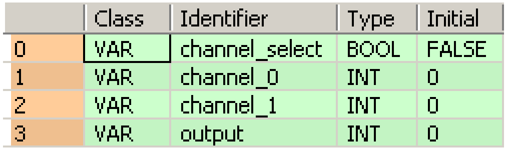
- **When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.**
- **For further information refer to the online help: Upgrade Problems with Data Type STRING**
- **The difference between the functions SEL and MUX (see page 255) is that in case of SEL a Boolean value serves for the channel selection, and in case of MUX an integral number (INT). Therefore, you can choose between more than two channels with MUX.**

Data types

Data type	I/O	Function
BOOL	G	selects between input value **IN0** or **IN1**
all data types	IN0	value is written into the output variable if **G** = FALSE
all data types	IN1	value is written into the output variable if **G** = TRUE
all data types	output	result value as **IN0** or **IN1**

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables which are required for programming the function are declared in the POU header.

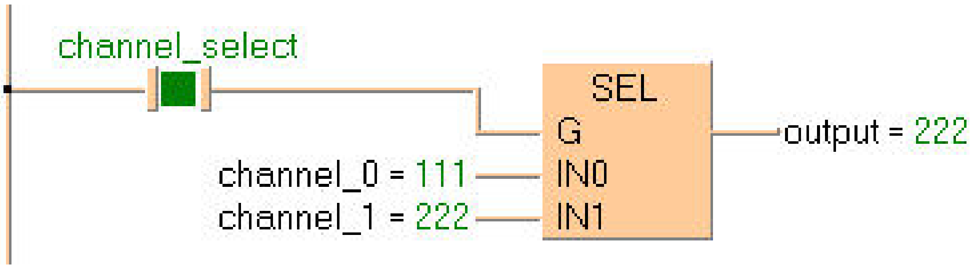


	Class	Identifier	Type	Initial
0	VAR	channel_select	BOOL	FALSE
1	VAR	channel_0	INT	0
2	VAR	channel_1	INT	0
3	VAR	output	INT	0

In this example the input variables (**channel_select**, **channel_0** and **channel_1**) have been declared. Instead, you may enter a constant directly at the input contact of a function.

Body If **channel_select** has the value 0, **channel_0** will be written into output, otherwise **channel_1**.

LD



ST When programming with structured text, enter the following:

```
output := SEL(G := channel_select, IN0 := channel_0, IN1 := channel_1);
```

Chapter 9

String instructions

LEN

String Length

Description LEN calculates the length of the input string and writes the result into the output variable.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** LEN **(see page 1328)**

- **If the string is longer than the length defined for the input variable (input_string) in the field "Type", an error occurs (see Special Internal Relays for Error Handling).**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**
- **When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.**
- **For further information refer to the online help: Upgrade Problems with Data Type STRING**

Data types

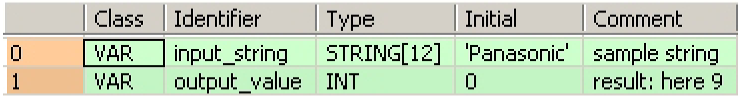
Data type	I/O	Function
STRING	input	input data type
INT	output	length of string

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input string is longer than the length defined for the input variable in the field "Type"
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

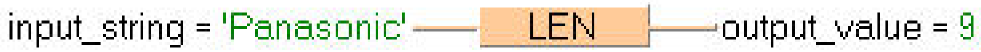


	Class	Identifier	Type	Initial	Comment
0	VAR	input_string	STRING[12]	'Panasonic'	sample string
1	VAR	output_value	INT	0	result: here 9

In this example the input variable (**input_string)** has been declared. Instead, you may enter the string (**'Panasonic'**) directly into the function. The string has to be put in inverted commas, both in the POU header and in the function.

Body The length (9) of **input_string** (‘**Panasonic**’) is written into **output_value**.

LD



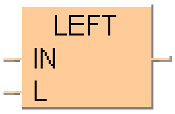
ST When programming with structured text, enter the following:

```
output_value:=LEN(input_value);
```

LEFT

Copy characters from the left

Description LEFT copies, starting from the left, **n** characters of the string of the first input variable to the output variable. You define the number of characters to be delivered **n** by the second input variable.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** LEFT **(see page 1328)**

☞

- **If the number of characters to be delivered is greater than the input string, the complete string will be copied to the output variable (output_string).**
- **If the output string is longer than the length defined for the output variable in the field "Type", only as many characters are copied from the left as the output variable can hold. The special internal relay R9009 (%MX0.900.9) is set.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

Data types

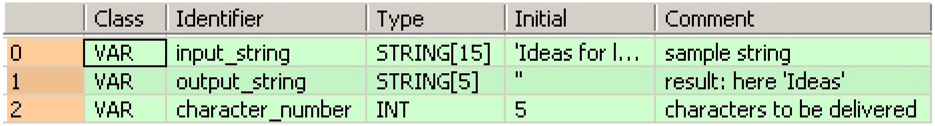
Data type	I/O	Function
STRING	1st input	input string
INT	2nd input	number of input string's characters that are copied, from the left
STRING	output	copied string

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input string is longer than the length defined for the input variable in the field "Type"
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	▪ output string is longer than the length defined for the output variable in the field "Type"

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

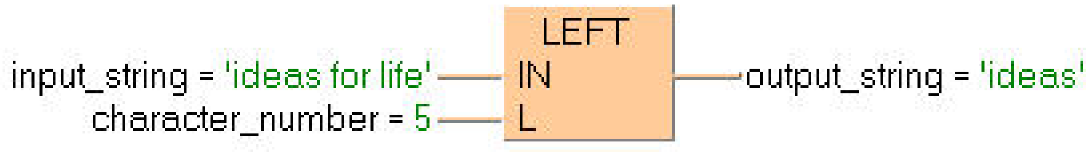


	Class	Identifier	Type	Initial	Comment
0	VAR	input_string	STRING[15]	'Ideas for I...	sample string
1	VAR	output_string	STRING[5]	''	result: here 'Ideas'
2	VAR	character_number	INT	5	characters to be delivered

In this example the input variables (**input_string** and **character_number**) have been declared. Instead, you may enter the string (**'Ideas for life'**) and the number of characters to be delivered directly into the function. The string has to be put in inverted commas, both in the POU header and in the function.

Body Starting from the left, **character_number** (5) of **input_string** (**‘Ideas for life’**) is copied to **output_string** (**‘Ideas’**).

LD



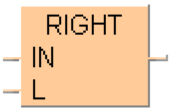
ST When programming with structured text, enter the following:

```
output_string:=LEFT(IN:=input_string, L:=character_number);
```

RIGHT

Copy characters from the right

Description RIGHT copies, starting from the right, n characters of the string of the first input variable to the output variable. You define the number of characters to be delivered n by the second input variable.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** RIGHT **(see page 1330)**

- **If the number of characters to be delivered is greater than the input string, the complete string will be copied to the output variable (output_string).**
- **If the output string is longer than the length defined for the output variable in the field "Type", only as many characters are copied from the left as the output variable can hold. The special internal relay R9009 (%MX0.900.9) is set.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help. (up to 200 steps)**

Data types

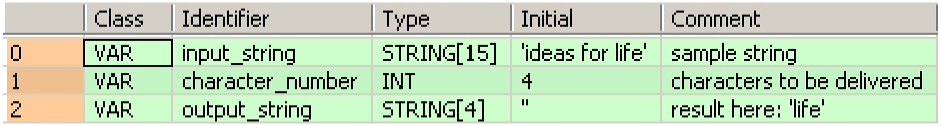
Data type	I/O	Function
STRING	1st input	input string
INT	2nd input	number of input string's characters that are copied, from the right
STRING	output	copied string

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input string is longer than the length defined for the input variable in the field "Type"
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	▪ output string is longer than the length defined for the output variable in the field "Type"

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables which are required for programming the function are declared in the POU header.

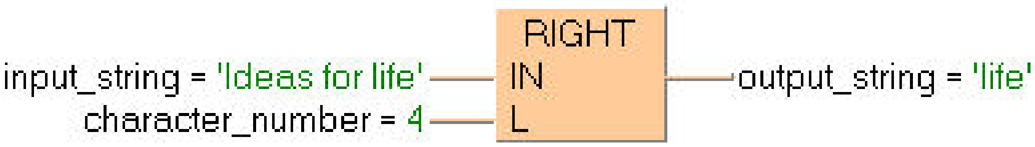


	Class	Identifier	Type	Initial	Comment
0	VAR	input_string	STRING[15]	'ideas for life'	sample string
1	VAR	character_number	INT	4	characters to be delivered
2	VAR	output_string	STRING[4]	''	result here: 'life'

In this example the input variables (**input_string** and **character_number**) have been declared. Instead, you may enter the string (**'Ideas for life'**) and the number of characters to be delivered directly into the function. The string has to be put in inverted commas, both in the POU header and in the function.

Body Starting from the right, **character_number** (4) of **input_string** (**‘Ideas for life’**) is copied to **output_string** (**‘life’**).

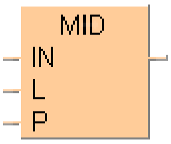
LD



MID

Copy characters from a middle position

Description MID copies **L** characters of the string **IN** starting at position **P** with 1 denoting the first character of the string. The result is written into the output variable.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of MID (see page 1328)**

- **The sum of start position and number of characters to be delivered should not be greater than the input string. If you want to receive for example 5 characters of a 10-character string, starting from position 7, only the last 4 characters are delivered.**
- **If the output string is longer than the length defined for the output variable (output_string) in the field "Type", only as many characters are copied from the start position as the output variable can hold. The special internal relay R9009 (%MX0.900.9) is set.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help. (up to 200 steps)**

Data types

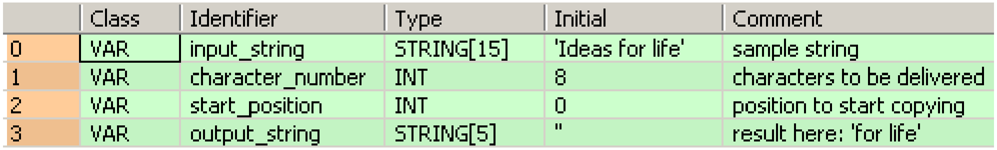
Data type	I/O	Function
STRING	1st input	input string
INT	2nd input	number of input string's characters that are copied
INT	3rd input	position where copying begins
STRING	output	copied string

Error Flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input string is longer than the length defined for the input variable in the field "Type" or start position is greater than the input string
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	▪ output string is longer than the length defined for the output variable in the field "Type"

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

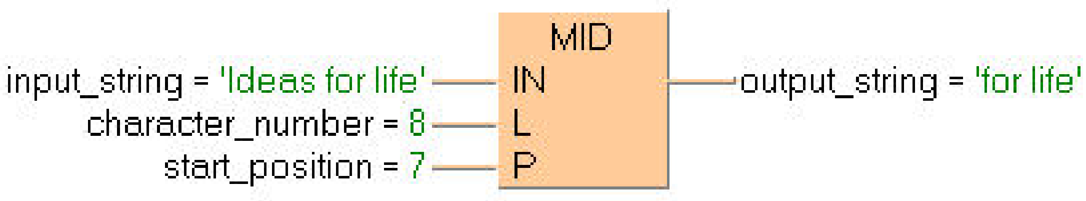


	Class	Identifier	Type	Initial	Comment
0	VAR	input_string	STRING[15]	'Ideas for life'	sample string
1	VAR	character_number	INT	8	characters to be delivered
2	VAR	start_position	INT	0	position to start copying
3	VAR	output_string	STRING[5]	''	result here: 'for life'

In this example the input variables (**input_string**, **character_number** and **start_position**) have been declared. Instead, you may enter the string (**'Ideas for life'**), the number of characters to be delivered and the start position directly into the function. The string has to be put in inverted commas, both in the POU header and in the function.

Body Starting from **start_position** (7), **character_number** (8) of **input_string** (**'Ideas for life'**) is copied to **output_string** (**'for life'**).

LD



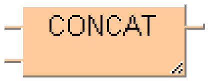
ST When programming with structured text, enter the following:

```
output_string:=MID(IN:=input_string, L:=character_number,
P:=start_position);
```

CONCAT

Concatenate (attach) a string

Description CONCAT concatenates (attaches) the second and the following input strings (IN1 + IN2 + ...) to the first input string and writes the resulting string into the output variable.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** CONCAT **(see page 1318)**

☞

- **If the output string is longer than the length defined for the output variable (output_string) in the field "Type", only as many characters are copied, starting from the left, as the output variable can hold. The special internal relay R9009 (%MX0.900.9) is set.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help.**

Data types

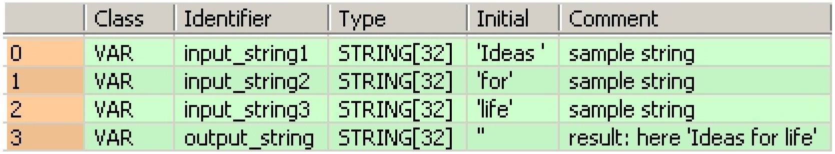
Data type	I/O	Function
STRING	1st input	beginning input string
STRING	2nd input	string that will be attached to the beginning string
STRING	output	resulting string

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input string is longer than the length defined for the input variable in the field "Type"
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	▪ output string is longer than the length defined for the output variable in the field "Type"

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

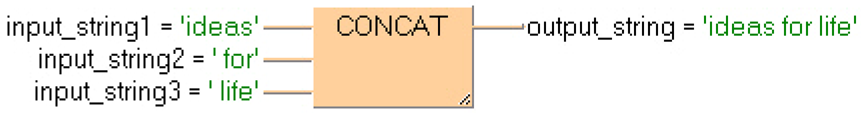


	Class	Identifier	Type	Initial	Comment
0	VAR	input_string1	STRING[32]	'Ideas '	sample string
1	VAR	input_string2	STRING[32]	'for'	sample string
2	VAR	input_string3	STRING[32]	'life'	sample string
3	VAR	output_string	STRING[32]	''	result: here 'Ideas for life'

In this example the input variables (**input_string1, input_string2** and **input_string3**) have been declared. However, you may enter the strings (**'Ideas'**, **' for' and ' life'**) directly into the function. The strings have to be put in inverted commas, both in the POU header and in the function.

Body **Input_string3** (**' life'**) is attached to **input_string2** (**' for'**) and this string is attached to **input_string1** (**'Ideas'**). The resulting string (**'Ideas for life'**) is written into **output_string**.

LD



input_string1 = 'ideas'
input_string2 = ' for'
input_string3 = ' life'
CONCAT
output_string = 'ideas for life'

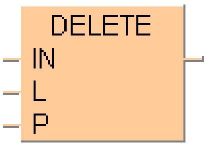
ST When programming with structured text, enter the following:

```
output_string:=CONCAT(input_string1, input_string2, input_string3);
```

DELETE

Delete characters from a string

Description DELETE deletes **L** characters in the string **IN** starting at position **P** with 1 denoting the first character of the string. The result is written into the output variable.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** DELETE **(see page 1319)**

- **If the output string is longer than the length defined for the output variable (output_string) in the field "Type", only as many characters are copied, starting from the left, as the output variable can hold. The special internal relay R9009 (%MX0.900.9) is set.**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help. (up to 200 steps)**

Data types

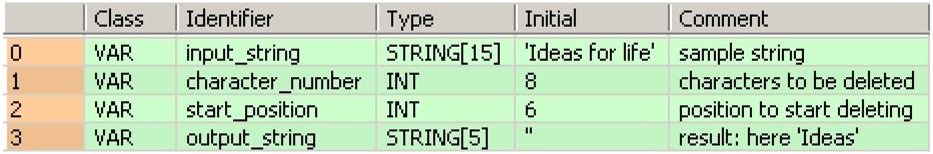
Data type	I/O	Function
STRING	1st input	input string
INT	2nd input	number of input string's characters that are deleted
INT	3rd input	position where deletion begins
STRING	output	resulting string

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input string is longer than the length defined for the input variable in the field "Type"
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	▪ output string is longer than the length defined for the output variable in the field "Type"

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

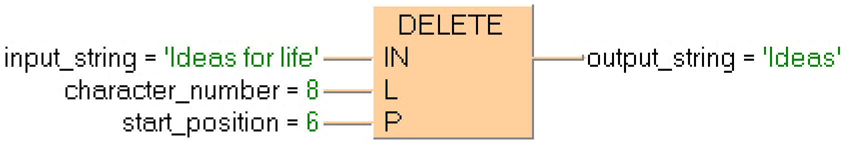


	Class	Identifier	Type	Initial	Comment
0	VAR	input_string	STRING[15]	'Ideas for life'	sample string
1	VAR	character_number	INT	8	characters to be deleted
2	VAR	start_position	INT	6	position to start deleting
3	VAR	output_string	STRING[5]	''	result: here 'Ideas'

In this example the input variables (**input_string**, **character_number** and **start_position**) have been declared. Instead, you may enter the string (**'Ideas for life'**), the number of characters to be deleted and the start position directly into the function. The string has to be put in inverted commas, both in the POU header and in the function.

Body Starting from **start_position** (**6**), **character_number** (**8**) is deleted from **input_string** (**'Ideas for life'**). The resulting string (**'Ideas')** is written into **output_string**.

LD



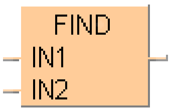
ST When programming with structured text, enter the following:

```
output_string:=DELETE(input_string, character_number, start_position);
```

FIND

Find string's position

Description FIND returns the position at which the second input string first occurs in the first input string. The result is written into the output variable. If the second input string does not occur in the first input string, the value ZERO is returned.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** FIND **(see page 1326)**

☞

- **If the strings are longer than the length defined for the input variables (input_string_1 and input_string_2) in the field "Type", an error occurs (see Special Internal Relays for Error Handling).**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help. (up to 200 steps)**

Data types

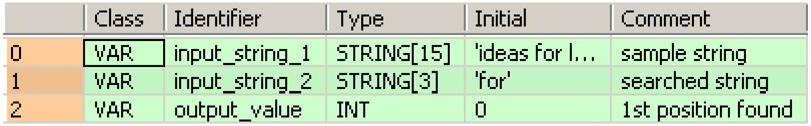
Data type	I/O	Function
STRING	1st input	input string
STRING	2nd input	string that is searched for in the input string
INT	output	position at which the string searched for is found

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input strings are longer than the length defined for the input variables in the field "Type"
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

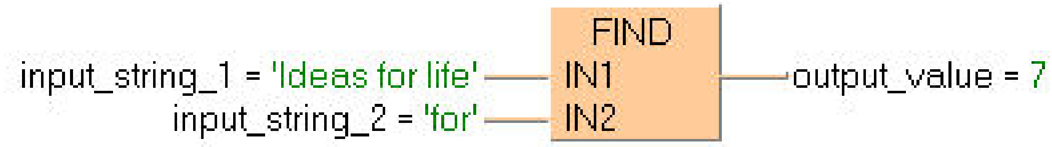


	Class	Identifier	Type	Initial	Comment
0	VAR	input_string_1	STRING[15]	'ideas for l...	sample string
1	VAR	input_string_2	STRING[3]	'for'	searched string
2	VAR	output_value	INT	0	1st position found

In this example the input variables (**input_string_1** and **input_string_2**) have been declared. Instead, you may enter the strings (**'Ideas for life'** and **'for'**) directly into the function. The strings have to be put in inverted commas, both in the POU header and in the function.

Body **Input_string_2** (**'for'**) is searched in **input_string_1** (**'Ideas for life'**). The position of the first occurrence (**7**) is written into **output_value**.

LD



input_string_1 = 'Ideas for life'

input_string_2 = 'for'

output_value = 7

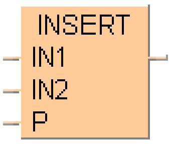
ST When programming with structured text, enter the following:

```
output_value:= FIND(input_string_1, input_string_2);
```

INSERT

Insert characters

Description INSERT inserts the string **IN2** into the string **IN1** beginning after the character position **P**, where 0 denotes the beginning of the string, 1 the position after the first string character, etc. The result is written into the output variable.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** INSERT **(see page 1327)**

- **If the strings are longer than the length defined for the input variables (input_string_1 and input_string_2) in the field "Type", an error occurs (see Special Internal Relays for Error Handling).**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help. (up to 200 steps)**

Data types

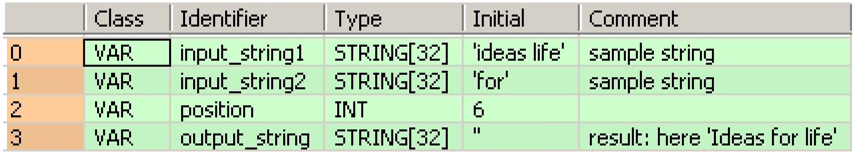
Data type	I/O	Function
STRING	1st input	input string
STRING	2nd input	string to be inserted into input string
INT	3rd input	position at which string is inserted
STRING	output	result string

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input strings are longer than the length defined for the input variables in the field "Type"
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

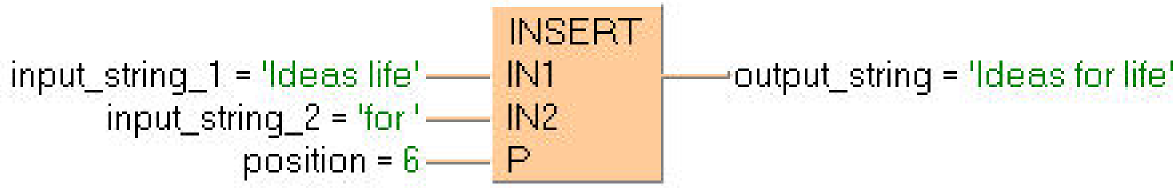
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	input_string1	STRING[32]	'ideas life'	sample string
1	VAR	input_string2	STRING[32]	'for'	sample string
2	VAR	position	INT	6	
3	VAR	output_string	STRING[32]	''	result: here 'Ideas for life'

Body In this example the input variables **input_string1**, **input_string2** and **position** have been declared. However, you may enter the values directly at the function's input contact pins instead. The STRING values have to be put in inverted commas, both in the POU header and at the contact pins. **input_string2** ('for ') is inserted into **input_string1** ('Ideas life') after character position 6. The result ('Ideas for life') is returned at **output_value**. In the LD example, 👓 (Monitoring) icon was activated while in online mode, hence you can see the results immediately.

LD



```
                              INSERT
input_string_1 = 'Ideas life' IN1    output_string = 'Ideas for life'
      input_string_2 = 'for ' IN2
                 position = 6 P
```

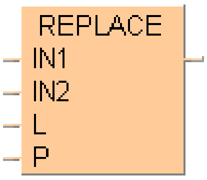
ST When programming with structured text, enter the following:

```
output_value:=INSERT(IN1:=input_string1, IN2:=input_string2, P:=6);
```

REPLACE

Replaces characters

Description REPLACE replaces the characters in the string **IN1** with **P** denoting the first position to be replaced and **L** denoting the number of characters to be replaced with the characters specified by **IN2**. The result is written into the output variable.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** REPLACE **(see page 1330)**

☞

- **If the strings are longer than the length defined for the input variables (input_string_1 and input_string_2) in the field "Type", an error occurs (see Special Internal Relays for Error Handling).**
- **The number of steps may vary depending on the PLC and parameters used, see also Table of Code Intensive Instructions in the online help. (up to 200 steps)**

Data types

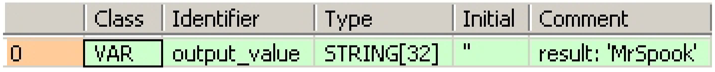
Data type	I/O	Function
STRING	1st input	input string
STRING	2nd input	replacement string
INT	3rd input	the number of characters in the input string to be replaced
INT	4th input	position at which characters begin to be replaced
STRING	output	resulting string

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ input strings are longer than the length defined for the input variables in the field "Type"
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

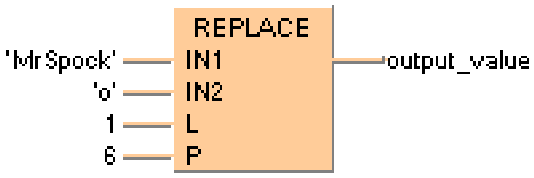
POU header All input and output variables which are required for programming the function are declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	output_value	STRING[32]	''	result: 'MrSpook'

Body In this example constant values are entered directly at the function's input contact pins. However, you may declare variables in the POU header. The STRING values have to be put in inverted commas, either in the POU header or at the contact pins. Here the 'c' in the STRING 'MrSpock' has been replaced with an 'o', yielding 'MrSpook'.

LD



Chapter 10

Date and time instructions

ADD_DT_TIME

Add TIME to DATE_AND_TIME

Description ADD_DT_TIME adds the value of a variable of the data type TIME to the date and time stored in the variable of the data type DATE_AND_TIME. The result is stored in a variable of the data type DATE_AND_TIME.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

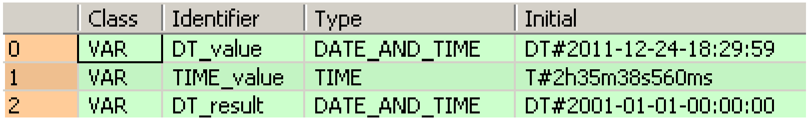
PLC types **Availability of ADD_DT_TIME (see page 1318)**

Data types

Data type	I/O	Function
DATE_AND_TIME	1st input	augend
TIME	2nd input	addend
DATE_AND_TIME	output	sum

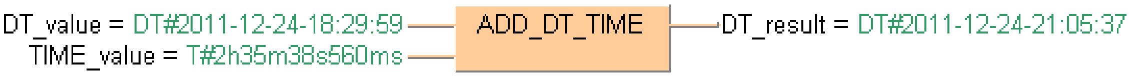
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	DT_value	DATE_AND_TIME	DT#2011-12-24-18:29:59
1	VAR	TIME_value	TIME	T#2h35m38s560ms
2	VAR	DT_result	DATE_AND_TIME	DT#2001-01-01-00:00:00

LD


DT_value = DT#2011-12-24-18:29:59
TIME_value = T#2h35m38s560ms
ADD_DT_TIME
DT_result = DT#2011-12-24-21:05:37

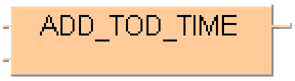
ST When programming with structured text, enter the following:

```
DT_result := ADD_DT_TIME(DT_value, TIME_value);
```

ADD_TOD_TIME

Add TIME to TIME_OF_DAY

Description ADD_TOD_TIME adds a variable of the data type TIME to the time of day. The result is stored in a variable of the data type TIME_OF_DAY.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

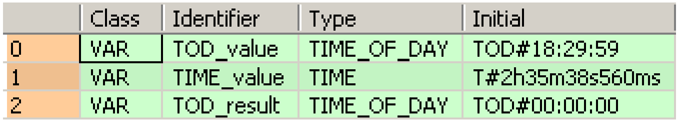
PLC types **Availability of ADD_TOD_TIME (see page 1318)**

Data types

Data type	I/O	Function
TIME_OF_DAY	1st input	augend
TIME	2nd input	addend
TIME_OF_DAY	output	sum

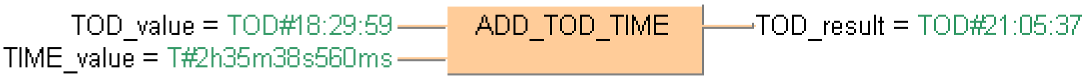
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	TOD_value	TIME_OF_DAY	TOD#18:29:59
1	VAR	TIME_value	TIME	T#2h35m38s560ms
2	VAR	TOD_result	TIME_OF_DAY	TOD#00:00:00

LD



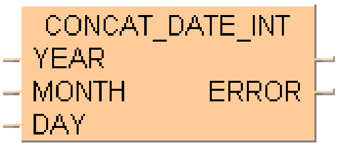
ST When programming with structured text, enter the following:

```
TOD_result := ADD_TOD_TIME(TOD_value, TIME_value);
```

CONCAT_DATE_INT

Concatenate INT values to form a date

Description CONCAT_DATE_INT concatenates the **INTEGER** values of year, month, and day.The result is stored in the output variable of the data type **DATE**. The Boolean output ERROR is set if the input values are invalid date or time values.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Inverse instruction: SPLIT_DATE_INT (see page 295)

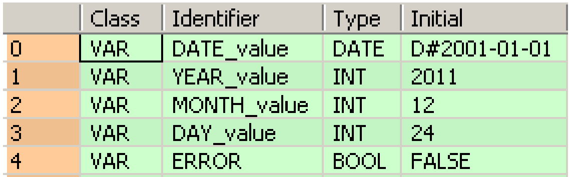
PLC types **Availability of CONCAT_DATE_INT (see page 1318)**

Data types

Data type	I/O	Function
INT	1st input 2nd input 3rd input	year month day
DATE	output	result
BOOL	output	The Boolean output ERROR is set if the input values are invalid date or time values.

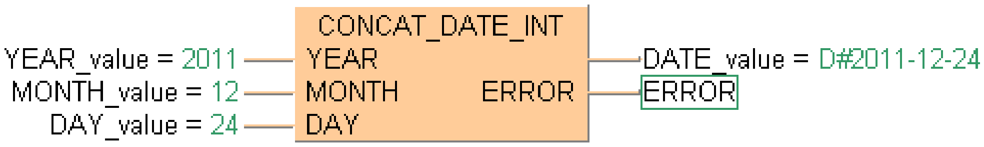
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	DATE_value	DATE	D#2001-01-01
1	VAR	YEAR_value	INT	2011
2	VAR	MONTH_value	INT	12
3	VAR	DAY_value	INT	24
4	VAR	ERROR	BOOL	FALSE

LD



ST When programming with structured text, enter the following:

```
DATE_value := CONCAT_DATE_INT(YEAR := YEAR_value,
          MONTH := MONTH_value,
          DAY := DAY_value,
          ERROR => ERROR);
```

CONCAT_DATE_TOD

Concatenate date and time of day

Description CONCAT_DATE_TOD concatenates a value of the data type DATE with a value of the data type TIME_OF_DAY.The result is stored in the output variable of the data type DATE_AND_TIME.

CONCAT_DATE_TOD

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

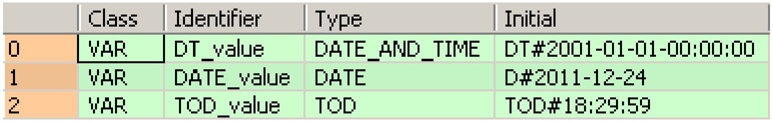
PLC types **Availability of ADD_TIME CONCAT_DATE_TOD (see page 1318)**

Data types

Data type	I/O	Function
DATE	1st input	date
TIME_OF_DAY	2nd input	time of day
DATE_AND_TIME	output	result

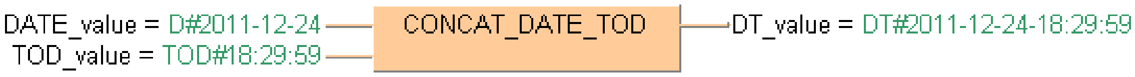
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	DT_value	DATE_AND_TIME	DT#2001-01-01-00:00:00
1	VAR	DATE_value	DATE	D#2011-12-24
2	VAR	TOD_value	TOD	TOD#18:29:59

LD



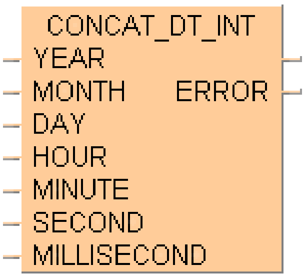
ST When programming with structured text, enter the following:

```
DT_value := CONCAT_DATE_TOD(DATE_value, TOD_value);
```

CONCAT_DT_INT

Concatenate INT values to form date and time

Description CONCAT_DT_INT concatenates the INT values of year, month, day, hour, minute, second, and millisecond. The result is stored in the output variable of the data type DATE_AND_TIME. The Boolean output ERROR is set if the input values are invalid date or time values.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Inverse instruction: SPLIT_DT_INT (see page 296)

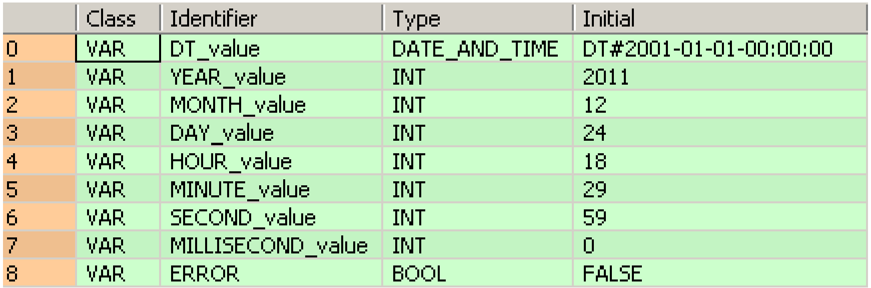
PLC types **Availability of CONCAT_DT_INT (see page 1318)**

Data types

Data type	I/O	Function
INT	1st input	year
	2nd input	month
	3rd input	day
	4th input	hour
	5th input	minute
	6th input	second
	7th input	millisecond
DATE_AND_TIME	output	result
BOOL	output	The Boolean output ERROR is set if the input values are invalid date or time values.

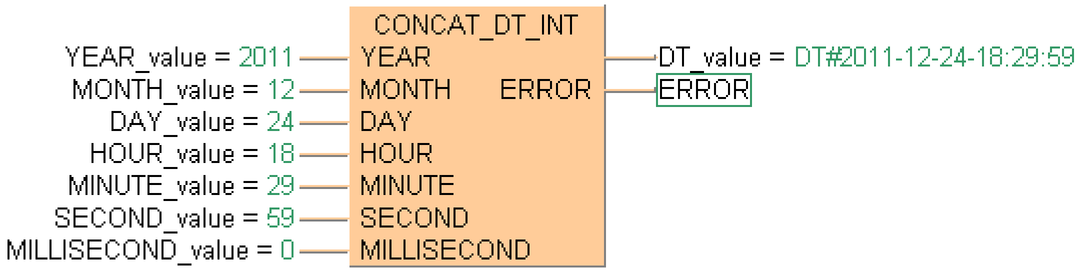
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	DT_value	DATE_AND_TIME	DT#2001-01-01-00:00:00
1	VAR	YEAR_value	INT	2011
2	VAR	MONTH_value	INT	12
3	VAR	DAY_value	INT	24
4	VAR	HOUR_value	INT	18
5	VAR	MINUTE_value	INT	29
6	VAR	SECOND_value	INT	59
7	VAR	MILLISECOND_value	INT	0
8	VAR	ERROR	BOOL	FALSE

LD



YEAR_value = 2011
MONTH_value = 12
DAY_value = 24
HOUR_value = 18
MINUTE_value = 29
SECOND_value = 59
MILLISECOND_value = 0

DT_value = DT#2011-12-24-18:29:59
ERROR

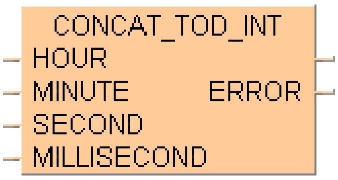
ST When programming with structured text, enter the following:

```
DT_value := CONCAT_DT_INT(YEAR := YEAR_value,
     MONTH := MONTH_value,
     DAY := DAY_value,
     HOUR := HOUR_value,
     MINUTE := MINUTE_value,
     SECOND := SECOND_value,
     MILLISECOND := MILLISECOND_value,
     ERROR => ERROR);
```

CONCAT_TOD_INT

Concatenate INT values to form the time of day

Description CONCAT_TOD_INT concatenates the INTEGER values for hour, minute, second, and millisecond. The result is stored in the output variable of the data type TIME_OF_DAY. The Boolean output ERROR is set if the input values are invalid date or time values.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Inverse instruction: SPLIT_TOD_INT (see page 298)

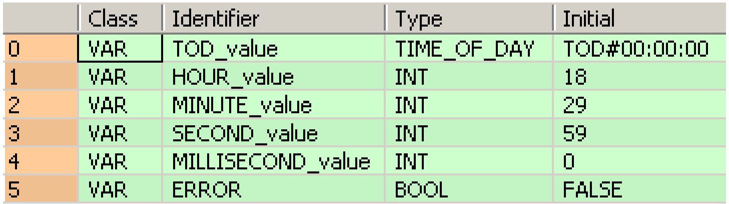
PLC types **Availability of CONCAT_TOD_INT (see page 1318)**

Data types

Data type	I/O	Function
INT	1st input	hour
	2nd input	minute
	3rd input	second
	4th input	millisecond
TIME_OF_DAY	output	result
BOOL	output	The Boolean output ERROR is set if the input values are invalid date or time values.

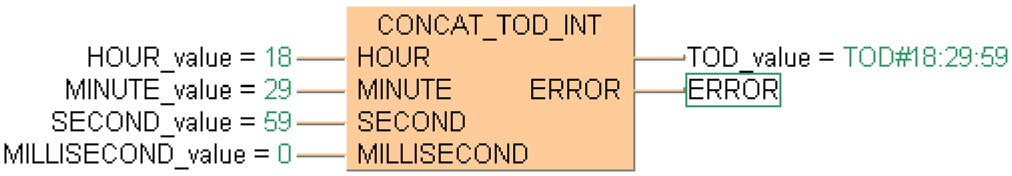
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	TOD_value	TIME_OF_DAY	TOD#00:00:00
1	VAR	HOUR_value	INT	18
2	VAR	MINUTE_value	INT	29
3	VAR	SECOND_value	INT	59
4	VAR	MILLISECOND_value	INT	0
5	VAR	ERROR	BOOL	FALSE

LD



ST When programming with structured text, enter the following:

```
TOD_value := CONCAT_TOD_INT(HOUR := HOUR_value,
                 MINUTE := MINUTE_value,
                 SECOND := SECOND_value,
                 MILLISECOND := MILLISECOND_value,
                 ERROR => ERROR);
```

DAY_OF_WEEK1

Return the day of the week

Description DAY_OF_WEEK1 returns the day of the week for any date as an INT. The number 1 corresponds to Monday; 7 corresponds to Sunday.

DAY_OF_WEEK1

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

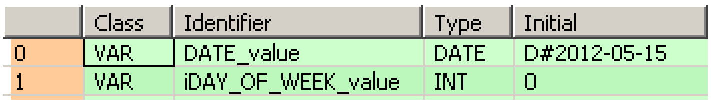
PLC types see page 1319

Data types

Data type	I/O	Function
DATE	input	date
ANY16	output	1 (Monday) – 7 (Sunday)

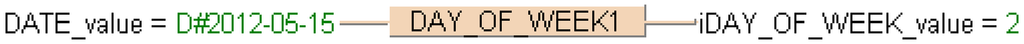
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	DATE_value	DATE	D#2012-05-15
1	VAR	iDAY_OF_WEEK_value	INT	0

LD



DATE_value = D#2012-05-15 DAY_OF_WEEK1 iDAY_OF_WEEK_value = 2

The value **iDAY_OF_WEEK_value** = 2 corresponds to Tuesday.

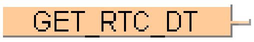
ST When programming with structured text, enter the following:

```
iDAY_OF_WEEK_value := DAY_OF_WEEK1(DATE_value);
```

GET_RTC_DT

Read the Real-Time Clock

Description GET_RTC_DT reads the PLC's real-time clock value for the clock/calendar function. If the PLC has no real-time clock or if the real-time clock is not functioning, the result is an invalid date and time value.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

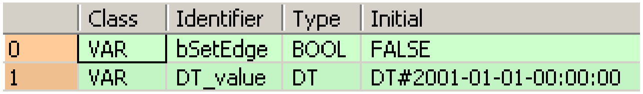
PLC types **Availability of GET_RTC_DT (see page 1326)**

Data types

Data type	I/O	Function
DATE_AND_TIME	output	date and time

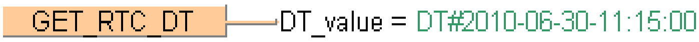
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bSetEdge	BOOL	FALSE
1	VAR	DT_value	DT	DT#2001-01-01-00:00:00

LD



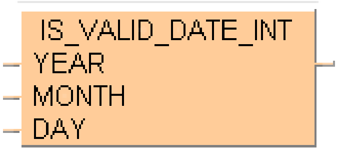
ST When programming with structured text, enter the following:

```
DT_value := GET_RTC_DT();
```

IS_VALID_DATE_INT

Check whether a DATE is valid

Description IS_VALID_DATE_INT checks whether the combination of the INT values for the year, month, and day is a valid DATE value. The Boolean output flag is set if the date is valid.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

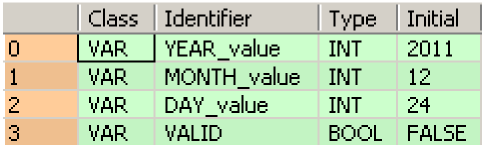
PLC types **Availability of IS_VALID_DATE_INT (see page 1327)**

Data types

Data type	I/O	Function
INT	1st input 2nd input 3rd input	year month day
BOOL	output	set to TRUE if the resulting date value is valid

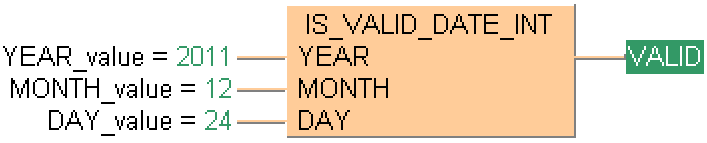
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	YEAR_value	INT	2011
1	VAR	MONTH_value	INT	12
2	VAR	DAY_value	INT	24
3	VAR	VALID	BOOL	FALSE

LD



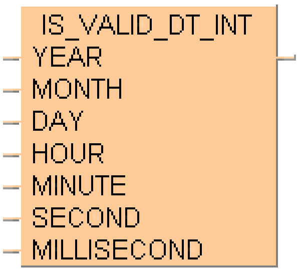
ST When programming with structured text, enter the following:

```
VALID := IS_VALID_DATE_INT(YEAR := YEAR_value,
     MONTH := MONTH_value,
     DAY := DAY_value);
```

IS_VALID_DT_INT

Check whether DATE_AND_TIME is valid

Description IS_VALID_DT checks whether the combination of INT values for year, month, day, hour, minute, second, and millisecond is a valid date and time value. The Boolean output flag is set if the date and time value is valid.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

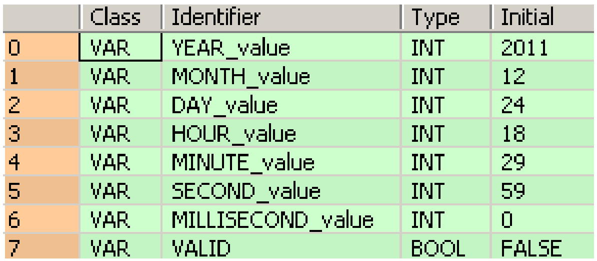
PLC types **Availability of IS_VALID_DT_INT (see page 1327)**

Data types

Data type	I/O	Function
INT	1st input	year
	2nd input	month
	3rd input	day
	4th input	hour
	5th input	minute
	6th input	second
	7th input	millisecond
BOOL	output	set to TRUE if the resulting date and time value is valid

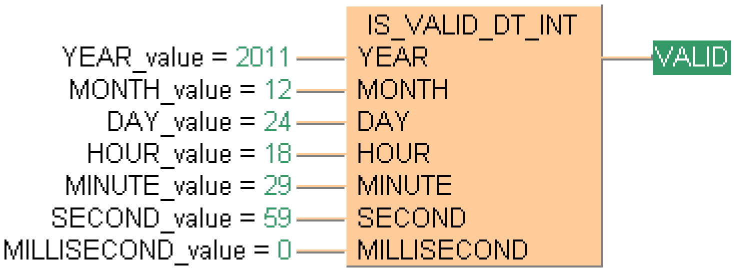
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	YEAR_value	INT	2011
1	VAR	MONTH_value	INT	12
2	VAR	DAY_value	INT	24
3	VAR	HOUR_value	INT	18
4	VAR	MINUTE_value	INT	29
5	VAR	SECOND_value	INT	59
6	VAR	MILLISECOND_value	INT	0
7	VAR	VALID	BOOL	FALSE

LD



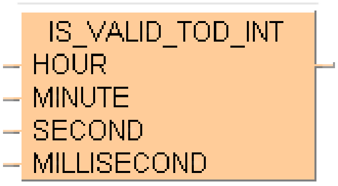
ST When programming with structured text, enter the following:

```
VALID := IS_VALID_DT_INT(YEAR := YEAR_value,
    MONTH := MONTH_value,
    DAY := DAY_value,
    HOUR := HOUR_value,
    MINUTE := MINUTE_value,
    SECOND := SECOND_value,
    MILLISECOND := MILLISECOND_value);
```

IS_VALID_TOD_INT

Check whether the TIME_OF_DAY is valid

Description IS_VALID_TOD_INT checks whether the combination of INT values for hour, minute, second, and millisecond is a valid time of day value. The Boolean output flag is set if the time of day value is valid.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

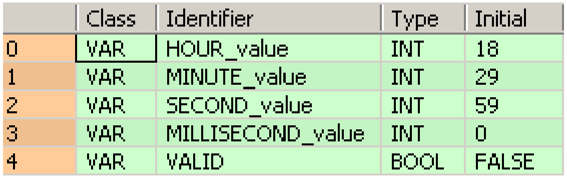
PLC types **Availability of IS_VALID_TOD_INT (see page 1328)**

Data types

Data type	I/O	Function
INT	1st input	hour
	2nd input	minute
	3rd input	second
	4th input	millisecond
BOOL	output	set to TRUE if the resulting time of day value is valid

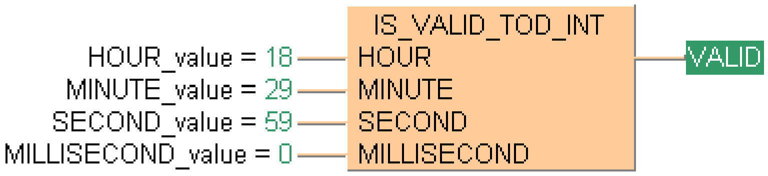
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	HOUR_value	INT	18
1	VAR	MINUTE_value	INT	29
2	VAR	SECOND_value	INT	59
3	VAR	MILLISECOND_value	INT	0
4	VAR	VALID	BOOL	FALSE

LD



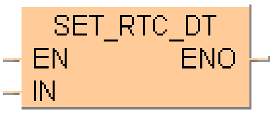
ST When programming with structured text, enter the following:

```
VALID := IS_VALID_TOD_INT(HOUR := HOUR_value,
    MINUTE := MINUTE_value,
    SECOND := SECOND_value,
    MILLISECOND := MILLISECOND_value);
```

SET_RTC_DT

Set the Real-Time Clock

Description SET_RTC_DT sets the real-time clock value in the PLC for the clock/calendar function. If the PLC has no real-time clock or if the real-time clock is not functioning, the result is an invalid date and time value.



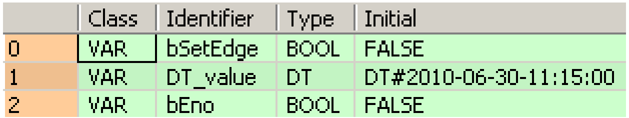
PLC types **Availability of SET_RTC_DT (see page 1330)**

Data types

Data type	I/O	Function
DATE_AND_TIME	input	date and time

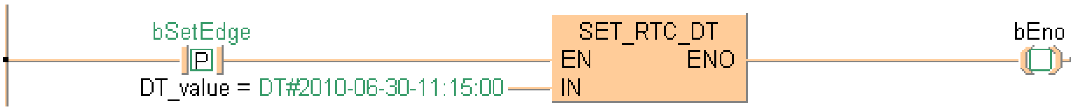
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bSetEdge	BOOL	FALSE
1	VAR	DT_value	DT	DT#2010-06-30-11:15:00
2	VAR	bEno	BOOL	FALSE

LD



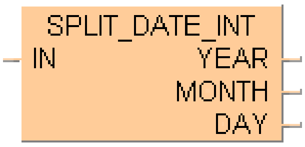
ST When programming with structured text, enter the following:

```
IF DF(bSetEdge) THEN
    SET_RTC_DT(DT_value);
END_IF;
```

SPLIT_DATE_INT

Split a date into INTEGER values

Description SPLIT_DATE_INT splits a value of the data type DATE into INT values for year, month, and day.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Inverse instruction: CONCAT_DATE_INT (see page 283)

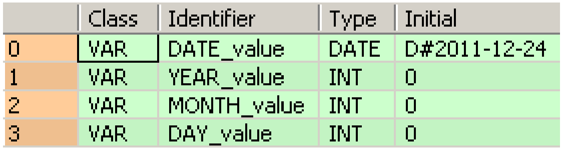
PLC types **Availability of SPLIT_DATE_INT (see page 1331)**

Data types

Data type	I/O	Function
DATE	input	date
INT	1st output	year
	2nd output	month
	3rd output	day

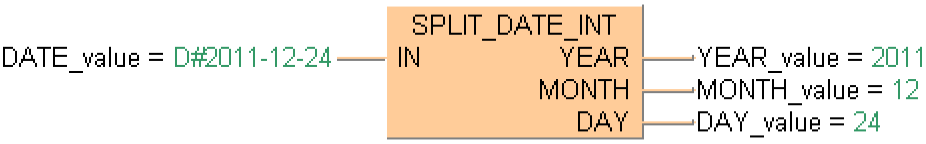
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	DATE_value	DATE	D#2011-12-24
1	VAR	YEAR_value	INT	0
2	VAR	MONTH_value	INT	0
3	VAR	DAY_value	INT	0

LD



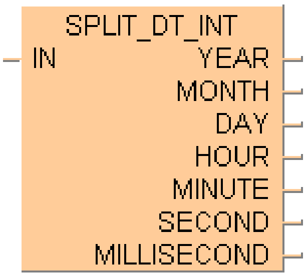
ST When programming with structured text, enter the following:

```
SPLIT_DATE_INT(IN := DATE_value,
                 YEAR => YEAR_value,
                 MONTH => MONTH_value,
                 DAY => DAY_value);
```

SPLIT_DT_INT

Split a date and time into INTEGER values

Description SPLIT_DT_INT splits a value of the data type DATE_AND_TIME into INT values for year, month, day, hour, minute, second, and millisecond.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Inverse instruction: CONCAT_DT_INT (see page 285)

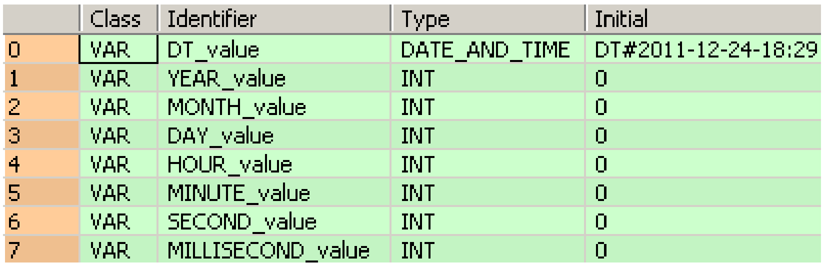
PLC types **Availability of SPLIT_DT_INT (see page 1331)**

Data types

Data type	I/O	Function
DATE_AND_TIME	input	date and time
INT	1st output	year
	2nd output	month
	3rd output	day
	4th output	hour
	5th output	minute
	6th output	second
	7th output	millisecond

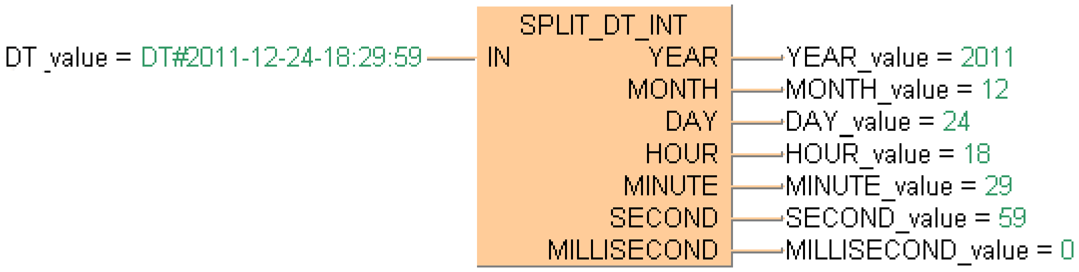
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	DT_value	DATE_AND_TIME	DT#2011-12-24-18:29
1	VAR	YEAR_value	INT	0
2	VAR	MONTH_value	INT	0
3	VAR	DAY_value	INT	0
4	VAR	HOUR_value	INT	0
5	VAR	MINUTE_value	INT	0
6	VAR	SECOND_value	INT	0
7	VAR	MILLISECOND_value	INT	0

LD



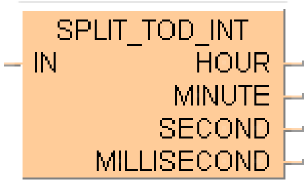
ST When programming with structured text, enter the following:

```
SPLIT_DT_INT(IN := DT_value,
               YEAR => YEAR_value,
               MONTH => MONTH_value,
               DAY => DAY_value,
               HOUR => HOUR_value,
               MINUTE => MINUTE_value,
               SECOND => SECOND_value,
               MILLISECOND => MILLISECOND_value);
```

SPLIT_TOD_INT

Split the time of day into INT values

Description SPLIT_TOD_INT splits a value of the data type TIME_OF_DAY into INT values for hour, minute, second, and millisecond.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Inverse instruction: CONCAT_TOD_INT (see page 287)

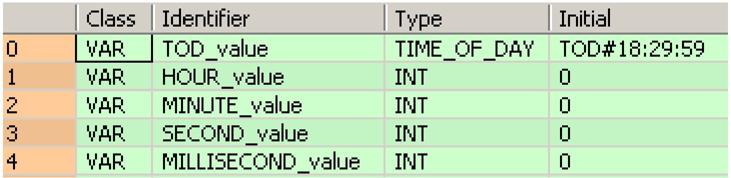
PLC types **Availability of SPLIT_TOD_INT (see page 1331)**

Data types

Data type	I/O	Function
TIME_OF_DAY	input	time of day
INT	1st output	hour
	2nd output	minute
	3rd output	second
	4th output	millisecond

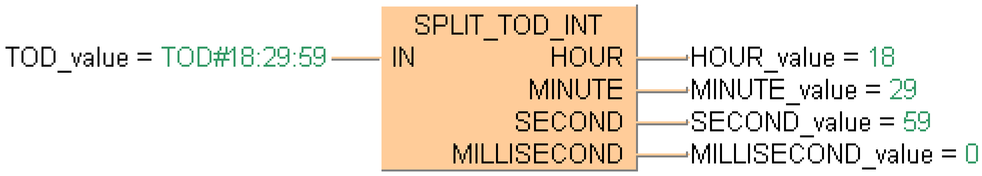
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	TOD_value	TIME_OF_DAY	TOD#18:29:59
1	VAR	HOUR_value	INT	0
2	VAR	MINUTE_value	INT	0
3	VAR	SECOND_value	INT	0
4	VAR	MILLISECOND_value	INT	0

LD



ST When programming with structured text, enter the following:

```
SPLIT_TOD_INT(IN := TOD_value,
                HOUR => HOUR_value,
                MINUTE => MINUTE_value,
                SECOND => SECOND_value,
                MILLISECOND => MILLISECOND_value);
```

SUB_DATE_DATE

Subtracts a date from another date

Description SUB_DATE_DATE subtracts a value of the data type DATE from another DATE value. The result is stored in the output variable of the data type TIME.

SUB_DATE_DATE

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** SUB_DATE_DATE **(see page 1331)**

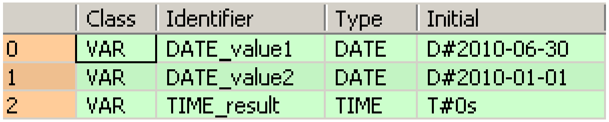
☞ **The TIME result is only valid if the difference between the minuend and subtrahend is smaller than or equal to the maximum TIME duration allowed. Otherwise an overflow of the TIME result variable occurs and the CARRY flag is set.**

Data types

Data type	I/O	Function
DATE	1st input	minuend
DATE	2nd input	subtrahend
TIME	output	result

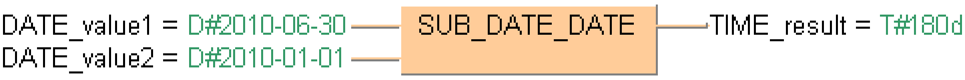
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	DATE_value1	DATE	D#2010-06-30
1	VAR	DATE_value2	DATE	D#2010-01-01
2	VAR	TIME_result	TIME	T#0s

LD



ST When programming with structured text, enter the following:

```
TIME_result := SUB_DATE_DATE(DATE_value1, DATE_value2);
```

SUB_DT_DT

Subtract date and time from date and time

Description SUB_DT_DT subtracts a value of the data type DATE_AND_TIME from another DATE_AND_TIME value. The result is stored in the output variable of the data type TIME.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of SUB_DT_DT (see page 1331)**

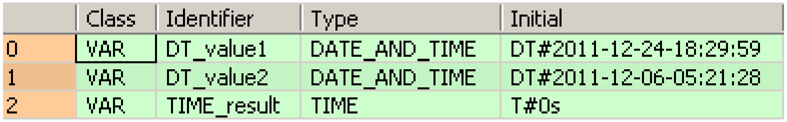
☞ **The TIME result is only valid if the difference between the minuend and subtrahend is smaller than or equal to the maximum TIME duration allowed. Otherwise an overflow of the TIME result variable occurs and the CARRY flag is set.**

Data types

Data type	I/O	Function
DATE_AND_TIME	1st input	minuend
DATE_AND_TIME	2nd input	subtrahend
TIME	output	result

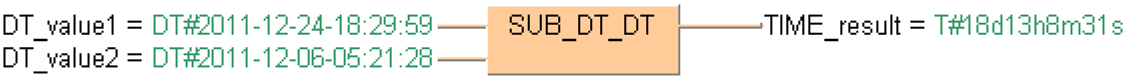
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	DT_value1	DATE_AND_TIME	DT#2011-12-24-18:29:59
1	VAR	DT_value2	DATE_AND_TIME	DT#2011-12-06-05:21:28
2	VAR	TIME_result	TIME	T#0s

LD


DT_value1 = DT#2011-12-24-18:29:59
DT_value2 = DT#2011-12-06-05:21:28
SUB_DT_DT
TIME_result = T#18d13h8m31s

ST When programming with structured text, enter the following:

```
TIME_result := SUB_DT_DT(DT_value1, DT_value2);
```

SUB_DT_TIME

Subtracts time from date and time

Description SUB_DT_TIME subtracts a value of the data type TIME from a value of the data type DATE_AND_TIME. The result is stored in the output variable of the data type TIME_OF_DAY.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

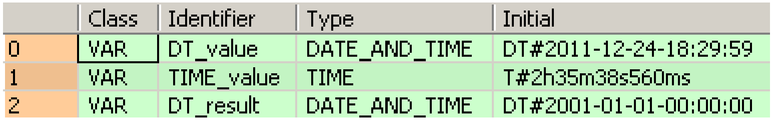
PLC types **Availability of SUB_DT_TIME (see page 1331)**

Data types

Data type	I/O	Function
DATE_AND_TIME	1st input	minuend
TIME	2nd input	subtrahend
DATE_AND_TIME	output	result

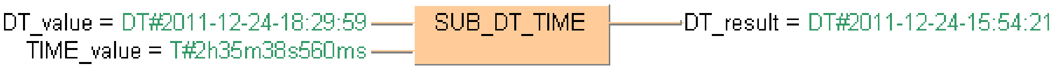
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	DT_value	DATE_AND_TIME	DT#2011-12-24-18:29:59
1	VAR	TIME_value	TIME	T#2h35m38s560ms
2	VAR	DT_result	DATE_AND_TIME	DT#2001-01-01-00:00:00

LD



DT_value = DT#2011-12-24-18:29:59
TIME_value = T#2h35m38s560ms
SUB_DT_TIME
DT_result = DT#2011-12-24-15:54:21

ST When programming with structured text, enter the following:

```
DT_result := SUB_DT_TIME(DT_value, TIME_value);
```

SUB_TOD_TIME

Subtracts a TIME value from the time of day

Description SUB_TOD_TIME subtracts a TIME value from a value of the data type TIME_OF_DAY. The result is stored in the output variable of the data type TIME_OF_DAY.

SUB_TOD_TIME

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

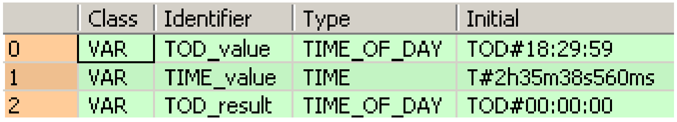
PLC types **Availability of SUB_TOD_TIME (see page 1331)**

Data types

Data type	I/O	Function
TIME_OF_DAY	1st input	minuend
TIME	2nd input	subtrahend
TIME_OF_DAY	output	result

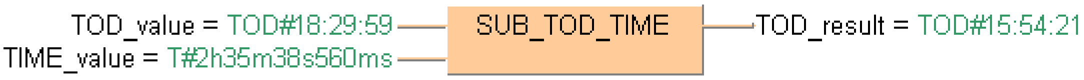
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	TOD_value	TIME_OF_DAY	TOD#18:29:59
1	VAR	TIME_value	TIME	T#2h35m38s560ms
2	VAR	TOD_result	TIME_OF_DAY	TOD#00:00:00

LD



TOD_value = TOD#18:29:59
TIME_value = T#2h35m38s560ms
SUB_TOD_TIME
TOD_result = TOD#15:54:21

ST When programming with structured text, enter the following:

```
TOD_result := SUB_TOD_TIME(TOD_value, TIME_value);
```

SUB_TOD_TOD

Subtract Time of Day from Time of Day

Description SUB_TOD_TOD subtracts a value of the data type TIME_OF_DAY from another TIME_OF_DAY value. The result is stored in the output variable of the data type TIME.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

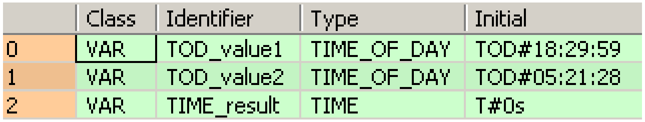
PLC types **Availability of** SUB_TOD_TOD **(see page 1331)**

Data types

Data type	I/O	Function
TIME_OF_DAY	1st input	minuend
TIME_OF_DAY	2nd input	subtrahend
TIME	output	result

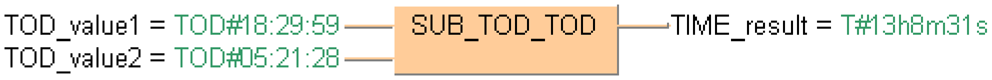
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	TOD_value1	TIME_OF_DAY	TOD#18:29:59
1	VAR	TOD_value2	TIME_OF_DAY	TOD#05:21:28
2	VAR	TIME_result	TIME	T#0s

LD



ST When programming with structured text, enter the following:

```
TIME_result := SUB_TOD_TOD(TOD_value1, TOD_value2);
```

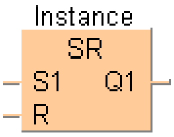
Chapter 11

Bistable instructions

SR

Set/reset

Description The function block SR (set/reset) allows you to both set and reset an output.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

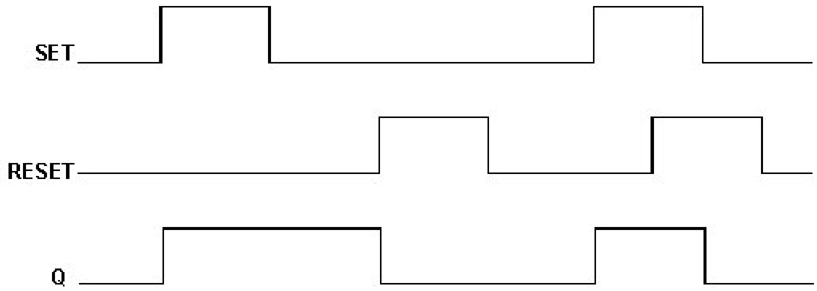
For SR declare the following:

SET (**S1**)	Set
	The output **Q** is set for each rising edge at SET
RESET (**R**)	reset
	The output **Q** is reset for each rising edge detected at RESET, except when SET is set (see time chart)
Q (**Q1**)	signal output
	is set if a rising edge is detected at SET; is reset if a rising edge is detected at RESET if SET is not set.

☞

- **The names in brackets are the valid parameter names of the ST-editor.**
- **Q is set if a rising edge is detected at both inputs (Set and Reset).**
- **Upon initialising, Q always has the status zero (reset).**

Time chart



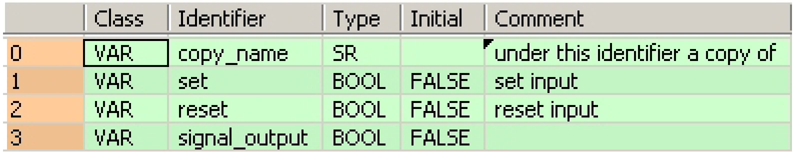
PLC types **Availability of SR (see page 1331)**

Data types

Data type	I/O	Function
BOOL	1st input	Set
	2nd input	reset
	output	set or reset depending on inputs

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

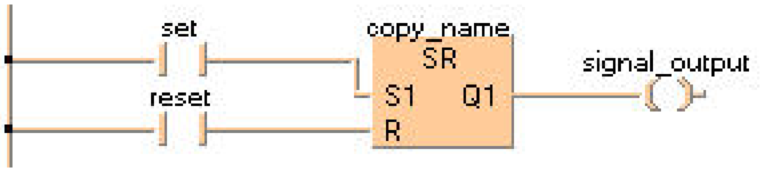
POU header All input and output variables which are used for programming the function block SR are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**, and a separate data area is reserved.



	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	SR		under this identifier a copy of
1	VAR	set	BOOL	FALSE	set input
2	VAR	reset	BOOL	FALSE	reset input
3	VAR	signal_output	BOOL	FALSE	

Body If **set** is set (status = TRUE), **signal_output** will be set. If only **reset** is set, the **signal_output** will be reset (status = FALSE). If both **set** and **reset** are set, **signal_output** will be set.

LD



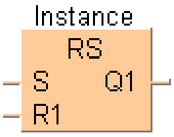
ST When programming with structured text, enter the following:

```
copy_name( SET:= set, RESET:= reset);
        signal_output:= signal_output;
```

RS

Reset/set

Description The function block RS (reset/set) allows you to both reset and set an output.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

For RS declare the following:

SET (S1) Set

The output **Q** is set for each rising edge at SET if RESET is not set.

RESET (R) reset

The output **Q** is reset for each rising edge at RESET.

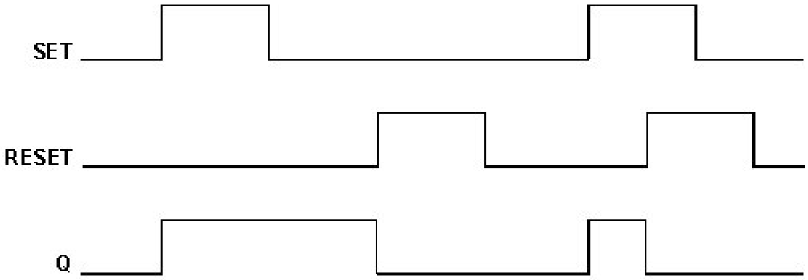
Q (Q1) signal output

is set if a rising edge is detected at SET and if RESET is not set; is reset if a rising edge is detected at RESET.

☞

- **The names in brackets are the valid parameter names of the ST-editor.**
- **Q is reset if a rising edge is detected at both inputs.**

Time chart



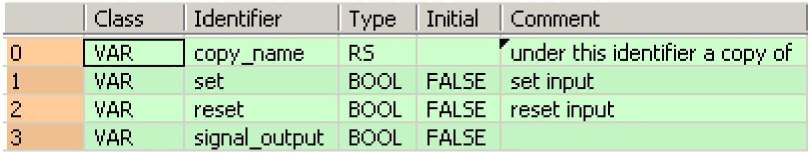
PLC types **Availability of RS (see page 1330)**

Data types

Data type	I/O	Function
BOOL	1st input	Set
	2nd input	reset
	output	set or reset depending on inputs

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

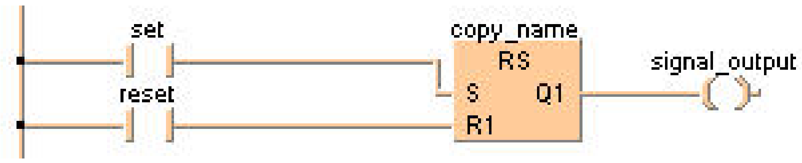
POU header All input and output variables which are used for programming the function block RS are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**, and a separate data area is reserved.



	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	RS		under this identifier a copy of
1	VAR	set	BOOL	FALSE	set input
2	VAR	reset	BOOL	FALSE	reset input
3	VAR	signal_output	BOOL	FALSE	

Body If **set** is set (status = TRUE) the **signal_output** will be set. If only **reset** is set, the **signal_output** will be reset (status = FALSE). If both **set** and **reset** are set, the **signal_output** will be reset to FALSE.

LD



ST When programming with structured text, enter the following:

```
copy_name( SET:= set, RESET:= reset);
       signal_output:= signal_output;
```

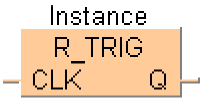
Chapter 12

Edge detection instructions

R_TRIG

Rising edge trigger

Description The function block R_TRIG (rising edge trigger) allows you to recognize a rising edge at an input.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

For R_TRIG declare the following:

CLK **signal input**

the output **Q** is set for each rising edge at the signal input (**CLK** = clock)

Q **signal output**

is set when a rising edge is detected at **CLK**.

PLC types **Availability of R_TRIG (see page 1330)**

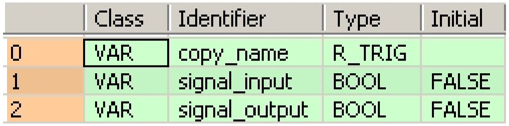
☞ **The output Q of a function block R_TRIG remains set for a complete PLC cycle after the occurrence of a rising edge (status change FALSE -> TRUE) at the CLK input and is then reset in the following cycle.**

Data types

Data type	I/O	Function
BOOL	input **CLK**	detects rising edge for clock
	output **Q**	set when rising edge detected

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

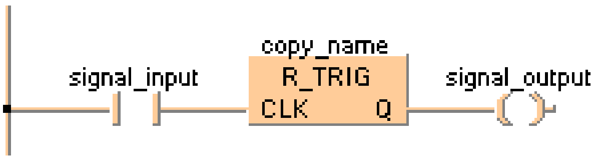
POU header All input and output variables which are used for programming the function block R_TRIG are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**, and a separate data area is reserved.



	Class	Identifier	Type	Initial
0	VAR	copy_name	R_TRIG	
1	VAR	signal_input	BOOL	FALSE
2	VAR	signal_output	BOOL	FALSE

Body **Signal_output** will be set, if a rising edge is detected at **signal_input.**

LD



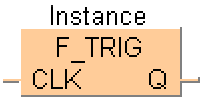
ST When programming with structured text, enter the following:

```
copy_name( CLK:= signal_input ,
        Q=> signal_output );
```

F_TRIG

Falling edge trigger

Description The function block **F_TRIG** (falling edge trigger) allows you to recognize a falling edge at an input.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

For F_TRIG declare the following:

CLK **signal input**
the output **Q** is set for each falling edge at the signal input (clk = clock)

Q **signal output**
is set if a falling edge is detected at **CLK**.

PLC types **Availability of F_TRIG (see page 1320)**

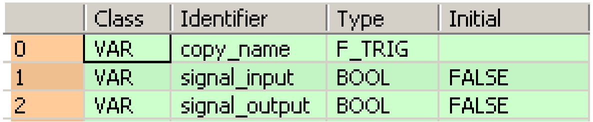
☞ **The output Q of a function block F_TRIG remains set for a complete PLC cycle after the occurrence of a falling edge (status change TRUE -> FALSE) at the CLK input and is then reset in the following cycle.**

Data types

Data type	I/O	Function
BOOL	input CLK	detects falling edge at input clock
	output Q	is set if falling edge is detected at input

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

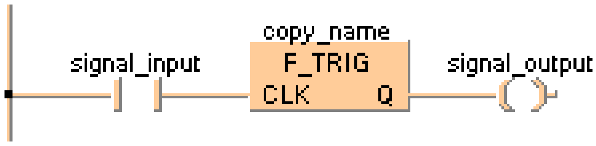
POU header All input and output variables which are used for programming the function block F_TRIG are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**, and a separate data area is reserved.



	Class	Identifier	Type	Initial
0	VAR	copy_name	F_TRIG	
1	VAR	signal_input	BOOL	FALSE
2	VAR	signal_output	BOOL	FALSE

Body **Signal_output** will be set, if a falling edge is detected at **signal_input.**

LD



ST When programming with structured text, enter the following:

```
copy_name( CLK:= signal_input ,
        Q=> signal_output );
```

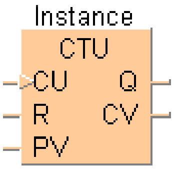
Chapter 13

Counter instructions

CTU

Up counter

Description The function block CTU (count up) allows you to program counting procedures.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

For CTU declare the following:

CU **clock generator**
the value 1 is added to **CV** for each rising edge at **CU**, except when RESET is set

RESET (R) **reset**
CV is reset to zero for each rising edge at RESET

PV **set value**
if **PV** (preset value) is reached, Q is set

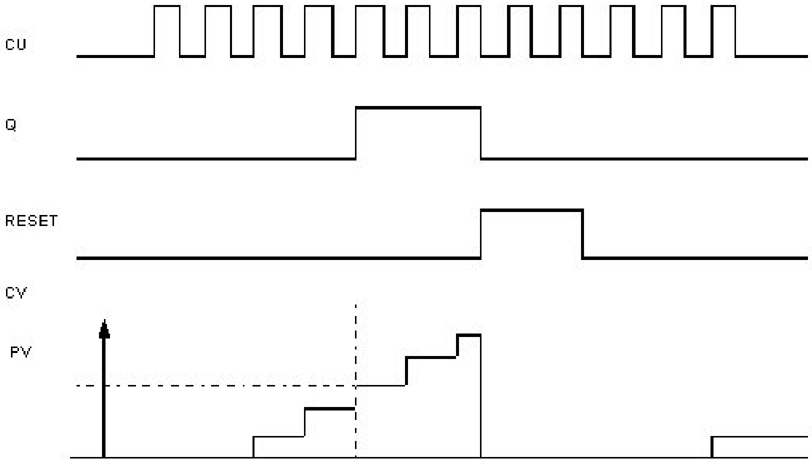
Q **signal output**
is set if **CV** is greater than/equal to **PV**

CV **current value**
contains the addition result (**CV** = current value)

☞ **The names in brackets are the valid parameter names of the ST-editor.**

PLC types **Availability of CTU (see page 1319)**

Time chart

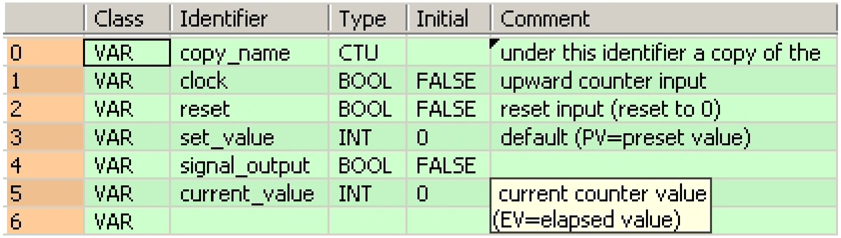


Data types

Data type	I/O	Function
BOOL	input **CU**	detects rising edge, adds 1 to **CV**
	input RESET	resets **CV** to 0 at rising edge
INT	input **PV**	set value
BOOL	output **Q**	set if **CV** >= **PV**
INT	output **CV**	current value

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

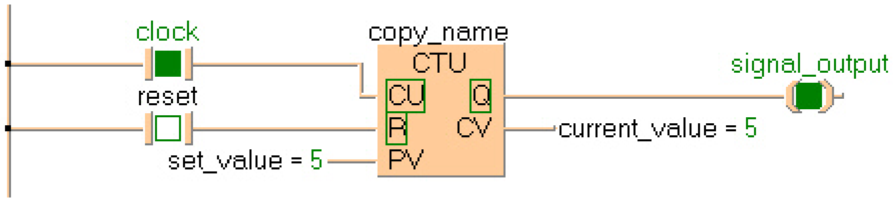
POU header All input and output variables which are used for programming the function block CTU are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**. A separate data area is reserved for this copy.



	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	CTU		under this identifier a copy of the
1	VAR	clock	BOOL	FALSE	upward counter input
2	VAR	reset	BOOL	FALSE	reset input (reset to 0)
3	VAR	set_value	INT	0	default (PV=preset value)
4	VAR	signal_output	BOOL	FALSE	
5	VAR	current_value	INT	0	current counter value (EV=elapsed value)
6	VAR				

Body If **reset** is set (status = TRUE), **current_value** (CV) will be reset. If a rising edge is detected at **clock**, the value 1 will be added to **current_value**. If a rising edge is detected at **clock,** this procedure will be repeated until **current_value** is greater than/equal to **set_value**. Then, **signal_output** will be set.

LD



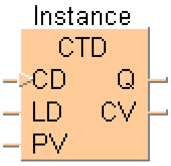
ST When programming with structured text, enter the following:

```
copy_name( CU:= clock, RESET:= reset, PV:= set_value, Q=> signal_output, CV=>
current_value);
```

CTD

Down counter

Description The function block CTD (count down) allows you to program counting procedures.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

For CTD declare the following:

CD **clock generator input**
the value 1 is subtracted from the current value **CV** for each rising edge detected at **CD**, except when LOAD is set or **CV** has reached the value zero.

LOAD (LD) **Set**
with LOAD the counter state is reset to **PV**

PV **preset value**
is the value subjected to subtraction during the first counting procedure

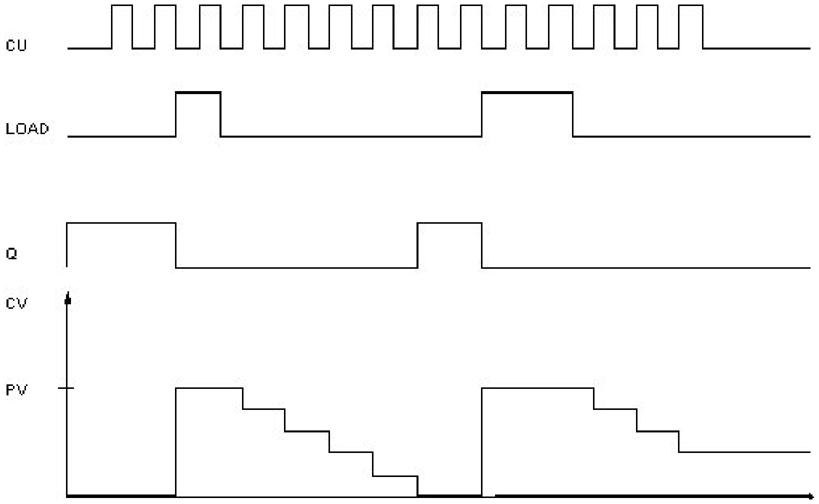
Q **signal output**
is set if **CV** = zero

CV **current value**
contains the current subtraction result (**CV** = current value)

☞ **The names in brackets are the valid parameter names of the ST-editor.**

PLC types **Availability of CTD (see page 1319)**

Time chart

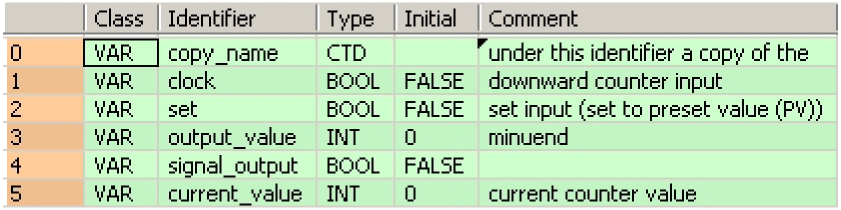


Data types

Data type	I/O	Function
BOOL	input **CD**	subtracts 1 from **CV** at rising edge
	input LOAD	resets counter to **PV**
INT	input **PV**	preset value
BOOL	output **Q**	signal output, set if CV = 0
INT	output **CV**	current value

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

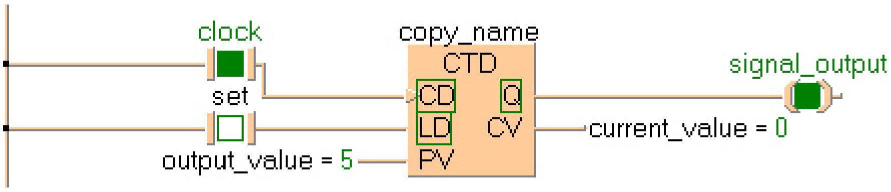
POU header All input and output variables which are used for programming the function block CTD are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**, and a separate data area is reserved.



	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	CTD		under this identifier a copy of the
1	VAR	clock	BOOL	FALSE	downward counter input
2	VAR	set	BOOL	FALSE	set input (set to preset value (PV))
3	VAR	output_value	INT	0	minuend
4	VAR	signal_output	BOOL	FALSE	
5	VAR	current_value	INT	0	current counter value

Body If **set** is set (status = TRUE), the **preset_value** (PV) is loaded in the **current_value** (CV). The value 1 will be subtracted from the **current_value** each time a rising edge is detected at **clock.** This procedure will be repeated until the **current_value** is greater than/equal to zero. Then, **signal_output** will be set.

LD



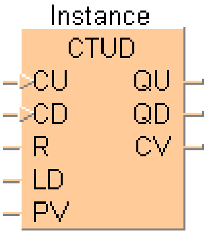
ST When programming with structured text, enter the following:

```
IF set THEN       (* first cycle *)
    load:=TRUE;         (* load has to be TRUE,
                         to set current_value to output_value *)
    clock:=FALSE;
END_IF;
copy_name(CD:= clock, LOAD:= set, PV:= output_value, Q=> signal_output, CV=>
current_value);
load:=FALSE;     (* now current_value got the right value, load doesn't need
to be *)
                 (* TRUE any longer *);
```

CTUD

Up/down counter

Description The function block CTUD (count up/down) allows you to program counting procedures (up and down).



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

For CTUD declare the following:

CU **count up**

the value 1 is added to the current **CV** for each rising edge detected at **CU**, except when RESET and/or LOAD is/are set.

CD **count down**

the value 1 is subtracted from the current **CV** for each rising edge detected at **CD**, except when RESET and/or LOAD is/are set and if **CU** and **CD** are simultaneously set. In the latter case, counting will be upwards.

RESET (R) **reset**

if RESET is set, **CV** will be reset

LOAD (LD) **Set**

if LOAD is set, **PV** is loaded to **CV**. This, however, does not apply if RESET is set simultaneously. In this case, LOAD will be ignored.

PV **preset value**

defines the preset value which is to be attained with the addition or subtraction (**PV** = preset value)

QU **signal output - count up**

is set if **CV** is greater than/equal to **PV**

QD **signal output - count down**

is set if **CV** = zero

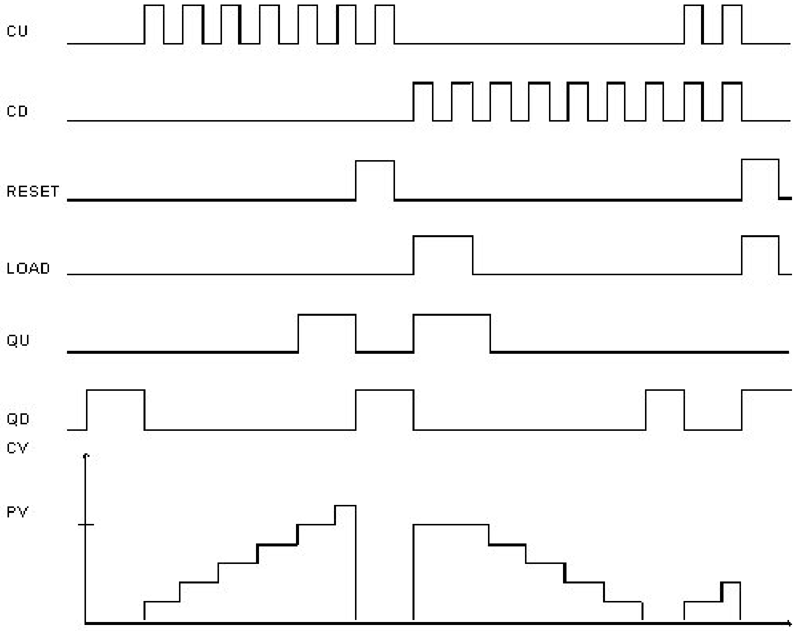
CV **current value**

is the addition/subtraction result (**CV** = current value)

☞ **The names in brackets are the valid parameter names of the ST-editor.**

PLC types **Availability of CTUD (see page 1319)**

Time chart



Data types

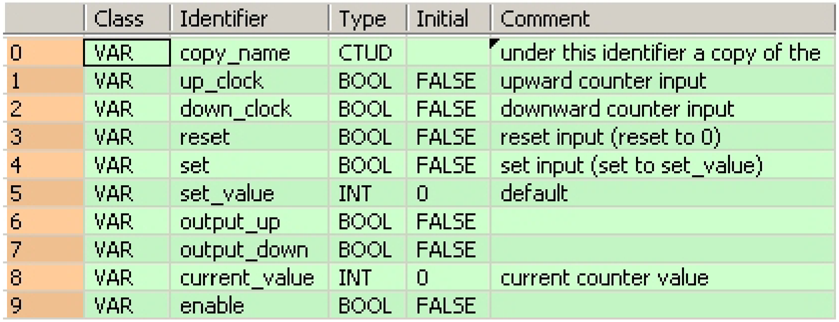
Data type	I/O	Function
BOOL	input **CU**	count up
	input **CD**	count down
	input **RESET**	resets **CV** if set
	input **LOAD**	loads **PV** to **CV**
INT	input **PV**	set value
BOOL	output **QU**	signal output count up
	output **QD**	signal output count down
INT	output **CV**	current value

Example

In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

All input and output variables which are used for programming the function block CTUD are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**. A separate data area is reserved for this copy.



	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	CTUD		under this identifier a copy of the
1	VAR	up_clock	BOOL	FALSE	upward counter input
2	VAR	down_clock	BOOL	FALSE	downward counter input
3	VAR	reset	BOOL	FALSE	reset input (reset to 0)
4	VAR	set	BOOL	FALSE	set input (set to set_value)
5	VAR	set_value	INT	0	default
6	VAR	output_up	BOOL	FALSE	
7	VAR	output_down	BOOL	FALSE	
8	VAR	current_value	INT	0	current counter value
9	VAR	enable	BOOL	FALSE	

Body

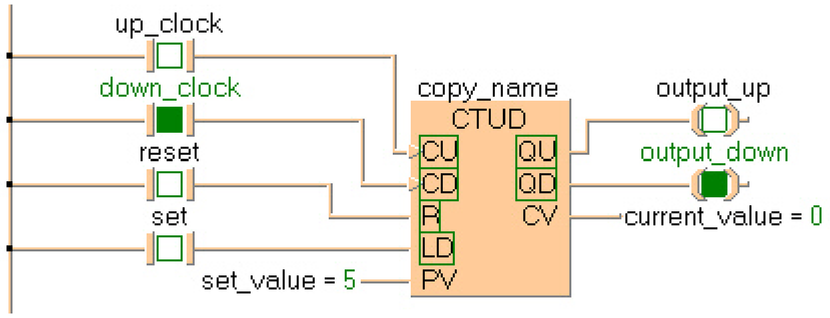
Count up:

If **reset** is set, the **current_value** (CV) will be reset. If up_**clock** is set, the value 1 is added to the **current_value**. This procedure is repeated for each rising edge detected at up_**clock** until the **current value** is greater than/equal to the **set_value**. Then **output_up** is set. The procedure is not conducted, if **reset** and/or **set** is/are set.

Count down:

If **set** is set (status = TRUE), the **set_value** (PV = preset value) will be loaded in the **current_value** (CV). If **down_clock** is set, the value 1 is subtracted from **set_value** at each clock. This procedure is repeated at each clock until the **current_value** is smaller than/equal to zero. Then, **signal_output** is set. The procedure will not be conducted, if **reset** and/or **set** is/are set or if CU and CV are set at the same time. In the latter case, counting will be downwards.

LD



ST

When programming with structured text, enter the following:

```
copy_name(CU:= up_clock, CD:= down_clock, RESET:= reset, LOAD:= set, PV:=
set_value,
             QU=> output_up, QD=> output_down, CV=> current_value);
```

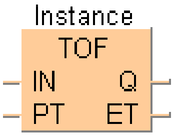
Chapter 14

Timer instructions

TOF

Timer with switch-off delay

Description The function block TOF allows you to program a switch-off delay, e.g. to switch off the ventilator of a machine at a later point in time than the machine itself.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

For TOF declare the following:

IN **timer ON**

an internal timer is started if a falling edge is detected at IN. If a rising edge is detected at IN before PT has reached its value, Q will not be switched off (see time chart, section ②)

PT **switch-off delay**

(16-bit value: 0 - 327.27s, 32-bit value: 0 - 21,474,836.47s; resolution 10ms each) the switch-off delay is defined here (PT = preset time)

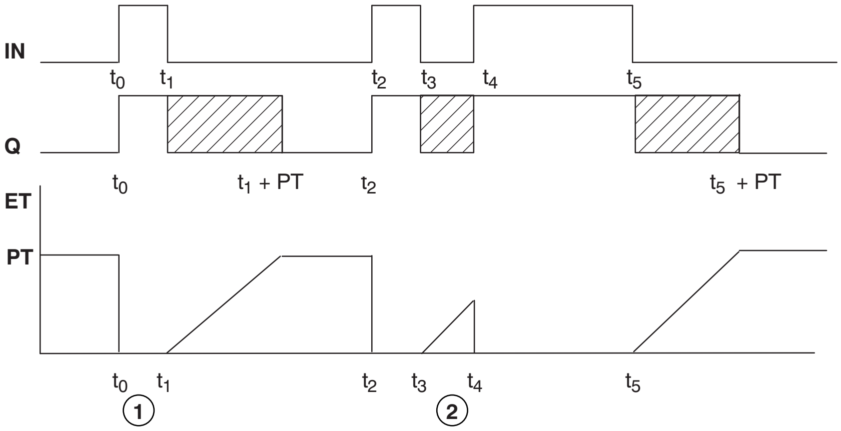
Q **signal output**

is reset if PT = ET

ET **elapsed time**

represents the current value of the elapsed time

Time chart



① **Q** is switched off with a delay corresponding to the time defined in **PT**. Switching on is carried out without delay.

② If **IN** (as in the time chart on top for t3 to t4) is set prior to the lapse of the delay time **PT**, **Q** remains set (time chart for t2 to t3).

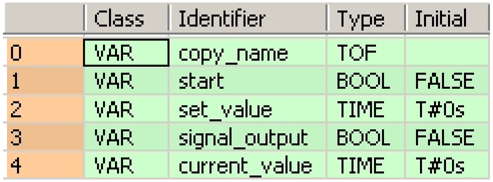
PLC types **Availability of TOF (see page 1332)**

Data types

Data type	I/O	Function
BOOL (IN)	input	internal timer on a falling edge
TIME (PT)	input	switch off delay
BOOL (Q)	output	signal output reset if PT = ET
TIME (ET)	output	elapsed time

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

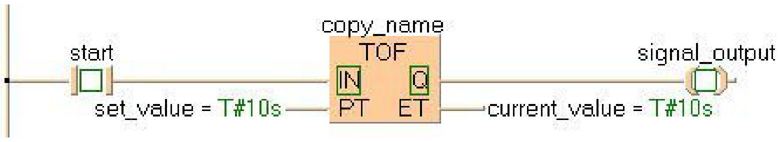
POU header All input and output variables which are used for programming the function block TOF are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**. A separate data area is reserved for this copy.



	Class	Identifier	Type	Initial
0	VAR	copy_name	TOF	
1	VAR	start	BOOL	FALSE
2	VAR	set_value	TIME	T#0s
3	VAR	signal_output	BOOL	FALSE
4	VAR	current_value	TIME	T#0s

Body If **start** is reset, this signal is transferred to **signal_output** with a delay corresponding to the period of time **set_value**.

LD



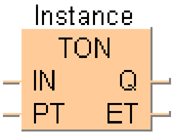
ST When programming with structured text, enter the following:

```
copy_name( IN:= start ,
       PT:= set_value ,
       Q=> signal_output ,
       ET=> current_value );
```

TON

Timer with switch-on delay

Description The function block TON allows you to program a switch-on delay.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

For TON declare the following:

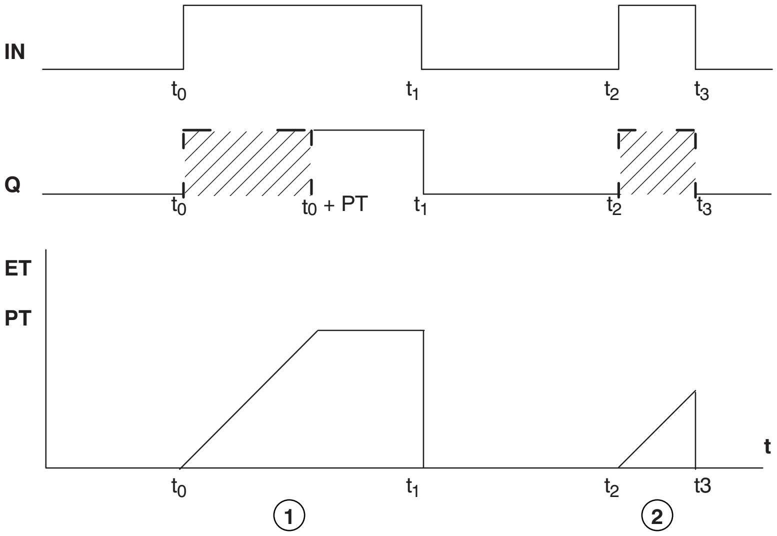
IN **timer ON**
an internal timer is started for each rising edge detected at IN

PT **switch-on delay**
(16-bit value: 0 - 327.27s, 32-bit value: 0 - 21,474,836.47s; resolution 10ms each) the switch-on delay is defined here (PT = preset time)

Q **signal output**
is set if PT = ET

ET **elapsed time**
indicates the current value of the elapsed time

Time chart



① **Q** is set delayed with the time defined in **PT**. Resetting is without any delay.

② If the input **IN** is only set for the period of the delay time **PT** or even for a shorter period of time (t3 - t2 < PT), **Q** will not be set.

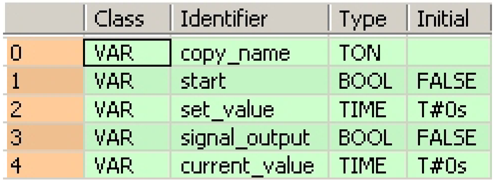
PLC types **Availability of TON (see page 1332)**

Data types

Data type	I/O	Function
BOOL (IN)	input	internal timer starts at rising edge
TIME (PT)	input	switch on delay
BOOL (Q)	output	signal output set if PT = ET
TIME (ET)	output	elapsed time

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

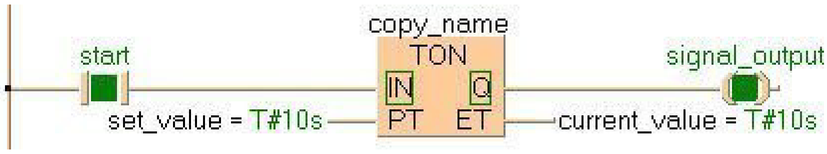
POU header All input and output variables which are used for programming the function block TON are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**. A separate data area is reserved for this copy.



	Class	Identifier	Type	Initial
0	VAR	copy_name	TON	
1	VAR	start	BOOL	FALSE
2	VAR	set_value	TIME	T#0s
3	VAR	signal_output	BOOL	FALSE
4	VAR	current_value	TIME	T#0s

Body If **start** is set (status = TRUE), the input signal is transferred to **signal_output** with a delay by the time period **set_value**.

LD



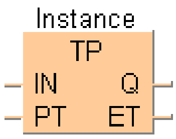
ST When programming with structured text, enter the following:

```
copy_name( IN:= start ,
       PT:= set_value ,
       Q=> signal_output ,
       ET=> current_value );
```

TP

Timer with defined period

Description The function block TP allows you to program a pulse timer with a defined clock period.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

For TP declare the following:

IN **clock generator**

if a rising edge is detected at **IN**, a clock is generated having the period defined in **PT**

PT **clock period**

(16-bit value: 0 - 327.27s, 32-bit value: 0 -21,474,836.47s; resolution 10ms each) a timer having the period **PT** is caused for each rising edge at **IN**. A new rising edge detected at **IN** within the pulse period does not cause a new timer (see time chart, section ②)

Q **signal output**

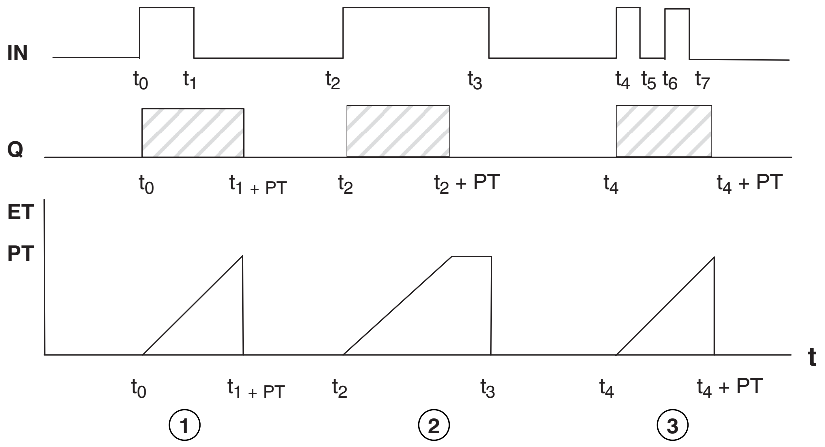
is set for the period of PT as soon as a rising edge is detected at **IN**

ET **elapsed time**

contains the elapsed period of the timer. If **PT** = **ET**, **Q** will be reset

☞ **FP2, FP2SH and FP10SH use a 32-bit value for PT.**

Time chart



① + ② Independent of the turn-on period of the **IN** signal, a clock is generated at the output **Q** having a length defined by **PT**. The function block TP is triggered if a rising edge is detected at the input **IN**.

③ A rising edge at the input **IN** does not have any influence during the processing of **PT**.

PLC types **Availability of TP (see page 1332)**

Data types

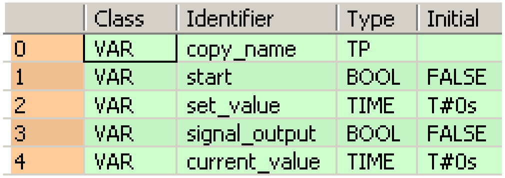
Data type	I/O	Function
BOOL	input **IN**	clock generated according to clock period at rising edge
TIME	input **PT**	clock period
BOOL	output **Q**	signal output
TIME	output **ET**	elapsed time

Example

In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

All input and output variables which are used for programming the function block TP are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**. A separate data area is reserved for this copy.

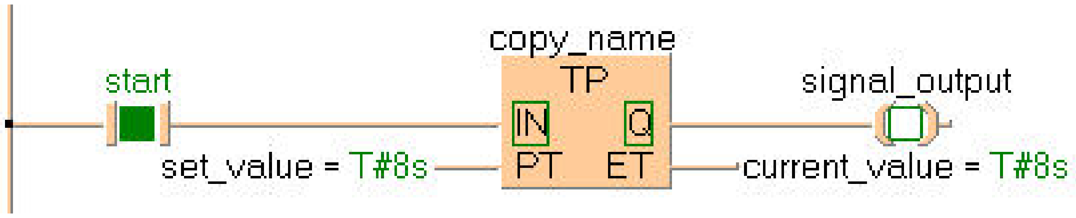


	Class	Identifier	Type	Initial
0	VAR	copy_name	TP	
1	VAR	start	BOOL	FALSE
2	VAR	set_value	TIME	T#0s
3	VAR	signal_output	BOOL	FALSE
4	VAR	current_value	TIME	T#0s

Body

If **start** is set (status = TRUE), the clock is emitted at **signal_output** until the **set_value** for the clock period is reached.

LD



ST

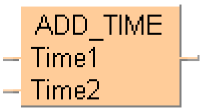
When programming with structured text, enter the following:

```
copy_name( IN:= start ,
       PT:= set_value ,
       Q=> signal_output ,
       ET=> current_value );
```

ADD_TIME

Add TIME

Description ADD_TIME adds the times of the two input variables and writes the sum in the output variable.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

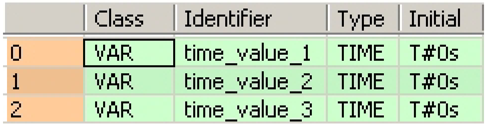
PLC types **Availability of ADD_TIME (see page 1318)**

Data types

Data type	I/O	Function
TIME	1st input	augend
TIME	2nd input	addend
TIME	output	sum

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

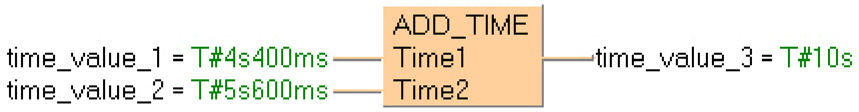


	Class	Identifier	Type	Initial
0	VAR	time_value_1	TIME	T#0s
1	VAR	time_value_2	TIME	T#0s
2	VAR	time_value_3	TIME	T#0s

In this example the input variables (**time_value_1** and **time_value_2**) have been declared. Instead, you may enter constants directly at the input contacts of a function.

Body **Time_value_1** and **time_value_2** are added. The result is written into **time_value_3**.

LD

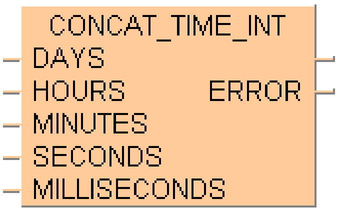


ST
```
time_value_3:=ADD_TIME(time_value_1, time_value_2);
```

CONCAT_TIME_INT

Concatenate INT values to form a time

Description The highest non-zero time unit may be greater than its apparent limit, e.g. T#25h is a valid time value whereas T#1d25h is not.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

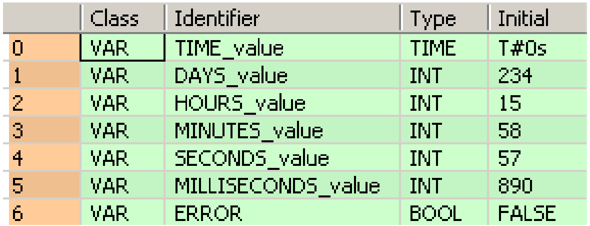
PLC types **Availability of CONCAT_TIME_INT (see page 1318)**

Data types

Data type	I/O	Function
INT	1st input	days
	2nd input	hours
	3rd input	minutes
	4th input	seconds
	5th input	milliseconds
TIME	output	result
BOOL	output	The Boolean output ERROR is set if the input values are invalid date or time values.

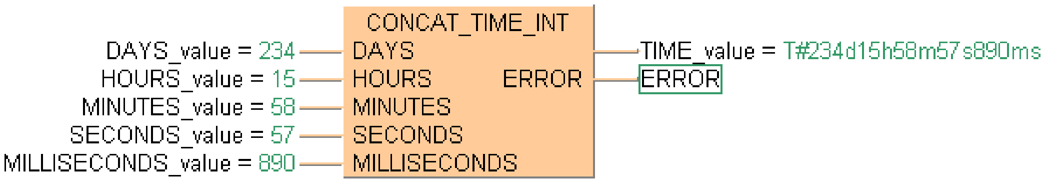
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	TIME_value	TIME	T#0s
1	VAR	DAYS_value	INT	234
2	VAR	HOURS_value	INT	15
3	VAR	MINUTES_value	INT	58
4	VAR	SECONDS_value	INT	57
5	VAR	MILLISECONDS_value	INT	890
6	VAR	ERROR	BOOL	FALSE

LD



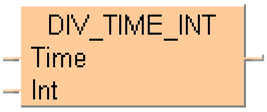
ST When programming with structured text, enter the following:

```
TIME_value := CONCAT_TIME_INT(DAYS := DAYS_value,
          HOURS := HOURS_value,
          MINUTES := MINUTES_value,
          SECONDS := SECONDS_value,
          MILLISECONDS := MILLISECONDS_value,
          ERROR => ERROR);
```

DIV_TIME_INT

Divide TIME by INTEGER

Description DIV_TIME_INT divides the value of the first input variable by the value of the second input variable and writes the result into the output variable.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

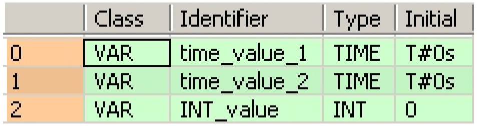
PLC types **Availability of DIV_TIME_INT (see page 1319)**

Data types

Data type	I/O	Function
TIME	1st input	dividend
INT	2nd input	divisor
TIME	output	result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

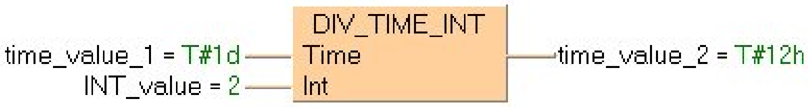


	Class	Identifier	Type	Initial
0	VAR	time_value_1	TIME	T#0s
1	VAR	time_value_2	TIME	T#0s
2	VAR	INT_value	INT	0

In this example the input variables (**time_value_1** and **INT_value**) have been declared. Instead, you may enter constants directly at the input contacts of a function.

Body **time_value_1** is divided by **INT_value**. The result is written into **time_value_2**.

LD



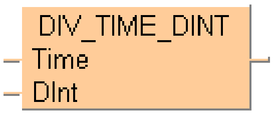
ST When programming with structured text, enter the following:

```
time_value_2:=DIV_TIME_INT(time_value_1, INT_value);
```

DIV_TIME_DINT

Divide TIME by DOUBLE INTEGER

Description DIV_TIME_DINT divides the value of the first input variable by the value of the second and writes the result into the output variable.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

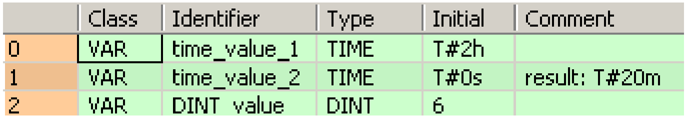
PLC types **Availability of** DIV_TIME_DINT **(see page 1319)**

Data types

Data type	I/O	Function
TIME	1st input	dividend
DINT	2nd input	divisor
TIME	output	result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header In the POU header, all input and output variables are declared that are used for programming this function.

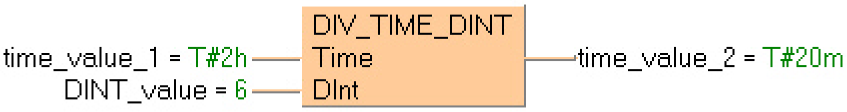


	Class	Identifier	Type	Initial	Comment
0	VAR	time_value_1	TIME	T#2h	
1	VAR	time_value_2	TIME	T#0s	result: T#20m
2	VAR	DINT_value	DINT	6	

In this example, the input variables (**time_value_1**, **DINT_value**) have been declared. However, you can write a constant directly at the input contact of the function instead.

Body **time_value_1** is divided by **DINT_value**. The result is written in **time_value_2**.

LD



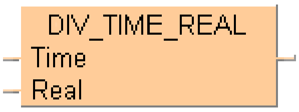
ST When programming with structured text, enter the following:

```
time_value_2:=DIV_TIME_DINT(time_value_1, INT_value);
```

DIV_TIME_REAL

Divide TIME by REAL

Description DIV_TIME_REAL divides the value of the first input variable of the data type TIME by the value of the second input variable of the data type REAL. The REAL value is rounded off to the nearest whole number. The result is written into the output variable.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

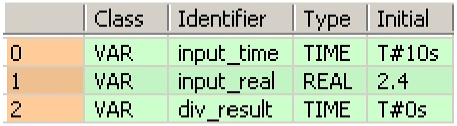
PLC types **Availability of** DIV_TIME_REAL **(see page** 1319**)**

Data types

Data type	I/O	Function
TIME	1st input	dividend
REAL	2nd input	divisor
TIME	output	result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

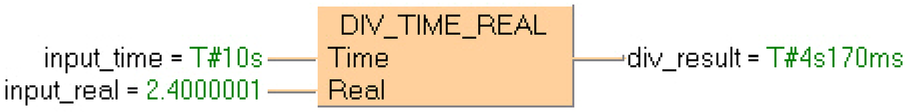
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	input_time	TIME	T#10s
1	VAR	input_real	REAL	2.4
2	VAR	div_result	TIME	T#0s

Body The value of variable **input_time** is divided by the value of the variable **input_real**. The result is written in **div_result**. In this example the input variables have been declared in the POU header. However, you may enter constants directly at the contact pins of the function.

LD



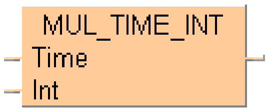
ST When programming with structured text, enter the following:

```
div_result:=DIV_TIME_REAL(input_time, input_real);
```

MUL_TIME_INT

Multiply TIME by INTEGER

Description **MUL_TIME_INT** multiplies the values of the two input variables with each other and writes the result into the output variable.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

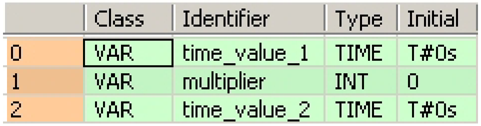
PLC types **Availability of** MUL_TIME_INT **(see page 1328)**

Data types

Data type	I/O	Function
TIME	1st input	multiplicand
INT	2nd input	multiplicator
TIME	output	result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

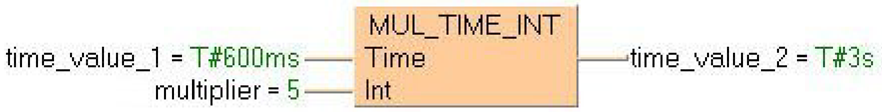


	Class	Identifier	Type	Initial
0	VAR	time_value_1	TIME	T#0s
1	VAR	multiplier	INT	0
2	VAR	time_value_2	TIME	T#0s

In this example the input variables (**time_value_1** and **multiplier**) have been declared. Instead, you may enter constants directly at the input contacts of a function.

Body **Time_value_1** is multiplied with **multipli**er. The result is written into **time_value_2**.

LD



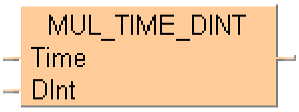
ST When programming with structured text, enter the following:

```
time_value_2:=MUL_TIME_INT(time_value_1, multiplier);
```

MUL_TIME_DINT

Multiply TIME by DOUBLE INTEGER

Description **MUL_TIME_DINT** multiplies the values of the input variables and writes the result to the output variable.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

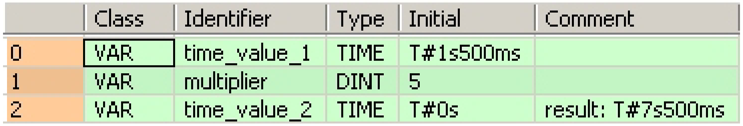
PLC types **Availability of** MUL_TIME_DINT **(see page 1328)**

Data types

Data type	I/O	Function
TIME	1st input	multiplicand
DINT	2nd input	multiplicator
TIME	output	result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header In the POU header, all input and output variables are declared that are used for programming this function.

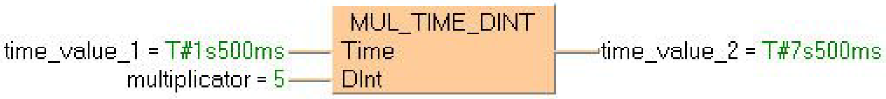


	Class	Identifier	Type	Initial	Comment
0	VAR	time_value_1	TIME	T#1s500ms	
1	VAR	multiplier	DINT	5	
2	VAR	time_value_2	TIME	T#0s	result: T#7s500ms

In this example, the input variables **time_value** and **multiplier** have been declared. However, you can write a constant directly at the input contact of the function instead.

Body **time_value_1** is multiplied by **multiplier**. The result is written in **time_value**_2.

LD



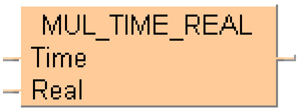
ST When programming with structured text, enter the following:

```
time_value_2:=MUL_TIME_DINT(time_value_1, multiplier);
```

MUL_TIME_REAL

Multiply TIME by REAL

Description **MUL_TIME_REAL** multiplies the value of the first input variable of the data type **TIME** by the value of the second input variable of the data type REAL. The **REAL** value is rounded off to the nearest whole number. The result is written into the output variable.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

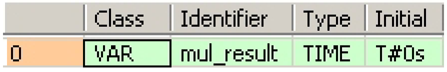
PLC types **Availability of** MUL_TIME_REAL **(see page 1328)**

Data types

Data type	I/O	Function
TIME	1st input	multiplicand
REAL	2nd input	multiplicator
TIME	output	result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

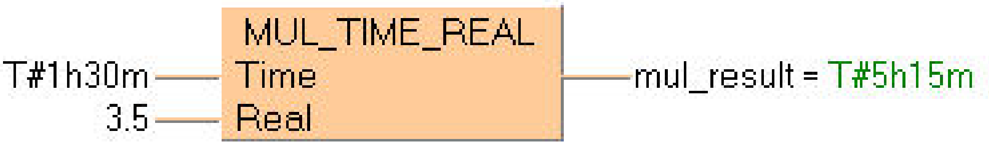
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	mul_result	TIME	T#0s

Body The constant **T#1h30m** is multiplied by the value **3.5**. The result is written in **mul_result**. By clicking on the (Monitoring) icon while in the online mode, you can see the result **T#5h15m0s0.00ms** immediately.

LD



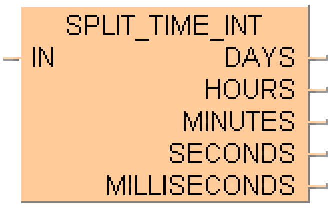
ST When programming with structured text, enter the following:

```
mul_result:=MUL_TIME_REAL(T#1h30m, 3.5);
```

SPLIT_TIME_INT

Split a time into INTEGER values

Description The highest non-zero time unit may be greater than its apparent limit, e.g. T#25h is a valid time value whereas T#1d25h is not.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

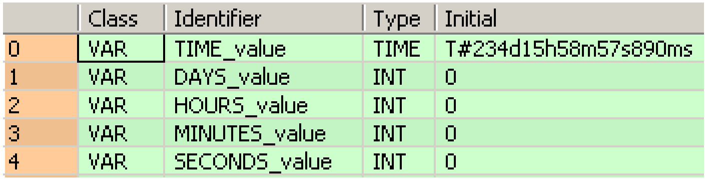
PLC types **Availability of SPLIT_TIME_INT (see page 1331)**

Data types

Data type	I/O	Function
TIME	input	time
INT	1st output 2nd output 3rd output 4th output 5th output	days hours minutes seconds milliseconds

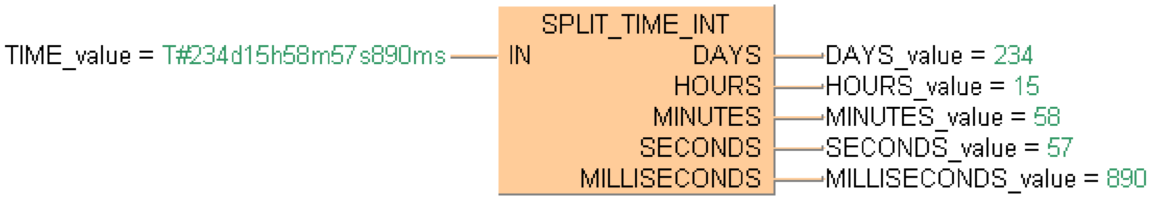
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	TIME_value	TIME	T#234d15h58m57s890ms
1	VAR	DAYS_value	INT	0
2	VAR	HOURS_value	INT	0
3	VAR	MINUTES_value	INT	0
4	VAR	SECONDS_value	INT	0

LD



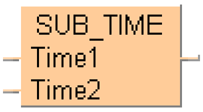
ST When programming with structured text, enter the following:

```
SPLIT_TIME_INT(IN := TIME_value,
                 DAYS => DAYS_value,
                 HOURS => HOURS_value,
                 MINUTES => MINUTES_value,
                 SECONDS => SECONDS_value,
                 MILLISECONDS => MILLISECONDS_value);
```

SUB_TIME

Subtract TIME

Description SUB_TIME subtracts the value of the second input variable from the value of the first input variable and writes the result into the output variable.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

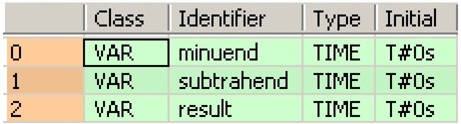
PLC types **Availability of** SUB_TIME **(see page 1331)**

Data types

Data type	I/O	Function
TIME	1st input	minuend
TIME	2nd input	subtrahend
TIME	output	result

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

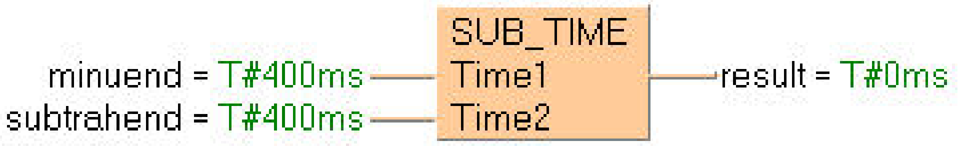


	Class	Identifier	Type	Initial
0	VAR	minuend	TIME	T#0s
1	VAR	subtrahend	TIME	T#0s
2	VAR	result	TIME	T#0s

In this example the input variables (**minuend** and **subtrahend**) have been declared. Instead, you may enter constants directly at the input contacts of a function.

Body **Subtrahend** is subtracted from **minuend**. The result will be written into **result.**

LD



ST When programming with structured text, enter the following:

```
result:= SUB_TIME(minuend, subtrahend);
```

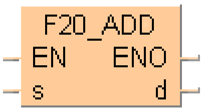
Chapter 15

Arithmetic instructions

F20_ADD

16-bit addition

Description The 16-bit equivalent constant or 16-bit area specified by **s** and the 16-bit area specified by **d** are added together if the trigger **EN** is in the ON-state. The result is stored in **d**. All 16-bit values are treated as integer values.



Example value 27

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0000	0000	0001	1011

+

Example value 16

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0000	0000	0001	0000

⬇

Result value 43 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0000	0000	0010	1011

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction ADD (see page 61). Please refer also to Advantages of the IEC instructions in the online help.

☞ **When this instruction is used, the area for the augend d is overwritten by the added result. If you want to avoid the overwrite, we recommend using the instruction F22_ADD2 (see page 345).**

PLC types see see page 1322

Data types

Variable	Data type	Function
s	ANY16	addend
d		augend and result

The variables **s** and **d** have to be of the same data type.

Operands

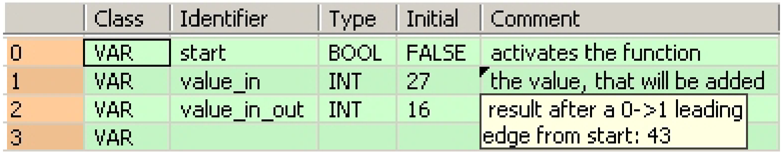
For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 16-bit data (overflow or underflow).

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

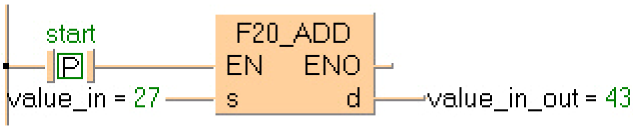
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_in	INT	27	the value, that will be added
2	VAR	value_in_out	INT	16	result after a 0->1 leading edge from start: 43
3	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



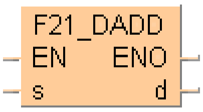
ST

```
IF start THEN
    F20_ADD(value_in, value_in_out);
END_IF;
```

F21_DADD

32-bit addition

Description The 32-bit equivalent constant or 32-bit area specified by **s** and the 32-bit data specified by **d** are added together if the trigger **EN** is in the ON-state. The result is stored in **d**. All 32-bit values are treated as double integer values.



Example value 1312896

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	0 1 0 0	0 0 0 0	1 0 0 0	1 0 0 0	0 0 0 0

32-bit area

+

Example value 558144

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0	1 0 0 0	0 1 0 0	0 1 0 0	0 0 0 0

⬇

Result value 1871040 if trigger is on

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	1 1 0 0	1 0 0 0	1 1 0 0	1 1 0 0	0 0 0 0

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction ADD (see page 61). Please refer also to Advantages of the IEC instructions in the online help.

☞ **When this instruction is used, the area for the augend d is overwritten by the added result. If you want to avoid the overwrite, we recommend using the instruction F23_DADD2 (see page 347).**

PLC types **Availability of F21_DADD (see page 1323)**

Data types

Variable	Data type	Function
s	ANY32	addend
d		augend and result

The variables **s** and **d** have to be of the same data type.

Operands

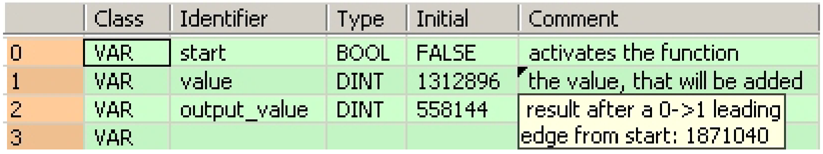
For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 32-bit data (overflow or underflow).

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

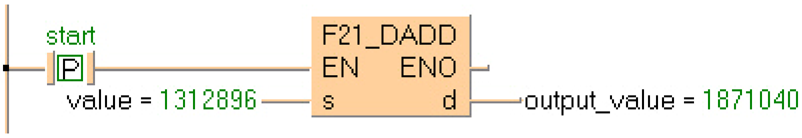
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value	DINT	1312896	the value, that will be added
2	VAR	output_value	DINT	558144	result after a 0->1 leading edge from start: 1871040
3	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD

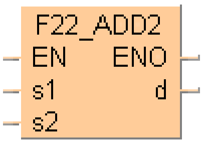


ST
```
IF start THEN
    F21_DADD(value, output_value);
END_IF;
```

F22_ADD2

16-bit addition, destination can be specified

Description The 16-bit data or 16-bit equivalent constant specified by **s1** and **s2** are added together if the trigger **EN** is in the ON-state. The result is stored in **d**. All 16-bit values are treated as integer values.



Example value 27

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0000	0000	0001	1011

+

Example value 16

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0000	0000	0001	0000

⬇

Result value 43 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0000	0000	0010	1011

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction ADD (see page 61). Please refer also to Advantages of the IEC instructions in the online help.

PLC types **Availability of** F22_ADD2 **(see page 1323)**

Data types

Variable	Data type	Function
s1	ANY16	augend
s2		addend
d		result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

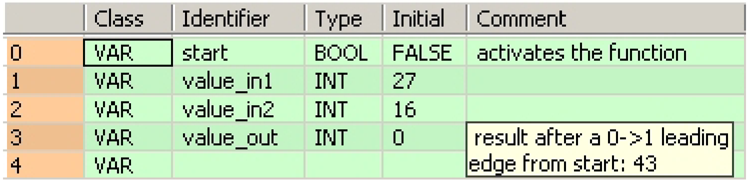
No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 16-bit data (overflow or underflow).

Example

In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header

All input and output variables used for programming this function have been declared in the POU header.

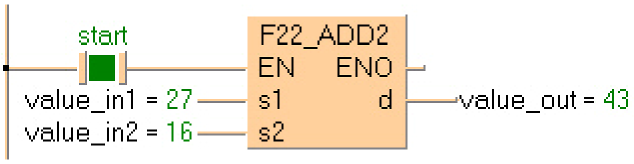


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_in1	INT	27	
2	VAR	value_in2	INT	16	
3	VAR	value_out	INT	0	result after a 0->1 leading edge from start: 43
4	VAR				

Body

When the variable **start** is set to TRUE, the function is carried out.

LD



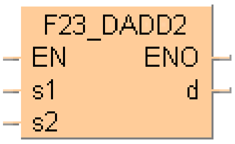
ST

```
IF start THEN
    F22_ADD2(value_in1, value_in2, value_out);
END_IF;
```

F23_DADD2

32-bit addition, destination can be specified

Description The 32-bit data or 32-bit equivalent constant specified by **s1** and **s2** are added together if the trigger **EN** is in the ON-state. The added result is stored in **d**. All 32-bit values are treated as double integer values.



Example value 1312896

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0000	0000	0001	0100	0000	1000	1000	0000

32-bit area

+

Example value 558144

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0000	0000	0000	1000	1000	0100	0100	0000

↓

Result value 1871040 if trigger is on

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0000	0000	0001	1100	1000	1100	1100	0000

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction ADD (see page 61). Please refer also to Advantages of the IEC instructions in the online help.

PLC types **Availability of** F23_DADD2 **(see page 1323)**

Data types

Variable	Data type	Function
s1	ANY32	augend
s2		addend
d		result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

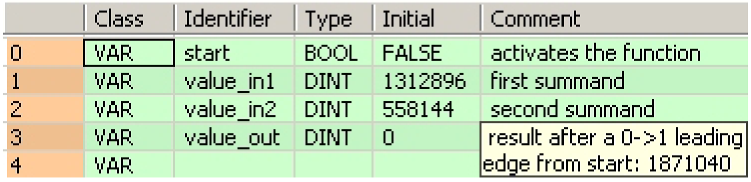
No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 32-bit data (overflow or underflow).

Example

In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

All input and output variables used for programming this function have been declared in the POU header.

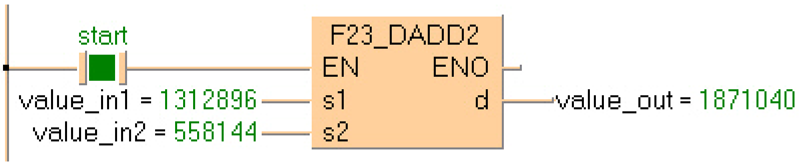


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_in1	DINT	1312896	first summand
2	VAR	value_in2	DINT	558144	second summand
3	VAR	value_out	DINT	0	result after a 0->1 leading edge from start: 1871040
4	VAR				

Body

When the variable **start** is set to TRUE, the function is carried out.

LD



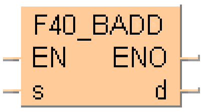
ST

```
IF start THEN
    F23_DADD2(value_in1, value_in2, value_out);
END_IF;
```

F40_BADD

4-digit BCD addition

Description The 4-digit BCD equivalent constant or 16-bit area for 4-digit BCD data specified by **s** and the 16-bit area for 4-digit BCD data specified by **d** are added together if the trigger **EN** is in the ON-state. The result is stored in **d**.



Example value 16#2111 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 1 0	0 0 0 1	0 0 0 1	0 0 0 1
16# (BCD)	2	1	1	1

+

Example value 16#0011 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1
16# (BCD)	0	0	1	1

↓

Result value 16#2122 (BCD) if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 1 0	0 0 0 1	0 0 1 0	0 0 1 0
16# (BCD)	2	1	2	2

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

☞ **When this instruction is used, the area for the augend d is overwritten by the added result. If you want to avoid the overwrite, we recommend using the instruction F41_DBADD (see page 351).**

PLC types **Availability of** F40_BADD **(see page 1325)**

Data types

Variable	Data type	Function
s	WORD	addend, 16-bit area for 4-digit BCD data or equivalent constant
d	WORD	augend and result, 16-bit area for 4-digit BCD data

Operands

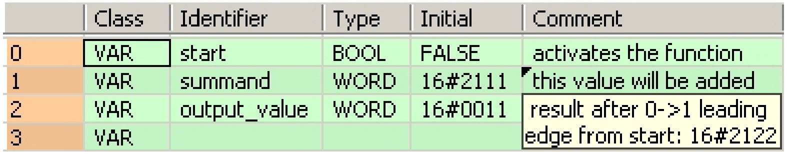
For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 4-digit BCD data (overflow).

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

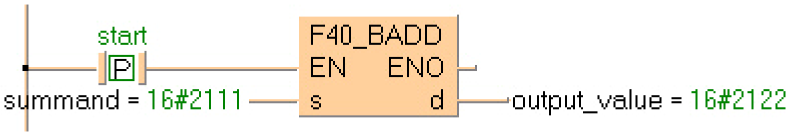
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	summand	WORD	16#2111	this value will be added
2	VAR	output_value	WORD	16#0011	result after 0->1 leading edge from start: 16#2122
3	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD

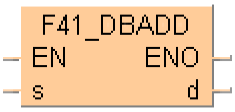


ST

```
IF start THEN
    F40_BADD(summand, output_value);
END_IF;
```

F41_DBADD 8-digit BCD addition

Description The 8-digit BCD equivalent constant or 8-digit BCD data specified by **s** and the 8-digit BCD data specified by **d** are added together if the trigger **EN** is in the ON-state. The result is stored in **d**.



Example value 16#12342000 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 0 1 0	0 0 0 0	0 0 0 0	0 0 0 0
16# BCD	1	2	3	4	2	0	0	0

32-bit area

Example value 16#00003678 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1	0 1 1 0	0 1 1 1	1 0 0 0
16# BCD	0	0	0	0	3	6	7	8

Result value 16#12345678 (BCD) if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0
16# BCD	1	2	3	4	5	6	7	8

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

When this instruction is used, the area for the augend d is overwritten by the added result. If you want to avoid the overwrite, we recommend using the instruction F43_DBADD2 (see page 355).

PLC types **Availability of** F41_DBADD **(see page 1325)**

Data types

Variable	Data type	Function
s	DWORD	addend, 32-bit area for 8-digit BCD data or equivalent constant
d	DWORD	augend and result, 32-bit area for 8-digit BCD data

Operands

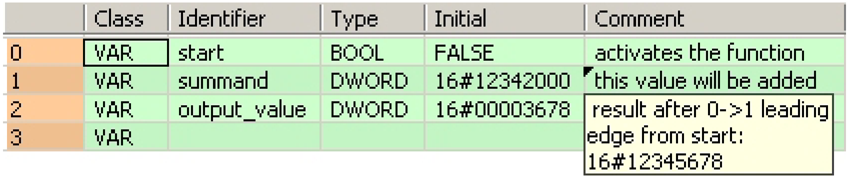
For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 8-digit BCD data (overflow).

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

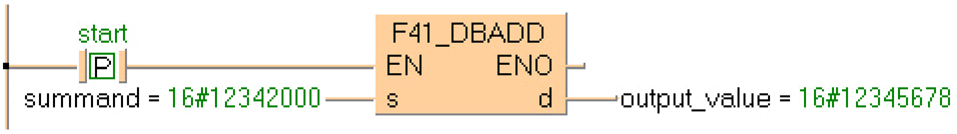
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	summand	DWORD	16#12342000	this value will be added
2	VAR	output_value	DWORD	16#00003678	result after 0->1 leading edge from start: 16#12345678
3	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD



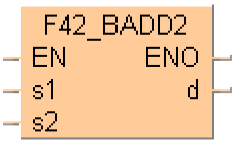
ST

```
IF DF(start) THEN
    F41_DBADD(summand, output_value);
END_IF;
```

F42_BADD2

4-digit BCD addition, destination can be specified

Description The 4-digit BCD equivalent constant or 16-bit area for 4-digit BCD data specified by **s1** and **s2** are added together if the trigger **EN** is in the ON-state. The result is stored in **d**.



Example value 16#4321 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0 1 0 0	0 0 1 1	0 0 1 0	0 0 0 1
16# (BCD)	4	3	2	1

+

Example value 16#1234 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0
16# (BCD)	1	2	3	4

⬇

Result value 16#5555 (BCD) if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 1 0 1	0 1 0 1	0 1 0 1	0 1 0 1
16# (BCD)	5	5	5	5

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F42_BADD2 **(see page 1325)**

Data types

Variable	Data type	Function
s1	WORD	augend, 16-bit area for 4-digit BCD data or equivalent constant
s2	WORD	addend, 16-bit area for 4-digit BCD data or equivalent constant
d	WORD	sum, 16-bit area for 4-digit BCD data

Operands

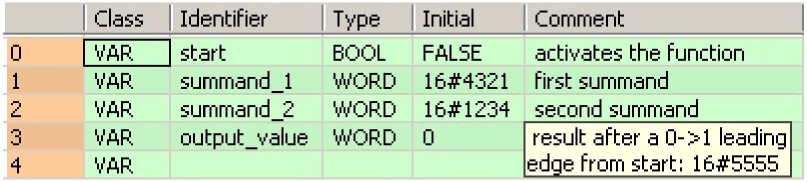
For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 4-digit BCD data (overflow).

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

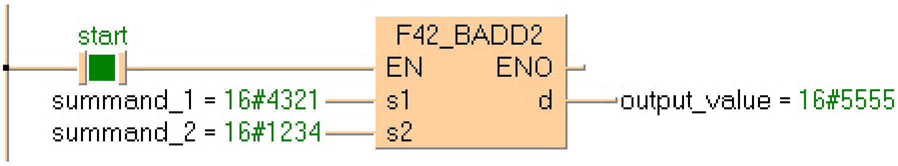
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	summand_1	WORD	16#4321	first summand
2	VAR	summand_2	WORD	16#1234	second summand
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 16#5555
4	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD

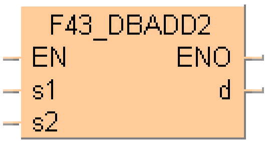


ST
```
IF start THEN
    F42_BADD2(summand_1, summand_2, output_value);
END_IF;
```

F43_DBADD2

8-digit BCD addition, destination can be specified

Description The 8-digit BCD equivalent constant or 32-bit area for 8-digit BCD data specified by **s1** and **s2** are added together if the trigger EN is in the ON-state. The result is stored in **d**.



Example value 16#12345678 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0
16# BCD	1	2	3	4	5	6	7	8

← 32-bit area →

+

Example value 16#87654321 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	1 0 0 0	0 1 1 1	0 1 1 0	0 1 0 1	0 1 0 0	0 0 1 1	0 0 1 0	0 0 0 1
16# BCD	8	7	6	5	4	3	2	1

⬇

Result value 16#99999999 (BCD) if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1
16# BCD	9	9	9	9	9	9	9	9

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F43_DBADD2 **(see page 1325)**

Data types

Variable	Data type	Function
s1	DWORD	augend, 32-bit area for 8-digit BCD data or equivalent constant
s2	DWORD	addend, 32-bit area for 8-digit BCD data or equivalent constant
d	DWORD	sum, 32-bit area for 8-digit BCD data

Operands

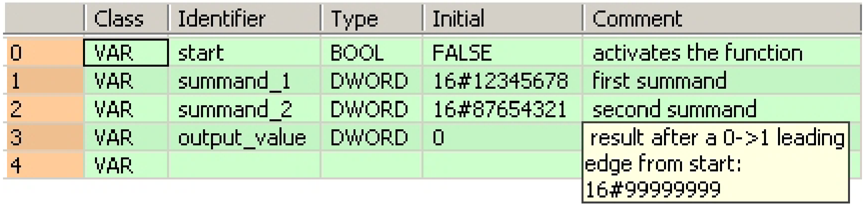
For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 8-digit BCD data (overflow).

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

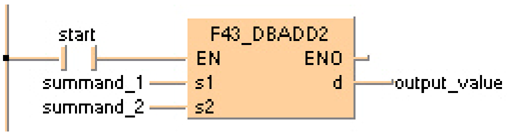
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	summand_1	DWORD	16#12345678	first summand
2	VAR	summand_2	DWORD	16#87654321	second summand
3	VAR	output_value	DWORD	0	result after a 0->1 leading edge from start: 16#99999999
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



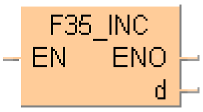
ST

```
IF start THEN
    F43_DBADD2( summand_1, summand_2, output_value);
END_IF;
```

F35_INC

16-bit increment

Description Adds "1" to the 16-bit data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.



Example value 17

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1

Result value 18 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	0 0 1 0

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F35_INC **(see page 1325)**

Data types

Variable	Data type	Function
d	ANY16	16-bit area to be increased by 1

Operands

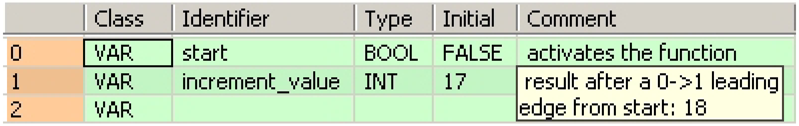
For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 16-bit data (overflow).

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

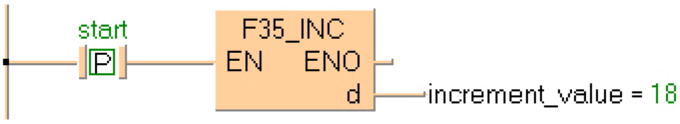
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	increment_value	INT	17	result after a 0->1 leading edge from start: 18
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD

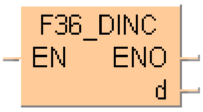


```
ST IF DF(start) THEN
       F35_INC(increment_value);
   END_IF;
```

F36_DINC

32-bit increment

Description Adds "1" to the 32-bit data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.



Example value 131081

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0000	0000	0000	0010	0000	0000	0000	1001

32-bit area

Result value 131082 if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0000	0000	0000	0010	1000	0000	0000	1010

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F36_DINC **(see page 1325)**

Data types

Variable	Data type	Function
d	ANY32	32-bit area to be increased by 1

Operands

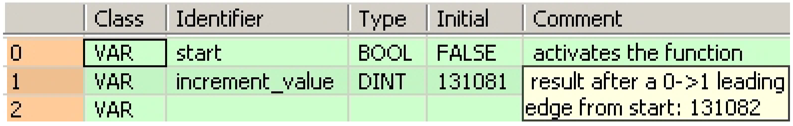
For	Relay				T/C		Register			Constant
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 32-bit data (overflow).

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

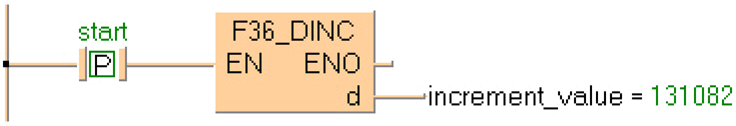
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	increment_value	DINT	131081	result after a 0->1 leading edge from start: 131082
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD



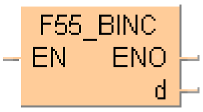
ST

```
IF DF(start) THEN
    F36_DINC(increment_value);
END_IF;
```

F55_BINC

4-digit BCD increment

Description Adds "1" to the 4-digit BCD data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.



Example value 16#4320 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 1 0 0	0 0 1 1	0 0 1 0	0 0 0 0
16# BCD	4	3	2	0

Result value 16#4321 (BCD) if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 1 0 0	0 0 1 1	0 0 1 0	0 0 0 1
16# BCD	4	3	2	1

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F55_BINC **(see page 1325)**

Operands

Variable	Data type	Function
d	WORD	16-bit area for 4-digit BCD data to be increased by 1

Operands

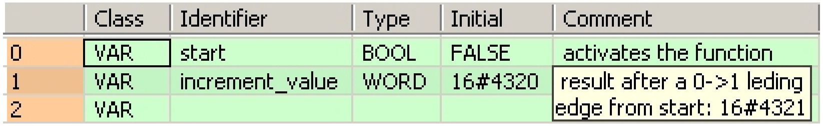
For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 4-digit BCD data (overflow).

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

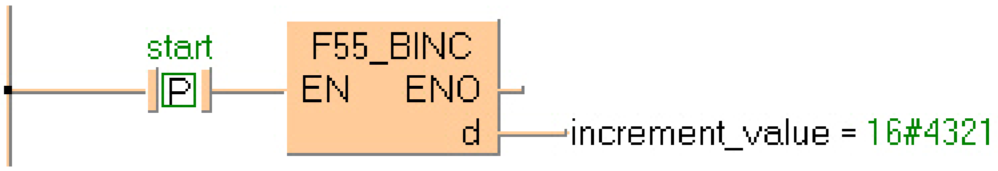
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	increment_value	WORD	16#4320	result after a 0->1 leding edge from start: 16#4321
2	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD

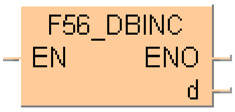


ST
```
IF DF(start) THEN
    F55_BINC(increment_value);
END_IF;
```

F56_DBINC

8-digit BCD increment

Description Adds "1" to the 8-digit BCD data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.



Example value 16#87654320 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	1 0 0 0	0 1 1 1	0 1 1 0	0 1 0 1	0 1 0 0	0 0 1 1	0 0 1 0	0 0 0 0
16# BCD	8	7	6	5	4	3	2	0

32-bit area

Result value 16#87654321 (BCD) if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	1 0 0 0	0 1 1 1	0 1 1 0	0 1 0 1	0 1 0 0	0 0 1 1	0 0 1 0	0 0 0 1
16# BCD	8	7	6	5	4	3	2	1

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F56_DBINC **(see page 1325)**

Data types

Variable	Data type	Function
d	DWORD	32-bit area for 8-digit BCD data to be increased by 1

Operands

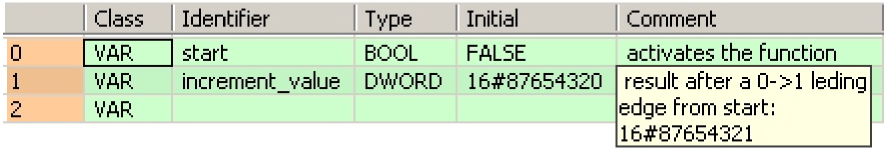
For	Relay				T/C		Register			Constant
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 8-digit BCD data (overflow).

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

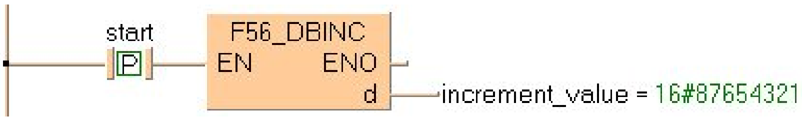
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	increment_value	DWORD	16#87654320	result after a 0->1 leding edge from start: 16#87654321
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD

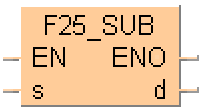


ST
```
IF DF(start) THEN
    F56_DBINC(increment_value);
END_IF;
```

F25_SUB

16-bit subtraction

Description Subtracts the 16-bit equivalent constant or 16-bit area specified by **s** from the 16-bit area specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d** (minuend area). All 16-bit values are treated as integer values.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction **SUB** (see page 62). Please refer also to Advantages of the IEC instructions in the online help.

Example value 16

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	1 0 1 1

−

Example value 27

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0

⬇

Result value -11 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	1 1 1 1	1 1 1 1	1 1 1 1	0 1 0 1

PLC types **Availability of** F25_SUB **(see page** 1323**)**

Data types

Variable	Data type	Function
s	ANY16	subtrahend
d		minuend and result

The variables **s** and **d** have to be of the same data type.

Operands

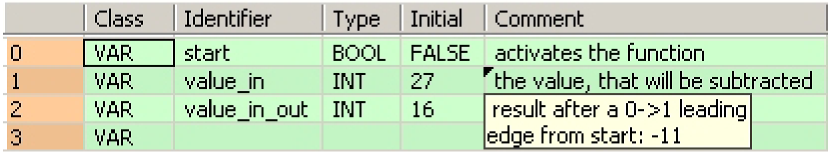
For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 16-bit data (overflow or underflow).

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

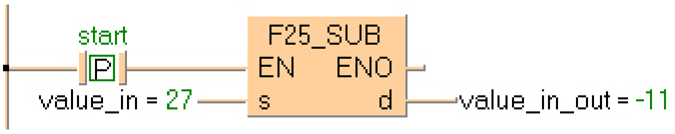
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_in	INT	27	the value, that will be subtracted
2	VAR	value_in_out	INT	16	result after a 0->1 leading edge from start: -11
3	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD

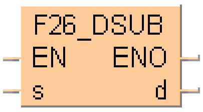


ST
```
IF start THEN
    F25_SUB(value_in, value_in_out);
END_IF;
```

F26_DSUB

32-bit subtraction

Description Subtracts the 32-bit equivalent constant or 32-bit data specified by **s** from the 32-bit data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d** (minuend area). All 32-bit values are treated as double integer values.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction SUB (see page 62). Please refer also to Advantages of the IEC instructions in the online help.

Example value 16778109

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0000	0001	0000	0000	0000	0011	0111	1101
	◄			32-bit area				►

−

Example value 524740

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0000	0000	0000	1000	0000	0001	1100	0100

⬇

Result value 16253369 if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0000	0000	1111	1000	0000	0001	1011	1001

PLC types **Availability of** F26_DSUB **(see page 1323)**

Data types

Variable	Data type	Function
s	ANY32	subtrahend
d		minuend and result

The variables **s** and **d** have to be of the same data type.

Operands

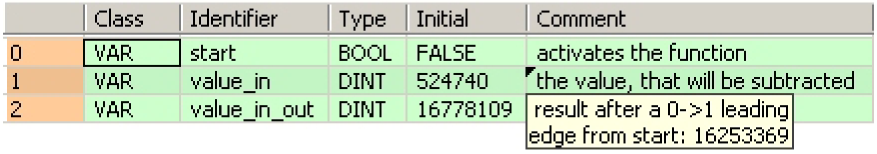
For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 32-bit data (overflow or underflow).

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

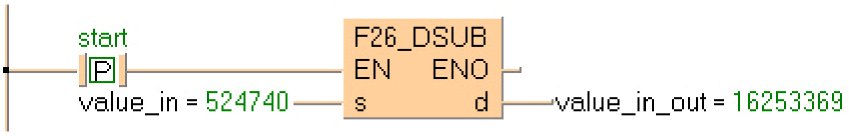
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_in	DINT	524740	the value, that will be subtracted
2	VAR	value_in_out	DINT	16778109	result after a 0->1 leading edge from start: 16253369

Body When the variable **start** is set to TRUE, the function is carried out.

LD

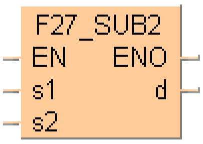


ST
```
IF start THEN
    F26_DSUB(value_in, value_in_out);
END_IF;
```

F27_SUB2

16-bit subtraction, destination can be specified

Description Subtracts the 16-bit data or 16-bit equivalent constant specified by **s2** from the 16-bit data or 16-bit equivalent constant specified by **s1** if the trigger **EN** is in the ON-state. The result is stored in **d**. All 16-bit values are treated as integer values.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction SUB (see page 62). Please refer also to Advantages of the IEC instructions in the online help.

Example value 27

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0

−

Example value 16

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	1 0 1 1

⬇

Result value 11 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	0 0 1 1

PLC types **Availability of** F27_SUB2 **(see page 1323)**

Data types

Variable	Data type	Function
s1	ANY16	minuend
s2		subtrahend
d		result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

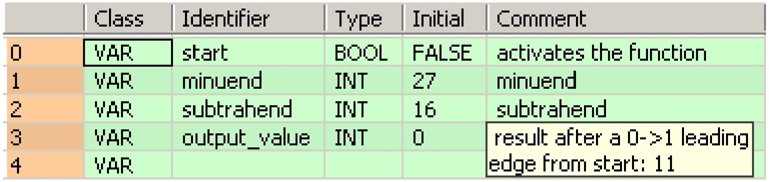
No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 16-bit data (overflow or underflow).

Example

In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

All input and output variables used for programming this function have been declared in the POU header.

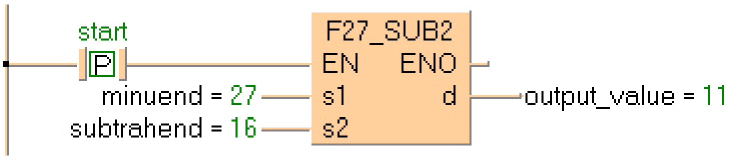


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	minuend	INT	27	minuend
2	VAR	subtrahend	INT	16	subtrahend
3	VAR	output_value	INT	0	result after a 0->1 leading edge from start: 11
4	VAR				

Body

When the variable **start** is set to TRUE, the function is carried out.

LD



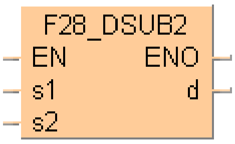
ST

```
IF start THEN
    F27_SUB2(minuend, subtrahend, output_value);
END_IF;
```

F28_DSUB2

32-bit subtraction, destination can be specified

Description Subtracts the 32-bit data or 32-bit equivalent constant specified by **s2** from the 32-bit data or 32-bit equivalent constant specified by **s1** if the trigger is in the ON-state. The result is stored in **d**. All 32-bit values are treated as double integer values.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction SUB (see page 62). Please refer also to Advantages of the IEC instructions in the online help.

Example value 16809984

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0000	0001	0000	0000	1000	0000	0000	0000

← 32-bit area →

−

Example value 525312

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	0000	0000	0000	1000	0000	0100	0000	0000

⬇

Result value 16284672 if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0000	0000	1111	1000	0111	1100	0000	0000

PLC types **Availability of** F28_DSUB2 **(see page 1323)**

Data types

Variable	Data type	Function
s1	ANY32	minuend
s2		subtrahend
d		result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

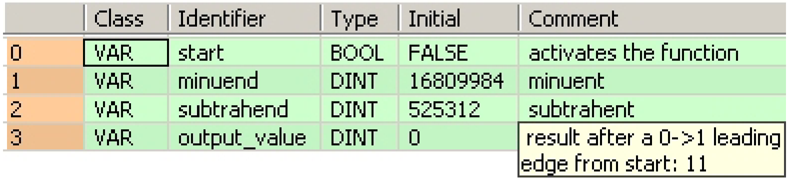
For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 32-bit data (overflow or underflow).

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

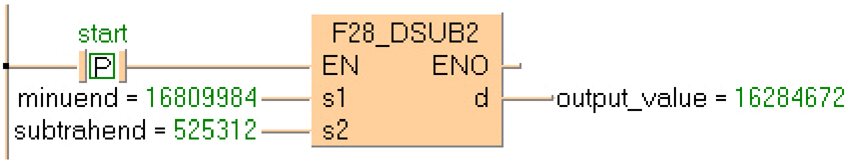
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	minuend	DINT	16809984	minuent
2	VAR	subtrahend	DINT	525312	subtrahent
3	VAR	output_value	DINT	0	result after a 0->1 leading edge from start: 11

Body When the variable **start** is set to TRUE, the function is carried out.

LD

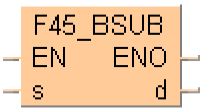


ST
```
IF start THEN
    F28_DSUB2(minuend, subtrahend, output_value);
END_IF;
```

F45_BSUB

4-digit BCD subtraction

Description Subtracts the 4-digit BCD equivalent constant or 16-bit area for 4-digit BCD data specified by **s** from the 16-bit area for 4-digit BCD data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.



Example value 16#2111 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 1 0	0 0 0 1	0 0 0 1	0 0 0 1
16# (BCD)	2	1	1	1

−

Example value 16#0011 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 1	0 1 0 1
16# (BCD)	0	0	1	1

⬇ **Trigger: ON**

Result value 16#2100 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 1 0	0 0 0 1	0 0 0 0	0 0 0 0
16# (BCD)	2	1	0	0

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F45_BSUB **(see page 1325)**

Data types

Variable	Data type	Function
s	WORD	subtrahend, 16-bit area for 4-digit BCD data or equivalent constant
d	WORD	minuend and result, 16-bit area for 4-digit BCD data

Operands

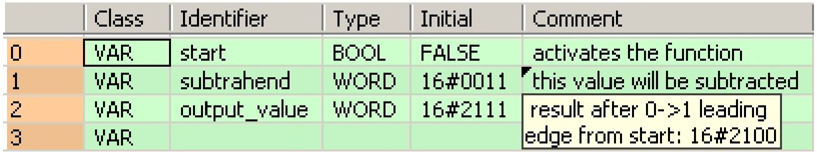
For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 4-digit BCD data (overflow).

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

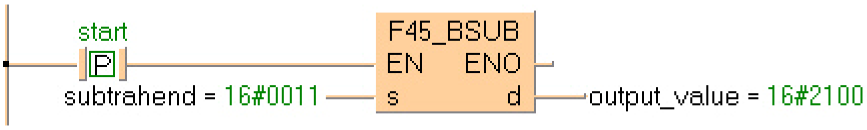
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	subtrahend	WORD	16#0011	this value will be subtracted
2	VAR	output_value	WORD	16#2111	result after 0->1 leading edge from start: 16#2100
3	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD



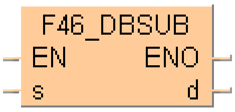
ST

```
IF DF(start) THEN
    F45_BSUB(subtrahend, output_value);
END_IF;
```

F46_DBSUB

8-digit BCD subtraction

Description Subtracts the 8-digit BCD equivalent constant or 8-digit BCD data specified by **s** from the 8-digit BCD data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.



Example value 16#23210044 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0010	0011	0001	0001	0000	0000	0100	0100
16# BCD	2	3	2	1	0	0	4	4

← 32-bit area →

−

Example value 16#00210011 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0000	0000	0010	0001	0000	0000	0001	0001
16# BCD	0	0	2	1	0	0	1	1

↓ **Trigger: ON**

Result value 16#23000033 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0010	0011	0000	0000	0000	0000	0011	0011
16# BCD	2	3	0	0	0	0	3	3

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F46_DBSUB **(see page 1325)**

Data types

Variable	Data type	Function
s	DWORD	subtrahend, 32-bit area for 8-digit BCD data or equivalent constant
d	DWORD	minuend and result, 32-bit area for 8-digit BCD data

Operands

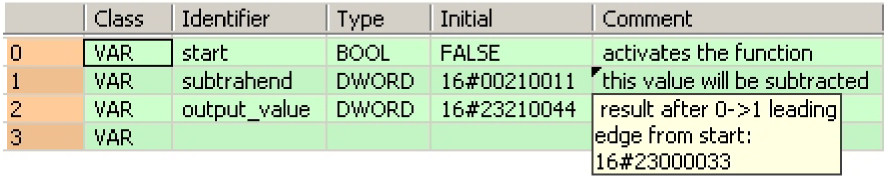
For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 8-digit BCD data (overflow).

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

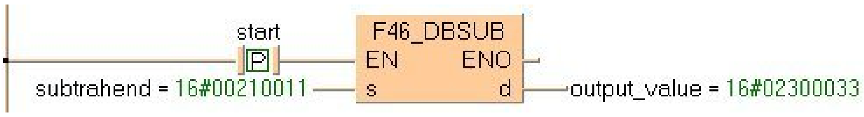
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	subtrahend	DWORD	16#00210011	this value will be subtracted
2	VAR	output_value	DWORD	16#23210044	result after 0->1 leading edge from start: 16#23000033
3	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD

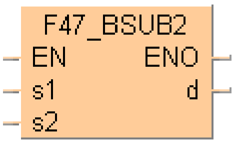


ST
```
IF DF(start) THEN
    F46_DBSUB(subtrahend, output_value);
END_IF;
```

F47_BSUB2

4-digit BCD subtraction, destination can be specified

Description Subtracts the 4-digit BCD equivalent constant or 16-bit area for 4-digit BCD data specified by **s2** from the 4-digit BCD equivalent constant or 16-bit area for 4-digit BCD data specified by **s1** if the trigger EN is in the ON-state. The result is stored in **d**.



Example value 16#16 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0 0 0 0	0 0 0 0	0 0 0 1	0 1 1 0
16# (BCD)	0	0	1	6

−

Example value 16#4 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0
16# (BCD)	0	0	0	4

↓ **Trigger: ON**

Result value 16#12 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	0 0 1 0
16# (BCD)	0	0	1	2

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F47_BSUB2 **(see page 1325)**

Data types

Variable	Data type	Function
s1	WORD	minuend, 16-bit area for 4-digit BCD data or equivalent constant
s2	WORD	subtrahend, 16-bit area for 4-digit BCD data or equivalent constant
d	WORD	result, 16-bit area for 4-digit BCD data

Operands

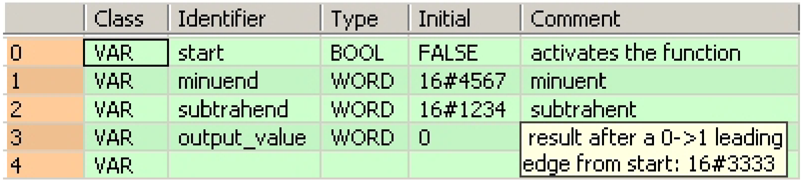
For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 4-digit BCD data (overflow).

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

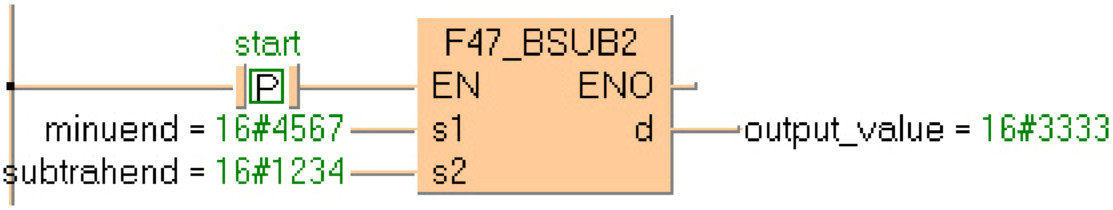
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	minuend	WORD	16#4567	minuent
2	VAR	subtrahend	WORD	16#1234	subtrahent
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 16#3333
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD

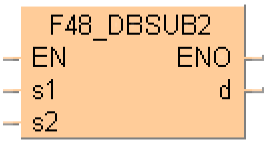


ST
```
IF start THEN
    F47_BSUB2(minuend, subtrahend, output_value);
END_IF;
```

F48_DBSUB2

8-digit BCD subtraction, destination can be specified

Description Subtracts the 8-digit BCD equivalent constant or 8-digit BCD data specified by **s2** from the 8-digit BCD equivalent constant or 8-digit BCD data specified by **s1** if the trigger **EN** is in the ON-state. The result is stored in **d**.



Example value 16#33555588 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0 0 0 1	0 0 1 0	0 1 0 1	0 1 0 1	0 1 0 1	0 1 0 1	1 0 0 0	1 0 0 0
16# BCD	3	3	5	5	5	5	8	8

← 32-bit area →

−

Example value 16#00110022 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1	0 0 0 0	0 0 0 0	0 0 1 0	0 0 1 0
16# BCD	0	0	1	1	0	0	2	2

⬇ **Trigger: ON**

Result value 16#33445566 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 1 1	0 0 1 1	0 1 0 0	0 1 0 0	0 1 0 1	0 1 0 1	0 1 1 0	0 1 1 0
16# BCD	3	3	4	4	5	5	6	6

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F48_DBSUB2 **(see page 1325)**

Data types

Variable	Data type	Function
s1	DWORD	minuend, 32-bit area for 8-digit BCD data or equivalent constant
s2	DWORD	subtrahend, 32-bit area for 8-digit BCD data or equivalent constant
d	DWORD	result, 32-bit area for 8-digit BCD data

Operands

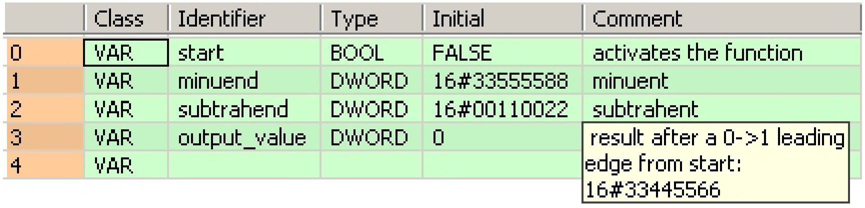
For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 8-digit BCD data (overflow).

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

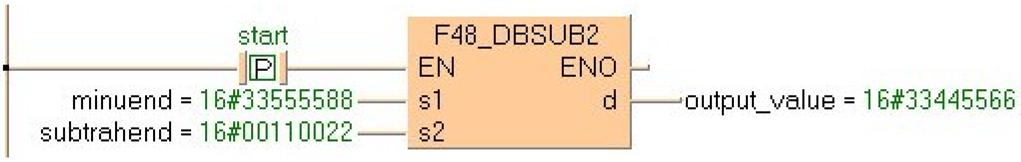
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	minuend	DWORD	16#33555588	minuent
2	VAR	subtrahend	DWORD	16#00110022	subtrahent
3	VAR	output_value	DWORD	0	result after a 0->1 leading edge from start: 16#33445566
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD

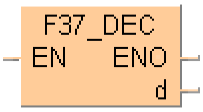


ST
```
IF start THEN
    F48_DBSUB2(minuend, subtrahend, output_value);
END_IF;
```

F37_DEC

16-bit decrement

Description Subtracts "1" from the 16-bit data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.



Example value 17

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1

Result value 16 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F37_DEC **(see page 1325)**

Data types

Variable	**Data type**	**Function**
d	INT, WORD	16-bit area to be decreased by 1

Operands

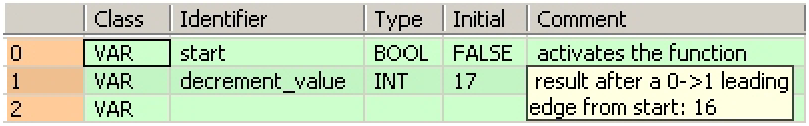
For	**Relay**				**T/C**		**Register**			**Constant**
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	**IEC address**	**Set**	**If**
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 16-bit data (underflow).

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

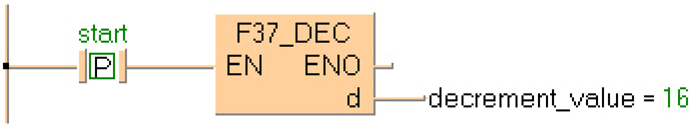
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	decrement_value	INT	17	result after a 0->1 leading edge from start: 16
2	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD

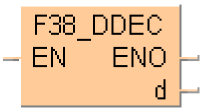


ST
```
IF DF(start) THEN
    F37_DEC(decrement_value);
END_IF;
```

F38_DDEC

32-bit decrement

Description Subtracts "1" to the 32-bit data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.



Example value 131081

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0	0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 1

32-bit area

Result 131080

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0	0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F38_DDEC **(see page 1325)**

Data types

Variable	Data type	Function
d	ANY32	32-bit area to be decreased by 1

Operands

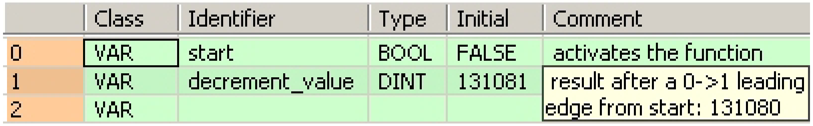
For	Relay				T/C		Register			Constant
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 32-bit data (underflow).

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

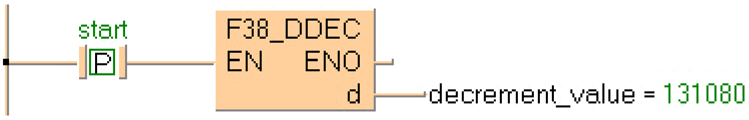
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	decrement_value	DINT	131081	result after a 0->1 leading edge from start: 131080
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD

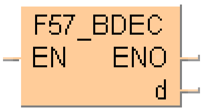


ST
```
IF DF(start) THEN
    F38_DDEC(decrement_value);
END_IF;
```

F57_BDEC

4-digit BCD decrement

Description Subtracts "1" from the 4-digit BCD data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.



Example value 4322 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 1 0 0	0 0 1 1	0 0 1 0	0 0 1 0
16# BCD	4	3	2	2

Trigger: ON

Result value 4321 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 1 0 0	0 0 1 1	0 0 1 0	0 0 0 1
16# BCD	4	3	2	1

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F57_BDEC **(see page 1325)**

Data types

Variable	Data type	Function
d	WORD	16-bit area for BCD data to be decreased by 1

Operands

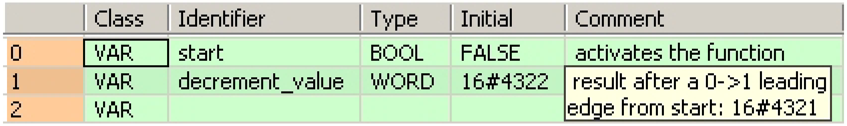
For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 4-digit BCD data (underflow).

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

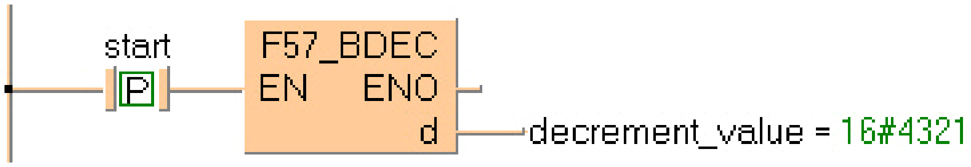
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	decrement_value	WORD	16#4322	result after a 0->1 leading edge from start: 16#4321
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

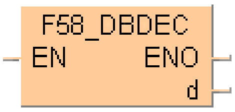
LD



ST
```
IF DF(start) THEN
    F57_BDEC(decrement_value);
END_IF;
```

F58_DBDEC 8-digit BCD decrement

Description Subtracts "1" from the 8-digit BCD data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.



Example value 87654322 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	1 0 0 0	0 1 1 1	0 1 1 0	0 1 0 1	0 1 0 0	0 0 1 1	0 0 1 0	0 0 1 0
16# BCD	8	7	6	5	4	3	2	2

32-bit area

Trigger: ON

Result value 87654321 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	1 0 0 0	0 1 1 1	0 1 1 0	0 1 0 1	0 1 0 0	0 0 1 1	0 0 1 0	0 0 0 1
16# BCD	8	7	6	5	4	3	2	1

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F58_DBDEC **(see page 1325)**

Data types

Variable	Data type	Function
d	DWORD	32-bit area for BCD data to be decreased by 1

Operands

For	Relay				T/C		Register			Constant
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

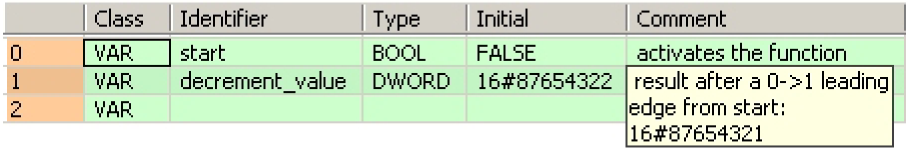
Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the result exceeds the range of 8-digit BCD data (underflow).

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header

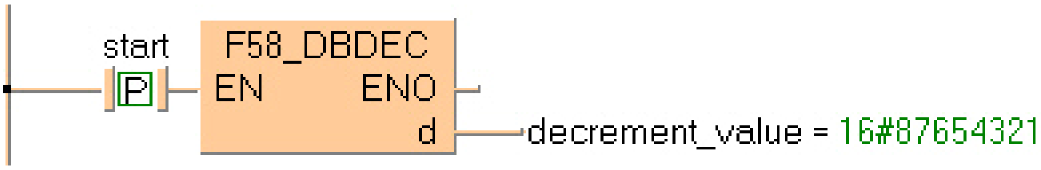
All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	decrement_value	DWORD	16#87654322	result after a 0->1 leading edge from start: 16#87654321
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD

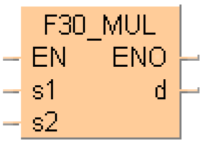


ST
```
IF DF(start) THEN
    F58_DBDEC(decrement_value);
END_IF;
```

F30_MUL

16-bit multiplication, destination can be specified

Description Multiplies the 16-bit data or 16-bit equivalent constant **s1** and the 16-bit data or 16-bit equivalent constant specified by **s2** if the trigger **EN** is in the ON-state. The result is stored in **d** (32-bit area). All 16-bit values are treated as integer values.



Example value 10

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0 0 0 0	0 0 0 0	0 0 0 0	1 0 1 0

X

Example value 17

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	1 0 0 0	0 1 0 0	0 0 0 1	0 0 0 1

Result value 170 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	1 0 1 0	1 0 1 0

← 32-bit area →

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction **MUL** (see page 63). Please refer also to Advantages of the IEC instructions in the online help.

PLC types **Availability of** F30_MUL **(see page 1323)**

Data types

Variable	Data type	Function
s1	ANY16	multiplicand
s2		multiplier
d	ANY32	result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

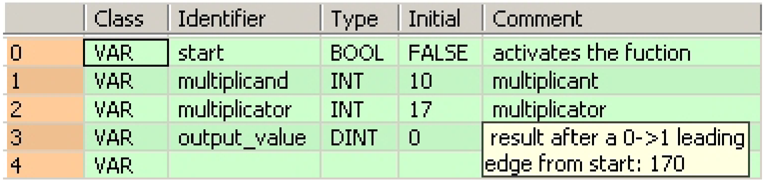
For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

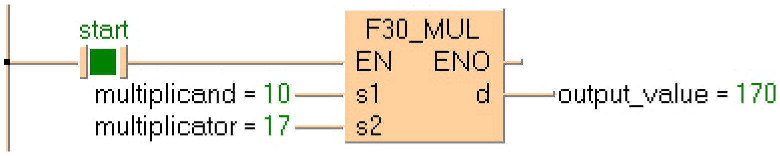
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the fuction
1	VAR	multiplicand	INT	10	multiplicant
2	VAR	multiplicator	INT	17	multiplicator
3	VAR	output_value	DINT	0	result after a 0->1 leading edge from start: 170
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD

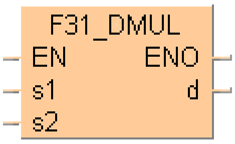


ST
```
IF start THEN
    F30_MUL(multiplicand, multiplicator, output_value);
END_IF;
```

F31_DMUL

32-bit multiplication, destination can be specified

Description Multiplies the 32-bit data or 32-bit equivalent constant specified by **s1** and the one specified by **s2** if the trigger **EN** is in the ON-state. The result is stored in **d[0]**, **d[1]** (64-bit area). All 32-bit values are treated as double integer values.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction MUL (see page 63). Please refer also to Advantages of the IEC instructions in the online help.

PLC types **Availability of** F31_DMUL **(see page 1324)**

Data types

Variable	Data type	Function
s1	ANY32	multiplicand
s2		multiplier
d	ARRAY **[0..1] of** ANY32	result

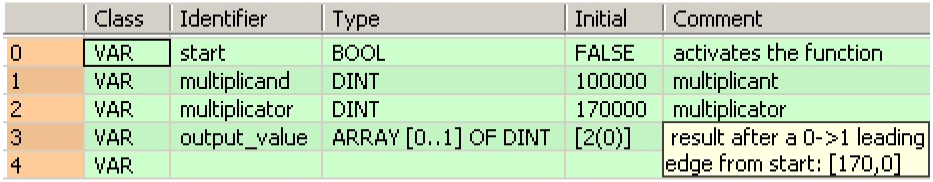
The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

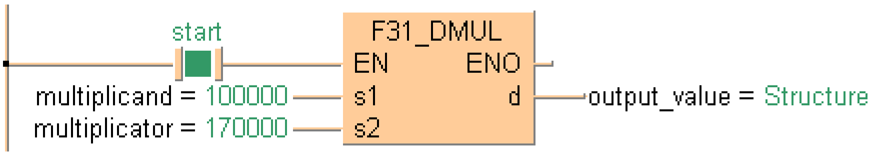
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	multiplicand	DINT	100000	multiplicant
2	VAR	multiplicator	DINT	170000	multiplicator
3	VAR	output_value	ARRAY [0..1] OF DINT	[2(0)]	result after a 0->1 leading edge from start: [170,0]
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



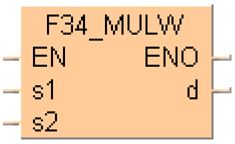
Access to the result is possible with output_value[0] and output_value[1].

```
ST IF start THEN
       F31_DMUL(multiplicand, multiplicator, output_value);
   END_IF;
```

F34_MULW

16-bit data multiply (result in 16 bits)

Description The function multiplies the value specified at input **s1** by the value specified at input **s2**. The result of the function is returned at output **d**. The result at output **d** lies between -32768 and 32767 (i.e. between 16#0 and 16#FFFF). All 16-bit values are treated as integer values.



Example value 6

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0 0 0 0	0 0 0 0	0 0 0 0	0 1 1 0

X

Example value 5

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 1

⬇

Result value 30 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 1	1 1 1 0

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F34_MULW **(see page 1324)**

Data types

Variable	Data type	Function
s1	ANY16	multiplicand
s2		multiplier
d		result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

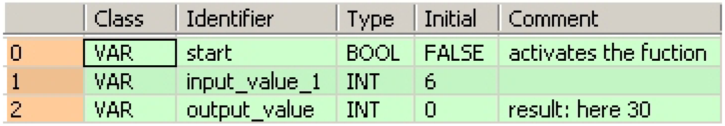
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the result calculated exceeds the 16-bit area specified at output **b**.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	for an instant	▪ the result calculated is 0.

Example

In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

All input and output variables used for programming this function have been declared in the POU header.

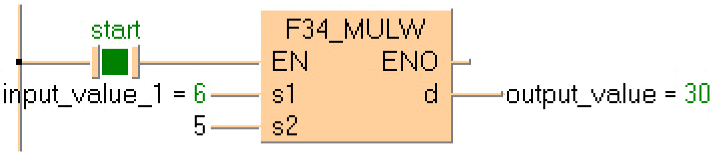


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the fuction
1	VAR	input_value_1	INT	6	
2	VAR	output_value	INT	0	result: here 30

Body

When the variable **start** is set to TRUE, the function is carried out.

LD

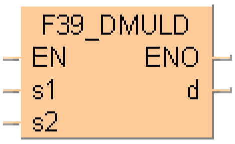


ST

```
IF start THEN
    F34_MULW(input_value_1, 5, output_value);
END_IF;
```

F39_DMULD 32-bit data multiply (result in 32 bits)

Description The function multiplies the value specified at input **s1** by the value specified at input **s2**. The result of the function is returned at output **d**. The result at output 'd' lies between -2147483648 and 2147483647 (i.e. between 16#0 and 16#FFFFFFFF). All 32-bit values are treated as double integer values.



Example value 17

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0000	0000	0001	0001

Result value 18 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0000	0000	0001	0010

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F39_DMULD **(see page 1325)**

Data types

Variable	Data type	Function
s1	ANY32	multiplicand
s2		multiplier
d		result

Operands

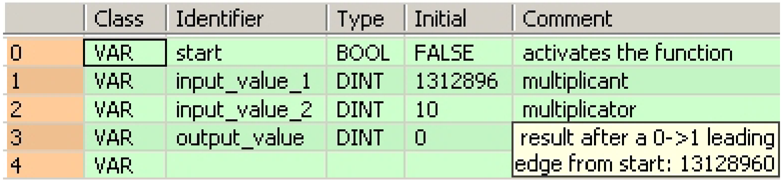
For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the result calculated exceeds the 32-bit area specified at output **d**.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	for an instant	▪ the result calculated is 0.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

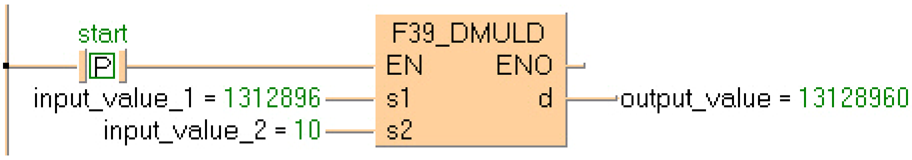


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value_1	DINT	1312896	multiplicant
2	VAR	input_value_2	DINT	10	multiplicator
3	VAR	output_value	DINT	0	result after a 0->1 leading edge from start: 13128960
4	VAR				

In this example the input variables **input_value_1** and **input_value _2** are declared. However, you can write constants directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD

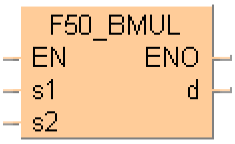


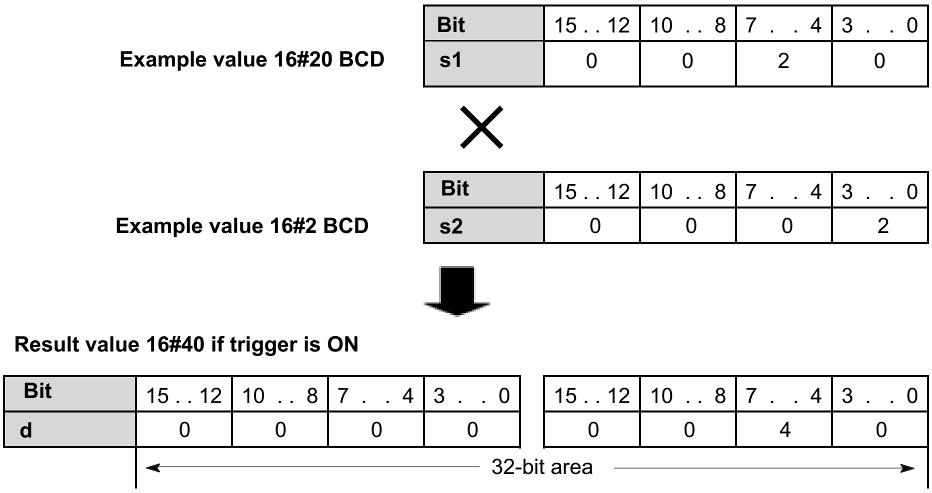
ST
```
IF start THEN
    F39_DMULD(input_value_1, input_value_2, output_value);
END_IF;
```

F50_BMUL

4-digit BCD multiplication, destination can be specified

Description Multiplies the 4-digit BCD equivalent constant or 16-bit area for 4-digit BCD data specified by **s1** and **s2** if the trigger **EN** is in the ON-state. The result is stored in **d** (8-digit area).





Example value 16#20 BCD

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0	0	2	0

×

Example value 16#2 BCD

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	0	0	0	2

⬇

Result value 16#40 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0	0	0	0	0	0	4	0

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F50_BMUL **(see page 1325)**

Data types

Variable	Data type	Function
s1	WORD	multiplicand, 16-bit area for 4-digit BCD data or equivalent constant
s2	WORD	multiplier, 16-bit area for 4-digit BCD data or equivalent constant
d	DWORD	result, 32-bit area for 8-digit BCD data

Operands

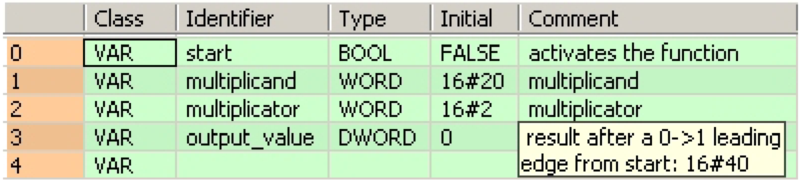
For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

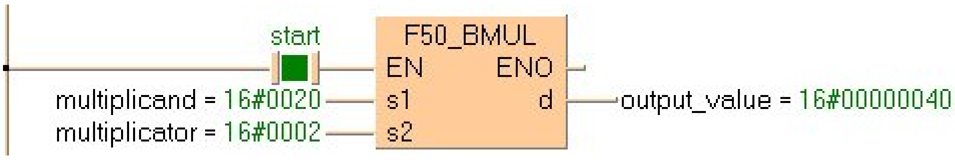
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	multiplicand	WORD	16#20	multiplicand
2	VAR	multiplicator	WORD	16#2	multiplicator
3	VAR	output_value	DWORD	0	result after a 0->1 leading edge from start: 16#40
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



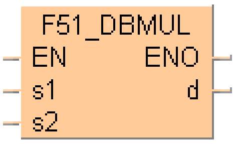
ST

```
IF start THEN
    F50_BMUL(multiplicand, multiplicator, output_value);
END_IF;
```

F51_DBMUL

8-digit BCD multiplication, destination can be 11 specified

Description Multiplies the 32-bit BCD (8-digit) equivalent constant or 8-digit BCD data specified by **s1** and the one specified by **s2** if the trigger **EN** is in the ON-state. The result is stored in the ARRAY **d[0]**, **d[1]** (64-bit area).



Example value 16#60008 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 1 1 0	0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0
16# BCD	0	0	0	6	0	0	0	8

← 32-bit area →

×

Example value 16#40002 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0
16# BCD	0	0	0	4	0	0	0	2

⬇

Result value 16#2400440016 (BCD) if trigger is ON stored in the ARRAY [0..1] of DWORD

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d array[0]	0 0 0 0	0 0 0 0	0 1 0 0	0 1 0 0	0 0 0 0	0 0 0 0	0 0 0 1	0 1 1 0
16# BCD	0	0	4	4	0	0	1	6

← output_array[0] →

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d array[1]	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0	0 1 0 0
16# BCD	0	0	0	0	0	0	2	4

← output_array[1] →

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F51_DBMUL **(see page 1325)**

Data types

Variable	Data type	Function
s1	DWORD	multiplicand, 32-bit area for 8-digit BCD data or equivalent constant
s1	DWORD	multiplier, 32-bit area for 8-digit BCD data or equivalent constant
d	ARRAY [0..1] of DWORD	result

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

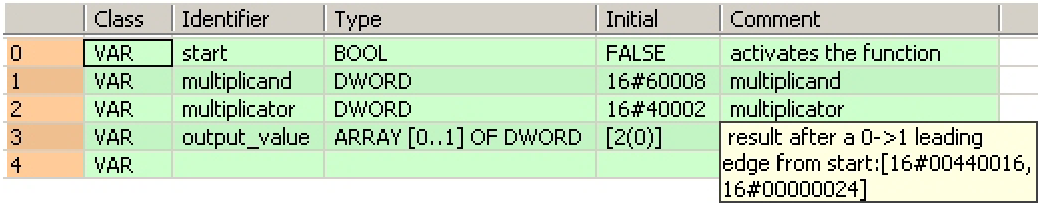
No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.

Example

In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header

All input and output variables used for programming this function have been declared in the POU header.

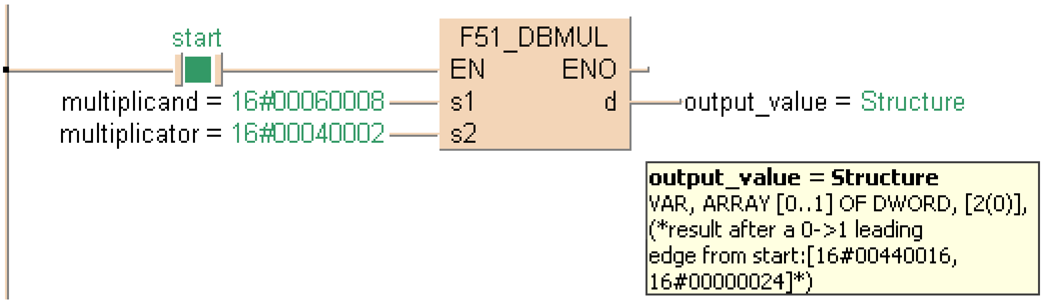


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	multiplicand	DWORD	16#60008	multiplicand
2	VAR	multiplicator	DWORD	16#40002	multiplicator
3	VAR	output_value	ARRAY [0..1] OF DWORD	[2(0)]	result after a 0->1 leading edge from start:[16#00440016, 16#00000024]
4	VAR				

Body

When the variable **start** is set to TRUE, the function is carried out.

LD



output_value = Structure
VAR, ARRAY [0..1] OF DWORD, [2(0)],
(*result after a 0->1 leading
edge from start:[16#00440016,
16#00000024]*)

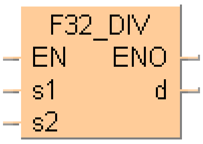
ST

```
IF start THEN
    F51_DBMUL(multiplicand, multiplicator, output_value);
END_IF;
```

F32_DIV

16-bit division, destination can be specified

Description The 16-bit data or 16-bit equivalent constant specified by **s1** is divided by the 16-bit data or 16-bit equivalent constant specified by **s2** if the trigger **EN** is in the ON-state.



The quotient is stored in **d** and the remainder is stored in the special data register DT9015 (DT90015 for FP2/2SH and FP10/10S/10SH). All 16-bit values are treated as integer values.

Example value 36

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0 0 0 0	0 0 0 0	0 0 1 0	0 1 0 0

÷

Example value 17

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1

⬇

Result value 2 if trigger is ON

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0

Remainder 2 stored in DT9015/90015

15 . . 12	10 . . 8	7 . . 4	3 . . 0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction DIV (see page 64). Please refer also to Advantages of the IEC instructions in the online help.

PLC types **Availability of** F32_DIV **(see page 1324)**

Data types

Variable	Data type	Function
s1	ANY16	dividend
s2		divisor
d		quotient

The variables **s1, s2** and **d** have to be of the same data type.

Operands

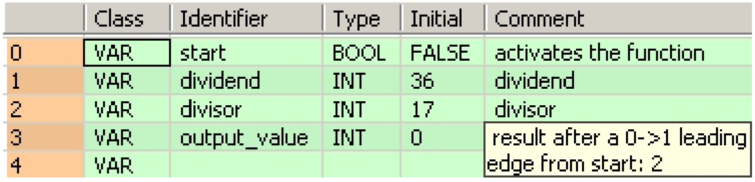
For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculated result is 0.
R9009	%MX0.900.9	for an instant	▪ the negative minimum value -32768 (16#8000) is divided by -1 (16#FFFF)

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

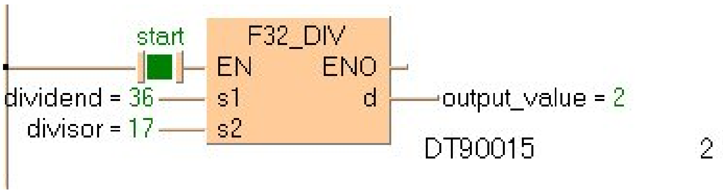
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	dividend	INT	36	dividend
2	VAR	divisor	INT	17	divisor
3	VAR	output_value	INT	0	result after a 0->1 leading edge from start: 2
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD

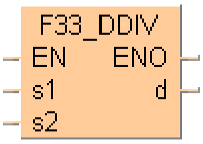


ST
```
IF start THEN
    F32_DIV(dividend, divisor, output_value);
END_IF;
```

F33_DDIV

32-bit division, destination can be specified

Description The 32-bit data or 32-bit equivalent constant specified by **s1** is divided by the 32-bit data or 32-bit equivalent constant specified by **s2** if the trigger **EN** is in the ON-state. The quotient is stored in **d** and the remainder is stored in the special data registers DDT9015 (DDT90015 for FP2/2SH and FP10/10S/10SH). All 32-bit values are treated as double integer values.



Example value 16908416

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0000	0001	0000	0010	0000	0000	1000	0000

32-bit area

Example value 589828

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	0000	0000	0000	1001	0000	0000	0000	0100

Result value 28 if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0000	0000	0000	0000	0000	0000	0001	1100

Remainder 393232

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
	0000	0000	0000	0110	0000	0000	0001	0000

DT9016/DDT90016 DT9015/DDT90015

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction DIV (see page 64). Please refer also to Advantages of the IEC instructions in the online help.

PLC types **Availability of** F33_DDIV **(see page** 1324**)**

Data types

Variable	Data type	Function
s1	ANY32	dividend
s2		divisor
d		quotient

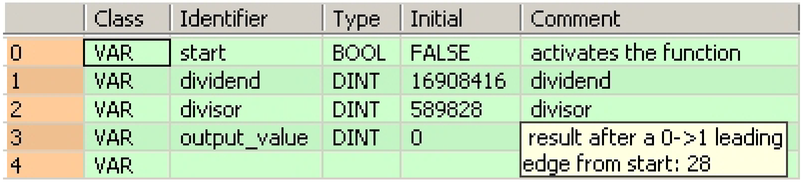
The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages.

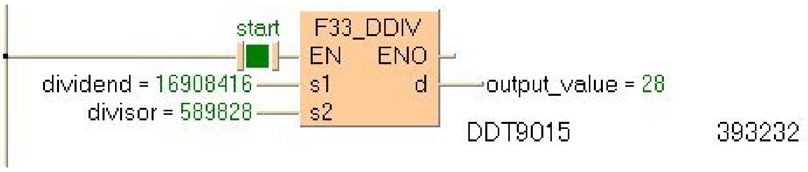
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	dividend	DINT	16908416	dividend
2	VAR	divisor	DINT	589828	divisor
3	VAR	output_value	DINT	0	result after a 0->1 leading edge from start: 28
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD

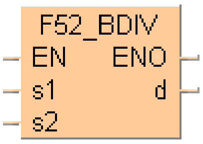


ST
```
IF start THEN
    F33_DDIV(dividend, divisor, output_value);
END_IF;
```

F52_BDIV

4-digit BCD division, destination can be specified

Description The 4-digit BCD equivalent constant or the 16-bit area for 4-digit BCD data specified by **s1** is divided by the 4-digit BCD equivalent constant or the 16-bit area for 4-digit BCD data specified by **s2** if the trigger **EN** is in the ON-state.



The quotient is stored in the area specified by **d** and the remainder is stored in special data register DT9015 (DT90015 for FP2/2SH and FP10/10S/10SH).

Example value 16#0037 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 1 1	0 1 1 1
16# (BCD)	0	0	3	7

÷

Example value 16#0015 (BCD)

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 0 0 0	0 0 0 1	0 1 0 1
16# (BCD)	0	0	1	5

⬇ **Trigger: ON**

Result value 16#0002

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0
16# (BCD)	0	0	0	2

Remainder 16#0007

Bit	15 . . 12	10 . . 8	7 . . 4	3 . . 0
DT9015	0 0 0 0	0 0 0 0	0 0 0 0	0 1 1 1
16# (BCD)	0	0	0	7

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F52_BDIV **(see page 1325)**

Data types

Variable	Data type	Function
s1	WORD	dividend, 16-bit area for BCD data or 4-digit BCD equivalent constant
s2	WORD	divisor, 16-bit area for BCD data or 4-digit BCD equivalent constant
d	WORD	quotient, 16-bit area for BCD data (remainder stored in special data register DT9015/DT90015)

Operands

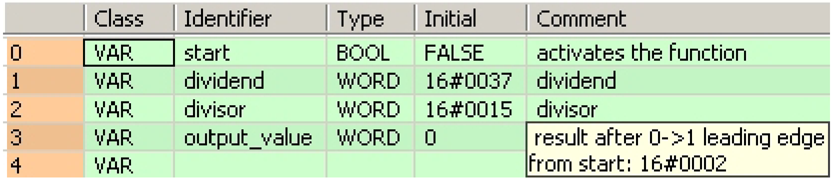
For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the result calculated is 0.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

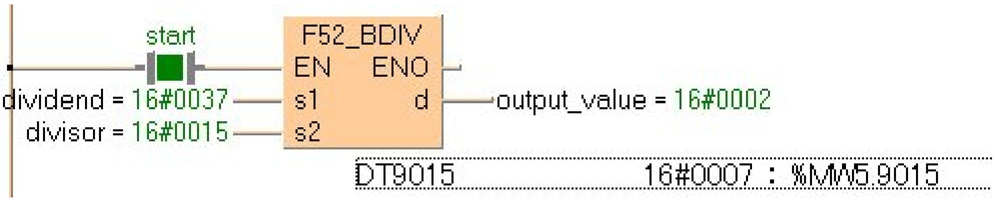
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	dividend	WORD	16#0037	dividend
2	VAR	divisor	WORD	16#0015	divisor
3	VAR	output_value	WORD	0	result after 0->1 leading edge from start: 16#0002
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD

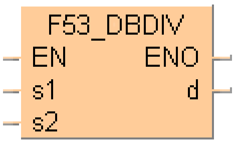


ST
```
IF start THEN
    F52_BDIV(dividend, divisor, output_value);
END_IF;
```

F53_DBDIV

8-digit BCD division, destination can be specified

Description The result is stored in the area specified by **d**, and the remainder is stored in the special data registers DT9016 and DT9015 (DT90016 and DT90015 for FP2/2SH and FP10/10S/10SH).



Example value 16#00001110 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1	0 0 0 0
16# BCD	0	0	0	0	0	1	1	0

32-bit area

÷

Example value 16#0000011 (BCD)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1
16# BCD	0	0	0	0	0	0	1	1

⬇

Result value 16#00000100 (BCD) if trigger is ON

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0	0 0 0 0
16# BCD	0	0	0	0	0	1	0	0

Remainder 16#00000010 (BCD) if trigger is ON stored in DT9015 to DT9016 (DDT90015 to DDT90016)

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0
16# BCD	0	0	0	0	0	0	1	0
	DT9016/DDT90016				DT9015/DDT90015			

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F53_DBDIV **(see page 1325)**

Data types

Variable	Data type	Function
s1	DWORD	dividend, 32-bit area for BCD data or 8-digit BCD equivalent constant
s2	DWORD	divisor, 32-bit area for BCD data or 8-digit BCD equivalent constant
d	DWORD	quotient, 32-bit area for BCD data (remainder stored in special data register DT9016 and DT9015/DT90016 and DT90015)

Operands

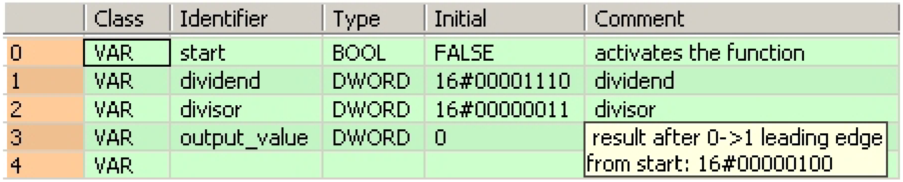
For	Relay				T/C		Register			Constant
s1, s2, s3	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the result calculated is 0.

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

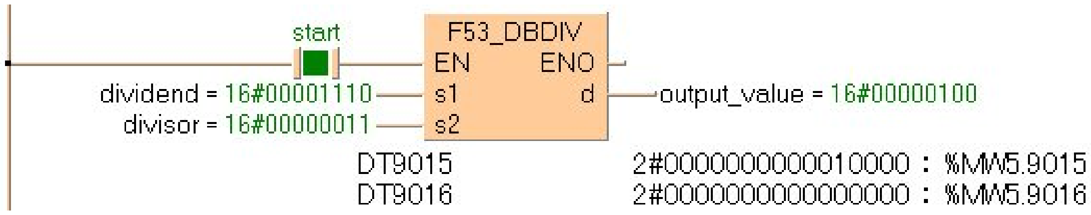
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	dividend	DWORD	16#00001110	dividend
2	VAR	divisor	DWORD	16#00000011	divisor
3	VAR	output_value	DWORD	0	result after 0->1 leading edge from start: 16#00000100
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD

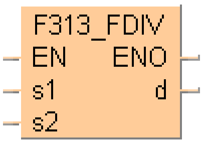

start
F53_DBDIV
EN
ENO
dividend = 16#00001110
s1
d
output_value = 16#00000100
divisor = 16#00000011
s2
DT9015
DT9016
2#0000000000010000 : %MW5.9015
2#0000000000000000 : %MW5.9016

ST
```
IF start THEN
    F53_DBDIV(dividend, divisor, output_value);
END_IF;
```

F313_FDIV

Floating Point Data Divide

Description The real number data specified by **s1** is divided by the real number data specified by **s2** when the trigger turns on. The result is stored in **d**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction DIV (see page 64). Please refer also to Advantages of the IEC instructions in the online help.

PLC types **Availability of** F313_FDIV **(see page 1324)**

☞ **This instruction cannot be programmed in the interrupt program.**

Data types

Variable	Data type	Function
s1	REAL	Real number data for dividend.
s2	REAL	Real number data for divisor.
d	REAL	32-bit area for result (destination).

Operands

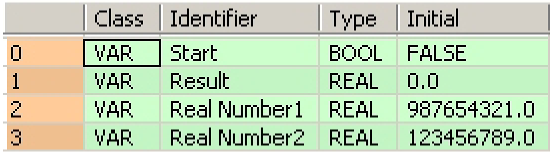
For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ data other than real number data is specified in s1 and s2. ▪ the real number data (floating point data) for the divisor specified by s2 is "0.0".
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	▪ the result is overflowed.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

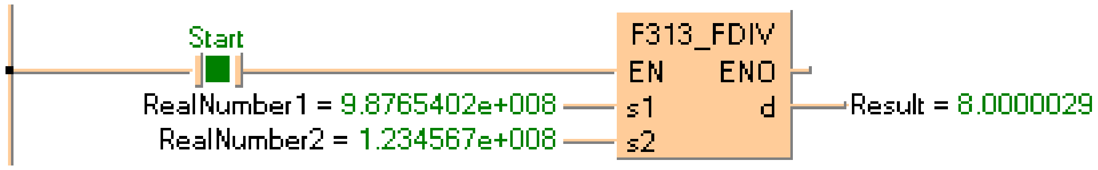
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Start	BOOL	FALSE
1	VAR	Result	REAL	0.0
2	VAR	Real Number1	REAL	987654321.0
3	VAR	Real Number2	REAL	123456789.0

Body When the variable **Start** is set to TRUE, the real number entered for the variable **RealNumber1** is divided by the real number entered for **RealNumber2** and the result stored at the address assigned by the compiler to the variable **Result**. The monitor value icon is activated.

LD



RealNumber1 = 9.8765402e+008

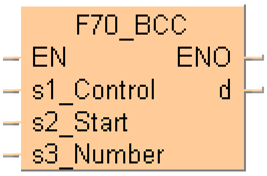
RealNumber2 = 1.234567e+008

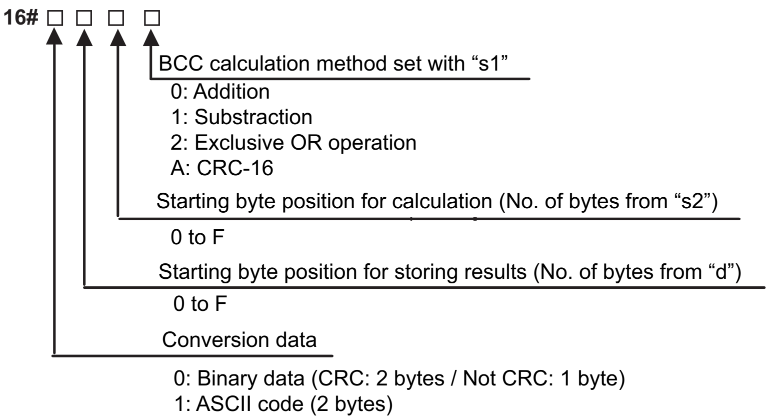
Result = 8.0000029

F70_BCC

Block check code calculation

Description Calculates the Block Check Code (**BCC**), which is used to detect errors in message transmission, of **s3** bytes of ASCII data starting from the 16-bit area specified by **s2** according to the calculation method specified by **s1**. The Block Check Code (**BCC**) is stored in the lower byte of the 16-bit area specified by **d**. (BCC is one byte. The higher byte of **d** does not change.)




16# □ □ □ □

BCC calculation method set with “s1”
0: Addition
1: Substraction
2: Exclusive OR operation
A: CRC-16

Starting byte position for calculation (No. of bytes from “s2”)
0 to F

Starting byte position for storing results (No. of bytes from “d”)
0 to F

Conversion data
0: Binary data (CRC: 2 bytes / Not CRC: 1 byte)
1: ASCII code (2 bytes)

If CRC-16 is specified as the calculation method, ASCII code cannot be specified for the conversion data.

PLC types **Availability of** F70_BCC **(see page** 1326**)**

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Data types

Variable	Data type	Function
s1	INT	specifies BCC calculation method: 0 = addition, 1 = subtraction, 2 = exclusive OR operation
s2	ANY16	starting 16-bit area to calculate BCC
s3	INT	specifies number of bytes for BCC calculation
d	ANY16	16-bit area for storing BCC

Operands

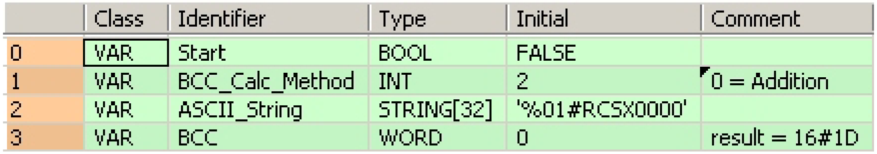
For	Relay				T/C		Register			Constant
s1, s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the number of specified bytes for the target data exceeds the limit of the specified data area.
R9008	%MX0.900.8	for an instant	

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

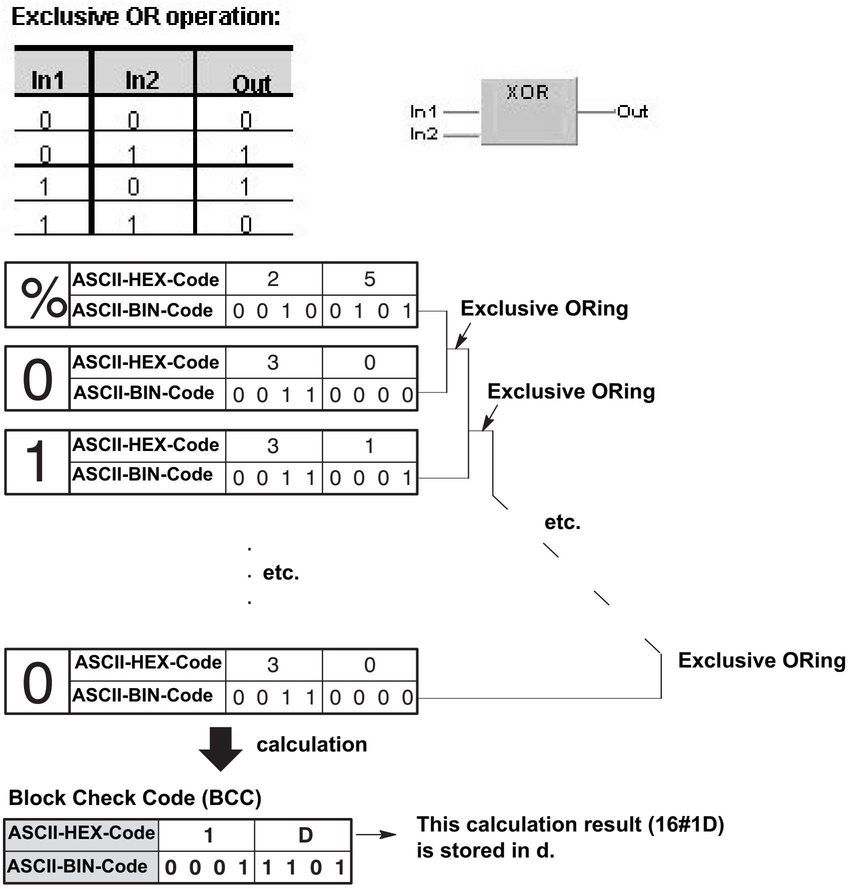
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	BCC_Calc_Method	INT	2	0 = Addition
2	VAR	ASCII_String	STRING[32]	'%01#RCSX0000'	
3	VAR	BCC	WORD	0	result = 16#1D

Body A block check code is performed on the value entered for the variable **ASCII_String** when **Start** becomes TRUE. The exclusive OR operation, which is more suitable when large amounts of data are transmitted, has been chosen for the BCC method.

How the BCC is calculated using the exclusive OR operation:



Exclusive OR operation:

In1	In2	Out
0	0	0
0	1	1
1	0	1
1	1	0

%									
ASCII-HEX-Code	2				5				
ASCII-BIN-Code	0	0	1	0	0	1	0	1	

0								
ASCII-HEX-Code	3				0			
ASCII-BIN-Code	0	0	1	1	0	0	0	0

1								
ASCII-HEX-Code	3				1			
ASCII-BIN-Code	0	0	1	1	0	0	0	1

0								
ASCII-HEX-Code	3				0			
ASCII-BIN-Code	0	0	1	1	0	0	0	0

Block Check Code (BCC)

ASCII-HEX-Code	1				D			
ASCII-BIN-Code	0	0	0	1	1	1	0	1

This calculation result (16#1D) is stored in d.

The ASCII BIN code bits of the first two characters are compared with each other to yield an 8-character exclusive OR operation result:

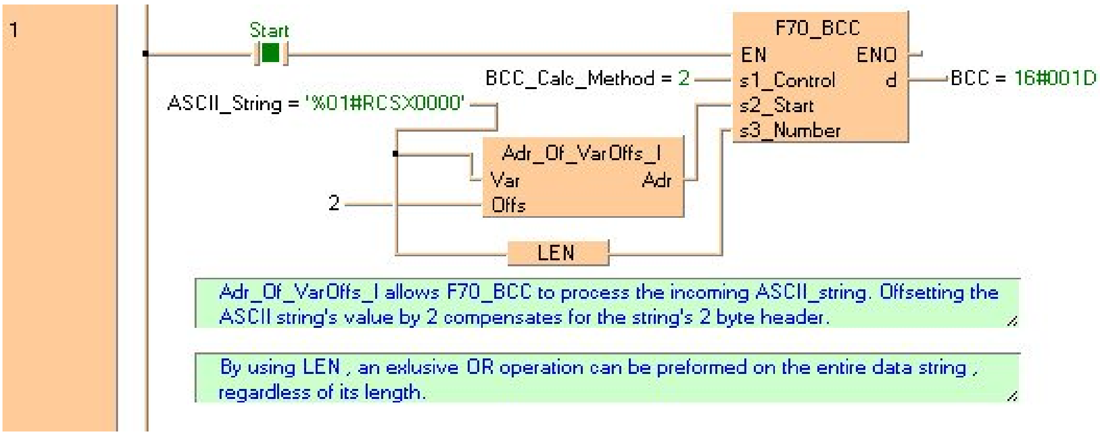
Sign for comparison	ASCII BIN code
%	00100101
0	00110000
Exclusive OR result	00010101

This result is then compared to the ASCII BIN code of the next character, i.e. "1".

Sign for comparison	ASCII BIN code
Exclusive OR result	00010101
1	00110001
Next exclusive OR	00100100

And so on until the final character is reached.

LD



Adr_Of_VarOffs_I allows F70_BCC to process the incoming ASCII_string. Offsetting the ASCII string's value by 2 compensates for the string's 2 byte header.

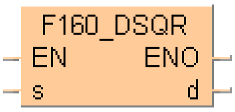
By using LEN , an exlusive OR operation can be preformed on the entire data string , regardless of its length.

ST
```
IF start THEN
     F70_BCC( s1_Control:= BCC_Calc_Methode,
         s2_Start:= Adr_Of_VarOffs( Var:= ASCII_String,
         Offs:= 2),
         s3_Number:= LEN( ASCII_String),
         d=> BCC);
END_IF;
```

F160_DSQR

32-bit data square root

Description The square root of the 32-bit data or constant value specified by **s** is calculated if the trigger **EN** is in the ON-state. The result (square root) is stored in **d**.



The figures of the first decimal place and below are disregarded.

Example value 64

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
Binary	0000	0000	0000	0000	0000	0000	0100	0000
Decimal	64							
	32-bit area							

Trigger: ON

Result value 8

Bit	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
Binary	0000	0000	0000	0000	0000	0000	0000	0100
Decimal	8							

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction SQRT (see page 68). Please refer also to Advantages of the IEC instructions in the online help.

PLC types **Availability of** F160_DSQR **(see page** 1321**)**

Data types

Variable	**Data type**	**Function**
s	DINT, DWORD	source, 32-bit area to be calculated
d	DINT, DWORD	square root (decimal places deleted)

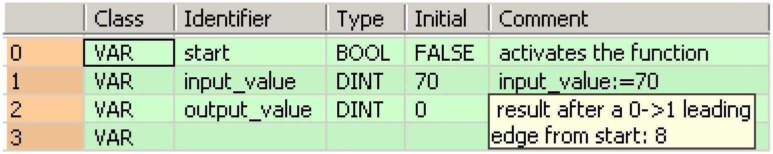
The variables **s1** and **d** have to be of the same data type.

Operands

For	**Relay**				**T/C**		**Register**			**Constant**
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

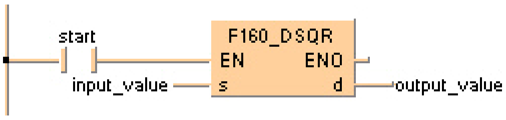
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	DINT	70	input_value:=70
2	VAR	output_value	DINT	0	result after a 0->1 leading edge from start: 8
3	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD

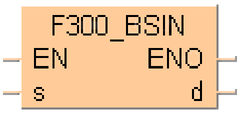


ST
```
IF start THEN
    F160_DSQR(input_value, output_value);
END_IF;
```

F300_BSIN

BCD type Sine operation

Description The function calculates the sine of **BCD** code angular data (input **s**) and stores the result (output **d**) as a **BCD** value in an array with three elements.



BCD values for input **s** lie in the area from 0° to 360° (16#0 to 16#360) in 1° steps. With this, output **d** can yield a result in the range of -1.0000 to 1.0000. The result is returned as follows:

ARRAY[0]	preceding sign (0 when input is +, 1 when input is -)
ARRAY[1]	whole number before the decimal point (0 or 1)
ARRAY[2]	numbers after the decimal point with 4 significant figures as a BCD value (16#0000 to 16#9999).

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F300_BSIN **(see page 1323)**

Data types

Variable	Data type	Function
s	WORD	16-bit area where angle data is stored
d	ARRAY [0..2] of WORD	result stored in 3 words

Operands

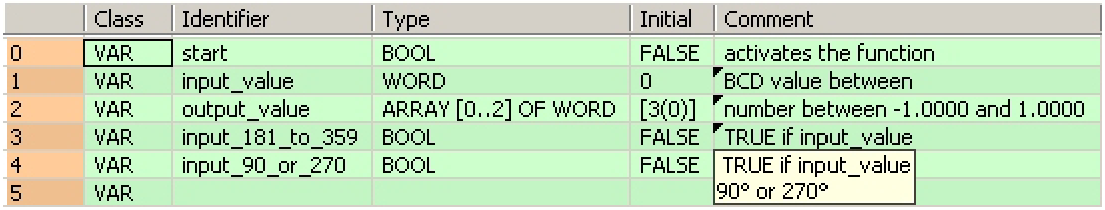
For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the input value for s is not a BCD value or is not between 0° and 360°.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	for an instant	▪ the result is 0.
R9009	%MX0.900.9	for an instant	▪ the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help. In addition, an analytical program is created that interprets the result. The same POU header is used for both programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.

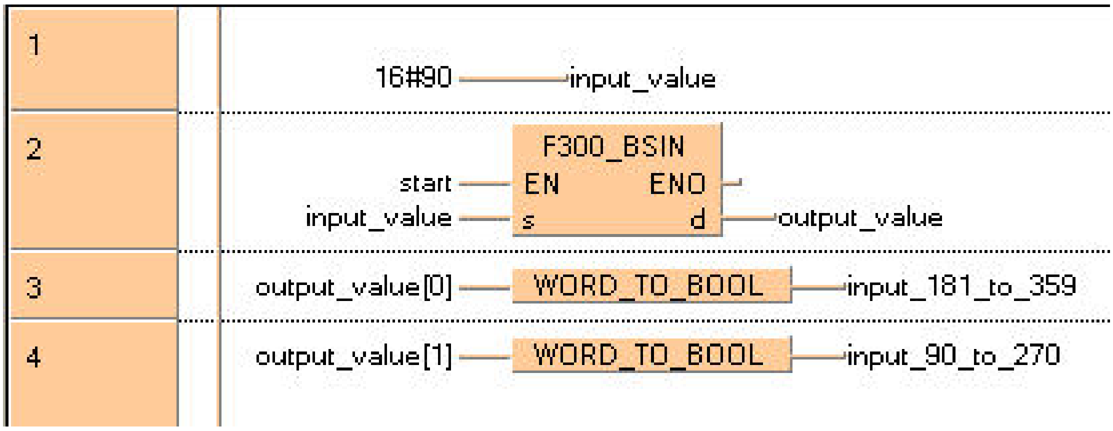


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	0	BCD value between
2	VAR	output_value	ARRAY [0..2] OF WORD	[3(0)]	number between -1.0000 and 1.0000
3	VAR	input_181_to_359	BOOL	FALSE	TRUE if input_value
4	VAR	input_90_or_270	BOOL	FALSE	TRUE if input_value 90° or 270°
5	VAR				

In this example, the input variable **input_value** is declared. However, you can write a constant (e.g. 16#45 for 45°) directly at the input contact of the function.

Body In the body, the value 90° is assigned to the variable **input_value**. When the variable **start** is set to TRUE, the function F300_BSIN is carried out. It stores the result in the variable **output_value**. If the **input_value** is between 181° and 359°, **output_value** has a minus sign. The function WORD_TO_BOOL sets the variable **input_181_to_359** to TRUE. With an **input_value** of 90° or 270°, the **output_value** is 1, which represents the value before the decimal point. If this is the case, then WORD_TO_BOOL sets the value of the variable **input_90_or_270** to TRUE.

LD

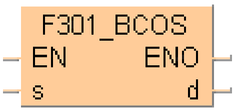


ST
```
input_value:=16#90;
IF start THEN
    F300_BSIN( input_value, output_value );
END_IF;
input_181_to_359:=WORD_TO_BOOL(output_value[0]);
input_90_or_270:=WORD_TO_BOOL(output_value[1]);
```

F301_BCOS

BCD type Cosine operation

Description The function calculates the cosine of **BCD** code angular data (input **s**) and stores the result (output **d**) as a **BCD** value in an array with three elements.



BCD values for input **s** lie in the area from 0° to 360° (16#0 to 16#360) in 1° steps. With this output **d** can yield a result in the range of -1.0000 to 1.0000. The result is returned as follows:

ARRAY[0]	preceding sign (0 when input is +, 1 when input is -)
ARRAY[1]	whole number before the decimal point (0 or 1)
ARRAY[2]	numbers after the decimal point with 4 significant figures as a BCD value (16#0000 to 16#9999).

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F301_BCOS **(see page 1323)**

Data types

Variable	Data type	Function
s	WORD	area where angle data is stored
d	ARRAY [0..2] of WORD	result stored in 3 words

Operands

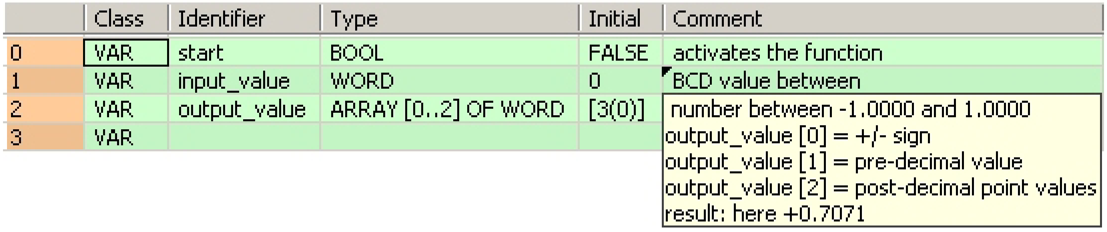
For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the input value for s is not a BCD value or is not between 0° and 360°.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	for an instant	▪ the result is 0.
R9009	%MX0.900.9	for an instant	▪ the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header In the POU header, all input and output variables are declared that are used for programming this function.



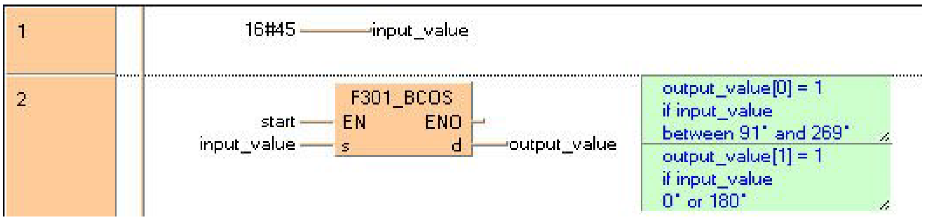
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	0	BCD value between
2	VAR	output_value	ARRAY [0..2] OF WORD	[3(0)]	
3	VAR				

number between -1.0000 and 1.0000
output_value [0] = +/- sign
output_value [1] = pre-decimal value
output_value [2] = post-decimal point values
result: here +0.7071

In this example, the input variable **input_value** is declared. However, you can write a constant (e.g. 16#45 for 45°) directly at the input contact of the function.

Body In the body, the value 16#45° is assigned to the variable **input_value**. When the variable **start** is set to TRUE, the function is carried out. The result at output d **isoutput_value[0]** = 0, **output_value[1]** = 0, **output_value[2]** = 7071.

LD



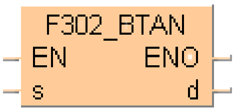
output_value[0] = 1
if input_value
between 91° and 269°
output_value[1] = 1
if input_value
0° or 180°

ST
```
input_value:=16#45;
IF start THEN
    F301_BCOS( input_value, output_value );
END_IF;
```

F302_BTAN

BCD type Tangent operation

Description The function calculates the tangent of **BCD** code angular data (input **s**) and stores the result (output **d**) as a **BCD** value in an array with three elements.



BCD values for input **s** lie in the area from 0° to 360° (16#0 to 16#360) in 1° steps. With this output **d** yields a result in the range of -57.2900 to 57.2900. The result is returned as follows:

ARRAY[0]	preceding sign (0 when input is +, 1 when input is -)
ARRAY[1]	whole number before the decimal point as BCD value (16#0 to 16#57)
ARRAY[2]	numbers after the decimal point with 4 significant figures as BCD value (16#0000 to 16#9999)

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F302_BTAN **(see page 1324)**

Data types

Variable	Data type	Function
s	WORD	area where angle data is stored
d	ARRAY [0..2] of WORD	result stored in 3 words

Operands

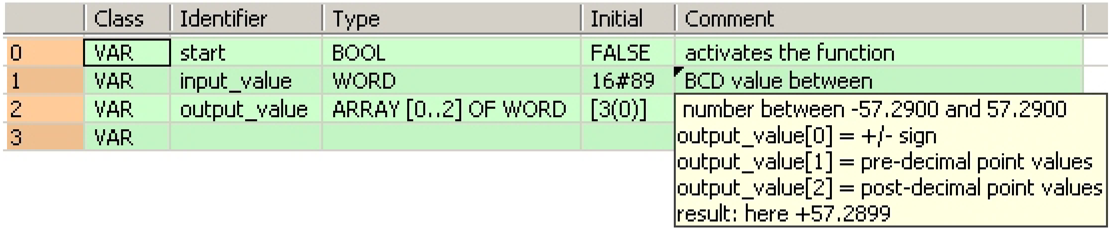
For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the input value for **s** is not a BCD value ▪ **s** = 90° (16#90) or 270° (16#270)
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	▪ the result is 0.
R9009	%MX0.900.9	for an instant	▪ the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header In the POU header, all input and output variables are declared that are used for programming this function.



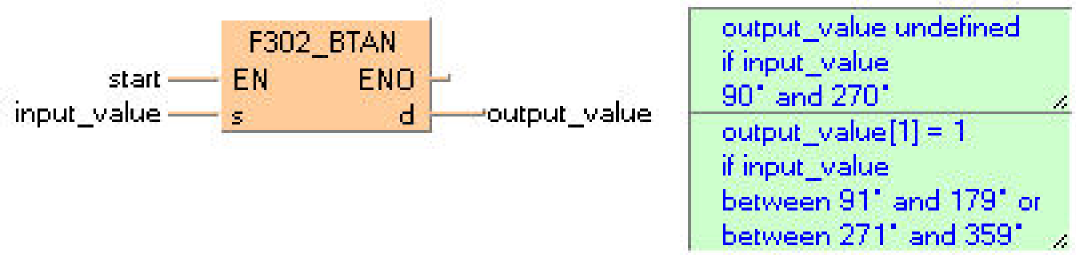
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	16#89	BCD value between
2	VAR	output_value	ARRAY [0..2] OF WORD	[3(0)]	
3	VAR				

number between -57.2900 and 57.2900
output_value[0] = +/- sign
output_value[1] = pre-decimal point values
output_value[2] = post-decimal point values
result: here +57.2899

In this example, the input variable **input_value** is declared. However, you can write a constant (e.g. 16#89 for 89°) directly at the input contact of the function.

Body When the variable **start** is set to TRUE, the function is carried out. The **input_value** was initialized with the value 16#89 (89°) in the POU header. The result is written to the ARRAY **output_value**. Here in the first element of the ARRAY, the **output_value** = 16# (+ sign). In the second element, 16#57 represents the number before the decimal point, and 16#2899 comes after the decimal point in the third element.

LD



output_value undefined
if input_value
90° and 270°

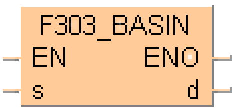
output_value[1] = 1
if input_value
between 91° and 179° or
between 271° and 359°

ST
```
IF start THEN
    F302_BTAN(input_value, output_value);
END_IF;
```

F303_BASIN

BCD type Arcsine operation

Description The function calculates the arcsine of a **BCD** value that is entered at input **s** as an ARRAY with three elements. The result is returned as **BCD** angular data in the range of 0° to 360° (16#0 to 16#360) at output **d**.



BCD values for input **s** lie in the area from -1.0000 to 1.0000. They are entered as follows:

ARRAY[0]	preceding sign (0 when input is +, 1 when input is -)
ARRAY[1]	whole number before the decimal point (0 or 1)
ARRAY[2]	numbers after the decimal point with 4 significant figures as a BCD value (16#0000 to 16#9999).

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F303_BASIN **(see page 1324)**

Data types

Variable	Data type	Function
s	ARRAY [0..2] of WORD	area where angle data is stored
d	WORD	result stored in 3 words

Operands

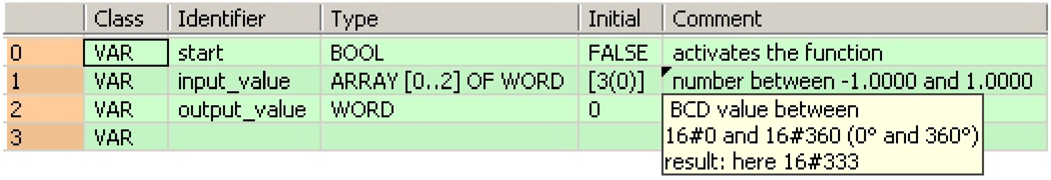
For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the input value for s is not a BCD value or is not between -1.0000 and 1.0000
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	▪ the result is 0.
R9009	%MX0.900.9	for an instant	▪ the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

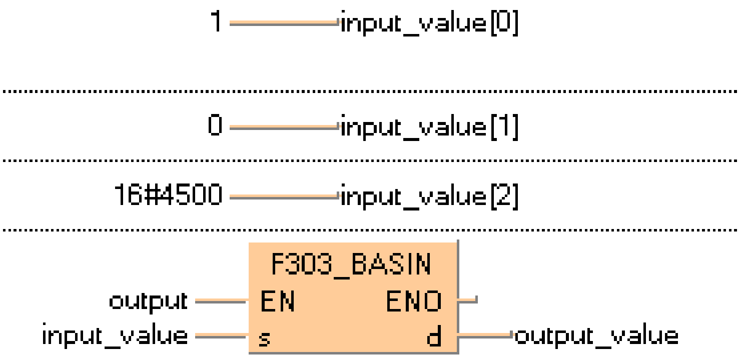
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	ARRAY [0..2] OF WORD	[3(0)]	number between -1.0000 and 1.0000
2	VAR	output_value	WORD	0	BCD value between 16#0 and 16#360 (0° and 360°) result: here 16#333
3	VAR				

Body The first element of the ARRAY’s **input_value** is given the value 1 (- sign). The second element has 0 as its whole number value, and in the third element 16#4500 is written as the value after the decimal point. When the variable **start** is set to TRUE, the function is carried out. The result for the **output_value** = 16#333 (333°).

LD



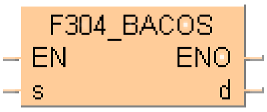
ST

```
input_value[0]:=1;
input_value[1]:=0;
input_value[2]:=16#4500;
IF start THEN
    F303_BASIN(input_value, output_value);
END_IF;
```

F304_BACOS

BCD type Arccosine operation

Description The function calculates the arccosine of a **BCD** value that is entered at input **s** as an ARRAY with three elements. The result is returned as **BCD** angular data in the range of 0° to 360° (16#0 to 16#360) at output **d**.



BCD values for input **s** lie in the area from -1.0000 to 1.0000. They are entered as follows:

ARRAY[0]	preceding sign (0 when input is +, 1 when input is -)
ARRAY[1]	whole number before the decimal point (0 or 1)
ARRAY[2]	numbers after the decimal point with 4 significant figures as a BCD value (16#0000 to 16#9999).

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F304_BACOS **(see page 1324)**

Data types

Variable	Data type	Function
s	ARRAY [0..2] of WORD	area where angle data is stored in 3 words
d	WORD	result

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

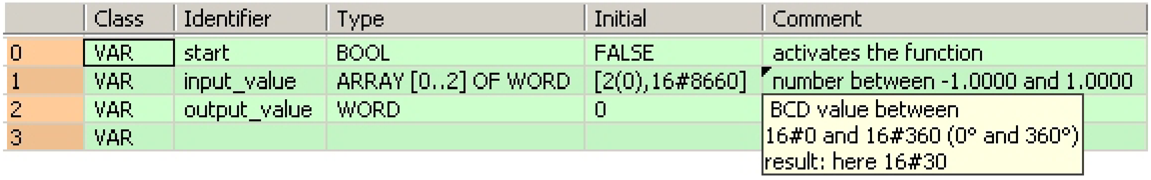
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the input value for s is not a BCD value or is not between -1.0000 and 1.0000.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	▪ the result is 0.
R9009	%MX0.900.9	for an instant	▪ the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

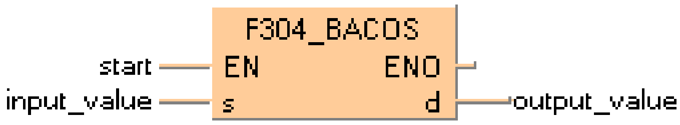


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	ARRAY [0..2] OF WORD	[2(0),16#8660]	number between -1.0000 and 1.0000
2	VAR	output_value	WORD	0	
3	VAR				

BCD value between
16#0 and 16#360 (0° and 360°)
result: here 16#30

Body When the variable **start** is set to TRUE, the function is carried out. The **input_value** = 0 (+ sign) in the ARRAY’s first element. 0 represents the whole in the second element, and the value after the decimal point is 8660. The function thus calculates the **output_value** = 16#30 (30°).

LD

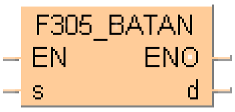


ST
```
IF start THEN
    F304_BACOS(input_value, output_value);
END_IF;
```

F305_BATAN

BCD type Arctangent operation

Description The function calculates the arctangent of a **BCD** value that is entered at input **s** as an ARRAY with three elements. The result is returned as **BCD** angular data in the range 0° to 90° (16#0 to 16#90) or 270° to 360° (16#270 to 16#360) at output **d**.



BCD values for input **s** lie in the area from -9999.9999 to 9999.9999. They are entered as follows:

ARRAY[0]	preceding sign (0 when input is +, 1 when input is -)
ARRAY[1]	whole number before the decimal point as BCD value (16#0 to 16#9999)
ARRAY[2]	numbers after the decimal point with 4 significant figures as a BCD value (16#0000 to 16#9999).

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F305_BATAN **(see page 1324)**

Data types

Variable	Data type	Function
s	ARRAY [0..2] of WORD	area where angle data is stored in 3 words
d	WORD	result

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

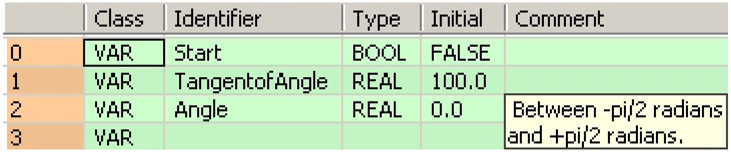
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the input value at s is not a BCD value.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	▪ the result is 0.
R9009	%MX0.900.9	for an instant	▪ the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

All input and output variables used for programming this function have been declared in the POU header.

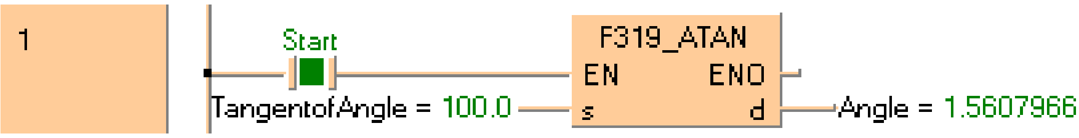


	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	TangentofAngle	REAL	100.0	
2	VAR	Angle	REAL	0.0	Between -pi/2 radians and +pi/2 radians.
3	VAR				

Body

When the variable **Start** is set to TRUE, the Arctangent of the real number entered for the variable **TangentofAngle** is calculated and the result stored at the address assigned by the compiler to the variable **Angle** (units are radians).

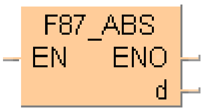
LD



F87_ABS

16-bit data absolute value

Description Gets the absolute value of 16-bit data with the sign specified by **d** if the trigger **EN** is in the ON-state.



The absolute value of the 16-bit data with +/- sign is stored in **d**. This instruction is useful for handling data whose sign (+/-) may vary.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction ABS (see page 66). Please refer also to Advantages of the IEC instructions in the online help.

PLC types **Availability of** F87_ABS **(see page 1326)**

Data types

Variable	Data type	Function
d	ANY16	16-bit area for storing original data and its absolute value

Operands

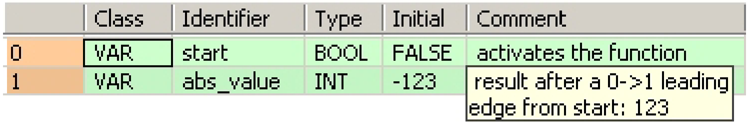
For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the 16-bit data is the negative minimum value -32768 (16#8000).
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	▪ the 16-bit data is the negative value in the range from -1 to -32767 (16#FFFF to 16#8001).

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

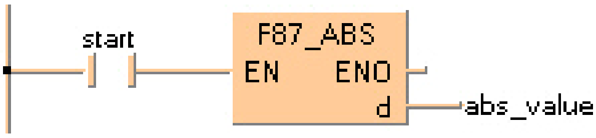
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	abs_value	INT	-123	result after a 0->1 leading edge from start: 123

Body When the variable **start** is set to TRUE, the function is carried out.

LD

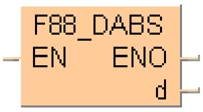


```
ST IF start THEN
       F87_ABS(abs_value);
   END_IF;
```

F88_DABS

32-bit data absolute value

Description Gets the absolute value of 32-bit data with the sign specified by **d** if the trigger **EN** is in the ON-state. The absolute value of the 32-bit data with sign is stored in **d**. This instruction is useful for handling data whose sign (+/-) may vary.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction ABS (see page 66). Please refer also to Advantages of the IEC instructions in the online help.

PLC types **Availability of** F88_DABS **(see page 1326)**

Data types

Variable	Data type	Function
d	ANY32	32-bit area for storing original data and its absolute value

Operands

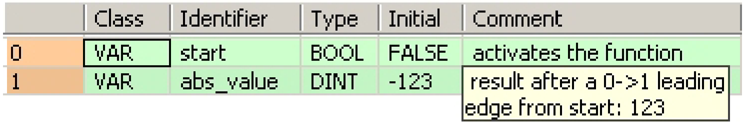
For	Relay				T/C		Register			Constant
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the 32-bit data is the negative minimum value -2147483648 (16#80000000).
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	▪ the 32-bit data is the negative value in the range from -1 to -2147483647 (16#FFFFFFFF to 16#80000001).

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

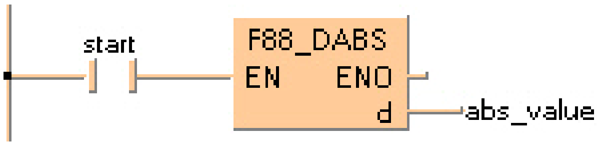
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	abs_value	DINT	-123	result after a 0->1 leading edge from start: 123

Body When the variable **start** is set to TRUE, the function is carried out.

LD

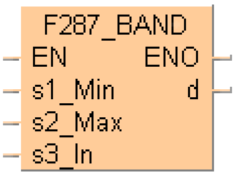


```
ST IF start THEN
            F88_DABS(abs_value);
   END_IF;
```

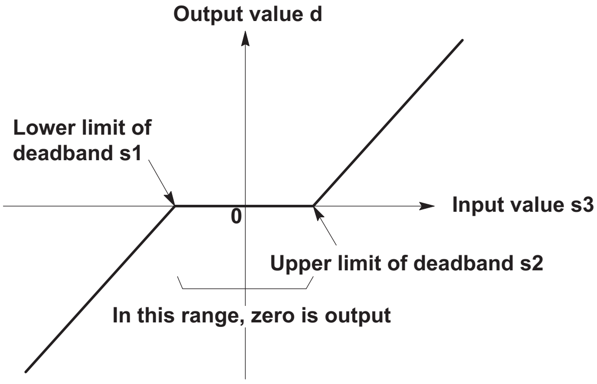
F287_BAND

16-bit data deadband control

Description The function compares the input value at input **s3** with a deadband whose lower limit is specified at input **s1** and whose upper limit is specified at **s2**. The result of the function is returned at output **d** as follows:



- If the input value at input **s3** < **s1**, the lower limit at input **s1** is subtracted from the input value at **s3**, and the result is stored as the output value at **d**.
- If the input value at input **s3** > **s2**, the upper limit at input **s2** is subtracted from the input value at **s3**, and the result is stored as the output value at **d**.
- If the input value at **s2** ≥ **s3** ≥ **s1**, 0 is returned as the output value at **d**.


Output value d
Lower limit of deadband s1
0
Input value s3
Upper limit of deadband s2
In this range, zero is output

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F287_BAND **(see page 1323)**

Data types

Variable	Data type	Function
s1	ANY16	the area where the lower limit is stored or the lower limit data
s2		the area where the upper limit is stored or the upper limit data
s3		the area where the input value is stored or the input value data
d		the area where the output value data is stored

Operands

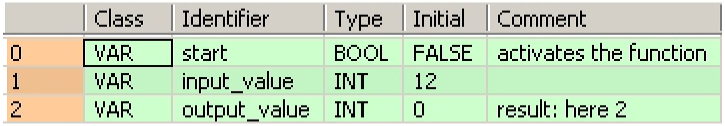
For	Relay				T/C		Register			Constant
s1, s2, s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the value at **s1** > **s2**.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	TRUE	▪ the input value at s3 is 0.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

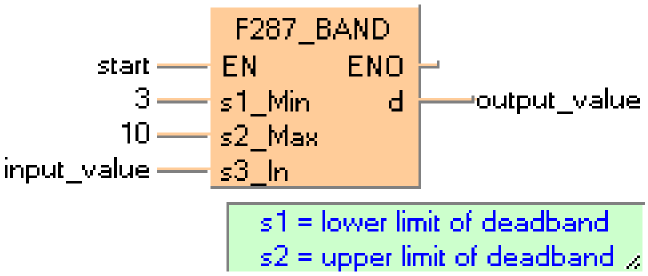


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	INT	12	
2	VAR	output_value	INT	0	result: here 2

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out. The constant 3 (lower limit of the deadband) and 10 (upper limit of the deadband) are assigned to inputs s1 and s2. However, you can declare variables in the POU header and write them in the function in the body at the inputs.

LD

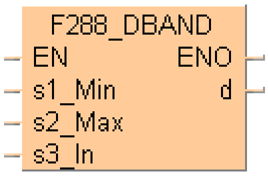


ST
```
IF start THEN
    F287_BAND( 3, 10, input_value, output_value);
END_IF;     (* 3=lower limit of deadband, 10=upper limit of deadband *)
```

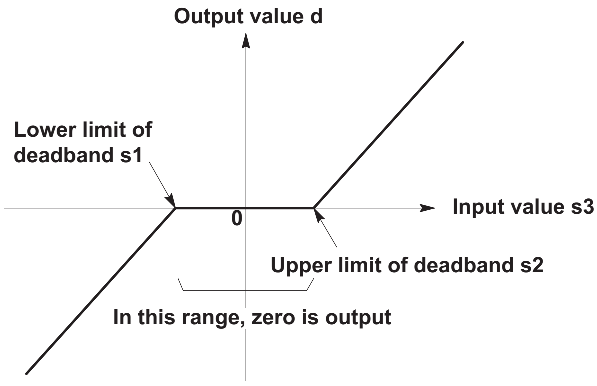
F288_DBAND

32-bit data deadband control

Description The function compares the input value at input **s3** with a deadband whose lower limit is specified at input **s1** and whose upper limit is specified at **s2**. The result of the function is returned at output **d** as follows:



- If the input value at input **s3** < **s1**, the lower limit at input **s1** is subtracted from the input value at **s3**, and the result is stored as the output value at **d**.
- If the input value at input **s3** > **s2**, the upper limit at input **s2** is subtracted from the input value at **s3**, and the result is stored as the output value at **d**.
- If the input value at **s2** ≥ **s3** ≥ **s1**, 0 is returned as the output value at **d**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F288_DBAND **(see page 1323)**

Data types

Variable	Data type	Function
s1	ANY32	the area where the lower limit is stored or the lower limit data
s2		the area where the upper limit is stored or the upper limit data
s3		the area where the input value is stored or the input value data
d		the area where the output value data is stored

Operands

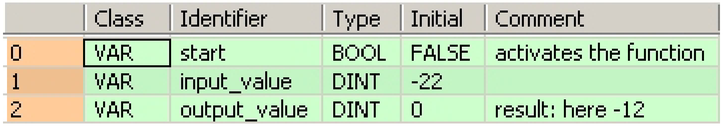
For	Relay				T/C		Register			Constant
s1, s2, s3	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the value at **s1** > **s2**.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	▪ the input value at **s3** is 0.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

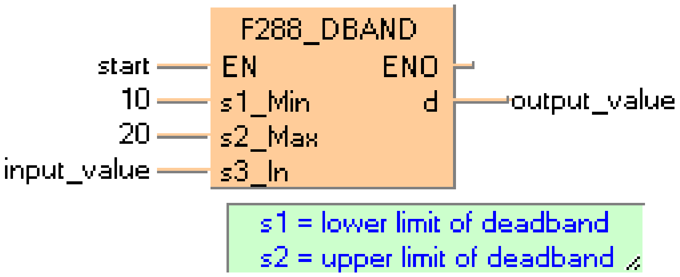


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	DINT	-22	
2	VAR	output_value	DINT	0	result: here -12

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out. The constant -10 (lower limit of the deadband) and 20 (upper limit of the deadband) are assigned to inputs s1 and s2. However, you can declare variables in the POU header and write them in the function in the body at the inputs.

LD



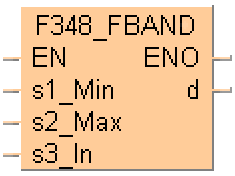
s1 = lower limit of deadband
s2 = upper limit of deadband

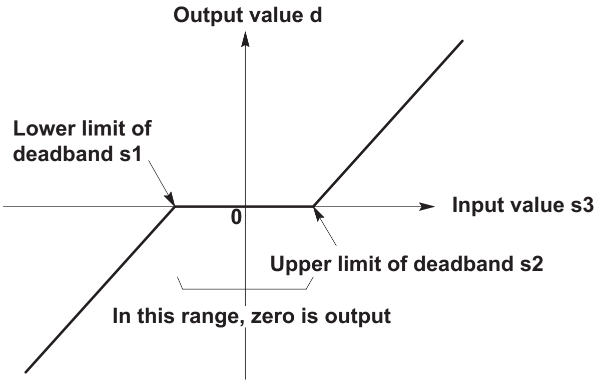
ST
```
IF start THEN
    F288_DBAND( -10, 20, input_value, output_value);
END_IF;    (* 10=lower limit of deadband, 20=upper limit of deadband *)
```

F348_FBAND

Floating point data deadband control

Description The function compares the input value at input **s3_In** with a deadband whose lower limit is specified at input **s1_Min** and whose upper limit is specified at **s2_Max**. The result of the function is returned at output **d** as follows:




Output value d
Lower limit of deadband s1
0
Input value s3
Upper limit of deadband s2
In this range, zero is output

Comparison between s1 and s2	**Flag**		
	R900A (> flag)	R900B (= flag)	R900C (< flag)
s1 < s2	off	off	on
s1 ≤ s3 and s2 ≤ s1	off	on	off
s3 < s1	on	off	off

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F348_FBAND **(see page 1324)**

Data types

Variable	Data type	Function
s1_Min	REAL	the area where the lower limit is stored or the lower limit data
s2_Max	REAL	the area where the upper limit is stored or the upper limit data
s3_In	REAL	the area where the input value is stored or the input value data
d	REAL	the area where the output value data is stored

Operands

For	Relay				T/C		Register			Constant
s1_Min, s2_Max, s3_In	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.

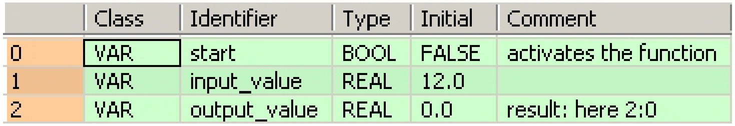
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the values at inputs **s1_Min**, **s2_Max**, and **s3_In** are not REAL numbers or the value at **s1_Min** > **s2_Max**.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	▪ the result is 0.
R9009	%MX0.900.9	for an instant	▪ the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

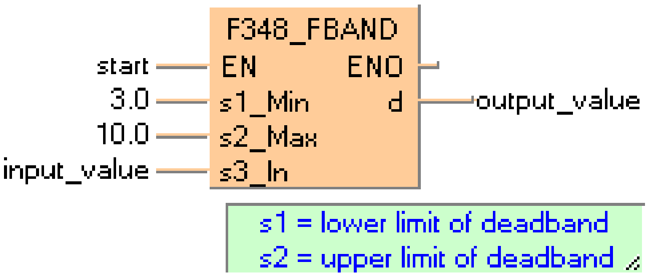


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	12.0	
2	VAR	output_value	REAL	0.0	result: here 2:0

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body The constants 3.0 and 10.0 are assigned to the inputs **s1_Min** (lower limit of the deadband) and **s2_Max** (upper limit of the deadband). However, you can declare two variables in the POU header and write them in the function in the body at the inputs. When the variable **start** is set to TRUE, the function is carried out. Since the **input_value** = 12.0 is larger than the value of the upper limit of the deadband at **s2_Max**, the **output_value** = 12.0 -10.0 = 2.0.

LD

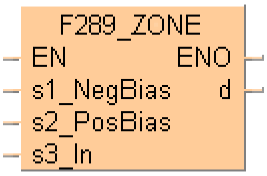


```
ST IF start THEN
       F348_FBAND( s1_Min:= 3.0 ,
           s2_Max:= 10.0 ,
           s3_In:= input_value ,
           d=> output_value )
   END_IF;      (* 3.0=lower limit of deadband, 10.0=upper limit *)
```

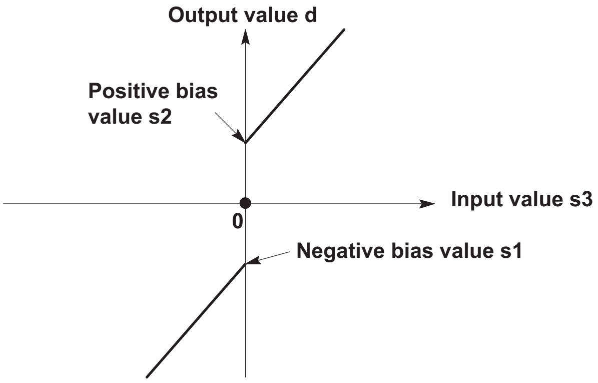
F289_ZONE

16-bit data zone control

Description The function adds an offset value to the input value at input **s3**. The offset values for the negative and positive areas are entered at inputs **s1** and **s2**. The result of the function is returned at output **d** as follows:



- If the input value at input **s3** < 0, the negative offset value at input **s1** is added to the input value at **s3**, and the result is stored as the output value at **d**.
- If the input value at input **s3** = 0, 0 is returned at the output value to output **d**.
- If the input value at input **s3 >** 0, the positive offset value at input **s2** is added to the input value at **s3**, and the result is stored as the output value at **d**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F289_ZONE **(see page 1323)**

Data types

Variable	Data type	Function
s1	ANY16	area where negative bias value is stored or negative bias value data
s2		area where positive bias value is stored or positive bias value data
s3		area where input value is stored or input value data
d		area where output value is stored

Operands

For	Relay				T/C		Register			Constant
s1, s2, s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

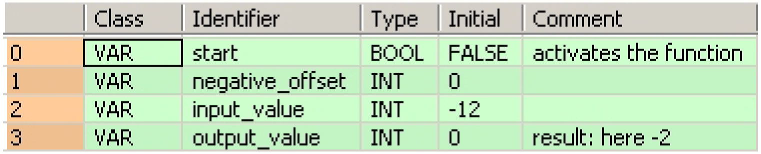
No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculation results in an overflow or an underflow of output **d**.
R9009	%MX0.900.9	for an instant	▪ the input value **s3** is 0.

Example

In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

All input and output variables used for programming this function have been declared in the POU header.



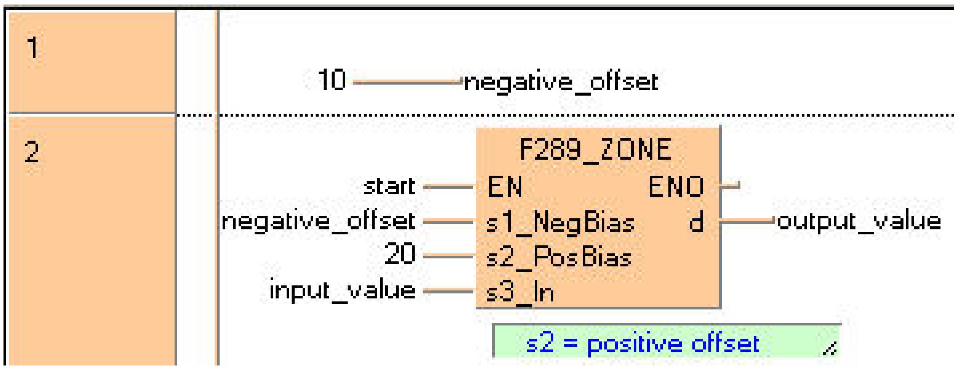
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	negative_offset	INT	0	
2	VAR	input_value	INT	-12	
3	VAR	output_value	INT	0	result: here -2

In this example the input variables **input_value** and **negative_offset** are declared. However, you can write constants directly at the input contact of the function instead.

Body

When the variable **start** is set to TRUE, the function is carried out. It adds the corresponding negative offset value = 10 to the negative **input_value** = -12. However, you can declare a variable in the POU header and assign it to the function's input in the body.

LD



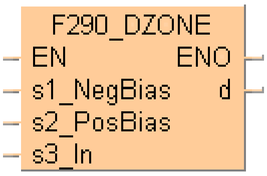
ST

```
IF start THEN
    F289_ZONE( negative_offset, 20, input_value, output_value);
END_IF;      (*negative_offset=neg. offset, 20=pos. offset *)
```

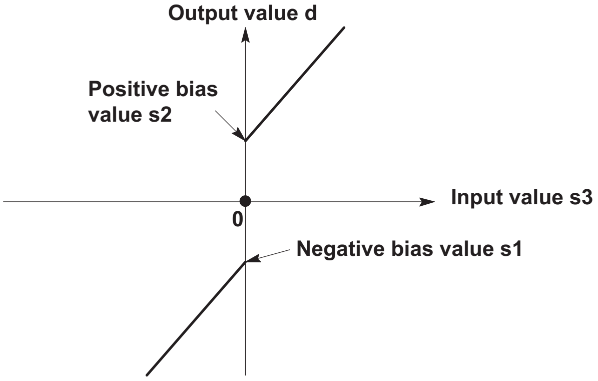
F290_DZONE

32-bit data (double word data) zone control

Description The function adds an offset value to the input value at input **s3**. The offset value for the negative and positive area are entered at inputs **s1** and **s2**. The result of the function is returned at output **d** as follows:



- If the input value at input **s3** < 0, the negative offset value at input **s1** is added to the input value at **s3**, and the result is stored as the output value at **d**.
- If the input value at input **s3** = 0, 0 is returned at the output value to output **d**.
- If the input value at input **s3** > 0, the positive offset value at input **s2** is added to the input value at **s3**, and the result is stored as the output value at **d**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F290_DZONE **(see page 1323)**

Data types

Variable	Data type	Function
s1	ANY32	area where negative bias value is stored or negative bias value data
s2		area where positive bias value is stored or positive bias value data
s3		area where input value is stored or input value data
d		area where output value is stored

Operands

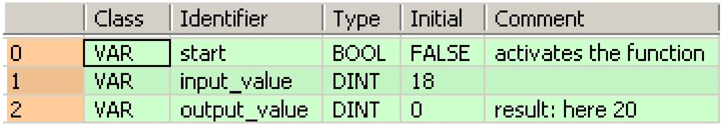
For	Relay				T/C		Register			Constant
s1, s2, s3	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the calculation results in an overflow or an underflow of output **d**.
R9009	%MX0.900.9	for an instant	▪ the input value **s3** is 0.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

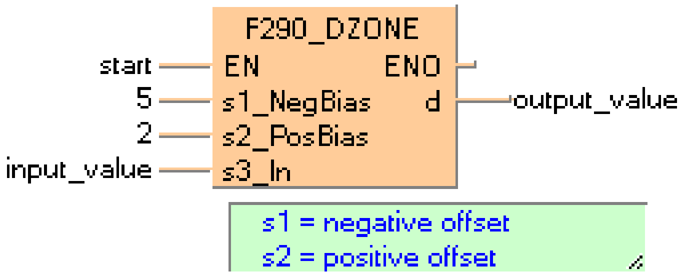


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	DINT	18	
2	VAR	output_value	DINT	0	result: here 20

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out. It adds the corresponding positive offset value = 2 to the positive input value = 18. The constants 5 (negative offset) and 2 (positive offset) are assigned to inputs s1 and s2 respectively. However, you can declare variables in the POU header and write them in the function in the body at the inputs.

LD

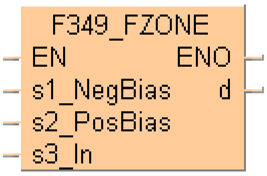


ST
```
IF start THEN
     F290_DZONE( s1_NegBias:= 5,
          s2_PosBias:= 2,
          s3_In:= input_value,
          d=> output_value);
END_IF;       (*5=neg. offset, 2=pos. offset *)
```

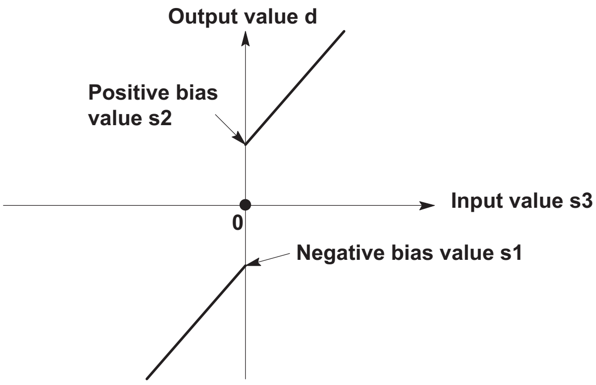
F349_FZONE

Floating point data zone control

Description The function adds an offset value to the input value at input **s3**. The offset value for the negative and positive area are entered at inputs **s1** and **s2**. The result of the function is returned at output **d** as follows:



- If the input value at input **s3** < 0.0, the negative offset value at input **s1** is added to the input value at **s3**, and the result is stored as the output value at **d**.
- If the input value at input **s3** = 0.0, 0.0 is returned as the output value to output **d**.
- If the input value at input **s3** > 0.0, the positive offset value at input **s2** is added to the input value at **s3**, and the result is stored as the output value at **d**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F349_FZONE **(see page 1324)**

Data types

Variable	Data type	Function
s1	REAL	area where negative bias value is stored or negative bias value data
s2	REAL	area where positive bias value is stored or positive bias value data
s3	REAL	area where input value is stored or input value data
d	REAL	area where output value is stored

Operands

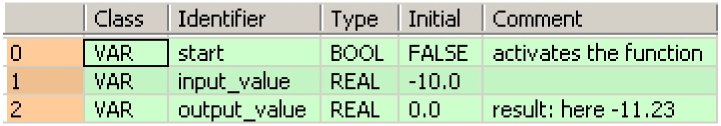
For	Relay				T/C		Register			Constant
s1, s2, s3	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the values at inputs s1, s2, and s3 are not REAL numbers.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	▪ the result is 0.
R9009	%MX0.900.9	for an instant	▪ the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

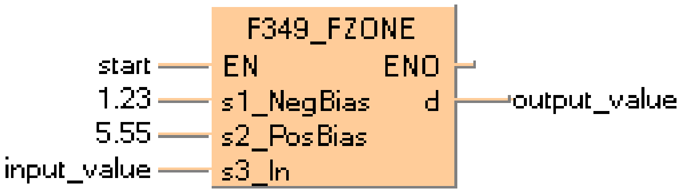


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	-10.0	
2	VAR	output_value	REAL	0.0	result: here -11.23

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body The constant -1.23 is assigned to input s1 (negative offset) and the constant 5.55 is assigned to input s2 (positive offset). However, you can declare two variables in the POU header and write them in the function in the body at the inputs. When the variable **start** is set to TRUE, the function is carried out. Since the **input_value** is negative (-10.0), the negative offset -1.23 is added to it. The result here is: **output_value** = -11.23.

LD

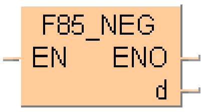


```
ST IF start THEN
       F349_FZONE( s1_NegBias:= -1.23 ,
           s2_PosBias:= 5.55 ,
           s3_In:= input_value ,
           d=> output_value );
   END_IF;        (*-1.23=neg. offset, 5.55=pos. offset *)
```

F85_NEG

16-bit data two's complement

Description Takes two's complement of 16-bit data specified by **d** if the trigger **EN** is in the ON-state. Two's complement of the original 16-bit data is stored in **d**.



Two’s complement is a number system used to express positive and negative numbers in binary format. In this system, the number becomes negative if the most significant bit (MSB) of data is 1. Two’s complement is obtained by inverting all bits and adding 1 to the inverted result.

This instruction is useful for inverting the sign of 16-bit data from positive to negative or from negative to positive.

Destination

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1
Decimal data	3			

start: ON

Destination

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
d	1 1 1 1	1 1 1 1	1 1 1 1	1 1 0 1
Decimal data	-3			

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F85_NEG **(see page 1326)**

Data types

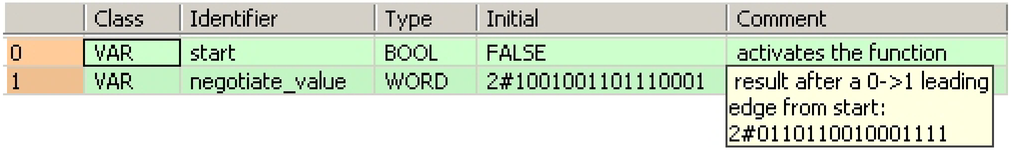
Variable	Data type	Function
d	ANY16	16-bit area for storing original data and its two's complement

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

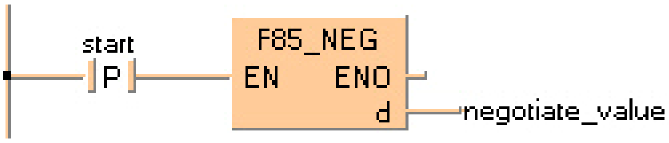
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	negotiate_value	WORD	2#1001001101110001	result after a 0->1 leading edge from start: 2#0110110010001111

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD

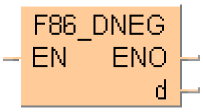


```
ST IF DF(start) THEN
       F85_NEG(negotiate_value);
   END_IF;
```

F86_DNEG

32-bit data two's complement

Description Takes two's complement of 32-bit data specified by **d** if the trigger **EN** is in the ON-state. Two's complement of the original 32-bit data is stored in **d**.



Two’s complement is a number system used to express positive and negative numbers in binary format. In this system, the number becomes negative if the most significant bit (MSB) of data is 1. Two’s complement is obtained by inverting all bits and adding 1 to the inverted result.

This instruction is useful for inverting the sign of 32-bit data from positive to negative or from negative to positive.

Destination	DT1				DT0			
Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0	15 · · 12	11 · · 8	7 · · 4	3 · · 0
Binary data	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 0 1
Decimal data	-3							

32-bit area

start: ON

Destination	DT1				DT0			
Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0	15 · · 12	11 · · 8	7 · · 4	3 · · 0
Binary data	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1
Decimal data	3							

32-bit area

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F86_DNEG **(see page 1326)**

Data types

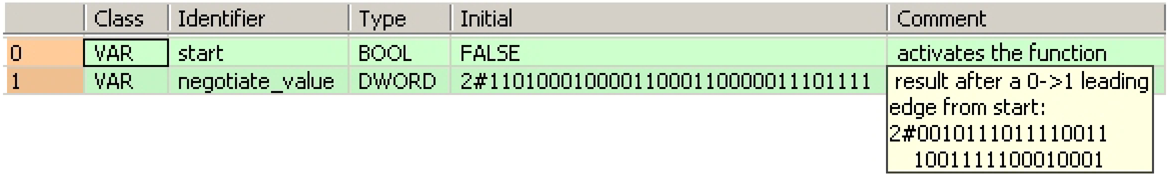
Variable	Data type	Function
d	ANY32	32-bit area for storing original data and its two's complement

Operands

For	Relay				T/C		Register			Constant
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

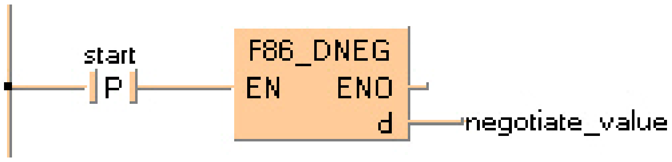
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	negotiate_value	DWORD	2#11010001000011000110000011101111	result after a 0->1 leading edge from start: 2#0010111011110011 1001111100010001

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD



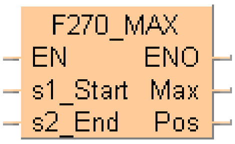
ST

```
IF DF(start) THEN
    F86_DNEG(negotiate_value);
END_IF;
```

F270_MAX

Maximum value search in 16-bit data table

Description The function searches for the maximum value and its position in a 16-bit data table.



Input **s1** specifies the starting area of the data table, and **s2** specifies the end. The maximum value is returned at output **max** and its position at output **pos**.

The position **pos** is relative to the position at the beginning of the data table to the first occurrence of the maximum value.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F270_MAX **(see page 1323)**

Data types

Variable	Data type	Function
s1	ANY16	starting area of data table
s2		ending area of data table
max	INT	specifies maximum value
pos	INT	position where maximum value was found

Operands

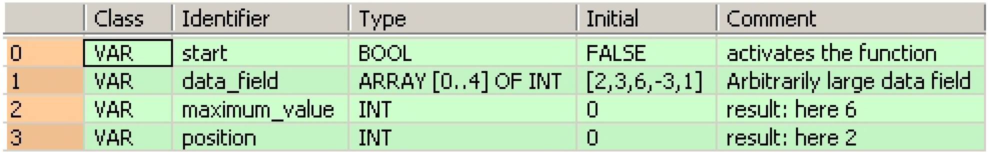
For	Relay				T/C		Register			Const.
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
max, pos	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the address of the variable at input **s1** > **s2**. ▪ the address areas of s1 and s2 are different.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

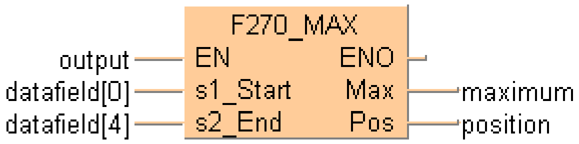
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF INT	[2,3,6,-3,1]	Arbitrarily large data field
2	VAR	maximum_value	INT	0	result: here 6
3	VAR	position	INT	0	result: here 2

Body When the variable **start** is set to TRUE, the function is carried out. It searches for the maximum value and its position in the **data_field**. The result is here: **maximum_value** = 6 and **position** = 2.

LD

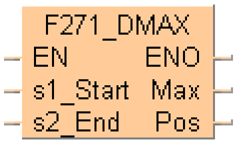


ST
```
IF start THEN
        F270_MAX( s1_Start:= data_field[0],
             s2_End:= data_field[4],
             Max=> maximum_value,
             Pos=> position);
END_IF;
```

F271_DMAX

Maximum value search in 32-bit data table

Description The function searches for the maximum value and its position in a 32-bit data table.



Input **s1** specifies the starting area of the data table, and **s2** specifies the end. The maximum value is returned at output **max** and its position at output **pos**.

The position **pos** is relative to the position at the beginning of the data table to the first occurrence of the maximum value.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F271_DMAX **(see page** 1323**)**

Data types

Variable	Data type	Function
s1	DINT, DWORD	starting area of data table
s2	DINT, DWORD	ending area of data table
max	DINT	specifies maximum value
pos	WORD	position where maximum value was found

Operands

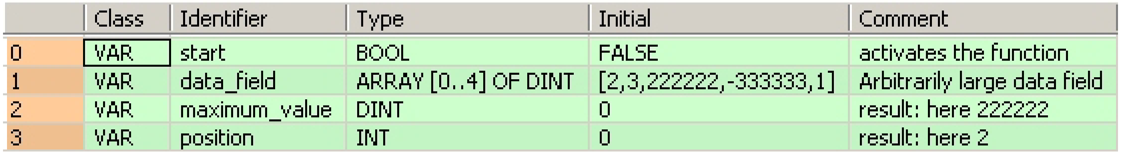
For	Relay				T/C		Register			Const.
s1, s2	DWX	DWY	DWF	DWL	DSV	DEV	DDT	DLD	DFL	-
max	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
pos	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the address of the variable at input **s1** > **s2**. ▪ the address areas of s1 and s2 are different.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

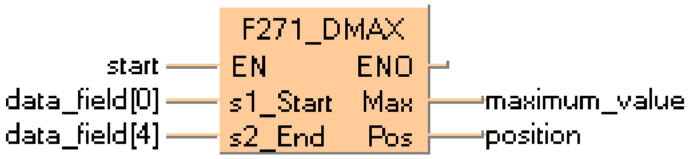
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF DINT	[2,3,222222,-333333,1]	Arbitrarily large data field
2	VAR	maximum_value	DINT	0	result: here 222222
3	VAR	position	INT	0	result: here 2

Body When the variable **start** is set to TRUE, the function is carried out. It searches for the maximum value and its position in the **data_field**. The result is here: **maximum_value** = 222222 and **position** = 2.

LD

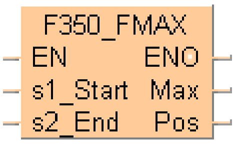


ST
```
IF start THEN
     F271_DMAX( s1_Start:= data_field[0],
         s2_End:= data_field[4],
         Max=> maximum_value,
         Pos=> position);
END_IF;
```

F350_FMAX

Maximum value search in real number data table (floating point data)

Description The function searches for the maximum value and its position in a floating point data table.



Input **s1** specifies the starting area of the data table, and **s2** specifies the end. The maximum value is returned at output **max** and its position at output **pos**.

The address of the maximum value at output **pos** is relative to the beginning address in the data table as specified at input **s1**.

If more than one maximum value is found, the first one found beginning from the starting address specified at **s1** is stored in **d**.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F350_FMAX **(see page 1325)**

Data types

Variable	Data type	Function
s1	REAL	starting area of data table
s2	REAL	ending area of data table
max	REAL	specifies maximum value
pos	INT	position where maximum value was found

Operands

For	Relay				T/C		Register			Const.
s1, s2	DWX	DWY	DWF	DWL	DSV	DEV	DDT	DLD	DFL	-
max	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
pos	-	WY	WR	WL	SV	EV	DT	LD	FL	-

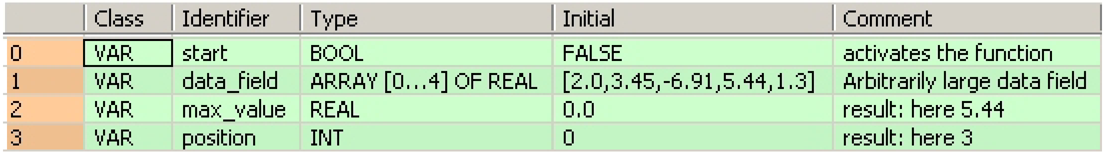
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the addresses of variables at inputs s1 > s2. ▪ the address areas are different. ▪ the floating point values exceed the processing range.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

All input and output variables used for programming this function have been declared in the POU header.

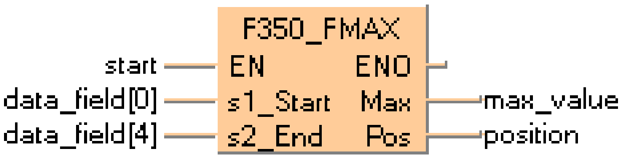


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0...4] OF REAL	[2.0,3.45,-6.91,5.44,1.3]	Arbitrarily large data field
2	VAR	max_value	REAL	0.0	result: here 5.44
3	VAR	position	INT	0	result: here 3

Body

When the variable **start** is set to TRUE, the function is carried out. It then searches the **data_field** for a maximum value and its position. The result here is: **max_value** = 5.44 and **position** = 3.

LD



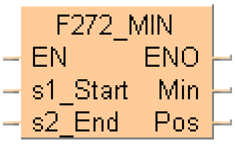
ST

```
IF start THEN
    F350_FMAX( s1_Start:= data_field[0],
        s2_End:= data_field[4],
        Max=> max_value,
        Pos=> position);
END_IF;
```

F272_MIN

Minimum value search in 16-bit data table

Description The function searches for the minimum value and its position in a 16-bit data table.



Input **s1** specifies the starting area of the data table, and **s2** specifies the end. The minimum value is returned at output **min** and its position at output **pos**.

The position **pos** is relative to the position at the beginning of the data table to the first occurrence of the minimum value.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F272_MIN **(see page 1323)**

Data types

Variable	Data type	Function
s1	ANY16	starting area of data table
s2		ending area of data table
min	INT	specifies minimum value
pos	INT	position where minimum value was found

Operands

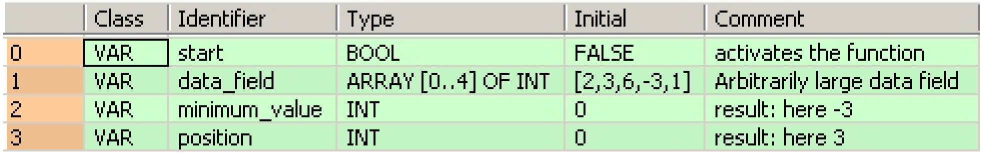
For	Relay				T/C		Register			Const.
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
min, pos	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the address of the variable at input **s1** > **s2**. ▪ the address areas of s1 and s2 are different.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

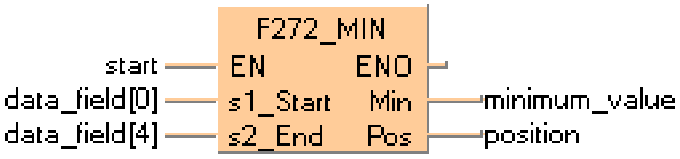
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF INT	[2,3,6,-3,1]	Arbitrarily large data field
2	VAR	minimum_value	INT	0	result: here -3
3	VAR	position	INT	0	result: here 3

Body When the variable **start** is set to TRUE, the function is carried out. It searches for the minimum value and its position in the **data_field**. The result is here: **minimum_value** = -3 and **position** = 3.

LD

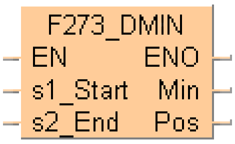


```
ST IF start THEN
        F272_MIN( s1_Start:= data_field[0],
            s2_End:= data_field[4],
            Min=> minimum_value,
            Pos=> position);
   END_IF;
```

F273_DMIN

Minimum value search in 32-bit data table

Description The function searches for the minimum value and its position in a 32-bit data table.



Input **s1** specifies the starting area of the data table, and **s2** specifies the end. The minimum value is returned at output **min** and its position at output **pos**.

The position **pos** is relative to the position at the beginning of the data table to the first occurrence of the minimum value.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F273_DMIN **(see page 1323)**

Data types

Variable	Data type	Function
s1	ANY32	starting area of data table
s2		ending area of data table
min	DINT	specifies minimum value
pos	INT	position where minimum value was found

Operands

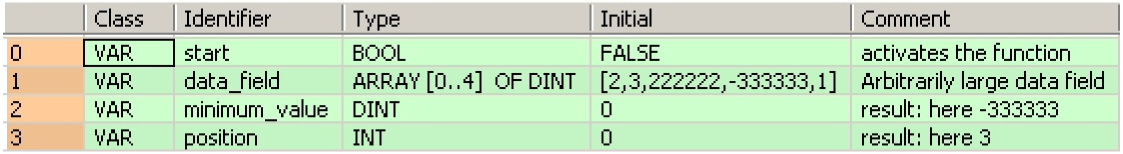
For	Relay				T/C		Register			Const.
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
min	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
pos	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the address of the variable at input **s1** > **s2**. ▪ the address areas of **s1** and **s2** are different.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

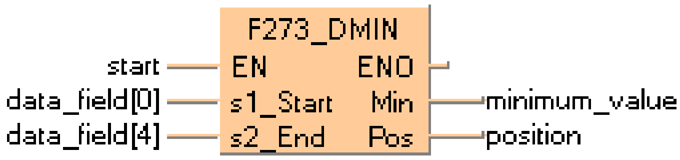
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF DINT	[2,3,222222,-333333,1]	Arbitrarily large data field
2	VAR	minimum_value	DINT	0	result: here -333333
3	VAR	position	INT	0	result: here 3

Body When the variable **start** is set to TRUE, the function is carried out. It searches for the minimum value and its position in the **data_field**. The result is here: **minimum_value** = -333333 and **position** = 3.

LD

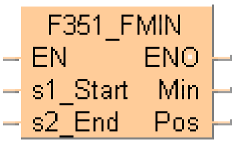


ST
```
IF start THEN
    F273_DMIN( s1_Start:= data_field[0],
        s2_End:= data_field[4],
        Min=> minimum_value,
        Pos=> position);
END_IF;
```

F351_FMIN

Minimum value search in real number data table (floating point data)

Description The function searches for the minimum value and its position in a floating point data table.



Input **s1** specifies the starting area of the data table, and **s2** specifies the end. The minimum value is returned at output **min** and its position at output **pos**.

The address of the minimum value at output **pos** is relative to the beginning address in the data table as specified at input **s1**.

If more than one minimum value is found, the first one found beginning from the starting address specified at **s1** is stored in **d**.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F351_FMIN **(see page 1325)**

Data types

Variable	Data type	Function
s1	REAL	starting area of data table
s2	REAL	ending area of data table
min	REAL	specifies minimum value
pos	INT	position where minimum value was found

Operands

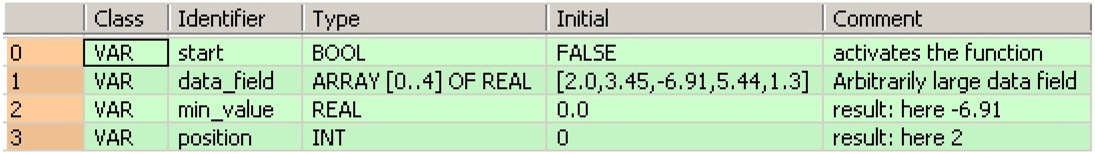
For	Relay				T/C		Register			Const.
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
min	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
pos	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the addresses of variables at inputs s1 > s2. ▪ the address areas are different. ▪ the floating point values exceed the processing range.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

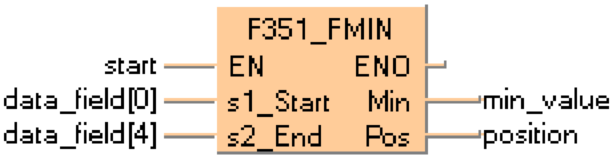
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF REAL	[2.0,3.45,-6.91,5.44,1.3]	Arbitrarily large data field
2	VAR	min_value	REAL	0.0	result: here -6.91
3	VAR	position	INT	0	result: here 2

Body When the variable **start** is set to TRUE, the function is carried out. It then searches the **data_field** for a minimum value and its position. The result here is: **min_value** = 6.91 and **position** = 2.

LD

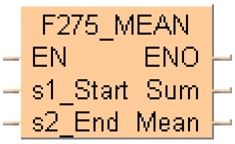


ST
```
IF start THEN
    F351_FMIN( s1_Start:= data_field[0],
        s2_End:= data_field[4],
        Min=> min_value ,
        Pos=> position );
END_IF;
```

F275_MEAN

Total and mean numbers calculation in 16-bit data table

Description This function calculates the sum and the arithmetic mean of numbers (both with +/- signs) in the specified 16-bit data table.



Input **s1_Start** specifies the starting area of the data table, and **s2_End** specifies the end. The sum of all elements in the data table is returned at output **Sum** and the arithmetic mean of all elements in the data table is returned at output **Mean**. The arithmetic mean is rounded off if it is not a whole number.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of F275_MEAN (see page 1323)**

Data types

Variable	Data type	Function
s1_Start	ANY16	starting area of data table
s2_End		ending area of data table
Mean	INT	mean of all elements in data table area specified
Sum	DINT	sum of all elements in data table area specified

Operands

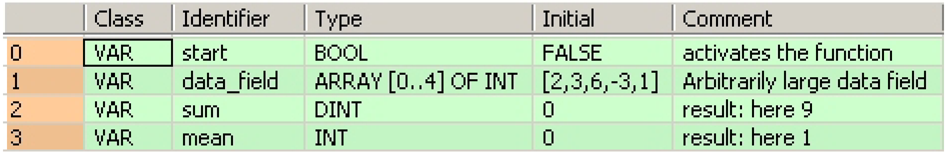
For	Relay				T/C		Register			Const.
s1_Start, s2_End	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
Mean	-	WY	WR	WL	SV	EV	DT	LD	FL	-
Sum	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the address of the variable at input **s1_Start** > **s2_End**. ▪ the address areas are different.
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	▪ the total value range overflows or underflows the 16-bit range.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

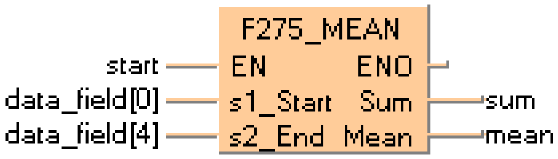
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF INT	[2,3,6,-3,1]	Arbitrarily large data field
2	VAR	sum	DINT	0	result: here 9
3	VAR	mean	INT	0	result: here 1

Body When the variable **output** is set to TRUE, the function F275_MEAN is carried out. The function calculates the sum of all elements of the data table (sum = 4 + 3 + 8 + (-2) + 1 + (-6) = 8) and writes the result (in this case 8) to the variable **sum.** Additionally, the function calculates the arithmetic mean of all elements of the data table (mean = sum/6 = (4 + 3 + 8 + (-2) + 1 + (-6)) / 6 = 1.333) and writes the roanded-off number (in this case 1) to the variable **mean**.

LD

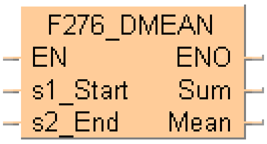


ST
```
IF start THEN
     F275_MEAN( s1_Start:= data_field[0],
         s2_End:= data_field[4],
         Sum=> sum,
         Mean=> mean);
END_IF;
```

F276_DMEAN

Total and mean numbers calculation in 32-bit data table

Description This function calculates the sum and the arithmetic mean of numbers (both with +/- signs) in the specified 32-bit data table.



Input **s1** specifies the starting area of the data table, and **s2** specifies the end. The sum of all elements in the data table are returned at output **sum** and the arithmetic mean of all elements in the data table are returned at output **mean**. The arithmetic mean is rounded off if it is not already a whole number.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F276_DMEAN **(see page 1323)**

Data types

Variable	Data type	Function
s1	ANY32	starting area of data table
s2		ending area of data table
mean	DINT	mean of all elements in data table area specified
sum	ARRAY [0..1] of DINT	sum of all elements in data table area specified

Operands

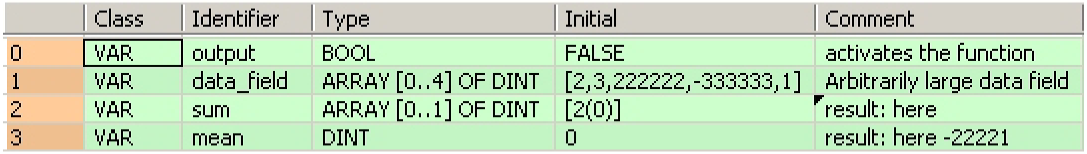
For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
mean, sum	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the address of the variable at input **s1** > **s2**. ▪ the address areas are different.
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	▪ the total value range overflows or underflows the 32-bit range.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

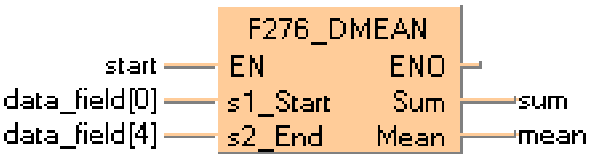
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	output	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF DINT	[2,3,222222,-333333,1]	Arbitrarily large data field
2	VAR	sum	ARRAY [0..1] OF DINT	[2(0)]	result: here
3	VAR	mean	DINT	0	result: here -22221

Body When the variable **start** is set to TRUE, the function is carried out. The function calculates the sum of all elements of ARRAY **data_field** (sum = 2 + 3 + 222222 + (-333333) + 1 = -111105) and transfers the result to the variable **sum**. In addition, the function calculates the mean (mean = sum/5 = -111105/5 = -22221) and transfers the result to the variable **mean**.

LD

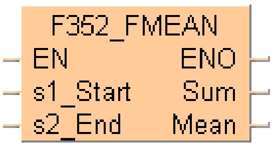


ST
```
IF start THEN
    F276_DMEAN( s1_Start:= data_field[0],
        s2_End:= data_field[4],
        Sum=> sum,
        Mean=> mean);
END_IF;
```

F352_FMEAN

Total and mean numbers calculation in floating point data table

Description This function calculates the sum and the arithmetic mean (both with +/- signs) of floating point values in the specified 32-bit data table.



Input **s1** specifies the starting area of the data table, and **s2** specifies the end. The sum of all elements in the data table are returned at output **sum**, and the arithmetic mean of all elements in the data table are returned at output **mean**.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of F352_FMEAN (see page 1325)**

Data types

Variable	Data type	Function
s1	REAL	starting area of data table
s2	REAL	ending area of data table
mean	REAL	mean of all elements in data table area specified
sum	REAL	sum of all elements in data table area specified

Operands

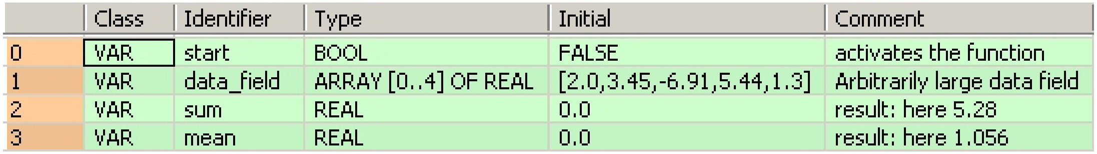
For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
mean, sum	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the addresses of variables at inputs s1 > s2. ▪ the address areas are different. ▪ the floating point values exceed the processing range.
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	▪ the result leads to an overflow or an underflow.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

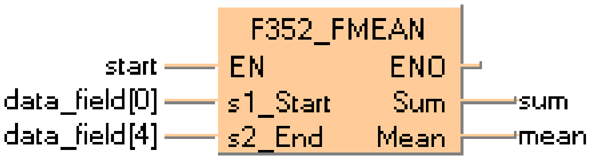
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF REAL	[2.0,3.45,-6.91,5.44,1.3]	Arbitrarily large data field
2	VAR	sum	REAL	0.0	result: here 5.28
3	VAR	mean	REAL	0.0	result: here 1.056

Body When the variable **start** is set to TRUE, the function is carried out. It calculates the **sum** = 2.0 + 3.45 + (-6.91) + 5.44 + 1.3 = 5.28 and the **mean = Sum**/5 = 5.28/5 = 1.056 of the elements of the **data_field**.

LD

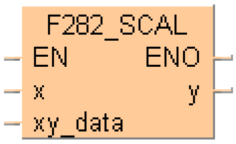


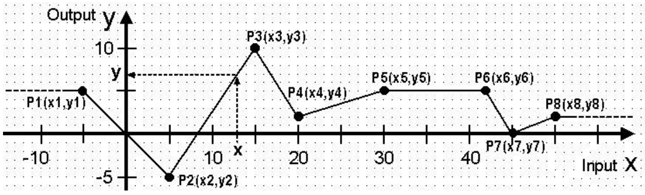
ST
```
IF start THEN
    F352_FMEAN( s1_Start:= data_field[0] ,
        s2_End:= data_field[4] ,
        Sum=> sum ,
        Mean=> mean );
END_IF;
```

F282_SCAL

Linearization of 16-bit data

Description The function renders the value **y** at position **x** by performing a linear interpolation based on the neighboring reference points $\mathbf{Pw}_{(xw,\ yw)}$ and $\mathbf{Pw+1}_{(xw+1,\ yw+1)}$. In this example, **w** is the nearest reference point whose **x** value is smaller than the input value **x**, i.e. the function connects the individual reference points in series and renders the output value **y** based on the input value **x**.





The function can be used for:

- linearizing measured values, e.g. with non-linear sensors
- rendering a heater's flow temperature **y** in relation to the outside temperature **x**
- etc.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F282_SCAL **(see page 1323)**

Data types

Variable	Data type	Function
x	INT	Input value **x**
xy_data	DUT	The first element of an DUT-type variable that contains the **xy** value pairs.
y	INT	Output value **y**
EN	BOOL	Activation of the function (when EN = TRUE, the function is executed during each PLC cycle)
ENO	BOOL	ENO is set to TRUE as soon the function is executed. Helpful when cascading function blocks with EN functions.

Operands

For	Relay				T/C		Register			Constant
x	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
y	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the number of reference points is not between 2 ... 100, or the **x** values are not in ascending order (**x1** < **x2** < **x3** < ...).
R9008	%MX0.900.8	for an instant	

■ Limitations of the output value y:

If the input value **x** is smaller than the **x**-coordinate of the first reference point (**P1**: **x** < **x1**), the output **y** is set to the first reference point's **y**-coordinate (output **y** = **y1**, horizontal dashed line in the graph's upper left corner).

If the input value **x** is greater than the x-coordinate of the last reference point (**P8**: **x** > **x8**), the output **y** is set to the last reference point's **y**-coordinate (output **y** = **y8**, horizontal dashed line in the graphic's upper right corner).

■ DUT for the xy value pairs (reference points P1, P2, ...):

The reference points (**P1**, **P2**, ...) are copied to the function via an DUT-type variable that contains the number of reference points and the **xy** value pairs (number; **x1**, **x2**, ...; **y1**, **y2**; ...).

Structure of the DUT:

1. Entry: Variable of the data type INT that contains the number of reference points.The number of reference points (**xy** value pairs) can be set anywhere between 2 ... 100. In the graph, eight reference points (**P1** ... **P8**) are used.
2. Entry: Variable of the data type ARRAY [0..z] OF INT that contains the **x** values. Here **z** represents the place marker for the number of reference points (see entry 1).
3. Entry: Variable of the data type ARRAY [0..z] OF INT that contains the **y** values. Here **z** represents the place marker for the number of reference points (see entry 1).

■ Important information:

x values

The x values have to be entered in ascending order (**x1** < **x2** < **x3** < ...). If the x values are the same (e.g. **x2** = **x3** = **x4**) the reference points P2(**x2**,**y2**) and P3(**x3**,**y3)** are ignored.

Overflow of the function:

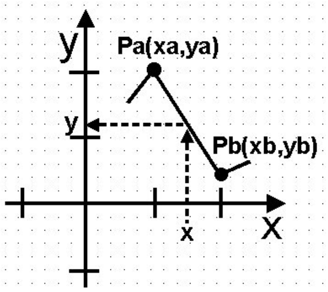
In order to avoid an overflow in the calculation, neighboring reference points must fulfill the following conditions:

$|ya - yb| < 32767$

$|x - xb| < 32767$

$|(ya - yb)*(x - xb)| < 32767$

$|xa - xb| < 32767$

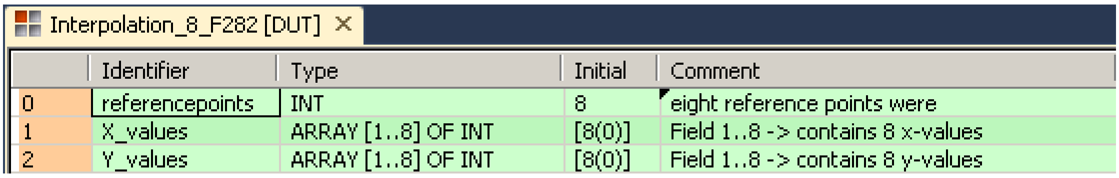


Accuracy of the calculation:

This function can only process whole numbers. Numbers that follow the decimal point are cut out when calculating the value **y**. For example, if at the position **x**, **y** = 511,13, the function returns the value 511.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

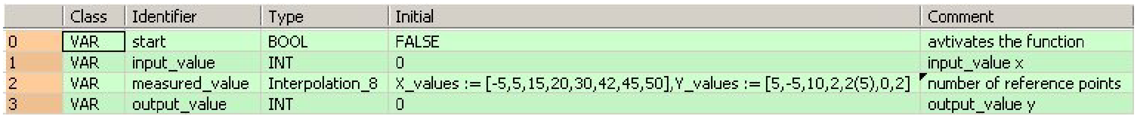
DUT In the DUT Pool the number of reference points and the xy value pairs are declared.



Interpolation_8_F282 [DUT]

	Identifier	Type	Initial	Comment
0	referencepoints	INT	8	eight reference points were
1	X_values	ARRAY [1..8] OF INT	[8(0)]	Field 1..8 -> contains 8 x-values
2	Y_values	ARRAY [1..8] OF INT	[8(0)]	Field 1..8 -> contains 8 y-values

POU header In the POU header, all input and output variables are declared that are used for programming this function.

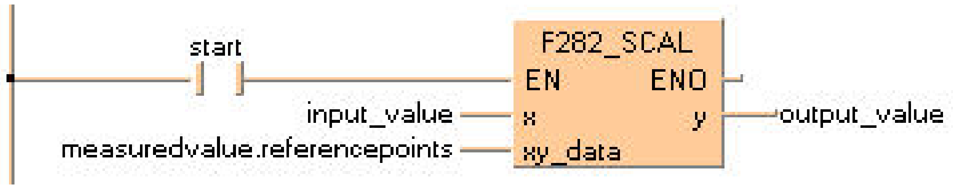


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	avtivates the function
1	VAR	input_value	INT	0	input_value x
2	VAR	measured_value	Interpolation_8	X_values := [-5,5,15,20,30,42,45,50],Y_values := [5,-5,10,2,2(5),0,2]	number of reference points
3	VAR	output_value	INT	0	output_value y

Here the input variable **measured_value** was declared, corresponding to the type of the DUT defined above. Assigning the x values and y values was done in the POU header. However, you can change the x values and y values in the body by assigning a value to the variable, e.g. **Measuredvalues.X_Values[1]** for x**.**

Body When the variable **start** is set to TRUE, the function is carried out. For the input value at position x, the output value y is calculated via linear interpolation of the neighboring reference points stored in the variable **measured_value**.

LD

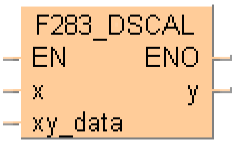


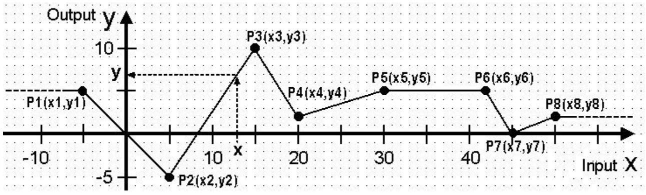
ST
```
IF start THEN
    F282_SCAL(input_value, measured_value.referencepoints, output_value);
END_IF;
```

F283_DSCAL

Linearization of 32-bit data

Description The function renders the value **y** at position **x** by performing a linear interpolation based on the neighboring reference points **Pw**(**xw**, **yw**) and **Pw**+1(**xw+1**, **yw+1**). In this example, w is the nearest reference point whose **x** value is smaller than the input value x, i.e. the function connects the individual reference points in series and renders the output value **y** based on the input value **s**.





This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

The function can be used for:

- linearizing measured values, e.g. with non-linear sensors
- rendering a heater's flow temperature y in relation to the outside temperature **x**
- etc.

PLC types **Availability of** F283_DSCAL **(see page 1323)**

Data types

Variable	Data type	Function
x	DINT	Input value **x**
xy_data	DUT	The first element of a DUT-type variable that contains the xy value pairs.
y	DINT	Output value **y**
EN	BOOL	Activation of the function (when EN = TRUE, the function is executed during each PLC cycle)
ENO	BOOL	ENO is set to TRUE as soon the function is executed. Helpful when cascading function blocks with EN functions.

Operands

For	Relay				T/C		Register			Constant
x	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
y	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the number of reference points is not between 2 ... 100, or the x values are not in ascending order (x1 < x2 < x3 < ...).
R9008	%MX0.900.8	for an instant	

■ Limitations of the output value y:

If the input value **x** is smaller than the x-coordinate of the first reference point (**P1**: **x** < **x1**), the output **y** is set to the first reference point's **y**-coordinate (output **y** = **y1**, horizontal dashed line in the graph's upper left corner).

If the input value **x** is greater than the x-coordinate of the last reference point (**P8**: **x** > **x8**), the output **y** is set to the last reference point's **y**-coordinate (output **y** = **y8**, horizontal dashed line in the graphic's upper right corner).

■ DUT for the xy value pairs (reference points P1, P2, ...):

The reference points (P1, P2, ...) are copied to the function via a DUT-type variable that contains the number of reference points and the xy value pairs (number; **x1**, **x2**, ...; **y1**, y**2**; ...).

Structure of the DUT:

1. Entry: Variable of the data type INT that contains the number of reference points.

The number of reference points (xy value pairs) can be anywhere between 2 ... 100. In the graph, eight reference points (P1 ... P8) are used.

2. Entry: Variable of the data type ARRAY [0..z] OF DINT that contains the **x** values.

Here **z** represents the place marker for the number of reference points (see entry 1).

3. Entry: Variable of the data type ARRAY [0..z] OF DINT that contains the **y** values.

Here **z** represents the place marker for the number of reference points (see entry 1).

■ Important information:

x values

The **x** values have to be entered in an ascending order (**x1** < **x2** < **x3** < ...). If the **x** values are the same (e.g. **x2** = **x3** = **x4**) the reference points P2(**x2**,**y2**) and P3(**x3**,**y3**) are ignored.

Overflow of the function:

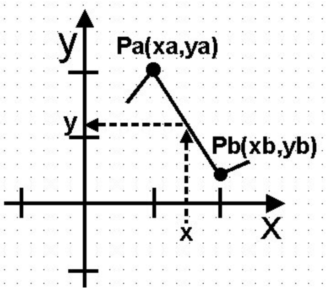
In order to avoid an overflow in the calculation, neighboring reference points must fulfill the following conditions:

$|ya - yb| < 2147483647$

$|x - xb| < 2147483647$

$|(ya - yb)*(x - xb)| < 2147483647$

$|xa - xb| < 2147483647$

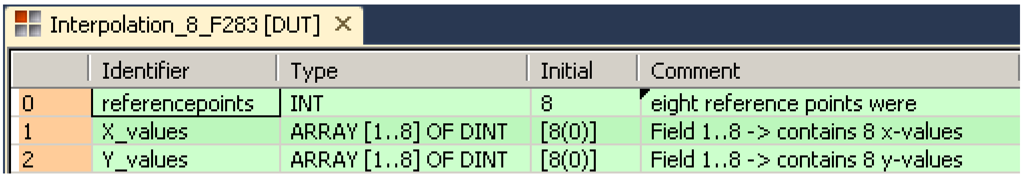


Accuracy of the calculation:

This function can only process whole numbers. Numbers that follow the decimal point are cut out when calculating the value **y**. For example, if at the position **x**, **y** = 511,13, the function returns the value 511.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

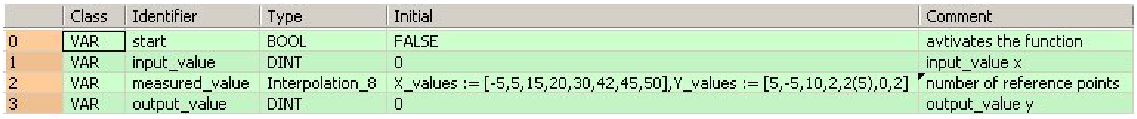
DUT In the DUT Pool, the number of reference points and the xy value pairs are declared.



Interpolation_8_F283 [DUT]

	Identifier	Type	Initial	Comment
0	referencepoints	INT	8	eight reference points were
1	X_values	ARRAY [1..8] OF DINT	[8(0)]	Field 1..8 -> contains 8 x-values
2	Y_values	ARRAY [1..8] OF DINT	[8(0)]	Field 1..8 -> contains 8 y-values

POU header All input and output variables used for programming this function have been declared in the POU header.

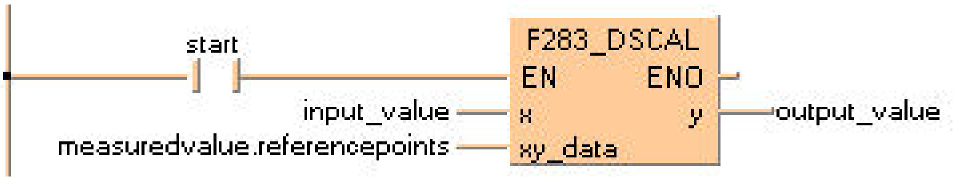


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	avtivates the function
1	VAR	input_value	DINT	0	input_value x
2	VAR	measured_value	Interpolation_8	X_values := [-5,5,15,20,30,42,45,50],Y_values := [5,-5,10,2,2(5),0,2]	number of reference points
3	VAR	output_value	DINT	0	output_value y

Here the input variable **measured_value** was declared, corresponding to the type of the DUT defined above. Assigning the x values and y values was done in the POU header. However, you can change the x values and y values in the body by assigning a value to the variable, e.g. **Measuredvalues.Y_Values[3]** for y3**.**

Body When the variable **start** is set to TRUE, the function is carried out. For the **input value** at position x, the output value y is calculated via linear interpolation between the neighboring reference points stored in the variable **measured value**.

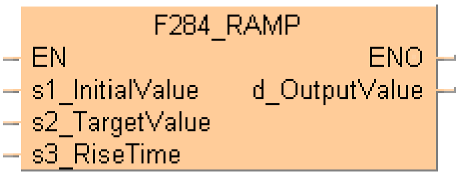
LD



ST
```
IF start THEN
    F283_DSCAL(input_value, measured_value.referencepoints, output_value);
END_IF;
```

F284_RAMP Inclination output of 16-bit data

Description Executes linear ramp output based on the parameters set.


```
                 F284_RAMP
EN                            ENO
s1_InitialValue     d_OutputValue
s2_TargetValue
s3_RiseTime
```

PLC types **Availability of** F284_RAMP **(see page 1323)**

Data types

Variable	Data type	Function
s1_InitialValue	INT	The initial value from which the output value increases or decreases after the trigger's rising edge has been detected by the system
s2_TargetValue	INT	The target value to which the output value increases or decreases
s3_RiseTime	INT	The time range in ms for the output value to increase or decrease from the initial value to the target value
d_OutputValue	INT	The output value

Operands

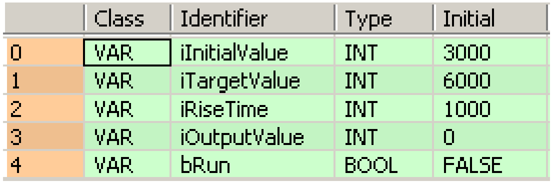
For	Relay				T/C		Register			Constant
s1, s2, s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the area specified using the index modifier exceeds the limit. ▪ the output time range specified by **s3_RiseTime** is smaller than 1 or larger than 30000.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



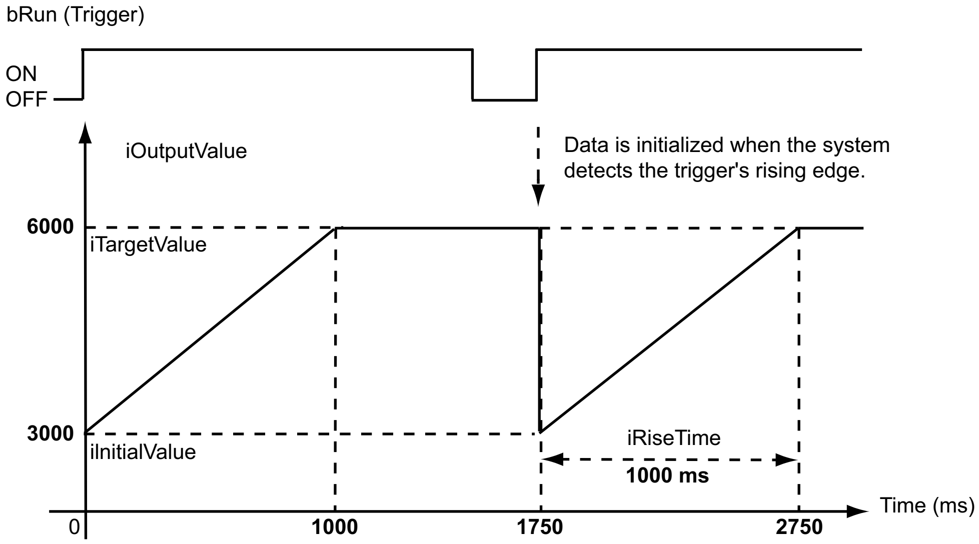
	Class	Identifier	Type	Initial
0	VAR	iInitialValue	INT	3000
1	VAR	iTargetValue	INT	6000
2	VAR	iRiseTime	INT	1000
3	VAR	iOutputValue	INT	0
4	VAR	bRun	BOOL	FALSE

In this example, the input variables **iInitialValue**, **iTargetValue** and **iRiseTime** are declared. However, you can write a constant directly at the input contact of the function instead. Additionally, the variable **bRun** is declared to start the ramp function and the variable **iOutputValue** is declared for storing the result.

Body When the variable **bRun** is switched to TRUE, the function is carried out and **iOutputValue** increases from 3000 (the initial value of **iInitialValue**) to 6000 (the initial value of **iTargetValue**) in 1000ms (according to the initial value of **iRiseTime**).

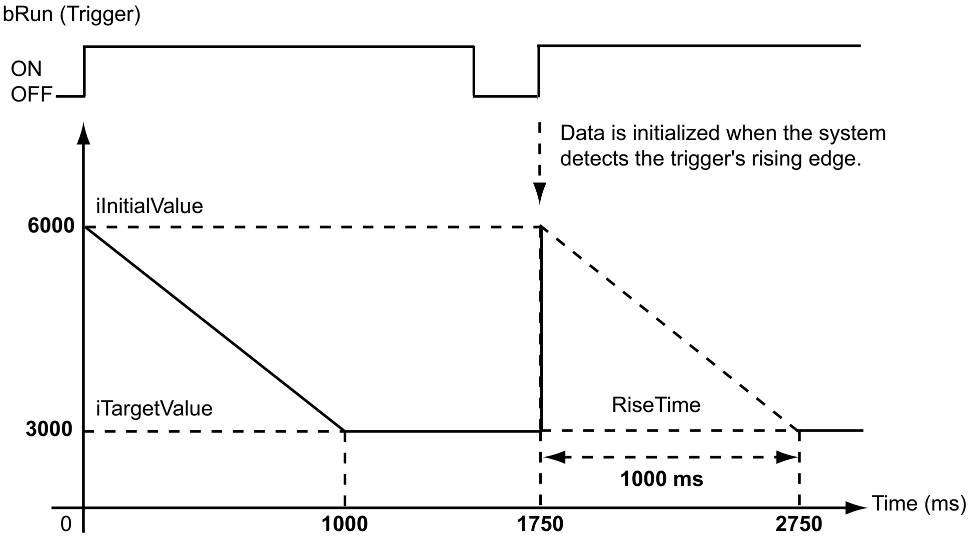
Time chart for increasing the output value:

Example values: **iInitialValue = 3000, iTargetValue = 6000, iRiseTime = 1000**

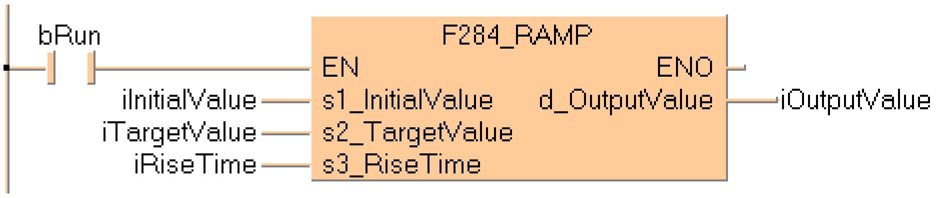


Time chart for decreasing the output value:

Example values: **iInitialValue = 6000, iTargetValue = 3000, iRiseTime = 1000**



LD



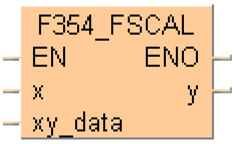
ST

```
IF bRun THEN
   F284_RAMP(iInitialValue, iTargetValue, iRiseTime, iOutputValue);
END_IF;
```

F354_FSCAL

Scaling of Real Number Data

Description This function performs scaling (linearization) of a real number data table and renders the output (Y) for an input value (X).



For a detailed description, refer to the instructions: F282_SCAL (see page 468) and F283_DSCAL (see page 471).

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F354_FSCAL **(see page 1325)**

Data types

Variable	Data type	Function
x	REAL	Input value (X)
xy_data	INT	First element of the data unit type table used for scaling
y	REAL	Output value (Y)

Operands

For	Relay				T/C		Register			Constant
x	WX	WY	WR	WL	SV	EV	DT	LD	FL	real
xy_data	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
y	-	WY	WR	WL	SV	EV	DT	LD	FL	-

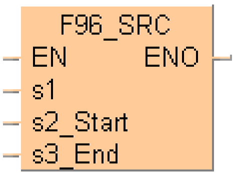
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the specified address using the index modifier exceeds a limit. ▪ a non-real number value is input into 'x'. ▪ the number of values (first element of the DUT) < 2 or > 99. ▪ a non- real number value is specified to be the real numerical value (xt, yt) specified in 'xy_data'. ▪ the linear table of 'xy_data' is not registered in ascending order of the x-sequence. ▪ the linear table of 'xy_data' exceeds the area. ▪ an overflow (operation is unable) occurs during the scaling operation.
R9008	%MX0.900.8	for an instant	

F96_SRC

Table data search (16-bit search)

Description Searches for the value that is the same as **s1** in the block of 16-bit areas specified by **s2** (starting area) through **s3** (ending area) if the trigger **EN** is in the ON-state.



When the search operation is performed, the search results are stored as follows:

- The number of data that is the same as **s1** is transferred to special data register DT9037 (or DT90037 for FP2/2SH, FP10/10S/10SH).
- The position the data is first found in, counting from the starting 16-bit area, is transferred to special data register DT9038 (or DT90038 for FP2/2SH, FP10/10S/10SH).

Be sure that **s2** ≤ **s3**.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F96_SRC **(see page 1326)**

Data types

Variable	Data type	Function
s1	ANY16	16-bit area or equivalent constant to store the value searched for
s2		starting 16-bit area of the block
s3		ending 16-bit area of the block

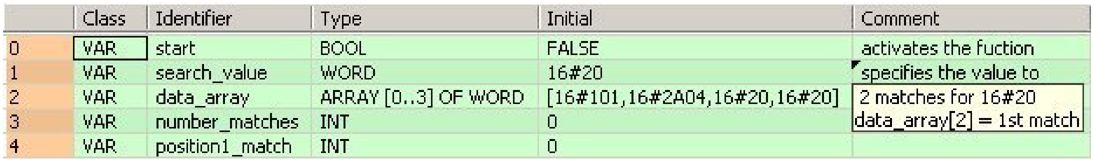
The variables **s1, s2** and **s3** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s2, s3	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

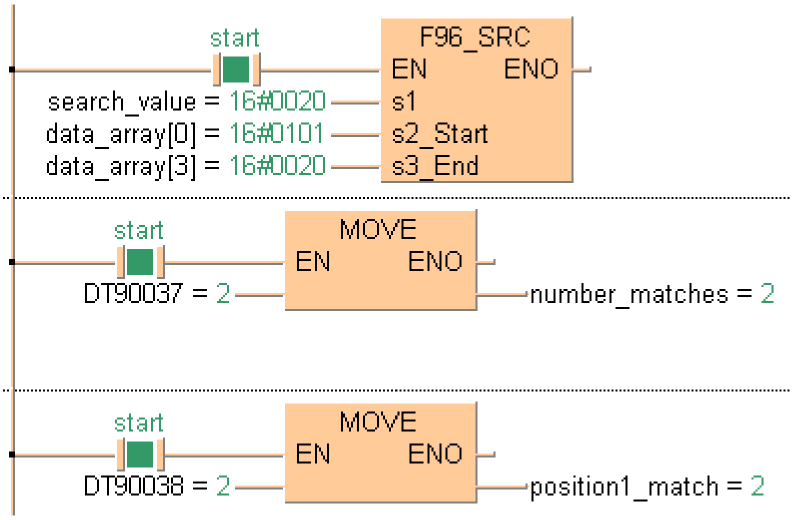
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the fuction
1	VAR	search_value	WORD	16#20	specifies the value to
2	VAR	data_array	ARRAY [0..3] OF WORD	[16#101,16#2A04,16#20,16#20]	2 matches for 16#20 data_array[2] = 1st match
3	VAR	number_matches	INT	0	
4	VAR	position1_match	INT	0	

Body When the variable **start** is set to TRUE, the function is carried out.

LD

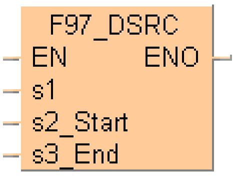


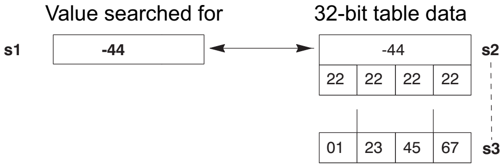
ST
```
IF start THEN
    F96_SRC( s1:= search_value ,
        s2_Start:= data_array[0] ,
        s3_End:= data_array[3] );
    number_matches:=DT90037;
    position_1match:=DT90038;
END_IF;
```

F97_DSRC

32-bit table data search

Description The function searches for the value specified at input **s1** in a block of 32-bit areas whose beginning is specified at input **s2** and whose end is specified at input **s3**.





The number of data items that match **s1** is stored in special data register DT90037.

The relative position of the first matching data item, counting from the starting 32-bit area **s2**, is stored in special data register DT90038.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F97_DSRC **(see page 1326)**

Data types

Variable	Data type	Function
s1	ANY32	32-bit area or equivalent constant to store the value searched for
s2		starting 32-bit area of the block
s3		ending 32-bit area of the block

The adresses of the variables at inputs **s2** and **s3** must be of the same adress type.

Operands

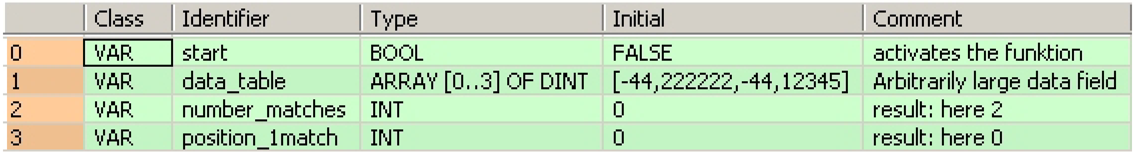
For	Relay				T/C		Register			Constant
s1	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s2, s3	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the address of the variables at outputs **s2** > **s3**.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

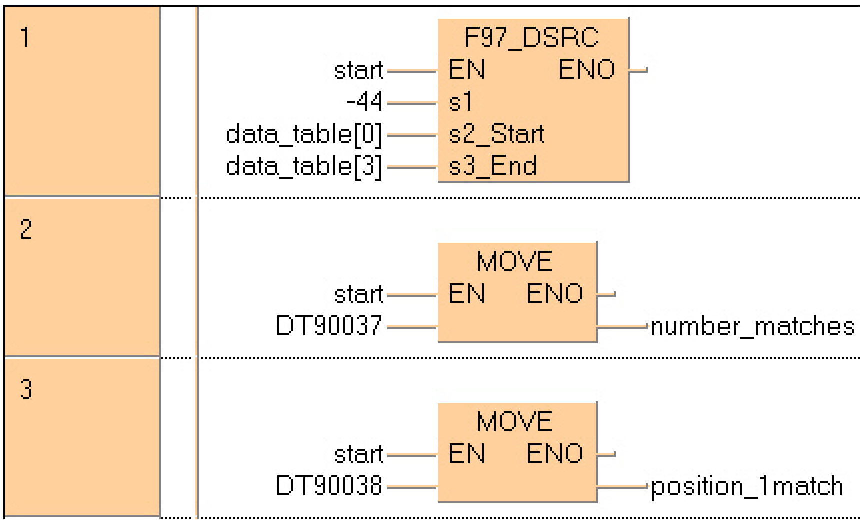
POU header All input and output variables used for programming this function have been declared in the POU header.



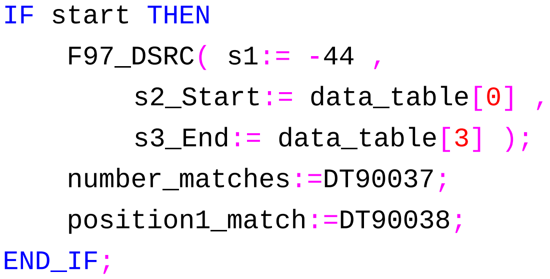
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the funktion
1	VAR	data_table	ARRAY [0..3] OF DINT	[-44,222222,-44,12345]	Arbitrarily large data field
2	VAR	number_matches	INT	0	result: here 2
3	VAR	position_1match	INT	0	result: here 0

Body When the variable **start** is set to TRUE, the function is carried out. Instead of using an input variable in this example, a constant (-44) is assigned to input s1. The result is stored in special data registers DT90037 and DT90038. The two E_MOVE functions copy the results to the two variables **number_matches** and **position_1match**.

LD



ST



```
IF start THEN
    F97_DSRC( s1:= -44 ,
        s2_Start:= data_table[0] ,
        s3_End:= data_table[3] );
    number_matches:=DT90037;
    position1_match:=DT90038;
END_IF;
```

15.1 Introduction into the FIFO buffer

The FIFO buffer is a first-in-first-out buffer area realized as a ring buffer. Data is stored in the order in which it is written to the buffer, and then read out in the order stored, starting from the first data item stored. It is convenient for buffering objects in sequential order.

Usage procedure

- The area to be used is defined as the FIFO buffer using the F115_FIFT (see page 483) instruction. (This should be done only once, before reading or writing is done.)
- Data should be written to the buffer using the F117_FIFW (see page 491) instruction, and read out of the buffer using the F116_FIFR (see page 487) instruction.

Writing data

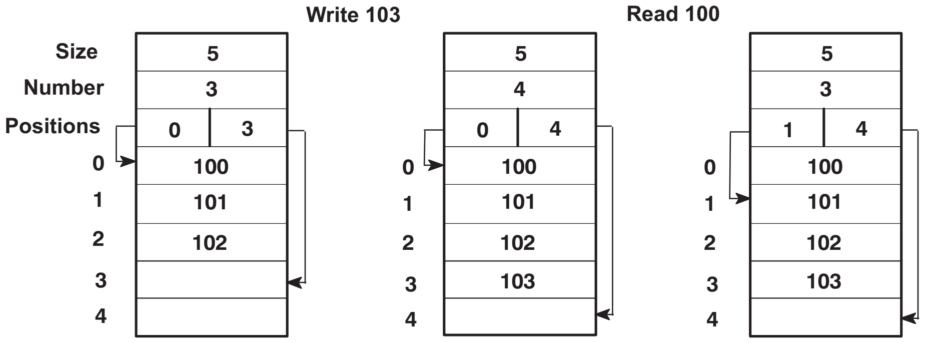
- When data is written, the data items are stored in sequential order, starting from the first data storage area. The writing pointer indicates the next area to which data is to be written. The number of words stored increases by 1.
- If the data storage area becomes full, i.e. the number of words stored is equal to n-1, further data writing is inhibited.

Reading data

- When data is read, data is transferred starting from the first data item stored. The reading pointer indicates the next area from which data is to be read. The number of words stored decreases by 1.
- An error occurs if an attempt is made to read data when the data storage area is empty, the number of words stored is equal to the memory size of the FIFO buffer or is equal to zero.

Data storage area

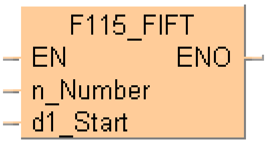
If data is written while the FIFO buffer is in the status shown below, the data will be stored in the area indicated by 3. The writing pointer moves to 4, i.e. the next data item will be written to 4. If data is read, it will be read from the area indicated by 0. The reading pointer then moves to 1, i.e. the next data item will be read from 1. (For more information on the reading and writing pointer, see F115_FIFT (see page 483)).



F115_FIFT

FIFO buffer area definition

Description F115 specifies the starting area d1 for the FIFO (First-In-First-Out) buffer and the memory size n of the FIFO buffer.

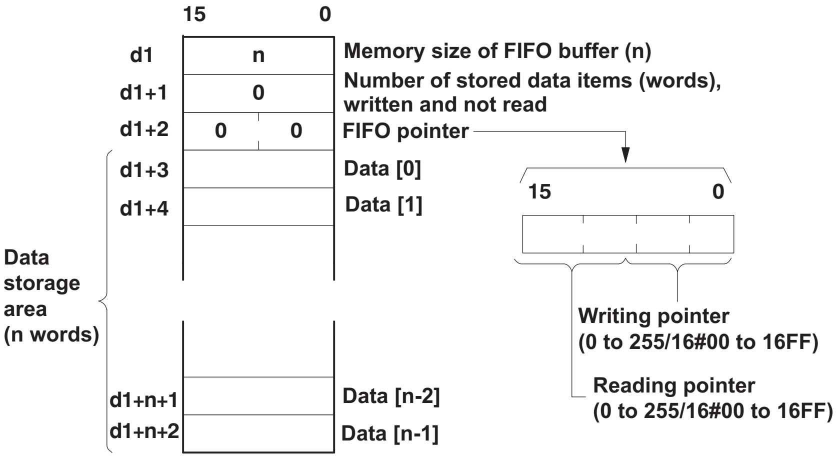


n: memory size (number of words (16-bit)) of FIFO buffer,
n = 1 to 256.

d1: the starting 16-bit area of FIFO buffer

How to use the FIFO buffer (see page 483)

Definition of the area using the FIFT instruction should be carried out only once, before writing to or reading from the FIFO buffer. When the FIFT instruction is executed, the FIFO buffer area is defined as follows:


15
0
d1
n
Memory size of FIFO buffer (n)
d1+1
0
Number of stored data items (words), written and not read
d1+2
0
0
FIFO pointer
d1+3
Data [0]
d1+4
Data [1]
15
0
Data storage area (n words)
Writing pointer (0 to 255/16#00 to 16FF)
Reading pointer (0 to 255/16#00 to 16FF)
d1+n+1
Data [n-2]
d1+n+2
Data [n-1]

When the FIFT instruction is executed, the following are stored as default values: **d1** = **n** (the value specified by the FIFT instruction), **d1** + 1 = 0, and **d1** + 2 = 16#0000.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of F115_FIFT (see page 1320)**

Data types

Variable	Data type	Function
n	INT	specifies the memory size of FIFO buffer
d1	ANY16	starting 16-bit area of FIFO buffer

Operands

For	Relay				T/C		Register			Constant
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1	-	WY	WR	WL	SV	EV	DT	LD	FL	-

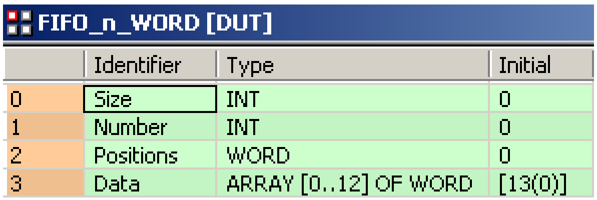
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ **n** = 0 ▪ **n** > 256 ▪ The area specified by **n** exceeds the limit
R9008	%MX0.900.8	for an instant	

Example

This example illustrates the FIFO buffer by incorporating the functions F115_FIFT (see page 483), F116_FIFR (see page 487) and F117_FIFW (see page 491). The function has been programmed in ladder diagram (LD) and structured text (ST).

DUT

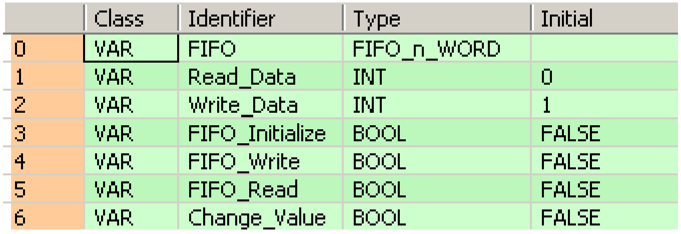


FIFO_n_WORD [DUT]

	Identifier	Type	Initial
0	Size	INT	0
1	Number	INT	0
2	Positions	WORD	0
3	Data	ARRAY [0..12] OF WORD	[13(0)]

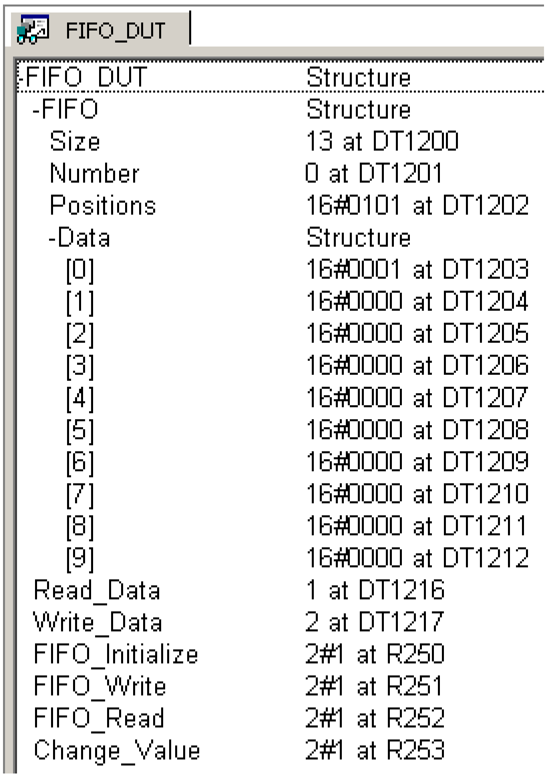
POU header

All input and output variables used for programming this function have been declared in the POU header.



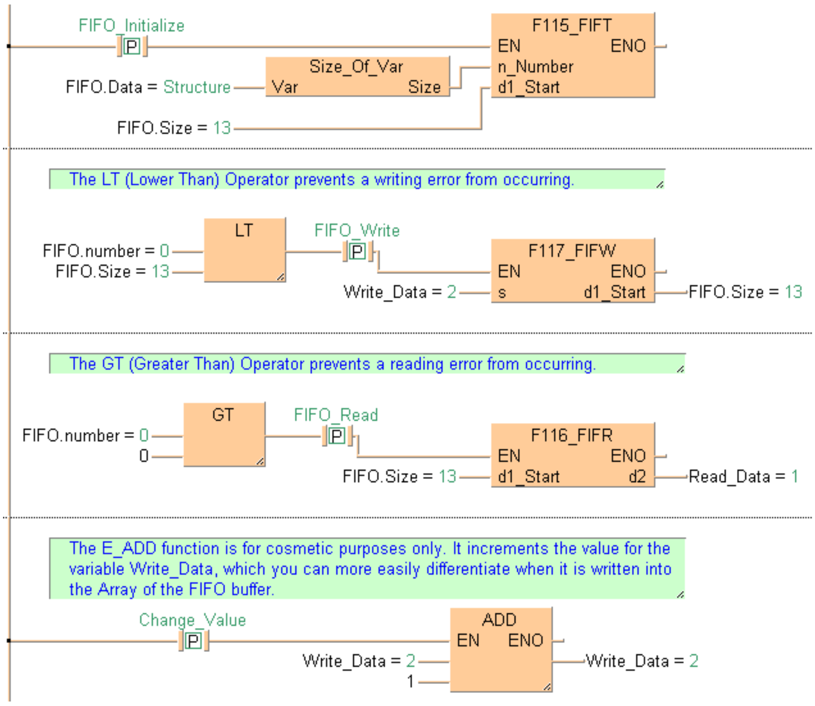
	Class	Identifier	Type	Initial
0	VAR	FIFO	FIFO_n_WORD	
1	VAR	Read_Data	INT	0
2	VAR	Write_Data	INT	1
3	VAR	FIFO_Initialize	BOOL	FALSE
4	VAR	FIFO_Write	BOOL	FALSE
5	VAR	FIFO_Read	BOOL	FALSE
6	VAR	Change_Value	BOOL	FALSE

Body The example below illustrates the status of the buffer after **FIFO_Write** has been enabled twice and **FIFO_Read** once. When **FIFO_Write** was activated the first time, the value 1 was written into **FIFO.Data[0]**. When **FIFO_Read** was enabled, **Read_Data** then read this value. When **FIFO_Write** was enabled the second time, the Writing pointer was incremented by one and the value 2 written into **FIFO.Data[1]**. see Entry Data Monitor 1



```
FIFO_DUT
-FIFO_DUT            Structure
 -FIFO               Structure
   Size              13 at DT1200
   Number            0 at DT1201
   Positions         16#0101 at DT1202
   -Data             Structure
     [0]             16#0001 at DT1203
     [1]             16#0000 at DT1204
     [2]             16#0000 at DT1205
     [3]             16#0000 at DT1206
     [4]             16#0000 at DT1207
     [5]             16#0000 at DT1208
     [6]             16#0000 at DT1209
     [7]             16#0000 at DT1210
     [8]             16#0000 at DT1211
     [9]             16#0000 at DT1212
 Read_Data           1 at DT1216
 Write_Data          2 at DT1217
 FIFO_Initialize     2#1 at R250
 FIFO_Write          2#1 at R251
 FIFO_Read           2#1 at R252
 Change_Value        2#1 at R253
```

LD



The LT (Lower Than) Operator prevents a writing error from occurring.

The GT (Greater Than) Operator prevents a reading error from occurring.

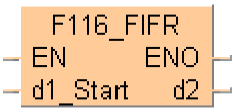
The E_ADD function is for cosmetic purposes only. It increments the value for the variable Write_Data, which you can more easily differentiate when it is written into the Array of the FIFO buffer.

```
ST IF DF(FIFO_Initalize) THEN
       (* Create the FIFO buffer *)
       F115_FIFT( n_Number:= Size_Of_Var(FIFO.Data), d1_Start:= FIFO.Size);
       REPEAT
          (* Initialize FIFO buffer with values *)
          Write_Data:=Write_Data+1;
          F117_FIFW( s:= Write_Data, d1_Start:=  FIFO.Size);
       UNTIL(FIFO.Number>=FIFO.Size)
       END_REPEAT;
   END_IF;

   IF DF( FIFO_Write) THEN
       (* Write value of Write_Data to FIFO buffer *)
       (* at rising edge of FIFO_Write *)
       F117_FIFW( s:= Write_Data, d1_Start:=  FIFO.Size);
   END_IF;

   IF DF(FIFO_Read) THEN
       (* Read value from FIFO buffer *)
       (* at rising edge of FIFO_Read *)
       F116_FIFR( d1_Start:= FIFO.Size, d2:= Read_Data);
   END_IF;
```

F116_FIFR

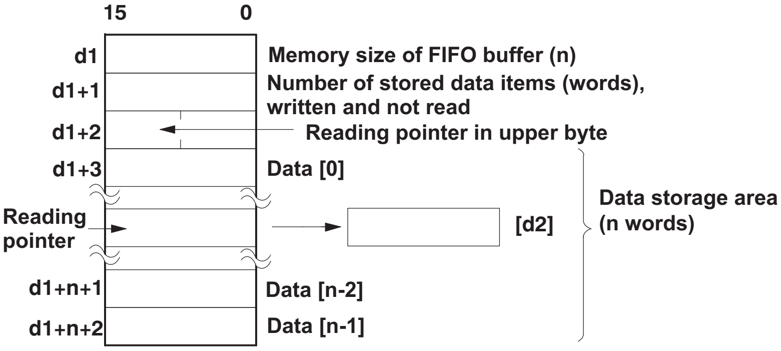
Read from FIFO buffer

Description F/P116 reads the data **d1** from the FIFO (First-In-First-Out) buffer and stores the data in area specified by **d2**.



How to use the FIFO buffer (see page 483)

Reading of data is done starting from the address specified by the reading pointer when the instruction is executed.


15
0
d1
d1+1
d1+2
d1+3
Reading pointer
d1+n+1
d1+n+2
Memory size of FIFO buffer (n)
Number of stored data items (words), written and not read
Reading pointer in upper byte
Data [0]
[d2]
Data [n-2]
Data [n-1]
Data storage area (n words)

- (0), (n–2) and (n–1) are addresses assigned to the data storage area.
- n is the value specified by the F115_FIFT (see page 483) instruction.

The reading pointer is stored in the upper eight bits of the third word of the FIFO buffer area. The actual address is the value of the leading address in the FIFO buffer area specified by d1 plus 3, plus the value of reading pointer (the value of which only the first byte is a decimal value).

When the reading is executed, 1 is subtracted from the number of stored data items, and the reading pointer is incremented by 1, or reset to zero if the reading pointer pointed to the final element.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of F116_FIFR (see page 1320)**

Data types

Variable	Data type	Function
d1	ANY16	starting 16-bit area of FIFO buffer
d2		16-bit area for storing data read from FIFO buffer

The variables **d1** and **d2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Const.
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-

☞

- **An error occurs if this is executed when the number of stored data items is 0 or when the reading pointer is equal to the writing pointer.**
- **Reading is only carried out when the reading pointer is not equal to the writing pointer.**
- **If this is executed when the reading pointer is indicating the final address in the FIFO buffer (the n defined by the FIFO instruction minus 1), the reading pointer is set to 0.**

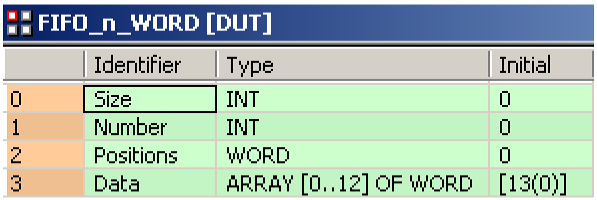
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the size (n) of the FIFO specified by **d1** is **n** = 0, or when **n** > 256. ▪ the number of stored data items of the FIFO = 0. ▪ the number of stored data items of the FIFO > FIFO size (n). ▪ the final address of the FIFO based on the FIFO size (n) exceeds the area. ▪ the FIFO reading pointer > FIFO size (n). ▪ the FIFO reading pointer is 256 (16#100) or higher after the data has been read.
R9008	%MX0.900.8	for an instant	

Example

This example illustrates the FIFO buffer by incorporating the functions F115_FIFT (see page 483), F116_FIFR (see page 487) and F117_FIFW (see page 491). The function has been programmed in ladder diagram (LD) and structured text (ST).

DUT

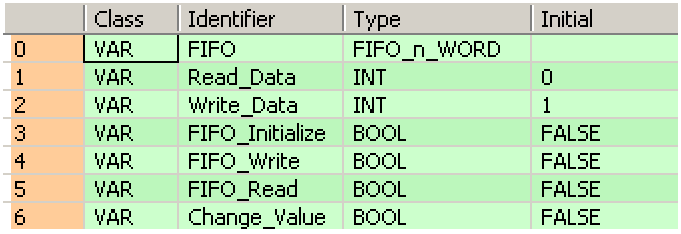


FIFO_n_WORD [DUT]

	Identifier	Type	Initial
0	Size	INT	0
1	Number	INT	0
2	Positions	WORD	0
3	Data	ARRAY [0..12] OF WORD	[13(0)]

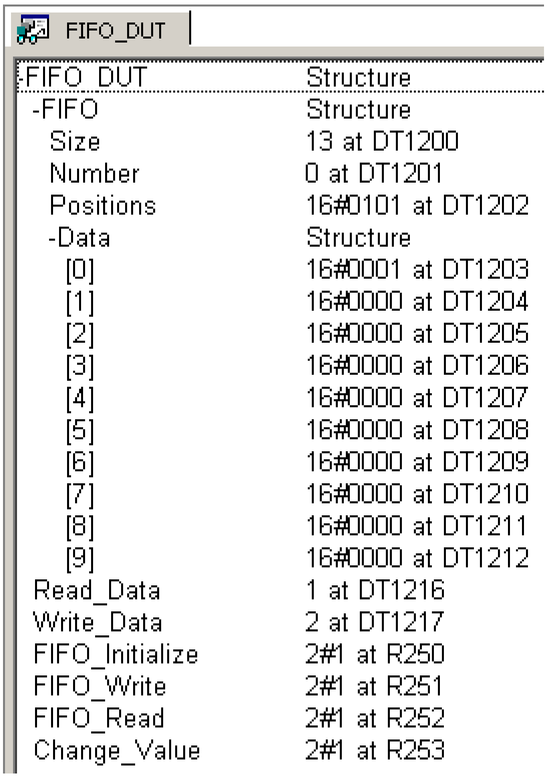
POU header

All input and output variables used for programming this function have been declared in the POU header.



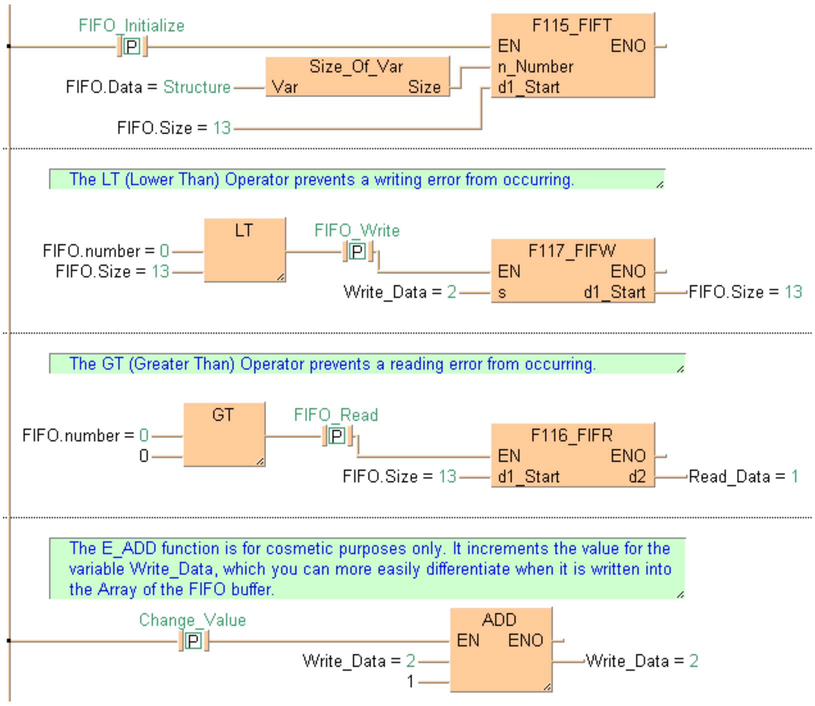
	Class	Identifier	Type	Initial
0	VAR	FIFO	FIFO_n_WORD	
1	VAR	Read_Data	INT	0
2	VAR	Write_Data	INT	1
3	VAR	FIFO_Initialize	BOOL	FALSE
4	VAR	FIFO_Write	BOOL	FALSE
5	VAR	FIFO_Read	BOOL	FALSE
6	VAR	Change_Value	BOOL	FALSE

Body The example below illustrates the status of the buffer after **FIFO_Write** has been enabled twice and **FIFO_Read** once. When **FIFO_Write** was activated the first time, the value 1 was written into **FIFO.Data[0]**. When **FIFO_Read** was enabled, **Read_Data** then read this value. When **FIFO_Write** was enabled the second time, the Writing pointer was incremented by one and the value 2 written into **FIFO.Data[1]**. see Entry Data Monitor 1



```
FIFO_DUT

-FIFO_DUT            Structure
 -FIFO               Structure
   Size              13 at DT1200
   Number            0 at DT1201
   Positions         16#0101 at DT1202
   -Data             Structure
     [0]             16#0001 at DT1203
     [1]             16#0000 at DT1204
     [2]             16#0000 at DT1205
     [3]             16#0000 at DT1206
     [4]             16#0000 at DT1207
     [5]             16#0000 at DT1208
     [6]             16#0000 at DT1209
     [7]             16#0000 at DT1210
     [8]             16#0000 at DT1211
     [9]             16#0000 at DT1212
 Read_Data           1 at DT1216
 Write_Data          2 at DT1217
 FIFO_Initialize     2#1 at R250
 FIFO_Write          2#1 at R251
 FIFO_Read           2#1 at R252
 Change_Value        2#1 at R253
```

LD



The LT (Lower Than) Operator prevents a writing error from occurring.

The GT (Greater Than) Operator prevents a reading error from occurring.

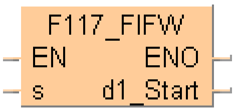
The E_ADD function is for cosmetic purposes only. It increments the value for the variable Write_Data, which you can more easily differentiate when it is written into the Array of the FIFO buffer.

```
ST IF DF(FIFO_Initalize) THEN
       (* Create the FIFO buffer *)
       F115_FIFT( n_Number:= Size_Of_Var(FIFO.Data), d1_Start:= FIFO.Size);
       REPEAT
          (* Initialize FIFO buffer with values *)
          Write_Data:=Write_Data+1;
          F117_FIFW( s:= Write_Data, d1_Start:=  FIFO.Size);
       UNTIL(FIFO.Number>=FIFO.Size)
       END_REPEAT;
   END_IF;

   IF DF( FIFO_Write) THEN
       (* Write value of Write_Data to FIFO buffer *)
       (* at rising edge of FIFO_Write *)
       F117_FIFW( s:= Write_Data, d1_Start:=  FIFO.Size);
   END_IF;

   IF DF(FIFO_Read) THEN
       (* Read value from FIFO buffer *)
       (* at rising edge of FIFO_Read *)
       F116_FIFR( d1_Start:= FIFO.Size, d2:= Read_Data);
   END_IF;
```

F117_FIFW

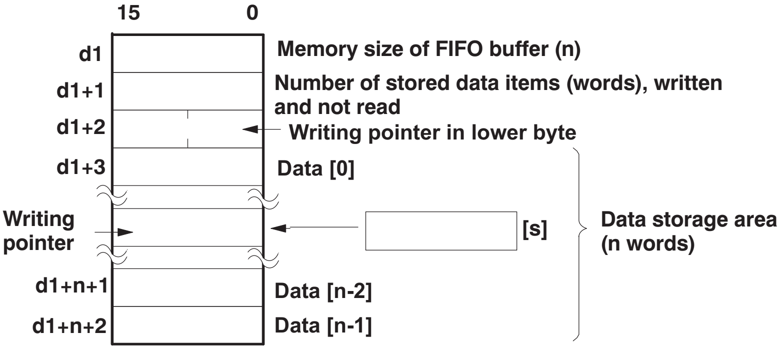
Write to FIFO buffer

Description F/P117 writes the data specified by **s** into the FIFO buffer specified by **d1**.



How to use the FIFO buffer (see page 483)

The specified data is written to the address indicated by the writing pointer when the instruction is executed.


15
0
d1
d1+1
d1+2
d1+3
Writing pointer
d1+n+1
d1+n+2
Memory size of FIFO buffer (n)
Number of stored data items (words), written and not read
Writing pointer in lower byte
Data [0]
[s]
Data storage area (n words)
Data [n-2]
Data [n-1]

- (0), (n-2) and (n-1) are addresses assigned to the data storage area.
- **n** is the value specified by the F115_FIFT (see page 483) instruction.

The writing pointer is stored in the lower eight bits of the third word of the FIFO buffer area, and is indicated by a relative position in the data storage area. The actual address to which data is being written is specified by d1 plus the offset 3 plus the value of the writing pointer (the value of which only the lower byte is a decimal value).

When the writing is executed, 1 is added to the number of stored data items, and the writing pointer is incremented by 1, or reset to zero if the writing pointer pointed to the final element.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of F117_FIFW (see page 1320)**

Data types

Variable	Data type	Function
s	ANY16	16-bit area or equivalent constant for storing data to write in the FIFO buffer
d1		starting 16-bit area of FIFO buffer

The variables **s** and **d1** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

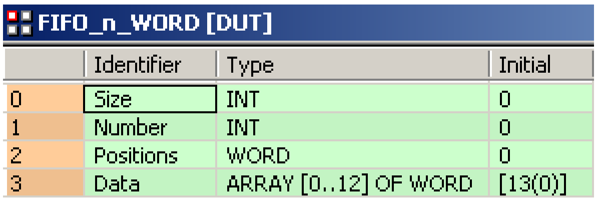
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the size (n) of the FIFO specified by **d1** is **n** = 0, or when **n** > 256. ▪ the number of stored data items of the FIFO = 0. ▪ the number of stored data items of the FIFO > FIFO size (n). ▪ the final address of the FIFO based on the FIFO size (n) exceeds the area. ▪ the FIFO writing pointer > FIFO size (n). ▪ the FIFO writing pointer is 256 (16#100) or higher after the data has been written.
R9008	%MX0.900.8	for an instant	

☞

- **An error occurs if this is executed when the FIFO buffer is full (the number of stored data items = the size n of the FIFO defined by the FIFT instruction). Writing is inhibited.**
- **If this is executed when the writing pointer is indicating the final address in the FIFO buffer (the "n" value defined by the FIFT instruction), the writing pointer will be set to 0.**

Example This example illustrates the FIFO buffer by incorporating the functions F115_FIFT (see page 483), F116_FIFR (see page 487) and F117_FIFW (see page 491). The function has been programmed in ladder diagram (LD) and structured text (ST).

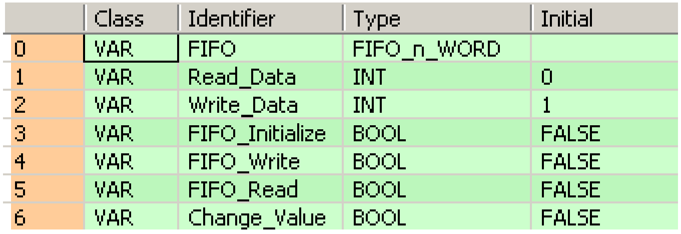
DUT



FIFO_n_WORD [DUT]

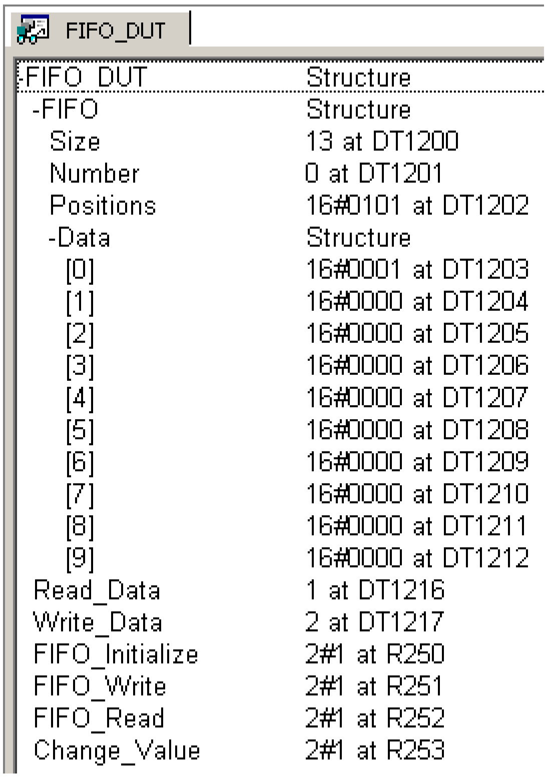
	Identifier	Type	Initial
0	Size	INT	0
1	Number	INT	0
2	Positions	WORD	0
3	Data	ARRAY [0..12] OF WORD	[13(0)]

POU header All input and output variables used for programming this function have been declared in the POU header.



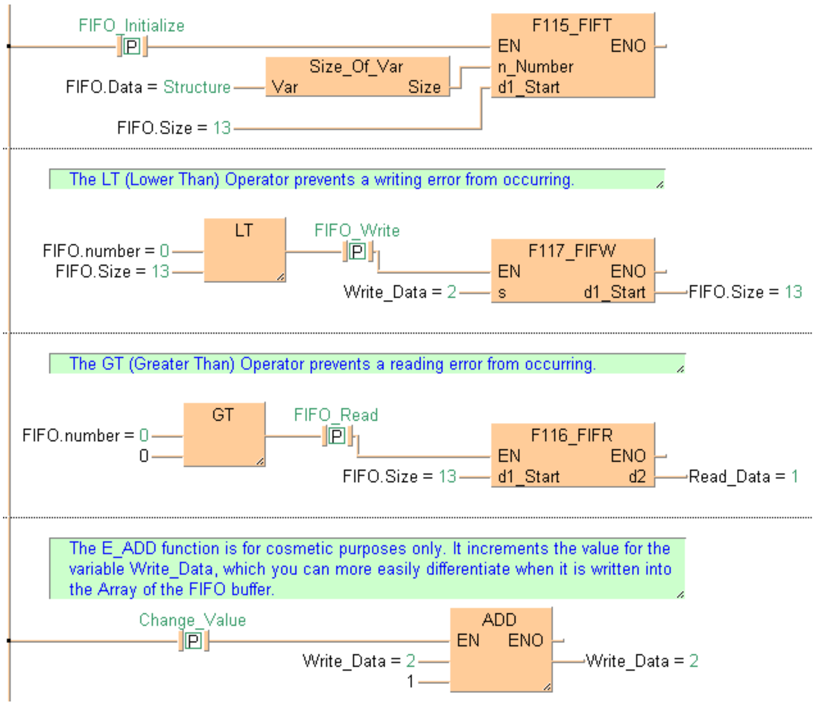
	Class	Identifier	Type	Initial
0	VAR	FIFO	FIFO_n_WORD	
1	VAR	Read_Data	INT	0
2	VAR	Write_Data	INT	1
3	VAR	FIFO_Initialize	BOOL	FALSE
4	VAR	FIFO_Write	BOOL	FALSE
5	VAR	FIFO_Read	BOOL	FALSE
6	VAR	Change_Value	BOOL	FALSE

Body The example below illustrates the status of the buffer after **FIFO_Write** has been enabled twice and **FIFO_Read** once. When **FIFO_Write** was activated the first time, the value 1 was written into **FIFO.Data[0]**. When **FIFO_Read** was enabled, **Read_Data** then read this value. When **FIFO_Write** was enabled the second time, the Writing pointer was incremented by one and the value 2 written into **FIFO.Data[1]**. see Entry Data Monitor 1



```
FIFO_DUT
-FIFO_DUT               Structure
 -FIFO                  Structure
   Size                 13 at DT1200
   Number               0 at DT1201
   Positions            16#0101 at DT1202
   -Data                Structure
     [0]                16#0001 at DT1203
     [1]                16#0000 at DT1204
     [2]                16#0000 at DT1205
     [3]                16#0000 at DT1206
     [4]                16#0000 at DT1207
     [5]                16#0000 at DT1208
     [6]                16#0000 at DT1209
     [7]                16#0000 at DT1210
     [8]                16#0000 at DT1211
     [9]                16#0000 at DT1212
 Read_Data              1 at DT1216
 Write_Data             2 at DT1217
 FIFO_Initialize        2#1 at R250
 FIFO_Write             2#1 at R251
 FIFO_Read              2#1 at R252
 Change_Value           2#1 at R253
```

LD



The LT (Lower Than) Operator prevents a writing error from occurring.

The GT (Greater Than) Operator prevents a reading error from occurring.

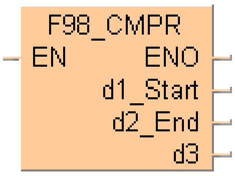
The E_ADD function is for cosmetic purposes only. It increments the value for the variable Write_Data, which you can more easily differentiate when it is written into the Array of the FIFO buffer.

```
ST IF DF(FIFO_Initalize) THEN
       (* Create the FIFO buffer *)
       F115_FIFT( n_Number:= Size_Of_Var(FIFO.Data), d1_Start:= FIFO.Size);
       REPEAT
          (* Initialize FIFO buffer with values *)
          Write_Data:=Write_Data+1;
          F117_FIFW( s:= Write_Data, d1_Start:=  FIFO.Size);
       UNTIL(FIFO.Number>=FIFO.Size)
       END_REPEAT;
   END_IF;

   IF DF( FIFO_Write) THEN
       (* Write value of Write_Data to FIFO buffer *)
       (* at rising edge of FIFO_Write *)
       F117_FIFW( s:= Write_Data, d1_Start:=  FIFO.Size);
   END_IF;

   IF DF(FIFO_Read) THEN
       (* Read value from FIFO buffer *)
       (* at rising edge of FIFO_Read *)
       F116_FIFR( d1_Start:= FIFO.Size, d2:= Read_Data);
   END_IF;
```

F98_CMPR

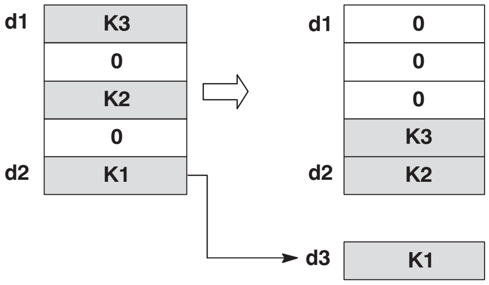
Data table shift-out and compress

Description Shifts out non-zero data stored at the highest address of the table to the specified area and compresses the data in the table to the higher address. The data in the table specified by **d1** and **d2** is rearranged as follows:



- Contents of **d2** (highest address) are shifted out to the area specified by **d3**.

Non-zero data is shifted (compressed) in sequential order, in the direction of the higher address in the specified range.



- Starting area **d1** and ending area **d2** should be the same type of operand.
- Be sure to specify **d1** and **d2** with "**d1** ≤ **d2**".

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F98_CMPR **(see page** 1326**)**

Data types

Variable	Data type	Function
d1	ANY16	starting (lowest) address of data to be compressed
d2		final (highest) address of data to be compressed, data at **d2** is shifted out
d3		receives data shifted out from **d2**

Operands

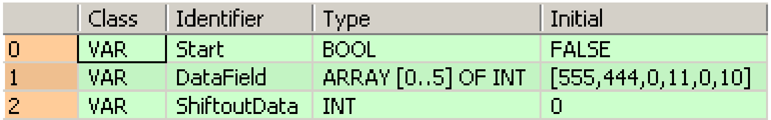
For	Relay				T/C		Register			Const.
d1, d2, d3	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ **d1** > **d2**
R9008	%MX0.900.8	for an instant	▪ **d1** and **d2** are not in the same memory area

Example 1 In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

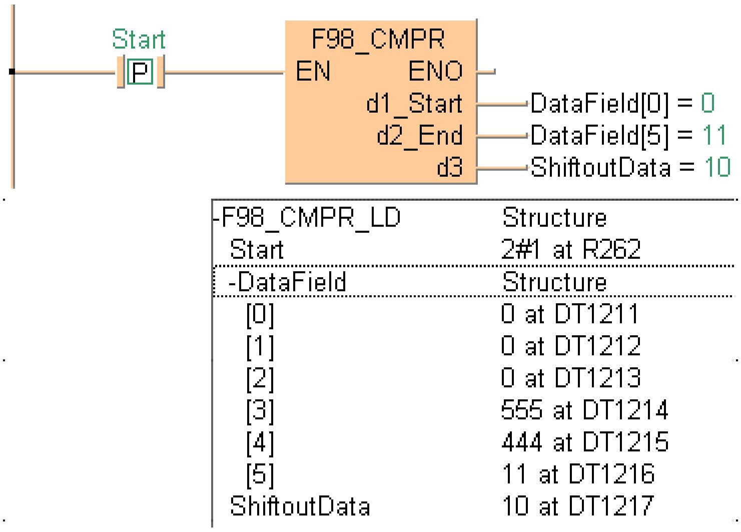
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Start	BOOL	FALSE
1	VAR	DataField	ARRAY [0..5] OF INT	[555,444,0,11,0,10]
2	VAR	ShiftoutData	INT	0

Body When the variable **start** is set to TRUE, the function is carried out. The data in the lower addresses is compressed toward the higher addresses, and the value defined at the highest address, i.e. 10, is shifted out.

LD



-F98_CMPR_LD	Structure
Start	2#1 at R262
-DataField	Structure
[0]	0 at DT1211
[1]	0 at DT1212
[2]	0 at DT1213
[3]	555 at DT1214
[4]	444 at DT1215
[5]	11 at DT1216
ShiftoutData	10 at DT1217

Example 2 In combination with the F99_ CMPW/ P99_CMPW instruction, this can be used to construct an optional buffer. (Use a FIFO buffer for non-zero values.)

1. Executing the F99_CMPW/ P99_CMPW instruction

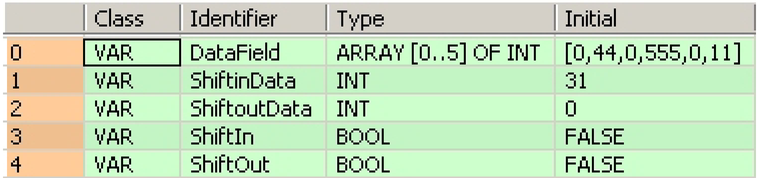
When data items are written to the first address of the buffer (the area of the specified range), they are stored and accumulated in the buffer in sequential order. The oldest data will be stored in the last address of the buffer.

2. Executing the F98_CMPR/ P98_CMPR instruction

When the data in the last address of the buffer (the area of the specified range) has been read, data can be extracted in sequential order, starting from the oldest data.

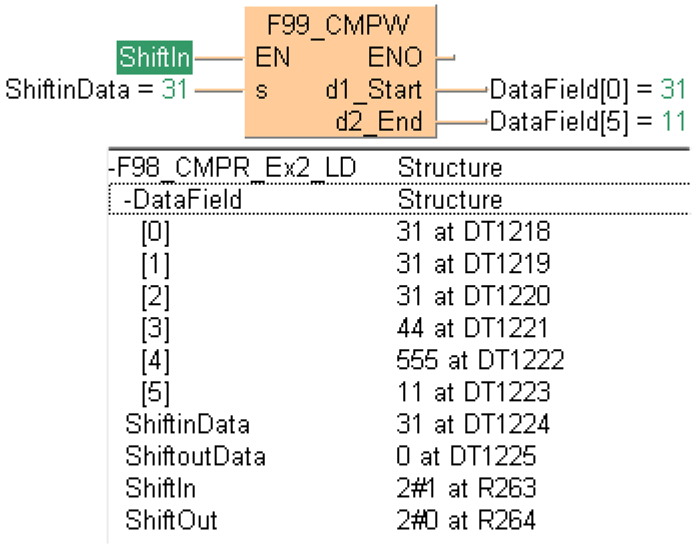
The rest of the data in the buffer is shifted in the direction of the first address, so normally, the oldest data at that point is stored in the last address of the buffer.

POU header



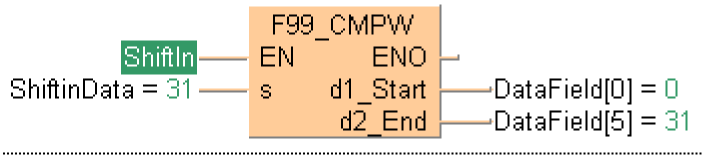
	Class	Identifier	Type	Initial
0	VAR	DataField	ARRAY [0..5] OF INT	[0,44,0,555,0,11]
1	VAR	ShiftinData	INT	31
2	VAR	ShiftoutData	INT	0
3	VAR	ShiftIn	BOOL	FALSE
4	VAR	ShiftOut	BOOL	FALSE

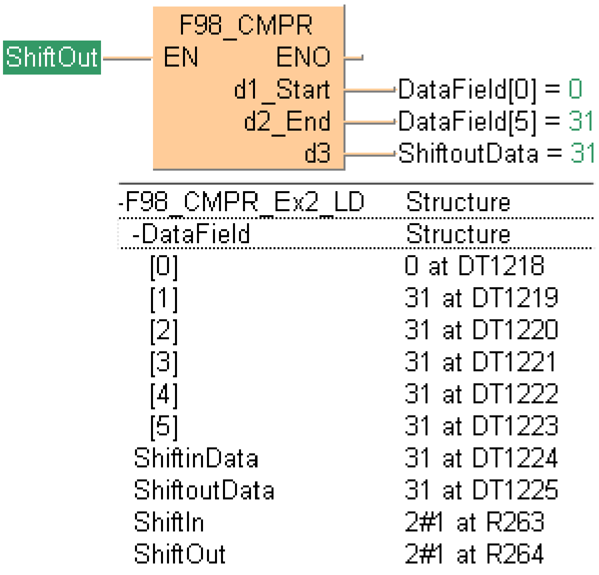
LD In Step 1 the F99 function is activated, shifting in the value given in the variable **ShiftinData** at **s**, i.e. 31, and compressing the rest of the data.



-F98_CMPR_Ex2_LD	Structure
-DataField	Structure
[0]	31 at DT1218
[1]	31 at DT1219
[2]	31 at DT1220
[3]	44 at DT1221
[4]	555 at DT1222
[5]	11 at DT1223
ShiftinData	31 at DT1224
ShiftoutData	0 at DT1225
ShiftIn	2#1 at R263
ShiftOut	2#0 at R264

In Step 2 the F98 function is activated, and the value defined in the variable at **d3**, i.e. 11, is shifted out.



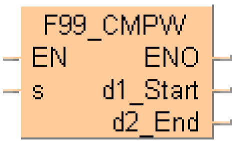


-F98_CMPR_Ex2_LD	Structure
-DataField	Structure
[0]	0 at DT1218
[1]	31 at DT1219
[2]	31 at DT1220
[3]	31 at DT1221
[4]	31 at DT1222
[5]	31 at DT1223
ShiftinData	31 at DT1224
ShiftoutData	31 at DT1225
ShiftIn	2#1 at R263
ShiftOut	2#1 at R264

F99_CMPW

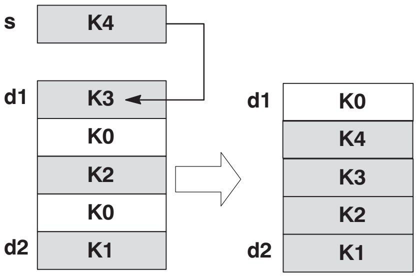
Data table shift-in and compress

Description Shifts in data to the smallest address of the specified data table and compresses the data in the table toward the higher address. The data in the table specified by **d1** and **d2** is rearranged as follows:



- Data specified by **s** is shifted in to the area specified by **d1** (starting address).

Non-zero data is shifted (compressed) in sequential order, in the direction of the higher address in the specified range.



- Starting area **d1** and ending area **d2** should be the same type of operand.
- Be sure to specify **d1** and **d2** with "**d1** ≤ **d2**".
- If the content of **s** is "0", only a compressed shift is carried out.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F99_CMPW **(see page 1326)**

For an example on how to construct a FIFO buffer using F/P99 and F/P98, see Example 2 from F/P98.

Data types

Variable	Data type	Function
s	ANY16	data to be shifted in
d1		starting address of area that is compressed into which data from **s** is shifted
d2		end address of area where data is compressed

Operands

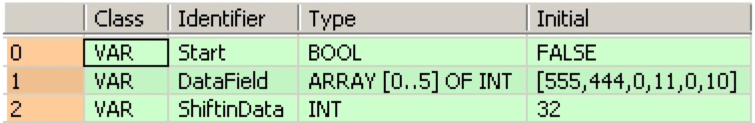
For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ **d1** > **d2**
R9008	%MX0.900.8	for an instant	▪ **d1** and **d2** are not in the same memory area

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

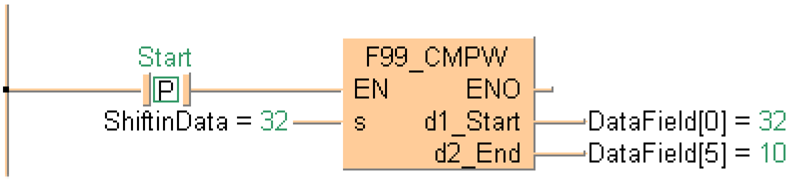
POU header All input and output variables used for programming this function have been declared in the POU header.

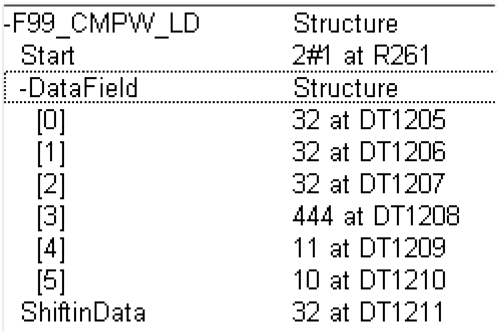


	Class	Identifier	Type	Initial
0	VAR	Start	BOOL	FALSE
1	VAR	DataField	ARRAY [0..5] OF INT	[555,444,0,11,0,10]
2	VAR	ShiftinData	INT	32

Body After the variable **Start** is set to TRUE, the value of the variable **ShiftinData**, i.e. 32, at the contact **s** is shifted into the specified area of the data table, and the data is compressed.

LD



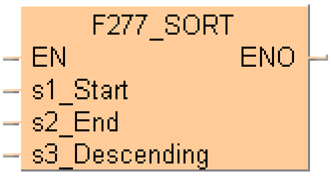


```
-F99_CMPW_LD        Structure
 Start              2#1 at R261
 -DataField         Structure
   [0]              32 at DT1205
   [1]              32 at DT1206
   [2]              32 at DT1207
   [3]              444 at DT1208
   [4]              11 at DT1209
   [5]              10 at DT1210
 ShiftinData        32 at DT1211
```

F277_SORT

Sort data in 16-bit data table (in smaller or larger number order)

Description The function sorts values (with +/- sign) in a data table in ascending or descending order.



Input **s1** specifies the starting area of the data table, and **s2** specifies the end. You determine the sorting order at input **s3**.

At input **s3** you can enter the following values:

0	ascending order, i.e. begin with the smallest value
1	descending order, i.e. begin with the largest value

The data are sorted via bubble sort in the order specified according to the value entered at input **s3**. Since the number of word comparisons increases in proportion to the square of the number of words, the sorting process can take some time when there are a large number of words. When the address of the variable at input **s1 = s2**, no sorting takes place.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F277_SORT **(see page 1323)**

Data types

Variable	Data type	Function
s1	INT	starting area of data table to be sorted
s2	INT	ending area of data table to be sorted
s3	INT	specifies sorting order: 0 = ascending, 1 = descending

Operands

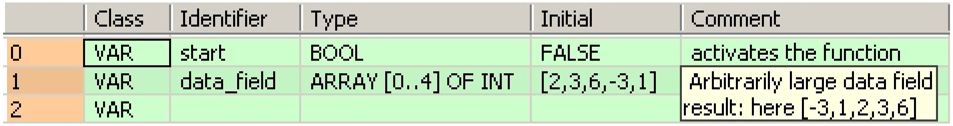
For	Relay				T/C		Register			Constant
s1, s2	-	WY	WR	WL	SV	EV	DT	LD	FL	-
s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the address of the variable at input **s1** > **s2** ▪ the address areas of the values at inputs **s1** and **s2** are different
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

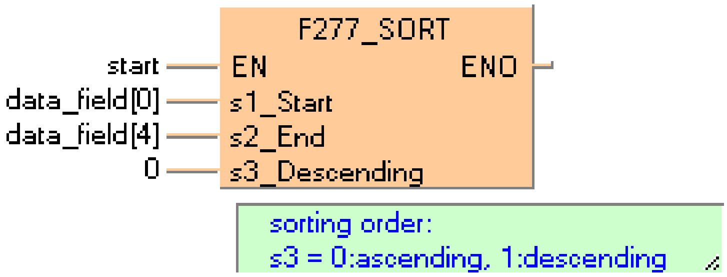
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF INT	[2,3,6,-3,1]	Arbitrarily large data field result: here [-3,1,2,3,6]
2	VAR				

Body When the variable **start** is set to TRUE, the function is carried out. The constant 0 is specified at input s3, which means the sorting is carried out in an ascending order. However, you can declare a variable in the POU header and write it in the function in the body at input s3.

LD



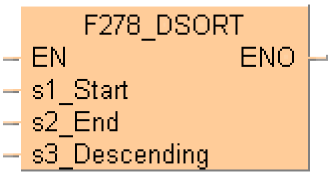
sorting order:
s3 = 0:ascending, 1:descending

ST
```
IF start THEN
    F277_SORT( s1_Start:= data_field[0],
        s2_End:= data_field[4],
        s3_Descending:= 0);
END_IF;
```

F278_DSORT

Sort data in 32-bit data table (in smaller or larger number order)

Description The function sorts values (with +/- sign) in a data table in ascending or descending order.



Input **s1** specifies the starting area of the data table, and **s2** specifies the end. You determine the sorting order at input **s3**.

At input **s3** you can enter the following values:

0	ascending order, i.e. begin with the smallest value
1	descending order, i.e. begin with the largest value

The data are sorted via bubble sort in the order specified according to the value entered at input **s3**. Since the number of word comparisons increases in proportion to the square of the number of words, the sorting process can take some time when there are a large number of words. When the address of the variable at input **s1 = s2**, no sorting takes place.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of F278_DSORT (see page 1323)**

☞ **Although this is a 32-bit instruction, the number of steps is the same as the 16-bit instruction.**

Data types

Variable	Data type	Function
s1	DINT	starting area of data table to be sorted
s2	DINT	ending area of data table to be sorted
s3	INT	specifies sorting order: 0 = ascending, 1 = descending

Operands

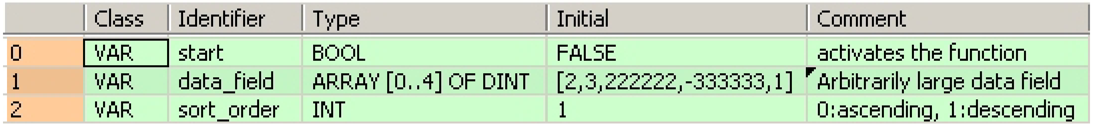
For	Relay				T/C		Register			Constant
s1, s2	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the address of the variable at input **s1** > **s2** ▪ the address areas of the values at inputs **s1** and **s2** are different
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

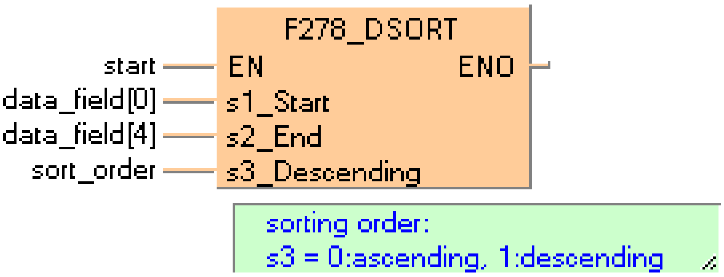


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF DINT	[2,3,222222,-333333,1]	Arbitrarily large data field
2	VAR	sort_order	INT	1	0:ascending, 1:descending

In this example, the input variable **sort_order** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out. Since the variable **sort_order** is set to 1, the specified data field in sorted in descending order.

LD



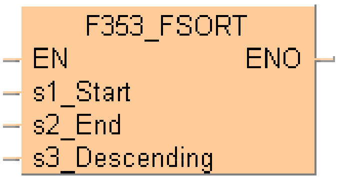
sorting order:
s3 = 0:ascending, 1:descending

ST
```
IF start THEN
    F278_DSORT( s1_Start:= data_field[0],
        s2_End:= data_field[4],
        s3_Descending:= sort_order);
END_IF;
```

F353_FSORT

Sort data in real number data table (floating point data table)

Description The function sorts values (with +/- sign) in a data table in ascending or descending order.



Input **s1** specifies the starting area of the data table, and **s2** specifies the end. You determine the sorting order at input **s3**.

At input **s3** you can enter the following values:

0	ascending order, i.e. begin with the smallest value
1	descending order, i.e. begin with the largest value

The data are sorted via bubble sort in the order specified according to the value entered at input **s1**. Since the number of word comparisons increases in proportion to the square of the number of words, the sorting process can take some time when there are a large number of words. When the value at inputs **s1 = s2**, no sorting takes place.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F353_FSORT **(see page 1325)**

Data types

Variable	Data type	Function
s1	REAL	starting area of data table to be sorted
s2	REAL	ending area of data table to be sorted
s3	INT	specifies sorting order: 0 = ascending, 1 = descending

Operands

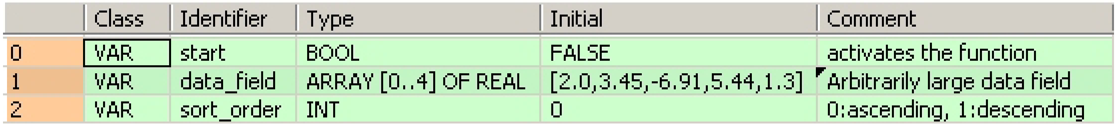
For	Relay				T/C		Register			Constant
s1, s2	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the addresses of variables at inputs **s1** > **s2**. ▪ the address areas are different. ▪ the floating point values exceed the processing range.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

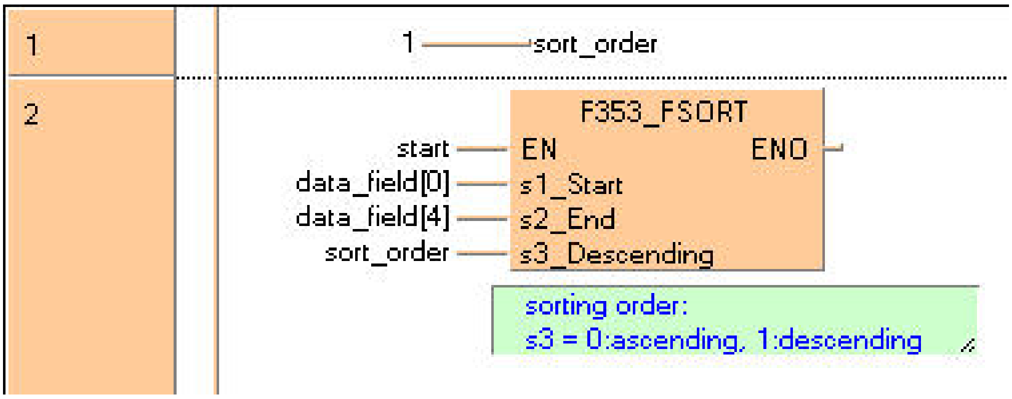


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..4] OF REAL	[2.0,3.45,-6.91,5.44,1.3]	Arbitrarily large data field
2	VAR	sort_order	INT	0	0:ascending, 1:descending

In this example, the input variable **sort_order** is declared. However, you can write a constant (e.g. 1 for a descending sorting order) directly at the input contact of the function in the body.

Body The variable **sort_order** is specified as the value 1. When the variable **start** is set to TRUE, the function is carried out. It sorts the elements of the ARRAY **data_field** in descending order.

LD



sorting order:
s3 = 0:ascending, 1:descending

ST
```
sort_order:=1;
IF start THEN
    F353_FSORT( s1_Start:= data_field[0],
        s2_End:= data_field[4],
        s3_Descending:= sort_order);
END_IF;
```

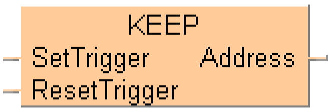
Chapter 16

Bistable instructions

KEEP

Serves as a relay with set and reset inputs

Description KEEP serves as a relay with set and reset points.



When the **SetTrigger** turns ON, output of the specified relay goes ON and maintains its condition. Output relay goes OFF when the **ResetTrigger** turns ON. The output relay's ON state is maintained until a **ResetTrigger** turns ON regardless of the ON or OFF states of the **SetTrigger**. If the **SetTrigger** and **ResetTrigger** turn ON simultaneously, the **ResetTrigger** is given priority.

PLC types **Availability of KEEP (see page 1328)**

Data types

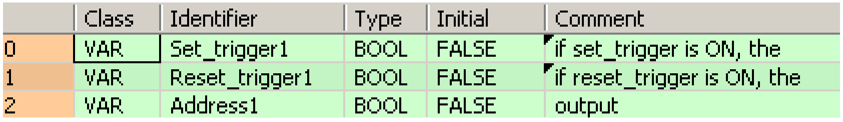
Variable	Data type	Function
Set Trigger	BOOL	sets Address output, i.e. turns in ON
Reset Trigger	BOOL	resets Address output, i.e. turns it OFF
Address	BOOL	specifed relay whose status (set or reset) is kept

Operands

For	Relay				T/C		Register			Constant
Set Trigger, Reset Trigger	X	Y	R	L	T	C	-	-	-	-
o	-	Y	R	L	-	-	-	-	-	-

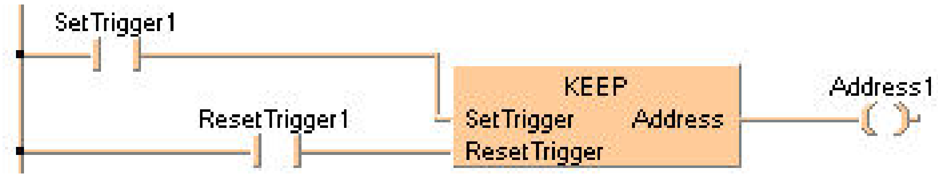
Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	Set_trigger1	BOOL	FALSE	if set_trigger is ON, the
1	VAR	Reset_trigger1	BOOL	FALSE	if reset_trigger is ON, the
2	VAR	Address1	BOOL	FALSE	output

LD



ST When programming with structured text, enter the following:

```
Address1:=KEEP(SetTrigger1, ResetTrigger1);
```

SET

SET, RESET

Description SET: When the execution conditions have been satisfied, the output is turned on, and the on status is retained.

RST: When the execution conditions have been satisfied, the output is turned off, and the off status is retained.

- You can use relays with the same number as many times as you like with the **SET** and **RST** instructions. (Even if a total check is run, this is not handled as a syntax error.)
- When the **SET** and **RST** instructions are used, the output changes with each step during processing of the operation.
- To output a result while operation is still in progress, use a partial I/O update instruction (**F143**).
- The output destination of a **SET** instruction is held even during the operation of an **MC** instruction.
- The output destination of a **SET** instruction is reset when the mode is changed from RUN to PROG. or when the power is turned off, except when a hold type internal relay is specified as the output destination.
- Placing a DF instruction (or specifying a rising edge in LD) before the **SET** and **RST** instructions ensures that the instruction is only executed at a rising edge.

Relays:

- Relays can be turned off using the **RST** instruction.
- Using the various relays with the **SET** and **RST** instructions does not result in double output.
- It is not possible to specify a pulse relay (P) as the output destination for a **SET** or **RST** instruction.

Operands

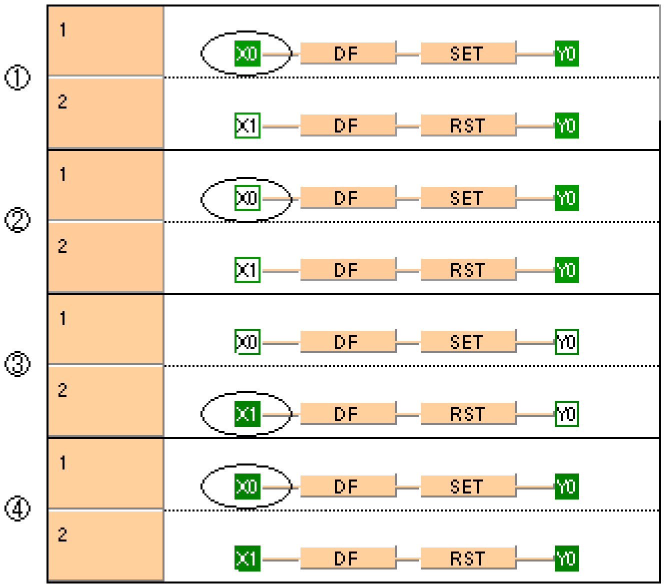
For	Relay				T/C		Register			Constant
SET **RST**	-	Y	R	L	-	-	-	E	-	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help. Since addresses are assigned directly using FP addresses, no POU header is necessary.

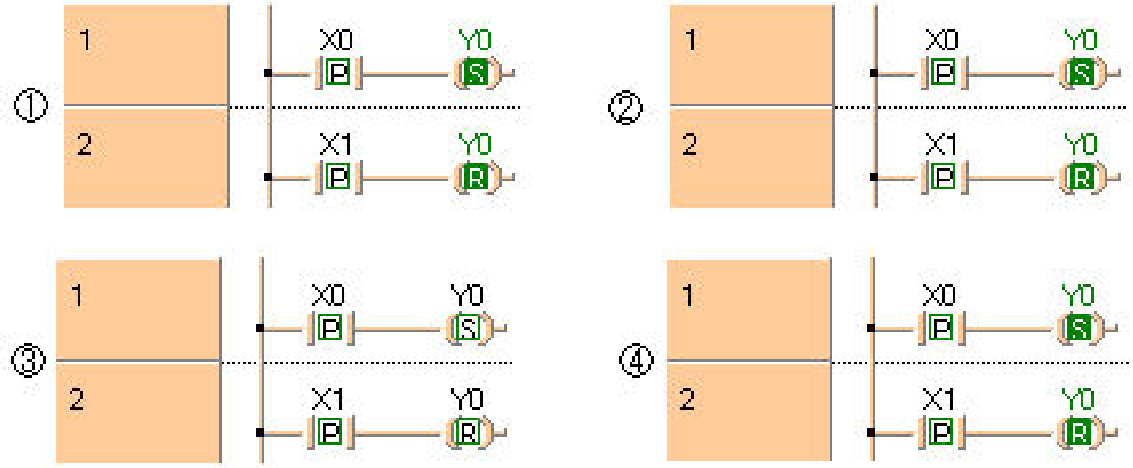
POU header All input and output variables used for programming this function have been declared in the POU header.

Using the DF command or specifying a rising edge refines the program by making the programming step valid for one scan only:
(1) When the input X0 is activated, the output Y0 is set.
(2) When the input X0 is turned off, the output Y0 remains set.
(3) When the input X1 is activated, the output Y0 is reset.
(4) When the input X0 is reactivated, the output Y0 is set.

FBD



LD In ladder diagram, specify a rising edge in the contact and SET or RESET in the coil:



ST When programming with structured text, enter the following:

```
(*TRUE and FALSE are assigned to Y0*)
IF DF(X0) THEN
      Y0:= TRUE;
END_IF;

IF DF(X1) THEN
      Y0:= FALSE;
END_IF;
```

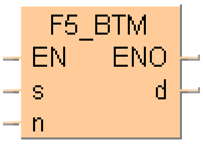
Chapter 17

Bitwise Boolean instructions

F5_BTM

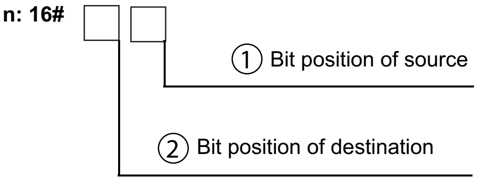
Bit data move

Description 1 bit of the 16-bit data or constant value specified by **s** is copied to a bit of the 16-bit area specified by **d** according to the content specified by **n** if the trigger **EN** is in the ON-state. When the 16-bit equivalent constant is specified by **s**, the bit data move operation is performed internally converting it to 16-bit binary expression.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

The operand **n** specifies the bit number as follows:



Bit No.	Description
0–3	source bit No. (16#0 to 16#F)
4–7	FP2/2SH and 10SH: number of bits to be transferred (16#0 to 16#F) FP3: invalid
8–11	destination bit No. (16#0 to 16#F)
12–15	invalid

For example, reading from the right, n = 16#C01 would move from bit position one, one bit to bit position 12 (16#C).

PLC types **Availability of** F5_BTM **(see page** 1325**)**

Data types

Variable	Data type	Function
s	ANY16	source 16-bit area
n		specifies source and destination bit positions
d		destination 16-bit area

The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Explanation with example value 16#8888 and bit at position 2 moves to destination value at bit position 15

source

bit pos	15	.	.	12	11	.	.	8	7	.	.	5	4	.	.	0
	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0

target

bit pos	15	.	.	12	11	.	.	8	7	.	.	5	4	.	.	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

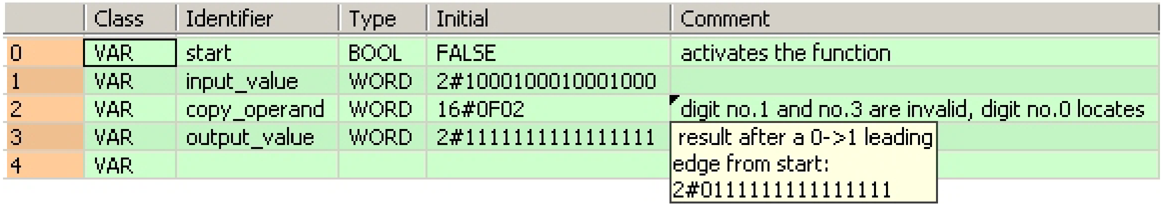
result

bit pos	15	.	.	12	11	.	.	8	7	.	.	5	4	.	.	0
	0	1	1	1												

Bit at position 15 is exchanged, destination value in this example: 16#7FFF

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

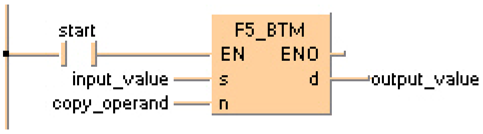
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	2#1000100010001000	
2	VAR	copy_operand	WORD	16#0F02	digit no.1 and no.3 are invalid, digit no.0 locates result after a 0->1 leading edge from start: 2#0111111111111111
3	VAR	output_value	WORD	2#1111111111111111	
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



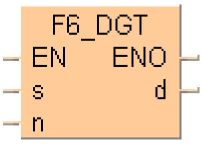
ST When programming with structured text, enter the following:

```
IF start THEN
    F5_BTM( s:= input_value,
        n:= copy_operand,
        d=> output_value);
END_IF;
```

F6_DGT

Digit data move

Description The hexadecimal digits in the 16-bit data or in the 16-bit equivalent constant specified by **s** are copied to the 16-bit area specified by **d** as specified by **n**.

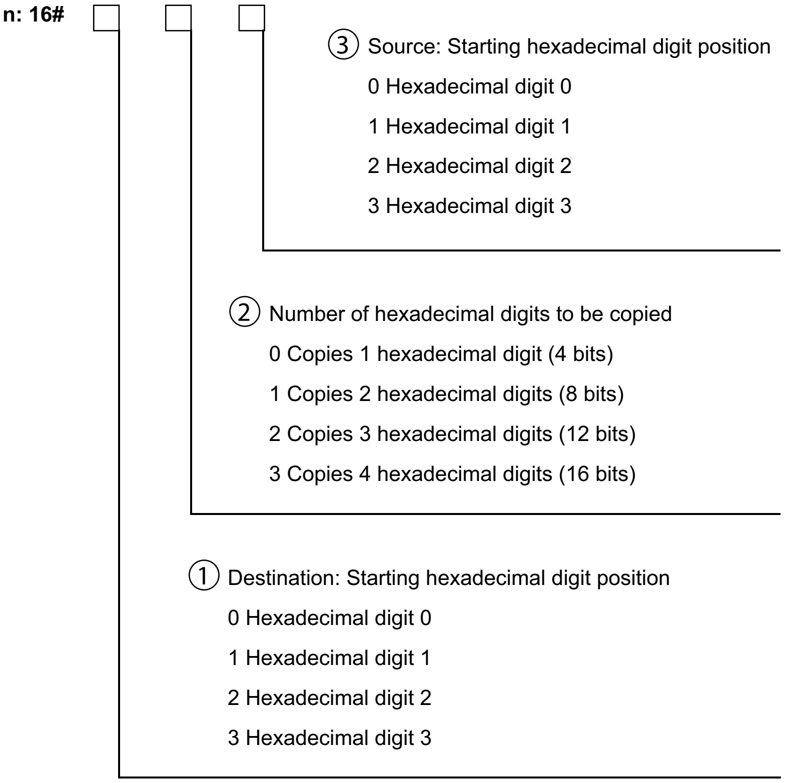


This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Digits are units of 4 bits used when handling data. With this instruction, 16-bit data is separated into four digits. The digits are called in order hexadecimal digit 0, digit 1, digit 2 and digit 3, beginning from the least significant four bits:

	16-bit data			
bit	15 . . 12	11 . . 8	7 . . 4	3 . . 0
	0 0 0 0	0 0 0 0	1 0 0 0	1 0 0 1
	hexadec. digit 3	hexadec. digit 2	hexadec. digit 1	hexadec. digit 0

n specifies the ③ source hexadecimal digit position, the ② number of digits and the ① destination hexadecimal digit position to be copied using hexadecimal data as follows:



③ Source: Starting hexadecimal digit position

0 Hexadecimal digit 0

1 Hexadecimal digit 1

2 Hexadecimal digit 2

3 Hexadecimal digit 3

② Number of hexadecimal digits to be copied

0 Copies 1 hexadecimal digit (4 bits)

1 Copies 2 hexadecimal digits (8 bits)

2 Copies 3 hexadecimal digits (12 bits)

3 Copies 4 hexadecimal digits (16 bits)

① Destination: Starting hexadecimal digit position

0 Hexadecimal digit 0

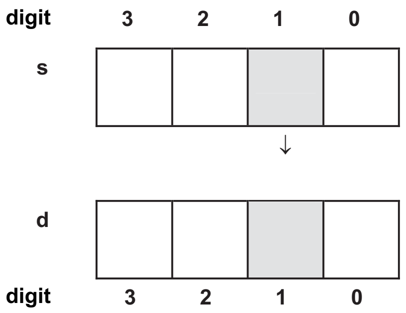
1 Hexadecimal digit 1

2 Hexadecimal digit 2

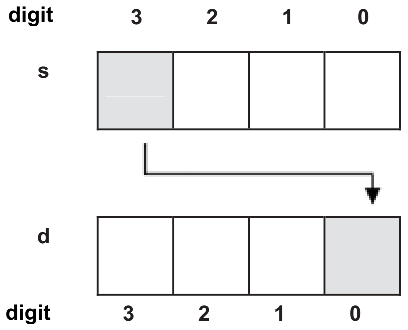
3 Hexadecimal digit 3

Following are some patterns of digit transfer based on the specification of n.

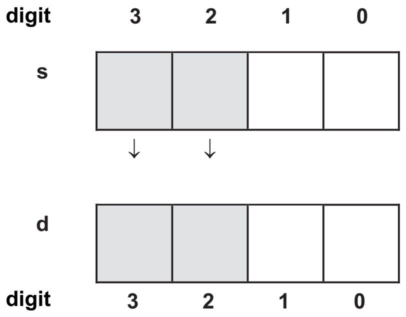
- Specify **n: 16#101** when hexadecimal digit 1 of the source is copied to hexadecimal digit 1 of the destination.



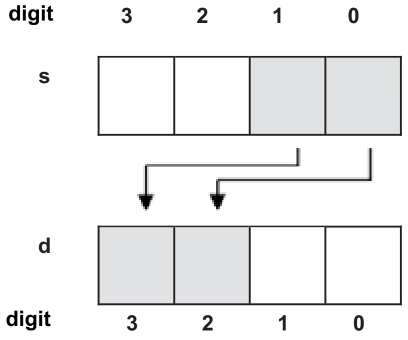
- Specify **n: 16#003** (short form: **16#3**) when hexadecimal digit 3 of the source is copied to hexadecimal digit 0 of the destination.



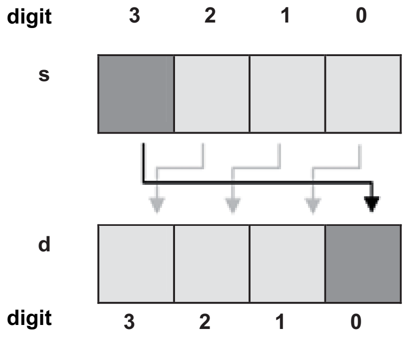
- Specify **n: 16#212** when multiple hexadecimal digits (hexadecimal digits 2 and 3) of the source are copied to multiple hexadecimal digits (hexadecimal digits 2 and 3) of the destination.



- Specify **n: 16#210** when multiple hexadecimal digits (hexadecimal digits 0 and 1) of the source are copied to multiple hexadecimal digits (hexadecimal digits 2 and 3) of the destination.



- Specify **n: 16#130** when 4 hexadecimal digits (hexadecimal digits 0 to 3) of the source are copied to 4 hexadecimal digits (hexadecimal digits 0 to 3) of the destination.



PLC types **Availability of** F6_DGT **(see page** 1325**)**

Data types

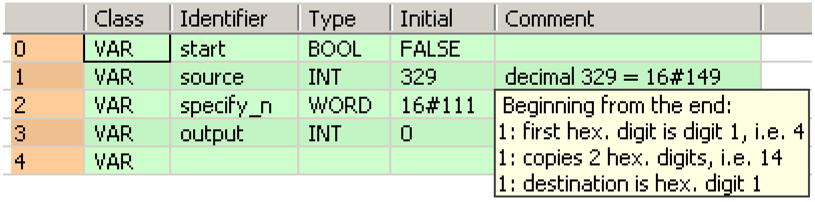
Variable	Data type	Function
s	ANY16	16-bit area source
n		Specifies source and destination hexadecimal digit position and number of hexadecimal digits
d		16-bit area destination

Operands

For	Relay				T/C		Register			Constant
s, n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

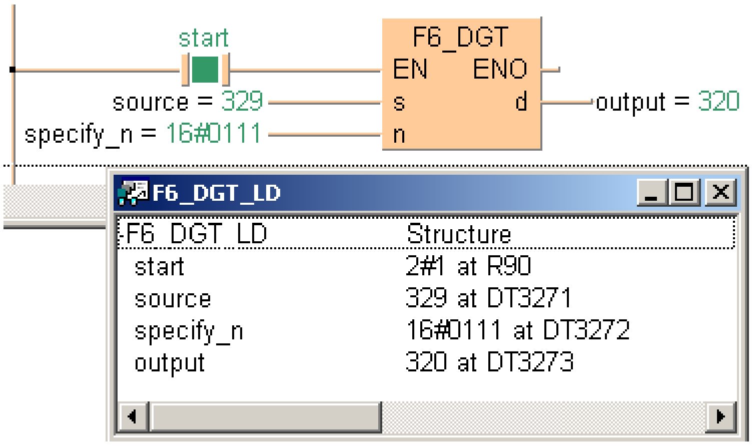


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	
1	VAR	source	INT	329	decimal 329 = 16#149
2	VAR	specify_n	WORD	16#111	
3	VAR	output	INT	0	
4	VAR				

Beginning from the end:
1: first hex. digit is digit 1, i.e. 4
1: copies 2 hex. digits, i.e. 14
1: destination is hex. digit 1

Body When the variable **start** is set to TRUE, the function is carried out. The values for **source** and **output** in the Monitor Header of the ladder diagram body have been set to display the hexadecimal value by activating the Hex button in the tool bar.

LD



F6_DGT_LD	Structure
start	2#1 at R90
source	329 at DT3271
specify_n	16#0111 at DT3272
output	320 at DT3273

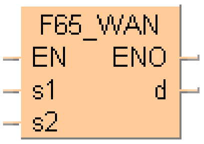
ST When programming with structured text, enter the following:

```
IF start THEN
    F6_DGT( s:= source,
        n:= specify_n,
        d=> output);
END_IF;
```

F65_WAN

16-bit data AND

Description Executes AND operation of each bit in 16-bit equivalent constant or 16-bit data specified by **s1** and **s2** if the trigger EN is in the ON-state. The AND operation result is stored in the 16-bit area specified by **d**. When 16-bit equivalent constant is specified by **s1** or **s2**, the AND operation is performed internally converting it to 16-bit binary expression. You can use this instruction to turn OFF certain bits of the 16-bit data.



Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
s1	0 1 0 0	1 1 0 1	1 0 1 1	1 0 0 1

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
s2	0 0 0 0	0 0 0 0	1 1 1 1	1 1 1 1

start: ON

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
d	0 0 0 0	0 0 0 0	1 0 1 1	1 0 0 1

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F65_WAN **(see page 1326)**

Data types

Variable	Data type	Function
s1, s2	ANY16	16-bit area or 16-bit equivalent constant to be compared
d		16-bit area for storing AND operation result

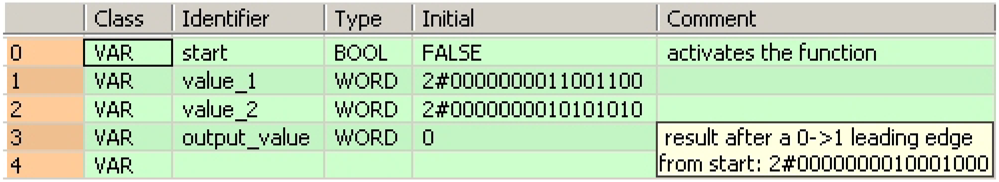
The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

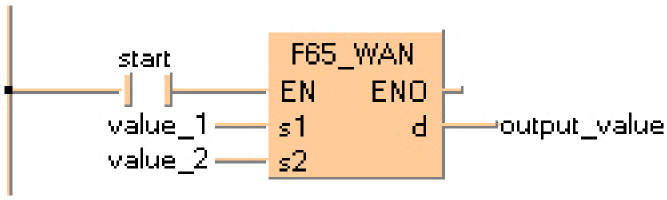
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	WORD	2#0000000011001100	
2	VAR	value_2	WORD	2#0000000010101010	
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#0000000010001000
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



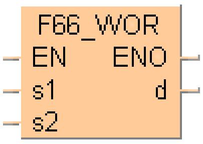
ST When programming with structured text, enter the following:

```
IF start THEN
    F65_WAN(value_1, value_2, output_value);
END_IF;
```

F66_WOR

16-bit data OR

Description Executes OR operation of each bit in 16-bit equivalent constant or 16-bit data specified by **s1** and **s2** if the trigger **EN** is in the ON-state. The OR operation result is stored in the 16-bit area specified by **d**. When 16-bit equivalent constant is specified by **s1** or **s2**, the OR operation is performed internally converting it to 16-bit binary expression. You can use this instruction to turn ON certain bits of the 16-bit data.



Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
s1	0 1 0 0	1 1 0 1	1 0 1 1	1 0 0 1

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
s2	0 0 0 0	0 0 0 0	1 1 1 1	1 1 1 1

start: ON

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
d	0 1 0 0	1 1 0 1	1 1 1 1	1 1 1 1

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F66_WOR **(see page 1326)**

Data types

Variable	Data type	Function
s1, s2	ANY16	16-bit area or 16-bit equivalent constant to be compared
d		16-bit area for storing OR operation result

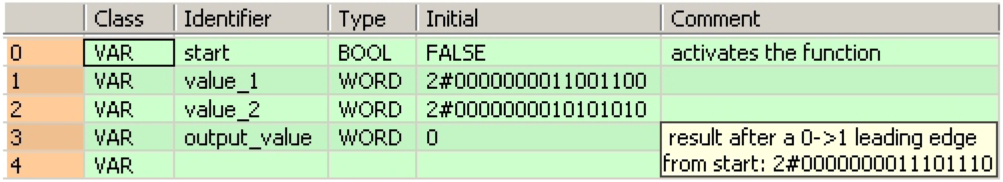
The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

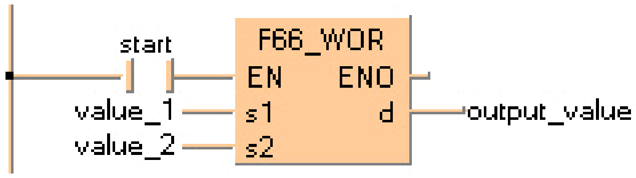
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	WORD	2#0000000011001100	
2	VAR	value_2	WORD	2#0000000010101010	
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#0000000011101110
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



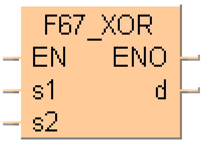
ST When programming with structured text, enter the following:

```
IF start THEN
    F66_WOR(value_1, value_2, output_value);
END_IF;
```

F67_XOR

16-bit data exclusive OR

Description Executes exclusive OR operation of each bit in 16-bit equivalent constant or 16-bit data specified by **s1** and **s2** if the trigger **EN** is in the ON-state. The exclusive OR operation result is stored in the 16-bit area specified by **d**. When 16-bit equivalent constant is specified by **s1** or **s2**, the exclusive OR operation is performed internally converting it to 16-bit binary expression. You can use this instruction to review the number of identical bits in the two 16-bit data.



Bit position	15	·	·	12	11	·	·	8	7	·	·	4	3	·	·	0
s1	0	1	0	0	1	1	0	1	1	0	1	1	1	0	0	1

Bit position	15	·	·	12	11	·	·	8	7	·	·	4	3	·	·	0
s2	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

start: ON

Bit position	15	·	·	12	11	·	·	8	7	·	·	4	3	·	·	0
d	0	1	0	0	1	1	0	1	0	1	0	0	0	1	1	0

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F67_XOR **(see page 1326)**

Data types

Variable	Data type	Function
s1, s2	ANY16	16-bit area or 16-bit equivalent constant to be compared
d		16-bit area for storing XOR operation result

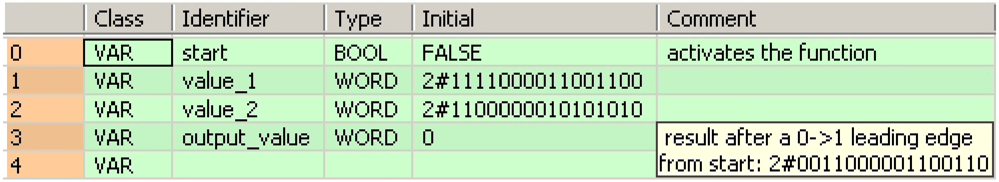
The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

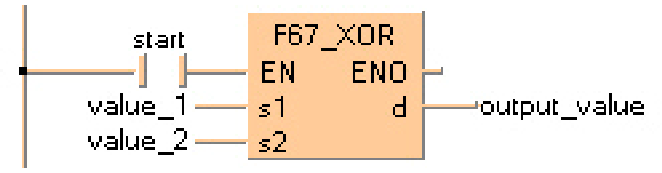
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	WORD	2#1111000011001100	
2	VAR	value_2	WORD	2#1100000010101010	
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#0011000001100110
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



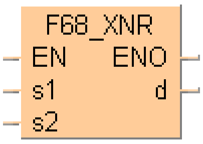
ST When programming with structured text, enter the following:

```
IF start THEN
    F67_XOR(value_1, value_2, output_value);
END_IF;
```

F68_XNR

16-bit data exclusive NOR

Description Executes exclusive NOR operation of each bit in 16-bit equivalent constant or 16-bit data specified by **s1** and **s2** if the trigger **EN** is in the ON-state. The exclusive NOR operation result is stored in the 16-bit area specified by **d**. When 16-bit equivalent constant is specified by **s1** or **s2**, the exclusive NOR operation is performed internally converting it to 16-bit binary expression. You can use this instruction to review the number of identical bits in the two 16-bit data.



Bit position	15	·	·	12	11	·	·	8	7	·	·	4	3	·	·	0
s1	0	1	0	0	1	1	0	1	1	0	1	1	1	0	0	1

Bit position	15	·	·	12	11	·	·	8	7	·	·	4	3	·	·	0
s2	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

start: ON

Bit position	15	·	·	12	11	·	·	8	7	·	·	4	3	·	·	0
d	1	0	1	1	0	0	1	0	1	0	1	1	1	0	0	1

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of F68_XNR (see page 1326)**

Data types

Variable	Data type	Function
s1, s2	ANY16	16-bit area or 16-bit equivalent constant to be compared
d		16-bit area for storing NOR operation result

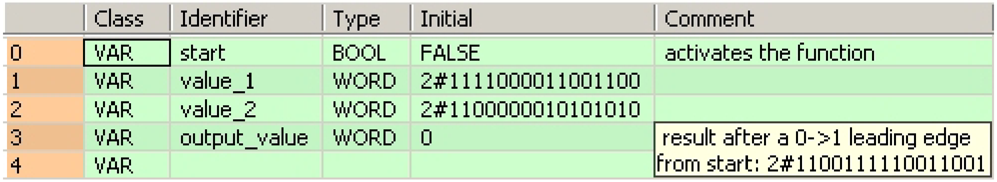
The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

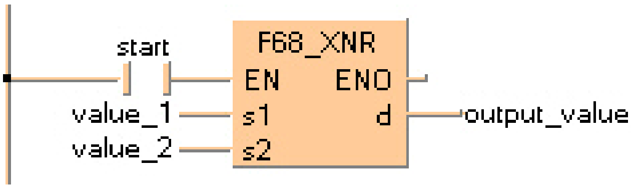
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	WORD	2#1111000011001100	
2	VAR	value_2	WORD	2#1100000010101010	
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#1100111110011001
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



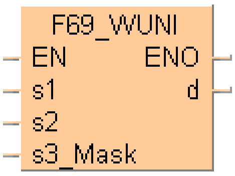
ST When programming with structured text, enter the following:

```
IF start THEN
    F68_XNR(value_1, value_2, output_value);
END_IF;
```

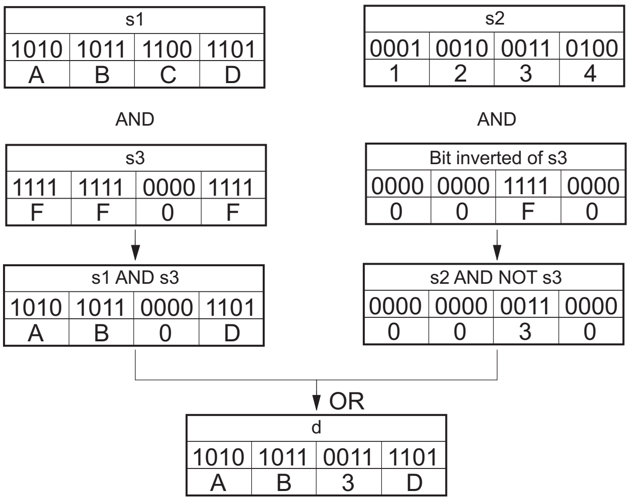
F69_WUNI

16-bit data unite

Description The function combines the two values at inputs **s1** and **s2** with the value at input **s3** by bit-unit processing. The result of the function is returned at output **d**. The data-unite is calculated as follows:



[d] = ([s1] AND [s3]) OR ([s2] AND (NOT[s3]))



s1			
1010	1011	1100	1101
A	B	C	D

s2			
0001	0010	0011	0100
1	2	3	4

AND

AND

s3			
1111	1111	0000	1111
F	F	0	F

Bit inverted of s3			
0000	0000	1111	0000
0	0	F	0

s1 AND s3			
1010	1011	0000	1101
A	B	0	D

s2 AND NOT s3			
0000	0000	0011	0000
0	0	3	0

OR

d			
1010	1011	0011	1101
A	B	3	D

When the value at input **s3** = 16#0, the value at input **s2** is returned at output **d**.

When the value at input **s3** = 16#FFFF, the value at input **s1** is returned at output **d**.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F69_WUNI **(see page 1326)**

Data types

Variable	Data type	Function
s1, s2	ANY16	16-bit area or 16-bit equivalent constant to be compared
s3		16-bit area that stores master data for combination or 16-bit equivalent constant data
d		16-bit area for storing calculated result

The variables **s1**, **s2**, **s3** and **d** have to be of the same data type.

Operands

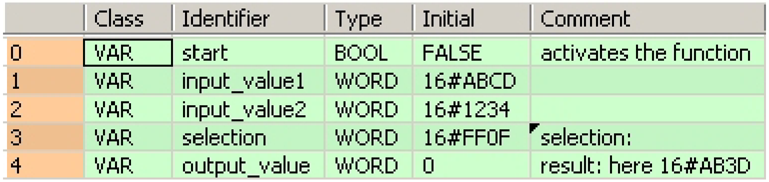
For	Relay				T/C		Register			Constant
s1, s2, s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	the result calculated is 0.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

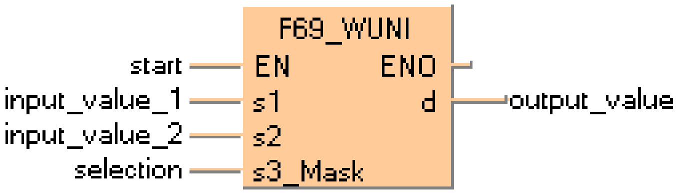


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value1	WORD	16#ABCD	
2	VAR	input_value2	WORD	16#1234	
3	VAR	selection	WORD	16#FF0F	selection:
4	VAR	output_value	WORD	0	result: here 16#AB3D

In this example the input variables **input_value_1, input_value _2** and **selection** are declared. However, you can write constants directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD



ST When programming with structured text, enter the following:

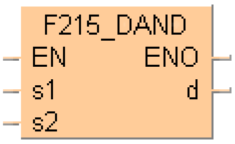
```
IF start THEN
    F69_WUNI( s1:= input_value1,
        s2:= input_value2,
        s3_Mask:= selection,
        d=> output_value);
END_IF;
```

F215_DAND

32-bit data AND

Description

The function performs a bit-wise AND operation on two 32-bit data items at inputs **s1** and **s2**. The result of the function is returned at output **d**.



Truth Table:

s1	s2	d
0	0	0
0	1	0
1	0	0
1	1	1

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F215_DAND **(see page 1323)**

Data types

Variable	Data type	Function
s1	ANY32	32-bit equivalent constant or 32-bit area
s2		32-bit equivalent constant or 32-bit area
d		32-bit area for storing AND operation result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

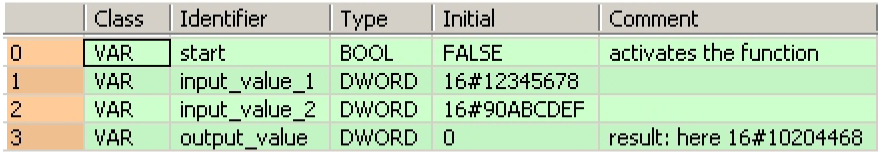
For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the result calculated (output d) is 0.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

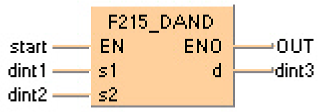


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value_1	DWORD	16#12345678	
2	VAR	input_value_2	DWORD	16#90ABCDEF	
3	VAR	output_value	DWORD	0	result: here 16#10204468

In this example the input variables **input_value_1** and **input_value _2** are declared. However, you can write constants directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD



ST When programming with structured text, enter the following:

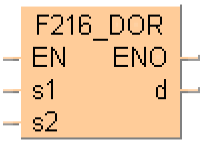
```
IF START THEN
    F215_DAND(dint1, dint2, dint3);
END_IF;
```

F216_DOR

32-bit data OR

Description

The function performs a bit-wise OR operation on two 32-bit data items at inputs **s1** and **s2**. The result of the function is returned at output **d**.



Truth Table:

s1	s2	d
0	0	0
0	1	1
1	0	1
1	1	1

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F216_DOR **(see page 1323)**

Data types

Variable	Data type	Function
s1	ANY32	32-bit equivalent constant or 32-bit area
s2		32-bit equivalent constant or 32-bit area
d		32-bit area for storing OR operation result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

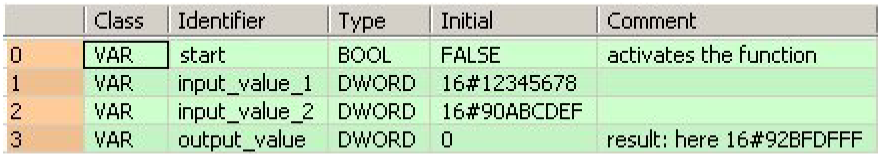
For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the result calculated (output d) is 0.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

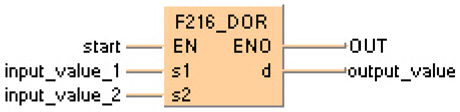


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value_1	DWORD	16#12345678	
2	VAR	input_value_2	DWORD	16#90ABCDEF	
3	VAR	output_value	DWORD	0	result: here 16#92BFDFFF

In this example the input variables **input_value_1** and **input_value _2** are declared. However, you can write constants directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD



ST When programming with structured text, enter the following:

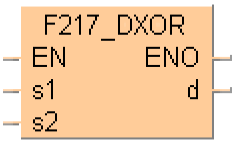
```
IF start THEN
    F216_DOR(input_value_1, input_value_2, output_value);
END_IF;
```

F217_DXOR

32-bit data XOR

Description

The functions a bit-wise exclusive OR operation on two 32-bit data items at inputs **s1** and **s2**. The result of the function is returned at output **d**.



Truth Table:

s1	s2	d
0	0	0
0	1	1
1	0	1
1	1	0

Using this instruction you can check how many bits in the two 32-bit data items are different, for example. At each position in which the bits at inputs **s1** and **s2** are different, a 1 is added in the result.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F217_DXOR **(see page 1323)**

Data types

Variable	Data type	Function
s1	ANY32	32-bit equivalent constant or 32-bit area
s2		32-bit equivalent constant or 32-bit area
d		32-bit area for storing XOR operation result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

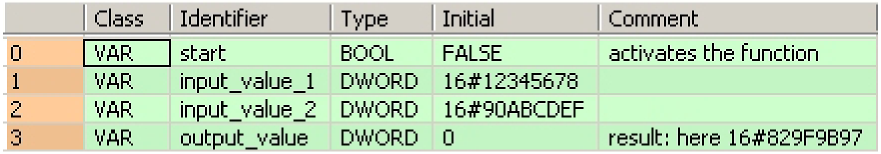
For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the result calculated (output d) is 0.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

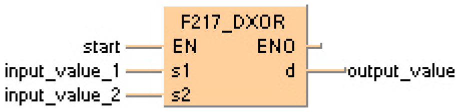


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value_1	DWORD	16#12345678	
2	VAR	input_value_2	DWORD	16#90ABCDEF	
3	VAR	output_value	DWORD	0	result: here 16#829F9B97

In this example the input variables **input_value_1** and **input_value _2** are declared. However, you can write constants directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD



ST When programming with structured text, enter the following:

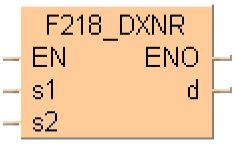
```
IF start THEN
    F217_DXOR(input_value_1, input_value_2, output_value);
END_IF
```

F218_DXNR

32-bit data XNR

Description

The function performs a bit-wise exclusive NOR operation on two 32-bit data items at inputs **s1** and **s2**. The result of the function is returned at output **d**.



Truth Table:

s1	s2	d
0	0	1
0	1	0
1	0	0
1	1	1

Using this instruction you can check how many bits in the two 32-bit data items are the same. At each position in which the bits at inputs **s1** and **s2** match, a 1 is produced in the result.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of F218_DXNR (see page 1323)**

Data types

Variable	Data type	Function
s1	ANY32	32-bit equivalent constant or 32-bit area
s2		32-bit equivalent constant or 32-bit area
d		32-bit area for storing XNR operation result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

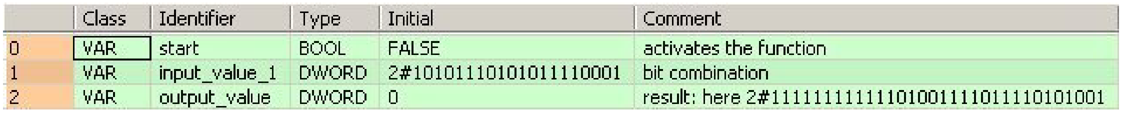
For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the result calculated (output d) is 0.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

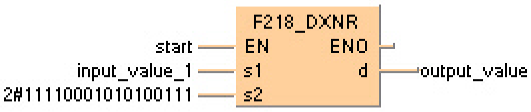
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value_1	DWORD	2#10101110101011110001	bit combination
2	VAR	output_value	DWORD	0	result: here 2#1111111111110100111011110101001

Body When the variable **output** is set to TRUE, the function F218_DXNR is carried out.

LD



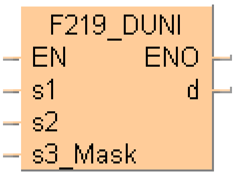
ST When programming with structured text, enter the following:

```
IF start THEN
    F218_DXNR(input_value_1, 2#1111000101010 0111, output_value);
END_IF;
```

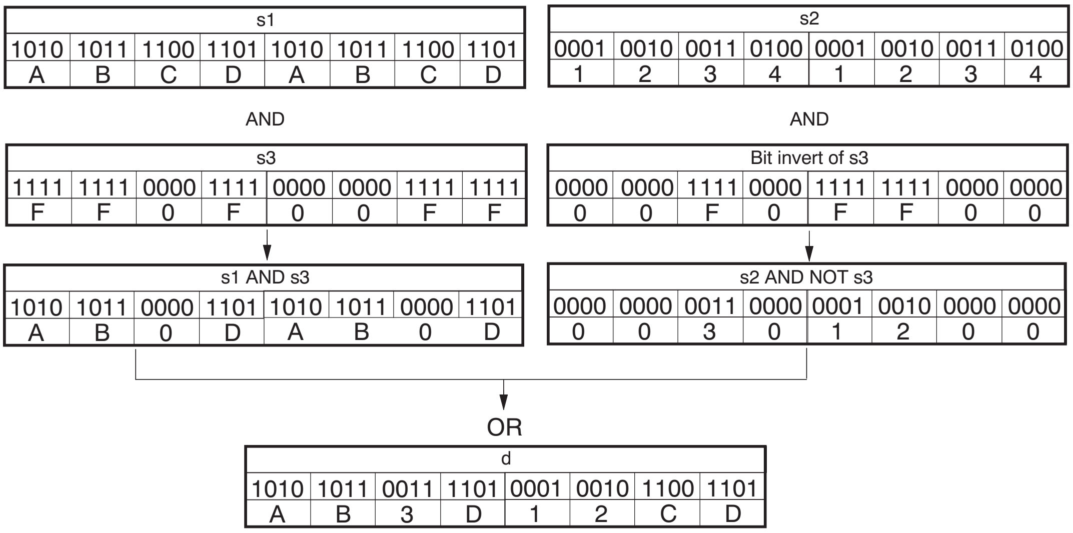
F219_DUNI

32-bit data unites 12

Description The function combines the two values at inputs **s1** and **s2** bit-wise with the value at input **s3**. The result of the function is returned at output **d**. The data-unite is calculated as follows:



[d] = ([s1**] AND [**s3**]) OR ([**s2**] AND (NOT[**s3**]))**



s1							
1010	1011	1100	1101	1010	1011	1100	1101
A	B	C	D	A	B	C	D

s2							
0001	0010	0011	0100	0001	0010	0011	0100
1	2	3	4	1	2	3	4

AND

AND

s3							
1111	1111	0000	1111	0000	0000	1111	1111
F	F	0	F	0	0	F	F

Bit invert of s3							
0000	0000	1111	0000	1111	1111	0000	0000
0	0	F	0	F	F	0	0

s1 AND s3							
1010	1011	0000	1101	1010	1011	0000	1101
A	B	0	D	A	B	0	D

s2 AND NOT s3							
0000	0000	0011	0000	0001	0010	0000	0000
0	0	3	0	1	2	0	0

OR

d							
1010	1011	0011	1101	0001	0010	1100	1101
A	B	3	D	1	2	C	D

When the value at input **s3** = 16#0, then the value at input **s2** is returned at output **d**.

When the value at input **s3** = 16#FFFFFFFF, then the value at input **s1** is returned at output **d**.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F219_DUNI **(see page 1323)**

Data types

Variable	Data type	Function
s1	ANY32	32-bit equivalent constant or 32-bit area
s2		32-bit equivalent constant or 32-bit area
s3		32-bit area that stores master data for combination or 32-bit equivalent constant
d		32-bit area for storing result

The variables **s1, s2, s3** and **d** have to be of the same data type.

Operands

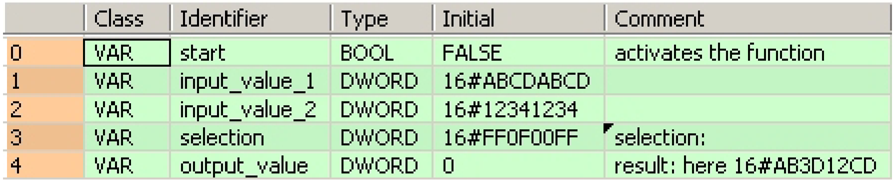
For	Relay				T/C		Register			Constant
s1, s2, s3	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the result calculated (output d) is 0.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

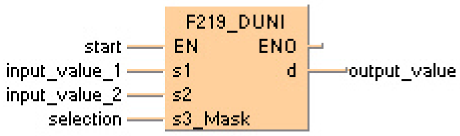


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value_1	DWORD	16#ABCDABCD	
2	VAR	input_value_2	DWORD	16#12341234	
3	VAR	selection	DWORD	16#FF0F00FF	selection:
4	VAR	output_value	DWORD	0	result: here 16#AB3D12CD

In this example the input variables **input_value_1, input_value _2** and **selection** are declared. However, you can write constants directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD



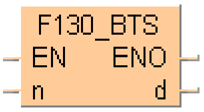
ST When programming with structured text, enter the following:

```
IF start THEN
    F219_DUNI( s1:= input_value1,
        s2:= input_value2,
        s3_Mask:= selection,
        d=> output_value);
END_IF;
```

F130_BTS

16-bit data bit set

Description Turns ON the bit specified by the bit position at **n** of the 16-bit data specified by **d** if the trigger **EN** is in the ON-state. Bits other than the bit specified do not change. The range of **n** is 0 to 15.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F130_BTS **(see page 1321)**

Data types

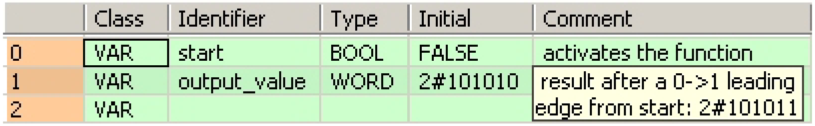
Variable	Data type	Function
d	ANY16	16-bit area
n	INT	specifies bit position to be set

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

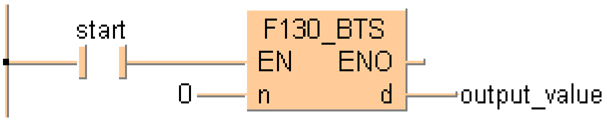
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	output_value	WORD	2#101010	result after a 0->1 leading edge from start: 2#101011
2	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



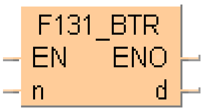
ST When programming with structured text, enter the following:

```
IF start THEN
    F130_BTS( n:= 0,
        d=> output_value);
END_IF;
```

F131_BTR

16-bit data bit reset

Description Turns OFF the bit specified by the bit position at **n** of the 16-bit data specified by **d** if the trigger **EN** is in the ON-state. Bits other than the bit specified do not change. The range of **n** is 0 to 15.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F131_BTR **(see page 1321)**

Data types

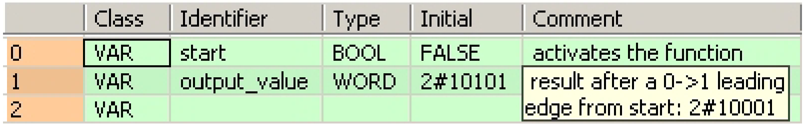
Variable	Data type	Function
d	ANY16	16-bit area
n	INT	specifies bit position to be reset

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

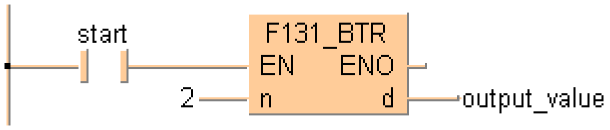
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	output_value	WORD	2#10101	result after a 0->1 leading edge from start: 2#10001
2	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



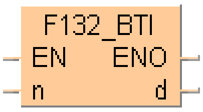
ST When programming with structured text, enter the following:

```
IF start THEN
    F131_BTR( n:= 2,
        d=> output_value);
END_IF;
```

F132_BTI

16-bit data bit invert

Description Inverts [1 (ON) → 0 (OFF) or 0 (OFF) → 1 (ON)] the bit at bit position **n** in the 16-bit data area specified by **d** if the trigger **EN** is in the ON-state. Bits other than the bit specified do not change. The range of **n** is 0 to 15.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F132_BTI **(see page 1321)**

Data types

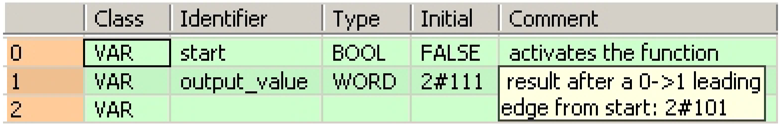
Variable	Data type	Function
d	ANY16	16-bit area
n	INT	specify bit position to be inverted

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

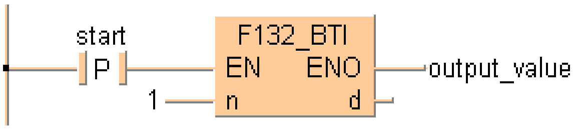
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	output_value	WORD	2#111	result after a 0->1 leading edge from start: 2#101
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD



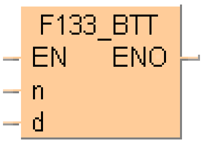
ST When programming with structured text, enter the following:

```
IF DF(start) THEN
    F132_BTI( n:= 1,
        d=> output_value);
END_IF;
```

F133_BTT

16-bit data test

Description Checks the state [1 (ON) or 0 (OFF)] of bit position **n** in the 16-bit data specified by **d** if the trigger **EN** is in the ON-state.



The specified bit is checked by special internal relay R900B.

- When specified bit is 0 (OFF), special internal relay R900B (=flag) turns ON.
- When specified bit is 1 (ON), special internal relay R900B (=flag) turns OFF.

n specifies the bit position to be checked in decimal data.
Range of **n**: 0 to 15

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F133_BTT **(see page 1321)**

Data types

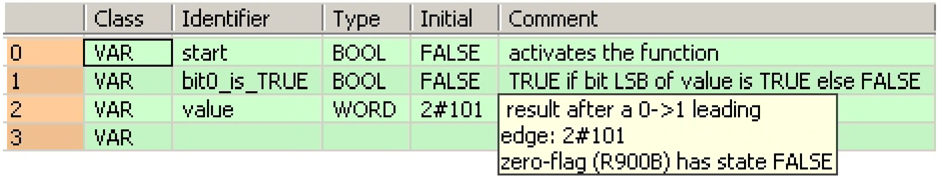
Variable	Data type	Function
d	ANY16	16-bit area
n	INT	specifies bit position to be tested

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

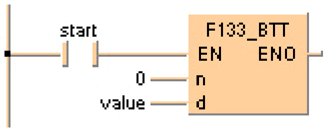
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	bit0_is_TRUE	BOOL	FALSE	TRUE if bit LSB of value is TRUE else FALSE
2	VAR	value	WORD	2#101	result after a 0->1 leading edge: 2#101 zero-flag (R900B) has state FALSE
3	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



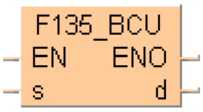
ST When programming with structured text, enter the following:

```
IF start THEN
    F133_BTT( n:= 0,
        d:= value);
    IF R900B THEN
        bit0_is_TRUE := FALSE;
    ELSE
        bit0_is_TRUE := TRUE;
    END_IF;
END_IF;
```

F135_BCU

Number of ON bits in 16-bit data

Description Counts the number of bits in the ON state (1) in the 16-bit data specified by **s** if the trigger **EN** is in the ON-state.



The number of 1 (ON) bits is stored in the 16-bit area specified by **d**.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F135_BCU **(see page 1321)**

Data types

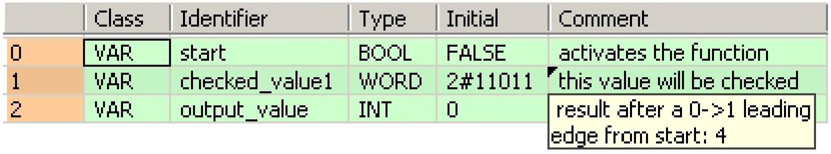
Variable	Data type	Function
s	ANY16	source
d	INT	destination area for storing the number of bits in the ON (1) state

Operands

For	Relay				T/C		Register			Constant
s	-	WY	WR	WL	SV	EV	DT	LD	FL	-
d	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

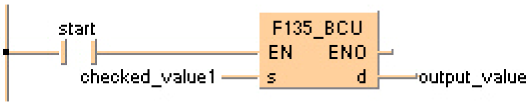
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	checked_value1	WORD	2#11011	this value will be checked
2	VAR	output_value	INT	0	result after a 0->1 leading edge from start: 4

Body When the variable **start** is set to TRUE, the function is carried out.

LD



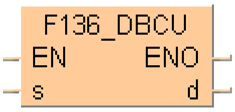
ST When programming with structured text, enter the following:

```
IF start THEN
    F135_BCU(checked_value1, output_value);
END_IF;
```

F136_DBCU

Number of ON bits in 32-bit data

Description Counts the number of bits in the ON state (1) in the 32-bit data specified by **s** if the trigger **EN** is in the ON-state.



The number of 1 (ON) bits is stored in the 16-bit area specified by **d**.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F136_DBCU **(see page** 1321**)**

Data types

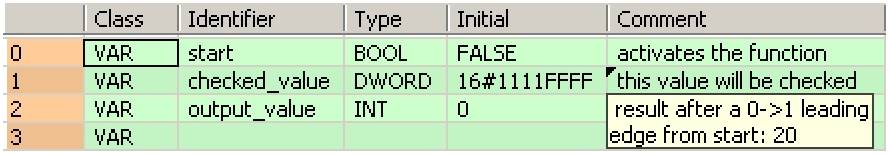
Variable	Data type	Function
s	ANY32	source
d	INT	destination area for storing the number of bits in the ON (1) state

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

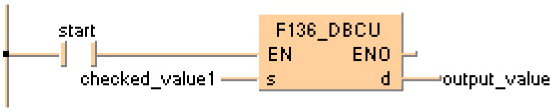
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	checked_value	DWORD	16#1111FFFF	this value will be checked
2	VAR	output_value	INT	0	result after a 0->1 leading edge from start: 20
3	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



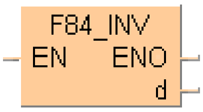
ST When programming with structured text, enter the following:

```
IF start THEN
    F136_DBCU(checked_value, output_value);
END_IF;
```

F84_INV

16-bit data invert (one's complement)

Description Inverts each bit (0 or 1) of the 16-bit data specified by **d** if the trigger **EN** is in the ON-state. The inverted result is stored in the 16-bit area specified by **d**. This instruction is useful for controlling an external device that uses negative logic operation.



Destination

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
d	0 1 0 1	1 1 1 0	1 0 1 1	1 1 0 1

⬇ start: ON

Destination

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
d	1 0 1 0	0 0 0 1	0 1 0 0	0 0 1 0

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F84_INV **(see page 1326)**

Data types

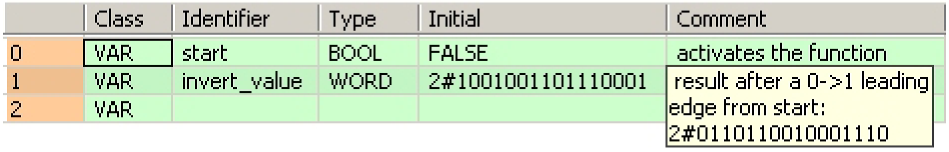
Variable	Data type	Function
d	ANY16	16-bit area to be inverted

Operands

For	Relay				T/C		Register			Const.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

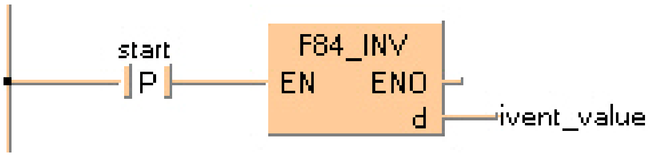
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	invert_value	WORD	2#1001001101110001	result after a 0->1 leading edge from start: 2#0110110010001110
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD



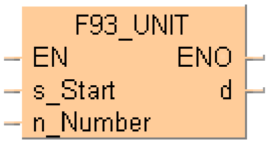
ST When programming with structured text, enter the following:

```
IF DF(start) THEN
    F84_INV(invert_value);
END_IF;
```

F93_UNIT

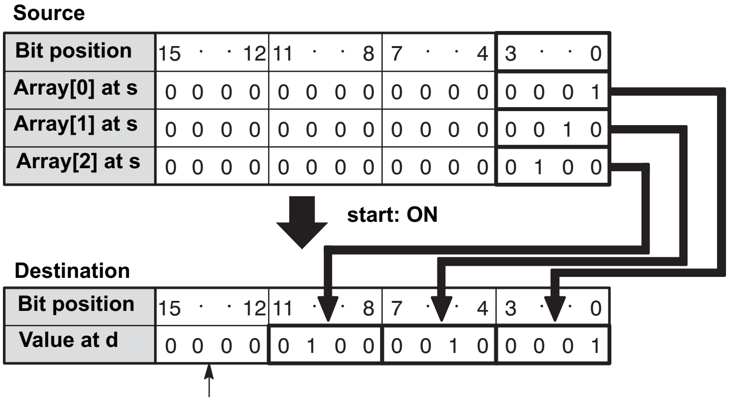
16-bit data combine

Description Extracts each lower 4 bits (bit position 0 to 3) starting with the 16-bit area specified by **s** and combines the extracted data into 1 word if the trigger **EN** is in the ON-state. The result is stored in the 16-bit area specified by **d**.



n specifies the number of data to be extracted. The range of **n** is 0 to 4.

The programming example provided below can be envisioned thus:



Source

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
Array[0] at s	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1
Array[1] at s	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0
Array[2] at s	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0

Destination

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
Value at d	0 0 0 0	0 1 0 0	0 0 1 0	0 0 0 1

Bit positions 12 to 15 are filled with 0s.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F93_UNIT **(see page 1326)**

Data types

Variable	Data type	Function
s	WORD	starting 16-bit area to be extracted (source)
n	INT	specifies number of data to be extracted
d	WORD	16-bit area for storing combined data (destination)

Operands

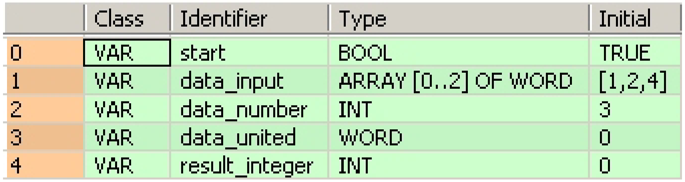
For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the area specified using the index modifier exceeds the limit ▪ the value at n ≥ 5
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

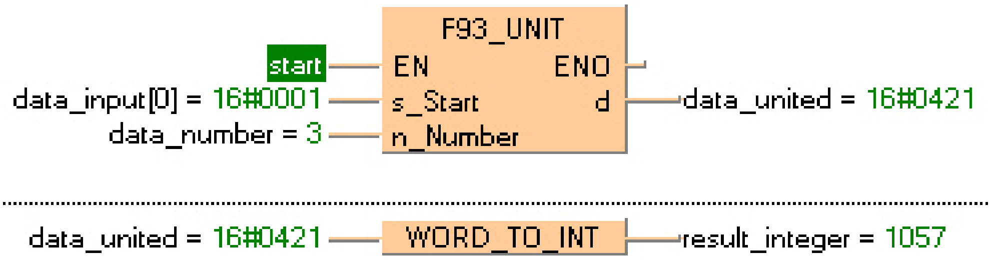
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	start	BOOL	TRUE
1	VAR	data_input	ARRAY [0..2] OF WORD	[1,2,4]
2	VAR	data_number	INT	3
3	VAR	data_united	WORD	0
4	VAR	result_integer	INT	0

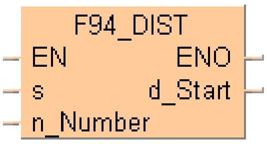
Body When the variable **start** is set to TRUE, the function is carried out. The binary values in the illustration on the main help page serve as the array values in **data_input**. In this example, variables are declared in the POU header. However, you may assign constants directly at the input function's contact pins instead.

LD In this example, (Monitoring) was activated so you can see the results immediately.



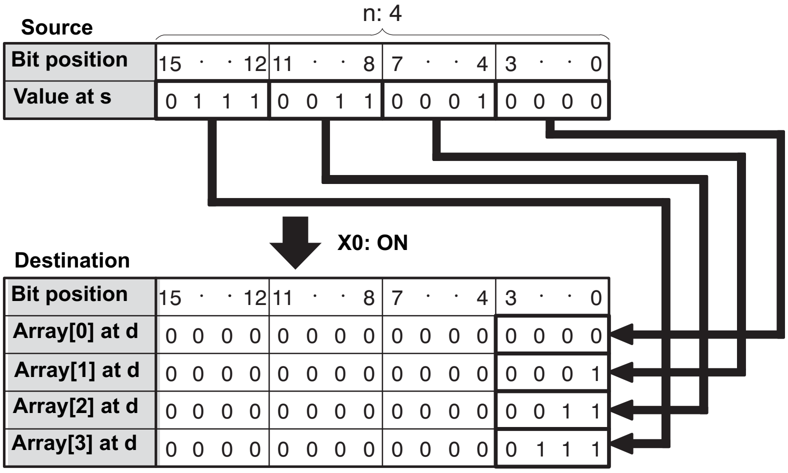
F94_DIST 16-bit data distribution

Description Divides the 16-bit data specified by **s** into 4-bit units and distributes the divided data into the lower 4 bits (bit position 0 to 3) of 16-bit areas starting with **d** if the trigger **EN** is in the ON-state.



n specifies the number of data to be divided. The range of **n** is 0 to 4. When 0 is specified by **n**, this instruction is not executed.

The programming example provided below can be envisioned thus:



Source

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
Value at s	0 1 1 1	0 0 1 1	0 0 0 1	0 0 0 0

Destination

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
Array[0] at d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
Array[1] at d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1
Array[2] at d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1
Array[3] at d	0 0 0 0	0 0 0 0	0 0 0 0	0 1 1 1

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F94_DIST **(see page 1326)**

Data types

Variable	Data type	Function
s	WORD	16-bit area or equivalent constant to be divided (source)
n	INT	specifies number of data to be divided
d	WORD	starting 16-bit area for storing divided data (destination)

Operands

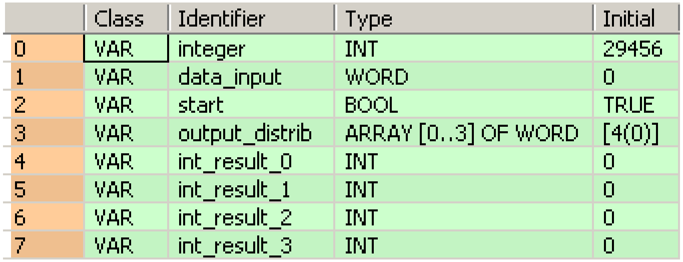
For	Relay				T/C		Register			Constant
s, n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the area specified using the index modifier exceeds the limit ▪ the value at n ≥ 5 the last area for the result exceeds the limit
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

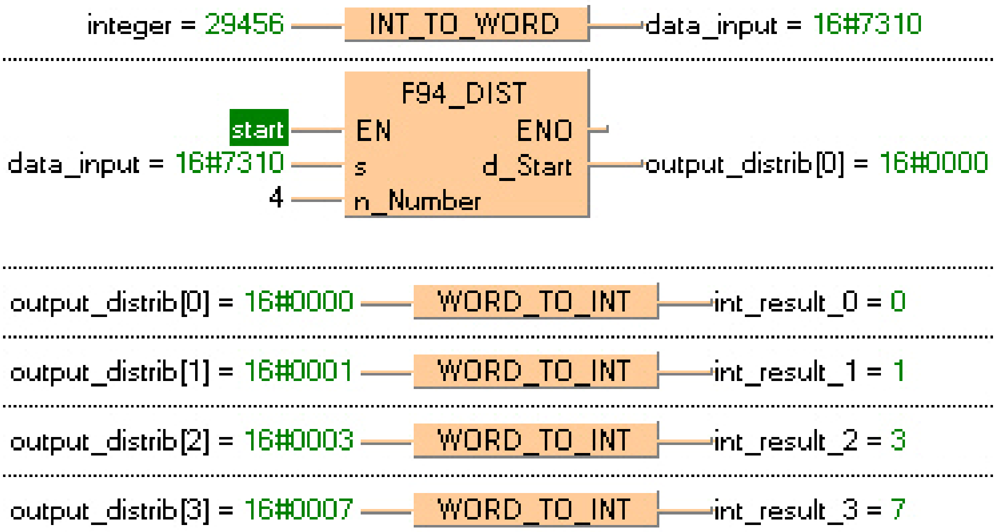
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	integer	INT	29456
1	VAR	data_input	WORD	0
2	VAR	start	BOOL	TRUE
3	VAR	output_distrib	ARRAY [0..3] OF WORD	[4(0)]
4	VAR	int_result_0	INT	0
5	VAR	int_result_1	INT	0
6	VAR	int_result_2	INT	0
7	VAR	int_result_3	INT	0

Body When the variable **start** is set to TRUE, the function is carried out. The binary values in the illustration on main help page serve as the values calculated. In this example, variables are declared in the POU header. Also, a constant value of 4 is assigned directly at the contact pin for **n_Number**.

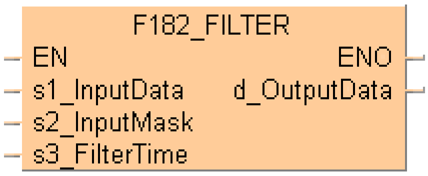
LD In this example, (Monitoring) was activated so you can see the results immediately.



F182_FILTER

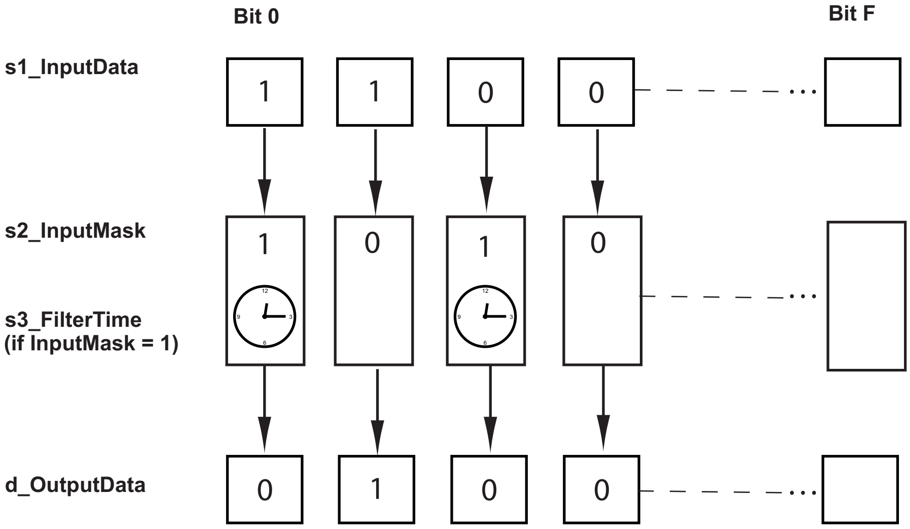
Time constant processing

Description Filter processing is executed for specified bits and output bitwise. The instruction can be useful to negate the effects of bounce, e.g. for a switching device.



For bits stored in the area specified by **s1_InputData** a debouncing is executed if the resulting value for **s2_InputMask** is "1". The result of the debouncing operation is output to **d_OutputData.** The debouncing time is defined via **s3_FilterTime** (0 to 30000ms). If **s2_InputMask** is "0" no debouncing takes place and the corresponding bit at **s1_InputData** passes unchanged to **d_OutputData**.

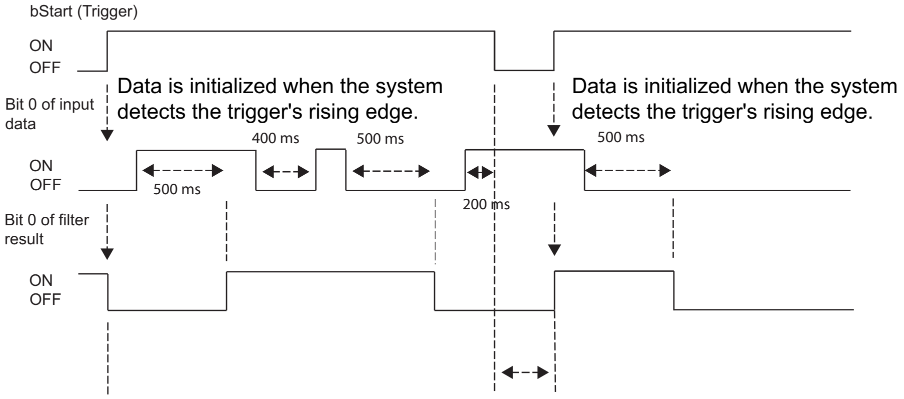
In the following figure, the bits in **d_OutputData** and their values will be the same as **s1_InputData** after the filter time has elapsed or, for example, if no masking takes place, e.g. **s2_InputMask** is assigned the value 0.



Pre-cautions during programming

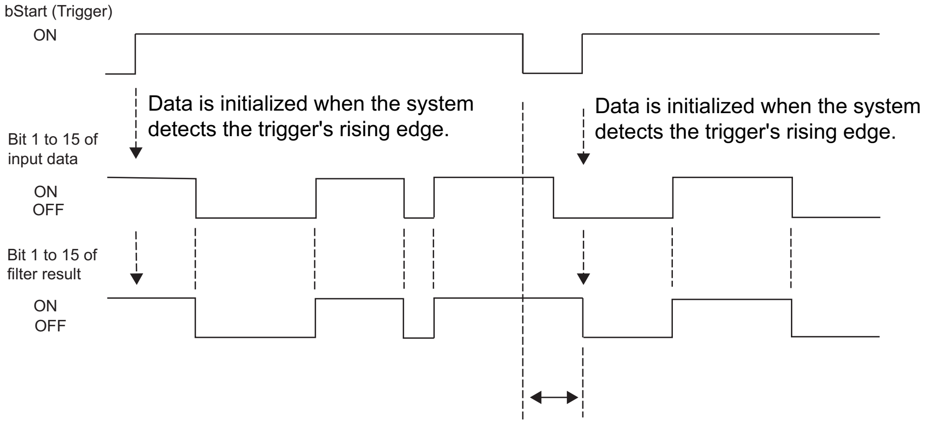
When the system detects a trigger's rising edge, all the bits of the input specified by **s1_InputData** are output directly in **d_OutputData** and the effects of bounce are not prevented. A scan time error may occur during filter processing, for a maximum of 1 scan.

Time charts for the filter when the value assigned to s2_InputMask **is 1 (16#0001), i.e. bit 0 will be filtered, the other bits will not be filtered, and the value assigned to** s3_FilterTime **is 500ms.**



The instruction does not operate while the trigger is OFF.

Time chart when the value assigned to s2_InputMask **is 0 (16#0000), i.e. bit 0 to F will be not filtered**



The instruction does not operate while the trigger is OFF.

PLC types **Availability of** F182_FILTER **(see page 1322)**

Data types

Variable	Data type	Function
s1_InputData	ANY16	Input data whose bits will be filtered according to the input mask
s2_InputMask		Input mask which specifies which bits will be filtered
s3_FilterTime		Specifies the minimum off- and on-time in ms
d_OutputData		Filtered data

Operands

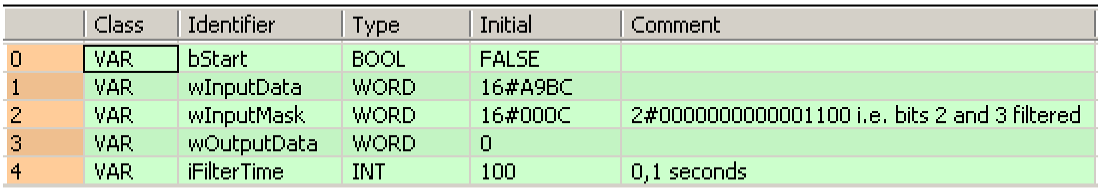
For	Relay				T/C		Register			Const.
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2, s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ The filter processing time specified by **s3_FilterTime** is less than 0 or greater than 30000.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.

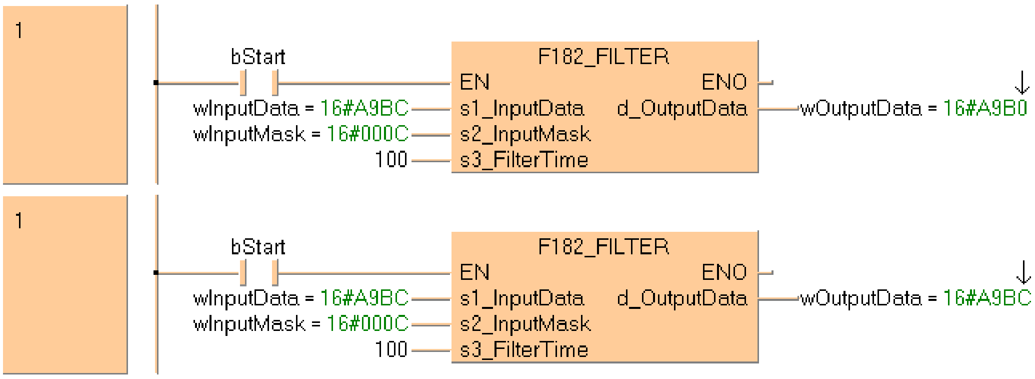


	Class	Identifier	Type	Initial	Comment
0	VAR	bStart	BOOL	FALSE	
1	VAR	wInputData	WORD	16#A9BC	
2	VAR	wInputMask	WORD	16#000C	2#0000000000001100 i.e. bits 2 and 3 filtered
3	VAR	wOutputData	WORD	0	
4	VAR	iFilterTime	INT	100	0,1 seconds

In this example, the input variables **wInputData**, **wInputMask** and **iFilterTime** are declared. However, for **wInputMask** and **iFilterTime**, you can write a constant directly at the input contact of the function instead. Additionally, the variable **bStart** is declared to start the filter function and the variable **wOutputData** is declared for storing the result.

Body The filtered bits will only be written to **wOutputData** after the filter time has elapsed (see LD example). See time charts (see page 550) for a detailed explanation. **wOutputdata** has the value 16#A9B0 for 100ms, when this time has been elapsed **wOutputData** has the value 16#A9BC.

LD



ST When programming with structured text, enter the following:

```
IF bStart Then
      F182_FILTER(wInputData, wInputMask, iFilterTime, wOutputData);
End_If;
```

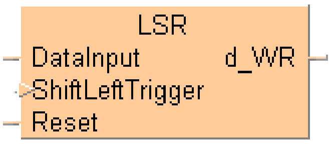
Chapter 18

Bit-shift instructions

LSR

Left shift register

Description Shifts 1 bit of the specified data area (**d_WR**) to the left (to the higher bit position). When programming the **LSR** instruction, be sure to program the data input (**DataInput**), shift (**ShiftLeftTrigger**) and reset triggers (**Reset**).



DataInput: specifies the state of new shift-in data:

- new shift-in data 1: when the input is ON
- new shift-in data 0: when the input is OFF

ShiftLeftTrigger: shifts 1 bit to the left when the leading edge of the trigger is detected

Reset: turns all the bits of the data area to 0 if the trigger is in the ON-state

The area available for this instruction is only the word internal relay (WR).

PLC types **Availability of LSR (see page 1328)**

☞ **Word internal relay (WR) number range, depends on the free area in the Extras → Options → Compile Options → Address Ranges menu.**

Data types

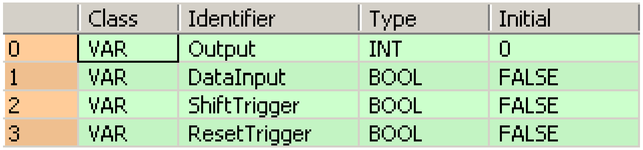
Variable	Data type	Function
DataInput	BOOL	when ON, shift-in data = 1, when OFF, shift-in data = 0
ShiftLeftTrigger	BOOL	shifts one bit to the left when ON
Reset	BOOL	resets data area to 0 when ON
d_WR	ANY16	specified data area where data shift takes place

Operands

For	Relay				T/C	
DataInput, ShiftLeftTrigger, Reset	X	Y	R	L	T	C
d_WR	-	-	WR	-	-	-

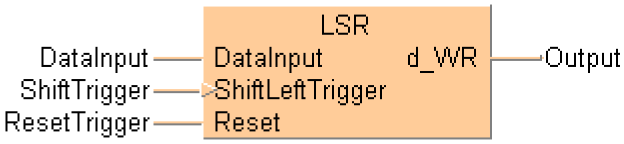
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Output	INT	0
1	VAR	DataInput	BOOL	FALSE
2	VAR	ShiftTrigger	BOOL	FALSE
3	VAR	ResetTrigger	BOOL	FALSE

Body



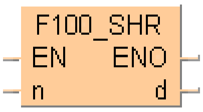
ST When programming with structured text, enter the following:

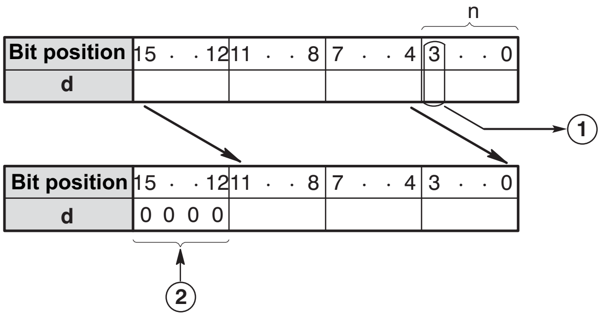
```
Output:=LSR(DataInput, ShiftTrigger, ResetTrigger);
```

F100_SHR

Right shift of 16-bit data in bit units

Description Shifts **n** bits of 16-bit data area specified by **d** to the right (to the lower bit position) if the trigger **EN** is in the ON-state.





When **n** bits are shifted to the right, the data in the **n**th bit ① is transferred to special internal relay R9009 (carry-flag) and the higher **n** bits of the 16-bit data area ② specified by **d** are filled with 0s.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F100_SHR **(see page 1320)**

Data types

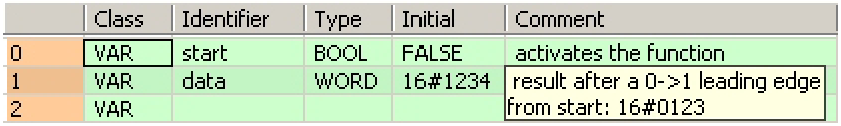
Variable	Data type	Function
d	ANY16	16-bit area to be shifted to the right
n	INT	number of bits to be shifted

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

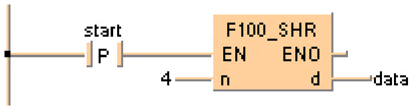
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	WORD	16#1234	result after a 0->1 leading edge from start: 16#0123
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD



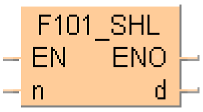
ST When programming with structured text, enter the following:

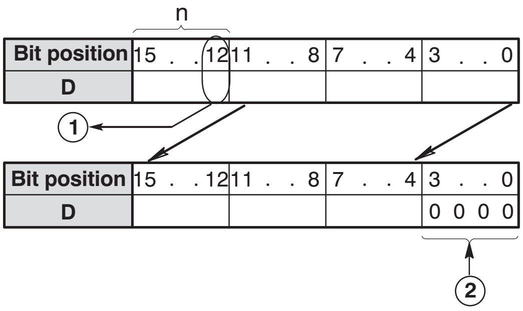
```
IF DF(start) THEN
    F100_SHR( n:= 4 ,
        d=> data );
END_IF;
```

F101_SHL

Left shift of 16-bit data in bit units

Description Shifts **n** bits of 16-bit data area specified by **d** to the left (to the higher bit position) if the trigger **EN** is in the ON-state.





When **n** bits are shifted to the left, the data in the **n**th bit ① is transferred to special internal relay R9009 (carry-flag) and **n** bits ② starting with bit position 0 are filled with 0s.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of F101_SHL (see page 1320)**

Data types

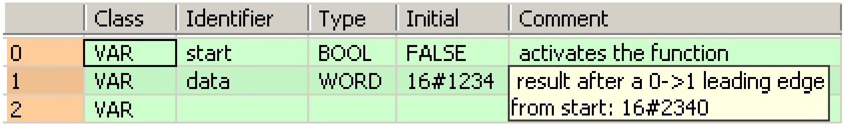
Variable	Data type	Function
d	ANY16	16-bit area to be shifted to the left
n	INT	number of bits to be shifted

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

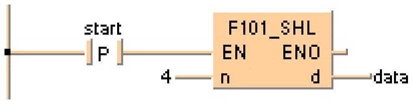
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	WORD	16#1234	result after a 0->1 leading edge from start: 16#2340
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD



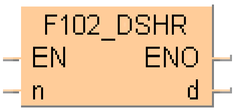
ST When programming with structured text, enter the following:

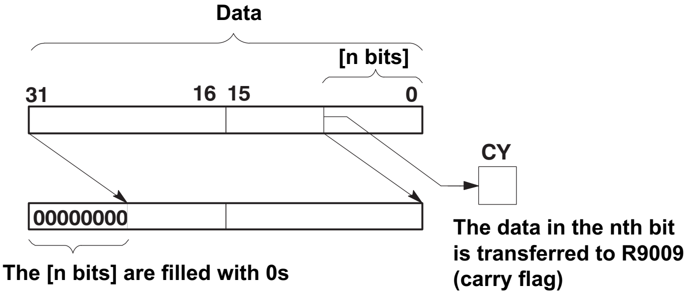
```
IF DF(start) THEN
    F101_SHL( n:= 4,
        d=> data);
END_IF;
```

F102_DSHR

Right shift of 32-bit data in bit units

Description The function shifts the value at output **d** to the right. The number of bits at output **d** to be shifted to the right is specified by the value assigned at input **n**. This shift can lie between 0 and 255 (only the lower value byte of **n** is effective). Bits cleared because of the shift become 0. When input **n** = 0, no shift takes place. A shifting distance larger than 32 does not make sense, since when **n** = 32 the value at output **d** is already filled with zeros. The bit at position **n** - 1 (the last bit shifted out to the right) is simultaneously stored in special internal relay R9009 (carry flag) so that it can be evaluated accordingly. When **n** = 0 the content of the carry flag does not change.




Data
[n bits]
31
16
15
0
CY
00000000
The [n bits] are filled with 0s
The data in the nth bit is transferred to R9009 (carry flag)

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F102_DSHR **(see page 1320)**

Data types

Variable	Data type	Function
n	INT	number of bits to be shifted (range: 16#0 to 16#FF)
d	ANY32	32-bit area to be shifted to the right

Operands

For	Relay				T/C		Register			Constant
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

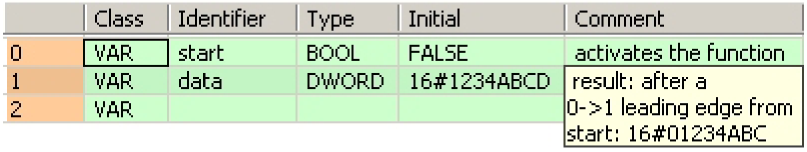
Error flags

No.	IEC address	Set	If
R9009	%MX0.900.9	for an instant	▪ the bit at position n - 1 has the value 1.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

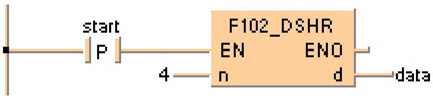


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	DWORD	16#1234ABCD	result: after a 0->1 leading edge from start: 16#01234ABC
2	VAR				

Body

When the variable **start** changes from FALSE to TRUE, the function is carried out. It shifts out 4 bits (corresponds to one position in a hexadecimal representation) to the right. The 4 bits in **data** resulting from the shift are filled with zeros. At input n the constant 4 is assigned directly to the function. You may, however, declare an input variable in the POU header instead.

LD



ST

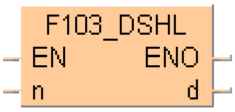
When programming with structured text, enter the following:

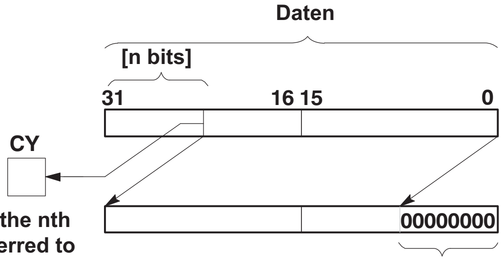
```
IF DF(start) THEN
    F102_DSHR( n:= 4 ,
        d=> data);
END_IF;
```

F103_DSHL

Left shift of 32-bit data in bit units

Description The function rotates the value at output **d** to the left. The number of bits at output **d** to be shifted to the left is specified by the value assigned at input **n**. This shift can lie between 0 and 255 (only the lower value byte of **n** is effective). Bits cleared because of the shift become 0. When input **n** = 0, no shift takes place. A shifting distance larger than 32 does not make sense, since when **n** = 32 the value at output **d** is already filled with zeros. The bit at position 31 - **n** (the last bit shifted out to the left) is simultaneously stored in special internal relay R9009 (carry flag) so that it can be evaluated accordingly. When **n** = 0 the content of the carry flag does not change.





The data in the nth bit is transferred to R9009 (carry flag).

[n bits] starting from bit position 0 are filled with 0s.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F103_DSHL **(see page 1320)**

Data types

Variable	Data type	Function
n	INT	number of bits to be shifted (range: 16#0 to 16#FF)
d	ANY32	32-bit area to be shifted to the left

Operands

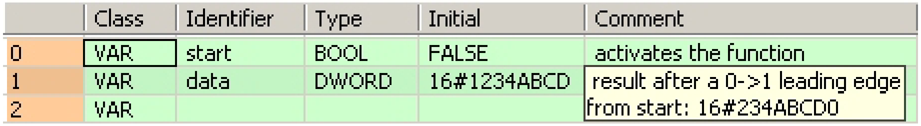
For	Relay				T/C		Register			Constant
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9009	%MX0.900.9	for an instant	▪ the bit at position 31 - n has the value 1.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

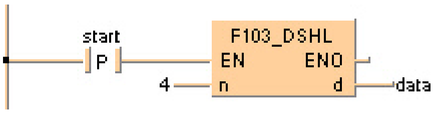
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	DWORD	16#1234ABCD	result after a 0->1 leading edge from start: 16#234ABCD0
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out. It shifts out 4 bits (corresponds to one position in a hexadecimal representation) to the left. The 4 bits in **data** resulting from the shift are filled with zeros. At input n the constant 4 is assigned directly to the function. You may, however, declare an input variable in the POU header instead.

LD



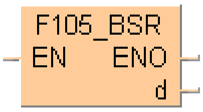
ST When programming with structured text, enter the following:

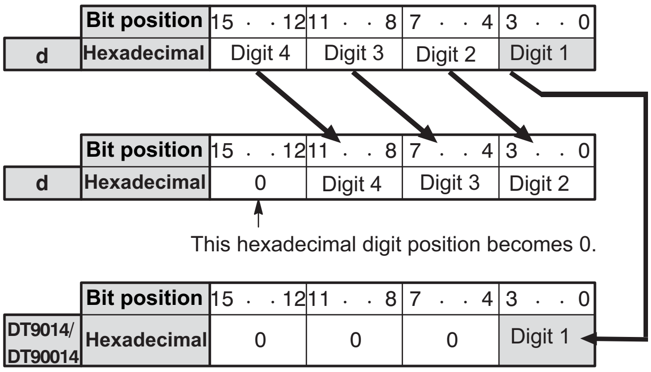
```
IF DF(start) THEN
    F103_DSHL( n:= 4,
        d=> data);
END_IF;
```

F105_BSR

Right shift of one hexadecimal digit (4 bits) of 16-bit data

Description Shifts one hexadecimal digit (4 bits) of the 16-bit area specified by **d** to the right (to the lower digit position) if the trigger **EN** is in the ON-state.





When one hexadecimal digit (4 bits) is shifted to the right,

- hexadecimal digit position 0 (bit position 0 to 3) of the data specified by **d** is shifted out and is transferred to the lower digit (bit position 0 to 3) of special data register DT9014 (DT90014 for FP2/2SH and FP10/10S/10SH).
- hexadecimal digit position 3 (bit position 12 to 15) of the 16-bit area specified by **d** becomes 0.
- This instruction is useful when the hexadecimal or BCD data is handled.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F105_BSR **(see page 1320)**

Data types

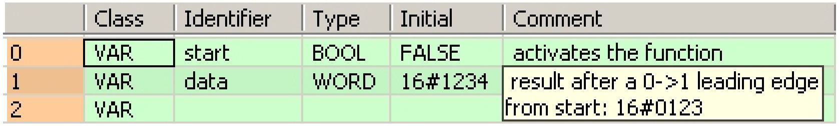
Variable	Data type	Function
d	ANY16	16-bit area to be shifted to the right

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

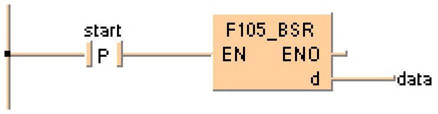
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	WORD	16#1234	result after a 0->1 leading edge from start: 16#0123
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD



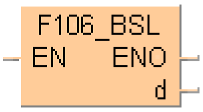
ST When programming with structured text, enter the following:

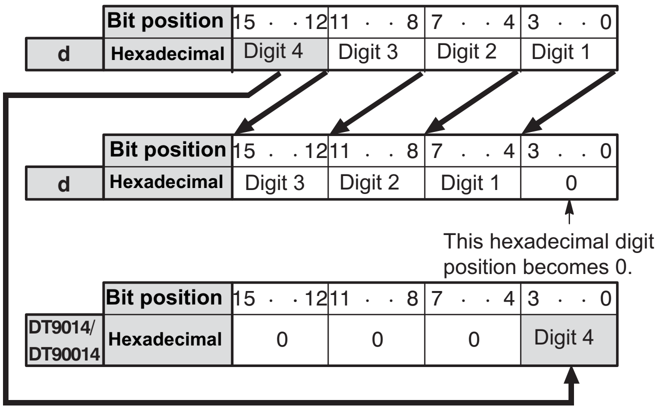
```
IF DF(start) THEN
    F105_BSR(data);
END_IF;
```

F106_BSL

Left shift of one hexadecimal digit (4 bits) of 16-bit data

Description Shifts one hexadecimal digit (4 bits) of the 16-bit area specified by **d** to the left (to the higher digit position) if the trigger **EN** is in the ON-state.





- When one hexadecimal digit (4 bits) is shifted to the left,
- hexadecimal digit position 3 (bit position 12 to 15) of the data specified by **d** is shifted out and is transferred to the lower digit (bit position 0 to 3) of special data register DT9014 (DT90014 for FP2/2SH and FP10/10S/10SH).
- hexadecimal digit position 0 (bit position 0 to 3) of the 16-bit area specified by **d** becomes 0.

This instruction is useful when the hexadecimal or BCD data is handled.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F106_BSL **(see page 1320)**

Data types

Variable	Data type	Function
d	ANY16	16-bit area to be shifted to the left

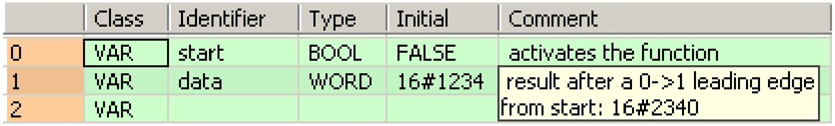
Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

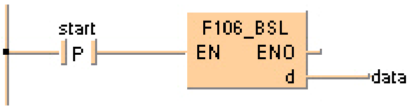
All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	WORD	16#1234	result after a 0->1 leading edge from start: 16#2340
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD



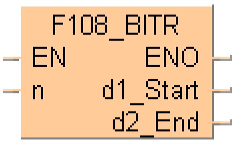
ST When programming with structured text, enter the following:

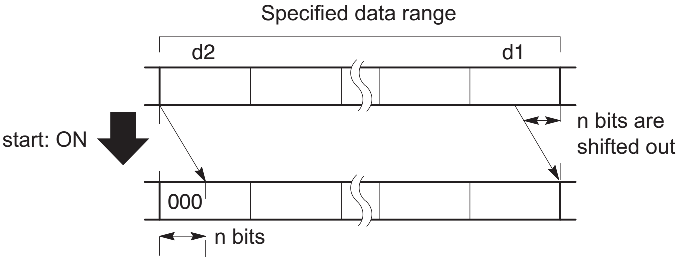
```
IF DF(start) THEN
    F106_BSL(data);
END_IF;
```

F108_BITR

Right shift of multiple bits of 16-bit data range

Description The function shifts the bits of a specified data range, whose beginning and end are specified by the outputs **d1** and **d2** to the right. The number of bits by which the data range is to be shifted to the right is specified by the value assigned at input **n**. The value may lie between 0 and 16. Bits cleared because of the shift become 0. When input **n** = 0, no shift takes place. When input **n** = 16, a shift of one WORD occurs, i.e. the same process takes place as with function F110_WHSL (see page 573).





This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F108_BITR **(see page 1320)**

Data types

Variable	Data type	Function
d1	ANY16	starting 16-bit area
d2		ending 16-bit area
n	INT	number of bits to be shifted

The addresses of the variables at inputs **d1** and **d2** have to have the same address type.

Operands

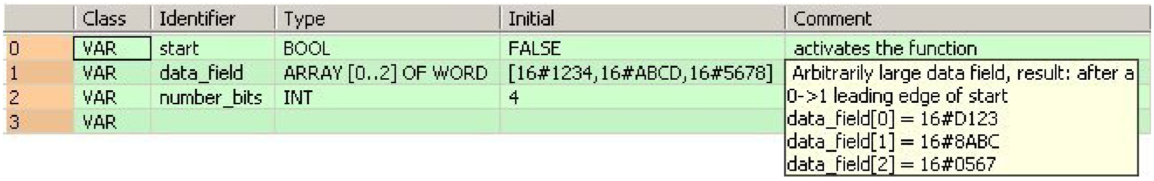
For	Relay				T/C		Register			Constant
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the address of the variables at the outputs **d1** > **d2** or the value at input is **n** ≥ 16.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

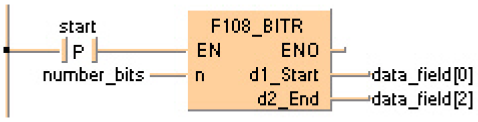


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..2] OF WORD	[16#1234,16#ABCD,16#5678]	Arbitrarily large data field, result: after a 0->1 leading edge of start data_field[0] = 16#D123 data_field[1] = 16#8ABC data_field[2] = 16#0567
2	VAR	number_bits	INT	4	
3	VAR				

In this example, the input variable **number_bits** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** changes from FALSE to TRUE, the function is carried out. It shifts out 4 bits (corresponds to one position in a hexadecimal representation) to the right. The 4 bits in **data_field[2]** resulting from the shift are filled with zeros.

LD



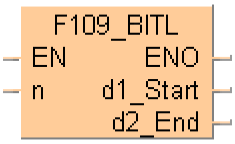
ST When programming with structured text, enter the following:

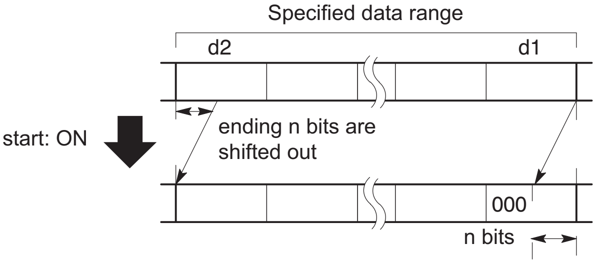
```
IF DF(start) THEN
    F108_BITR( n:=number_bits,
        d1_Start=> data_field[0],
        d2_End=> data_field[2]);
END_IF;
```

F109_BITL

Left shift of multiple bits of 16-bit data range

Description The function shifts the bits of a specified data range, whose beginning and end are specified by the outputs **d1** and **d2** to the left. The number of bits by which the data range is to be shifted to the left is specified by the value assigned at input **n**. The value may lie between 0 and 16. Bits cleared because of the shift become 0. When input **n** = 0, no shift takes place. When input **n** = 16, a shift of one WORD occurs, i.e. the same process takes place as with function F111_WSHL (see page 575).





This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F109_BITL **(see page 1320)**

Data types

Variable	Data type	Function
d1	ANY16	starting 16-bit area
d2		ending 16-bit area
n	INT	number of bits to be shifted

The addresses of the variables at inputs **d1** and **d2** have to have the same address type.

Operands

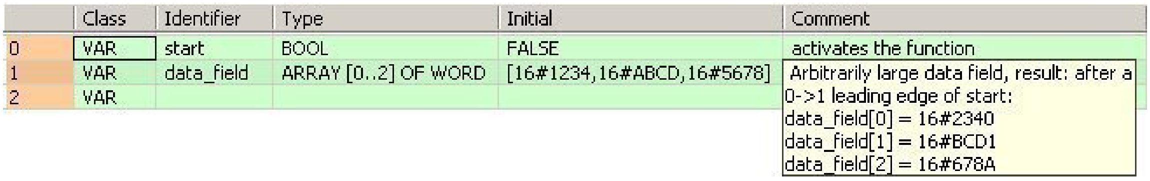
For	Relay				T/C		Register			Constant
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the address of the variables at the outputs **d1** > **d2** or the value at input is **n** ≥ 16.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

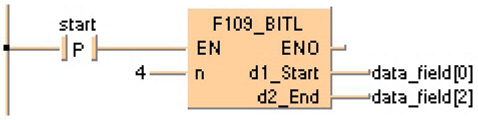
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field	ARRAY [0..2] OF WORD	[16#1234,16#ABCD,16#5678]	Arbitrarily large data field, result: after a 0->1 leading edge of start: data_field[0] = 16#2340 data_field[1] = 16#BCD1 data_field[2] = 16#678A
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out. It shifts out 4 bits (corresponds to one position in a hexadecimal representation) to the left. The 4 bits in **data_field[0]** resulting from the shift are filled with zeros. At input **n** the constant 4 is assigned directly to the function. You may, however, declare an input variable in the POU header instead.

LD



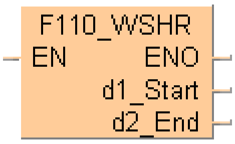
ST When programming with structured text, enter the following:

```
IF DF(start) THEN
    F109_BITL( n:=4,
        d1_Start=> data_field[0],
        d2_End=> data_field[2]);
END_IF;
```

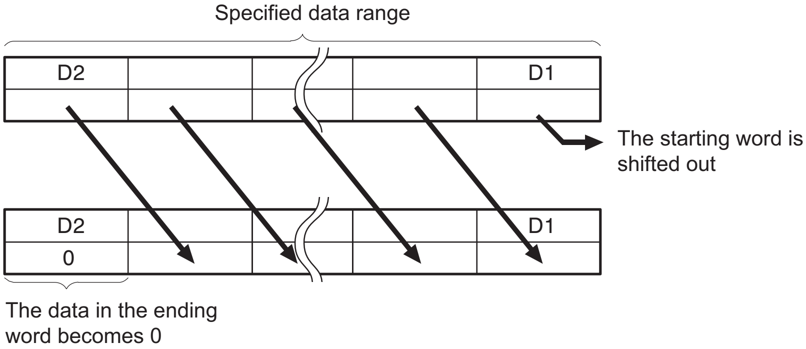
F110_WSHR

Right shift of one word (16 bits) of 16-bit data range

Description Shifts one word (16 bits) of the data range specified by **d1** (starting) and **d2** (ending) to the right (to the lower word address) if the trigger **EN** is in the ON-state.



When one word (16 bits) is shifted to the right, the starting word is shifted out and the data in the ending word becomes 0.


Specified data range
D2
D1
The starting word is shifted out
D2
0
D1
The data in the ending word becomes 0

d1 and **d2** should be:

- the same type of operand
- **d1 ≤ d2**

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of F110_WSHR (see page 1320)**

Data types

Variable	Data type	Function
d1	ANY16	starting 16-bit area
d2		ending 16-bit area

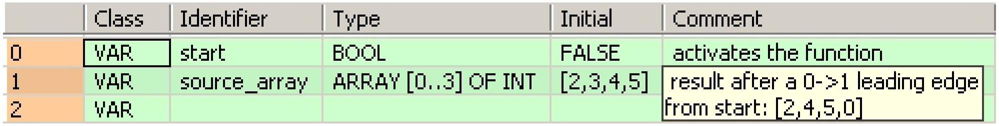
The variables **d1** and **d2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

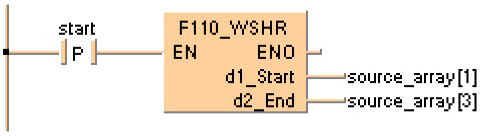
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source_array	ARRAY [0..3] OF INT	[2,3,4,5]	result after a 0->1 leading edge from start: [2,4,5,0]
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD

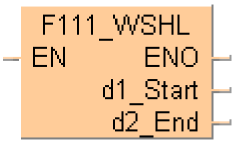


ST When programming with structured text, enter the following:

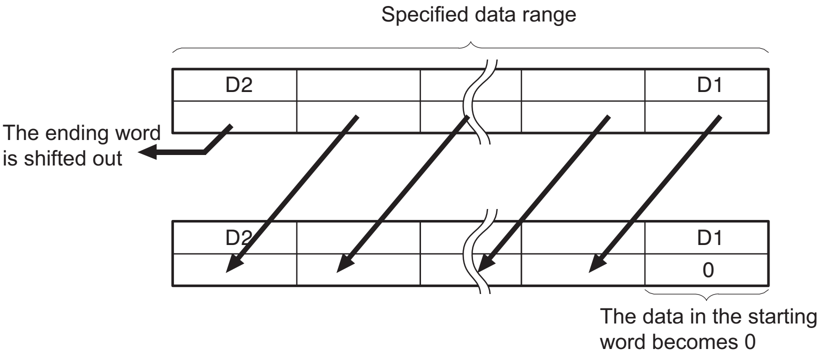
```
IF DF(start) THEN
    F110_WSHR( d1_Start=> source_array[1],
        d2_End=> source_array[3]);
END_IF;
```

F111_WSHL Left shift of one word (16 bits) of 16-bit data range

Description Shifts one word (16 bits) of the data range specified by **d1** (starting) and **d2** (ending) to the left (to the higher word address) if the trigger **EN** is in the ON-state.



When one word (16 bits) is shifted to the left, the ending word is shifted out and the data in the starting word becomes 0.


Specified data range
D2
D1
The ending word is shifted out
D2
D1
0
The data in the starting word becomes 0

d1 and **d2** should be:

- the same type of operand
- **d1 ≤ d2**

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F111_WSHL **(see page 1320)**

Data types

Variable	Data type	Function
d1	ANY16	starting 16-bit area
d2		ending 16-bit area

The variables **d1** and **d2** have to be of the same data type.

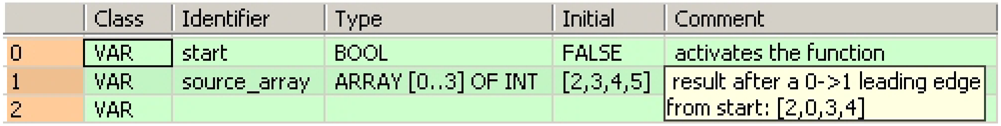
Operands

For	Relay				T/C		Register			Constant
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

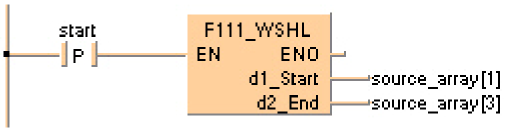
All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source_array	ARRAY [0..3] OF INT	[2,3,4,5]	result after a 0->1 leading edge from start: [2,0,3,4]
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD



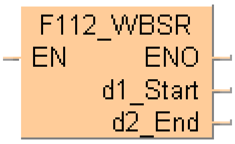
ST When programming with structured text, enter the following:

```
IF DF(start) THEN
    F111_WSHL( d1_Start=> source_array[1],
        d2_End=> source_array[3]);
END_IF;
```

F112_WBSR

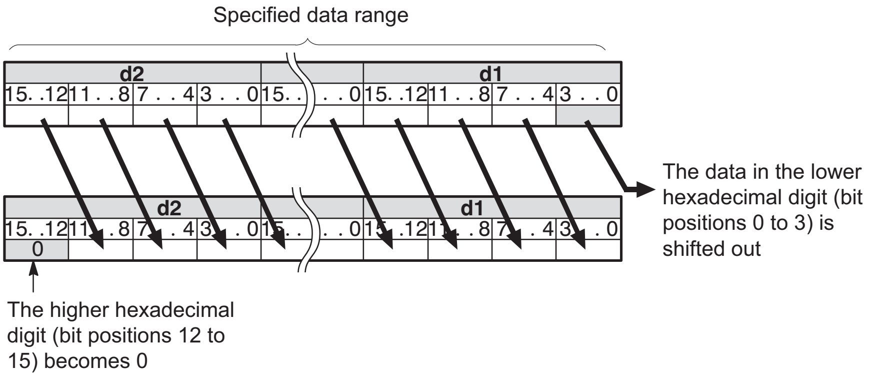
Right shift of one hex. digit (4 bits) of 16-bit 5 data range

Description Shifts one hexadecimal digit (4 bits) of the data range specified by **d1** (starting) and **d2** (ending) to the right (to the lower digit position) if the trigger **EN** is in the ON-state.



When one hexadecimal digit (4 bits) is shifted to the right:

- the data in the lower hexadecimal digit (bit position 0 to 3) of the 16-bit data specified by **d1** is shifted out.
- the data in the higher hexadecimal digit (bit position 12 to 15) of the 16-bit data specified by **d2** becomes 0.


Specified data range
d2
d1
15. .12
11 . . 8
7 . . 4
3 . . 0
15. .
. . 0
0
The data in the lower hexadecimal digit (bit positions 0 to 3) is shifted out
The higher hexadecimal digit (bit positions 12 to 15) becomes 0

d1 and **d2** should be:

- the same type of operand
- **d1 ≤ d2**

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of F112_WBSR (see page 1320)**

Data types

Variable	Data type	Function
d1	ANY16	starting 16-bit area
d2		ending 16-bit area

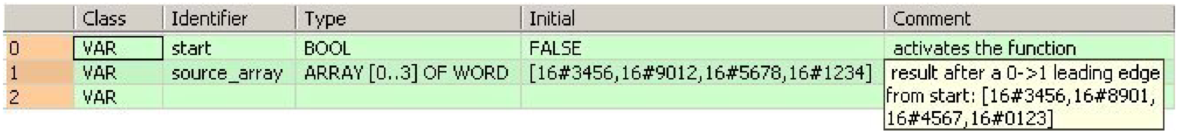
The variables **d1** and **d2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

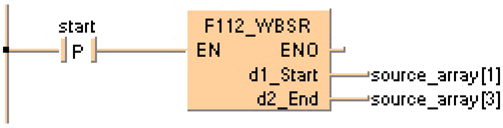
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source_array	ARRAY [0..3] OF WORD	[16#3456,16#9012,16#5678,16#1234]	result after a 0->1 leading edge from start: [16#3456,16#8901, 16#4567,16#0123]
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD



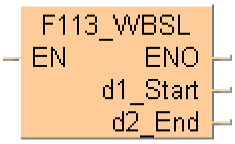
ST When programming with structured text, enter the following:

```
IF DF(start) THEN
    F112_WBSR( d1_Start=> source_array[1],
        d2_End=> source_array[3]);
END_IF;
```

F113_WBSL

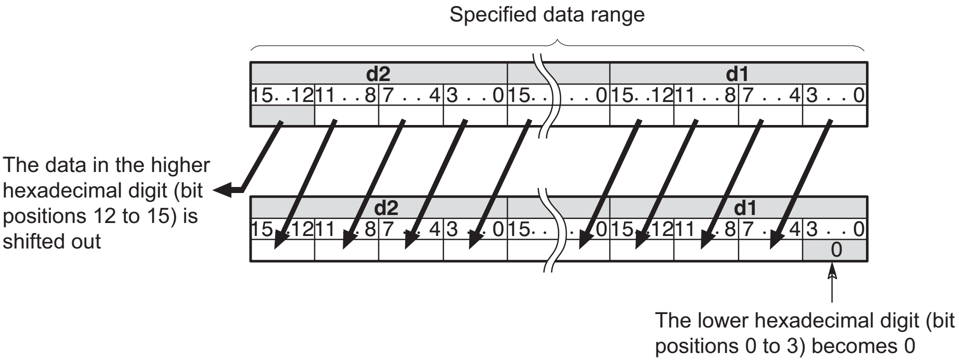
Left shift of one hex. digit (4 bits) of 16-bit data range

Description Shifts one hexadecimal digit (4 bits) of the data range specified by **d1** (starting) and **d2** (ending) to the left (to the higher digit position) if the trigger **EN** is in the ON-state.



When one hexadecimal digit (4 bits) is shifted to the left,

- the data in the higher hexadecimal digit (bit position 12 to 15) of the 16-bit data specified by **d2** is shifted out.
- the data in the lower hexadecimal digit (bit position 0 to 3) of the 16-bit data specified by **d1** becomes 0.


Specified data range
d2
15. .12
11 . . 8
7 . . 4
3 . . 0
15. .
. . 0
d1
15. .12
11 . . 8
7 . . 4
3 . . 0
The data in the higher hexadecimal digit (bit positions 12 to 15) is shifted out
d2
d1
0
The lower hexadecimal digit (bit positions 0 to 3) becomes 0

d1 and d2 should be:

- the same type of operand
- **d1 ≤ d2**

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F113_WBSL **(see page 1320)**

Data types

Variable	Data type	Function
d1	ANY16	starting 16-bit area
d2		ending 16-bit area

The variables **d1** and **d2** have to be of the same data type.

Operands

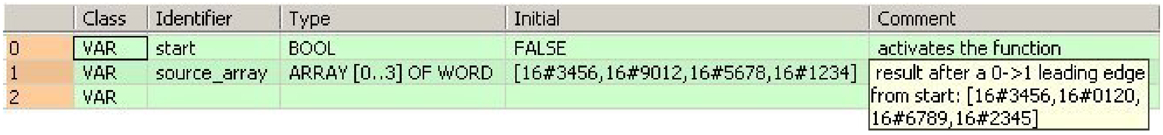
For	Relay				T/C		Register			Constant
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example

In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

All input and output variables used for programming this function have been declared in the POU header.

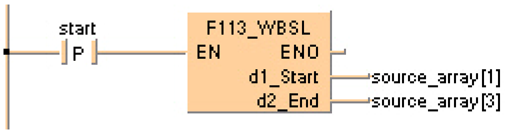


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source_array	ARRAY [0..3] OF WORD	[16#3456,16#9012,16#5678,16#1234]	result after a 0->1 leading edge from start: [16#3456,16#0120, 16#6789,16#2345]
2	VAR				

Body

When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD



ST

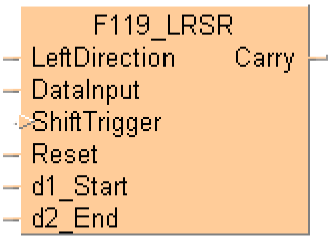
When programming with structured text, enter the following:

```
IF DF(start) THEN
    F112_WBSR( d1_Start=> source_array[1],
        d2_End=> source_array[3]);
END_IF;
```

F119_LRSR

LEFT/RIGHT shift register

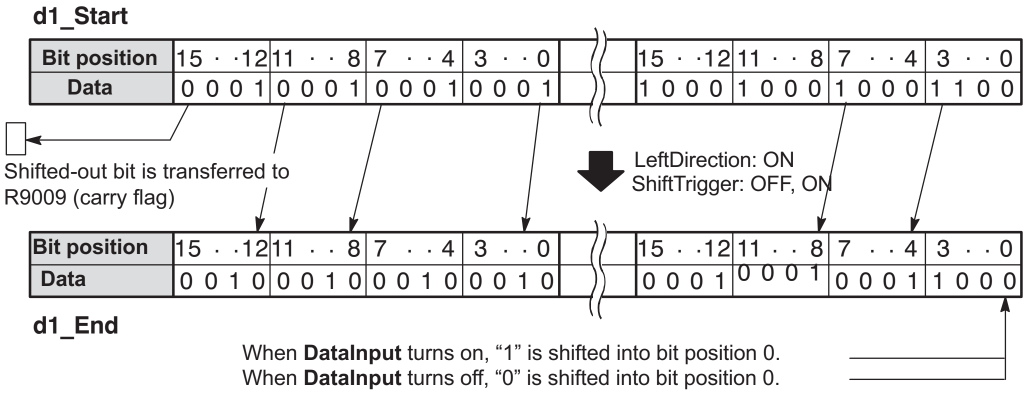
Description Shifts 1 bit of the 16-bit data range to the left or to the right.



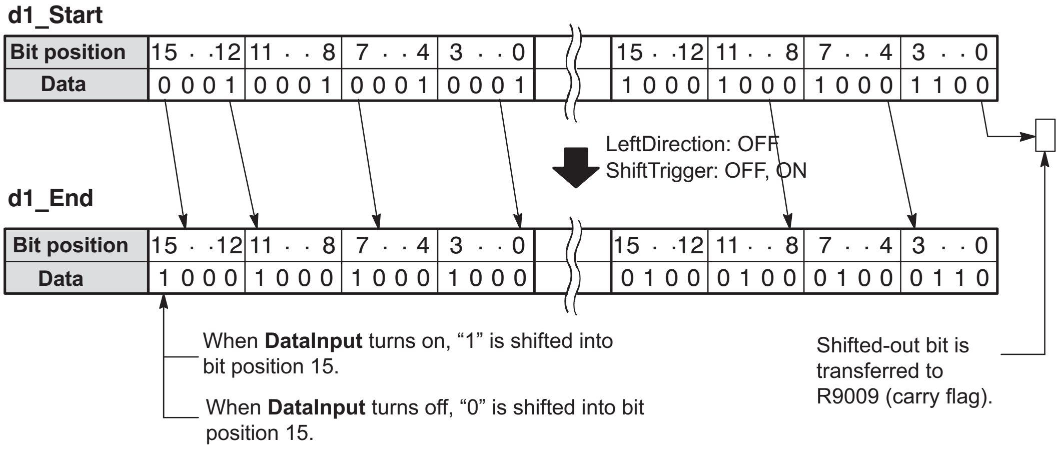
Left/right shift is a shift register which shifts 1 bit of the specified data area to the left (to the higher bit position) or to the right (to the lower bit position).

LeftDirection	Left/right trigger; specifies the direction of the shift-out.	
LeftDirection	= TRUE	shifting out to the left.
LeftDirection	= FALSE	shifting out to the right.
DataInput	Specifies the new shift-in data.	
	New shift-in data = TRUE:	when the data input is in the TRUE-state.
	New shift-in data = FALSE:	when the data input is in the FALSE-state.
ShiftTrigger	Shifts 1 bit to the left or right when the rising edge of the trigger is detected (FALSE → TRUE).	
Reset	Turns all the bits of the data range specified by **d1_Start** and **d2_End** to 0 if this trigger is in the TRUE-state.	
d1_Start	Start of 16-bit area.	
d2_End	End of 16-bit area.	
Carry	Shifted-out bit.	

Left shift operation


d1_Start
Bit position | 15 · ·12 | 11 · · 8 | 7 · · 4 | 3 · · 0 | 15 · ·12 | 11 · · 8 | 7 · · 4 | 3 · · 0
Data | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 1 0 0
Shifted-out bit is transferred to R9009 (carry flag)
LeftDirection: ON
ShiftTrigger: OFF, ON
Bit position | 15 · ·12 | 11 · · 8 | 7 · · 4 | 3 · · 0 | 15 · ·12 | 11 · · 8 | 7 · · 4 | 3 · · 0
Data | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 1 0 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 1 0 0 0
d1_End
When **DataInput** turns on, “1” is shifted into bit position 0.
When **DataInput** turns off, “0” is shifted into bit position 0.

Right shift operation


d1_Start
Bit position | 15 · ·12 | 11 · · 8 | 7 · · 4 | 3 · · 0 | 15 · ·12 | 11 · · 8 | 7 · · 4 | 3 · · 0
Data | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 1 0 0
LeftDirection: OFF
ShiftTrigger: OFF, ON
d1_End
Bit position | 15 · ·12 | 11 · · 8 | 7 · · 4 | 3 · · 0 | 15 · ·12 | 11 · · 8 | 7 · · 4 | 3 · · 0
Data | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 1 0 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 0 0 | 0 1 1 0
When **DataInput** turns on, “1” is shifted into bit position 15.
When **DataInput** turns off, “0” is shifted into bit position 15.
Shifted-out bit is transferred to R9009 (carry flag).

PLC types ☞

Availability of F119_LRSR **(see page 1320)**

- **The variables 'd1 and d2' have to be of the same data type.**
- **This function does not require a variable at the output "Carry".**

Data types

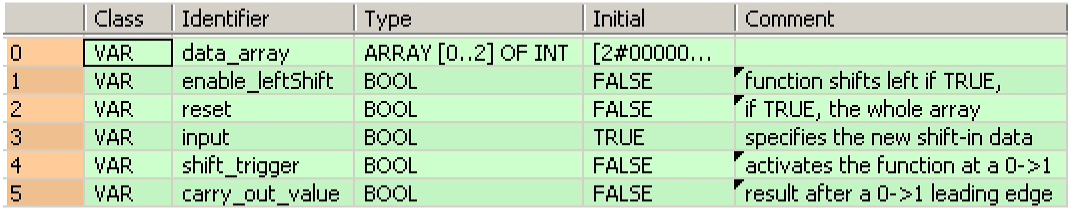
Variable	Data type	Function
LeftDirection	BOOL	specifies direction of shift, TRUE = left, FALSE = right
DataInput	BOOL	shift-in data, TRUE = 1, FALSE = 0
ShiftTrigger	BOOL	activates shift
Reset	BOOL	resets data in area specified by d1 and d2 to 0
Carry	BOOL	bit shifted out
d1	ANY16	starting 16-bit area
d2		ending 16-bit area

Operands

For	Relay				T/C		Register			Constant
LeftDirection, DataInput, ShiftTrigger, Reset	X	Y	R	L	T	C	-	-	-	-
Carry	-	Y	R	L	T	C	-	-	-	-
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

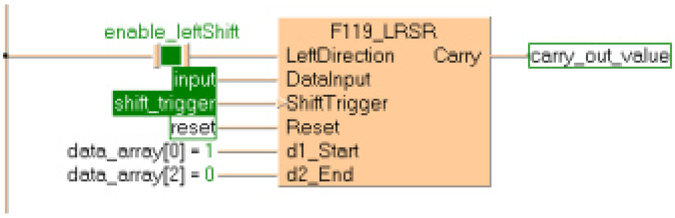
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	data_array	ARRAY [0..2] OF INT	[2#00000...	
1	VAR	enable_leftShift	BOOL	FALSE	function shifts left if TRUE,
2	VAR	reset	BOOL	FALSE	if TRUE, the whole array
3	VAR	input	BOOL	TRUE	specifies the new shift-in data
4	VAR	shift_trigger	BOOL	FALSE	activates the function at a 0->1
5	VAR	carry_out_value	BOOL	FALSE	result after a 0->1 leading edge

Body When the variable **enable_leftShift** is set to TRUE, the function shifts left, else it shifts right.

LD



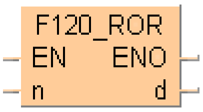
ST When programming with structured text, enter the following:

```
carry_out_value:=F119_LRSR( LeftDirection:= enable_leftShift,
        DataInput:= input,
        ShiftTrigger:= shift_trigger,
        Reset:= reset,
        d1_Start:= data_array[0],
        d1_End:= data_array[2]);
```

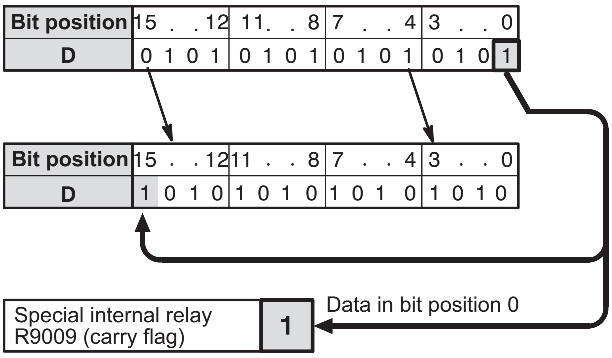
F120_ROR

16-bit data right rotate

Description Rotates **n** bits of the 16-bit data specified by **d** to the right if the trigger **EN** is in the ON-state.



The following example rotates one bit to the right:



Bit position	15 . . 12	11. . 8	7 . . 4	3 . . 0
D	0 1 0 1	0 1 0 1	0 1 0 1	0 1 0 1

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
D	1 0 1 0	1 0 1 0	1 0 1 0	1 0 1 0

When **n** bits are rotated to the right,

- the data in bit position **n**-1 (**n**th bit starting from bit position 0) is transferred to the special internal relay R9009 (carry-flag).
- **n** bits starting from bit position 0 are shifted out to the right and into the higher bit positions of the 16-bit data specified by **d**.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F120_ROR **(see page 1320)**

Data types

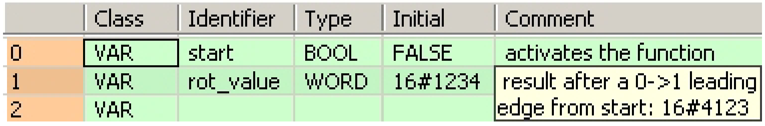
Variable	Data type	Function
d	ANY16	16-bit area
n	INT	number of bits to be rotated

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

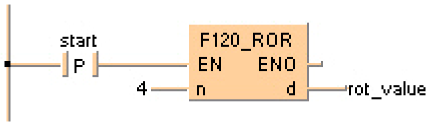
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	rot_value	WORD	16#1234	result after a 0->1 leading edge from start: 16#4123
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD



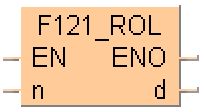
ST When programming with structured text, enter the following:

```
IF DF(start) THEN
    F120_ROR( n:= 4,
        d=> rot_value);
END_IF;
```

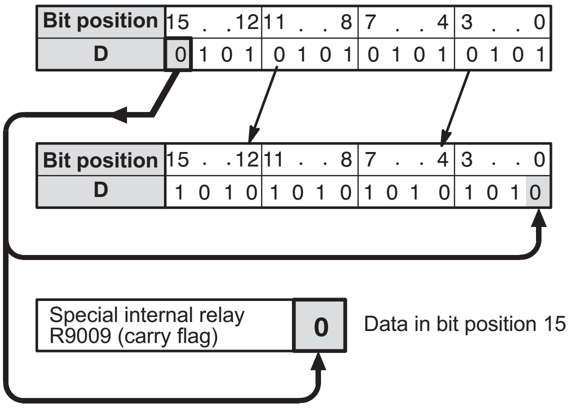
F121_ROL

16-bit data left rotate

Description Rotates **n** bits of the 16-bit data specified by **d** to the left if the trigger **EN** is in the ON-state.



The following example rotates one bit to the left:



When **n** bits are rotated to the left,

- the data in bit position 16-**n** (**n**th bit starting from bit position 15) is transferred to special internal relay R9009 (carry-flag).
- **n** bits starting from bit position 15 are shifted out to the left and into the lower bit positions of the 16-bit data specified by **d**.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F121_ROL **(see page** 1320**)**

Data types

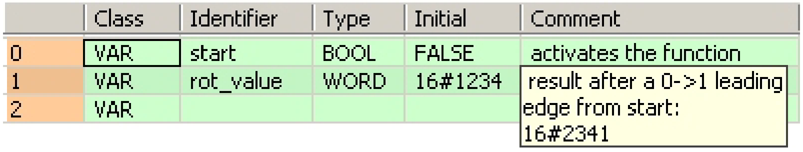
Variable	Data type	Function
d	ANY16	16-bit area
n	INT	number of bits to be rotated

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

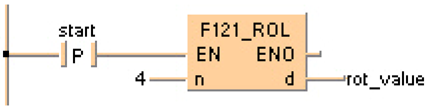
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	rot_value	WORD	16#1234	result after a 0->1 leading edge from start: 16#2341
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD



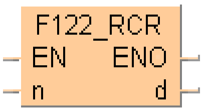
ST When programming with structured text, enter the following:

```
IF DF(start) THEN
    F121_ROL( n:= 4,
        d=> rot_value);
END_IF;
```

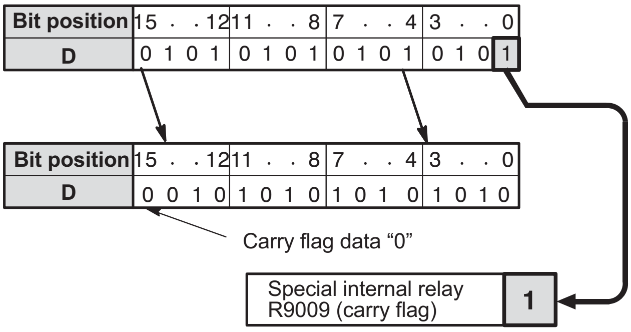
F122_RCR

16-bit data right rotate with carry-flag data

Description Rotates **n** bits of the 16-bit data specified by **d** including the data of carry-flag to the right if the trigger **EN** is in the ON-state.



This example rotates one bit to the right:



Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
D	0 1 0 1	0 1 0 1	0 1 0 1	0 1 0 1

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
D	0 0 1 0	1 0 1 0	1 0 1 0	1 0 1 0

When **n** bits with carry-flag data are rotated to the right,

- the data in bit position **n**-1 (**n**th bit starting from bit position 0) are transferred to special internal relay R9009 (carry-flag).
- **n** bits starting from bit position 0 are shifted out to the right and carry-flag data and **n**-1 bits starting from bit position 0 are subsequently shifted into the higher bit positions of the 16-bit data specified by **d**.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F122_RCR **(see page 1320)**

Data types

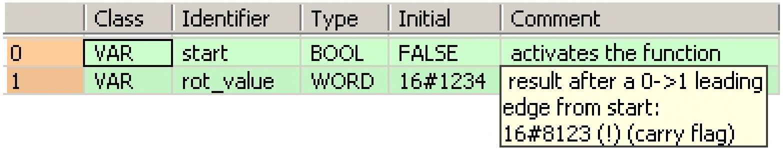
Variable	Data type	Function
d	ANY16	16-bit area
n	INT	number of bits to be rotated

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

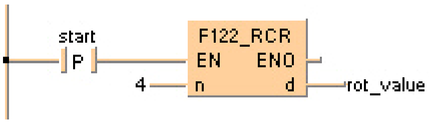
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	rot_value	WORD	16#1234	result after a 0->1 leading edge from start: 16#8123 (!) (carry flag)

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD



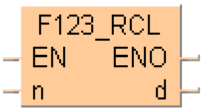
ST When programming with structured text, enter the following:

```
IF DF(start) THEN
    F122_RCR( n:= 4,
        d=> rot_value);
END_IF;
```

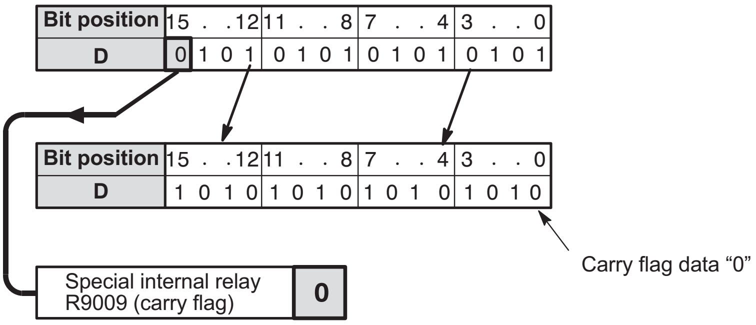
F123_RCL

16-bit data left rotate with carry-flag data

Description Rotates **n** bits of the 16-bit data specified by **d** including the data of carry-flag to the left if the trigger **EN** is in the ON-state.



This example rotates one bit to the left:



Bit position	15 . .12	11 . . 8	7 . . 4	3 . . 0
D	0 1 0 1	0 1 0 1	0 1 0 1	0 1 0 1

Bit position	15 . .12	11 . . 8	7 . . 4	3 . . 0
D	1 0 1 0	1 0 1 0	1 0 1 0	1 0 1 0

When **n** bits with carry-flag data are rotated to the left,

the data in bit position 16-**n** (**n**th bit starting from bit position 15) is transferred to special internal relay R9009 (carry-flag).

n bits starting from bit position 15 are shifted out to the left and carry-flag data and **n**-1 bits starting from bit position 15 are shifted into lower bit positions of the 16-bit data specified by **d**.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F123_RCL **(see page 1320)**

Data types

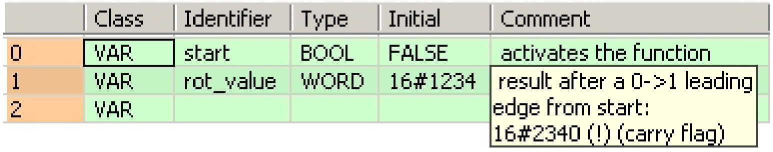
Variable	Data type	Function
d	ANY16	16-bit area
n	INT	number of bits to be rotated

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

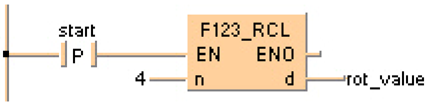
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	rot_value	WORD	16#1234	result after a 0->1 leading edge from start: 16#2340 (!) (carry flag)
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out.

LD



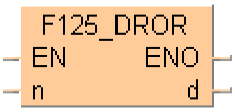
ST When programming with structured text, enter the following:

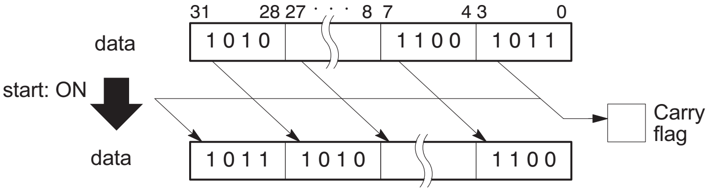
```
IF DF(start) THEN
    F123_RCL( n:= 4,
        d=> rot_value);
END_IF;
```

F125_DROR

32-bit data right rotate

Description The function rotates the value at output **d** to the right. The number of bits at output **d** to be rotated to the right is specified by the value assigned at input **n**. This shift can lie between 0 and 255 (only the lower value byte of **n** is effective). Right rotate means that the bits shifted out of bit position 0 (LSB) are shifted via bit position 31 (MSB) into the value at output **d.** When input **n** = 0, no rotation takes place. When at input **n** > 32, the same result is achieved as with a number **n** < 32: e.g. **n** = 32 produces the same result as when **n** = 0; **n** = 33 the same as **n** = 1. The bit at position **n** - 1 (the last bit shifted out to the right) is simultaneously stored in special internal relay R9009 (carry flag) so that it can be evaluated accordingly.





This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F125_DROR **(see page 1320)**

Data types

Variable	Data type	Function
n	INT	number of bits to be rotated (range: 0 to 255)
d	ANY32	32-bit area

Operands

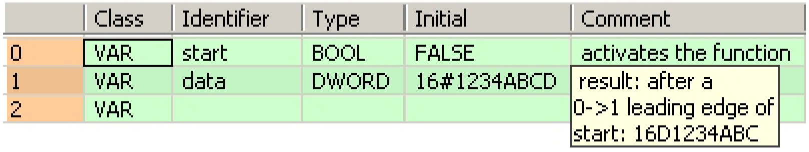
For	Relay				T/C		Register			Constant
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9009	%MX0.900.9	for an instant	▪ the bit at position n - 1 of d has the value 1.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

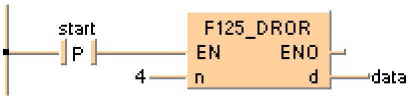
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	DWORD	16#1234ABCD	result: after a 0->1 leading edge of start: 16D1234ABC
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out. It rotates 4 bits (corresponds to one position in a hexadecimal representation) to the right. At input n the constant 4 is assigned directly to the function. You may, however, declare an input variable in the POU header instead.

LD



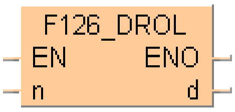
ST When programming with structured text, enter the following:

```
IF DF(start) THEN
    F125_DROR( n:= 4,
        d=> data);
END_IF;
```

F126_DROL

32-bit data left rotate

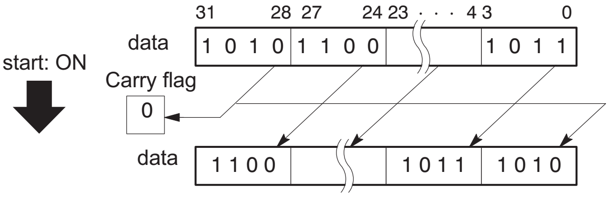
Description The function rotates the value at output d to the left. The number of bits at output d to be rotated to the left is specified by the value assigned at input n. This shift can lie between 0 and 255 (only the lower value byte of n is effective).Left rotate means that the bits shifted out of bit position 31 (MSB) are shifted via bit position 0 (LSB) into the value at output d.



When input n = 0, no rotation takes place.

When at input n > 32, the same result is achieved as with a number n < 32: e.g. n = 33 produces the same result as when n = 0; n = 34 the same as n = 1.

The bit at position 32 - n (the last bit shifted out to the right) is simultaneously stored in special internal relay R9009 (carry flag) so that it can be evaluated accordingly.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F126_DROL **(see page 1321)**

Data types

Variable	Data type	Function
n	INT	number of bits to be rotated (range: 0 to 255)
d	ANY32	32-bit area

Operands

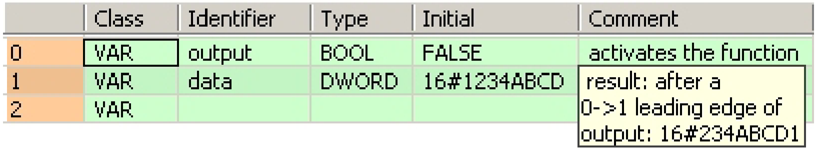
For	Relay				T/C		Register			Constant
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9009	%MX0.900.9	for an instant	▪ the bit at position 32 - n of d has the value 1.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

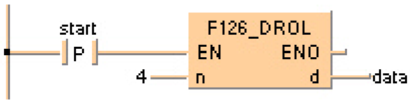
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	output	BOOL	FALSE	activates the function
1	VAR	data	DWORD	16#1234ABCD	result: after a 0->1 leading edge of output: 16#234ABCD1
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out. It rotates 4 bits (corresponds to one position in a hexadecimal representation) to the left. At input n the constant 4 is assigned directly to the function. You may, however, declare an input variable in the POU header instead.

LD



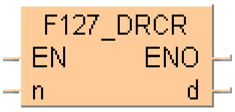
ST When programming with structured text, enter the following:

```
IF DF(start) THEN
    F126_DROL( n:= 4,
        d=> data);
END_IF;
```

F127_DRCR

32-bit data right rotate with carry flag data

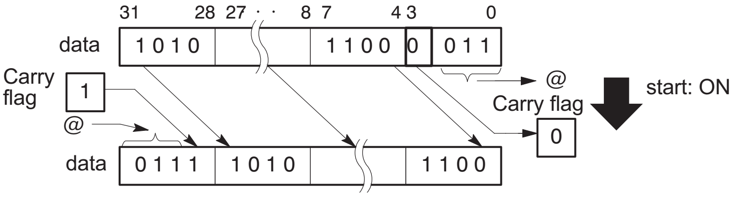
Description The function rotates the value at output **d** via the carry flag to the right. The number of bits at output **d** to be rotated to the right is specified by the value assigned at input **n**. This shift can lie between 0 and 255 (only the lower value byte of **n** is effective).



The bit value at bit position **n** - 1 is stored in the carry flag. The function shifts out **n** bits from bit 0 to the right, and then along with the inverted carry flag first, continues via bit 31 into the higher bit positions. Position 32 - **n** now has the inverted value of the carry flag.

When input **n** = 0, no rotation occurs and the carry flag remains unchanged.

When at input **n** > 32, the same result is achieved as with a number **n** < 32: e.g. **n** = 33 produces the same result as when **n** = 0; **n** = 34 the same as **n** = 1.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F127_DRCR **(see page 1321)**

Data types

Variable	Data type	Function
d	ANY32	32-bit data area
n	INT	number of bits to be rotated (range: 0 to 255)

Operands

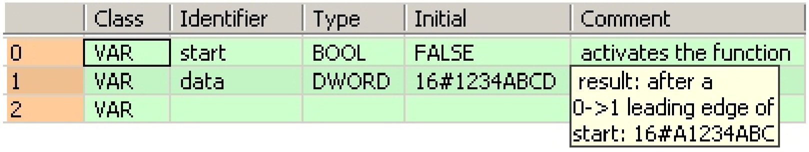
For	Relay				T/C		Register			Constant
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Error flags

No.	IEC address	Set	If
R9009	%MX0.900.9	for an instant	▪ the bit at position n - 1 has the value 1.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

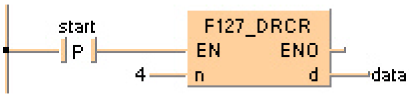
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	DWORD	16#1234ABCD	result: after a 0->1 leading edge of start: 16#A1234ABC
2	VAR				

Body When the variable **start** changes from FALSE to TRUE, the function is carried out. In this example the constant (4) is assigned to the function at input n. You may, however, declare a variable in the POU header instead.

LD



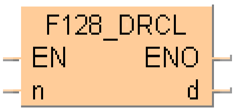
ST When programming with structured text, enter the following:

```
IF DF(start) THEN
    F127_DRCR( n:= 4,
        d=> data);
END_IF;
```

F128_DRCL

32-bit data right rotate with carry flag data

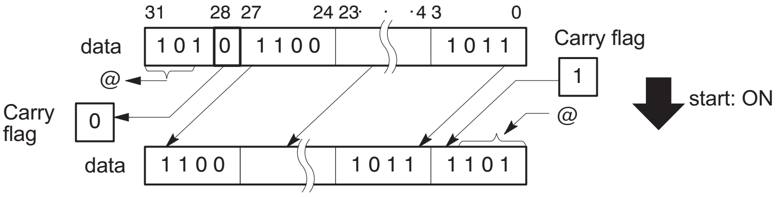
Description The function rotates the value at output **d** via the carry flag to the left. The number of bits at output **d** to be rotated to the left is specified by the value assigned at input **n**. This shift can lie between 0 and 255 (only the lower value byte of **n** is effective).



The bit value at bit position 32 - **n** is stored in the carry flag. The function shifts out **n** bits to the left via bit 31 (MSB), and then along with the inverted carry flag first, continues via bit 0 (LSB) into the storage range. Position **n** - 1 now has the inverted value of the carry flag.

When input **n** = 0, no rotation occurs and the carry flag remains unchanged.

When at input **n** > 32, the same result is achieved as with a number **n** < 32: e.g. **n** = 33 produces the same result as when **n** = 0; **n** = 34 the same as **n** = 1.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F128_DRCL **(see page 1321)**

Data types

Variable	Data type	Function
d	ANY32	32-bit area
n	INT	number of bits to be rotated (range: 0 to 255)

Operands

For	Relay				T/C		Register			Constant
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

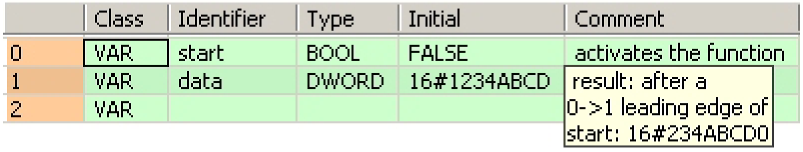
Error flags

No.	IEC address	Set	If
R9009	%MX0.900.9	for an instant	▪ the bit at position 32 - **n** has the value 1.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

All input and output variables used for programming this function have been declared in the POU header.

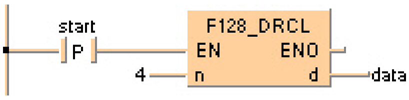


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	DWORD	16#1234ABCD	result: after a 0->1 leading edge of start: 16#234ABCD0
2	VAR				

Body

When the variable **start** changes from FALSE to TRUE, the function is carried out. In this example the constant (4) is assigned to the function at input n. You may, however, declare a variable in the POU header instead.

LD



ST

When programming with structured text, enter the following:

```
IF DF(start) THEN
    F128_DRCL( n:= 4,
        d=> data);
END_IF;
```

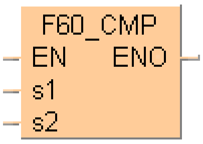
Chapter 19

Comparison instructions

F60_CMP

16-bit data compare

Description Compares the 16-bit data specified by **s1** with one specified by **s2** if the trigger **EN** is in the ON-state. The compare operation result is stored in special internal relays R9009, R900A to R900C.



Instead of using this FP instruction, we recommend using the related IEC instruction of the comparison instructions. Please refer also to Advantages of the IEC instructions in the online help.

Data	Comparison between s1 and s2	Flags			
		R900A (>flag)	R900B (=flag)	R900C (<flag)	R9009 (carry-flag)
16-bit data with sign	s1<s2	Off	Off	On	#
	s1=s2	Off	On	Off	Off
	s1>s2	On	Off	Off	#
16-bit data without sign	s1<s2	#	Off	#	On
	s1=s2	Off	On	Off	Off
	s1>s2	#	Off	#	Off

# turns ON or OFF depending on the conditions

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F60_CMP **(see page 1325)**

Data types

Variable	Data type	Function
s1, s2	ANY16	16-bit area or 16-bit equivalent constant to be compared

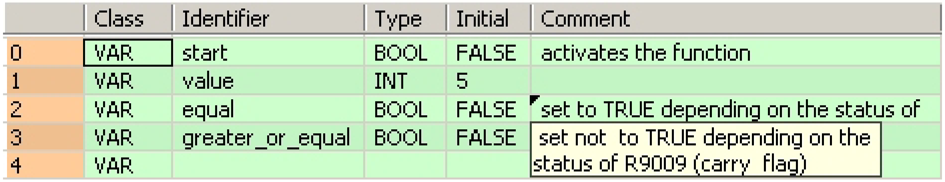
The variables **s1** and **s2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

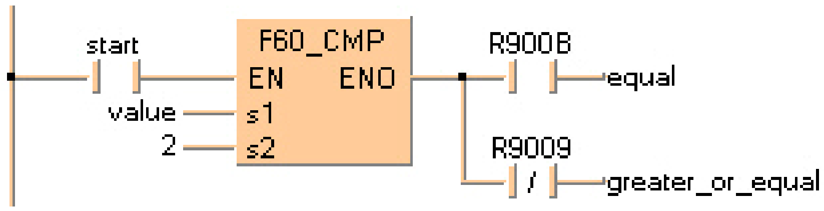
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value	INT	5	
2	VAR	equal	BOOL	FALSE	set to TRUE depending on the status of
3	VAR	greater_or_equal	BOOL	FALSE	set not to TRUE depending on the status of R9009 (carry flag)
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



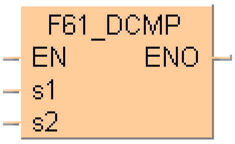
ST When programming with structured text, enter the following:

```
equal:= FALSE;
greater_or_equal:= FALSE;
IF start THEN
    F60_CMP(value, 2);
    IF R900B THEN
        equal := TRUE;
    END_IF;
    IF NOT(R9009) THEN
        greater_or_equal:= TRUE;
    END_IF;
END_IF;
```

F61_DCMP

32-bit data compare

Description Compares the 32-bit data or 32-bit equivalent constant specified by **s1** with one specified by **s2** if the trigger **EN** is in the ON-state. The compare operation result is stored in special internal relays R9009, R900A to R900C.



Instead of using this FP instruction, we recommend using the related IEC instruction of the comparison instructions. Please refer also to Advantages of the IEC instructions in the online help.

Data	Comparison between s1 and s2	Flags			
		R900A (>flag)	R900B (=flag)	R900C (<flag)	R9009 (carry-flag)
32-bit data with sign	s1<s2	Off	Off	On	#
	s1=s2	Off	On	Off	Off
	s1>s2	On	Off	Off	#
32-bit data without sign	s1<s2	#	Off	#	On
	s1=s2	Off	On	Off	Off
	s1>s2	#	Off	#	Off

# turns ON or OFF depending on the conditions

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F61_DCMP **(see page 1325)**

Data types

Variable	Data type	Function
s1, s2	ANY32	32-bit area or 32-bit equivalent constant to be compared

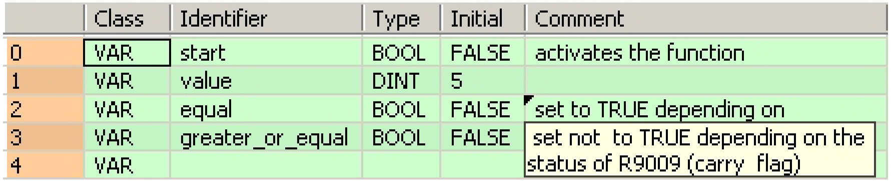
The variables **s1** and **s2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

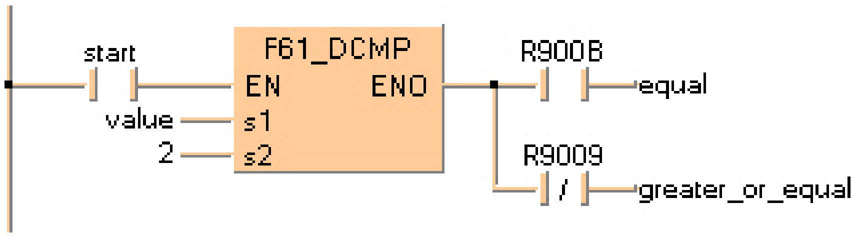
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value	DINT	5	
2	VAR	equal	BOOL	FALSE	set to TRUE depending on
3	VAR	greater_or_equal	BOOL	FALSE	set not to TRUE depending on the status of R9009 (carry flag)
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



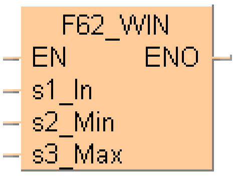
ST When programming with structured text, enter the following:

```
equal:= FALSE;
greater_or_equal:= FALSE;
IF start THEN
    F61_DCMP(value, 2);
    IF R900B THEN
        equal:= TRUE;
    END_IF;
    IF NOT(R9009) THEN
        greater_or_equal:= TRUE;
    END_IF;
END_IF;
```

F62_WIN

16-bit data band compare

Description Compares the 16-bit equivalent constant or 16-bit data specified by **s1_In** with the data band specified by **s2_Min** and **s3_Max** if the trigger **EN** is in the ON-state. This instruction checks that **s1_In** is in the data band between **s2_Min** (lower limit) and **s3_Max** (higher limit), larger than **s3_Max**, or smaller than **s2_Min**. The compare operation considers +/- sign. Since the BCD data is also treated as 16-bit data with sign, we recommend using BCD data within the range of 0 to 7999 to avoid confusion. The compare operation result is stored in special internal relays R9009, R900A to R900C.



Comparison between s1, s2 and s3	Flags		
	R900A (>flag)	R900B (=flag)	R900C (<flag)
s1_In < **s2_Min**	Off	Off	On
s2_Min ≤ **s1_In** ≤ **s3_Max**	Off	On	Off
s1_In > **s3**	On	Off	Off

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F62_WIN **(see page 1325)**

Data types

Variable	Data type	Function
s1_In	ANY16	16-bit area or 16-bit equivalent constant to be compared
s2_Min		lower limit, 16-bit area or 16-bit equivalent constant
s3_Max		upper limit, 16-bit area or 16-bit equivalent constant

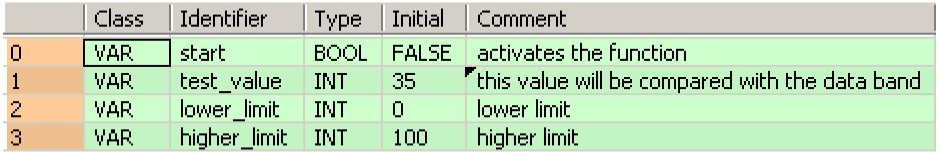
The variables **s1, s2** and **s3** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2, s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

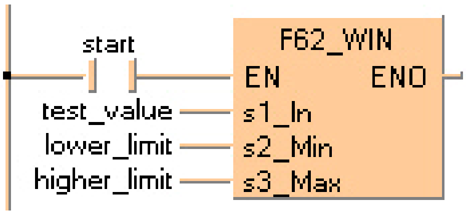
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	test_value	INT	35	this value will be compared with the data band
2	VAR	lower_limit	INT	0	lower limit
3	VAR	higher_limit	INT	100	higher limit

Body When the variable **start** is set to TRUE, the function is carried out.

LD



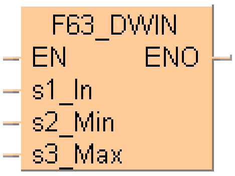
ST When programming with structured text, enter the following:

```
IF start THEN
    F62_WIN( s1_In:= test_value,
         s2_Min:= lower_limit,
         s3_Max:= higher_limit);
END_IF;
```

F63_DWIN

32-bit data band compare

Description Compares the 32-bit equivalent constant or 32-bit data specified by **s1_In** with the data band specified by **s2_Min** and **s3_Max** if the trigger **EN** is in the ON-state. This instruction checks that **s1_In** is in the data band between **s2_Min** (lower limit) and **s3_Max** (higher limit), larger than **s3_Max**, or smaller than **s2_Min**. The compare operation considers +/- sign. Since the BCD data is also treated as 32-bit data with sign, we recommend using BCD data within the range of 0 to 79999999 to avoid confusion. The compare operation result is stored in special internal relays R9009, R900A to R900C.



Comparison between s1, s2 and s3	Flags		
	R900A (>flag)	R900B (=flag)	R900C (<flag)
s1_In < s2_Min	Off	Off	On
s2_Min ≤ s1_In ≤ s3_Max	Off	On	Off
s1_In > s3	On	Off	Off

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F63_DWIN **(see page 1326)**

Data types

Variable	Data type	Function
s1_In	ANY32	32-bit area or 32-bit equivalent constant to be compared
s2_Min		lower limit, 32-bit area or 32-bit equivalent constant
s3_Max		upper limit, 32-bit area or 32-bit equivalent constant

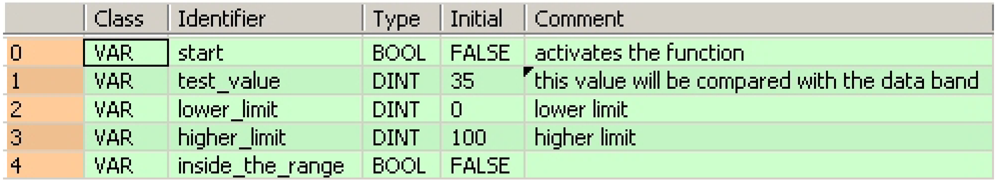
The variables **s1, s2** and **s3** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1_In, s2_Min, s3_Max	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

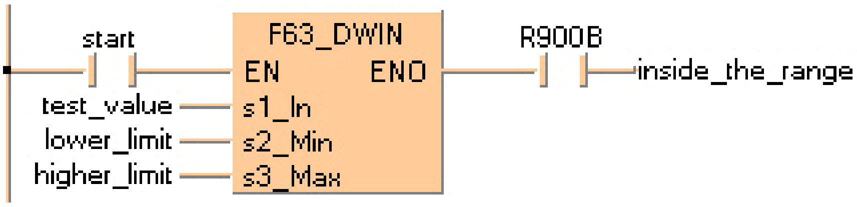
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	test_value	DINT	35	this value will be compared with the data band
2	VAR	lower_limit	DINT	0	lower limit
3	VAR	higher_limit	DINT	100	higher limit
4	VAR	inside_the_range	BOOL	FALSE	

Body When the variable **start** is set to TRUE, the function is carried out.

LD

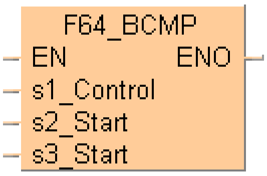


ST
```
inside_the_range:= FALSE;
IF start THEN
    F63_DWIN( s1_In:= test_value,
         s2_Min:= lower_limit,
         s3_Max:= higher_limit);
    IF R900B THEN
        inside_the_range:= TRUE;
    END_IF;
END_IF;
```

F64_BCMP

Block data compare

Description Compares the contents of data block specified by **s2** with the contents of data block specified by **s3** according to the contents specified by **s1** if the trigger **EN** is in the ON-state.



s1 specifications

16#	1	0	04
	⇑	⇑	⇑
	A	B	C

A = Starting byte position of data block specified by **s3**

1: Starting from higher byte

0: Starting from lower byte

B = Starting byte position of data block specified by **s2**

1: Starting from higher byte

0: Starting from lower byte

C = Number of bytes to be compared

range: 16#01–16#99 (BCD)

The compare operation result is stored in the special internal relay R900B. When **s2 = s3**, the special internal relay is in the ON-state.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F64_BCMP **(see page 1326)**

☞ **The flag R900B used for the compare instruction is renewed each time a compare instruction is executed. Therefore the program that uses R900B should be just after F64_BCMP.**

Data types

Variable	Data type	Function
s1_Control	WORD	control code specifying byte positions and number of bytes to be compared
s2_Start	ANY16	starting 16-bit area to be compared to **s3**
s3_Start		starting 16-bit area to be compared to **s2**

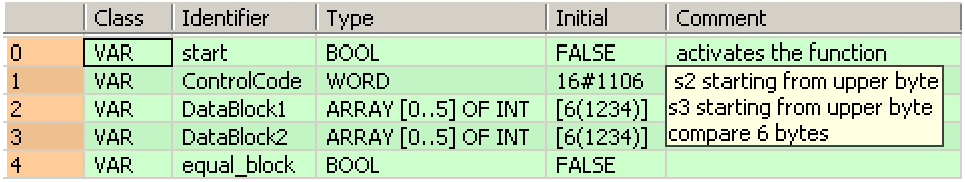
The variables **s2** and **s3** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s2, s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

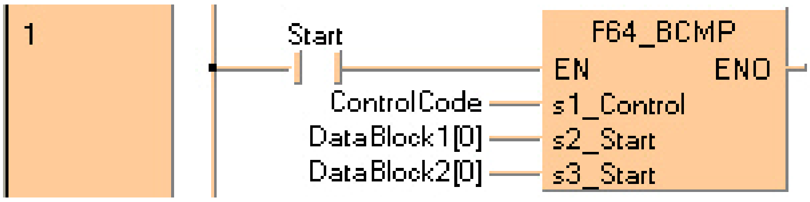
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	ControlCode	WORD	16#1106	s2 starting from upper byte
2	VAR	DataBlock1	ARRAY [0..5] OF INT	[6(1234)]	s3 starting from upper byte
3	VAR	DataBlock2	ARRAY [0..5] OF INT	[6(1234)]	compare 6 bytes
4	VAR	equal_block	BOOL	FALSE	

Body When the variable **start** is set to TRUE, the function is carried out.

LD



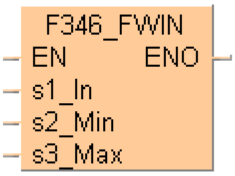
ST When programming with structured text, enter the following:

```
IF start THEN
    F64_BCMP( s1_Control:= ControlCode,
         s2_Start:= DataBlock1[0],
         s3_Start:= DataBlock2[0]);
END_IF;
```

F346_FWIN

Floating point data band compare

Description The function compares a data band whose upper and lower limits are specified at inputs **s2_Min** and **s3_Max** with a value that is entered at input **s1_In**. The result is returned as follows:



Comparison between s1, s2 and s3	Flags		
	R900A (>flag)	R900B (=flag)	R900C (<flag)
s1_In < **s2_Min**	Off	Off	On
s2_Min ≤ **s1_In** ≤ **s3_Max**	Off	On	Off
s1_In > **s3**	On	Off	Off

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F346_FWIN **(see page 1324)**

Data types

Variable	Data type	Function
s1_In	REAL	REAL number data to be compared to **s2_Min** and **s3_Max**
s2_Min		lower limit
s3_Max		upper limit

Operands

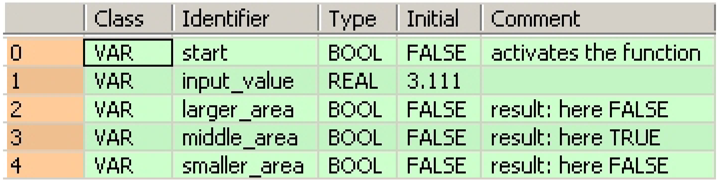
For	Relay				T/C		Register			Constant
s1, s2, s3	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the values at inputs **s1_In**, **s2_Min**, and **s3_Max** are not REAL numbers ▪ if the value at **s2_Min** > **s3_Max**.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

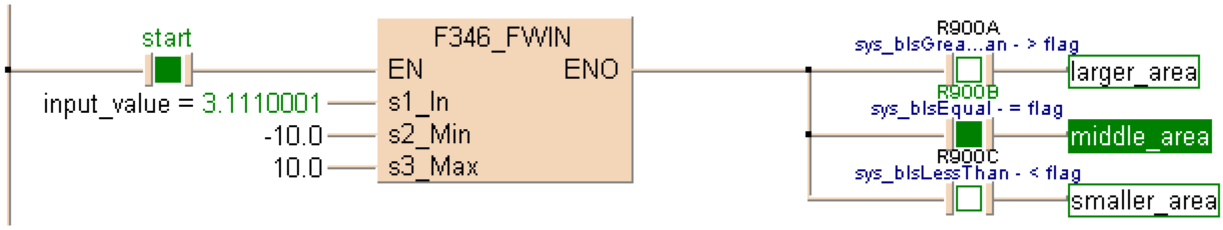


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	3.111	
2	VAR	larger_area	BOOL	FALSE	result: here FALSE
3	VAR	middle_area	BOOL	FALSE	result: here TRUE
4	VAR	smaller_area	BOOL	FALSE	result: here FALSE

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body The constants -10.0 and 10.0 are assigned to the inputs **s2_Min** (lower limit) and **s3_Max** (upper limit). You may, however, declare two variables in the POU header instead. When the variable **start** is set to TRUE, the function is carried out. The values of special internal relays R900A (> flag), R900B (= flag) and R900C (< flag) are transferred to the variables **larger_area**, **middle_area** and **smaller_area**. Since the **input_value** = 3.111 is within the range of the limits set (-10.0 to 10.0), the = relay and hence the variable **middle_area** are set to TRUE.

LD



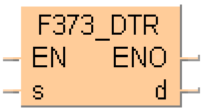
ST When programming with structured text, enter the following:

```
input_value:=3.111;
IF start THEN
    F346_FWIN( s1_In:= input_value , s2_Min:= -10.0 , s3_Max:= 10.0 );
END_IF;(* -10.0 =lower limit, 10.0 upper limit *)

IF R900A THEN
    larger_area:=TRUE;
END_IF;
IF R900B THEN
    middle_area:=TRUE;
END_IF;
IF R900C THEN
    smaller_area:=TRUE;
END_IF;
```

F373_DTR

16-bit data revision detection

Description The function detects changes in a value at input s by comparing it with its former value that is stored at output **d**. If the new input value at **s** does not coincide with the old value, the function assigns the new value to output **d**. To signal the change, the carry flag R9009 is set simultaneously.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F373_DTR **(see page 1325)**

☞ **The status of the carry flag is updated at each execution of the instruction. Therefore, programs that use the carry flag should utilize it immediately after F373_DTR is executed.**

Data types

Variable	Data type	Function
s	ANY16	16-bit area for detecting data changes
d		area where data of previous execution is stored.

Operands

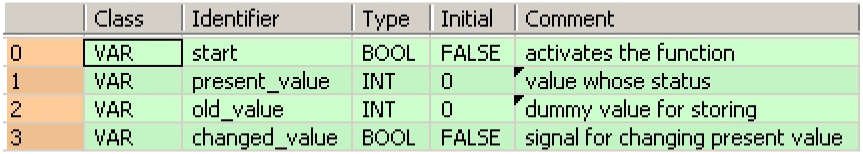
For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9009	%MX0.900.9	to TRUE	▪ the input value at **s** has changed in comparison to the former value.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

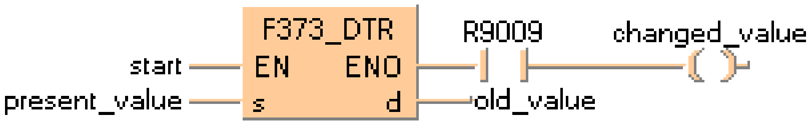
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	present_value	INT	0	value whose status
2	VAR	old_value	INT	0	dummy value for storing
3	VAR	changed_value	BOOL	FALSE	signal for changing present value

Body When the variable **start** is set to TRUE, the function is carried out. If the input value **present_value** has changed in comparison to the output value **old_value** the carry flag R9009 is set. The status of the carry flag is then assigned to the variable **changed_value**.

LD



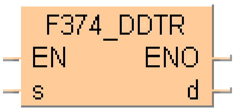
ST When programming with structured text, enter the following:

```
IF start THEN
    F373_DTR(present_value, old_value);
    IF R9009 THEN
        changed_value:=TRUE;
    END_IF;
END_IF;
```

F374_DDTR

32-bit data revision detection

Description The function detects changes in a value at input **s** by comparing it with its former value that is stored at output **d**. If the new input value at **s** does not coincide with the old value, the function assigns the new value to output **d**. To signal the change, the carry flag **R9009** is set simultaneously.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F374_DDTR **(see page 1325)**

☞ **The status of the carry flag is updated at each execution of the instruction. Therefore, programs that use the carry flag should utilize it immediately after F374_DDTR is executed.**

Data types

Variable	Data type	Function
s	ANY32	32-bit area for detecting data changes
d		32-bit area where data of previous execution is stored

Operands

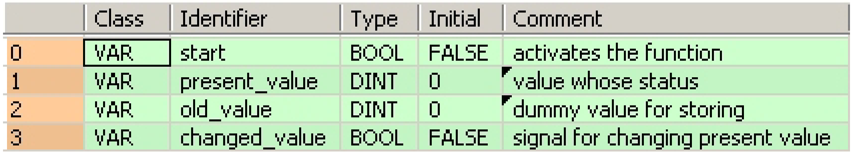
For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9009	%MX0.900.9	to TRUE	▪ the input value at **s** has changed in comparison to the former value.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

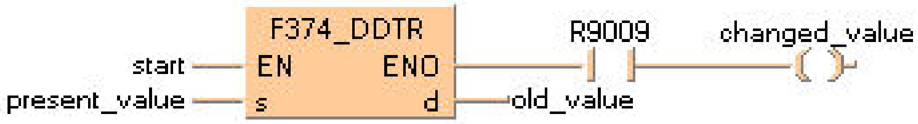
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	present_value	DINT	0	value whose status
2	VAR	old_value	DINT	0	dummy value for storing
3	VAR	changed_value	BOOL	FALSE	signal for changing present value

Body When the variable **start** is set to TRUE, the function is carried out. If the input value **present_value** has changed in comparison to the output value **old_value** the carry flag R9009 is set. The status of the carry flag is then assigned to the variable **changed_value**.

LD



ST When programming with structured text, enter the following:

```
IF start THEN
    F374_DDTR(present_value, old_value);
    IF R9009 THEN
        changed_value:=TRUE;
    END_IF;
END_IF;
```

19.1 Further comparison instructions

If you need information on one of the following comparison instructions, please refer to the corresponding standard operators in the online help:

ST=	AN=	OR=	STD=	AND=	ORD=
ST<>	AN<>	OR<>	STD<>	AND<>	ORD<>
ST>	AN>	OR>	STD>	AND>	ORD>
ST>=	AN>=	OR>=	STD>=	AND>=	ORD>=
ST<	AN<	OR<	STD<	AND<	ORD<
ST<=	AN<=	OR<=	STD<=	AND<=	ORD<=

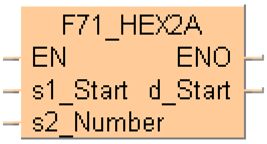
Chapter 20

Conversion instructions

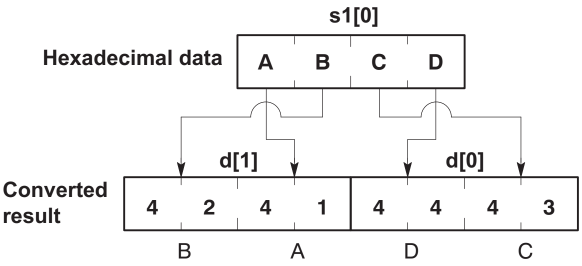
F71_HEX2A

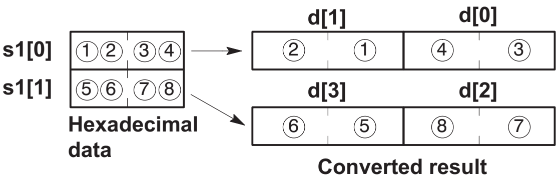
HEX -> ASCII conversion

Description Converts the data byte-wise from the 16-bit area specified by **s1_Start** to ASCII codes that express the equivalent hexadecimals if the trigger **EN** is in the TRUE-state. The number of bytes to be converted is specified by **s2_Number**. The converted result is stored in the area starting with the 16-bit area specified by **d_Start**. ASCII code requires 8 bits (one byte) to express one hexadecimal character. Upon conversion to ASCII, the data length will thus be twice the length of the source data.



The two characters that make up one byte are interchanged when stored. Two bytes are converted as one segment of data.





ASCII HEX codes to express hexadecimal characters:

Hexadecimal number	ASCII HEX code
0	16#30
1	16#31
2	16#32
3	16#33
4	16#34
5	16#35
6	16#36
7	16#37
8	16#38
9	16#39
A	16#41
B	16#42
C	16#43
D	16#44
E	16#45
F	16#46

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F71_HEX2A **(see page** 1326**)**

Data types

Variable	Data type	Function
s1_Start	ANY16	starting 16-bit area for hexadecimal number (source)
s2_Number	INT	specifies number of source data bytes to be converted
d_Start	ANY16	starting 16-bit area for storing ASCII code (destination)

Operands

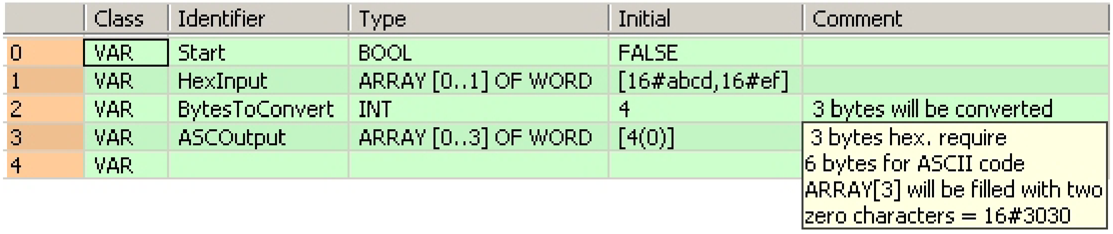
For	Relay				T/C		Register			Constant
s1_Start	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2_Number	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d_Start	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the byte number specified by **s2_Number** exceeds the area specified by **s1_Start** ▪ the calculated result exceeds the area specified by **d_Start**. ▪ the data specified by **s2_Number** is recognized as "0".
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

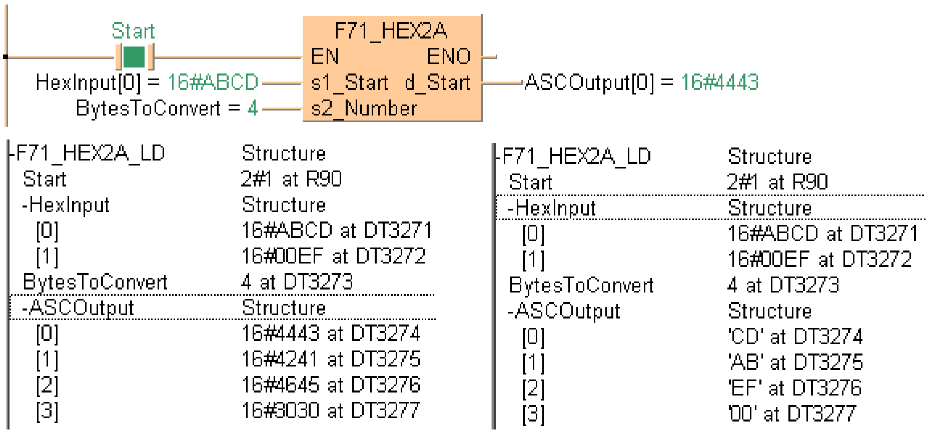
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	HexInput	ARRAY [0..1] OF WORD	[16#abcd,16#ef]	
2	VAR	BytesToConvert	INT	4	3 bytes will be converted
3	VAR	ASCOutput	ARRAY [0..3] OF WORD	[4(0)]	3 bytes hex. require 6 bytes for ASCII code ARRAY[3] will be filled with two zero characters = 16#3030
4	VAR				

Body When the variable **Start** is set to true, the number of data bytes given in **BytesToConvert** in **HexInput** is converted to ASCII code and stored in **ASCOutput**. Note that two characters that make up one byte are interchanged when stored. One Monitor Header shows the Hex values, and the other the ASCII values.

LD



-F71_HEX2A_LD	Structure
Start	2#1 at R90
-HexInput	Structure
[0]	16#ABCD at DT3271
[1]	16#00EF at DT3272
BytesToConvert	4 at DT3273
-ASCOutput	Structure
[0]	16#4443 at DT3274
[1]	16#4241 at DT3275
[2]	16#4645 at DT3276
[3]	16#3030 at DT3277

-F71_HEX2A_LD	Structure
Start	2#1 at R90
-HexInput	Structure
[0]	16#ABCD at DT3271
[1]	16#00EF at DT3272
BytesToConvert	4 at DT3273
-ASCOutput	Structure
[0]	'CD' at DT3274
[1]	'AB' at DT3275
[2]	'EF' at DT3276
[3]	'00' at DT3277

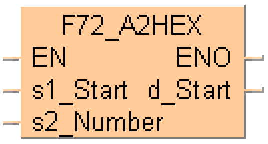
ST When programming with structured text, enter the following:

```
IF start THEN
    F71_HEX2A( s1_Start:= HexInput[0],
         s2_Number:= BytesToConvert,
         d_Start=> ASCOutput[0]);
END_IF;
```

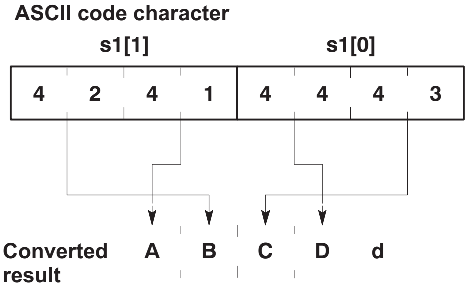
F72_A2HEX

ASCII -> HEX conversion

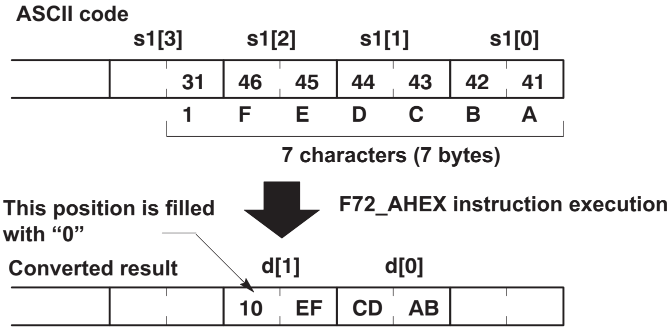
Description Converts the ASCII codes that express the hexadecimal characters starting from the 16-bit area specified by **s1** to hexadecimal numbers if the trigger **EN** is in the ON-state. **s2** specifies the number of ASCII (number of characters) to be converted. The converted result is stored in the area starting from the 16-bit area specified by **d**. ASCII code requires 8 bits (one byte) to express one hexadecimal character. Upon conversion to a hexadecimal number, the data length will thus be half the length of the ASCII code source data.



The data for two ASCII code characters is converted to two numeric digits for one word. When this takes place, the characters of the upper and lower bytes are interchanged. Four characters are converted as one segment of data.



Converted results are stored in byte units. If an odd number of characters is being converted, "0" will be entered for bits 0 to 3 of the final data (byte) of the converted results. Conversion of odd number of source data bytes:



Hexadecimal characters and ASCII codes:

ASCII HEX code	Hexadecimal number
16#30	0
16#31	1
16#32	2
16#33	3
16#34	4
16#35	5
16#36	6
16#37	7
16#38	8
16#39	9
16#41	A
16#42	B
16#43	C
16#44	D
16#45	E
16#46	F

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types Availability of F72_A2HEX **(see page 1326)**

Data types

Variable	Data type	Function
s1	WORD	starting 16-bit area for ASCII code (source)
s2	INT	specifies number of source data bytes to be converted
d	ANY16	starting 16-bit area for storing converted data (destination)

Operands

For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

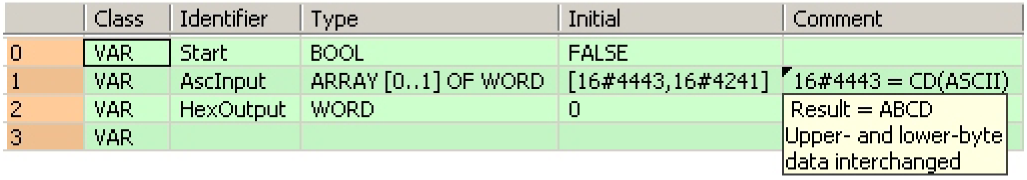
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the number of bytes specified by s2 exceeds the area specified by s1. ▪ the converted result exceeds the area specified by d. ▪ the data specified by s2 is recognized as "0". ▪ ASCII code, not a hexadecimal number (0 to F), is specified.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

All input and output variables used for programming this function have been declared in the POU header.

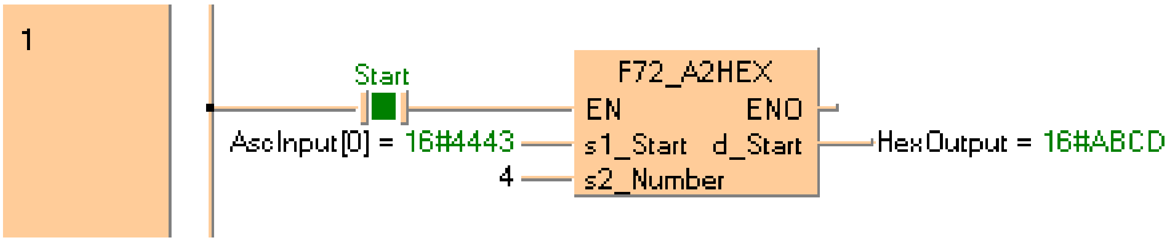


	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	AscInput	ARRAY [0..1] OF WORD	[16#4443,16#4241]	16#4443 = CD(ASCII) Result = ABCD Upper- and lower-byte data interchanged
2	VAR	HexOutput	WORD	0	
3	VAR				

Body

When the variable **start** is set to TRUE, the function is carried out. In this example, the value for **s2**, i.e. the number of bytes to be converted from ASCII code to hexadecimal code, is entered directly at the contact pin.

LD



ST

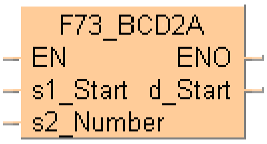
When programming with structured text, enter the following:

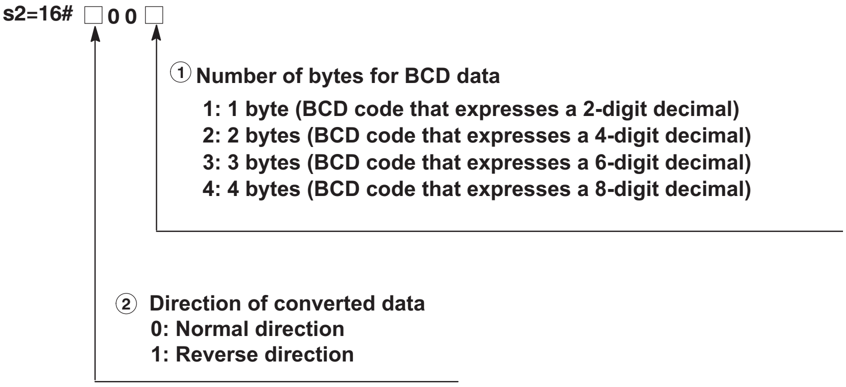
```
IF start THEN
    F72_A2HEX( s1_Start:= AscInput[0],
         s2_Number:= 4,
         d_Start=> HexOutput);
```

F73_BCD2A

BCD -> ASCII conversion

Description Converts the BCD code starting from the 16-bit area specified by **s1** to the ASCII code that expresses the equivalent decimals according to the contents specified by **s2** if the trigger **EN** is in the ON-state. **s2** specifies the number of source data bytes and the direction of converted data (normal/reverse).

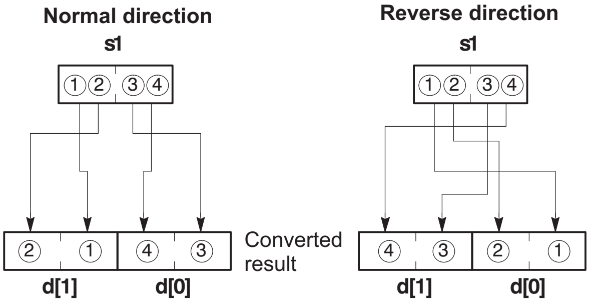




① Number of bytes for BCD data
1: 1 byte (BCD code that expresses a 2-digit decimal)
2: 2 bytes (BCD code that expresses a 4-digit decimal)
3: 3 bytes (BCD code that expresses a 6-digit decimal)
4: 4 bytes (BCD code that expresses a 8-digit decimal)

② Direction of converted data
0: Normal direction
1: Reverse direction

The two characters that make up one byte are interchanged when stored. Two bytes are converted as one segment of data:



The converted result is stored in the area specified by **d**. ASCII code requires 8 bits (one byte) to express one BCD character. Upon conversion to ASCII, the data length will thus be twice the length of the BCD source data.

ASCII HEX code to express BCD character:

BCD character	ASCII HEX code
0	H30
1	H31
2	H32
3	H33
4	H34
5	H35
6	H36
7	H37
8	H38
9	H39

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the

"Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of F73_BCD2A (see page 1326)**

Data types

Variable	Data type	Function
s1	WORD	starting 16-bit area for BCD data (source)
s2	ANY16	specifies number of source data bytes to be converted, and how it is arranged
d	WORD	starting 16-bit area for storing converted result (destination)

Operands

For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

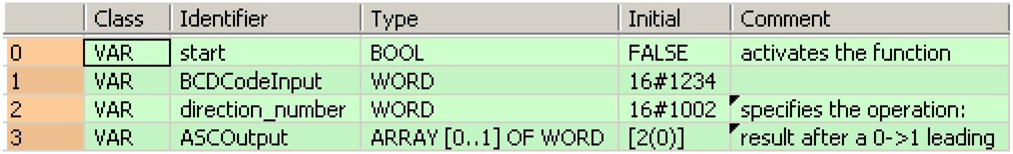
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

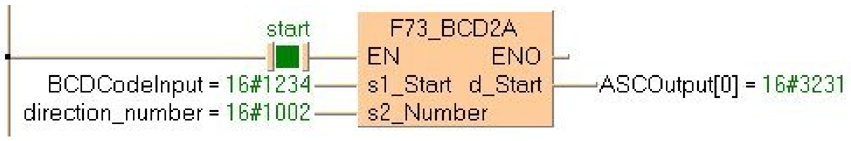
POU header All input and output variables used for programming this function have been declared in the POU header.

Body When the variable **start** is set to TRUE, the function is carried out. In this example, the variable **direction_number** specifies that from the input variable **BCDCodeInput**, 2 bytes will be converted in the reverse direction and stored in **ASCOutput**.

LD



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	BCDCodeInput	WORD	16#1234	
2	VAR	direction_number	WORD	16#1002	specifies the operation:
3	VAR	ASCOutput	ARRAY [0..1] OF WORD	[2(0)]	result after a 0->1 leading



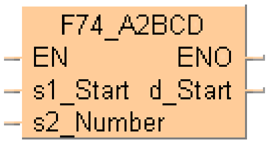
ST When programming with structured text, enter the following:

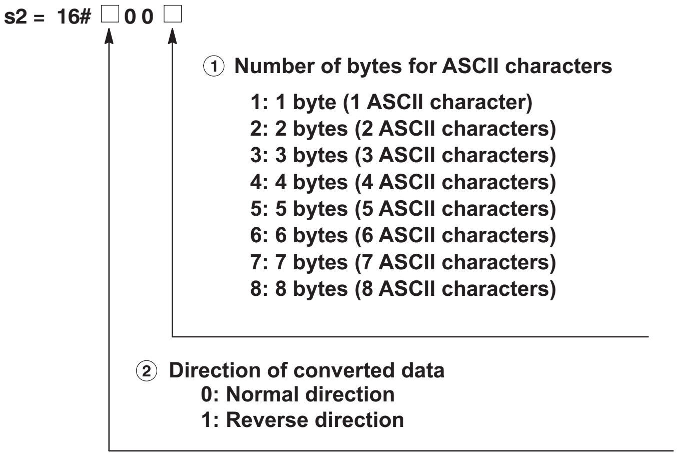
```
IF start THEN
    F73_BCD2A( s1_Start:= BCDCodeInput ,
         s2_Number:= direction_number ,
         d_Start=> ASCOutput[0] );
END_IF;
```

F74_A2BCD

ASCII -> BCD conversion

Description Converts the ASCII codes that express the decimal characters starting from the 16-bit area specified by **s1** to BCD if the trigger **EN** is in the ON-state. **s2** specifies the number of source data bytes and the direction of converted code source data.





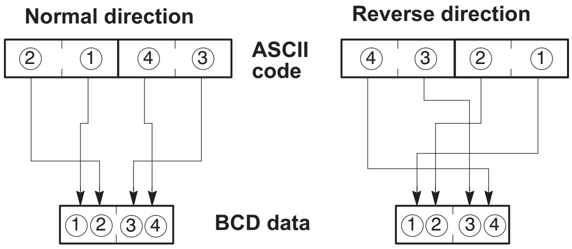
① Number of bytes for ASCII characters

1: 1 byte (1 ASCII character)
2: 2 bytes (2 ASCII characters)
3: 3 bytes (3 ASCII characters)
4: 4 bytes (4 ASCII characters)
5: 5 bytes (5 ASCII characters)
6: 6 bytes (6 ASCII characters)
7: 7 bytes (7 ASCII characters)
8: 8 bytes (8 ASCII characters)

② Direction of converted data

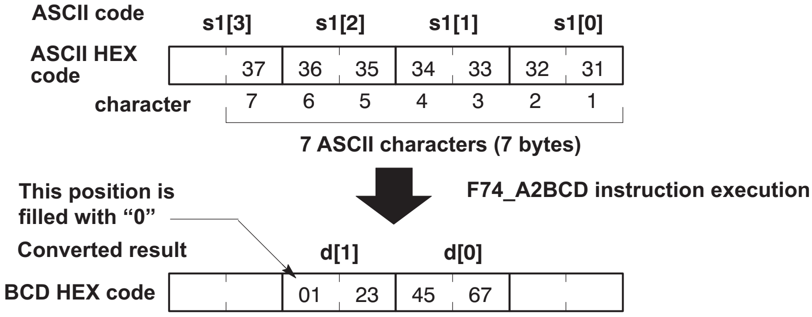
0: Normal direction
1: Reverse direction

Four characters are converted as one segment of data:



The converted result is stored in byte units in the area starting from the 16-bit area specified by **d**. ASCII code requires 8 bits (1 byte) to express 1 BCD character. Upon conversion to a BCD number, the data length will thus be half the length of the ASCII code source data.

If an odd number of characters is being converted, "0" will be entered for bit position 0 to 3 of the final data (byte) of the converted results if data is sequenced in the normal direction, and "0" will be entered for bit position 4 to 7 if data is being sequenced in the reverse direction:



ASCII HEX code to express BCD character:

BCD character	ASCII HEX code
0	H30
1	H31
2	H32
3	H33
4	H34
5	H35
6	H36
7	H37
8	H38
9	H39

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F74_A2BCD **(see page** 1326**)**

Data types

Variable	Data type	Function
s1	WORD	starting 16-bit area for storing ASCII code (source)s
s2	ANY16	specifies number of source data bytes to be converted, and how it is arranged
d	WORD	starting 16-bit area for storing converted result (destination)

Operands

For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

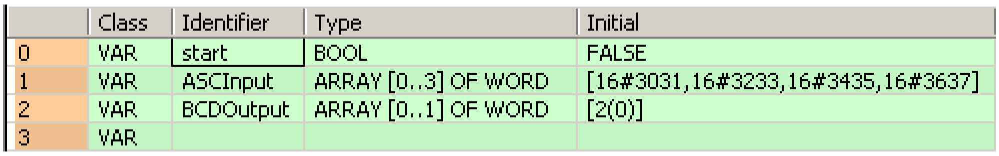
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ ASCII code not corresponding to decimal numbers (0 to 9) is specified. ▪ the number of bytes specified by **s2** exceeds the area specified by **s1**. ▪ the converted result exceeds the area specified by **d**. ▪ the data specified by **s2** is recognized as "0". ▪ the number of bytes for ASCII characters in **s2** is more than 16#8.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

All input and output variables used for programming this function have been declared in the POU header.

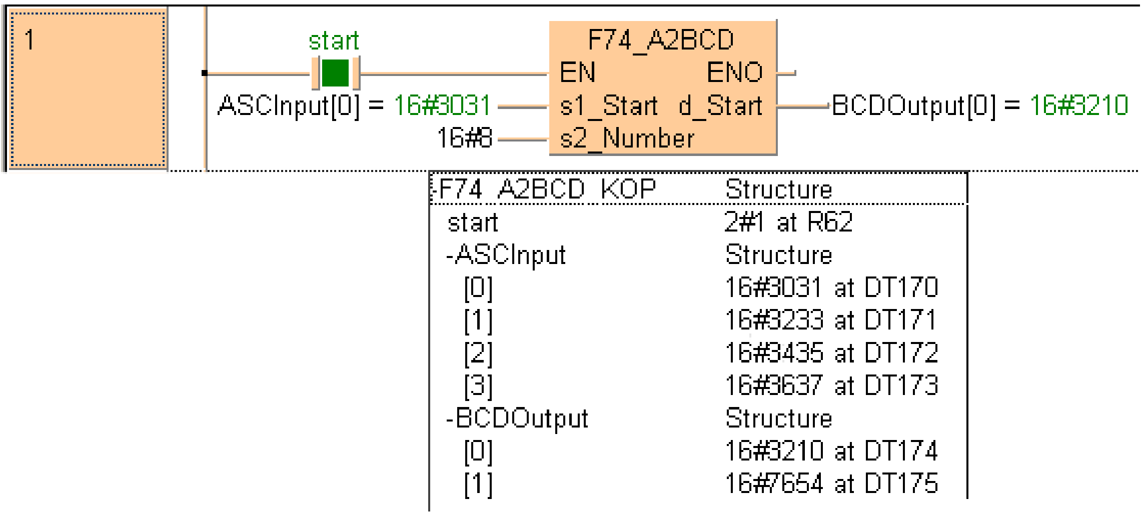


	Class	Identifier	Type	Initial
0	VAR	start	BOOL	FALSE
1	VAR	ASCInput	ARRAY [0..3] OF WORD	[16#3031,16#3233,16#3435,16#3637]
2	VAR	BCDOutput	ARRAY [0..1] OF WORD	[2(0)]
3	VAR			

Body

When the variable **start** is set to TRUE, the function is carried out. For the variable at **s1**, you define an ARRAY with a minimum of four word elements because 8 ASCII characters require 8 bytes of memory and the function cannot convert more than 8 bytes. In this example, the value for **s2** is entered directly at the contact pin.

LD



-F74_A2BCD_KOP	Structure
start	2#1 at R62
-ASCInput	Structure
[0]	16#3031 at DT170
[1]	16#3233 at DT171
[2]	16#3435 at DT172
[3]	16#3637 at DT173
-BCDOutput	Structure
[0]	16#3210 at DT174
[1]	16#7654 at DT175

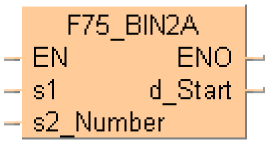
ST When programming with structured text, enter the following:

```
IF start THEN
    F74_A2BCD( s1_Start:= ASCInput[0] ,
         s2_Number:= 16#8 ,
         d_Start=> BCDOutput[0] );
END_IF;
```

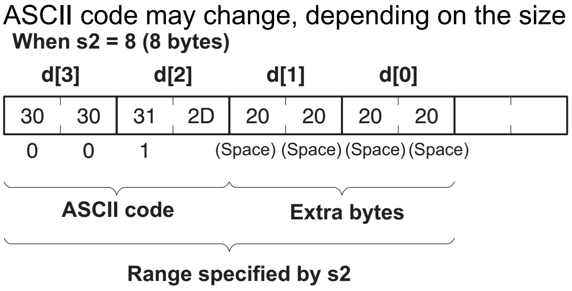
F75_BIN2A

16-bit BIN -> ASCII conversion

Description Converts the 16-bit data specified by **s1** to ASCII codes that express the equivalent decimal value. The converted result is stored in the area starting from the 16-bit area specified by **d** as specified by **s2**. Specify the number of bytes in decimal number in **s2**. (This specification cannot be made with BCD data.)



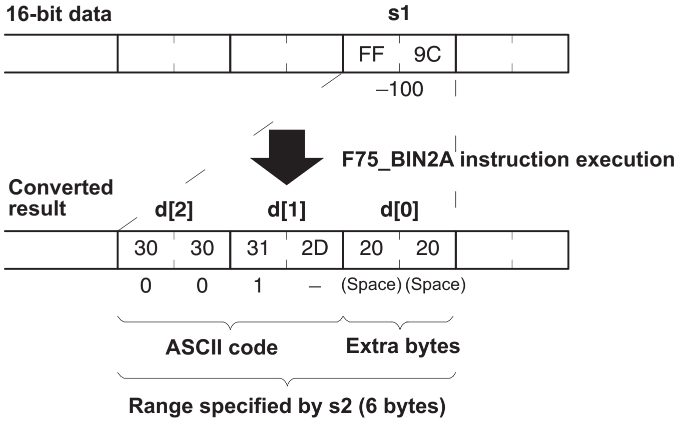
- If a positive number is converted, the "+" sign is not converted.
- When a negative number is converted, the "-" sign is also converted to ASCII code (ASCII HEX code: 16#2D).
- If the area specified by **s2** is more than that required by the converted data, the ASCII code for "SPACE" (ASCII HEX code: 16#20) is stored in the extra area.
- Data is stored in the direction towards the final address, so the position of the ASCII code may change, depending on the size of the data storage area.



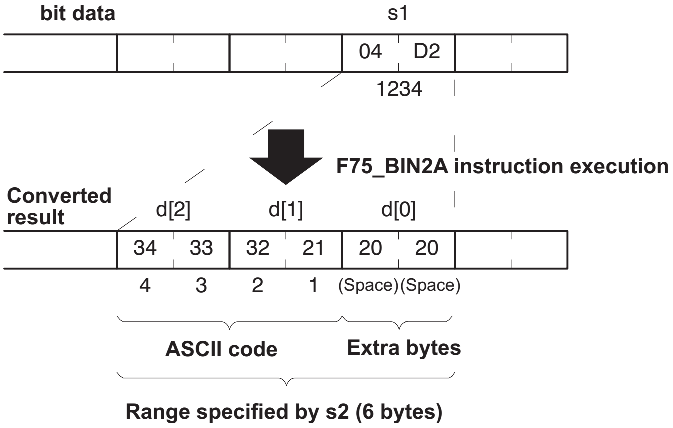
- If the number of bytes of ASCII codes following conversion (including the minus sign) is larger than the number of bytes specified by the **s2**, an operation error occurs. Make sure the sign is taken into consideration when specifying the object of conversion for the **s2**.

The following illustrations show conversions from 16-bit decimal data to ASCII codes.

When a negative number is converted:



When a positive number is converted



Decimal characters to express ASCII HEX code:

Decimal characters	ASCII HEX code
SPACE	16#20
-	16#2D
0	16#30
1	16#31
2	16#32
3	16#33
4	16#34
5	16#35
6	16#36
7	16#37
8	16#38
9	16#39

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F75_BIN2A **(see page 1326)**

Data types

Variable	Data type	Function
s1	ANY16	16-bit area to be converted (source)
s2	INT	specifies number of bytes used to express destination data (ASCII codes)
d	WORD	16-bit area for storing ASCII codes (destination)

Operands

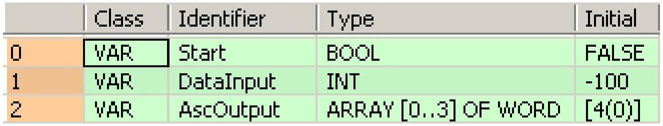
For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the number of bytes specified by **s2** exceeds the area specified by **d**. ▪ the data specified by **s2** is recognized as "0". ▪ the converted result exceeds the area specified by **d**. ▪ the number of bytes of converted result exceeds the number of bytes specified by **s2**.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

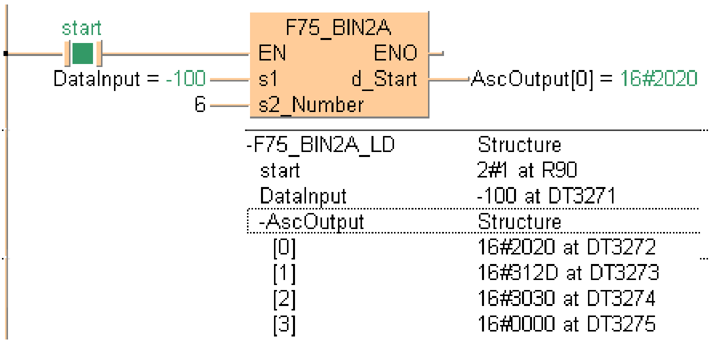
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Start	BOOL	FALSE
1	VAR	DataInput	INT	-100
2	VAR	AscOutput	ARRAY [0..3] OF WORD	[4(0)]

Body When the variable **start** is set to TRUE, the function is carried out. This programming example is based on the example for the conversion of a negative number outlined above. The monitor value icon is activated for both the LD and IL bodies; the monitor header icon is activated for the LD body.

LD



-F75_BIN2A_LD	Structure
start	2#1 at R90
DataInput	-100 at DT3271
-AscOutput	Structure
[0]	16#2020 at DT3272
[1]	16#312D at DT3273
[2]	16#3030 at DT3274
[3]	16#0000 at DT3275

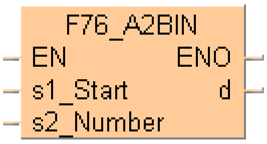
ST When programming with structured text, enter the following:

```
IF start THEN
    F75_BIN2A( s1:= DataInput ,
         s2_Number:= 6 ,
         d_Start=> ASCOutput[0] );
END_IF;
```

F76_A2BIN

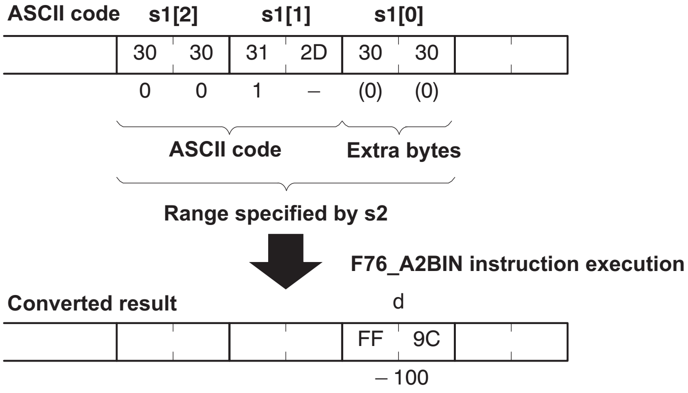
ASCII -> 16-bit BIN conversion

Description Converts the ASCII codes that express the decimal digits, starting from the 16-bit area specified by **s1** to 16-bit data as specified by **s2**. The converted result is stored in the area specified by **d**. **s2** specifies the number of source data bytes to be converted using decimal number. (This specification cannot be made with BCD data.)

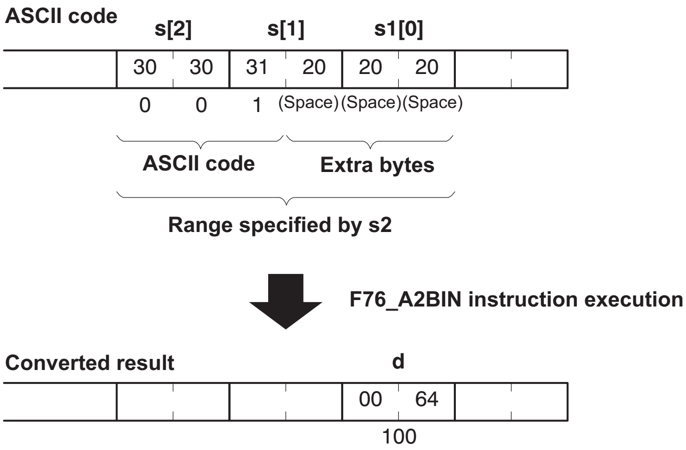


- The **ASCII** codes being converted should be stored in the direction of the last address in the specified area.
- If the area specified by **s1** and **s2** is more than that required for the data you want to convert, place "0" (ASCII HEX code: 16#30) or "SPACE" (ASCII HEX code: 16#20) into the extra bytes.
- ASCII codes with signs (such as +: 16#2B and -: 16#2D) are also converted. The + codes can be omitted.

Example of converting an ASCII **code indicating a negative number**



Example of converting an ASCII **code indicating a positive number**



ASCII HEX code to express decimal characters:

ASCII HEX code	Decimal characters
16#20	SPACE
16#2B	+
16#2D	-
16#30	0
16#31	1
16#32	2
16#33	3
16#34	4
16#35	5
16#36	6
16#37	7
16#38	8
16#39	9

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F76_A2BIN **(see page** 1326**)**

Data types

Variable	Data type	Function
s1	WORD	16-bit area for ASCII code (source)
s2	INT	specifies number of source data bytes to be converted
d	ANY16	16-bit area for storing converted data (destination)

Operands

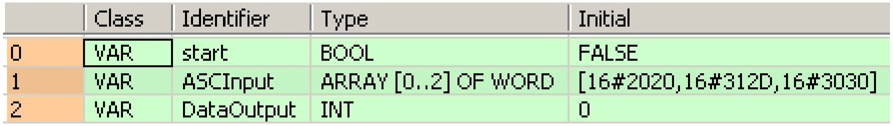
For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the number of bytes specified by **s2** exceeds the area specified by **s1**. ▪ the data specified by **s2** is recognized as "0". ▪ the converted result exceeds the 16-bit area specified by **d**. ▪ ASCII code not corresponding to decimal numbers (0 to 9) or ASCII characters (+, -, and SPACE) is specified.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

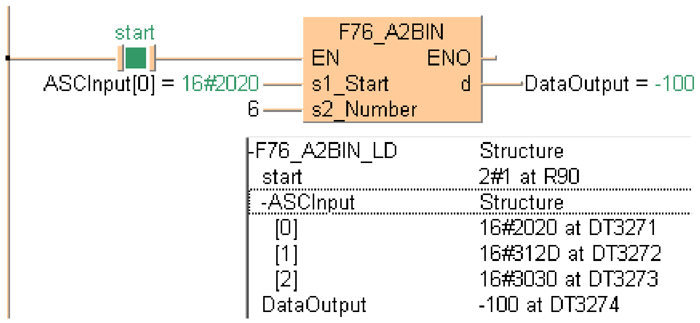
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	start	BOOL	FALSE
1	VAR	ASCInput	ARRAY [0..2] OF WORD	[16#2020,16#312D,16#3030]
2	VAR	DataOutput	INT	0

Body When the variable **start** is set to TRUE, the function is carried out. The number of bytes to be converted is entered directly at the contact pin for s2. This programming example is based on the example for the conversion of a negative number on the main page of F76_A2BIN.

LD



-F76_A2BIN_LD	Structure
start	2#1 at R90
-ASCInput	Structure
[0]	16#2020 at DT3271
[1]	16#312D at DT3272
[2]	16#3030 at DT3273
DataOutput	-100 at DT3274

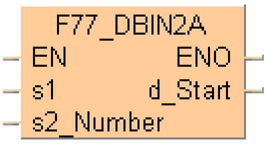
ST When programming with structured text, enter the following:

```
IF start THEN
    F76_A2BIN( s1_Start:= ASCInput[0] ,
         s2_Number:= 6 ,
         d=> DataOutput );
END_IF;
```

F77_DBIN2A

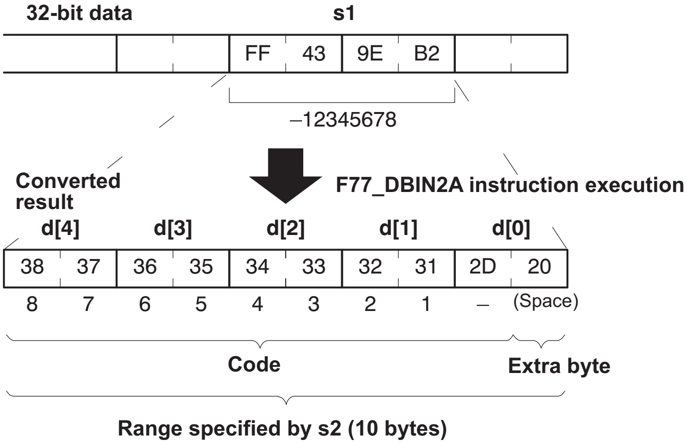
32-bit BIN -> ASCII conversion

Description Converts the 32-bit data specified by **s1** to ASCII code that expresses the equivalent decimals. The converted result is stored in the area starting from the 16-bit area specified by **d** as specified by **s2**. **s2** specifies the number of bytes used to express the destination data using decimal.



- When a positive number is converted, the "+" sign is not converted.
- When a negative number is converted, the "-" sign is also converted to ASCII code (ASCII HEX code: 16#2D).
- If the area specified by **s2** is more than that required by the converted data, the ASCII code for "SPACE" (ASCII HEX code: 16#20) is stored in the extra area.
- Data is stored in the direction of the last address, so the position of the ASCII code may change depending on the size of the data storage area.
- If the number of bytes of ASCII codes following conversion (including the minus sign) is larger than the number of bytes specified by the **s2**, an operation error occurs. Make sure the sign is taken into consideration when specifying the object of conversion for the **s2**.

Example of converting a negative number from 32–bit decimal format to ASCII codes



Decimal characters to express ASCII HEX code:

Decimal characters	ASCII HEX code
SPACE	16#20
+	16#2B
-	16#2D
0	16#30
1	16#31
2	16#32
3	16#33
4	16#34
5	16#35
6	16#36
7	16#37
8	16#38
9	16#39

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F77_DBIN2A **(see page 1326)**

Data types

Variable	Data type	Function
s1	ANY32	32-bit data area to be converted (source)
s2	INT	specifies number of bytes to express destination data (ASCII codes)
d	WORD	16-bit area for storing ASCII codes (destination)

Operands

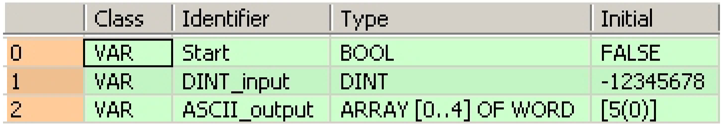
For	Relay				T/C		Register			Constant
s1	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the number of bytes specified by **s2** exceeds the area specified by **d**. ▪ the data specified by **s2** is recognized as "0". ▪ the converted result exceeds the area specified by **d**. ▪ the number of bytes of converted result exceeds the number of bytes specified by **s2**.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

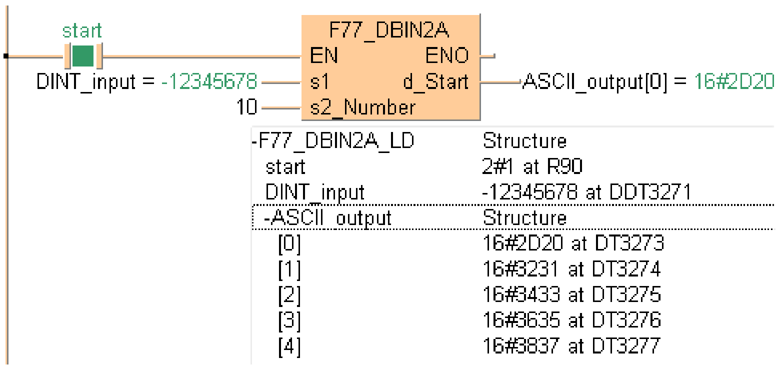
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Start	BOOL	FALSE
1	VAR	DINT_input	DINT	-12345678
2	VAR	ASCII_output	ARRAY [0..4] OF WORD	[5(0)]

Body When the variable **start** is set to TRUE, the function is carried out. The number of bytes to be converted is entered directly at the contact pin for s2.

LD



-F77_DBIN2A_LD	Structure
start	2#1 at R90
DINT_input	-12345678 at DDT3271
-ASCII_output	Structure
[0]	16#2D20 at DT3273
[1]	16#3231 at DT3274
[2]	16#3433 at DT3275
[3]	16#3635 at DT3276
[4]	16#3837 at DT3277

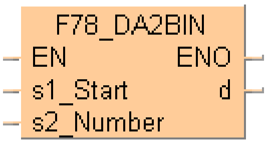
ST When programming with structured text, enter the following:

```
IF start THEN
    F77_DBIN2A( s1:= DINT_input ,
         s2_Number:= 10 ,
         d_Start=> ASCII_output[0] );
END_IF;
```

F78_DA2BIN

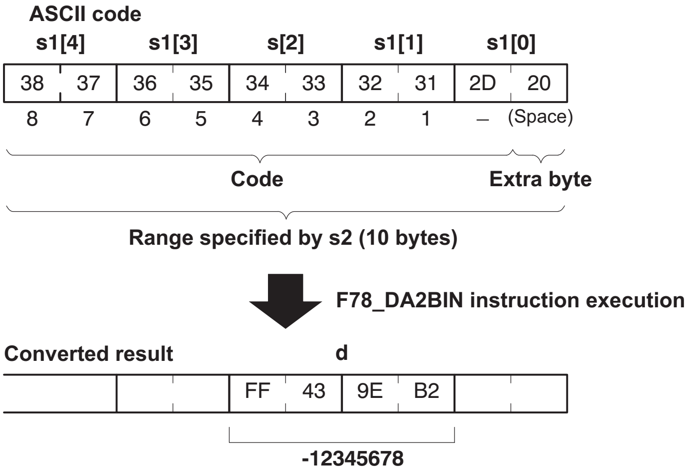
ASCII -> 32 bit BIN conversion

Description Converts ASCII code that expresses the decimal digits, starting from the 16-bit area specified by **s1** to 32-bit data as specified by **s2**. The converted result is stored in the area starting from the 32-bit area specified by **d**. **s2** specifies the number of bytes used to express the destination data using decimals.



- The ASCII codes being converted should be stored in the direction of the last address in the specified area.
- If the area specified by **s1** and **s2** is more than that required by the data you want to convert, place "0" (ASCII HEX code: 16#30) or "SPACE" (ASCII HEX code: 16#20) in the extra bytes.
- ASCII codes with signs (such as +: 16#2B and -: 16#2D) are also converted. The + codes can be omitted.

Example of converting an ASCII **code indicating a negative number**



ASCII code

s1[4]		s1[3]		s[2]		s1[1]		s1[0]	
38	37	36	35	34	33	32	31	2D	20
8	7	6	5	4	3	2	1	–	(Space)

Code
Extra byte
Range specified by s2 (10 bytes)
F78_DA2BIN instruction execution

Converted result **d**

			FF	43	9E	B2		

-12345678

ASCII HEX code to express decimal characters:

ASCII HEX code	Decimal characters
16#20	SPACE
16#2B	+
16#2D	-
16#30	0
16#31	1
16#32	2
16#33	3
16#34	4
16#35	5
16#36	6
16#37	7
16#38	8
16#39	9

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F78_DA2BIN **(see page 1326)**

Data types

Variable	Data type	Function
s1	WORD	starting 16-bit area for ASCII code (source)
s2	INT	specifies number of source data bytes to be converted
d	ANY32	area for 32-bit data storage (destination)

Operands

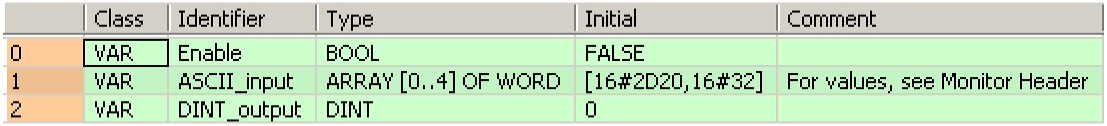
For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the number of bytes specified by s2 exceeds the area specified by s1. ▪ the data specified by s2 is recognized as "0". ▪ the converted result exceeds the area specified by d. ▪ the converted result exceeds the 32-bit area. ▪ ASCII code not corresponding to decimal numbers (0 to 9) or ASCII characters (+, -, and SPACE) is specified.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

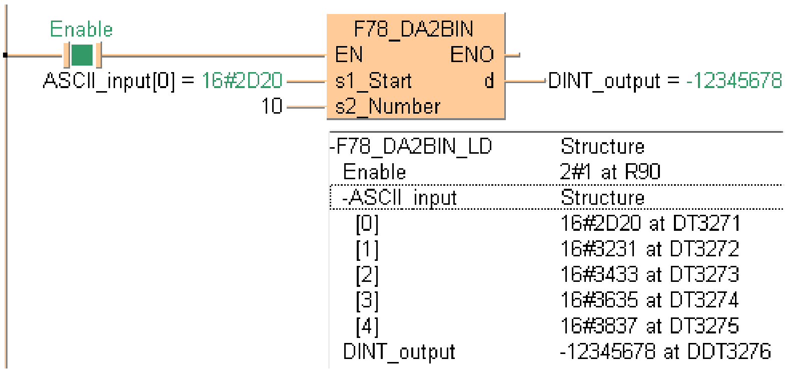
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	Enable	BOOL	FALSE	
1	VAR	ASCII_input	ARRAY [0..4] OF WORD	[16#2D20,16#32]	For values, see Monitor Header
2	VAR	DINT_output	DINT	0	

Body When the variable **Enable** is set to TRUE, the function is executed. The number of bytes to be converted is entered directly at the contact pin for s2. This programming example is based on the example for the conversion of a negative number outlined above.

LD



-F78_DA2BIN_LD	Structure
Enable	2#1 at R90
-ASCII_input	Structure
[0]	16#2D20 at DT3271
[1]	16#3231 at DT3272
[2]	16#3433 at DT3273
[3]	16#3635 at DT3274
[4]	16#3837 at DT3275
DINT_output	-12345678 at DDT3276

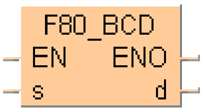
ST When programming with structured text, enter the following:

```
IF Enable THEN
    F78_DA2BIN( s1_Start:= ASCII_input[0] ,
         s2_Number:= 10 ,
         d=> DINT_output );
END_IF;
```

F80_BCD

16-bit BIN -> 4-digit BCD conversion

Description Converts the 16-bit binary data specified by **s** to the BCD code that expresses 4-digit decimals if the trigger **EN** is in the ON-state. The converted data is stored in **d**. The binary data that can be converted to BCD code are in the range of 0 (0 hex) to 9999 (270F hex).



Source [s]: 16

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
Binary data	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0
Decimal	16			

Conversion (to BCD code)

Destination [d]: 16#16 (BCD)

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
Binary data	0 0 0 0	0 0 0 0	0 0 0 1	0 1 1 0
BCD Hex code	0	0	1	6

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

See also: BCD data in the online help

PLC types **Availability of F80_BIN (see page 1326)**

Data types

Variable	Data type	Function
s	ANY16	binary data (source), range: 0 to 9999
d	WORD	16-bit area for 4-digit BCD code (destination)

Operands

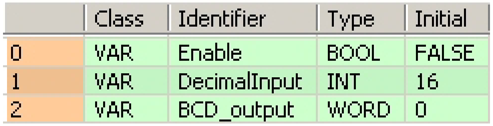
For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ 16-bit binary data outside the range of 0 (16#0) to 9999 (16#270F) is converted
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

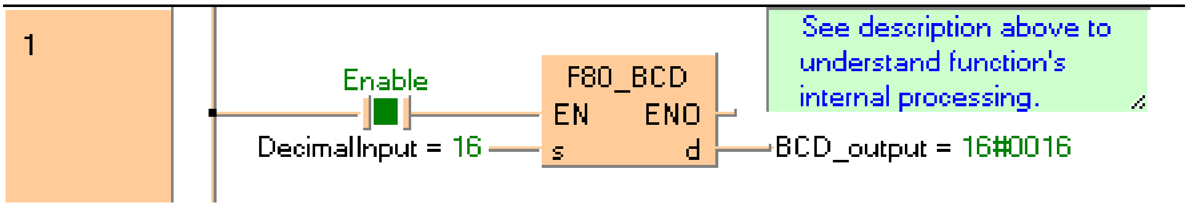
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Enable	BOOL	FALSE
1	VAR	DecimalInput	INT	16
2	VAR	BCD_output	WORD	0

Body When the variable **Enable** is set to TRUE, the function is executed. The decimal value in **DecimalInput** is converted to a BCD hexadecimal value and stored in the variable **BCD_output**.

LD



See description above to understand function's internal processing.

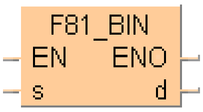
ST When programming with structured text, enter the following:

```
IF Enable THEN
    F80_BCD(DecimalInput, BCD_output);
END_IF;
```

F81_BIN

4-digit BCD -> 16-bit BIN conversion

Description Converts the BCD code that expresses 4-digit decimals specified by **s** to 16-bit binary data if the trigger **EN** is in the ON-state. The converted result is stored in the area specified by **d**.



Source [s]: 16#15 (BCD)

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
BCD code	0 0 0 0	0 0 0 0	0 0 0 1	0 1 0 1
BCD Hex code	0	0	1	5

Conversion (to binary data)

Destination [d]: 15

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
Binary data	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 1
Decimal	15			

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

See also: BCD data

PLC types **Availability of** F81_BIN **(see page 1326)**

Data types

Variable	Data type	Function
s	WORD	16-bit area for 4-digit BCD data (source)
d	ANY16	16-bit area for storing 16-bit binary data (destination)

Operands

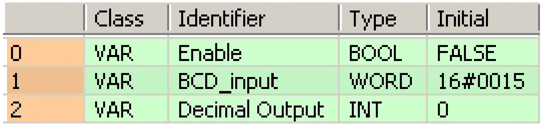
For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the data specified by **s** is not BCD data.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

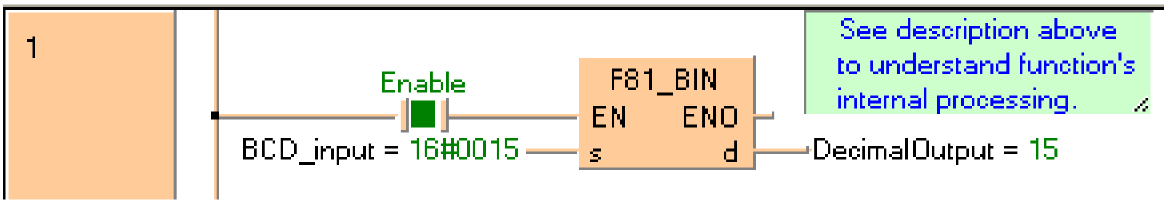
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Enable	BOOL	FALSE
1	VAR	BCD_input	WORD	16#0015
2	VAR	Decimal Output	INT	0

Body When the variable **Enable** is set to TRUE, the function is executed. The BCD value assigned to the variable **BCD_input** is converted to a decimal value and stored in the variable **DecimalOutput**. The monitor value icon is activated for both the LD and IL bodies.

LD



See description above to understand function's internal processing.

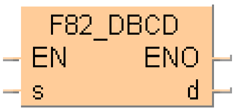
ST When programming with structured text, enter the following:

```
IF Enable THEN
    F81_BIN(BCD_Input, DecimalOutput);
END_IF;
```

F82_DBCD

32-bit BIN -> 8-digit BCD conversion

Description Converts the 32-bit binary data specified by **s** to the BCD code that expresses 8-digit decimals if the trigger **EN** is in the ON-state. The converted data is stored in **d**. The binary data that can be converted to BCD code are in the range of 0 (0 hex) to 99,999,999 (5F5E0FF hex).



Source [s]: 72811730

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0	15 · · 12	11 · · 8	7 · · 4	3 · · 0
Binary data	0 0 0 0	0 1 0 0	0 1 0 1	0 1 1 1	0 0 0 0	0 1 0 0	1 1 0 1	0 0 1 0
Decimal	72811730							

← 32-bit area →

Destination [d]: 16#72811730

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0	15 · · 12	11 · · 8	7 · · 4	3 · · 0
BCD code	0 1 1 1	0 0 1 0	1 0 0 0	0 0 0 1	0 0 0 1	0 1 1 1	0 0 1 1	0 0 0 0
BCD Hex code	7	2	8	1	1	7	3	0

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

See also: BCD data

PLC types **Availability of** F82_DBCD **(see page 1326)**

Data types

Variable	Data type	Function
s	ANY32	binary data (source), range: 0 to 99,999,999
d	DWORD	32-bit area for 8-digit BCD code (destination)

Operands

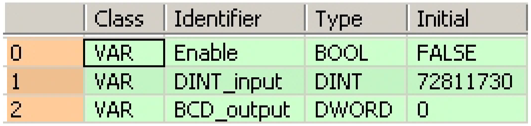
For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ 32-bit data specified by s outside the range of 0 (16#0) to 99999999 (16#5F5E0FF) is converted.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

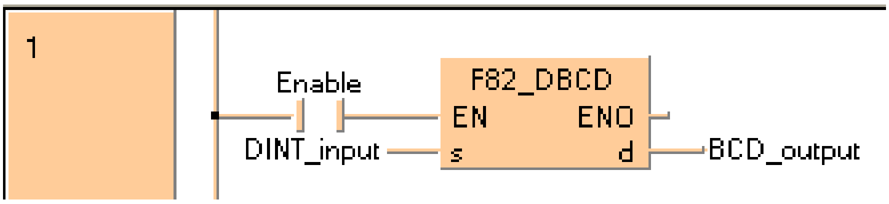
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Enable	BOOL	FALSE
1	VAR	DINT_input	DINT	72811730
2	VAR	BCD_output	DWORD	0

Body When the variable **Enable** is set to TRUE, the function is executed. The decimal value in **DINT_input** is converted to a BCD hexadecimal value and stored in the variable **BCD_output**. You may also assign a decimal, binary (prefix 2#), or hexadecimal (prefix 16#) value directly at the contact pin for s.

LD



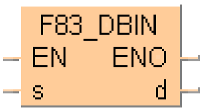
ST When programming with structured text, enter the following:

```
IF Enable THEN
    F82_DBCD(DINT_input, BCD_output);
END_IF;
```

F83_DBIN

8-digit BCD -> 32-bit BIN conversion

Description Converts the BCD code that expresses 8-digit decimals specified by **s** to 32-bit binary data if the trigger **EN** is in the ON-state. The converted result is stored in the area specified by **d**.



Source [s]: 16#72811730 (BCD)

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0	15 · · 12	11 · · 8	7 · · 4	3 · · 0
BCD code	0 1 1 1	0 0 1 0	1 0 0 0	0 0 0 1	0 0 0 1	0 1 1 1	0 0 1 1	0 0 0 0
BCD Hex code	7	2	8	1	1	7	3	0

32-bit area

⬇

Destination [d]: 72811730

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0	15 · · 12	11 · · 8	7 · · 4	3 · · 0
Binary data	0 0 0 0	0 1 0 0	0 1 0 1	0 1 1 1	0 0 0 0	0 1 0 0	1 1 0 1	0 0 1 0
Decimal	72811730							

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

See also: BCD data

PLC types **Availability of** F83_DBIN **(see page 1326)**

Data types

Variable	Data type	Function
s	DWORD	area for 8-digit BCD data (source)
d	ANY32	32-bit area for storing 32-bit data (destination)

Operands

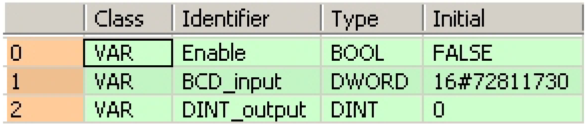
For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the data specified by **s** is not BCD data.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

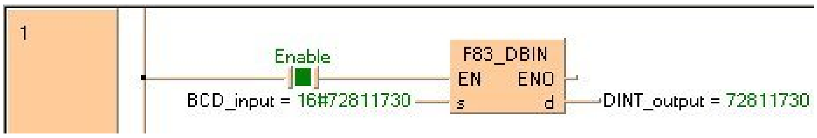
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Enable	BOOL	FALSE
1	VAR	BCD_input	DWORD	16#72811730
2	VAR	DINT_output	DINT	0

Body When the variable **Enable** is set to TRUE, the function is executed. The BCD value assigned to the variable **BCD_input** is converted to a decimal value and stored in the variable **DINT_output**.

LD



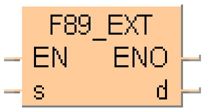
ST When programming with structured text, enter the following:

```
IF Enable THEN
    F83_DBIN(BCD_input, DINT_Output);
END_IF;
```

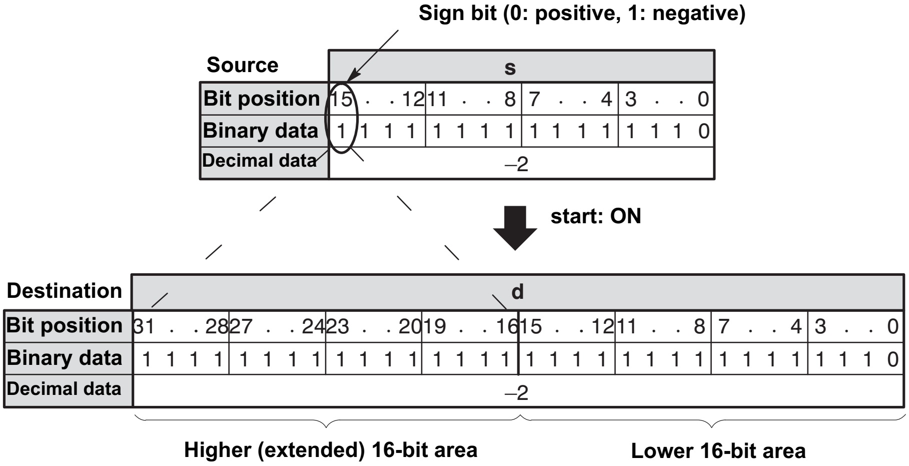
F89_EXT

16-bit data sign extension, INT -> DINT

Description 16-bit data is converted to 32-bit data without signs and values being changed. F89 copies the sign bit of the 16-bit data specified in **s** to all the bits of the higher 16-bit area (extended 16-bit area) in **d**.



If the sign bit (bit position 15) of the 16-bit data specified by **s** is 0, all higher 16 bits in the variable assigned to **d** will be 0. If the sign bit of **s** is 1, the higher 16 bits of **d** will be 1.


Sign bit (0: positive, 1: negative)
start: ON
Higher (extended) 16-bit area
Lower 16-bit area

Source	s			
Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
Binary data	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 0
Decimal data	–2			

Destination	d							
Bit position	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	11 . . 8	7 . . 4	3 . . 0
Binary data	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 0
Decimal data	–2							

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F89_EXT **(see page 1326)**

Data types

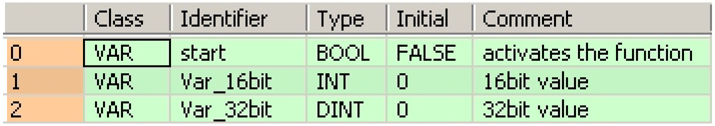
Variable	Data type	Function
s	ANY16	16-bit source data area, bit 15 is sign bit
d	ANY32	32-bit destination area, **s** copied to lower 16 bits, higher 16 bits filled with sign bit of **s**

Operands

For	Relay				T/C		Register			Constant
s	-	WY	WR	WL	SV	EV	DT	LD	FL	-
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

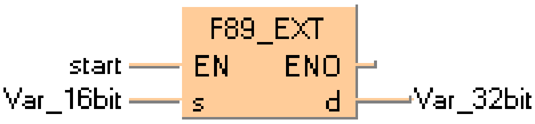
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	Var_16bit	INT	0	16bit value
2	VAR	Var_32bit	DINT	0	32bit value

Body When the variable **start** is set to TRUE, the function is carried out.

LD



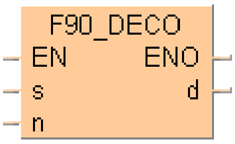
ST When programming with structured text, enter the following:

```
IF start THEN
    F89_EXT(Var_16bit, Var_32bit);
END_IF;
```

F90_DECO

Decode hexadecimal -> bit state

Description Decodes the contents of 16-bit data specified by **s** according to the contents of **n** if the trigger EN is in the ON-state. The decoded result is stored in the area starting with the 16-bit area specified by **d**.



n specifies the starting bit position and the number of bits to be decoded using hexadecimal data:

- Bit No. 0 to 3: number of bits to be decoded
- Bit No. 8 to 11: starting bit position to be decoded

(The bits No. 4 through No. 7 and No. 12 through No. 15 are invalid.)

e.g. when **n** = 16#0404, four bits beginning at bit position four are decoded.

Relationship between number of bits and occupied data area for decoded result:

Number of bits to be decoded	Data area required for the result	Valid bits in the area for the result
1	1-word	2-bit*
2	1-word	4-bit*
3	1-word	8-bit*
4	1-word	16-bit
5	2-word	32-bit
6	4-word	64-bit
7	8-word	128-bit
8	16-word	256-bit

*Invalid bits in the data area required for the result are set to 0.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F90_DECO **(see page 1326)**

Data types

Variable	Data type	Function
s	ANY16	source 16-bit area or equivalent constant to be decoded
n		control data to specify the starting bit position and number of bits to be decoded
d		starting 16-bit area for storing decoded data (destination)

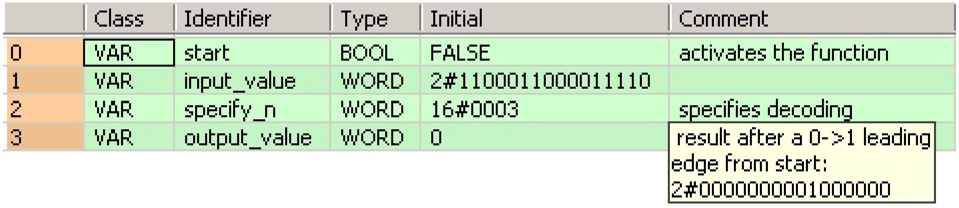
The variables **s, n** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s, n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

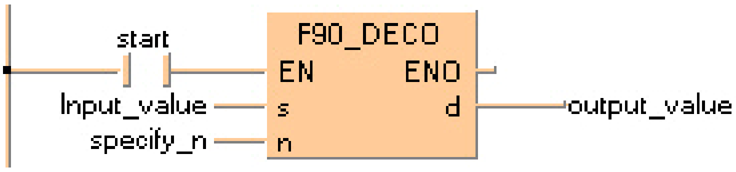
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	2#1100011000011110	
2	VAR	specify_n	WORD	16#0003	specifies decoding
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#0000000001000000

Body When the variable **start** is set to TRUE, the function is carried out.

LD



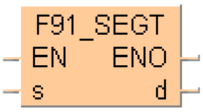
ST When programming with structured text, enter the following:

```
IF start THEN
    F90_DECO( s:= input_value ,
        n:= specify_n ,
        d=> output_value );
END_IF;
```

F91_SEGT

16-bit data 7-segment decode

Description Converts the 16-bit equivalent constant or 16-bit data specified by **s** to 4-digit data for 7-segment indication if the trigger **EN** is in the ON-state. The converted data is stored in the area starting with the 16-bit area specified by **d**. The data for 7-segment indication occupies 8 bits (1 byte) to express 1 digit.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F91_SEGT **(see page 1326)**

Data types

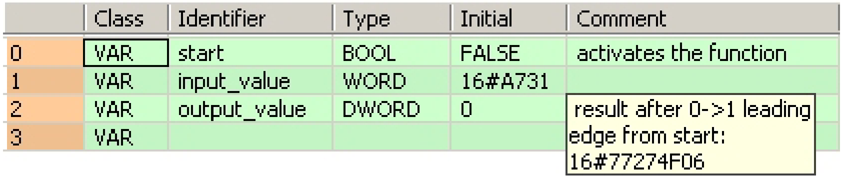
Variable	Data type	Function
s	ANY16	16-bit area or equivalent constant to be converted to 7-segment indication (source)
d	ANY32	32-bit area for storing 4-digit data for 7-segment indication (destination)

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

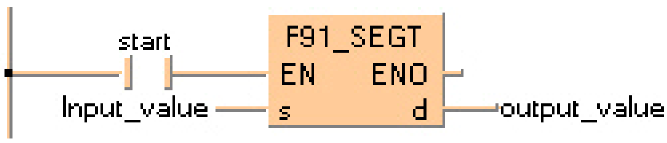
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	16#A731	
2	VAR	output_value	DWORD	0	result after 0->1 leading edge from start: 16#77274F06
3	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



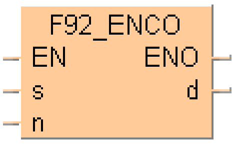
ST When programming with structured text, enter the following:

```
IF start THEN
    F91_SEGT(input_value, output_value);
END_IF;
```

F92_ENCO

Encode bit state -> hexadecimal

Description Encodes the contents of data specified by **s** according to the contents of **n** if the trigger **EN** is in the ON-state. The encoded result is stored in the 16-bit area specified by **d** starting with the specified bit position. Invalid bits in the area specified for the encoded result are set to 0.



b specifies the starting bit position of destination data **d** and the number of bits to be encoded using hexadecimal data:

Bit No. 0 to 3	number of bits to be encoded
Bit No. 8 to 11	starting bit position of destination data to be encoded

(The bits No. 4 through No. 7 and No. 12 through No. 15 are invalid.)

☞

- **Put at least one bit into the area to be checked to avoid an error message from the PLC.**
- **When several bits are set, the uppermost bit is evaluated.**

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F92_ENCO **(see page 1326)**

Data types

Variable	Data type	Function
s	ANY16	starting 16-bit area to be encoded (source)
n		control data to specify the starting bit position and number of bits to be encoded
d		16-bit area for storing encoded data (destination)

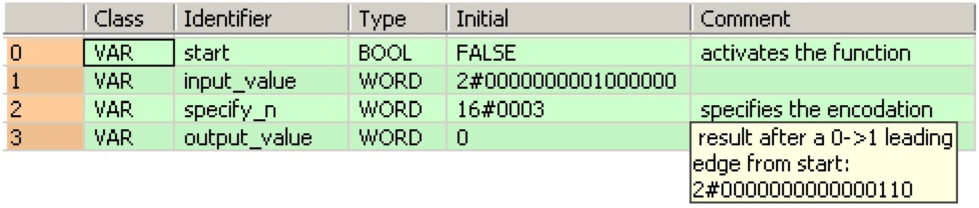
The variables **s, n** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

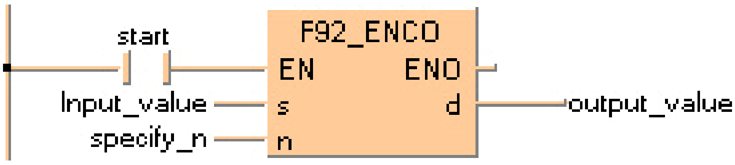
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	2#0000000001000000	
2	VAR	specify_n	WORD	16#0003	specifies the encodation
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#0000000000000110

Body When the variable **start** is set to TRUE, the function is carried out.

LD



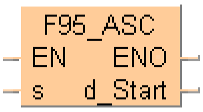
ST When programming with structured text, enter the following:

```
IF start THEN
    F92_ENCO( s:= input_value ,
         n:= specify_n ,
         d=> output_value );
END_IF;
```

F95_ASC

12 Character -> ASCII transfer

Description Converts the character constants specified by **s** to hexadecimal ASCII code. The hexadecimal code is stored in 6 words starting from the 16-bit area specified by **d**.



[s]	Character constants	ABC1230˽DEF

⬇

	Data register	d[5]		d[4]		d[3]		d[2]		d[1]		d[0]	
[d]	ASCII HEX code	2 0	4 6	4 5	4 4	2 0	3 0	3 3	3 2	3 1	4 3	4 2	4 1
	ASCII character		F	E	D		0	3	2	1	C	B	A

SPACE

☞ **If the number of character constants specified by s is less than 12, the ASCII code 16#20 (SPACE) is stored in the extra destination area, e.g. s = ’12345’, d[0] = 3231, d[1] = 3433, d[2] = 2035, d[3] = 2020, d[4] = 2020, d[5] = 2020.**

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of F95_ASC (see page 1326)**

Data types

Variable	Data type	Function
s	constant, no variables possible	Character constants, max. 12 letters (source).
d	ANY16	Starting 16-bit area for storing 6-word ASCII code (destination).

Operands

For	Relay				T/C		Register			Constant
s	-	-	-	-	-	-	-	-	-	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

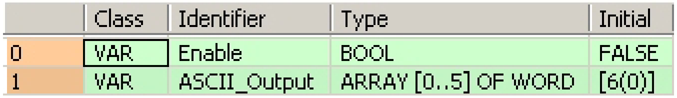
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the last area for ASCII code exceeds the limit (6 words: six 16-bit areas).
R9008	%MX0.900.8	for an instant	

ASCII Hex-Code

									b7								
									b6	0	0	0	0	1	1	1	1
									b5	0	0	1	1	0	0	1	1
									b4	0	1	0	1	0	1	0	1
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	ASCII HEX code		Most significant digit							
										0	1	2	3	4	5	6	7
				0	0	0	0	Least significant digit	0	NUL	DEL	SPACE	0	@	P		p
				0	0	0	1		1	SOH	DC1	!	1	A	Q	a	q
				0	0	1	0		2	STX	DC2	″	2	B	R	b	r
				0	0	1	1		3	ETX	DC3	#	3	C	S	c	s
				0	1	0	0		4	EOT	DC4	$	4	D	T	d	t
				0	1	0	1		5	ENQ	NAK	%	5	E	U	e	u
				0	1	1	0		6	ACK	SYN	&	6	F	V	f	v
				0	1	1	1		7	BEL	ETB	′	7	G	W	g	w
				1	0	0	0		8	BS	CAN	(	8	H	X	h	x
				1	0	0	1		9	HT	EM	)	9	I	Y	i	y
				1	0	1	0		A	LF	SUB	*	:	J	Z	j	z
				1	0	1	1		B	VT	ESC	+	;	K	[	k	{
				1	1	0	0		C	FF	FS	,	<	L	\	l	?
				1	1	0	1		D	CR	GS	–	=	M	]	m	}
				1	1	1	0		E	SO	RS	.	>	N	∧	n	~
				1	1	1	1		F	SI	US	/	?	O	_	o	DEL

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

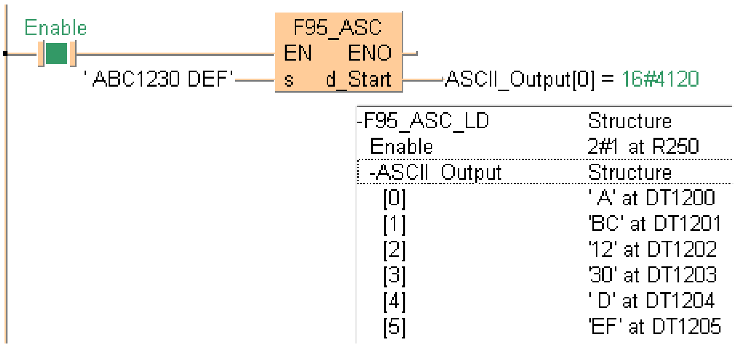
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Enable	BOOL	FALSE
1	VAR	ASCII_Output	ARRAY [0..5] OF WORD	[6(0)]

Body When the variable **Enable** is enabled, the character constants entered at the input s are converted to ASCII code and stored in the variable **ASCII_Output**.

LD



-F95_ASC_LD	Structure
Enable	2#1 at R250
-ASCII_Output	Structure
[0]	' A' at DT1200
[1]	'BC' at DT1201
[2]	'12' at DT1202
[3]	'30' at DT1203
[4]	' D' at DT1204
[5]	'EF' at DT1205

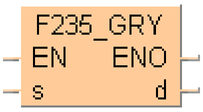
ST When programming with structured text, enter the following:

```
IF Enable THEN
    F95_ASC( s:= 'ABC1230 DEF' ,
         d_Start=> ASCII_Output[0] );
END_IF;
```

F235_GRY

16-bit data -> 16-bit Gray code

Description The function converts a value at input **s** to a gray code value. The result of the conversion is returned at output **d**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F235_GRY **(see page 1323)**

Data types

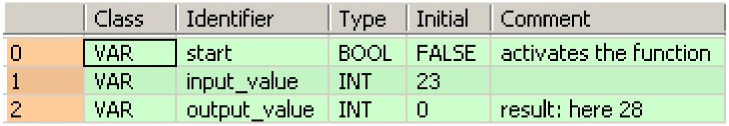
Variable	Data type	Function
s	ANY16	source data to be converted
d		destination for storing gray codes

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

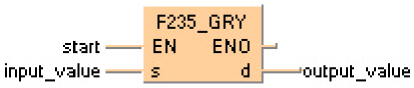


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	INT	23	
2	VAR	output_value	INT	0	result: here 28

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD



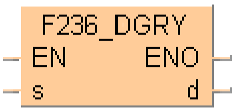
ST When programming with structured text, enter the following:

```
IF start THEN
    F235_GRY(input_value, output_value);
END_IF;
```

F236_DGRY

32-bit data -> 32-bit Gray code

Description The function converts a value at input **s** to a gray code value. The result of the conversion is returned at output **d**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F236_DGRY **(see page 1323)**

Data types

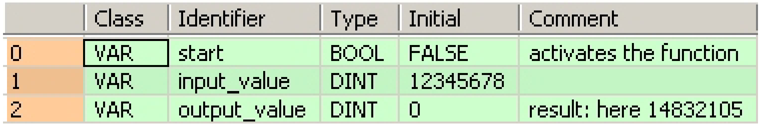
Variable	Data type	Function
s	ANY32	source data to be converted
d		destination for storing gray code

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

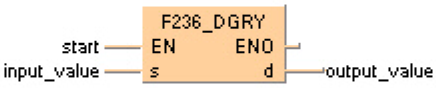


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	DINT	12345678	
2	VAR	output_value	DINT	0	result: here 14832105

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD

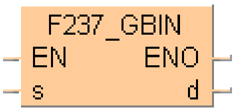


ST When programming with structured text, enter the following:

```
IF start THEN
    F236_DGRY(input_value, output_value);
END_IF;
```

F237_GBIN 16-bit Gray code -> 16-bit binary data

Description The function converts a gray-code value at input **s** to binary data. The result of the conversion is returned at output **d**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F237_GBIN **(see page 1323)**

Data types

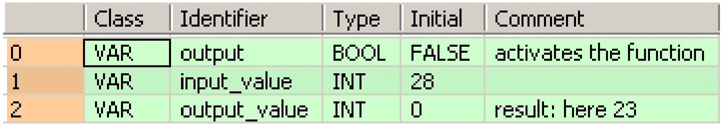
Variable	Data type	Function
s	ANY16	source area to gray code
d		destination for storing converted data

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

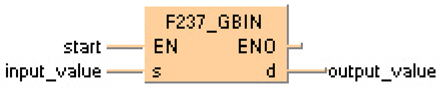


	Class	Identifier	Type	Initial	Comment
0	VAR	output	BOOL	FALSE	activates the function
1	VAR	input_value	INT	28	
2	VAR	output_value	INT	0	result: here 23

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD



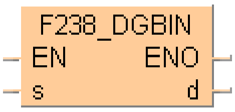
ST When programming with structured text, enter the following:

```
IF start THEN
    F237_GBIN(input_value, output_value);
END_IF;
```

F238_DGBIN

32-bit Gray code -> 32-bit binary data

Description The function converts a gray-code value at input **s** to binary data. The result of the conversion is returned at output **d**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F238_DGBIN **(see page 1323)**

Data types

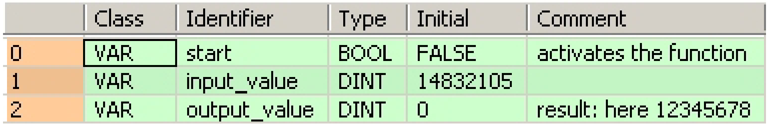
Variable	Data type	Function
s	ANY32	source area for gray code
d		destination area for storing converted data

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

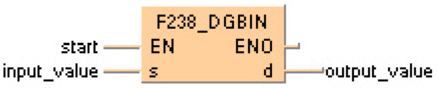


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	DINT	14832105	
2	VAR	output_value	DINT	0	result: here 12345678

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD



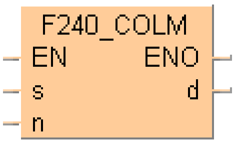
ST When programming with structured text, enter the following:

```
IF start THEN
    F238_DGBIN(input_value, output_value);
END_IF;
```

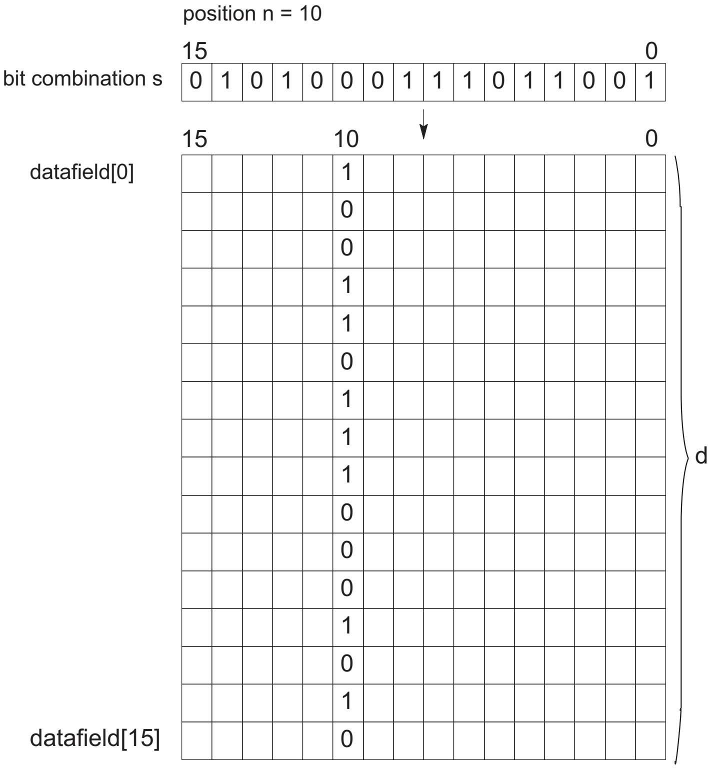
F240_COLM

Bit line to bit column conversion

Description The function creates a bit column out of a value given at input **s** that is returned within an ARRAY at output **d**. The position of the column in the ARRAY is specified at input **n**. The value assigned at **n** can be between 0 and 15.



The bits of the ARRAY that are not overwritten by the input value (input **s**) are not effected.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F240_COLM **(see page 1323)**

Data types

Variable	Data type	Function
s	ANY16	source
n		specifies bit position
d	ARRAY [0..15] of ANY16	destination area that will be rewritten with bit column

Operands

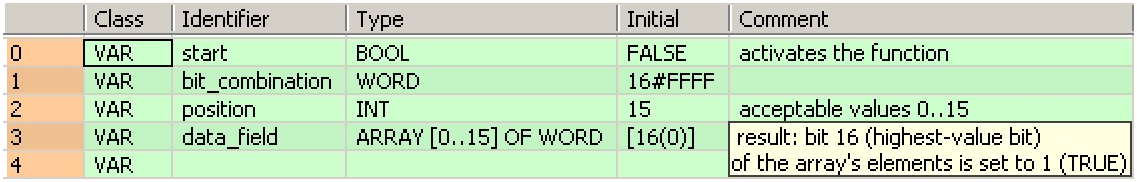
For	Relay				T/C		Register			Constant
s, n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the bit position specified at input n is not between 0 and 15 ▪ the conversion operation results in an overflow of the address area at output **d**.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

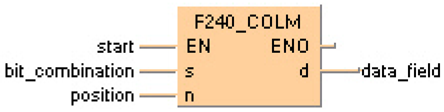


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	bit_combination	WORD	16#FFFF	
2	VAR	position	INT	15	acceptable values 0..15
3	VAR	data_field	ARRAY [0..15] OF WORD	[16(0)]	result: bit 16 (highest-value bit) of the array's elements is set to 1 (TRUE)
4	VAR				

In this example **bit_combination** and **position** are declared as input variables. However, you can write constants directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD



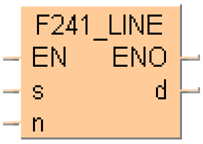
ST When programming with structured text, enter the following:

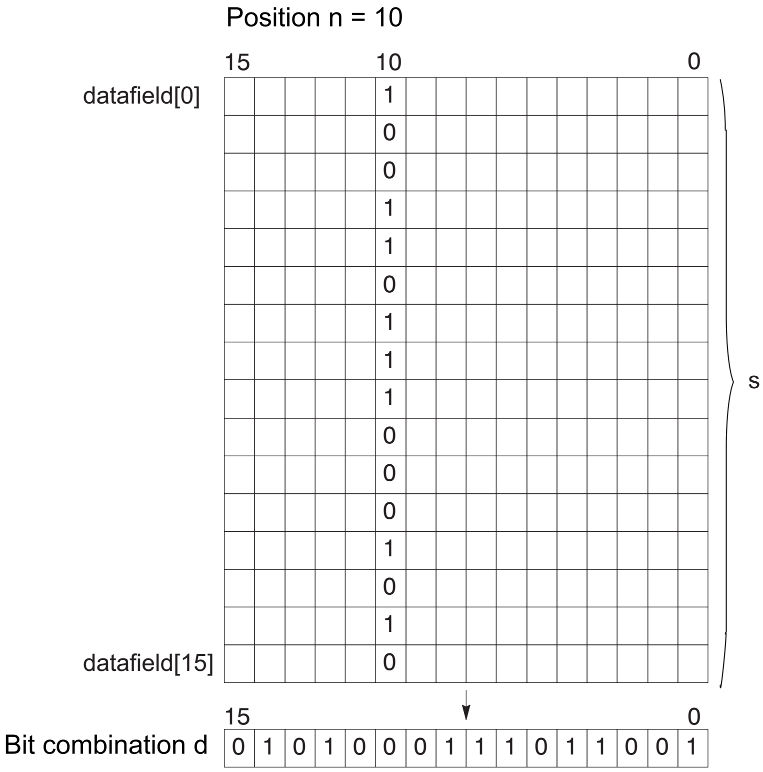
```
IF start THEN
    F240_COLM( s:= bit_combination,
         n:= position,
         d=> data_field );
END_IF;
```

F241_LINE

Bit column to bit line conversion

Description The function converts a bit column out of an ARRAY at input **s** and returns it at output **d**. The position at which the conversion takes place is specified at input **n**. The value assigned at input **n** should be between 0 and 15.





This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F241_LINE **(see page 1323)**

Data types

Variable	Data type	Function
s	ARRAY [0..15] OF ANY16	source area where bit column will be read
n	ANY16	specifies bit position
d		destination area for storing converted data

Operands

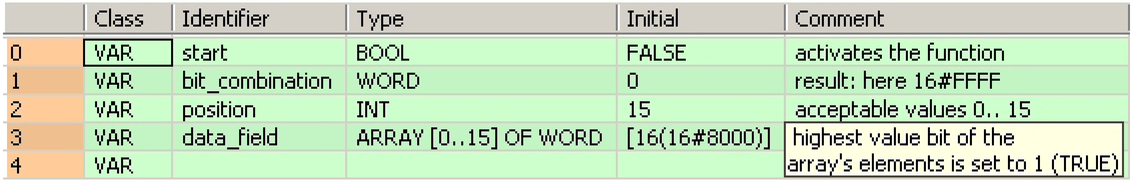
For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the bit position specified at input **n** is not between 0 and 15
R9008	%MX0.900.8	for an instant	▪ an overflow of the address area at input **s** occurs.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	bit_combination	WORD	0	result: here 16#FFFF
2	VAR	position	INT	15	acceptable values 0.. 15
3	VAR	data_field	ARRAY [0..15] OF WORD	[16(16#8000)]	highest value bit of the array's elements is set to 1 (TRUE)
4	VAR				

In this example **bit_combination** and **position** are declared as input variables. However, you can

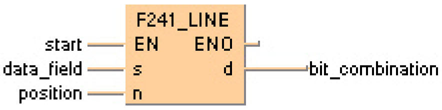
Body When the variable **start** is set to TRUE, the function is carried out.

write constants directly at the input contact of the function instead.

LD

ST When programming with structured text, enter the following:

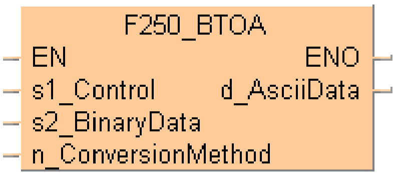
```
IF start THEN
    F241_LINE( s:= data_field ,
         n:= position ,
         d=> bit_combination );
END_IF;
```



F250_BTOA

Binary -> ASCII conversion

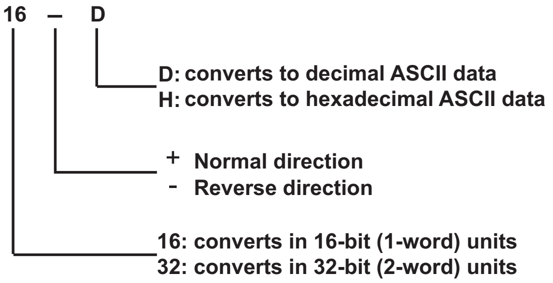
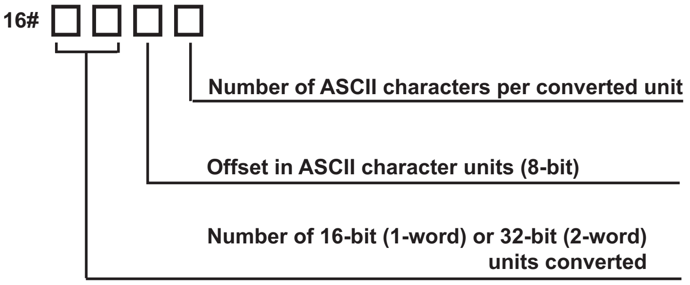
Description Converts 16-bit/32-bit binary data stored in the area specified by **s2_BinaryData** to ASCII code. The conversion method is specified by **n_ConversionMethod** according to the four control characters of **s1_Control**. The converted result is stored in the area specified by **d_AsciiData**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F250_BTOA **(see page 1323)**

Data types

Variable	Data type	Function
s1_Control	STRING	Control string  16 – D D: converts to decimal ASCII data H: converts to hexadecimal ASCII data + Normal direction - Reverse direction 16: converts in 16-bit (1-word) units 32: converts in 32-bit (2-word) units
s2_BinaryData	ANY	Starting area for storing binary data
n_Conversion Method	ANY16	Conversion method  16# Number of ASCII characters per converted unit Offset in ASCII character units (8-bit) Number of 16-bit (1-word) or 32-bit (2-word) units converted (for details, see explanation following the tables)
d_AsciiData	ANY	Starting area for storing ASCII data

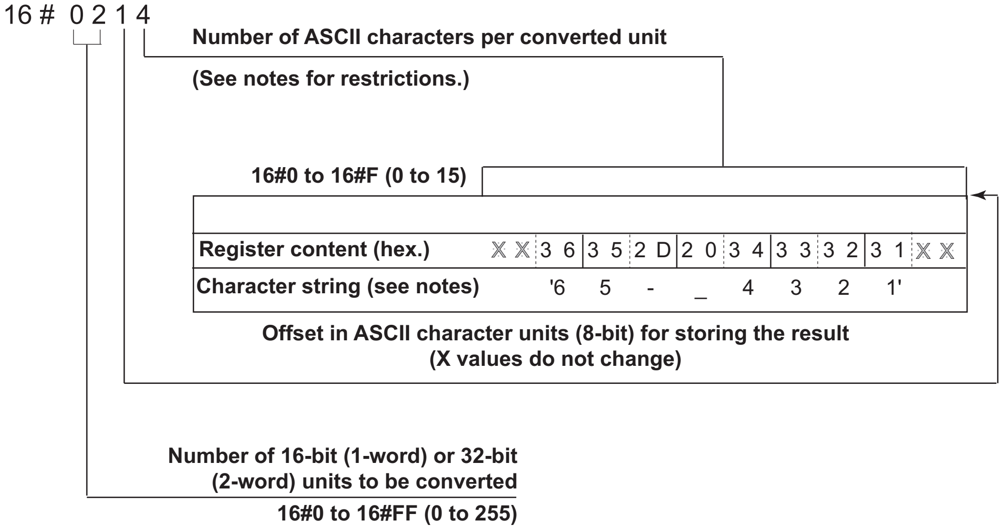
Operands

For	Relay				T/C		Register			Constant
s1_Control	WX	WY	WR	WL	SV	EV	DT	LD	-	-
s2_BinaryData	WX	WY	WR	WL	SV	EV	DT	LD	-	-
n_Conversion Method	-	WY	WR	WL	SV	EV	DT	LD	-	dec. or hex.
d_AsciiData	-	WY	WR	WL	SV	EV	DT	LD	-	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ there is an error in the control string specified by **s1_Control**. ▪ normal direction (+) is specified in **s1_Control** when the format is decimal. ▪ the number of ASCII characters per converted unit specified by **n_ConversionMethod** exceeds 4 for 16-bit data or 8 for 32-bit data when hexadecimal format is specifed by **s1_Control**. ▪ 0 is specified for the no. of 16- or 32-bit (1- or 2-word) units to be converted in **n_ConversionMethod**. ▪ the number of 16- or 32-bit decimal numbers to be converted specified by **n_ConversionMethod** exceeds the area for storing ASCII data. ▪ the converted result exceeds the area.
R9008	%MX0.900.8	for an instant	

■ Explanation of the conversion method, e.g. n_ConversionMethod = 16#0214


16 # 0 2 1 4
Number of ASCII characters per converted unit
(See notes for restrictions.)
16#0 to 16#F (0 to 15)
Offset in ASCII character units (8-bit) for storing the result
(X values do not change)
Number of 16-bit (1-word) or 32-bit
(2-word) units to be converted
16#0 to 16#FF (0 to 255)

Register content (hex.)	X X	3 6	3 5	2 D	2 0	3 4	3 3	3 2	3 1	X X
Character string (see notes)		'6	5	-	_	4	3	2	1'	

Register content (hex.)	F F C 8	0 4 D 2
Values (dec.)	-56	1234

☞ **About the number of ASCII characters (8-bit) per converted unit**

- When converting 16-bit binary units to hexadecimal ASCII data:
- Range: 16#1 to 16#4.
- When a range of less than 16#4 is set, the specified number of characters from the lower bytes are stored. If the original binary unit data cannot be accommodated by a setting less than 16#4, an error occurs.
- When converting 32-bit binary units to hexadecimal **ASCII** data:
- Range: 16#1 to 16#8.
- When a range of less than 16#8 is set, the specified number of characters from the lower bytes are stored. If the original binary data cannot be accommodated by a setting less than 16#8, an error occurs.
- When converting binary units to decimal **ASCII** data:
- Range: 16#1 to 16#F.
- Source data is treated as signed binary data. If it is a negative number, a minus sign "-" is added. A space "␣" will be stored in the leading blanks if the area specified in d_AsciiData is larger than the number of **ASCII** characters per converted unit.

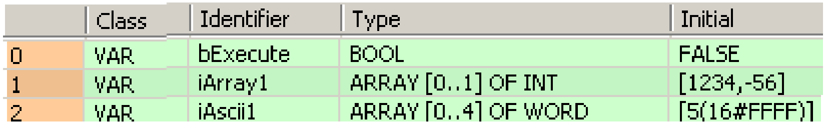
Conversion examples:

Binary data			**s1_ Con-trol**	**n_Con-version Method**	**Result ASCII data**				**Comment**
Data type	**Offs. in 16-bit word units**	**Hex. value**			**D+ 3**	**D+ 2**	**D+ 1**	**D**	
INT, WORD	0	16#5678	16+H	16#204	21	43	65	87	Normal direction. 2 x 4 ASCII characters.
	1	16#1234							
INT, WORD	0	16#5678	16-H	16#204	43	21	87	65	Reverse direction. 2 x 4 ASCII characters.
	1	16#1234							
INT, WORD	0	16#0456	16+H	16#203	XX	13	24	65	Normal direction. 2 x 3 **ASCII** characters.
	1	16#0123							
INT, WORD	0	16#0456	16-H	16#203	XX	32	16	54	Reverse direction. 2 x 3 **ASCII** characters.
	1	16#0123							
DINT, DWORD	0	16#12345678	32+H	16#108	21	43	65	87	Normal direction. 1 x 8 **ASCII** characters.
DINT, DWORD	0	16#12345678	32-H	16#108	87	65	43	21	Reverse direction. 1 x 8 **ASCII** characters.
DINT, DWORD	0	16#00012345	32+H	16#105	XX	X1	32	54	Normal direction. 1 x 5 **ASCII** characters.
DINT, DWORD	0	16#00012345	32-H	16#105	XX	X5	43	21	Reverse direction. 1 x 5 **ASCII** characters.

'**X**' values do not change.

Example In this example, the same POU header is used for all programming languages.

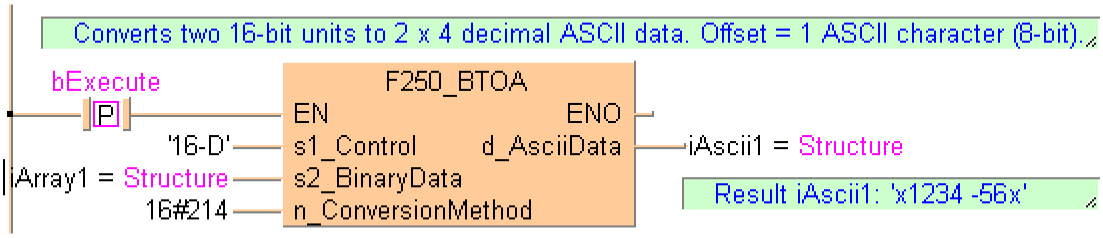
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bExecute	BOOL	FALSE
1	VAR	iArray1	ARRAY [0..1] OF INT	[1234,-56]
2	VAR	iAscii1	ARRAY [0..4] OF WORD	[5(16#FFFF)]

Body When **bExecute** is set to TRUE, the instruction is carried out. It converts two 16-bit units to 2 x 4 decimal ASCII data. Offset = 1 ASCII character (8-bit).

LD



Converts two 16-bit units to 2 x 4 decimal ASCII data. Offset = 1 ASCII character (8-bit).

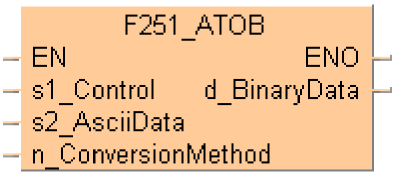
Result iAscii1: 'x1234 -56x'

ST When programming with structured text, enter the following:

```
IF DF(bExecute) THEN
    F250_BTOA(s1_Control := '16-D',
        s2_BinaryData := iArray1,
        n_ConversionMethod := 16#214,
        d_AsciiData => iAscii1);

END_IF;
```

F251_ATOB ASCII -> Binary conversion

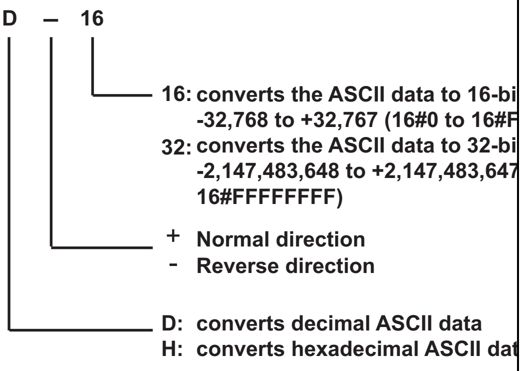
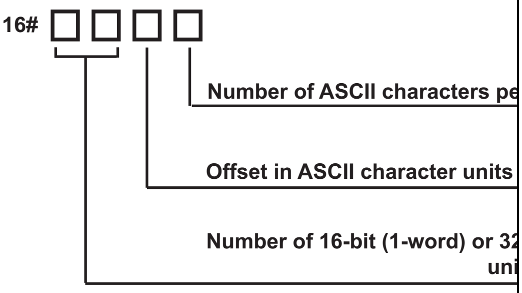
Description Converts ASCII code stored in the area specified by **s2_AsciiData** to 16-bit/32-bit binary data. The conversion method is specified by **n_ConversionMethod** according to the four control characters of **s1_Control**. The converted result is stored in the area specified by **d_BinaryData**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of F251_ATOB (see page 1323)**

Data types

Variable	Data type	Function
s1_Control	STRING	Control string  D – 16 16: converts the ASCII data to 16-bi -32,768 to +32,767 (16#0 to 16#F 32: converts the ASCII data to 32-bi -2,147,483,648 to +2,147,483,647 16#FFFFFFFF) + Normal direction - Reverse direction D: converts decimal ASCII data H: converts hexadecimal ASCII dat
s2_AsciiData	ANY	Starting area for storing ASCII data
n_ConversionMethod	ANY16	Conversion method  16# □□□□ Number of ASCII characters pe Offset in ASCII character units Number of 16-bit (1-word) or 32 uni (for details, see explanation following the tables)
d_BinaryData	ANY	Starting area for storing binary data

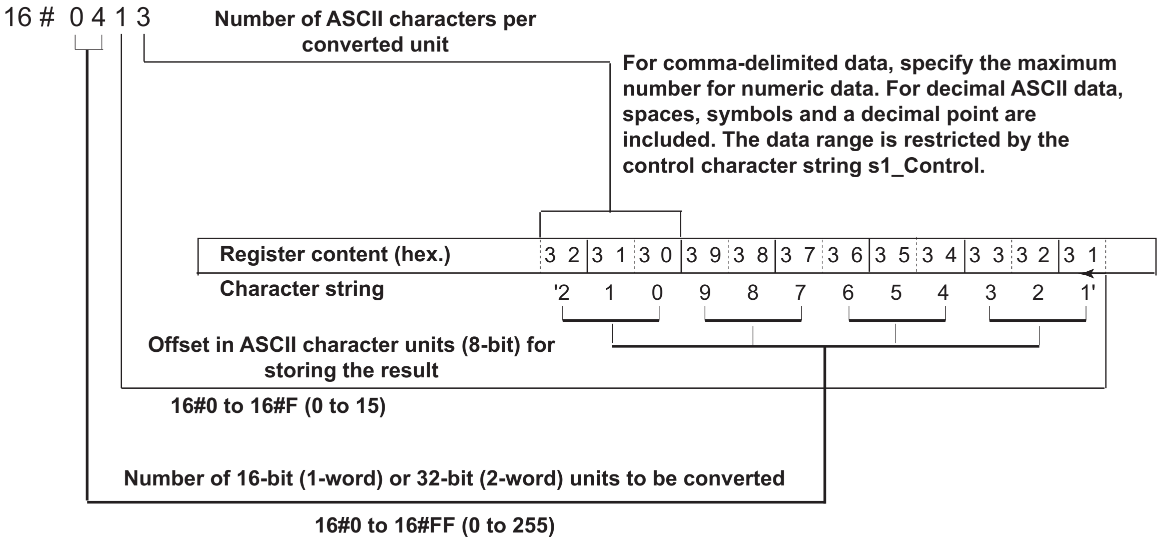
Operands

For	Relay				T/C		Register			Constant
s1_Control	WX	WY	WR	WL	SV	EV	DT	LD	-	-
s2_AsciiData	WX	WY	WR	WL	SV	EV	DT	LD	-	-
n_Conversion Method	-	WY	WR	WL	SV	EV	DT	LD	-	dec or hex
d_BinaryData	-	WY	WR	WL	SV	EV	DT	LD	-	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ there is an error in the control string specified by **s1_Control**. ▪ normal direction (+) is specified in **s1_Control** when the format is decimal. ▪ the number of ASCII characters per converted unit specified by **n_ConversionMethod** exceeds 4 for 16-bit data or 8 for 32-bit data when hexadecimal format is specifed by **s1_Control**. ▪ 0 is specified for the no. of 16- or 32-bit (1- or 2-word) units to be converted in **n_ConversionMethod**. ▪ the number of 16- or 32-bit decimal numbers to be converted specified by **n_ConversionMethod** exceeds the area for storing ASCII data. ▪ the converted result exceeds the area.
R9008	%MX0.900.8	for an instant	

■ Explanation of the conversion method, e.g. n_ConversionMethod = 16#0413 for ASCII data '0123456789012'


16 # 0 4 1 3
Number of ASCII characters per converted unit
For comma-delimited data, specify the maximum number for numeric data. For decimal ASCII data, spaces, symbols and a decimal point are included. The data range is restricted by the control character string s1_Control.
Register content (hex.)
3 2 3 1 3 0 3 9 3 8 3 7 3 6 3 5 3 4 3 3 3 2 3 1
Character string
'2 1 0 9 8 7 6 5 4 3 2 1'
Offset in ASCII character units (8-bit) for storing the result
16#0 to 16#F (0 to 15)
Number of 16-bit (1-word) or 32-bit (2-word) units to be converted
16#0 to 16#FF (0 to 255)

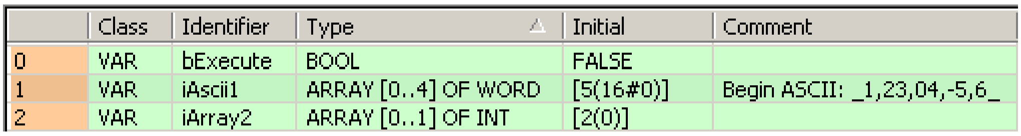
Conversion examples for ASCII **data '0123456789ABCDEF'**

n_Conversion method	s1_Control	ASCII data	Binary data			Comment
			Data type	Offs. in 16-bit word units	Hex. value	
H+16	16#404	0123 4567 89AB CDEF	INT, WORD	0	16#2301	Normal direction 4 x 4 ASCII characters
				1	16#6745	
				2	16#AB89	
				3	16#EFCD	
H+16	16#404		INT, WORD	0	6#0123	Reverse direction 4 x 4 ASCII characters
				1	16#4567	
				2	16#89AB	
				3	16#CDEF	
H+16	16#403		INT, WORD	0	16#*201	Normal direction 3 x 4 ASCII characters
				1	16#*534	
				2	16#*867	
				3	16#*B9A	
H-16	16#403		INT, WORD	0	16#*012	Reverse direction 3 x 4 ASCII characters
				1	16#*345	
				2	16#*678	
				3	6#*9AB	
H+32	16#208		DINT, DWORD	0	16#67452301	Normal direction 8 x 2 ASCII characters
				2	16#EFCDAB89	
H-32	16#208		DINT, DWORD	0	16#01234567	Reverse direction 8 x 2 ASCII characters
				2	16#89ABCDEF	
H+32	16#205		DINT, DWORD	0	16#***42301	Reverse direction 8 x 2 ASCII characters
				2	16#***97856	
H-32	16#205		DINT, DWORD	0	16#***01234	Reverse direction 5 x 2 ASCII characters
				2	16#***56789	

*The extra characters become '0'.

Example In this example, the same POU header is used for all programming languages.

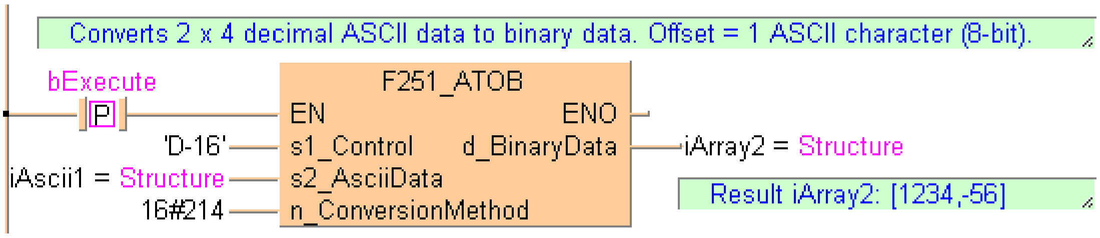
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	bExecute	BOOL	FALSE	
1	VAR	iAscii1	ARRAY [0..4] OF WORD	[5(16#0)]	Begin ASCII: _1,23,04,-5,6_
2	VAR	iArray2	ARRAY [0..1] OF INT	[2(0)]	

Body When **bExecute** is set to TRUE, the instruction is carried out. It converts 2 x 4 decimal ASCII characters to binary data. Offset = 1 ASCII character (8-bit)

LD



Converts 2 x 4 decimal ASCII data to binary data. Offset = 1 ASCII character (8-bit).

Result iArray2: [1234,-56]

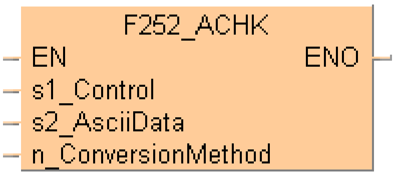
ST When programming with structured text, enter the following:

```
IF DF(bExecute) THEN
    F251_ATOB(s1_Control := 'D-16',
    s2_AsciiData := iAscii1,
    n_ConversionMethod := 16#214,
    d_BinaryData => iArray2);
END_IF;
```

F252_ACHK

ASCII data check

Description Checks whether the ASCII codes stored in the area specified by **s2_AsciiData** can be converted correctly using the conversion method specified in by **n_ConversionMethod** and the 4 control characters specified by **s1_Control**.

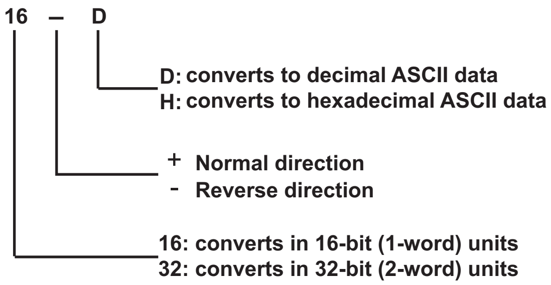


- If the results are correct, the special internal relay (R900B) turns on.
- If the results are incorrect, the special internal relay (R900B) turns off.

For an detailed description of **s1_Control** and **n_ConversionMethod**, please refer to F251_ATOB.

PLC types **Availability of** F252_ACHK **(see page 1323)**

Data types

Variable	Data type	Function
s1_Control	STRING	Control string  **16 – D** **D: converts to decimal ASCII data** **H: converts to hexadecimal ASCII data** **+ Normal direction** **- Reverse direction** **16: converts in 16-bit (1-word) units** **32: converts in 32-bit (2-word) units**
s2_AsciiData	ANY	Starting area for storing ASCII data
n_Conversion Method	ANY16	16-bit equivalent constant or 16-bit area for storing conversion method

Operands

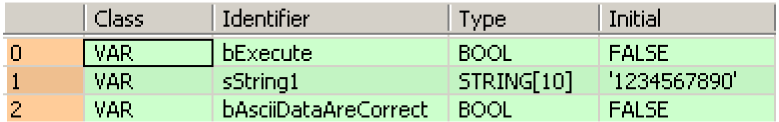
For	Relay				T/C		Register			Constant
s1_Control	WX	WY	WR	WL	SV	EV	DT	LD	-	-
s2_AsciiData	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
n_Conversion Method	WX	WY	WR	WL	SV	EV	DT	LD	-	dec or hex

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ there is an error in the control string specified by **s1_Control**. ▪ normal direction (+) is specified in **s1_Control** when the format is decimal. ▪ the number of ASCII characters per converted ▪ unit specified by **n_ConversionMethod** exceeds 4 for 16-bit data or 8 for 32-bit data when hexadecimal format is specifed by **s1_Control**. ▪ 0 is specified for the no. of 16- or 32-bit (1- or 2-word) units to be converted in **n_ConversionMethod**. ▪ the number of 16- or 32-bit decimal numbers to be converted specified ▪ by **n_ConversionMethod** exceeds the area for storing ASCII data. ▪ the converted result exceeds the area.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages.

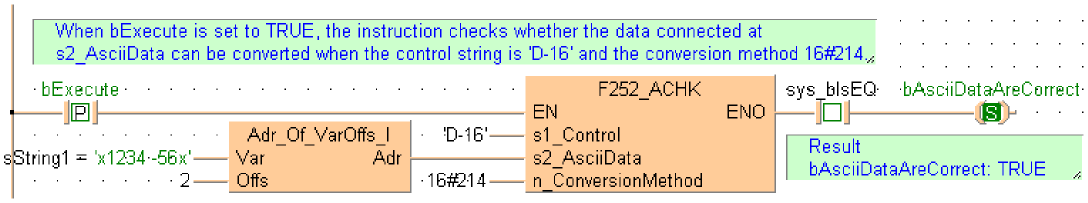
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bExecute	BOOL	FALSE
1	VAR	sString1	STRING[10]	'1234567890'
2	VAR	bAsciiDataAreCorrect	BOOL	FALSE

Body When **bExecute** is set to TRUE, the instruction checks whether the data connected at **s2_AsciiData** can be converted when the control string is 'D-16' and the conversion method 16#214.

LD



When bExecute is set to TRUE, the instruction checks whether the data connected at s2_AsciiData can be converted when the control string is 'D-16' and the conversion method 16#214

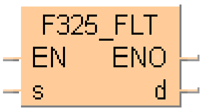
Result
bAsciiDataAreCorrect: TRUE

ST When programming with structured text, enter the following:

```
IF DF(bExecute) THEN
    F252_ACHK(s1_Control := 'D-16', s2_AsciiData := Adr_Of_VarOffs(Var
:= sString1, Offs := 2), n_ConversionMethod := 16#214);
    IF (sys_bIsEQ) THEN
        bAsciiDataAreCorrect := TRUE;
    END_IF;
END_IF;
```

F325_FLT 16-Bit Integer Data to Floating Point Data Conversion

Description Converts the 16-bit integer data with sign specified by **s** to real number data. The converted data is stored in **d**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F325_FLT **(see page** 1324**)**

- **F325_FLT cannot be programmed in the interrupt program.**
- **Instead of using F325_FLT, you can use variables of the type REAL with the more flexible instruction INT_TO_REAL (see page 193).**

Data types

Variable	Data type	Function
s	INT	16-bit integer data (source).
d	REAL	Floating point real number data for result (destination).

Operands

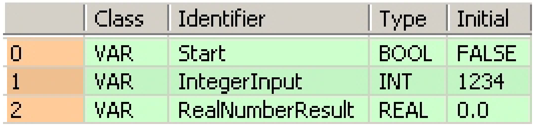
For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the result of processing is recognized as "0".

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

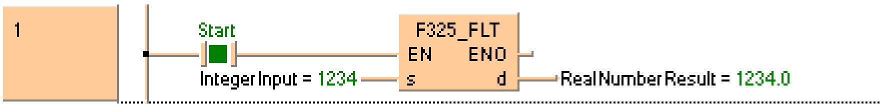
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Start	BOOL	FALSE
1	VAR	IntegerInput	INT	1234
2	VAR	RealNumberResult	REAL	0.0

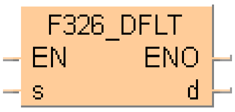
Body When the variable **Start** is set to TRUE, the integer value entered for the variable **IntegerInput** is converted to floating point data, and the result is stored at the address assigned by the compiler to the variable **RealNumberResult**. The monitor value icon is activated for both the LD and IL bodies.

LD



F326_DFLT 32-Bit Integer Data to Floating Point Data Conversion

Description Converts the 32-bit integer data with sign specified by **s** to real number data. The converted data is stored in **d**.



```
F326_DFLT
EN      ENO
s         d
```

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F326_DFLT **(see page** 1324**)**

☞

- **F326_DFLT cannot be programmed in the interrupt program.**
- **Instead of using F326_DFLT, you can use variables of the type REAL with the more flexible instruction DINT_TO_REAL (see page 194).**

Data types

Variable	Data type	Function
s	DINT	32-bit integer data (source).
d	REAL	Floating point real number data for result (destination).

Operands

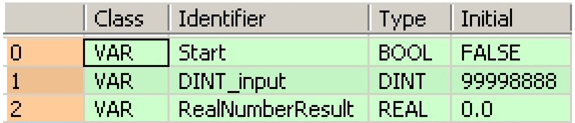
For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ there are too many significant digits in mantissa of converted real number data.
R9009	%MX0.900.9	for an instant	▪ the result of processing is recognized as "0".

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

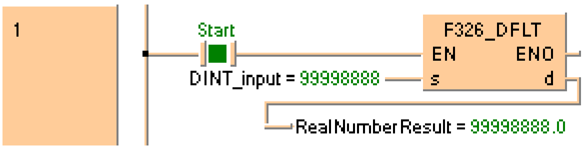
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Start	BOOL	FALSE
1	VAR	DINT_input	DINT	99998888
2	VAR	RealNumberResult	REAL	0.0

Body When the variable **Start** is set to TRUE, the double integer value entered for the variable **DINT_input** is converted to floating point data, and the result is stored at the address assigned by the compiler to the variable **RealNumberResult**. The monitor value icon is activated for both the LD and IL bodies.

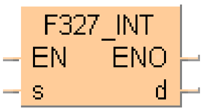
LD


1
Start
F326_DFLT
EN
ENO
DINT_input = 99998888
s
d
RealNumberResult = 99998888.0

F327_INT

Floating point data -> 16-bit integer data (the largest integer not exceeding the floating point data)

Description The function converts a floating point data at input **s** in the range -32767.99 to 32767.99 into integer data (including +/- sign). The result of the function is returned at output **d**.



The converted integer value at output **d** is always less than or equal to the floating point value at input **s**:

- When there is a positive floating point value at the input, a positive pre-decimal value is returned at the output.
- When there is a negative floating point value at the input, the next smallest pre-decimal value is returned at the output.
- If the floating point value has only zeros after the decimal point, its pre-decimal point value is returned.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types Availability of F327_INT **(see page 1324)**

Data types

Variable	Data type	Function
s	REAL	source REAL number data (2 words)
s2	INT, WORD	destination for storing converted data

Operands

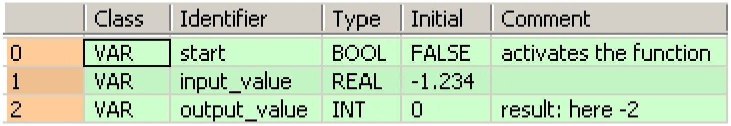
For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the value at input **s** is not a REAL number, or the converted result exceeds the 16-bit area at output **d**.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	for an instant	▪ the result is 0.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

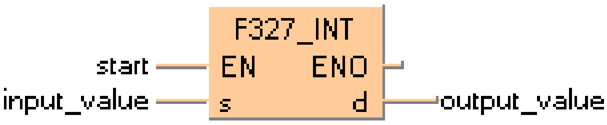


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	-1.234	
2	VAR	output_value	INT	0	result: here -2

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out. It converts the floating point value -1.234 into the whole number value -2, which is transferred to the variable **output_value** at the output. Since the whole number may not exceed the floating point value, the function rounds down here.

LD



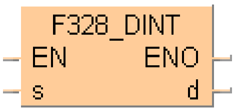
ST When programming with structured text, enter the following:

```
IF start THEN
    F327_INT(input_value, output_value);
END_IF;
```

F328_DINT

Floating point data -> 32-bit integer data (the largest integer not exceeding the floating point data)

Description The function converts a floating point data at input **s** in the range -2147483000 to 214783000 into integer data (including +/- sign). The result of the function is returned at output **d**.



The converted integer value at output **d** is always less than or equal to the floating point value at input **s**:

- When there is a positive floating point value at the input, a positive pre-decimal value is returned at the output.
- When there is a negative floating point value at the input, the next smallest pre-decimal value is returned at the output.
- If the floating point value has only zeros after the decimal point, its pre-decimal point value is returned.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F328_DINT **(see page 1324)**

Data types

Variable	Data type	Function
s	REAL	source REAL number data (2 words)
d	DINT, DWORD	destination for storing converted data

Operands

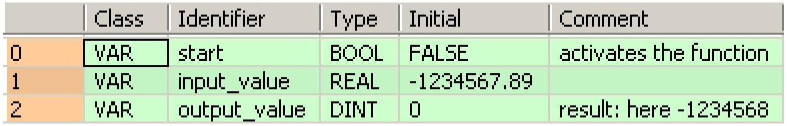
For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the value at input **s** is not a REAL number, or the converted result exceeds the 32-bit area of output **d**.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	for an instant	▪ the result is 0.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

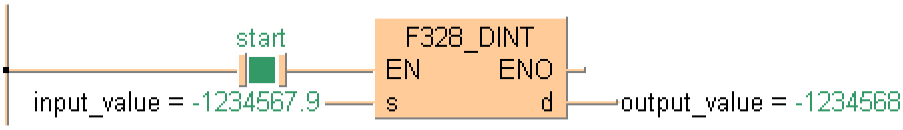


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	-1234567.89	
2	VAR	output_value	DINT	0	result: here -1234568

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out. It converts the floating point value -1234567.89 into the whole number value -1234568, which is transferred to the variable **output_value** at the output. Since the whole number may not exceed the floating point value, the function rounds down here.

LD



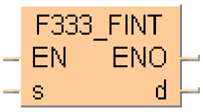
ST When programming with structured text, enter the following:

```
IF start THEN
    F328_DINT(input_value, output_value);
END_IF;
```

F333_FINT

Rounding the first decimal point down

Description The function rounds down the decimal part of the real number data and returns it at output **d**.



The converted whole-number value at output **d** is always less than or equal to the floating-point value at input **s**:

- If a positive floating-point value is at the input, a positive pre-decimal point value is returned at the output.
- If a negative floating-point value is at the input, the next smallest pre-decimal point value is returned at the output.
- If the negative floating-point value has only zeros after the decimal point, its pre-decimal point position is returned.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F333_FINT **(see page 1324)**

Data types

Variable	Data type	Function
s	REAL	source
d	REAL	destination

Operands

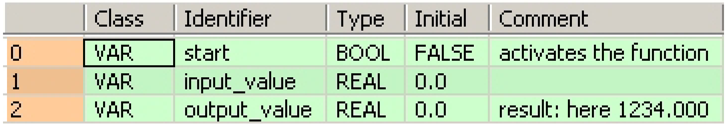
For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the value at input **s** is not a **REAL** number.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	▪ the result is 0.
R9009	%MX0.900.9	for an instant	▪ the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

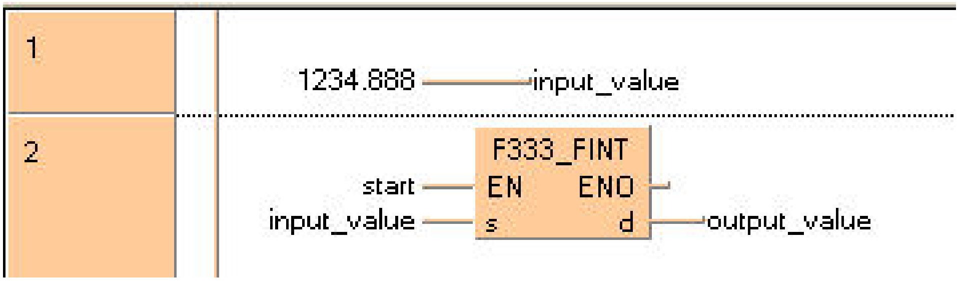


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	0.0	
2	VAR	output_value	REAL	0.0	result: here 1234.000

In this example, the input variable input_value is declared. However, you can write a constant directly at the input contact of the function instead.

Body The value 1234.888 is assigned to the variable **input_value**. When the variable **start** is set to TRUE, the function is carried out. It rounds down the **input_value** after the decimal point and returns the result (here: 1234.000) at the variable **output_value**.

LD



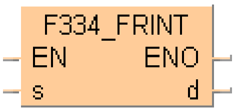
ST When programming with structured text, enter the following:

```
input_value:=1234.888;
IF start THEN
    F333_FINT(input_value, output_value);
END_IF;
```

F334_FRINT

Rounding the first decimal point off

Description The function rounds off the decimal part of the real number data and returns it at output **d**.



If the first post-decimal digit is between 0..4, the pre-decimal value is rounded down. If the first post-decimal digit is between 5..9, the pre-decimal value is rounded up.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F334_FRINT **(see page 1324)**

Data types

Variable	Data type	Function
s	REAL	source
d	REAL	destination

Operands

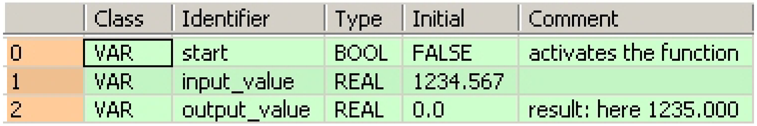
For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the value at input **s** is not a REAL number.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	▪ the result is 0.
R9009	%MX0.900.9	for an instant	▪ the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

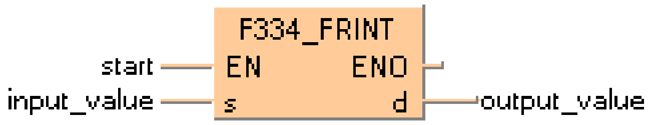


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	1234.567	
2	VAR	output_value	REAL	0.0	result: here 1235.000

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out. It rounds off the **input_value** = 1234.567 after the decimal point and returns the result (here: 1235.000) at the variable **output_value**.

LD



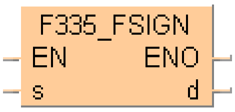
ST When programming with structured text, enter the following:

```
IF start THEN
    F334_FRINT(input_value, output_value);
END_IF;
```

F335_FSIGN

Floating point data sign changes (negative/positive conversion)

Description The function changes the sign of the floating point value at input **s** and returns the result at output **d**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F335_FSIGN **(see page 1324)**

Data types

Variable	Data type	Function
s	REAL	source
d	REAL	destination

Operands

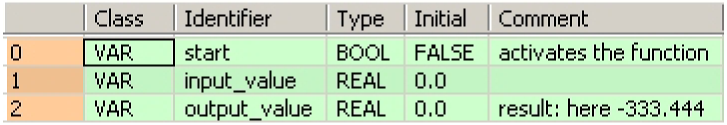
For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the value at input s is not a REAL number.
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	▪ the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

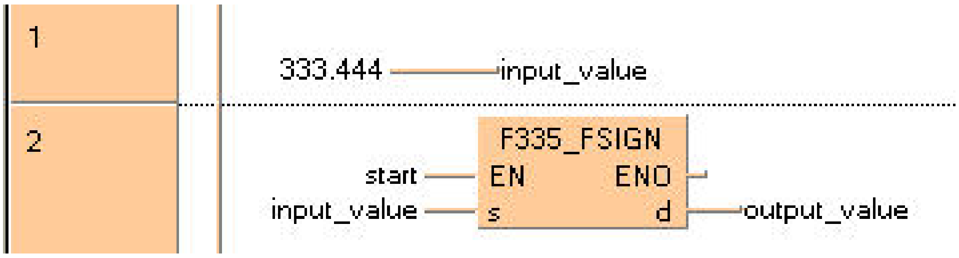


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	0.0	
2	VAR	output_value	REAL	0.0	result: here -333.444

In this example, the input variable input_value is declared. However, you can write a constant directly at the input contact of the function instead.

Body The value 333.4 is assigned to the variable **input_value**. When the variable **start** is set to TRUE, the function is carried out. The **output_value** is then -333.4.

LD



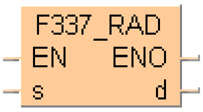
ST When programming with structured text, enter the following:

```
input_value:=333.444;
IF start THEN
    F335_FSIGN(input_value, output_value);
END_IF;
```

F337_RAD

Conversion of angle units (Degrees -> Radians)

Description The function converts the value of an angle entered at input **s** from degrees to radians and returns the result at output **d**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F337_RAD **(see page 1324)**

Data types

Variable	Data type	Function
s	REAL	source angle data (degrees), 2 words
d	REAL	destination for storing converted data

Operands

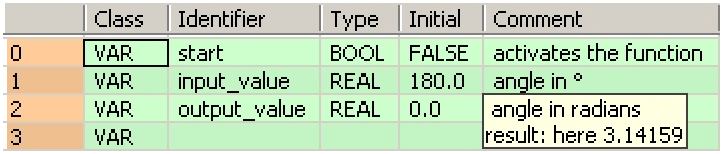
For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the value at input s is not a REAL number.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	▪ the result is 0.
R9009	%MX0.900.9	for an instant	▪ the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

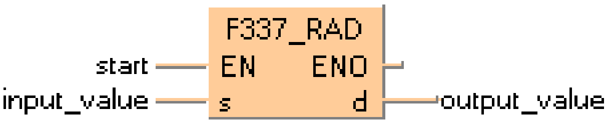


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	180.0	angle in °
2	VAR	output_value	REAL	0.0	angle in radians result: here 3.14159
3	VAR				

In this example, the input variable input_value is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD



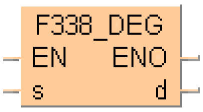
ST When programming with structured text, enter the following:

```
IF start THEN
    F337_RAD(input_value, output_value);
END_IF;
```

F338_DEG

Conversion of angle units (Radians -> Degrees)

Description The function converts the value of an angle entered at input **s** from radians to degrees and returns the result at output **d**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F338_DEG **(see page** 1324**)**

Data types

Variable	Data type	Function
s	REAL	source angle data (radians), 2 words
d	REAL	destination for storing converted data

Operands

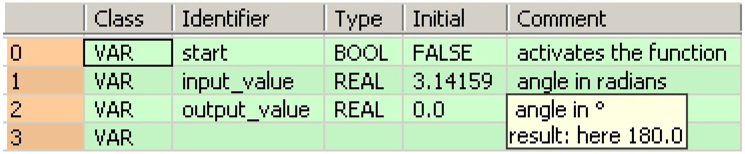
For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the value at input s is not a REAL number.
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	▪ the result is 0.
R9009	%MX0.900.9	for an instant	▪ the result causes an overflow.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

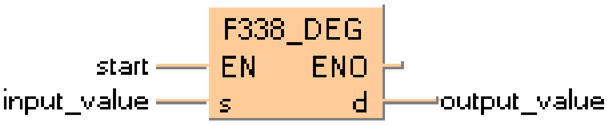


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	3.14159	angle in radians
2	VAR	output_value	REAL	0.0	angle in ° result: here 180.0
3	VAR				

In this example, the input variable input_value is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD



ST When programming with structured text, enter the following:

```
IF start THEN
    F338_DEG(input_value, output_value);
END_IF;
```

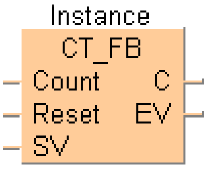
Chapter 21

Counter instructions

CT_FB

Down Counter

Description Counters realized with the CT_FB function block are down counters. The count area **SV** (set value) is 1 to 32767.



For the CT_FB function block declare the following:

Count **count contact**

each time a rising edge is detected at **Count**, the value 1 is subtracted from the elapsed value **EV** until the value 0 is reached

Reset **reset contact**

each time a rising edge is detected at **Reset**, the value 0 is assigned to **EV** and the signal output **C** is reset; each time a falling edge is detected at **Reset**, the value at **SV** is assigned to **EV**

SV **set value**

value of **EV** after a reset procedure

C **signal output**

is set when **EV** becomes 0

EV **elapsed value**

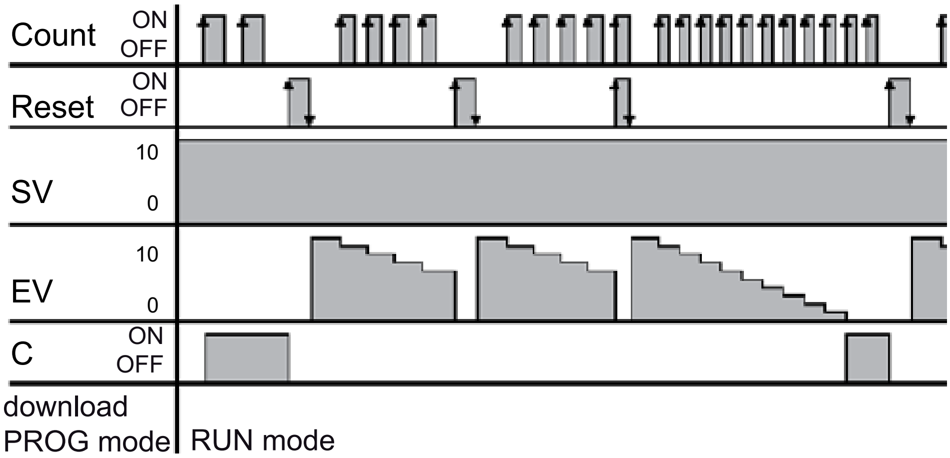
current counter value

PLC types **Availability of CT_FB (see page 1319)**

Data types

Variable	Data type	Function
Count	BOOL	count contact (down)
Reset		reset contact
SV	INT	set value
C	BOOL	set when **EV** = 0
EV	INT	elapsed value

Time chart

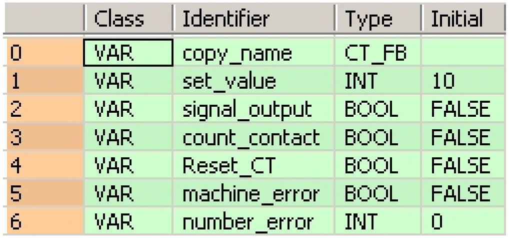


☞

- **In order to work correctly, the CT_FB function block needs to be reset each time before it is used.**
- **The number of available counters is limited and depends on the settings in the system registers 5 and 6. The compiler assigns a NUM* address to every counter instance. The addresses are assigned counting downwards, starting at the highest possible address.**
- **The basic CT (see page 704) function (down counter) uses the same NUM* address area (Num* input). In order to avoid errors (address conflicts), the CT function and the CT_FB function block should not be used together in a project.**

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

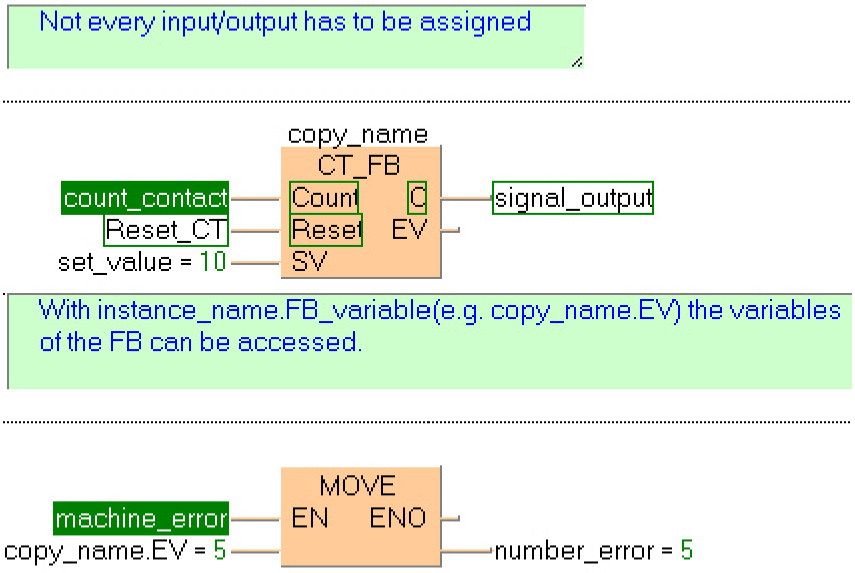
POU header All input and output variables which are used for programming the function block CT_FB are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **copy_name**, and a separate data area is reserved.



	Class	Identifier	Type	Initial
0	VAR	copy_name	CT_FB	
1	VAR	set_value	INT	10
2	VAR	signal_output	BOOL	FALSE
3	VAR	count_contact	BOOL	FALSE
4	VAR	Reset_CT	BOOL	FALSE
5	VAR	machine_error	BOOL	FALSE
6	VAR	number_error	INT	0

Body This example uses variables. You may also use constants for the input variables. Each rising edge detected at **count_contact** the value 1 is subtracted from the elapsed value **EV**. **Signal_output** is set to TRUE if the elapsed value **EV** becomes zero.

LD

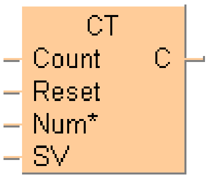


Not every input/output has to be assigned

With instance_name.FB_variable(e.g. copy_name.EV) the variables of the FB can be accessed.

CT Counter

Description Decrements a preset counter. The function has the following parameters: **Count**, **Reset**, **Num***, **SV**, and **C**. Their functions are listed in the Data types table below.



1. When the **Reset** input is on, the elapsed value is reset to 0.
2. When the **Reset** input changes from on to off, the set value (**SV**) is preset to the value assigned to it. The set value can be set to a decimal constant from 0 to 32767.
3. Each time the **Count** input changes from off to on, the value 1 is subtracted from the set value **SV**. When the elapsed value reaches 0, the output **C** turns on.
4. If the **Count** input and **Reset** input both turn on at the same time, the **Reset** input is given priority.
5. If the **Count** input rises and the **Reset** input falls at the same time, the count input is ignored and preset is executed.

PLC types **Availability of** CT **(see page** 1319**)**

☞ **This function does not require a variable at the output C.**

Data types

Variable	Data type	Function
Count	BOOL	subtracts 1 from the set value each time it is activated
Reset	BOOL	▪ resets the elapsed value when it is ON ▪ presets the set value when changing from on to off
Num*	ANY16	▪ Must be a constant ▪ number assigned to the counter (see System Register 5)
SV	ANY16	set value is the number the counter starts subtracting from
C	BOOL	the counter turns on when it reaches the **SV**

Operands

For	Relay				T/C		Register			Constant
Count	X	Y	R	L	T	C	-	-	-	-
Reset	X	Y	R	L	T	C	-	-	-	-
Num*	-	-	-	-	-	-	-	-	-	ANY16
C	-	Y	R	L	-	-	-	-	-	-
SV	-	-	-	-	SV	-	-	-	-	-

■ Details about points of Down Counter CT:

Type	Number of points	Nos. that can be used
FP-e	44	100–143
FP0 C10, C14, C16	44	100–143
Non-hold type	40	100–139
Hold type	4	140–143
FP0 C32	44	100–143
Non-hold type	28	100–127
Hold type	16	128–143
FP2SH/FP10SH	72	3000–3071
FP3	56	200–255
FP2	24	1000–1023
FP-Sigma	24	1000–1023

The number of counter points can be changed using System Register 5. The number of points can be increased up to 3,072 with the FP2SH and FP10SH, up to 256 with the FP-C and FP3, up to 1,024 with the FP-Sigma and up to 1,024 with the FP2 and up to 144 with the FP0. Be aware that increasing the number of counter points decreases the number of timer points.

For all models except for the FP0 C10, C14, C16 and C32, there is a hold type, in which the counter status is retained even if the power supply is turned off, or if the mode is switched from RUN to PROG, and a non-hold type, in which the counter is reset under these conditions. System register 6 can be used to specify a non-hold type.

■ Set Value and Elapsed Value area

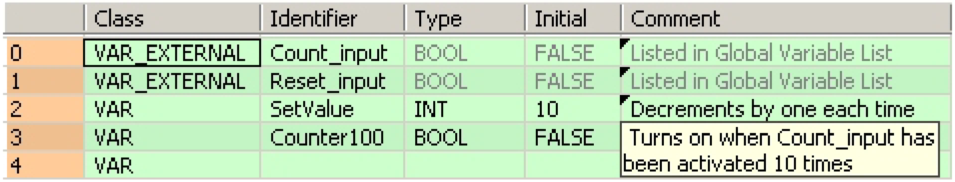
At the fall time when the reset input goes from on to off, the value of the set value area (**SV**) is preset in the elapsed value area (**EV**).

When the reset input is on, the elapsed value is reset to 0.

Each time the count input changes from off to on, the value 1 is subtracted from the set value and when the elapsed value reaches 0, the counter contact **Cn** (**n** is the counter number) turns on.

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

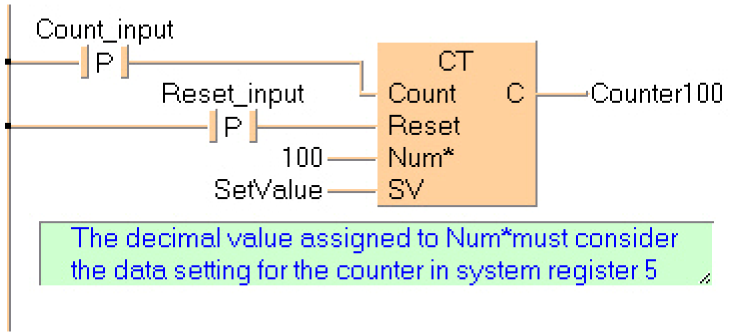
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	Count_input	BOOL	FALSE	Listed in Global Variable List
1	VAR_EXTERNAL	Reset_input	BOOL	FALSE	Listed in Global Variable List
2	VAR	SetValue	INT	10	Decrements by one each time
3	VAR	Counter100	BOOL	FALSE	Turns on when Count_input has been activated 10 times
4	VAR				

Body The set value SV is set to 10 when **Reset_input** is activated. Each time **Count_input** is activated, the value of **SV** decreases by 1. When this value reaches 0, **Counter100** turns on. **Num*** is assigned the counter number, which must be equal to or greater to the number assigned to Data in System Register 5.

LD



The decimal value assigned to Num*must consider the data setting for the counter in system register 5

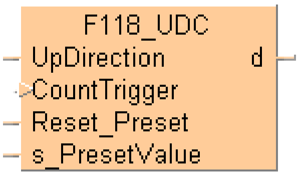
ST When programming with structured text, enter the following:

```
Counter100:=CT( Count:= Count_input ,
        Reset:= Reset_input ,
        Num:= 100 ,
        SV:= Setvalue );
            (* Num*, 100 in this example, must be a constant *)
```

F118_UDC

UP/DOWN counter

Description DOWN counting if the trigger **UpDirection** is in the OFF state. UP counting if the trigger **UpDirection** is in the ON state.



CountTrigger: Adds or subtracts one count at the rising edge of this trigger.

Reset_Preset: The condition is reset when this signal is on.

The area for the elapsed value **d** becomes 0 when the rising edge of the trigger is detected (OFF → ON). The value in **s_PresetValue** is transferred to **d** when the falling edge of the trigger is detected (ON → OFF).

s_PresetValue: Preset (Set) value or area for Preset (Set) value.

d: Area for count (elapsed) value.

PLC types **Availability of F118_UDC (see page 1320)**

Data types

Variable	Data type	Function
UpDirection	BOOL	sets counter to count up (ON) or down (0FF)
CountTrigger	BOOL	starts counter
Reset_Preset	BOOL	resets counter
s_PresetValue	ANY16	16-bit area or equivalent constant for counter preset value
d		16-bit area for counter elapsed value

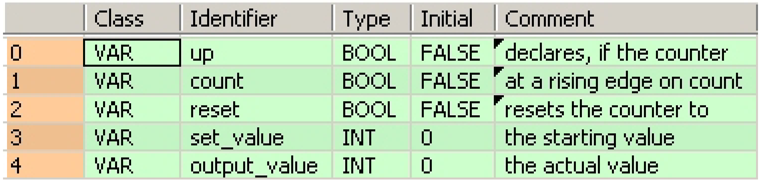
The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
UpDirection, CountTrigger, Reset_Preset	X	Y	R	L	T	C	-	-	-	-
s_PresetValue	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.

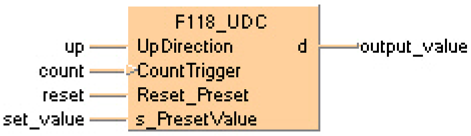


	Class	Identifier	Type	Initial	Comment
0	VAR	up	BOOL	FALSE	declares, if the counter
1	VAR	count	BOOL	FALSE	at a rising edge on count
2	VAR	reset	BOOL	FALSE	resets the counter to
3	VAR	set_value	INT	0	the starting value
4	VAR	output_value	INT	0	the actual value

Body

A rising edge at the input **Count_Trigger** activates the counter. The boolean variable at the input **UpDirection** sets the direction of the counter (TRUE = up, FALSE =down). TRUE at the input **Reset_Preset** resets the counter to the starting value.

LD



ST When programming with structured text, enter the following:

```
output_value:=F118_UDC( UpDirection:= up, Count_Trigger:= count,
Reset_Preset:= reset, s_PresetValue:= set_value);
(* output_value contains the count value *)
```

Chapter 22

Data transfer via communication ports

22.1 Description of the communication modes

Data transmission and reception can be carried out using the following communication modes. The communication mode is set in the system registers of the PLC.

MEWTOCOL-COM

This communication mode uses the proprietary MEWTOCOL-COM protocol to exchange data between a master and one or more slaves. This is called 1:1 or 1:N communication. A 1:N network is also known as a C-NET. There is a MEWTOCOL-COM master function and a MEWTOCOL-COM slave function. The side that issues commands is called master. The slave receives the commands, executes the process and sends back responses. The slave answers automatically to the commands received from the master, so no program is necessary on the slave. The MEWTOCOL-COM master function is not supported by all PLCs.

The master can be a PLC or any external device supporting the master function. To use the built-in master functionality of the PLC, select MEWTOCOL-COM Master/Slave in the system registers and implement a PLC program. The applicable instructions are F145_WRITE_DATA (see page 766) and F146_READ_DATA.

The slave can be a PLC or any external device which supports the MEWTOCOL-COM protocol. To use the built-in slave functionality of the PLC, select MEWTOCOL-COM Master / Slave in the system registers. For 1:N communication in a C-NET, the station number must be specified in the system registers of the slave.

Modbus RTU

This communication mode uses the Modbus RTU protocol to exchange data between a master and one or more slaves. This is called 1:1 or 1:N communication. There is a Modbus RTU master function and a Modbus RTU slave function. The side that issues commands is called master. The slave receives the commands, executes the process and sends back responses. The slave answers automatically to the commands received from the master, so no program is necessary on the slave.

The Modbus protocol supports both ASCII mode and RTU binary mode. However, the PLCs of the FP Series only support the RTU binary mode.

Program controlled mode

With program controlled communication, the user generates a program which governs the data transfer between a PLC and one or more external devices connected to the communication port. By this, any standard or user protocol can be programmed.

Typically, such a user program consists of sending and receiving the data. The data to be sent and the data received are stored in data register areas defined as send and receive buffers.

Sending can be controlled by the "transmission done" flag. For detailed information, see Flag Operation in Program Controlled Communication (see page 757).

For all PLC types	Sending includes generating the data for the send buffer and sending it using the instruction F159_MTRN (see page 741). For detailed information, see Sending Data to External Devices (see page 733). The "transmission done" flag can be evaluated using the IsTransmissionDone (see page 763) function. Or use the system variable sys_bIsComPort1TransmissionDone, sys_bIsComPort2TransmissionDone, or sys_bIsToolPortTransmissionDone, depending on the port.

Receiving includes processing the data in the receive buffer and preparing the system to receive further data. Reception can be controlled by the "reception done" flag or by directly evaluating the receive buffer. For detailed information, see Flag Operation in Program Controlled Communication (see page 757).

For all PLC types	The "reception done" flag can be evaluated using the IsReceptionDone (see page 760) function. Or use the system variable sys_bIsComPort1ReceptionDone, sys_bIsComPort2ReceptionDone, or sys_bIsToolPortReceptionDone, depending on the port. The end of reception can also be determined by time-out using the IsReceptionDoneByTimeOut (see page 761) function or by checking the contents of the receive buffer.

CPU communication ports	Data is automatically received in the receive buffer defined in the system registers. For detailed information, see Receiving Data from External Devices (see page 747).
MCU communication ports	Data is automatically received in the MCU unit. The data received can be moved to the CPU receive buffer using the instruction F161_MRCV (see page 755).

PLC link mode

PLC Link is an economic way of linking PLCs using a twisted-pair cable and the MEWNET protocol. Data is shared with all PLCs by means of dedicated internal relays called link relays (L) and data registers called link registers (LD). The statuses of the link relays and link registers of one PLC are automatically fed back to the other PLCs on the same network. The link relays and link registers of the PLCs contain areas for sending and areas for receiving data. Station numbers and link areas are allocated using the system registers.

For detailed information on setting the communication parameters and the link area, please refer to the hardware manuals of the corresponding units.

22.2 Setting the communication parameters

CPU: Setting the communication parameters for the COM ports

During PROG mode:	▪ via system registers (see page 714) ▪ via DIP switches (see page 715) (for FP10SH only)
During RUN time:	▪ F159_MTRN (switch communication mode (see page 717) with 16#8000) ▪ SYS1 (see page 980) with FP-Sigma and FP-X ▪ SYS2 (see page 993) with FP-Sigma and FP-X

MCU: Setting the communication parameters for the COM ports

During PROG mode:	▪ via MCU dialog ▪ via DIP switches (see page 718) (for FP2/2SH MCU only)
During RUN time:	▪ via F159_MWRT_PARA (see page 719) ▪ F159 (see page 717) (switch communication mode with 16#8000) ▪ Getting the COM Ports via the Input (X) Flags (see page 732) ▪ Setting the COM Ports via the Output (Y) Flags (see page 722)

Setting the CPU's Communication Parameters

22.2.1.1 Setting the CPU's COM Ports in PROG Mode via System Registers

For a general description on setting the system registers, please refer to the online help under setting the system registers.

◆ **Procedure**

1. **Double-click "PLC" in the navigator**
2. **Double-click "System Registers"**
3. **Double-click "COM Port"**

   The communication ports occupy different bit positions of the same system register, so individual settings for each communication port are possible.

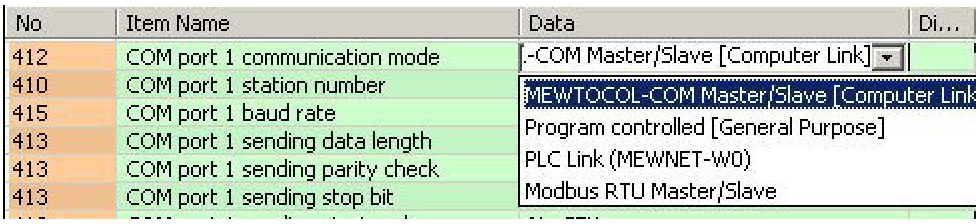
   To make settings for the TOOL port, select "TOOL Port" under "System Registers".

   The number of the system register for the respective settings may vary according to the PLC type used.

Make settings for the communication mode, communication format, baud rate, station number, and receive buffer if necessary.

Communication mode

Select a communication mode. The factory setting for the communication mode is "MEWTOCOL-COM Master/Slave".



No	Item Name	Data	Di...
412	COM port 1 communication mode	-COM Master/Slave [Computer Link]	
410	COM port 1 station number		
415	COM port 1 baud rate		
413	COM port 1 sending data length		
413	COM port 1 sending parity check		
413	COM port 1 sending stop bit		

MEWTOCOL-COM Master/Slave [Computer Link
Program controlled [General Purpose]
PLC Link (MEWNET-W0)
Modbus RTU Master/Slave

Station number

The station number must be set for MEWTOCOL-COM Master/Slave, Modbus RTU, and for PLC Link.

MEWTOCOL-COM and Modbus RTU: The station number can be set within a range of 1 to 99.

PLC Link: The station number can be set within a range of 1 to 16.

For detailed information on setting the station number with the station number setting switch, please refer to the hardware manuals of the corresponding units.

Baud rate

The setting must match the external device connected to the communication port.

Communication format setting

Default settings:

Data length:	8 bits
Parity:	Odd
Stop bit:	1 bit
Start code	No STX
End code:	CR

The setting must match the external device connected to the communication port.

MEWTOCOL-COM and Modbus RTU: The end code setting must always be "CR", and the start code setting must be "No STX".

PLC Link: The communication format settings are fixed.

For details on the format of the data in the send buffer and in the receive buffer, please see "Format of send and receive data" on page 745.

Receive buffer

For program controlled communication, a receive buffer must be specified in the system registers. Set a value for receive buffer starting address and receive buffer capacity.

22.2.1.2 Setting the CPU's COM Ports in PROG Mode via DIP Switches (FP10SH)

Procedure

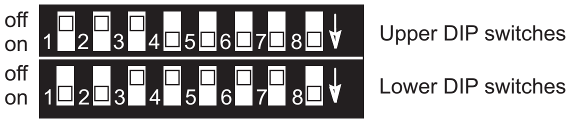
1. **Set the communication format**

   Default settings:

Data length:	8 bits
Parity:	Odd
Stop bit:	1 bit
Start code	No STX

End code: CR

The setting must match the external device connected to the communication port. Use the upper row of operation mode switches:



Operation mode switches (upper row)

Functions		Settings							
		SW1	SW2	SW3	SW4	SW5	SW6	SW7	SW8
MODEM control	Disabled	off							
	Enabled	on							
Start code	STX (16#02) invalid		off						
	STX (16#02) valid		on						
End code	None			off	off				
	CR (16#0D) and LF (16#0A)			on	off				
	CR (16#0D)			off	on				
	EXT (16#03)			on	on				
Stop bit	2 bits					off			
	1 bit					on			
Parity check	None						off	off	
	Even						on	off	
	Odd						on	on	
Data length (character bit)	7 bits								off
	8 bits								on

2. **Set the baud rate**

The default baud rate is 9600bit/s.

The setting must match the external device connected to the communication port. Use the lower row of operation mode switches:

Operation mode switches (lower row)

Functions	Settings							
Baud rate [bit/s]	SW1	SW2	SW3	SW4	SW5	SW6	SW7	SW8
115200	-	-	-	-	-	off	off	off
57600	-	-	-	-	-	on	off	off
38400	-	-	-	-	-	off	on	off
19200	-	-	-	-	-	on	on	off
9600	-	-	-	-	-	off	off	on
4800	-	-	-	-	-	on	off	on
2400	-	-	-	-	-	off	on	on
1200	-	-	-	-	-	on	on	on

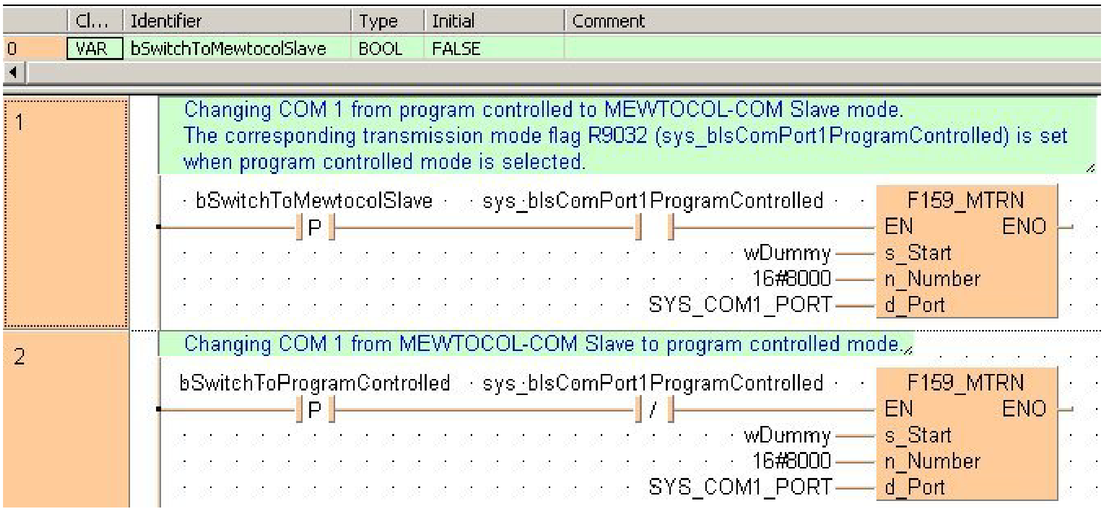
22.2.1.3 Setting in RUN Mode with SYS instructions (FP-Sigma, FP-X)

Please refer to the description of SYS1, communication condition setting (see page 980) and to the description of SYS2 (see page 993).

22.2.1.4 Changing the communication mode in RUN mode

The communication mode of the CPU's communication ports can be changed during RUN mode. You can toggle between program controlled mode and MEWTOCOL-COM mode by executing F159_MTRN and setting the variable **n_Number** (the number of bytes to be sent) to 16#8000.

POU Header and LD Body



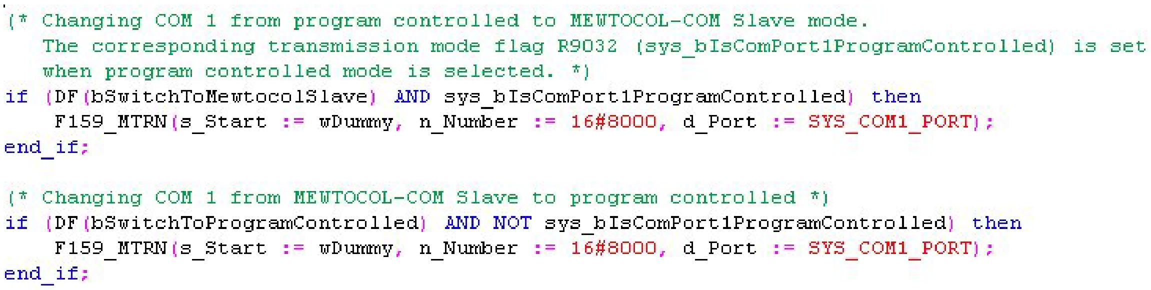
	Cl...	Identifier	Type	Initial	Comment
0	VAR	bSwitchToMewtocolSlave	BOOL	FALSE	

1
Changing COM 1 from program controlled to MEWTOCOL-COM Slave mode.
The corresponding transmission mode flag R9032 (sys_blsComPort1ProgramControlled) is set when program controlled mode is selected.

2
Changing COM 1 from MEWTOCOL-COM Slave to program controlled mode.

ST Body

The communication mode flag turns on when program controlled mode is active. It turns off when MEWTOCOL-COM mode is active. The flag can be evalutated using the system variable sys_blsComPort1ProgramControlled, sys_blsComPort2ProgramControlled, or sys_blsToolPortProgramControlled.



```
(* Changing COM 1 from program controlled to MEWTOCOL-COM Slave mode.
   The corresponding transmission mode flag R9032 (sys_bIsComPort1ProgramControlled) is set
   when program controlled mode is selected. *)
if (DF(bSwitchToMewtocolSlave) AND sys_bIsComPort1ProgramControlled) then
    F159_MTRN(s_Start := wDummy, n_Number := 16#8000, d_Port := SYS_COM1_PORT);
end_if;

(* Changing COM 1 from MEWTOCOL-COM Slave to program controlled *)
if (DF(bSwitchToProgramControlled) AND NOT sys_bIsComPort1ProgramControlled) then
    F159_MTRN(s_Start := wDummy, n_Number := 16#8000, d_Port := SYS_COM1_PORT);
end_if;
```

◆NOTE

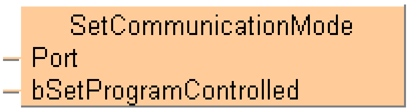
When the power is turned on, the communication mode selected in the system registers is set.

It is not possible to change to the Modbus RTU mode using F159_MTRN.

SetCommunication Mode

Switch communication mode between 'Program controlled' and 'MEWTOCOL-COM'

Description Sets the communication mode to the mode indicated by the value applied at **bSetProgramControlled**. If this value is TRUE then the communication mode is set to Program controlled mode (see page 712), if it is FALSE it is set to "MEWTOCOL-COM (see page 712) Slave [Computer Link]".



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

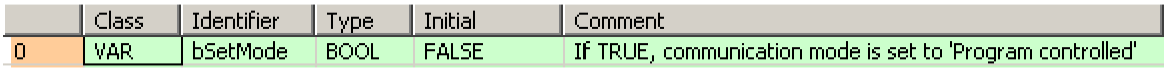
PLC types see see page 1330

Data types

Variable	Data type	Function
Port	INT	Specifies the CPU or MCU port number
bSetProgramControlled		Sets the communication mode: ▪ TRUE: Program controlled mode ▪ FALSE: MEWTOCOL-COM Slave [Computer Link] mode

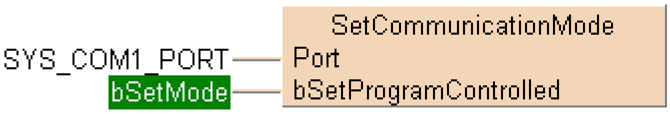
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	bSetMode	BOOL	FALSE	If TRUE, communication mode is set to 'Program controlled'

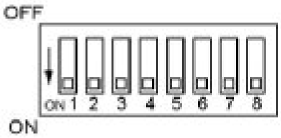
LD



ST
```
SetCommunicationMode(Port := SYS_COM1_PORT,
                     bSetProgramControlled := bSetMode);
```

Setting the MCU's COM Ports in PROG Mode via DIP Switches (FP2/2SH)

Use the DIP switches that are located at the back of the unit to set the operation mode and communication speed.



DIP switch settings

	Port	COM 1				COM 2			
	Switch No.	SW1	SW2	SW3	SW4	SW5	SW6	SW7	SW8
Operation mode	Spare	OFF	OFF	-	-	OFF	OFF	-	-
	PLC link	ON	OFF	-	-	ON	OFF	-	-
	Program controlled communication	OFF	ON	-	-	OFF	ON	-	-
	MEWTOCOL-COM Slave	ON	ON	-	-	ON	ON	-	-
Baud rate	115200bit/s	-	-	OFF	OFF	-	-	OFF	OFF
	19200bit/s	-	-	ON	OFF	-	-	ON	OFF
	9600bit/s	-	-	OFF	ON	-	-	OFF	ON
	Memory switch	-	-	ON	ON	-	-	ON	ON

☞ ◆**NOTE**

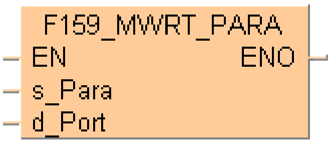
The factory setting for all DIP switches is ON.

Setting the MCU's COM Ports in PROG Mode via the MCU Dialog

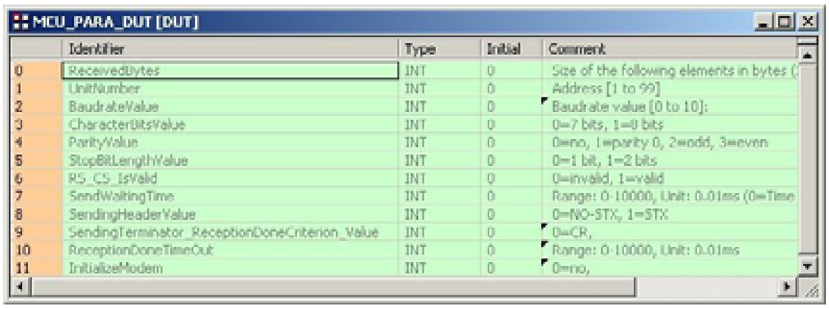
Please refer to the description of the MCU parameter settings in the online help.

Setting the MCU's communication ports in RUN Mode with F159_MWRT_PARA

Communication parameters in the predefined DUT MCU_PARA_DUT are written to the specified port of a Multi-Communication Unit.

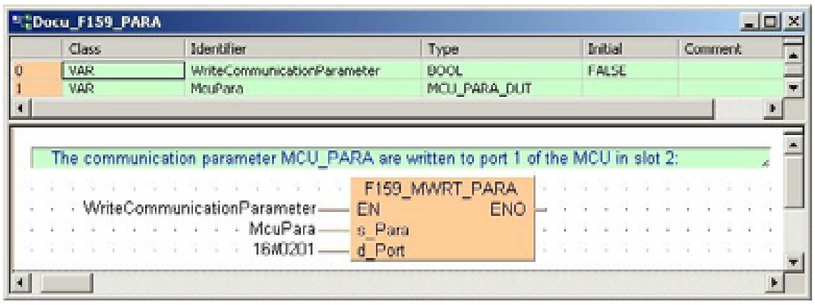


DUT settings


MCU_PARA_DUT [DUT]

	Identifier	Type	Initial	Comment
0	ReceivedBytes	INT	0	Size of the following elements in bytes (
1	UnitNumber	INT	0	Address [1 to 99]
2	BaudrateValue	INT	0	Baudrate value [0 to 10]:
3	CharacterBitsValue	INT	0	0=7 bits, 1=8 bits
4	ParityValue	INT	0	0=no, 1=parity 0, 2=odd, 3=even
5	StopBitLengthValue	INT	0	0=1 bit, 1=2 bits
6	RS_CS_IsValid	INT	0	0=invalid, 1=valid
7	SendWaitingTime	INT	0	Range: 0-10000, Unit: 0.01ms (0=Time
8	SendingHeaderValue	INT	0	0=NO-STX, 1=STX
9	SendingTerminator_ReceptionDoneCriterion_Value	INT	0	0=CR,
10	ReceptionDoneTimeOut	INT	0	Range: 0-10000, Unit: 0.01ms
11	InitializeModem	INT	0	0=no,

Example


Docu_F159_PARA

	Class	Identifier	Type	Initial	Comment
0	VAR	WriteCommunicationParameter	BOOL	FALSE	
1	VAR	McuPara	MCU_PARA_DUT		

The communication parameter MCU_PARA are written to port 1 of the MCU in slot 2:

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Configuration of communication parameters:

1. **UnitNumber** (station 1 to 99)
2. **BaudrateValue** (0 to 10) *2

*2. Baud rate setting value

Storage value	**Baud rate**
0	300
1	600
2	1200
3	2400
4	4800
5	9600
6	19200
7	38400
8	57600
9	115K
10	230K

3. **CharacterBitsValue** (0=7 bits, 1=8 bits)
4. **ParityValue** (0=no parity, 1=parity 0, 2=odd, 3=even)
5. **StopBitLengthValue** (0=1 bit, 1=2 bits)
6. **RS_CS_IsValid** (0=disable, 1=enable)
7. **SendWaitingTime** (0=time for about three characters/effective time=n*0.01ms (0 to 100ms))
8. **SendingHeaderValue** (0=No STX, 1=STX)
9. **SendingTerminator_ReceptionDoneCriterion_Value** (0=CR, 1=CR+LF, 2=No SendingTerminator, ReceptionDone by Timeout (24 bits), 3 =EXT)
10. **ReceptionDoneTimeOut** (0=immediate/effective time=n*0.01 ms (0 to 100 ms)
11. **InitModemWhenPowerTurnsOn** (0=not initialized, 1=initialized)

Data types

Variable	Data type	Function
s_Para	MCU_PARA_DUT	Communication parameters defined in the predefined DUT
d_Port	ANY16	Specification of slot number (high byte) and port number (low byte) of the MCU to which the data is transmitted. 16#xx01: COM1 on MCU in slot 16#xx 16#xx02: COM2 on MCU in slot 16#xx

Operands

For	Relay				T/C		Register			Constant
s_Para	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
d_Port	-	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	the MCU unit does not exist in the specified slot
R9008	%MX0.900.8	for an instant	

Setting the MCU's communication ports during RUN mode via the output (Y) flags

16 I/Os for Y are allocated. I/O numbers are determined depending on the installation location and the I/O allocations of the other units.

Output signal		**Name**	**Description (0: OFF, 1: ON)**	**Effective operation mode**
COM 1	**COM 2**			
Y10-Y17	Y10-Y17	Undefined	Default setting: 0 (Do not change)	None
Y18	Y19	RTS signal output	The transmission from the devices communicating with the MCU can be controlled by turning this output on. Permit transmission from communicating devices: 0 Prohibit the transmission from communicating devices: 1 The CTS signal sent from the communicating devices can be monitored via X8 and X9.	Effective only when setting the RS/CS to be valid and using the RS232C communication cassette.
Y1A-Y1D	Y1A-Y1D	Undefined	Default setting: 0 (Do not change)	None
Y1E	Y1F	Request to reset CH	Communication channels can be reset by turning on Y1E or Y1F. No request to reset: 0 Request to reset: 1 After 1 is output and the completion of the reset is confirmed by XE/XF, return to 0. The reset is performed only once when this signal rises. During reset, the following operations are performed: 1: Transmission discontinued 2: Reception discontinued 3: Receive buffer cleared 4: Communication parameters reset 5: Error information cleared (for errors which can be cleared) This function can be used to delete unnecessarily received data or to clear errors before starting normal reception.	Program controlled communication

◆NOTE

The channel reset can be automatically performed by one of the following (in these cases, the reset done signal by XE/YF does not turn on):

- **Setting/changing communication parameters using the instruction F159_MWRT_PARA (see page 719).**

- **Changing operation modes (see page 717) (switching between program controlled communication and MEWTOCOL-COM Slave) using F159_MTRN.**
- **Turning on the PLC power supply or changing from PROG to RUN mode if the MCU settings have been made via software.**

22.3 Getting the communication mode

You can check during RUN mode which communication mode has been set on the PLC. The following communication modes can be determined: PLC Link (see page 724), program controlled communication (see page 725), and MEWTOCOL-COM Master / Slave (see page 726).

There are three different ways to get the communication mode:

1. Using PLC-independent system variables. There are different system variables for each port. For detailed information on using system variables, please refer to Data transfer to and from special data registers (see page 859).
2. Using PLC-independent functions: The port number must be specified in a variable. The functions IsPlcLink (see page 724) and IsProgramControlled (see page 726) are available.
3. Using special relays: The relay numbers vary depending on the COM port and the PLC type!

For details on getting the communication mode of an MCU, please refer to Getting the MCU communication parameters (see page 729).

22.3.1 Checking for PLC link mode

PLC Link mode can be checked for the following devices:

FPΣ, FP-X:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
COM1	1	R9041	IsPlcLink (see page 724)	sys_bIsComPort1PlcLink	TRUE

MCU:

Port name	Port number	Function name	DUT
COM1	16#xx01	IsPlcLink (see page 724)	MCU-STATUS_DUT.CommunicationMode=2 (see page 729)
COM2	16#xx02		

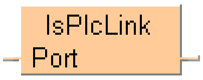
xx = slot number (hexadecimal)

IsPlcLink

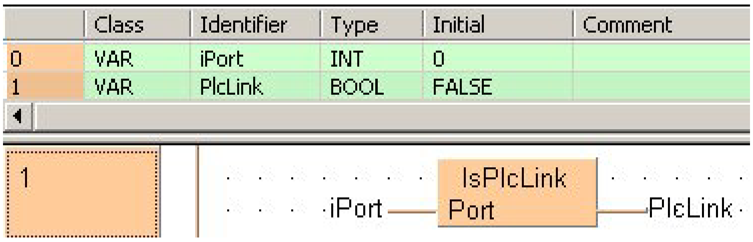
Evaluation of "PLC Link" flag for all ports

Description This instruction returns the value of the "PLC Link" flag. The "PLC Link" flag is TRUE if the communication port of the PLC has been set to PLC Link communication mode.

Symbol:



Example



	Class	Identifier	Type	Initial	Comment
0	VAR	iPort	INT	0	
1	VAR	PlcLink	BOOL	FALSE	

22.3.2 Checking for program controlled mode

Program controlled mode can be checked for the following devices:

FP0:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
COM	1	R9032	IsProgramControlled (see page 726)	sys_bIsComPort1ProgramControlled	TRUE

FPΣ, FP-X:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
TOOL	0	R9040	IsProgramControlled (see page 726)	sys_bIsToolPortProgramControlled	FALSE
COM1	1	R9032		sys_bIsComPort1ProgramControlled	
COM2	2	R9042		sys_bIsComPort2ProgramControlled	

FP2/FP2SH/FP10SH:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
COM	0	R9032	IsProgramControlled (see page 726)	sys_bIsComPort1ProgramControlled	TRUE

MCU:

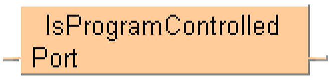
Port name	Port number	Function name	DUT
COM1	16#xx01	IsProgramControlled (see page 726)	MCU-STATUS_DUT.CommunicationMode =1 (see page 729)
COM2	16#xx02		

xx = slot number (hexadecimal)

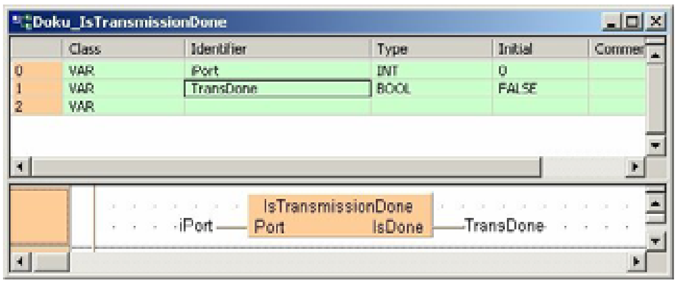
IsProgramControlled **Evaluates the "program controlled" flag**

Description This instruction returns the value of the "program controlled" flag. The "program controlled" flag is TRUE if the communication port of the PLC has been set to program controlled communication mode. It is FALSE if it has been set to "MEWTOCOL-COM Master / Slave".

Symbol:



Example



Doku_IsTransmissionDone

	Class	Identifier	Type	Initial	Commer
0	VAR	iPort	INT	0	
1	VAR	TransDone	BOOL	FALSE	
2	VAR				

22.3.3 Checking for MEWTOCOL-COM master / slave mode

MEWTOCOL-COM Master / Slave mode can be checked for the following devices:

FP0:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
COM	1	R9032	IsProgramControlled (see page 726)	sys_bIsComPort1ProgramControlled	FALSE

FPΣ, FP-X:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
TOOL	0	R9040	IsProgramControlled (see page 726)	sys_bIsToolPortProgramControlled	FALSE
COM1	1	R9032		sys_bIsComPort1ProgramControlled	
COM2	2	R9042		sys_bIsComPort2ProgramControlled	

FP2/FP2SH/FP10SH:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
COM	0	R9032	IsProgramControlled (see page 726)	sys_bIsComPort1ProgramControlled	FALSE

MCU:

Port name	Port number	Function name	DUT
COM1	16#xx01	IsProgramControlled (see page 726)	MCU-STATUS_DUT.CommunicationMode =0 (see page 729)
COM2	16#xx02		

xx = slot number (hexadecimal)

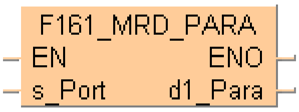
Getting the MCU's Communication Parameters

In this section:

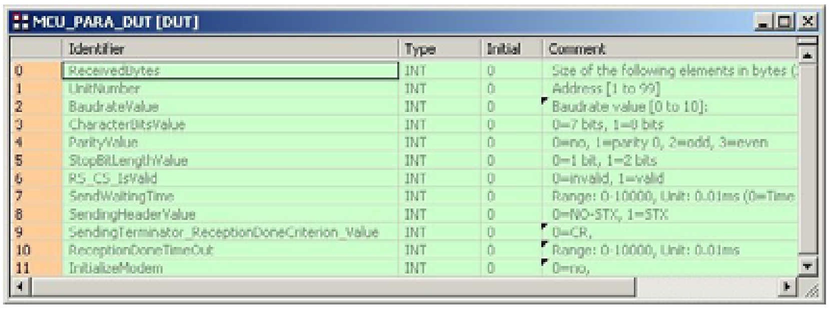
F161_MRD_PARA

Getting the communication modes in RUN mode from MCU's COM port

Description Communication parameters in the predefined DUT MCU_PARA_DUT are received from a port of a Multi-Communication Unit in a certain slot.



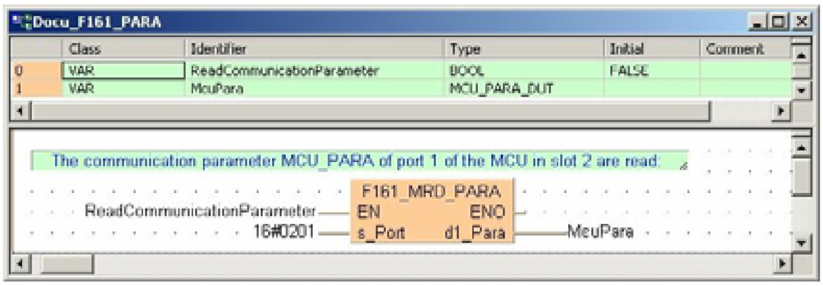
DUT settings



MCU_PARA_DUT [DUT]

	Identifier	Type	Initial	Comment
0	ReceivedBytes	INT	0	Size of the following elements in bytes (
1	UnitNumber	INT	0	Address [1 to 99]
2	BaudrateValue	INT	0	Baudrate value [0 to 10]:
3	CharacterBitsValue	INT	0	0=7 bits, 1=8 bits
4	ParityValue	INT	0	0=no, 1=parity 0, 2=odd, 3=even
5	StopBitLengthValue	INT	0	0=1 bit, 1=2 bits
6	RS_CS_IsValid	INT	0	0=invalid, 1=valid
7	SendWaitingTime	INT	0	Range: 0-10000, Unit: 0.01ms (0=Time
8	SendingHeaderValue	INT	0	0=NO-STX, 1=STX
9	SendingTerminator_ReceptionDoneCriterion_Value	INT	0	0=CR,
10	ReceptionDoneTimeOut	INT	0	Range: 0-10000, Unit: 0.01ms
11	InitializeModem	INT	0	0=no,

Example



Docu_F161_PARA

	Class	Identifier	Type	Initial	Comment
0	VAR	ReadCommunicationParameter	BOOL	FALSE	
1	VAR	MicuPara	MCU_PARA_DUT		

The communication parameter MCU_PARA of port 1 of the MCU in slot 2 are read:

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Configuration of communication parameters:

1. **UnitNumber** (station 1 to 99)
2. **BaudrateValue** (0 to 10) *2

*2. Baud rate setting value

Storage value	Baud rate
0	300
1	600
2	1200
3	2400
4	4800
5	9600
6	19200
7	38400
8	57600
9	115K
10	230K

3. **CharacterBitsValue** (0=7 bits, 1=8 bits)
4. **ParityValue** (0=no parity, 1=parity 0, 2=odd, 3=even)
5. **StopBitLengthValue** (0=1 bit, 1=2 bits)
6. **RS_CS_IsValid** (0=disable, 1=enable)
7. **SendWaitingTime** (0=time for about three characters/effective time=n*0.01ms (0 to 100ms))
8. **SendingHeaderValue** (0=No STX, 1=STX)
9. **SendingTerminator_ReceptionDoneCriterion_Value** (0=CR, 1=CR+LF, 2=No SendingTerminator, ReceptionDone by Timeout (24 bits), 3 =EXT)
10. **ReceptionDoneTimeOut** (0=immediate/effective time=n*0.01 ms (0 to 100 ms)
11. **InitModemWhenPowerTurnsOn** (0=not initialized, 1=initialized)

PLC types **Availability of** F161_MRD_PARA **(see page 1321)**

Data types

Variable	Data type	Function
s_Port	ANY16	Specification of slot number (high byte) and port number (low byte) of the MCU to which the data is transmitted. 16#xx01: COM1 on MCU in slot 16#xx 16#xx02: COM2 on MCU in slot 16#xx
d1_Para	MCU_PARA_DUT	Communication parameters defined in the predefined DUT

Operands

For	Relay				T/C		Register			Constant
s_Port	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1_Para	-	WY	WR	WL	SV	EV	DT	LD	FL	-

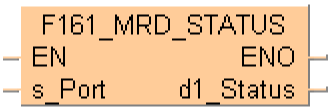
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the specified address using the index modifier exceeds a limit ▪ the MCU unit does not exist in the specified slot ▪ the specified communication port does not exist
R9008	%MX0.900.8	for an instant	

F161_MRD_STATUS

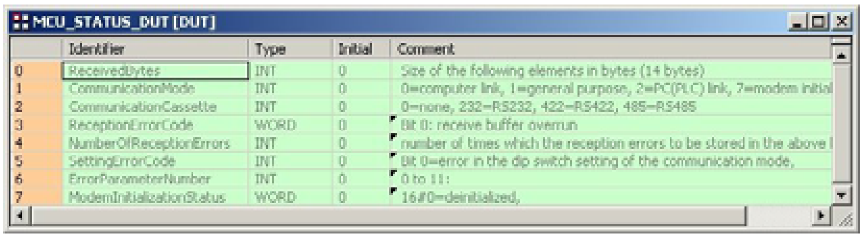
Getting the statuses in RUN mode from MCU's COM port

Description Status data is read from the specified COM port of a Multi-Communication Unit.



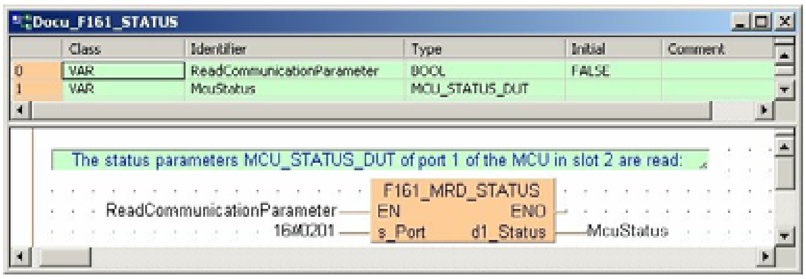
The DUT MCU_STATUS_DUT is predefined in the FP Library.

DUT settings


MCU_STATUS_DUT [DUT]

	Identifier	Type	Initial	Comment
0	ReceivedBytes	INT	0	Size of the following elements in bytes (14 bytes)
1	CommunicationMode	INT	0	0=computer link, 1=general purpose, 2=PC(PLC) link, 7=modem initial
2	CommunicationCassette	INT	0	0=none, 232=RS232, 422=RS422, 485=RS485
3	ReceptionErrorCode	WORD	0	Bit 0: receive buffer overrun
4	NumberOfReceptionErrors	INT	0	number of times which the reception errors to be stored in the above l
5	SettingErrorCode	INT	0	Bit 0=error in the dip switch setting of the communication mode,
6	ErrorParameterNumber	INT	0	0 to 11:
7	ModemInitializationStatus	WORD	0	16#0=deinitialized,

Example


Docu_F161_STATUS

	Class	Identifier	Type	Initial	Comment
0	VAR	ReadCommunicationParameter	BOOL	FALSE	
1	VAR	McuStatus	MCU_STATUS_DUT		

The status parameters MCU_STATUS_DUT of port 1 of the MCU in slot 2 are read:

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Configuration of monitor data

1. **CommunicationMode** (0 to 7)

   (0=MEWTOCOL-COM Slave, 1=Program controlled serial communication, 2=PLC Link, 7=modem initialization)

2. **CommunicationCassette**

   (0=no communication cassette, 232=RS232C, 422=RS422, 485=RS485)

3. **ReceptionErrorCode**

   (Bit 0=receive buffer overrun (hardware), bit 1=stop bit not detected, bit 2=parity unmatched)

   (Bit 8=receive buffer overflow, bit 9=receive buffer full)

4. **NumberReceptionErrors** (number of times the reception error stored in the lower byte of ReceptionErrorCode is detected)

5. **SettingErrorCode**

   (Bit0=error in the DIP switch setting of the operation mode, bit1=operation mode setting exceeds the usable limit of the unit)

   (Bit 8=error in the communication parameter setting, bit 9=error in the number of transmitted data)

6. **ErrorParameterNumber** (0 to 11)

7. **ModemInitializationStatus**

(16#0000=deinitialized, 16#0100=now initializing, 16#0200=initialization completed, 16#02FF=initialization failed.)

Data types

Variable	Data type	Function
s_Port	ANY16	Specification of slot number (high byte) and port number (low byte) of the MCU to which the data is transmitted. 16#xx01: COM1 on MCU in slot 16#xx 16#xx02: COM2 on MCU in slot 16#xx
d1_Status	MCU_STATUS_DUT	Communication parameters defined in the predefined DUT

Operands

For	Relay				T/C		Register			Constant
s_Port	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1_Status	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the specified address using the index modifier exceeds a limit ▪ the MCU unit does not exist in the specified slot ▪ the specified communication port does not exist
R9008	%MX0.900.8	for an instant	

22.3.3.1 Getting the communication modes and statuses via the input (X) flags from the MCU's COM ports in RUN mode

16 I/Os for X are allocated. I/O numbers are determined depending on the installation location and the I/O allocations of the other units.

Input signal		**Name**	**Description (0: OFF, 1: ON)**	**Effective operation mode**
COM 1	**COM 2**			
X0	X2	Reception done flag	When the MCU completes the data reception, it turns on. When waiting for the end of data reception: 0 When data reception completed: 1	Program controlled communication
X1	X3	PLC data reception done flag	When F161_MRCV (see page 755) is completed, it is set/reset. Reading completed: 1 Data not yet read: 0 Note 1)	
X4	X5	Transmission done flag	When transmission is available: 1 During transmission: 0 Note 2) When transmission is completed: 1	
X6	X7	Reception error	When F161_MRCV (see page 755) is completed, it is set/reset. Errors exist in the data read: 1 No error exists in the data read: 0 Note 1)	
X8	X9	CTS signal monitor	Status of the CTS signal sent from the device being communicated with. Transmission from MCU is possible: 0 Transmission from MCU is not possible: 1 The RTS signal from the MCU is controllable by Y18 and Y19.	Effective only when setting the RS/CS to be valid and using the communication block AFP2803
XA	XC	Latest reception error	Set when an error occurs during the reception of data by the MCU No reception error: 0 Reception error: 1 The details of the reception error can be confirmed by reading them to the PLC using the F161_MRD_STATUS (see page 729) instruction. Check X6/X7 to see whether or not there are errors in each receive buffer during multiple reception.	Program controlled communication (Reception)
XB	XD	Setting error	Operation mode switch setting error Usage restrictions for the unit	All operation modes
XE	XF	CH reset done	Communication channels can be reset by turning on Y1E or Y1F. The flag is reset upon completion. At completion: 1 When Y1E/Y1F is off: 0	Program controlled communication

22.4 Data transfer in program controlled mode

For all PLC types and all COM ports (including the COM ports of the Multi-Communication Unit) the following instructions are available:

- Tool instructions:
  - SendCharacters (see page 737)
  - SendCharactersAndClearString (see page 739)
  - ReceiveData (see page 750)
  - ReceiveCharacters (see page 752)
  - ClearReceiveBuffer (see page 753)

- FP instructions
  - F159_MTRN (see page 741), sending (see page 733) in program controlled mode
  - F161_MRCV (see page 755), receiving (see page 747) in program controlled mode

- Flag evaluation:
  - IsTransmissionDone (see page 763)
  - IsReceptionDone (see page 760)
  - IsReceptionDoneByTimeOut (see page 761)
  - IsCommunicationError (see page 764)

☞ ◆NOTE

F159_MTRN (see page 741) allows multiple communication ports to be accommodated. This instruction is an updated version of F144_TRNS. Both instructions are compatible with all PLCs: PLCs with multiple communication ports will compile F144_TRNS s, n to F159_MTRN s_Start, n_Number, d_Port*=1. PLCs with only one communication port will compile F159_MTRN to F144_TRNS s, n.

22.4.1 Sending data to external devices

Steps for sending data to external devices:

1. Setting the communication parameters to match the external device
2. Generating the data in the send buffer
3. Sending the data using the instruction F159_MTRN

☞ ◆NOTE

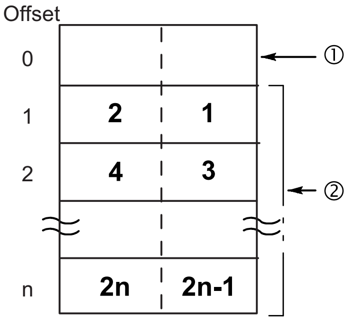
F159_MTRN (see page 741) allows multiple communication ports to be accommodated. This instruction is an updated version of F144_TRNS. Both instructions are compatible with all PLCs: PLCs with multiple communication ports will compile F144_TRNS s, n to F159_MTRN s_Start, n_Number, d_Port*=1. PLCs with only one communication port will compile F159_MTRN to F144_TRNS s, n.

1. Setting the communication parameters (see)

2. Generating the data in the send buffer

To generate the data in the send buffer, define a variable in the program and copy the data to the send buffer using a transfer instruction, e.g. F10_BKMV (see page 819).

The storage area for the data to be sent starts with the second word of the send buffer (offset 1). Offset 0 contains the number of bytes to be sent.



① Storage area for the number of bytes to be sent

② Storage area for the data to be sent

Bold numbers indicate the order of transmission.

The maximum volume of data that can be sent is 2048 bytes.

◆ **EXAMPLE**

Define a send buffer for 30 bytes (ARRAY [0...15] OF WORD) and copy 8 characters of a string ("ABCDEFGH") into the buffer.

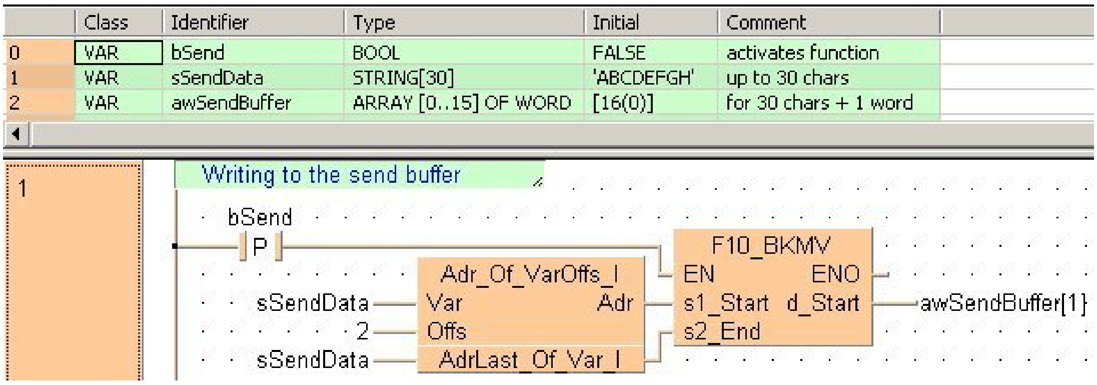
Send buffer layout:

Offset		
0	8	
1	16#42(B)	16#41(A)
2	16#44(D)	16#43(C)
3	16#46(F)	16#45(E)
4	16#48(H)	16#47(G)
15		

The first word of the send buffer (offset 0) is reserved for the number of bytes to be sent. Therefore, copy the data into offset 1 (**SendBuffer[1])**.

When sending begins (the execution condition for F159_MTRN (see page 741) turns to TRUE), the value in offset 0 is set to 8. At the end of transmission, the value in offset 0 is automatically reset to 0. The data in offset 1 to offset 4 is sent in order from the low order byte.

POU Header and LD Body



	Class	Identifier	Type	Initial	Comment
0	VAR	bSend	BOOL	FALSE	activates function
1	VAR	sSendData	STRING[30]	'ABCDEFGH'	up to 30 chars
2	VAR	awSendBuffer	ARRAY [0..15] OF WORD	[16(0)]	for 30 chars + 1 word

ST Body

```
if (DF(bSend)) then
    (* Creating the send buffer *)
    F10_BKMV(s1_Start := Adr_Of_VarOffs(Var := sSendData, Offs := 2),
    s2_End := AdrLast_Of_Var(sSendData), d_Start => awSendBuffer[1]);
```

When the variable **bSend** is set to TRUE, the function F10_BKMV copies the characters of the string **sSendData** to the buffer **awSendBuffer** beginning at **awSendBuffer[1]**.

The first two words of a string contain the string header information (maximum number of characters and the current number of characters). The string header must not be copied into the buffer. Therefore, enter an offset of 2 to the starting address of the string before copying the data.

Make sure that the send buffer is big enough for all the data to be sent. To determine its size you must take into account that two characters of the string **SendString** can be copied into each element of the array **SendBuffer**. **SendBuffer[0]** is reserved for the total number of bytes to be sent by F159_MTRN.

3. Sending the data using the instruction F159_MTRN

Execute F159_MTRN (see page 741) to

- specify the amount of data to be sent
- specify the communication port to be used
- output the data from the communication port to the external device.

When the execution condition of F159_MTRN turns to TRUE and the "transmission done" flag is TRUE, transmission starts. (For details on flag operation, see page 757.)

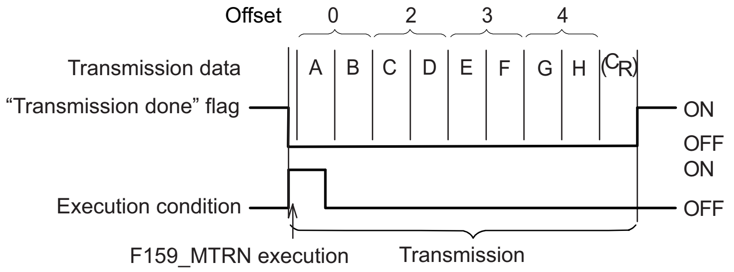
When sending data, operation is as follows:

- The number of bytes to be sent is set in offset 0 of the send buffer.
- The "transmission done" flag turns to FALSE.
- The data in the send buffer is sent starting with the low order byte in offset 1.
- The start and end codes specified in the system registers are automatically added to the data sent.
- During transmission, F159_MTRN cannot be executed again.
- The "reception done" flag turns to FALSE.
- The number of bytes received is set to 0 in offset 0 of the receive buffer.
- Data received is written into the receive buffer

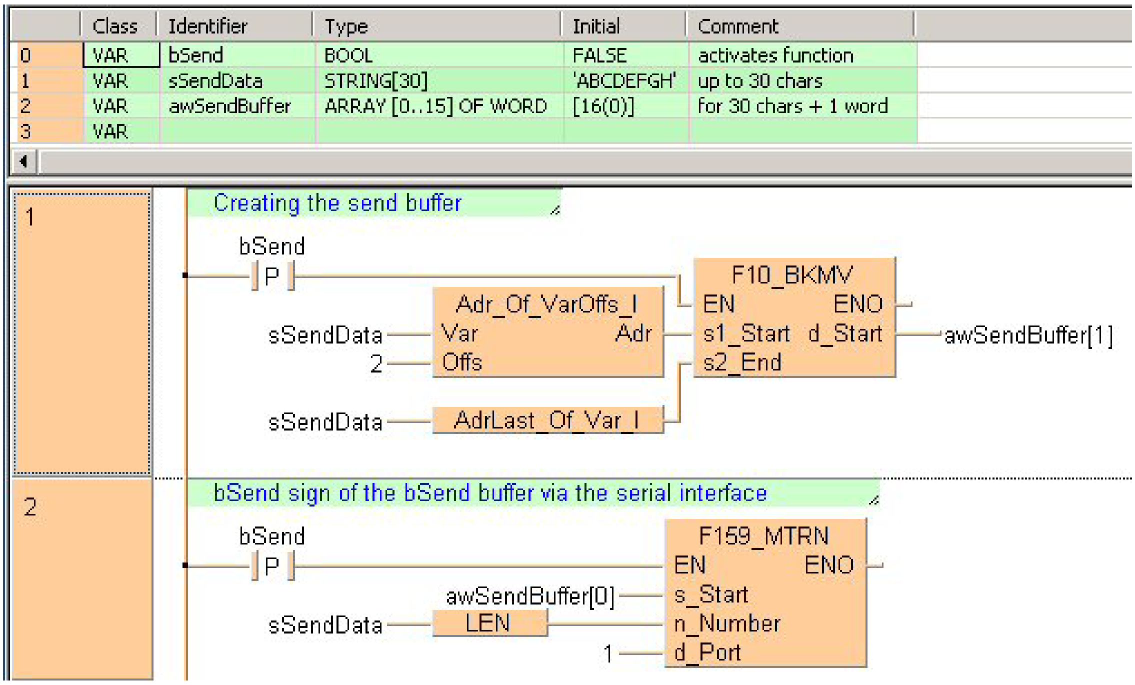
When the specified number of bytes has been sent, the "transmission done" flag turns to TRUE. The end code is automatically added to the data sent. At the end of transmission, the value in offset 0 is automatically reset to 0.

EXAMPLE

Transmit the characters "ABCDEFGH" to an external device connected to COM port 1. For start code and end code the default settings "No-STX" and "CR" are selected.



POU Header and LD Body



	Class	Identifier	Type	Initial	Comment
0	VAR	bSend	BOOL	FALSE	activates function
1	VAR	sSendData	STRING[30]	'ABCDEFGH'	up to 30 chars
2	VAR	awSendBuffer	ARRAY [0..15] OF WORD	[16(0)]	for 30 chars + 1 word
3	VAR				

ST Body

```
if (DF(bSend)) then
    (* Creating the send buffer *)
    F10_BKMV(s1_Start := Adr_Of_VarOffs(Var := sSendData, Offs := 2),
    s2_End := AdrLast_Of_Var(sSendData), d_Start => awSendBuffer[1]);
    (* Send contents of the send buffer via the serial interface *)
    F159_MTRN(s_Start := awSendBuffer[0], n_Number := LEN(sSendData), d_Port := 1);
end_if;
```

When the variable **bSend** is set to TRUE, the function F10_BKMV copies the characters of the string **sSendData** to the buffer **awSendBuffer** beginning at **awSendBuffer[1]**.

Then, F159_MTRN sends the data from the first element of the send buffer (**awSendBuffer[0])** as specified by **s_Start.** The length of the string to be sent (8 bytes) is set at **n_Number** (using the function LEN to calculate the number of bytes). The data is output from COM port 1 as specified by **d_Port.**

☞ ◆**NOTE**

For details on the operation of the "reception done" flag, the "transmission done" flag, and the communication error flag, see page 757.

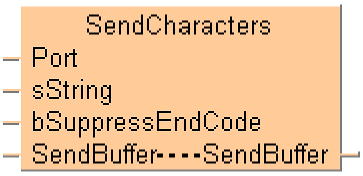
For details on the format of the data in the send buffer and in the receive buffer, please see "Format of send and receive data" on page 745.

Data cannot be sent unless the pin CS (Clear to Send) is on. When connecting to a three-wire port, short-circuit the RS and CS pins.

SendCharacters

Send characters to CPU or MCU port

Description This instruction first fills the send buffer applied at the VAR_INOUT variable **SendBuffer** with the relevant characters of the variable at **sString** according to the required data format for sending data "Sending data to external devices" on page 733. Then the send data instruction F159_MTRN (see page 741) is executed using the data of the send buffer. Setting the variable **bSuppressEndCode** to TRUE does not append the sending end code character even when specified in the according system register. In contrast to the instruction **SendCharactersAndClearString** (see page 739) the string variable applied at **sString** remains unchanged.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

See also:

- ReceiveCharacters (see page 752)
- ClearReceiveBuffer (see page 753)

PLC types see see page 1330

Data types

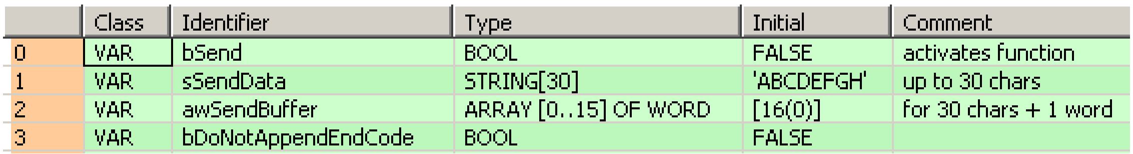
Variable	Data type	Function
Port	ANY16	▪ Communication port ▪ Must be a constant **FP-X, FPΣ and FP2, FP2SH (V1.4 or later):** ▪ PLC communication ports: Value: SYS_COM1_PORT or SYS_COM2_PORT or SYS_TOOL_PORT ▪ MCU communication port: Value: 16#xx01 (COM1), 16#xx02 (COM2) xx = slot number (hexadecimal) of the MCU (e.g. 16#0001: COM1 in slot 0, 16#0A02: COM2 in slot 10, 16#1401: COM1 in slot 20) **Other PLCs:** The command will be compiled to F144_TRNS, which works on the COM port of the CPU (the parameter **d_Port** will be ignored)
sString	STRING	Stores the send string
bSuppressEndCode	BOOL	When set to TRUE, the instruction does not append the sending end code character even if specified in the respective system register.
Input/output variable		
SendBuffer	ANY	Stores the send string temporarily

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the MCU unit does not exist in the specified slot ▪ 16#8000 is specified in MEWTOCOL-COM Master/Slave mode
R9008	%MX0.900.8	for an instant	

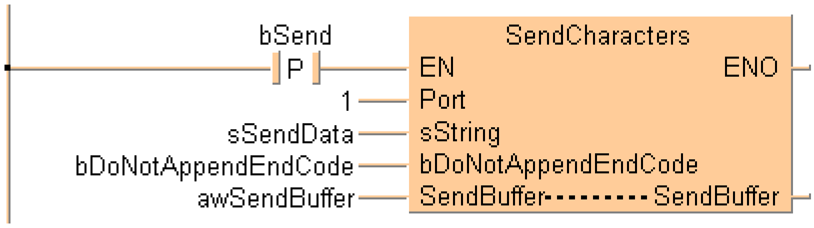
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	bSend	BOOL	FALSE	activates function
1	VAR	sSendData	STRING[30]	'ABCDEFGH'	up to 30 chars
2	VAR	awSendBuffer	ARRAY [0..15] OF WORD	[16(0)]	for 30 chars + 1 word
3	VAR	bDoNotAppendEndCode	BOOL	FALSE	

LD If **bSend** changes from FALSE to TRUE, the instruction sends the characters from **sSendData** to the MCU port 1. The characters are copied to the array **awSendBuffer**. **awSendBuffer[0]** is reserved for the length of the string to send.



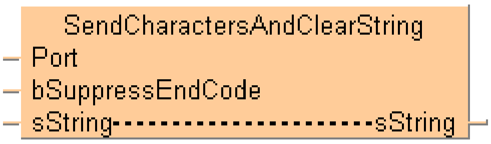
ST
```
if (DF(bSend)) then
    sResult:=SendCharacters(1, sSendData, awSendBuffer);
end_if;
```

SendCharactersAnd ClearString

Send characters and clesr string

Description This instruction directly executes the send data instruction F159_MTRN (see page 741) on the applied string without requiring an additional send buffer.

In contrast to the instruction SendCharacters (see page 737), the string variable applied at **sString** is cleared after execution.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

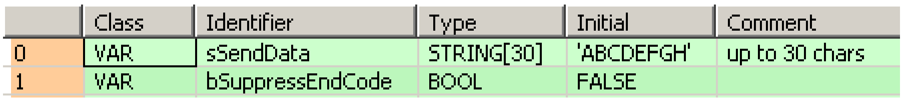
PLC types see see page 1330

Data types

Variable	Data type	Function
Port	INT	▪ Communication port ▪ Must be a constant **FP-X, FPΣ and FP2, FP2SH (V1.4 or later):** ▪ PLC communication ports: Value: SYS_COM1_PORT or SYS_COM2_PORT or SYS_TOOL_PORT ▪ MCU communication port: Value: 16#xx01 (COM1), 16#xx02 (COM2) xx = slot number (hexadecimal) of the MCU (e.g. 16#0001: COM1 in slot 0, 16#0A02: COM2 in slot 10, 16#1401: COM1 in slot 20) **Other PLCs:** The command will be compiled to F144_TRNS, which works on the COM port of the CPU (the parameter **d_Port** will be ignored)
bSuppressEndCode	BOOL	When set to TRUE, the instruction does not append the sending end code character even if specified in the respective system register.
VAR_INOUT		
sString	STRING	Stores the send string

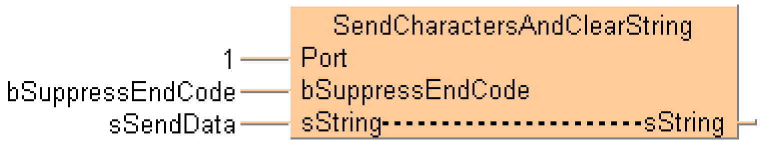
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	sSendData	STRING[30]	'ABCDEFGH'	up to 30 chars
1	VAR	bSuppressEndCode	BOOL	FALSE	

LD

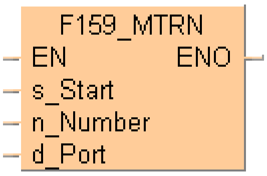


```
ST SendCharactersAndClearString(Port := 1,
                                bSuppressEndCode := bSuppressEndCode,
                                sString := sSendData);
```

F159_MTRN

Serial data communication to CPU or MCU port

Description This instruction is used to send data when an external device (computer, measuring instrument, bar code reader, etc.) has been connected to the specified RS232C port. If applied to the CPU's COM port, it also clears the receive buffer (see page 746), resets the "reception done flag" and allows further reception of data.



F159_MTRN is encapsulated in the following instructions:

- SendCharacters (see page 737)
- SendCharactersAndClearString (see page 739)
- ClearReceiveBuffer (see page 753)
- SetCommunicationMode (see page 717)

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

REFERENCE

- Data transfer in program controlled mode (see page 733)
- Changing the communication mode in RUN mode (see page 717)

PLC types

Availability of F159_MTRN **(see page 1321)**

Data types

Variable	Data type	Function
s_Start		Send buffer
n_Number		Bytes to send: ▪ Positive value: the end code is added in transmission. ▪ Negative value: the end code is not added in transmission. ▪ 16#8000: the communication mode of the specified communication port is changed.
d_Port		▪ Communication port ▪ Must be a constant **FP-X, FPΣ and FP2, FP2SH (V1.4 or later):** ▪ PLC communication ports: Value: SYS_COM1_PORT or SYS_COM2_PORT or SYS_TOOL_PORT ▪ MCU communication port: Value: 16#xx01 (COM1), 16#xx02 (COM2) xx = slot number (hexadecimal) of the MCU (e.g. 16#0001: COM1 in slot 0, 16#0A02: COM2 in slot 10, 16#1401: COM1 in slot 20) **Other PLCs:** The command will be compiled to F144_TRNS, which works on the COM port of the CPU (the parameter **d_Port** will be ignored)

Operands

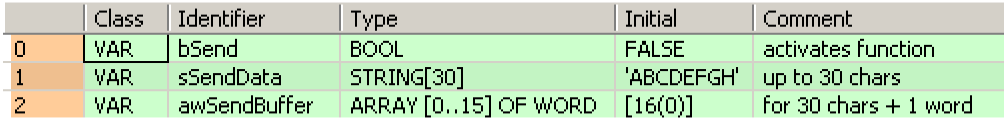
For	Relay				T/C		Register			Constant
s_Start	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
n_Number	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d_Port	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the specified address using the index modifier exceeds a limit ▪ the number of bytes to be sent specified by '**n_Number**' is outside of the specified area. **Flags only for the MCU:** ▪ the MCU unit does not exist in the specified slot ▪ 16#8000 is specified in MEWTOCOL-COM Master/Slave mode
R9008	%MX0.900.8	for an instant	

Example In this example the characters of the string **sSendData** are transmitted.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	bSend	BOOL	FALSE	activates function
1	VAR	sSendData	STRING[30]	'ABCDEFGH'	up to 30 chars
2	VAR	awSendBuffer	ARRAY [0..15] OF WORD	[16(0)]	for 30 chars + 1 word

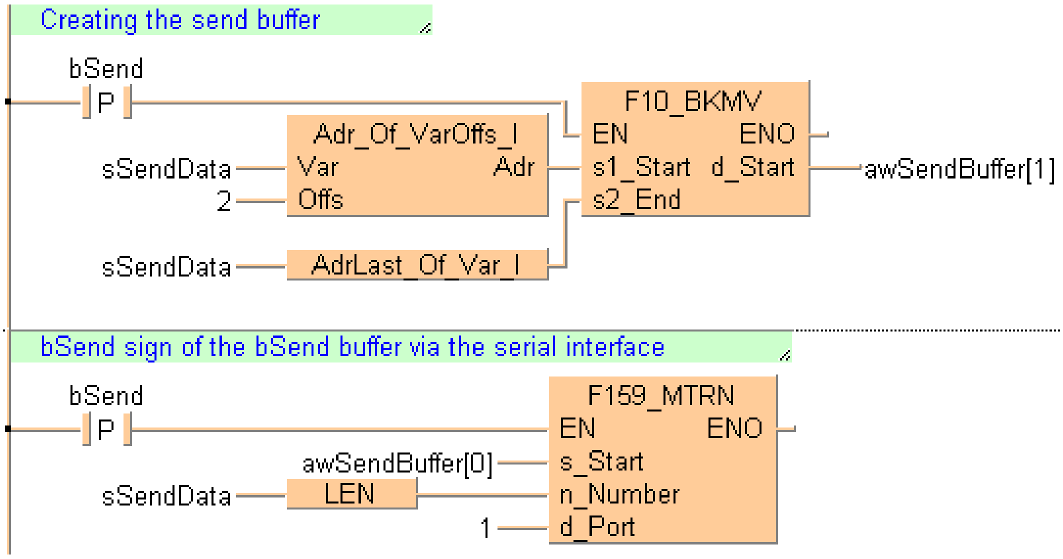
Body When the variable **bSend** is set to TRUE, the function F10_BKMV copies the characters of the string **sSendData** to the buffer **awSendBuffer** beginning at **awSendBuffer[1]**.

The first two words of a string contain the string header information (maximum number of characters and the current number of characters). The string header must not be copied into the buffer. Therefore, enter an offset of 2 to the starting address of the string before copying the data.

Make sure that the send buffer is big enough for all the data to be sent. To determine its size you must take into account that two characters of the string **SendString** can be copied into each element of the array **SendBuffer**. **SendBuffer[0]** is reserved for the total number of bytes to be sent by F159_MTRN.

Then, F159_MTRN sends the data from the first element of the send buffer (**awSendBuffer[0])** as specified by **s_Start**. The length of the string to be sent (8 bytes) is set at **n_Number** (using the function LEN to calculate the number of bytes). The data is output from COM port 1 as specified by **d_Port.**

LD



Creating the send buffer

bSend sign of the bSend buffer via the serial interface

ST When programming with structured text, enter the following:

```
if (DF(bSend)) then
      (* Copy all characters of the SendString to the SendBuffer from position
1 *)
      F10_BKMV(s1_Start := Adr_Of_VarOffs(Var := sSendData, Offs := 2),
                   s2_End   := AdrLast_Of_Var(sSendData),
                   d_Start  => awSendBuffer[1]);
      (* Send the data of the SendBuffer via the COM Port 2 of the MCU unit
in slot 3 *)
      (* In SendBuffer[0] the number of bytes not yet transmitted is stored
*)
      F159_MTRN(s_Start := SendBuffer[0], n_Number := LEN(sSendData), d_Port
:= 16#0302);
end_if;
```

Further information:

IsTransmissionDone (see page 763)

22.4.1.1 Format of send and receive data

Remember the following when accessing data in the send and receive buffers:

- The format of the data in the send buffer depends on the data type of the transmission data (e.g. STRING) and on the conversion function used in the PLC program (e.g. F95_ASC (see page 661)). There is no conversion when data in the send buffer is sent.
- The start and end codes specified in the system registers are automatically added to the data sent. The start code is added at the beginning, the end code at the end of the send string. Do not include start or end codes in the send string.
- The format of the data in the receive buffer depends on the data format used by the external device. Use a conversion function to convert the data into the desired format, e.g. F27_AHEX.
- Start and end codes in the data received are recognized if the corresponding start and end codes have been specified in the system registers. Start and end codes are not stored in the receive buffer. The end code serves as a reception done condition, i.e., the "reception done" flag turns to TRUE when the end code is received. The start code resets the receive buffer.
- If "None" is selected for the start code, a start code is not added to the data sent and is not recognized in the data received. Without start code, the receive buffer can only be reset by executing F159_MTRN.
- If "None" is selected for the end code, an end code is not added to the data sent and is not recognized in the data received. Without end code, the "reception done" flag does not turn to TRUE. The end of reception can only be determined by a time-out using the IsReceptionDoneByTimeOut function or by evaluating the data in the receive buffer.

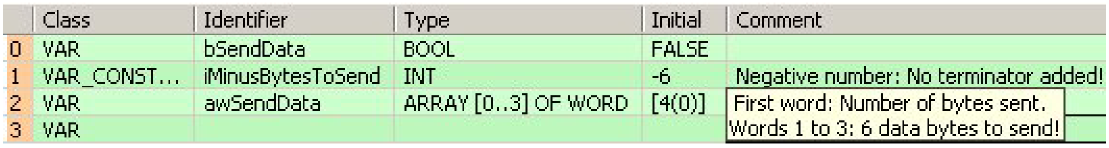
Different end code settings for sending and receiving

Sometimes you do not want to send an end code, but need an end code in the data received to set the "reception done" flag to TRUE. In this case, select the desired end code in the system registers and execute F159_MTRN specifying a negative number for **n_Number**.

◆ **EXAMPLE**

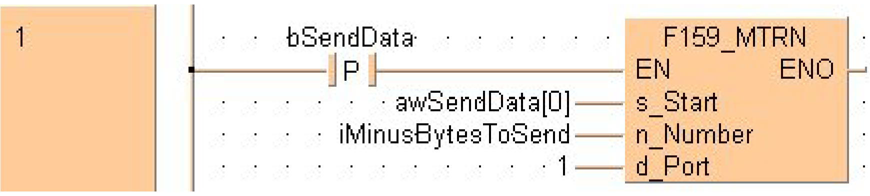
Send 4 bytes of data without adding an end code:

POU Header

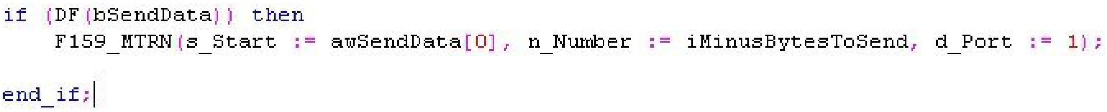


	Class	Identifier	Type	Initial	Comment
0	VAR	bSendData	BOOL	FALSE	
1	VAR_CONST...	iMinusBytesToSend	INT	-6	Negative number: No terminator added!
2	VAR	awSendData	ARRAY [0..3] OF WORD	[4(0)]	First word: Number of bytes sent. Words 1 to 3: 6 data bytes to send!
3	VAR				

LD Body



ST Body



```
if (DF(bSendData)) then
    F159_MTRN(s_Start := awSendData[0], n_Number := iMinusBytesToSend, d_Port := 1);

end_if;
```

Preparing the system for the reception of further data

■ Communication port of the CPU

In order to receive the next data, reset the receive buffer. This is done automatically when sending the next data with F159_MTRN:

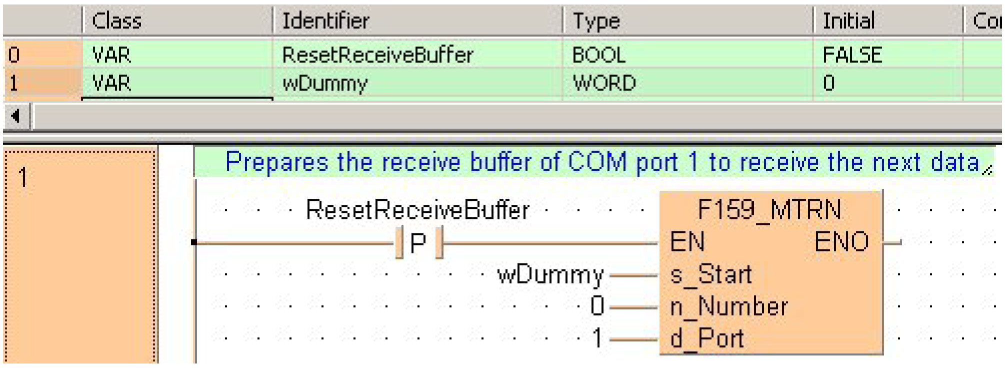
- Resetting the receive buffer sets the number of bytes received in offset 0 to 0 and moves the write pointer back to offset 1. Subsequent data will be stored in the receive buffer starting at offset 1. (The receive buffer is not cleared).
- The "reception done" flag turns to FALSE.

To reset the receive buffer without sending further data, execute F159_MTRN (see page 741) with **n_Number** = 0.

☞ **You can only execute F159_MTRN with the number of bytes equal to zero for the COM ports of a CPU; otherwise an operation error will occur.**

POU Header and LD Body

All input and output variables which are required for programming the function are declared in the POU header.



	Class	Identifier	Type	Initial	Co
0	VAR	ResetReceiveBuffer	BOOL	FALSE	
1	VAR	wDummy	WORD	0	

1

Prepares the receive buffer of COM port 1 to receive the next data

ST Body

```
if (DF(ClearTheReceiveBuffer)) then
      (* Clears the receive buffer of the COM1 port of the FP-SIGMA *)
      F159_MTRN(s_Start := wDummy, n_Number := 0, d_Port := 1);
end_if;
```

- **Communication port of the MCU:**

Receiving data from the MCU using F161_MRCV (see page 755) implicitly clears the reception area and resets the "reception done flag". Hence the communication port can again receive data.

22.4.2 Receiving data from external devices

Steps for receiving data from external devices:

1. Setting the communication parameters and specifying the receive buffer
2. Receiving the data
3. Processing the data in the receive buffer
4. Preparing the system to receive subsequent data

☞ ◆**NOTE**

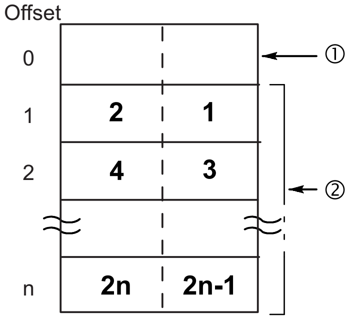
Data received via the communication ports of an MCU has to be moved to the CPU receive buffer using the instruction F161_MRCV (see page 755).

1. Setting the communication parameters (see)

2. Receiving the data

Data is automatically received in the receive buffer defined in the system registers. Reception can be controlled by the "reception done" flag or by directly evaluating the receive buffer. (For details on flag operation, see page 757.) When this flag is FALSE and data is sent to the communication port from an external device, operation takes place as follows. (The "reception done" flag turns to FALSE after switching to RUN mode.)

- Incoming data is stored in the receive buffer. Start and end codes are not stored in the receive buffer. The storage area for the data received starts with the second word of the receive buffer (offset 1). Offset 0 contains the number of bytes received. The initial value of offset 0 is 0.



① Storage area for the number of bytes received

② Storage area for the data received

Bold numbers indicate the order of reception.

- When the end code is received, the "reception done" flag turns to TRUE. Reception of any further data is prohibited. The "reception done" flag only turns to TRUE if an end code, e.g. CR, has been selected in the system registers.

3. Processing the data in the receive buffer

- Verify the end of reception.
- Copy the data in the receive buffer to a target area defined in the program using a transfer instruction, e.g. F10_BKMV (see page 819).

◆**NOTE**

For details on the operation of the "reception done" flag, see page 757

4. Preparing the system for the reception of further data

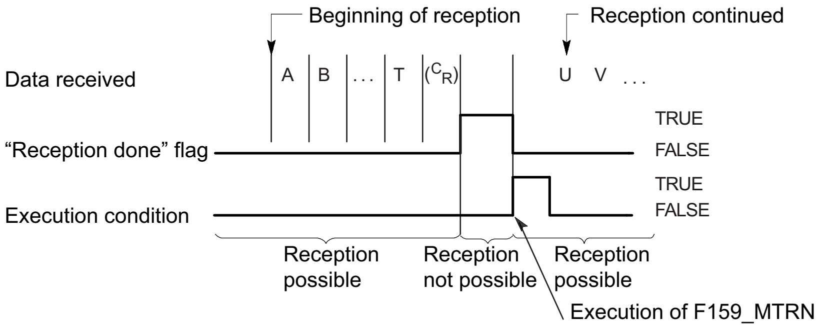
In order to receive the next data, reset the receive buffer. This is done automatically when sending the next data with F159_MTRN:

- Resetting the receive buffer sets the number of bytes received in offset 0 to 0 and moves the write pointer back to offset 1. Subsequent data will be stored in the receive buffer starting at offset 1. (The receive buffer is not cleared).
- The "reception done" flag turns to FALSE.

To reset the receive buffer without sending further data, execute F159_MTRN (see page 741) with **n_Number** = 0.

◆ EXAMPLE

Receive a string of 8 bytes containing the characters "ABCDEFGH" via COM port 1. The characters are stored in ASCII HEX code without start and end codes.

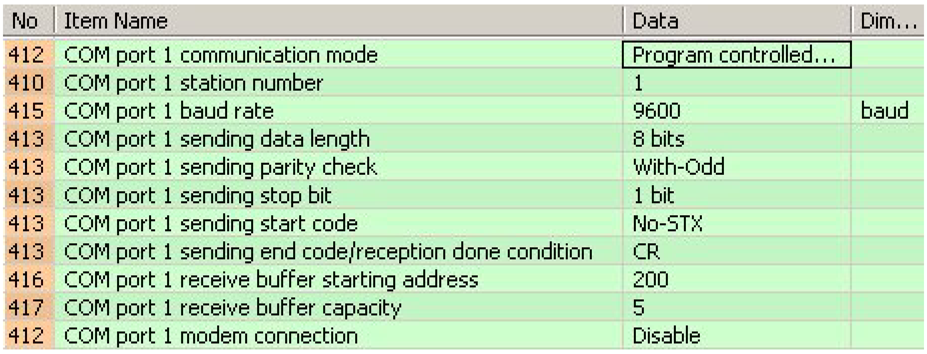


Receive buffer layout:

Offset		
0	8	
1	16#42(B)	16#41(A)
2	16#44(D)	16#43(C)
3	16#46(F)	16#45(E)
4	16#48(H)	16#47(G)

When reception begins, the value in offset 0 is 8. At the end of reception, the value in offset 0 is 0. The data in offset 1 to offset 4 is received in order from the low order byte.

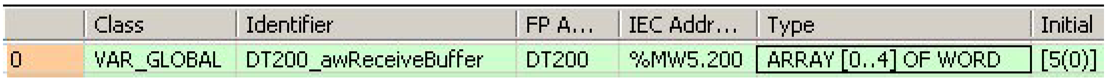
System register settings:



No	Item Name	Data	Dim...
412	COM port 1 communication mode	Program controlled...	
410	COM port 1 station number	1	
415	COM port 1 baud rate	9600	baud
413	COM port 1 sending data length	8 bits	
413	COM port 1 sending parity check	With-Odd	
413	COM port 1 sending stop bit	1 bit	
413	COM port 1 sending start code	No-STX	
413	COM port 1 sending end code/reception done condition	CR	
416	COM port 1 receive buffer starting address	200	
417	COM port 1 receive buffer capacity	5	
412	COM port 1 modem connection	Disable	

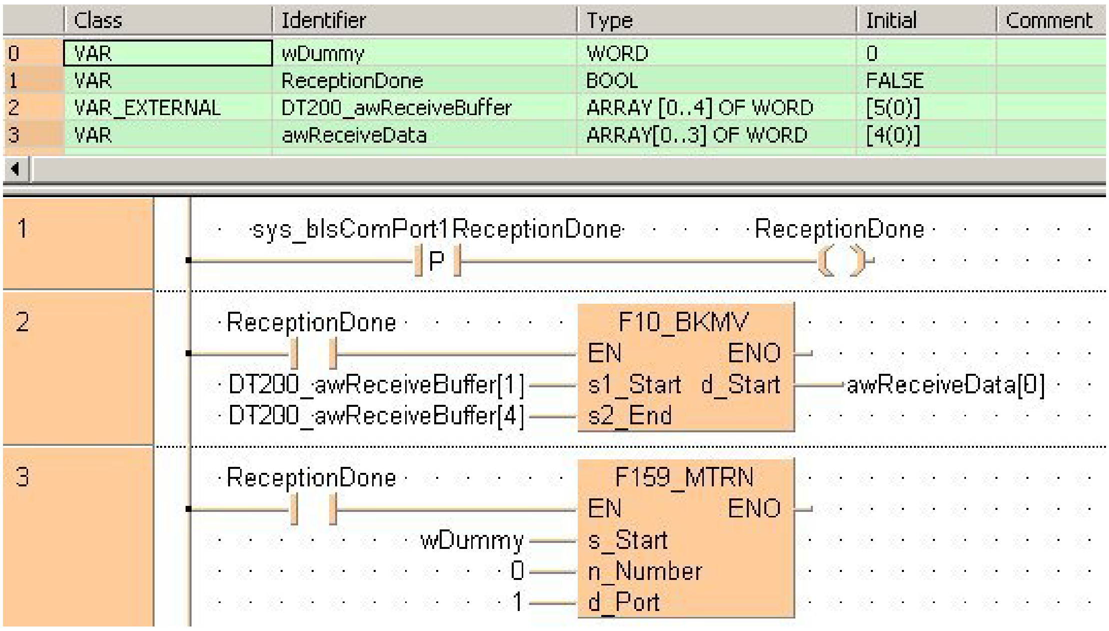
In order to use the data in the receive buffer, define a global variable having the same starting address and capacity. In this example, the starting address is 200 (VAR_GLOBAL ReceivedData) and the receive buffer capacity is 5 (ARRAY [0..4] OF WORD).

GVL



	Class	Identifier	FP A...	IEC Addr...	Type	Initial
0	VAR_GLOBAL	DT200_awReceiveBuffer	DT200	%MW5.200	ARRAY [0..4] OF WORD	[5(0)]

POU Header and LD Body



	Class	Identifier	Type	Initial	Comment
0	VAR	wDummy	WORD	0	
1	VAR	ReceptionDone	BOOL	FALSE	
2	VAR_EXTERNAL	DT200_awReceiveBuffer	ARRAY [0..4] OF WORD	[5(0)]	
3	VAR	awReceiveData	ARRAY[0..3] OF WORD	[4(0)]	

ST Body

```
if (sys_bIsComPort1ReceptionDone) then
    F10_BKMV(s1_Start := DT200_awReceiveBuffer[1], s2_End := DT200_awReceiveBuffer[4],
    d_Start => awReceiveData[0]);
    F159_MTRN(s_Start := wDummy, n_Number := 0, d_Port := 1);
end_if;
```

Data can be received when the "reception done" flag is FALSE. The "reception done" flag is evaluated by the system variable sys_bIsComPort1ReceptionDone. When the reception of the data is complete (the end code has been received), the "reception done" flag turns to TRUE, and subsequently, receiving data is prohibited. To prepare the system to receive the next data without immediately sending further data, the receive buffer is reset by executing F159_MTRN with **n_Number** = 0.

◆**NOTE**

The status of the "reception done" flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

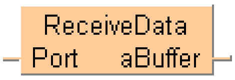
The start code "STX" resets the receive buffer. Resetting the receive buffer sets the number of bytes received in offset 0 to 0 and moves the write pointer back to offset 1. Subsequent data will be stored in the receive buffer starting at offset 1.

For details on the format of the data in the send buffer and in the receive buffer, please see "Format of send and receive data" on page 745.

ReceiveData

Receive data from CPU or MCU port

Description This instruction copies the received data of the port specified by the variable at **Port** into the data applied at **aBuffer**.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

See also:

- ReceiveCharacters (see page 752)
- ClearReceiveBuffer (see page 753)

PLC types see see page 1330

Data types

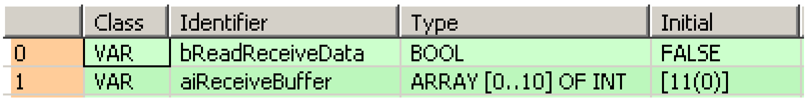
Input variable	Data type	Function
Port	ANY16	▪ Communication port ▪ Must be a constant **FP-X, FPΣ and FP2, FP2SH (V1.4 or later):** ▪ PLC communication ports: Value: SYS_COM1_PORT or SYS_COM2_PORT or SYS_TOOL_PORT ▪ MCU communication port: Value: 16#xx01 (COM1), 16#xx02 (COM2) xx = slot number (hexadecimal) of the MCU (e.g. 16#0001: COM1 in slot 0, 16#0A02: COM2 in slot 10, 16#1401: COM1 in slot 20) **Other PLCs:** The command will be compiled to F144_TRNS, which works on the COM port of the CPU (the parameter **d_Port** will be ignored)
Output variable		
aBuffer	ANY	stores the receive data

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the MCU unit does not exist at the slot no. specified by '**Port**'.
R9008	%MX0.900.8	for an instant	

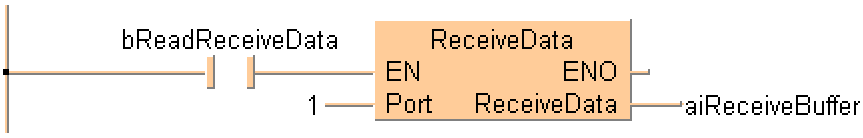
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bReadReceiveData	BOOL	FALSE
1	VAR	aiReceiveBuffer	ARRAY [0..10] OF INT	[11(0)]

LD



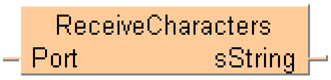
ST When programming with structured text, enter the following:

```
if (bReadReceiveData) then
      aiReceiveBuffer:=ReceiveData(1);
end_if;
```

ReceiveCharacters

Receive characters from CPU or MCU port

Description This instructions receives characters from a variable port number **Port** and stores the string in the variable **sString**.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types see see page 1330

Data types

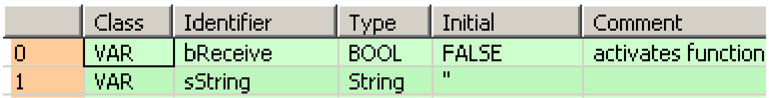
Input variable	Data type	Function
Port	ANY16	- Communication port - Must be a constant **FP-X, FPΣ and FP2, FP2SH (V1.4 or later):** - PLC communication ports: Value: SYS_COM1_PORT or SYS_COM2_PORT or SYS_TOOL_PORT - MCU communication port: Value: 16#xx01 (COM1), 16#xx02 (COM2) xx = slot number (hexadecimal) of the MCU (e.g. 16#0001: COM1 in slot 0, 16#0A02: COM2 in slot 10, 16#1401: COM1 in slot 20) **Other PLCs:** The command will be compiled to F144_TRNS, which works on the COM port of the CPU (the parameter **d_Port** will be ignored)
Output variable		
sString	STRING	string to be received

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	- the MCU unit does not exist at the slot no. specified by '**Port**'.
R9008	%MX0.900.8	for an instant	

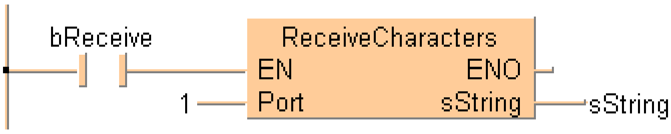
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	bReceive	BOOL	FALSE	activates function
1	VAR	sString	String	''	

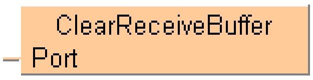
LD



ST
```
If (bReceive) Then
        sString:=ReceiveCharacters(1);
End_if;
```

ClearReceiveBuffer Reset the receive buffer

Description This instruction resets the receive buffer to be ready for the next data at the port number **Port**.



☞ **The "reception done" flag turns to FALSE.**

PLC types see see page 1318

Data types

Variable	Data type	Function
Port	ANY16	▪ Communication port ▪ Must be a constant **FP-X, FPΣ and FP2, FP2SH (V1.4 or later):** ▪ PLC communication ports: Value: SYS_COM1_PORT or SYS_COM2_PORT or SYS_TOOL_PORT ▪ MCU communication port: Value: 16#xx01 (COM1), 16#xx02 (COM2) xx = slot number (hexadecimal) of the MCU (e.g. 16#0001: COM1 in slot 0, 16#0A02: COM2 in slot 10, 16#1401: COM1 in slot 20) **Other PLCs:** The command will be compiled to F144_TRNS, which works on the COM port of the CPU (the parameter **d_Port** will be ignored)

Operands

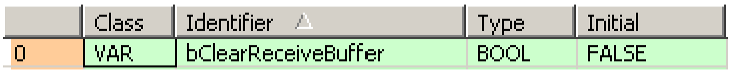
For	Relay				T/C		Register			Constant
Port	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Error flags

No.	IEC address	Set	If
R900B	%MX0.900.11	for an instant	▪ the communication port specified by **Port** does not exist.
R9009	%MX0.900.9	for an instant	

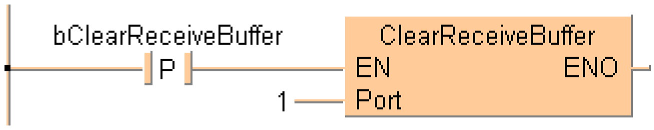
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bClearReceiveBuffer	BOOL	FALSE

LD

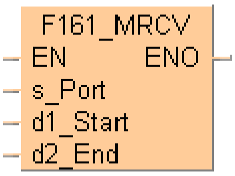


```
ST if (DF(bClearReceiveBuffer)) then
          ClearReceiveBuffer(1);
   end_if;
```

F161_MRCV

Read serial data from the MCU's COM port

Description Use this instruction to copy the data received in the MCU from the external device to the specified receive buffer in the CPU. The receive buffer is defined by **d1_Start** and **d2_End**.



```
F161_MRCV
EN        ENO
s_Port
d1_Start
d2_End
```

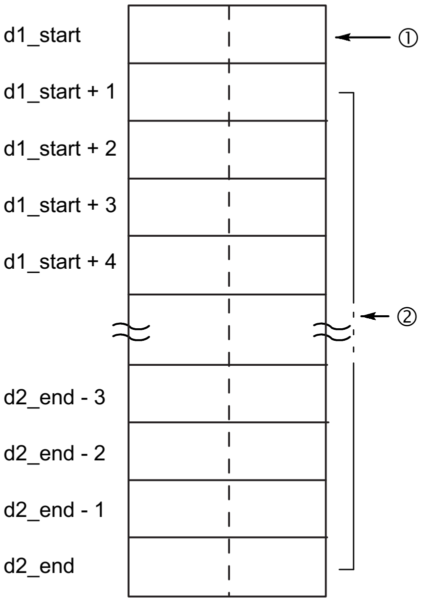
This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Do not execute F161_MRCV unless the end of reception has been verified by evaluating the "reception done" flag. Polling the data using F161_MRCV does not work correctly! The "reception done" flag can be evaluated using the IsReceptionDone (see page 760) function. Or use the system variable sys_bIsComPort1ReceptionDone, sys_bIsComPort2ReceptionDone, or sys_bIsToolPortReceptionDone, depending on the port. The end of reception can also be determined by time-out using the IsReceptionDoneByTimeOut (see page 761) function or by checking the contents of the receive buffer.

The number of bytes received is stored in the initial address specified by **d1_Start** of the receive buffer. If the data received exceeds the ending address specified by **b2_End**, an operation error is detected. The data which has been received up to **d2_End** will be stored. F161_MRCV also clears the receive buffer (see page 746), resets the "reception done flag" and allows further reception of data.

F161_MRCV is supported by all PLCs: If suitable functions and system variables are used instead of flags, PLC-independent programs can be created which handle communication for CPU communication ports as well as for MCU ports. PLCs not using MCU ports simply do not translate the F161_MRCV instruction.

Receive buffer



① Storage area for the number of bytes received

② Storage area for the data received

Data types

Variable	Data type	Function
s_Port	ANY16	Specification of slot number (high byte) and port number (low byte) of the MCU to which the data is transmitted. 16#xx01: COM1 on MCU in slot 16#xx 16#xx02: COM2 on MCU in slot 16#xx
d1_Start		Starting address of the receive buffer
d2_End		Ending address of the receive buffer

Operands

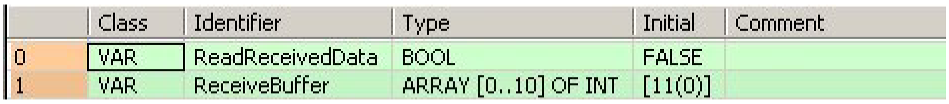
For	Relay				T/C		Register			Constant
s_Port	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1_Start	-	WY	WR	WL	SV	EV	DT	LD	FL	-
d2_End	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the specified address using the index modifier exceeds a limit ▪ the MCU unit does not exist in the specified slot ▪ the specified communication port does not exist
R9008	%MX0.900.8	for an instant	

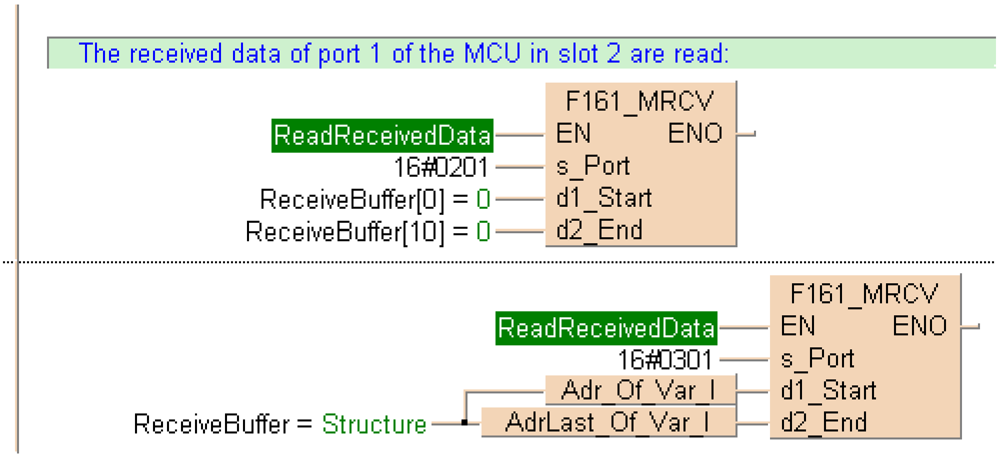
Example In this example the function is programmed in ladder diagram (LD). The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	ReadReceivedData	BOOL	FALSE	
1	VAR	ReceiveBuffer	ARRAY [0..10] OF INT	[11(0)]	

LD



The received data of port 1 of the MCU in slot 2 are read:

22.4.3 Flag operation in program controlled communication

Program controlled communication provides for half duplex communication, i.e. communication is possible in both directions, but not simultaneously. Sending can be controlled by the "transmission done" flag. Reception can be controlled by the "reception done" flag or by directly evaluating the receive buffer.

The flags are special internal relays which turn to TRUE or to FALSE under specific conditions. They can be evaluated using special functions or system variables.

"Reception done" flag

When the end code is received, the "reception done" flag turns to TRUE. Reception of any further data is prohibited. F159_MTRN (see page 741) turns the "reception done" flag to FALSE.

The "reception done" flag can be evaluated using the IsReceptionDone (see page 760) function. Or use the system variable sys_bIsComPort1ReceptionDone, sys_bIsComPort2ReceptionDone, or sys_bIsToolPortReceptionDone, depending on the port. The end of reception can also be determined by time-out using the IsReceptionDoneByTimeOut (see page 761) function or by checking the contents of the receive buffer.

The status of the "reception done" flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

FP0:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
COM	1	R9038	IsReceptionDone (see page 760)	sys_bIsComPort1ReceptionDone	TRUE

FP0R:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
TOOL	0	R903E	IsReceptionDone (see page 760)	sys_bIsToolPortReceptionDone	TRUE
COM1	1	R9038		sys_bIsComPort1ReceptionDone	

FPΣ, FP-X:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
TOOL	0	R903E	IsReceptionDone (see page 760)	sys_bIsToolPortReceptionDone	TRUE
COM1	1	R9038		sys_bIsComPort1ReceptionDone	
COM2	2	R9048		sys_bIsComPort2ReceptionDone	

FP2/FP2SH/FP10SH:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
COM	0	R9038	IsReceptionDone (see page 760)	sys_bIsComPort1ReceptionDone	TRUE

MCU:

Port name	Port number	Input	Function name	Bit status
COM1	16#xx01	X0	IsReceptionDone (see page 760)	TRUE
COM2	16#xx02	X2		

xx = slot number (hexadecimal)

For detailed information on the MCU input (X) flags, see Getting in RUN Mode via the Input (X) Flags (see page 732).

"Transmission done" flag

When the specified number of bytes has been sent, the "transmission done" flag turns to TRUE. New data may be sent or received. F159_MTRN (see page 741) turns the "transmission done" flag to FALSE. While F159_MTRN is executed, no data can be received.

The "transmission done" flag can be evaluated using the IsTransmissionDone (see page 763) function. Or use the system variable sys_bIsComPort1TransmissionDone, sys_bIsComPort2TransmissionDone, or sys_bIsToolPortTransmissionDone, depending on the port.

FP0:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
COM	1	R9039	IsTransmissionDone (see page 763)	sys_bIsComPort1TransmissionDone	TRUE

FP0R:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
TOOL	0	R903F	IsTransmissionDone (see page 763)	sys_bIsToolPortTransmissionDone	TRUE
COM1	1	R9039		sys_bIsComPort1TransmissionDone	

FPΣ, FP-X:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
TOOL	0	R903F	IsTransmissionDone (see page 763)	sys_bIsToolPortTransmissionDone	TRUE
COM1	1	R9039		sys_bIsComPort1TransmissionDone	
COM2	2	R9049		sys_bIsComPort2TransmissionDone	

FP2/FP2SH/FP10SH:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
COM	0	R9039	IsTransmissionDone (see page 763)	sys_bIsComPort1TransmissionDone	TRUE

MCU:

Port name	Port numbert	Input	Function name	Bit status
COM1	16#xx01	X4	IsTransmissionDone (see page 763)	TRUE
COM2	16#xx02	X5		

xx = slot number (hexadecimal)

For detailed information on the MCU input (X) flags, see Getting in RUN Mode via the Input (X) Flags (see page 732).

xx = slot number (hexadecimal)

Communication error flag

If the communication error flag turns to TRUE during reception, reception continues. Execute F159_MTRN (see page 741) to turn the error flag to FALSE and to move the write pointer back to offset 1.

The communication error flag can be evaluated using the IsCommunicationError function. Or use the system variable sys_bIsComPort1CommunicationError, sys_bIsComPort2CommunicationError, or sys_bIsToolPortCommunicationError, depending on the port.

FP0:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
COM	1	R9037	IsCommunicationError (see page 764)	sys_bIsComPort1CommunicationError	TRUE

FP0R:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
TOOL	0	R900E	IsCommunicationError (see page 764)	sys_bIsToolPortCommunicationError	TRUE
COM1	1	R9037		sys_bIsComPort1CommunicationError	

FPΣ, FP-X:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
TOOL	0	R900E	IsCommunicationError (see page 764)	sys_bIsToolPortCommunicationError	TRUE
COM1	1	R9037		sys_bIsComPort1CommunicationError	
COM2	2	R9047		sys_bIsComPort2CommunicationError	

FP2/FP2SH/FP10SH:

Port name	Port number	Special internal relay	Function name	System variable name	Bit status
COM	0	R9037	IsCommunicationError (see page 764)	sys_bIsComPort1CommunicationError	TRUE

MCU:

Port name	Port numbert	Input	Function name	Bit status
COM1	16#xx01	X6	IsCommunicationError (see page 764)	TRUE
COM2	16#xx02	X7		

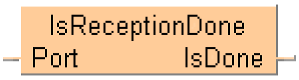
xx = slot number (hexadecimal)

For detailed information on the MCU input (X) flags, see Getting in RUN Mode via the Input (X) Flags (see page 732).

IsReceptionDone

Evaluation of "reception done" flag for all ports

Description This function returns the value of the "**reception done**" flag. The "reception done" flag is TRUE if the end code has been received at the assigned communication port of the PLC.



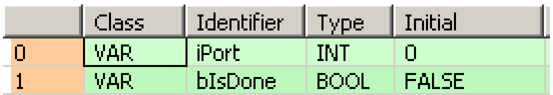
See also: IsReceptionDoneByTimeOut (see page 761)

Data types

Input variable	Data type	Function
Port	ANY16	- Communication port - Must be a constant **FP-X, FPΣ and FP2, FP2SH (V1.4 or later):** - PLC communication ports: Value: SYS_COM1_PORT or SYS_COM2_PORT or SYS_TOOL_PORT - MCU communication port: Value: 16#xx01 (COM1), 16#xx02 (COM2) xx = slot number (hexadecimal) of the MCU (e.g. 16#0001: COM1 in slot 0, 16#0A02: COM2 in slot 10, 16#1401: COM1 in slot 20) **Other PLCs:** The command will be compiled to F144_TRNS, which works on the COM port of the CPU (the parameter **d_Port** will be ignored)
Output variable		
IsDone	BOOL	set to TRUE, if the end code has been received. The end code is specified in the corresponding system register under COM port settings.

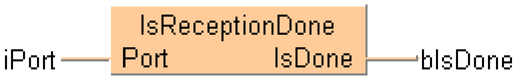
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iPort	INT	0
1	VAR	bIsDone	BOOL	FALSE

LD

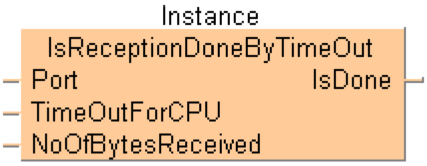


ST
```
bIsDone:=IsReceptionDone(Port := iPort);
```

IsReceptionDone ByTimeout

Evaluation of "reception done" condition by time-out for all ports

Description Depending on the PLC type and the input parameter **Port**, this function evaluates the "reception done" condition if no end code is expected in the data stream, e.g when transferring binary data.



The output **IsDone** is set to TRUE if the receive buffer is not empty and no more characters are received before the time-out specified at **TimeOutForCPU**.

Using this function block, connect the first word of the receive buffer to **NoOfBytesReceived (**number of bytes received).

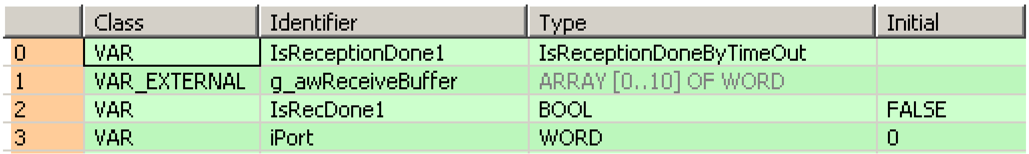
If a communication port of an MCU is selected, the MCU's "reception done" flag (see page 757) is evaluated. The timeout for this communication port must be entered via the "MCU Setting" dialog or during RUN mode via F159_MWRT_PARA (see page 719).

Data types

Input variable	Data type	Function
Port	ANY16	▪ Communication port ▪ Must be a constant **FP-X, FPΣ and FP2, FP2SH (V1.4 or later):** ▪ PLC communication ports: Value: SYS_COM1_PORT or SYS_COM2_PORT or SYS_TOOL_PORT ▪ MCU communication port: Value: 16#xx01 (COM1), 16#xx02 (COM2) xx = slot number (hexadecimal) of the MCU (e.g. 16#0001: COM1 in slot 0, 16#0A02: COM2 in slot 10, 16#1401: COM1 in slot 20) **Other PLCs:** The command will be compiled to F144_TRNS, which works on the COM port of the CPU (the parameter **d_Port** will be ignored)
TimeOutForCPU	TIME	Set the time-out. If no further data is received before the time-out, reception is done and **IsDone** is set to TRUE.
NoOfBytesReceived	ANY16	Connect the start address of the receive buffer. This address contains the number of bytes received.
Output variable		
IsDone	BOOL	Indicates that one or more bytes have been received and the number of bytes received was constant as specified in **TimeOutForCPU**.

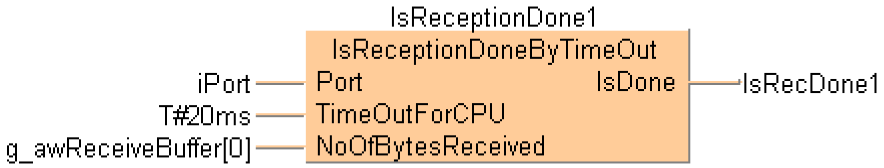
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	IsReceptionDone1	IsReceptionDoneByTimeOut	
1	VAR_EXTERNAL	g_awReceiveBuffer	ARRAY [0..10] OF WORD	
2	VAR	IsRecDone1	BOOL	FALSE
3	VAR	iPort	WORD	0

LD

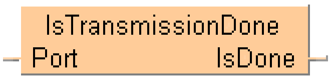


```
ST IsReceptionDone1(Port := iPort,
                            TimeOutForCPU := T#20ms,
                            NoOfBytesReceived := g_awReceiveBuffer[0],
                            IsDone => bIsRecDone1);
```

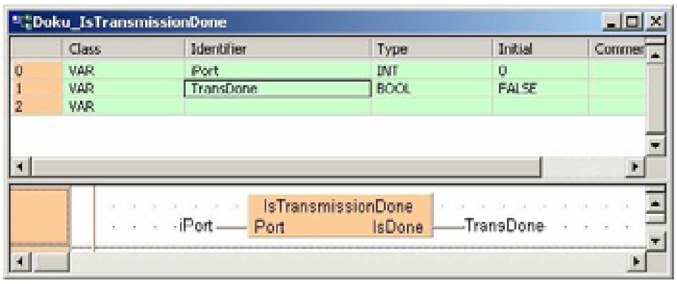
IsTransmissionDone

Evaluation of "transmission done" flag for all ports

Description This function returns the value of the "transmission done" flag. The "transmission done" flag (see page 757) is TRUE if the specified number of bytes has been sent from the assigned communication port of the PLC.



Example



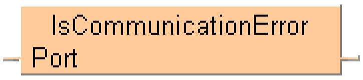
	Class	Identifier	Type	Initial	Commer
0	VAR	Port	INT	0	
1	VAR	TransDone	BOOL	FALSE	
2	VAR				

IsCommunicationError

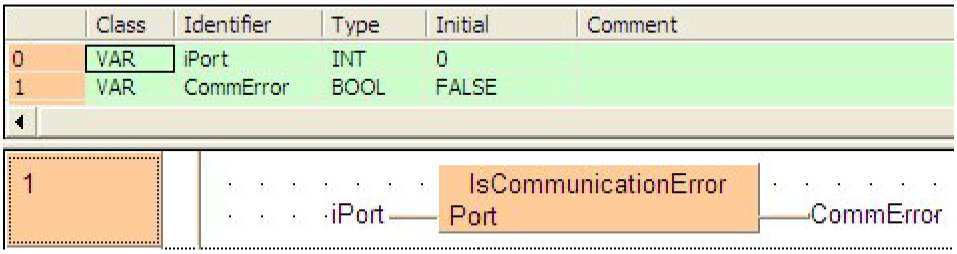
Evaluation of communication error flag for all ports

Description This instruction returns the value of the communication error flag. The communication error flag is TRUE if an error has occurred at the specified port during serial communication.

Symbol:



Example



	Class	Identifier	Type	Initial	Comment
0	VAR	iPort	INT	0	
1	VAR	CommError	BOOL	FALSE	

22.5 Data transfer in master/slave mode (MEWTOCOL/Modbus RTU)

General Programming Information for F145 and F146

- It is not possible to execute multiple F145_WRITE_DATA (see page 766) and F146_READ_DATA instructions for the same communication port simultaneously. The program should be set up so that these instructions are executed when the SEND/RECV execution enabled flag (R9044: COM1/R904A: COM2) is ON.

COM1	**sys_bIsComPort1F145F146NotActive**	**R9044**	**0:** Execution inhibited. (SEND/RECV instruction being executed.) **1:** Execution enabled.
COM2	**sys_bIsComPort2F145F146NotActive**	**R904A**	

- The SEND (i.e. F145_WRITE_DATA) instruction only requests that data be sent, but the actual processing takes place when the ED instruction is executed. The SEND/RECV execution end flag (R9045: COM1, R904B: COM2) can be used to check whether or not the transmission has been completed.

COM1	**sys_bIsComPort1F145F146Error**	**R9045**	**0**: Completed normally. **1:** Completed with error. (The error code is stored in DT90045.)
COM1	**sys_wComPort1F145F146ErrorCode**	**DT90124**	If the transmission has been completed with an error (R9045 is ON), the contents of the error (error code) are stored.
COM2	**sys_bIsComPort2F145F146Error**	**R904B**	**0**: Completed normally. **1**: Completed with error. (The error code is stored in DT90125.)
COM2	**sys_wComPort2F145F146ErrorCode**	**DT90125**	If the transmission has been completed with an error (R904B is ON), the contents of the error (error code) are stored.

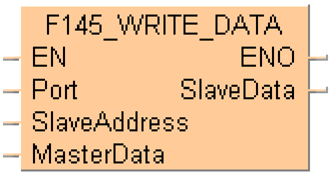
- For detailed information, please refer to error codes (see page 1306). If the error code is 16#73, a communication time-out error has occurred. The time-out length can be set from 10.0ms to 81.9s (in units of 10ms) using system register 32. The default value is 10s.

Error code	Description
16#73	Time-out: waiting for response

- For global transmission (the transmission performed by specifying 16#00 for the unit no.), the program should be set up so that the transmission is executed after the maximum scan time has elapsed.
- The F145 or F146 instruction cannot be executed if the target address is a special internal relay (from R9000) or a special data register (from DT90000).
- The compiler will use file registers in case the data registers are occupied.
- For the table of available Modbus commands, please refer to F145F146_MODBUS_COMMAND (see page 777) or F145F146_MODBUS_MASTER (see page 779).

F145_WRITE_DATA Write Data to Slave

Description Use this instruction to write data from a master to a slave via the serial port (COM1 or COM2) using MEWTOCOL or Modbus RTU protocol (see communication mode (see page 712)), as defined in the system register settings (see page 1273) of port used. Master and slave must both use the same protocol. The master must be configured in Master/Slave mode. The slave can be configured either in Master/Slave mode or in Slave mode only.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

The data specfied by **MasterWordAddress** for the master is written to the slave area specified by **SlaveWordAddress**. The variable **SlaveAddress** determines the slave's station number and the slave's COM port (1 or 2).

General programming information for F145 and F146 (see page 766)

PLC types **Availability of** F145_WRITE_DATA **(see page 1321)**

Data types

Variable	Data type	Function
Port	ANY16	Specifies the slave's COM port (1 or 2) via system variable: SYS_COM1_PORT SYS_COM2_PORT
SlaveAddress	ANY16	Address of the remote station (1-99).
MasterData	ANY	The master data which is written to the slave.
SlaveData	ANY	▪ The data of the slave to which the data is written. ▪ MasterData and SlaveData have to be of the same data type. ▪ To establish external data access from the master to the slave data please assign fixed user addresses (same addresses as slave data) in the global variable list.

Operands

For	Relay				T/C		Register			Constant
Port	WX	WY	WR	WL	-	-	DT	LD	FL	-
Slave Address	WX	WY	WR	WL	-	-	DT	LD	FL	dec. or hex.
Master Data	WX	WY	WR	WL	-	-	DT	LD	FL	-
Slave Data	WX	WY	WR	WL	-	-	DT	LD	FL	-

Error flags

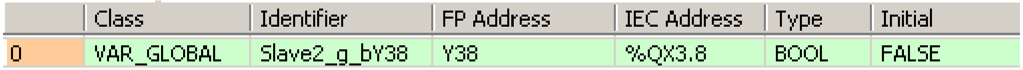
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ port 0 (global transmission) gets no response from COM1 or COM2. ▪ slave data or master data exceeds the available address range. ▪ the communication mode (see page 712) is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave. ▪ the COM port selected requires a communication cassette that has not been installed.
R9008	%MX0.900.8	permanently	

☞

- **If the slave data is not available in the user area of the master, please use either the instruction F145_WRITE_DATA_TYPE_OFFS (see page 769) or the F145F146_MODBUS_COMMAND (see page 777).**
- **For another station number outside the range (0-99) or another start register as available in the table of modbus commands (see page 777), please use the modbus function blocks of the Modbus Library for FPWIN Pro (NCL-MODBUS-LIB).**

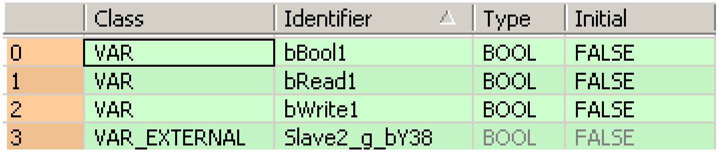
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

GVL In the global variable list you define variables that can be accessed by all POUs in the project.



	Class	Identifier	FP Address	IEC Address	Type	Initial
0	VAR_GLOBAL	Slave2_g_bY38	Y38	%QX3.8	BOOL	FALSE

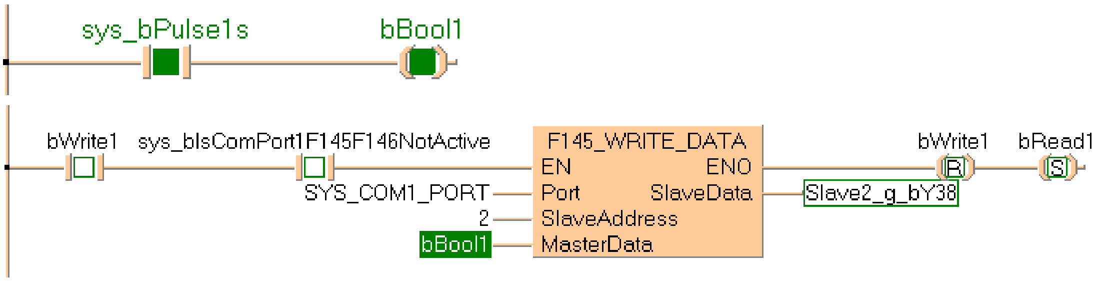
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bBool1	BOOL	FALSE
1	VAR	bRead1	BOOL	FALSE
2	VAR	bWrite1	BOOL	FALSE
3	VAR_EXTERNAL	Slave2_g_bY38	BOOL	FALSE

Body The system variable **sys_bPulse1s** is copied to **bBool1**. If **bWrite1** and **sys_bIsComPort1F145F146NotActive** are set to TRUE, **bBool1** is written to the output Y38 of slave 2.

LD



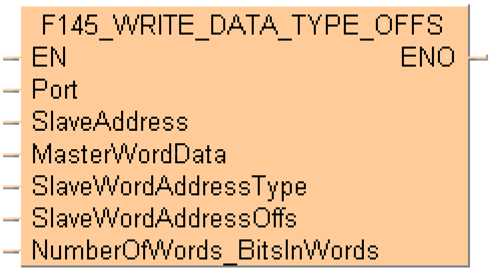
ST When programming with structured text, enter the following:

```
bBool1 := sys_bPulse1s;

if (bWrite1 and sys_bIsComPort1F145F146NotActive) then
      F145_WRITE_DATA(Port := SYS_COM1_PORT,
                            SlaveAddress := 2,
                            MasterData := bBool1,
                            SlaveData => Slave2_g_bR15);

      bRead1 := true;
      bWrite1 := false;
end_if;
```

F145_WRITE_DATA_TYPE_OFFS

Write Data to Slave with Type and Offset

Description Use this instruction to write data from a master to a slave via the serial port (COM1 or COM2) using MEWTOCOL or Modbus RTU protocol (see communication mode (see page 712)), as defined in the system register settings (see page 1273) of port used. Master and slave must both use the same protocol. The master must be configured in Master/Slave mode. The slave can be configured either in Master/Slave mode or in Slave mode only.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

The master data specfied by **MasterWordData** and by **NumberOfWords_ BitsInWords** is written to the slave area specified by **SlaveWordAddressType** and **SlaveAddressOffs**.

General programming information for F145 and F146 (see page 766)

PLC types **Availability of** F145_WRITE_DATA_TYPE_OFFS **(see page 1321)**

Data types

Variable	Data type	Function
Port	ANY16	Specifies the slave's COM port (1 or 2) via system variable: SYS_COM1_PORT SYS_COM2_PORT
SlaveAddress		Address of the remote station (1-99).
MasterWordData	ANY	The master data which is written to the slave.
SlaveWordAddressType	ANY16	Address type in the slave to which data is written. The offset must be zero e.g. DT0, WL0...
SlaveWordAddressOffs		The offset for the starting slave address whose type is defined by **SlaveWordAddressType** and to which the data is written.
NumberOfWords_ BitsInWords		Number of word units to be sent to the master (if the highest bit is not set) or bits in words (if the highest bit is set). Is identical to the lower word of **s1_ControlData**.

s1_ControlData

	Higher word				Lower word			
Hex	1	0 fixed	0	7	8	1	0 fixed	0

COM port (16#1 or 16#2)

Unit No. (16#00 to 16#63) (0 to 99)

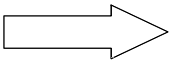
Bit unit transmission

Bit No. of Slave (16#0 to 16#F)

Bit No. of Master (16#0 to 16#F)

- To generate function code 05, bit unit transmission (16#8) must be specified.

s1_ControlData:	16#10078100
s2_MasterStartAddr:	16#0000
d_SlaveStartAddrType:	WY0
d_SlaveStartAddrOffs:	1



Modbus command

1	Slave address	07
2	Function code (16#05)	05
3	Coil No. (H)	00
4	Coil No. (L)	11
5	Setting status (H)	FF
6	Setting status (L)	00
7	CRC16 (H)	DC
8	CRC16 (L)	59

- After the ON or OFF value of bit 0 of s2_MasterStartAddr has been read in the master, this value is set in the slave (ON=FF00, OFF=0000).

☞

- **The compiler calculates the higher word from Port and SlaveAddress. The higher word is set implicitely.**
- **The lower word is specified by NumberOfWords_BitsInWords.**

Operands

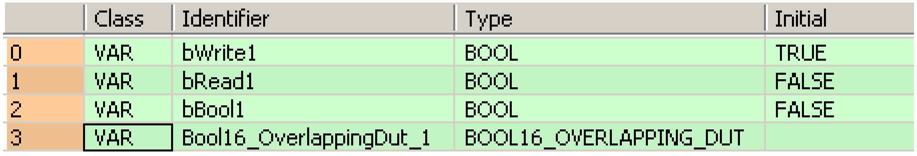
For	Relay				T/C		Register			Constant
Port	WX	WY	WR	WL	-	-	DT	LD	FL	-
Slave Address	WX	WY	WR	WL	-	-	DT	LD	FL	dec or hex
Master WordData	WX	WY	WR	WL	-	-	DT	LD	FL	-
SlaveWord Address Type	WX	WY	WR	WL	-	-	DT	LD	FL	-
SlaveWord AddressOffs	WX	WY	WR	WL	-	-	DT	LD	FL	dec or hex
NumberOfWords_ BitsInWords	WX	WY	WR	WL	-	-	DT	LD	FL	dec or hex

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ port 0 (global transmission) gets no response from COM1 or COM2. ▪ slave data or master data exceeds the available address range ▪ SlaveWordAddressType: Offset ≠ 0 ▪ the communication mode (see page 712) is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave. ▪ the selected COM port requires a communication cassette that has not been installed.
R9008	%MX0.900.8	permanently	

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

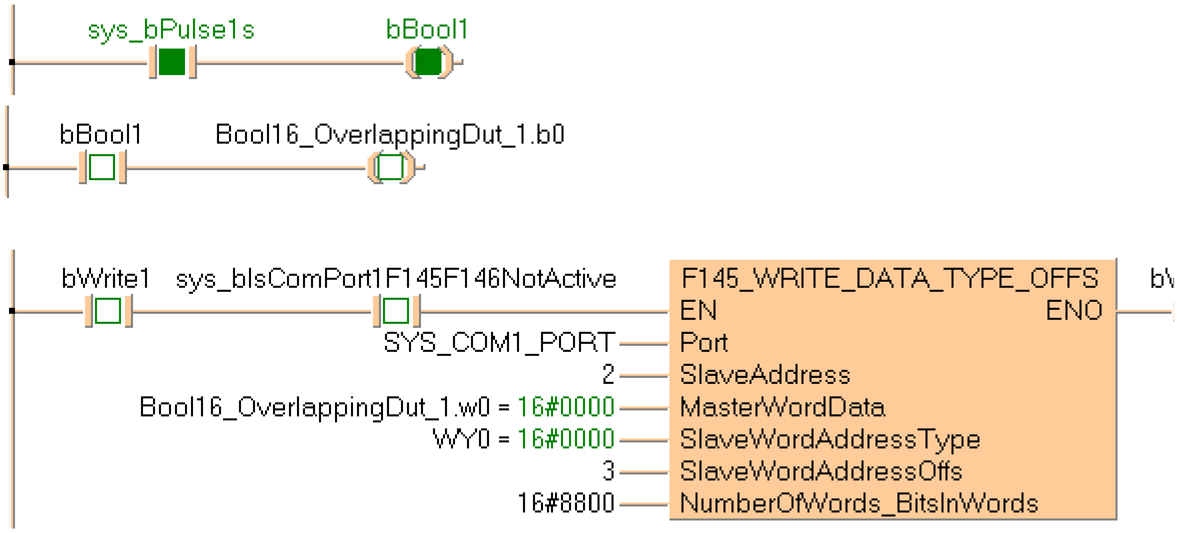
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bWrite1	BOOL	TRUE
1	VAR	bRead1	BOOL	FALSE
2	VAR	bBool1	BOOL	FALSE
3	VAR	Bool16_OverlappingDut_1	BOOL16_OVERLAPPING_DUT	

Body The system variable **sys_bPulse1s** is copied to **bBool1** and **Bool16_OverlappingDut_1.b0**. If **bWrite1** and **sys_bIsComPort1F145F146NotActive** are set to TRUE, **bBool1** is written to the output Y38 of slave 2 via **Bool16_OverlappingDut_1.b0**.

LD


sys_bPulse1s
bBool1
bBool1
Bool16_OverlappingDut_1.b0
bWrite1
sys_bIsComPort1F145F146NotActive
F145_WRITE_DATA_TYPE_OFFS
EN
ENO
b\
SYS_COM1_PORT
Port
2
SlaveAddress
Bool16_OverlappingDut_1.w0 = 16#0000
MasterWordData
WY0 = 16#0000
SlaveWordAddressType
3
SlaveWordAddressOffs
16#8800
NumberOfWords_BitsInWords

ST When programming with structured text, enter the following:

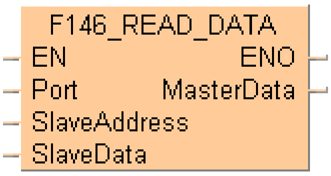
```
bBool1 := sys_bPulse1s;
Bool16_OverlappingDut_1.b0 := bBool1;

if (bWrite1 and sys_bIsComPort1F145F146NotActive) then
      F145_WRITE_DATA_TYPE_OFFS(Port := SYS_COM1_PORT,
                                          SlaveAddress := 2,
                                          MasterWordData :=
Bool16_OverlappingDut_1.w0,
                                          SlaveWordAddressType := WY0,
                                          SlaveWordAddressOffs := 3,
                                          NumberOfWords_BitsInWords :=
16#8800);

      bRead1 := true;
      bWrite1 := false;
end_if;
```

F146_READ_DATA

Read Data from Slave

Description

Use this instruction to request data from a slave via the serial port (COM1 or COM2) using MEWTOCOL or Modbus RTU protocol (see communication mode (see page 712)), as defined in the system register settings (see page 1273) of port used. Master and slave must both use the same protocol. The master must be configured in Master/Slave mode. The slave can be configured either in Master/Slave mode or in Slave mode only.



The data specfied by **MasterWordAddress** is requested by the master from the slave area specified by **SlaveWordAddress**. The variable **SlaveAddress** determines the slave's station number and the slave's COM port (1 or 2).

General programming information for F145 and F146 (see page 766)

PLC types

Availability of F146_READ_DATA **(see page 1321)**

Data types

Variable	Data type	Function
Port	ANY16	Specifies the slave's COM port (1 or 2) via system variable: SYS_COM1_PORT SYS_COM2_PORT
SlaveAddress		Address of the remote station (1-99).
SlaveData	ANY	The data of the slave to which the data is written.
MasterData		The data of the master to which the data (read by the slave) is written.

Operands

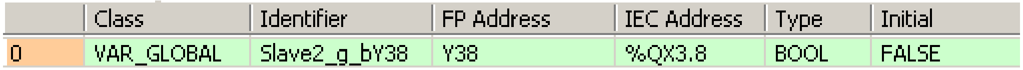
For	Relay				T/C		Register			Constant
Port	WX	WY	WR	WL	-	-	DT	LD	FL	-
Slave Address	WX	WY	WR	WL	-	-	DT	LD	FL	dec. or hex.
Master Data	WX	WY	WR	WL	-	-	DT	LD	FL	-
Slave Data	WX	WY	WR	WL	-	-	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ port 0 (global transmission) gets no response from COM1 or COM2. ▪ slave data or master data exceeds the available address range. ▪ the communication mode (see page 712) is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave. ▪ the COM port selected requires a communication cassette that has not been installed.
R9008	%MX0.900.8	permanently	

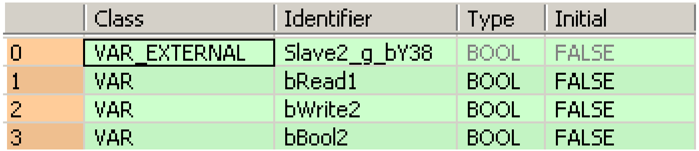
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

GVL In the global variable list you define variables that can be accessed by all POUs in the project.



	Class	Identifier	FP Address	IEC Address	Type	Initial
0	VAR_GLOBAL	Slave2_g_bY38	Y38	%QX3.8	BOOL	FALSE

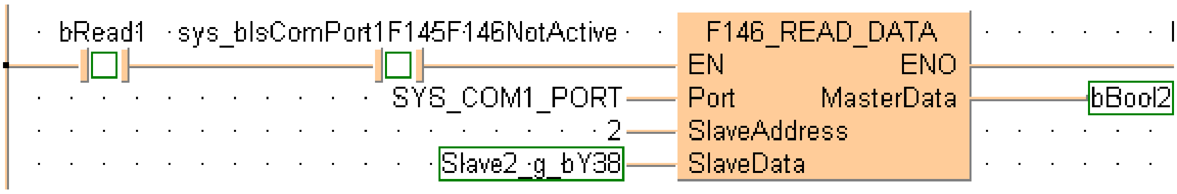
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR_EXTERNAL	Slave2_g_bY38	BOOL	FALSE
1	VAR	bRead1	BOOL	FALSE
2	VAR	bWrite2	BOOL	FALSE
3	VAR	bBool2	BOOL	FALSE

Body If **bRead1** and **sys_bIsComPort1F145F146NotActive** are set to TRUE, the global variable **Slave2_g_bY38**, which is assigned to Y38 of slave 2, is read and stored in **bBool2**.

LD



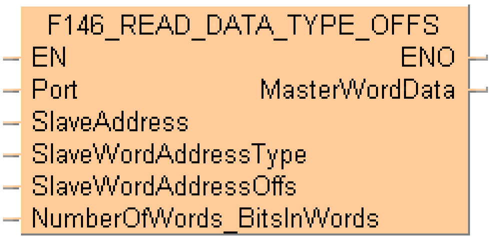
ST When programming with structured text, enter the following:

```
if (bRead1 and sys_bIsComPort1F145F146NotActive) then
            F146_READ_DATA(Port := SYS_COM1_PORT,
                                SlaveAddress := 2,
                                SlaveData := Slave2_g_bY38,
                                MasterData => bBool2);

      bRead1 := false;
      bWrite1 := true;
end_if;
```

F146_READ_DATA_TYPE_OFFS

Read Data from Slave with Type and Offset

Description Use this instruction to request data from a slave via the serial port (COM1 or COM2) using MEWTOCOL or Modbus RTU protocol (see communication mode (see page 712)), as defined in the system register settings (see page 1273) of port used. Master and slave must both use the same protocol. The master must be configured in Master/Slave mode. The slave can be configured either in Master/Slave mode or in Slave mode only.



The data is read from the memory area of the slave specified by **SlaveAddressType** and **SlaveAddressOffs**. It is stored in the area of the master specified by **MasterWordAddress**.

General programming information for F145 and F146 (see page 766)

PLC types **Availability of** F146_READ_DATA_TYPE_OFFS **(see page 1321)**

Data types

Variable	Data type	Function
Port	ANY16	Specifies the slave's COM port (1 or 2) via system variable: SYS_COM1_PORT SYS_COM2_PORT
SlaveAddress		Address of the remote station (1-99).
SlaveWordAddressType		Address type in the slave from which data is read.
SlaveWordAddressOffs		The offset for the starting slave address whose type is defined by **SlaveWordAddressType** and to which the data is written.
NumberOfWords_ BitsInWords		Number of word units to be read by the master (if the highest bit is not set) or bits in word (if the highest bit is set). Is identical to the lower word of **s1_ControlData**.
MasterWordData	ANY	The master data which is written to the slave.

s1_ControlData

	Higher word				Lower word			
Hex	1	0 fixed	1	1	8	5	0	7
	COM port (16#1 or 16#2)		Unit No. (16#00 to 16#63) (0 to 99)		Bit unit transmission	Bit No. of Master (16#0 to 16#F)		Bit No. of Slave (16#0 to 16#F)

- To generate function code 01, bit unit transmission (16#8) must be specified.

s1_ControlData:	16#10118507
s2_SlaveStartAddrType:	WY0
s2_SlaveStartAddrOffs:	1
d_MasterStartAddr:	16#0000

Command conversion

Modbus command

1	Slave address	11
2	Function code (16#01)	01
3	Starting No. (H)	00
4	Starting No. (L)	17
5	No. of coils to read (H)	00
6	No. of coils to read (L)	01
7	CRC16 (H)	DC
8	CRC16 (L)	59

- Starting No. is the first coil to read in the slave (here: Y17)
- The No. of coils to read must be 1

☞

- **The compiler calculates the higher word from Port and SlaveAddress. The higher word is set implicitely.**
- **The lower word is specified by NumberOfWords_BitsInWords.**

Operands

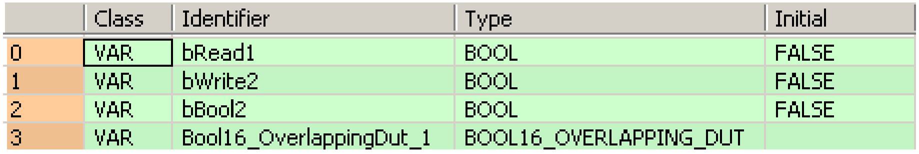
For	Relay				T/C		Register			Constant
Port	WX	WY	WR	WL			DT	LD	FL	-
Slave Address	WX	WY	WR	WL	-	-	DT	LD	FL	dec or hex
SlaveWord Address Type	WX	WY	WR	WL	-	-	DT	LD	FL	-
SlaveWord AddressOffs	WX	WY	WR	WL	-	-	DT	LD	FL	dec or hex
NumberOfWords_ BitsInWords	WX	WY	WR	WL	-	-	DT	LD	FL	dec or hex
Master WordData	WX	WY	WR	WL	-	-	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ port 0 (global transmission) gets no response from COM1 or COM2. ▪ slave data or master data exceeds the available address range ▪ SlaveWordAddressType: Offset ≠ 0 ▪ the communication mode (see page 712) is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave. ▪ the selected COM port requires a communication cassette that has not been installed.
R9008	%MX0.900.8	permanently	

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

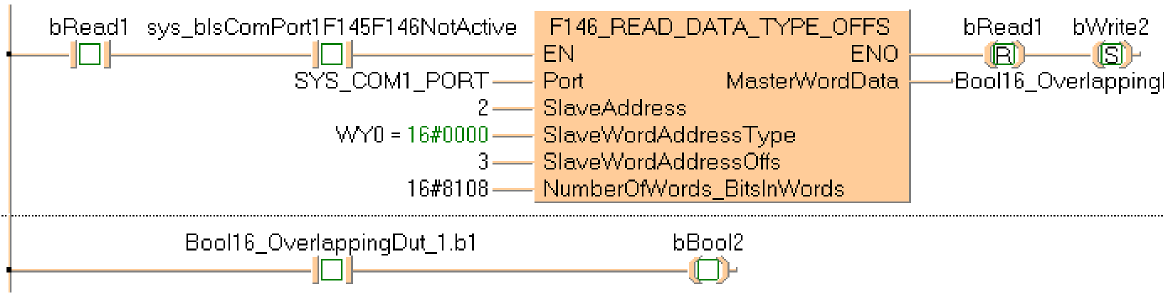
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bRead1	BOOL	FALSE
1	VAR	bWrite2	BOOL	FALSE
2	VAR	bBool2	BOOL	FALSE
3	VAR	Bool16_OverlappingDut_1	BOOL16_OVERLAPPING_DUT	

Body If **bRead1** and **sys_bIsComPort1F145F146NotActive** are set to TRUE, the output Y38 of slave 2 is read and written to bit 1 of **Bool16_OverlappingDut_1.w0.** This bit can be accessed by **Bool16_OverlappingDut_1.b1** and is copied to **bBool2**.

LD



ST

When programming with structured text, enter the following:

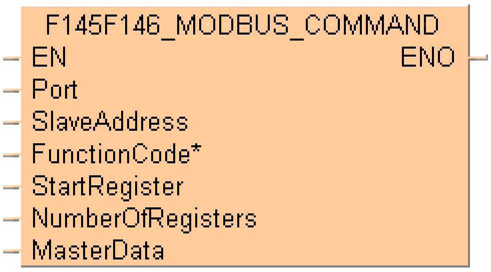
```
if (bRead1 and sys_bIsComPort1F145F146NotActive) then
      F146_READ_DATA_TYPE_OFFS(Port := SYS_COM1_PORT,
                               SlaveAddress := 2,
                               SlaveWordAddressType := WY0,
                               SlaveWordAddressOffs := 3,
                               NumberOfWords_BitsInWords := 16#8108,
                               MasterWordData =>
Bool16_OverlappingDut_1.w0);
end_if;

bBool2 := Bool16_OverlappingDut_1.b1;
```

F145F146_MODBUS _COMMAND

Write data to slave or read data from slave

Description Write data from a master to a slave or read data from a slave via the serial port (COM1 or COM2) depending on the function code. The Modbus RTU protocol (see communication mode (see page 712)) must be set in the system registers (see page 1273). Select "Modbus RTU Master/Slave" for the desired port.

For slave addresses higher than 99 or start register numbers outside the allowed range, use the instruction F145F146_MODBUS_MASTER (see page 779).



In contrast to other F145 or F146 instructions, the required Modbus command can directly be set by the parameter **FunctionCode***.

General programming information for F145 and F146 (see page 766)

Commands supported by the master:

Function code	System constant	Start register	Number of registers	Reference numbers
01	SYS_MODBUS_01_READ_COIL	0–9998	1 or multiple of 16	000001–009999
02	SYS_MODBUS_02_READ_INPUT	0–9998	1	100001–109999
03	SYS_MODBUS_03_READ_HOLDING_REGISTERS	0–32764	≥1	400001–432765
04	SYS_MODBUS_04_READ_INPUT_REGISTERS	0–127 2000–2255	≥1	300001–300128 302001–302256
05	SYS_MODBUS_05_FORCE_COIL	0–9998	1	000001–009999
06	SYS_MODBUS_06_PRESET_REGISTER	0–32764	1	400001–432765
15	SYS_MODBUS_15_FORCE_COILS	0–9998	multiple of 16	000001–009999
16	SYS_MODBUS_16_PRESET_REGISTERS	0–32764	≥1	400001–432765

Modbus specifications for Panasonic PLCs:

Reference numbers	Address area of Panasonic PLCs
From 000001	From Y0
From 002049	From R0
From 100001	From X0
From 400001	From DT0

Reference numbers	Address area of Panasonic PLCs
From 300001	From WL0
From 302001	From LD0

For reference number and address area ranges supported by the Panasonic PLCs, please refer to the User's Manual of the PLC. If the reference number is outside the supported range, an error is returned.

PLC types

Availability of F145F146_MODBUS_COMMAND (see page 1321)

Data types

Variable	Data type	Function
Port	ANY16	Specifies the slave's COM port (1 or 2) via system variable: SYS_COM1_PORT SYS_COM2_PORT
SlaveAddress		Address of the remote station (1-99).
FunctionCode*		SYS_MODBUS_01_READ_COIL SYS_MODBUS_02_READ_INPUT SYS_MODBUS_03_READ_HOLDING_REGISTERS SYS_MODBUS_04_READ_INPUT_REGISTERS SYS_MODBUS_05_FORCE_COIL SYS_MODBUS_06_PRESET_REGISTER SYS_MODBUS_15_FORCE_COILS SYS_MODBUS_16_PRESET_REGISTERS
StartRegister		Starting address. The address type depends on the command specified by **FunctionCode***.
NumberOfRegisters*		Number of transmission bits or words.
MasterData	ANY	The master data which is written to the slave.

Operands

For	Relay				T/C		Register			Constant
Port	WX	WY	WR	WL	-	-	DT	LD	FL	-
Slave Address	WX	WY	WR	WL	-	-	DT	LD	FL	-
Function Code*	-	-	-	-	-	-	-	-	-	system defined
Start Register	WX	WY	WR	WL	-	-	DT	LD	FL	-
NumberOf Registers*	-	-	-	-	-	-	-	-	-	dec. or hex.
Master Data	WX	WY	WR	WL	-	-	DT	LD	FL	-

Error flags

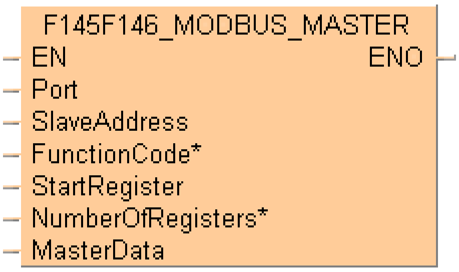
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ port 0 (global transmission) gets no response from COM1 or COM2. ▪ slave data or master data exceeds the available address range. ▪ the communication mode (see page 712) is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave. ▪ the COM port selected requires a communication cassette that has not been installed.
R9008	%MX0.900.8	permanently	

F145F146_MODBUS _MASTER

Write data to slave or read data from slave

Description Write data from a master to a slave or read data from a slave via the serial port (COM1 or COM2) depending on the function code. The Modbus RTU protocol (see communication mode (see page 712)) must be set in the system registers (see page 1273). Select "Modbus RTU Master/Slave" for the desired port.

This instruction is identical to F145F146_MODBUS_COMMAND (see page 777), but it also supports slave addresses higher than 99 and a wider range for the start register.



In contrast to other F145 or F146 instructions, the required Modbus command can directly be set by the parameter **FunctionCode***.

General programming information for F145 and F146 (see page 766)

Commands supported by the master:

Function code	System constant	Start register	Number of registers	Reference numbers (depending on Modbus slave)
01	SYS_MODBUS_01_READ_COIL	0–65535	1–2040	000001–065536
02	SYS_MODBUS_02_READ_INPUT	0–65535	1–2040	100001–165536
03	SYS_MODBUS_03_READ_HOLDING_REGISTER	0–65535	1–127	400001–465536
04	SYS_MODBUS_04_READ_INPUT_REGISTERS	0–65535	1–127	300001–365536
5	SYS_MODBUS_05_FORCE_COIL	0–65535	1	000001–065536
6	SYS_MODBUS_06_PRESET_REGISTER	0–65535	1	400001–465536
15	SYS_MODBUS_15_FORCE_COILS	0–65535	2–2040	000001–065536
16	SYS_MODBUS_16_PRESET_REGISTERS	0–65535	2–127	400001–465536

Modbus specifications for Panasonic PLCs:

Reference numbers	Address area of Panasonic PLCs
From 000001	From Y0
From 002049	From R0
From 100001	From X0
From 400001	From DT0
From 300001	From WL0
From 302001	From LD0

For reference number and address area ranges supported by the Panasonic PLCs, please refer to the User's Manual of the PLC. If the reference number is outside the supported range, an error is returned.

PLC types: Availability of F145F146_MODBUS_MASTER **(see page 1321)**

Data types

Variable	Data type	Function
Port	ANY16	Specifies the slave's COM port (1 or 2) via system variable: SYS_COM1_PORT, SYS_COM2_PORT
SlaveAddress		Address of the remote station (0–255).
FunctionCode*		SYS_MODBUS_01_READ_COIL SYS_MODBUS_02_READ_INPUT SYS_MODBUS_03_READ_HOLDING_REGISTER SYS_MODBUS_04_READ_INPUT_REGISTERS SYS_MODBUS_05_FORCE_COIL SYS_MODBUS_06_PRESET_REGISTER SYS_MODBUS_15_FORCE_COILS SYS_MODBUS_16_PRESET_REGISTERS
StartRegister		Starting address (0–65535). The address type depends on the command specified by **FunctionCode***.
NumberOfRegisters*		Number of transmission bits or words. 1–2040 for function codes 01, 02 2–2040 for function code 15 1–127 for function codes 03, 04 2–127 for function code 16
MasterData	ANY	The master data which is written to the slave.

Operands

For	Relay				T/C		Register			Constant
Port	WX	WY	WR	WL			DT	LD	FL	-
Slave Address	WX	WY	WR	WL	-	-	DT	LD	FL	-
Function Code*	-	-	-	-	-	-	-	-	-	system
Start Register	WX	WY	WR	WL	-	-	DT	LD	FL	-
NumberOf Registers*	-	-	-	-	-	-	-	-	-	dec. or hex.
Master Data	WX	WY	WR	WL	-	-	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ port 0 (global transmission) gets no response from COM1 or COM2. ▪ slave data or master data exceeds the available address range. ▪ the communication mode (see page 712) is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave. ▪ the COM port selected requires a communication cassette that has not been installed.
R9008	%MX0.900.8	permanently	

22.5.1 Evaluation of IsF145146NotActive flag

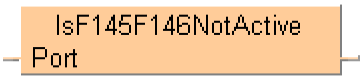
In this section:

- Is145F146NotActive (see page 787)

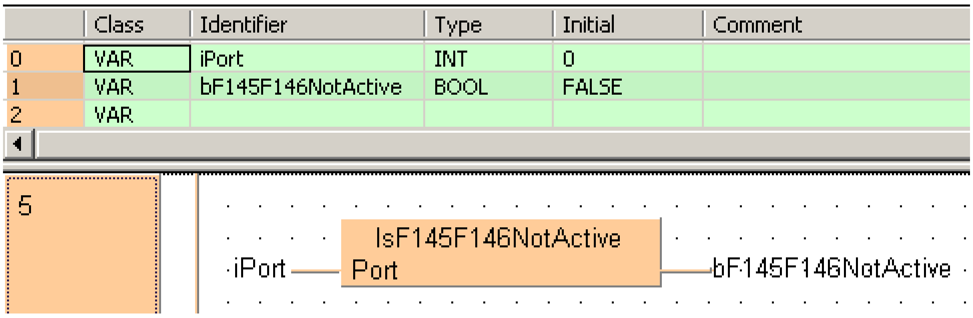
Is145F146NotActive

for all ports via a general function

Description This instruction returns the value of the "F145F146 Not Active" flag of the PLC's serial communication interface.



Example



	Class	Identifier	Type	Initial	Comment
0	VAR	iPort	INT	0	
1	VAR	bF145F146NotActive	BOOL	FALSE	
2	VAR				

This flag varies depending on the PLC type:

PLC	Port number	Port name	Flag	System variable
FP-Sigma, FP-X	0	TOOL port (not for FP-Sigma 12k)	returns always TRUE	-
	1	COM1 port	R9044	sys_bIsComPort1F145F146_ NotActive
	2	COM2 port	R904A	sys_bIsComPort2F145F146_ NotActive
FP0, FP-e	-	-	returns always TRUE	-
FP2, FP2SH	0	CPU COM port	returns always TRUE	-
	16#xx01	MCU COM1 port of MCU unit in slot xx	returns always TRUE	-
	16#xx02	MCU COM2 port of MCU unit in slot xx		

For detailed information on using system variables, please refer to data transfer to and from special data registers (see page 859).

IsReceptionDone for a special COM port via the corresponding System Variable

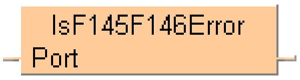
You can use the following system variables to evaluate the IsF145F146NotActive flag for a special COM port:

- sys_bIsComPort1F145F146NotActive
- sys_bIsComPort2F145F146NotActive

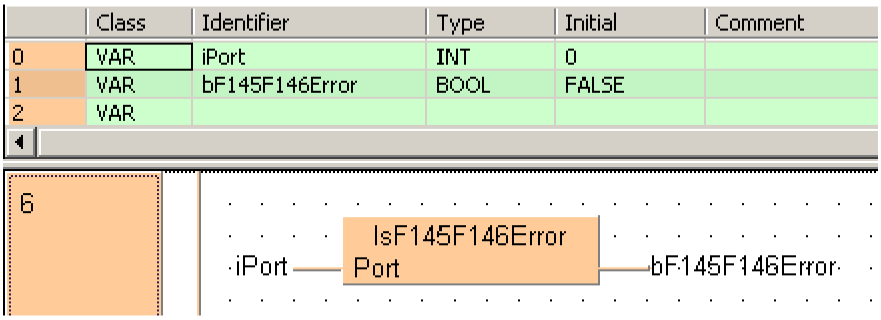
IsF145F146Error

Returns the value of the "F145F146 Error" flag

Description This instruction returns the value of the "F145F146 Error" flag of the PLC's serial communication interface.



Example



	Class	Identifier	Type	Initial	Comment
0	VAR	iPort	INT	0	
1	VAR	bF145F146Error	BOOL	FALSE	
2	VAR				

This flag varies depending on the PLC type:

PLC	Port number	Port name	Flag	System variable
FP-Sigma, FP-X	0	TOOL port (not for FP-Sigma 12k)	returns always FALSE	-
	1	COM1 port	R9045	sys_bIsComPort1F145F146Error
	2	COM2 port	R904B	sys_bIsComPort2F145F146Error
FP0R	0	TOOL port	returns always FALSE	-
	1	COM1 port	R9045	sys_bIsComPort1F145F146Error
FP0, FP-e	-	-	returns always FALSE	-
FP2, FP2SH	0	CPU COM port	returns always FALSE	-
	16#xx01	MCU COM1 port of MCU unit in slot xx	returns always FALSE	-
	16#xx02	MCU COM2 port of MCU unit in slot xx		

For detailed information on using system variables, please refer to data transfer to and from special data registers (see page 859).

Chapter 23

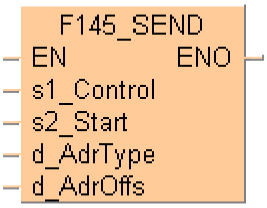
Data transfer via network

23.1 Data transfer via MEWNET link

In this section:

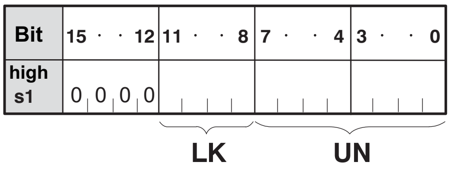
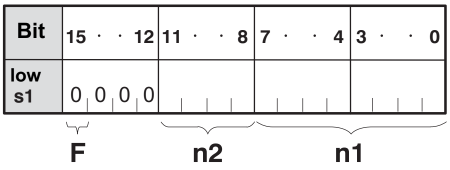
F145_SEND Data send (MEWNET link)

Description Sends data to another station through link modules in the network.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Specifications of s1:

s1 higher byte	s1 lower byte	
	 Bit 15 12 11 8 7 4 3 0 low s1 0 0 0 0 F n2 n1	
1. LINK No. selection (LK: 1 to 3, the station itself)	**1. Word unit send selection**	
Up to 3 Link Units can be connected to 1 CPU.	F = 0	Word unit selection
	n2 = 0	Set "0" when the word unit is selected
This (LK) selects the source Link Unit of the three.	n1 = 11–16	Specify the number of words to be sent
2. Link station No. selection (UN: 1 to 63, another station)	**2. Bit unit send selection**	
Up to 63 stations can be connected to 1 Link Unit.	F = 1	Bit unit selection
This (UN) then selects the destination station No.	n2 = 0–15	Destination bit No.
	n1 = 0–15	Source bit No.

REFERENCE

For detailed information, please refer to the relevant technical manual of the intelligent unit.

PLC types **Availability of F145_SEND (see page 1321)**

Data types

Variable	Data type	Function
s1	DWORD	32-bit area for storing control data
s2	ANY16	starting 16-bit area for storing source data (data area at the source station)
d		type of destination operands for storing data in the destination station. Be sure to select the area by setting the address 0 (e.g. DT0 or WR0, ...) (destination data area at another station)
n*		starting 16-bit area address for the destination operand specified in **d** (destination data area in another station) Must be a constant

The variables **s2** and **d** have to be the same data type.

Operands

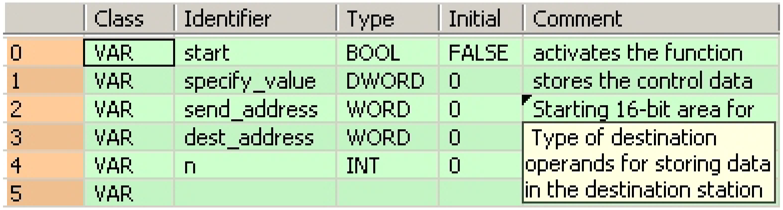
For	Relay				T/C		Register			Constant
s1	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
d	-	WY	WR	WL	SV	EV	DT	LD	FL	
n*	-	-	-	-	-	-	-	-	-	dec. or hex.

Example

In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header

All input and output variables used for programming this function have been declared in the POU header.

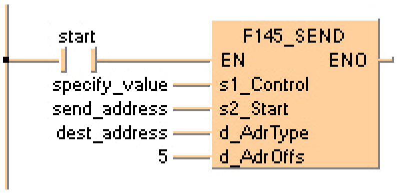


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	specify_value	DWORD	0	stores the control data
2	VAR	send_address	WORD	0	Starting 16-bit area for
3	VAR	dest_address	WORD	0	Type of destination operands for storing data in the destination station
4	VAR	n	INT	0	
5	VAR				

Body

When the variable **start** is set to TRUE, the function is carried out.

LD



ST

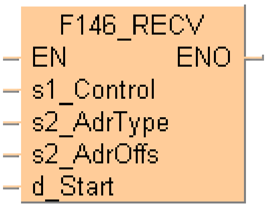
When programming with structured text, enter the following:

```
IF start THEN
    F145_SEND( specify_value, send_address, dest_address, 5);
END_IF;
```

F146_RECV

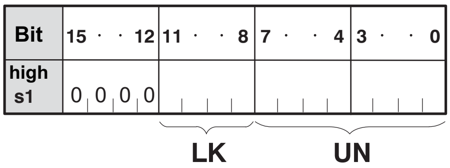
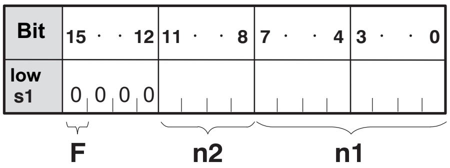
Data receive (MEWNET link)

Description Receives data from another station through link units in the network.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Specifications of s1:

s1 higher byte	**s1 lower byte**	
 Bit 15 · · 12 11 · · 8 7 · · 4 3 · · 0 high s1 0 0 0 0 LK UN	 Bit 15 · · 12 11 · · 8 7 · · 4 3 · · 0 low s1 0 0 0 0 F n2 n1	
1. LINK No. selection (LK: 1 to 3, the station itself)	**1. Word unit send selection**	
Up to 3 Link Units can be connected to 1 CPU.	F = 0	Word unit selection
	n2 = 0	Set "0" when the word unit is selected
This (LK) selects the source Link Unit of the three.	n1 = 11–16	Specify the number of words to be sent
2. Link station No. selection (UN: 1 to 63, another station)	**2. Bit unit send selection**	
Up to 63 stations can be connected to 1 Link Unit.	F = 1	Bit unit selection
This (UN) then selects the destination station No.	n2 = 0–15	Destination bit No.
	n1 = 0–15	Source bit No.

◆REFERENCE

For detailed information, please refer to the relevant technical manual of the intelligent unit.

PLC types **Availability of** F146_RECV **(see page 1321)**

Data types

Variable	Data type	Function
s1	DWORD	32-bit area for storing control data
s2	ANY16	type of source operands for storing data in the destination station. Be sure to select the area by setting the address 0 (e.g. DT0 or WR0, ...) (source data area at another station)
d		starting 16-bit area address for the source operand specified in **s2** (source data area at another station)
n*		starting 16-bit area address for storing data received (destination data area at source station) Must be a constant

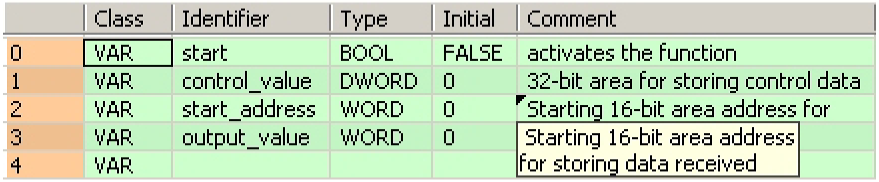
The variables **s2** and **d** have to be the same data type.

Operands

For	Relay				T/C		Register			Constant
s1	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
d	-	WY	WR	WL	SV	EV	DT	LD	FL	
n*	-	-	-	-	-	-	-	-	-	dec. or hex.

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

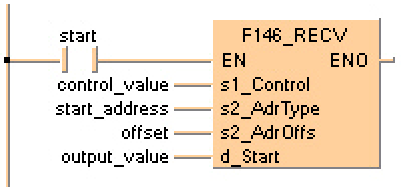
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	control_value	DWORD	0	32-bit area for storing control data
2	VAR	start_address	WORD	0	Starting 16-bit area address for
3	VAR	output_value	WORD	0	Starting 16-bit area address for storing data received
4	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



ST When programming with structured text, enter the following:

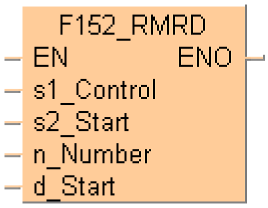
```
IF start THEN
    F146_RECV( s1_Control:= control_value, s2_AdrType:= start_address,
n_AdrOffs:= offset,
            d_Start:= output_value);
END_IF;
```

23.2 Data transfer via shared memory of a MEWNET-F-Slave station

In this section:

F152_RMRD Data read from the slave station

Description Reads data from the specified intelligent unit of the MEWNET-F Slave station.



s1 stores the control data for the configuration of the Master and Slave units in the network. **n** words are read beginning from the shared memory address number in the intelligent unit specified by **s2**. The result is stored in **d**.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Specifications of s1:

s1 higher word

Bit	15 · · 12	11 · · 8	7 · · 4	3 · · 0
s1 high word				

Bank No. (16#00 to 16#FF if there is a bank to specify, otherwise 16#00)

Slot No. (16#00 to 16#1F, FP3: to 16#17)

s1 lower word

Bit	15 · · 12	11 · · 8	7 · · 4	3 · · 0
s1 low word				

Master station No. (16#01 to 16#04)

Slave station No. (16#01 to 16#20)

Reference: Intelligent unit with bank

Name	Order Number
FP3 expansion data memory unit	AFP32091 AFP32092

PLC types **Availability of** F152_RMRD **(see page** 1321**)**

Data types

Variable	Data type	Function
s1	DWORD	stores control data for Master/Slave configuration
s2	ANY16	starting memory address number of words to be read
n	INT	number of words to be read (max. 32 words)
d	ANY16	starting 16-bit area where words read are stored, (see F153 (see page 794))

The variables **s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
s2, n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the control data **s1** exceeds the limit of specified range ▪ no MEWNET-F master unit is found ▪ the data read exceeds the area of **d**
R9008	%MX0.900.8	for an instant	

Precautions during programming

It is not possible to execute multiple **F152_RMRD** instructions and **F153_RMWT** instructions at the same time.
The program should be set up so that these instructions are executed when the **F152_RMRD/F153_RMWT** instruction execution enabled flag (R9035) is on.

R9035	0: Execution inhibited (RMRD/RMWT instruction being executed)
	1: Execution enabled

The **F152_RMRD** instruction only enables a request to be accepted. The actual processing is carried out at the end of the scan. The **F152_RMRD/F153_RMWT** instruction completed flag (R9036) can be used to confirm whether or not the instruction has been executed.

R9036	0: Completed normally
	1: Completed with error (The error code is stored in DT9036/DT90036)
DT9036 (DT90036)	If the transmission has been completed with an error (R9036 is on), the contents of the error (error code) are stored.

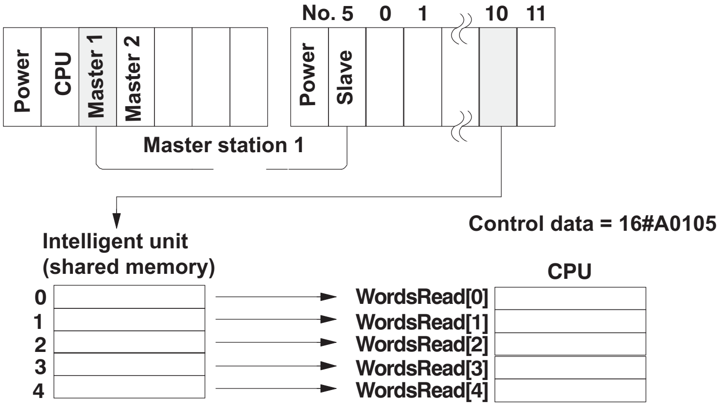
Reference: The error codes stored in the DT9036/DT90036

Error code (HEX)	Description
16#5B	Time-out error (no intelligent unit found at the specified location)
16#68	No memory error (no memory exists at the specified address)
16#71	Send answer time-out error
16#72	Send buffer full time-out error
16#73	Response time-out error

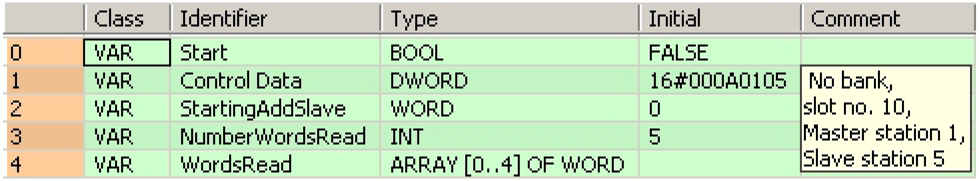
If the error code is 16#71 to 16#73, a communication time-out error has occurred. The time-out time can be changed within a range of 10.0ms to 81.9s (in units of 10ms), using the setting of system register 32. The default value is set to 2 seconds.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

Five words of data stored at address 0 to 4 in the shared memory of the intelligent unit of the slave station are read and the read data stored in ARRAY **WordsRead** of the master station "CPU" when **Start** turns on.

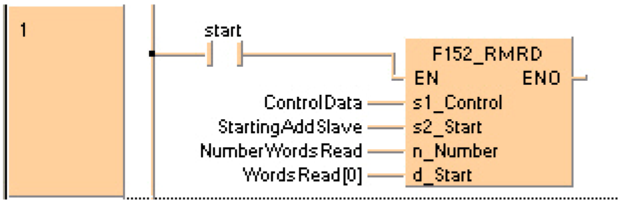


POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	Control Data	DWORD	16#000A0105	No bank, slot no. 10, Master station 1, Slave station 5
2	VAR	StartingAddSlave	WORD	0	
3	VAR	NumberWordsRead	INT	5	
4	VAR	WordsRead	ARRAY [0..4] OF WORD		

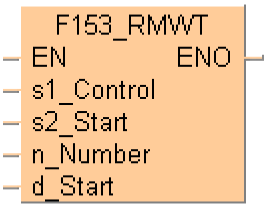
LD



F153_RMWT

Data write into the slave station

Description Writes data into the specified intelligent unit of the MEWNET-F slave station.



s1 stores the control data for the configuration of the Master and Slave units in the network. **n** words, beginning at the address in the CPU specified by **s2**, are written to the intelligent unit of the Slave unit beginning at the shared memory address number specified by **d**.

Specifications of s1:

s1 higher word

Bit	15 · · 12	11 · · 8	7 · · 4	3 · · 0
s1 high word				

Bank No. (16#00 to 16#FF if there is a bank to specify, otherwise 16#00)

Slot No. (16#00 to 16#1F, FP3: to 16#17)

s1 lower word

Bit	15 · · 12	11 · · 8	7 · · 4	3 · · 0
s1 low word				

Master station No. (16#01 to 16#04)

Slave station No. (16#01 to 16#20)

Reference: Intelligent unit with bank

Name	Order Number
FP3 expansion data memory unit	AFP32091 AFP32092

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F153_RMWT **(see page 1321)**

Data types

Variable	Data type	Function
s1	DWORD	stores control data for Master/Slave configuration
s2	ANY16	starting 16-bit area in CPU where words are read
n	INT	number of words to be read and then written to the Slave unit (max. 32 words)
d	ANY16	starting memory address number in the intelligent unit where words are written

The variables **s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
n, d	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Error flags

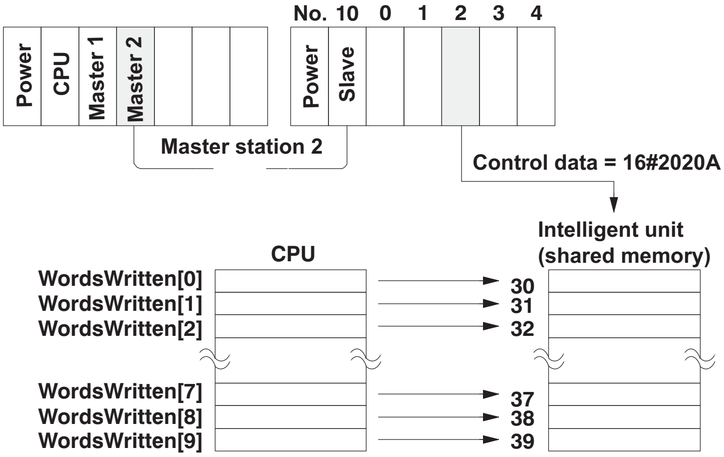
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the control data **s1** exceeds the limit of specified range ▪ no MEWNET-F master unit is found ▪ the data read exceeds the area of **s2**
R9008	%MX0.900.8	for an instant	

Precautions during programming: see **F152_RMRD** (see page 791)

Example

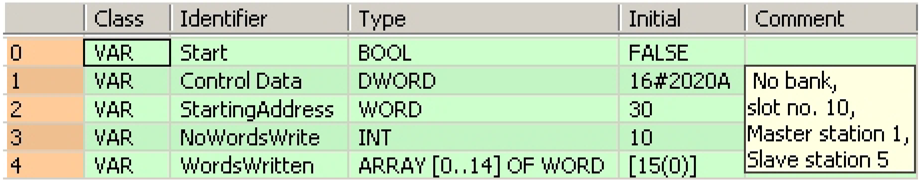
In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

Twenty words of data stored in the ARRAY **WordsWritten[0]..[9]** of the master station "CPU" are written into the shared memory of the intelligent unit of slave station starting from address 30 to 39 when **Start** turns on.



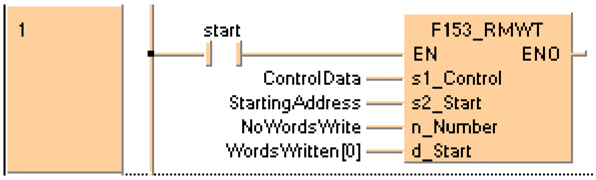
POU header

All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	Control Data	DWORD	16#2020A	No bank, slot no. 10, Master station 1, Slave station 5
2	VAR	StartingAddress	WORD	30	
3	VAR	NoWordsWrite	INT	10	
4	VAR	WordsWritten	ARRAY [0..14] OF WORD	[15(0)]	

LD


1
start
F153_RMWT
EN
ENO
ControlData
s1_Control
StartingAddress
s2_Start
NoWordsWrite
n_Number
WordsWritten[0]
d_Start

23.3 Data exchange with flexible network

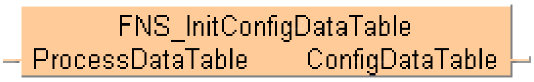
In this section:

FNS_InitConfigDataTable

Function

Description

The FNS_InitConfigDataTable function creates a ConfigDataTable from the variable **ProcessDataTable**, which can be a single-element data type or a mulit-element data type. This ConfigDataTable is necessary to configure the FP-FNS block using the function block FNS_ProfibusDP, FNS_DeviceNet, FNS_CANopen and FNS_ProfinetIO.



☞

Make sure that the size of the variable ConfigDataTable corresponds to the structure of the ProcessDataTable, e.g. if the ProcessDataTable consists of three entries, then the ConfigDataTable variable should be an "Array[0..2] of WORD", whose size matches the number of entries. If the ProcessDataTable variable has only one entry (e.g. WORD), then the ConfigDataTable variable should be an "Array[0..0] of WORD" (with size 1).

Allowed data types for the input of the FNS_InitConfigDataTable are all 16-bit (INT, WORD), 32-bit (DINT, DWORD, TIME (32 bits), REAL) and 64-bit variables or arrays of them. 64-bit variables are defined as 2-dimensional arrays, e.g. "Array[0..0,0..3] of INT" is a 64-bit variable, while "Array[0..3] of INT" represents an array with four elements of 16-bit variables.

The data types BOOL, STRING and arrays of these types are NOT allowed at the input of the function FNS_InitConfigDataTable.

The output ConfigDataTable of the function must be an array of WORD.

PLC types **Availability of** FNS_InitConfigDataTable **(see page 1326)**

Data types

Variable	Data types	Function
ProcessDataTable	INT, WORD, DINT, DWORD, REAL, TIME, and ARRAYS of these types	Input and output of process data variables
ConfigDataTable	ARRAY of WORD	Configuration data for FP-FNS blocks. The array-size of the variable ConfigDataTable has to correspond to the number of elements of the ProcessDataTable variable.

ProcessData Table

The following syntax table shows how to declare 16-bit, 32-bit and 64-bit variables and arrays thereof when using them as ProcessDataTable input for the FNS_InitConfigDataTable function.

Input Data type	Size of Input	Comment
INT, WORD	16-bit	
DINT, WORD, REAL, TIME	32-bit	
Array[0..0,0..3] of INT **Array[0..0,0..3] of WORD**	64-bit	2-dimensional array; size of second dimension = 4
Array[a ..b] of INT/Array[a ..b] of WORD	Array of 16-bit Size = b-a+1	1-dimensional array
Array[a ..b] of DINT/Array[a ..b] of DWORD/ **Array[a ..b] of REAL/Array[a ..b] of TIME**	Array of 32-bit Size = b-a+1	1-dimensional array
Array[0..x,0..3] of INT **Array[0..x,0..3] of WORD**	Array of 64-bit Size = x+1	2-dimensional array; size of second dimension = 4

Operands

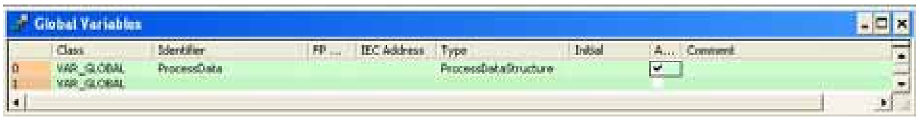
For	Relay				T/C		Register			Constant
ProcessDataTable	-	-	-	-	-	-	DT	FL	-	-
ConfigDataTable	-	-	-	-	-	-	DT	FL	-	-

Example

In this example, the same POU header is used for all programming languages.

GVL

In the global variable list, you define variables that can be accessed by all POUs in the project.

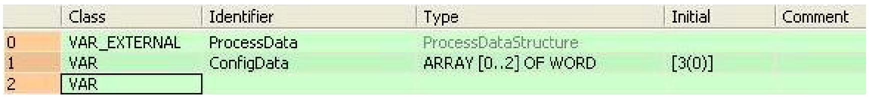


Global Variables

	Class	Identifier	FP ...	IEC Address	Type	Initial	A...	Comment
0	VAR_GLOBAL	ProcessData			ProcessDataStructure		☑	
1	VAR_GLOBAL							

POU header

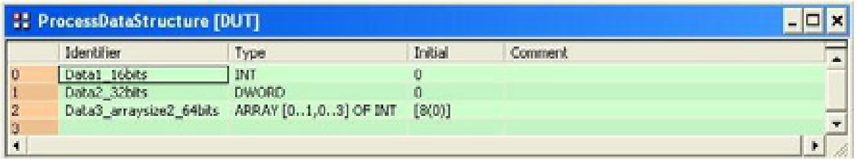
All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	ProcessData	ProcessDataStructure		
1	VAR	ConfigData	ARRAY [0..2] OF WORD	[3(0)]	
2	VAR				

The size of the variable **ConfigDataTable** has to correspond to the number of entries of the input variable **ProcessData**.

In this example, the variable **ProcessData** is a DUT of the type ProcessDataStructure with the following structure:



ProcessDataStructure [DUT]

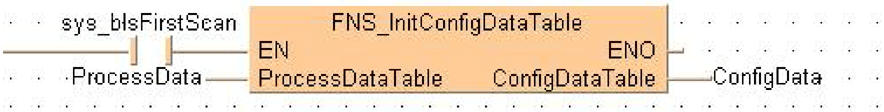
	Identifier	Type	Initial	Comment
0	Data1_16bits	INT	0	
1	Data2_32bits	DWORD	0	
2	Data3_arraysize2_64bits	ARRAY [0..1,0..3] OF INT	[8(0)]	
3				

As the DUT has three entries, the output variable **ConfigData** has to be an array of WORD with a size of three (e.g.: Array [0..2] of WORD).

Body

When **sys_bIsFirstScan** is TRUE, i.e. in the first cycle, the function is executed. The value of the variable **ConfigData** corresponds to the structure of the input variable **ProcessData**, its number and type of elements.

LD



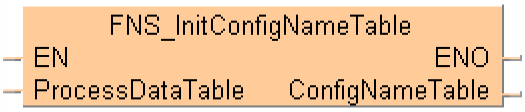
ST When programming with structured text, enter the following:

```
If sys_bIsFirstScan then
      ConfigData:=FNS_InitConfigDataTable(ProcessData);
end_if;
```

FNS_InitConfigNameTable

Function

Description This function creates a ConfigNameTable from the variable **ProcessDataTable**, which can be a single-element data type or a mulit-element data type.



☞ **Make sure that the size of the variable ConfigNameTable corresponds to the structure of the ProcessDataTable, e.g. if the ProcessDataTable consists of three entries, then the ConfigNameTable variable should be an "Array[0..2] of WORD" whose size matches the number of entries. If the ProcessDataTable variable has only one entry (e.g. WORD), then the ConfigNameTable variable should be an "Array[0..0] of WORD" (with size 1).**

Allowed input data types are all 16-bit (INT, WORD), 32-bit (DINT, DWORD, TIME (32 bits), REAL) and 64-bit variables or arrays of them. 64-bit variables are defined as 2-dimensional arrays, e.g. "Array[0..0,0..3] of INT" is a 64-bit variable, while "Array[0..3] of INT" represents an array with four elements of 16-bit variables.

The data types BOOL, STRING and arrays of these types are NOT allowed at the input variable.

The output ConfigNameTable of the function must be an array of WORD.

PLC types see page 1326

Data types

Variable	Data types	Function
ProcessDataTable	INT, WORD, DINT, DWORD, REAL, TIME, and ARRAYS of these types	Input and output of process data variables
ConfigNameTable	ARRAY of WORD	Configuration data for FP-FNS blocks. The array-size of the variable ConfigNameTable has to correspond to the number of elements of the ProcessDataTable variable.

ProcessData Table The following syntax table shows how to declare 16-bit, 32-bit and 64-bit variables and arrays thereof when using them as ProcessDataTable input.

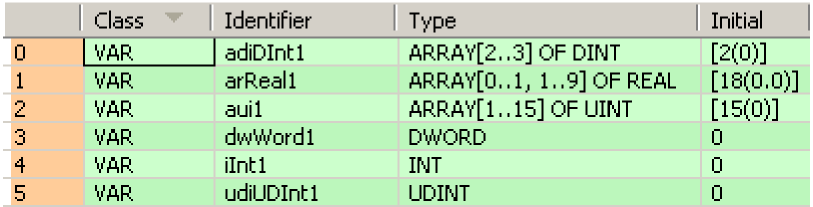
Input Data type	Size of Input	Comment
INT, WORD	16-bit	
DINT, WORD, REAL, TIME	32-bit	
Array[0..0,0..3] of INT **Array[0..0,0..3] of WORD**	64-bit	2-dimensional array; size of second dimension = 4
Array[a ..b] of INT/Array[a ..b] of WORD	Array of 16-bit Size = b-a+1	1-dimensional array
Array[a ..b] of DINT/Array[a ..b] of DWORD/ **Array[a ..b] of REAL/Array[a ..b] of TIME**	Array of 32-bit Size = b-a+1	1-dimensional array
Array[0..x,0..3] of INT **Array[0..x,0..3] of WORD**	Array of 64-bit Size = x+1	2-dimensional array; size of second dimension = 4

Operands

For	Relay				T/C		Register			Constant
ProcessDataTable	-	-	-	-	-	-	DT	FL	-	-
ConfigNameTable	-	-	-	-	-	-	DT	FL	-	-

Example In this example the function is programmed in ladder diagram (LD).

POU header All input and output variables used for programming this function have been declared in the POU header.



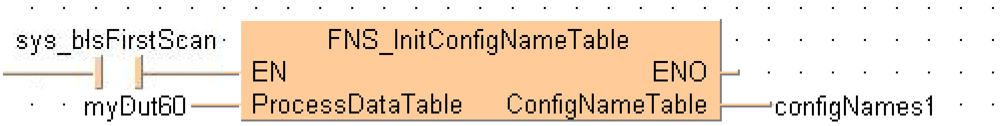
	Class	Identifier	Type	Initial
0	VAR	adiDInt1	ARRAY[2..3] OF DINT	[2(0)]
1	VAR	arReal1	ARRAY[0..1, 1..9] OF REAL	[18(0.0)]
2	VAR	aui1	ARRAY[1..15] OF UINT	[15(0)]
3	VAR	dwWord1	DWORD	0
4	VAR	iInt1	INT	0
5	VAR	udiUDInt1	UDINT	0

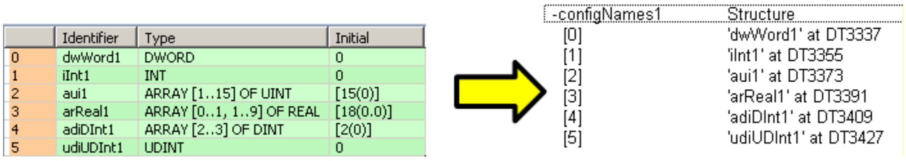
The size of the variable **configNames1** has to correspond to the number of entries of the input variable **myDUT60**.

As the DUT has three entries, the output variable **configNames1** has to be an array of WORD with a size of three (e.g.: Array [0..2] of WORD).

Body When **sys_bIsFirstScan** is TRUE, i.e. in the first cycle, the function is executed. The value of the variable **configNames1** corresponds to the structure of the input variable **myDUT60**, its number and type of elements.

LD





	Identifier	Type	Initial
0	dwWord1	DWORD	0
1	iInt1	INT	0
2	aui1	ARRAY [1..15] OF UINT	[15(0)]
3	arReal1	ARRAY [0..1, 1..9] OF REAL	[18(0.0)]
4	adiDInt1	ARRAY [2..3] OF DINT	[2(0)]
5	udiUDInt1	UDINT	0

```
-configNames1     Structure
  [0]             'dwWord1' at DT3337
  [1]             'iInt1' at DT3355
  [2]             'aui1' at DT3373
  [3]             'arReal1' at DT3391
  [4]             'adiDInt1' at DT3409
  [5]             'udiUDInt1' at DT3427
```

ST When programming with structured text, enter the following:

```
If sys_bIsFirstScan then
      configNames1:=FNS_InitConfigNameTable(myDUT60);
end_if;
```

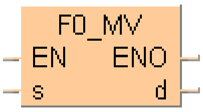
Chapter 24

Data transfer within the PLC

F0_MV

16-bit data move

Description The 16-bit data or 16-bit equivalent constant specified by **s** is copied to the 16-bit area specified by **d**, if the trigger **EN** is in the ON-state.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction MOVE (see page 59). Please refer also to Advantages of the IEC instructions in the online help.

PLC types **Availability of** F0_MV **(see page 1320)**

Data types

Variable	Data type	Function
s	ANY16	source 16-bit area
d		destination 16-bit area

The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Explanation with example value 16#0089

source

bit	15	.	.	12	11	.	.	8	7	.	.	4	3	.	.	0
s	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1

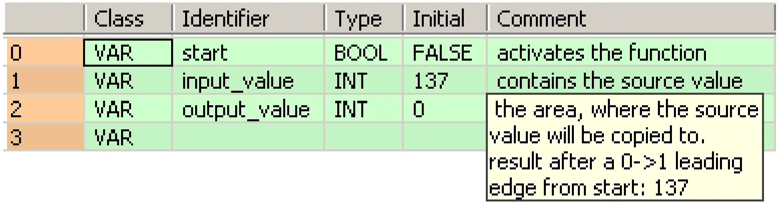
destination

bit	15	.	.	12	11	.	.	8	7	.	.	4	3	.	.	0
d	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1

Destination value in this example: 16#0089

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

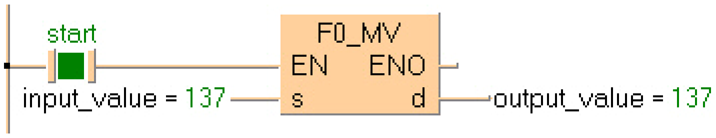
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	INT	137	contains the source value
2	VAR	output_value	INT	0	the area, where the source value will be copied to. result after a 0->1 leading edge from start: 137
3	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



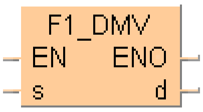
ST When programming with structured text, enter the following:

```
IF start THEN
    F0_MV(input_value, output_value);
END_IF;
```

F1_DMV

32-bit data move

Description The 32-bit data or 32-bit equivalent constant specified by **s** is copied to the 32-bit area specified by **d**, if the trigger **EN** is in the ON-state.



Instead of using this FP instruction, we recommend using the related IEC instruction MOVE (see page 59). Please refer also to Advantages of the IEC instructions in the online help.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F1_DMV **(see page 1320)**

Data types

Variable	Data type	Function
s	ANY32	source 32-bit area
d		destination 32-bit area

The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Explanation with example value 16#ACAEE486

source

bit	31 . . 28	27 . . 24	23 . . 20	19 . .16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	1 0 1 0	1 1 0 0	1 0 1 0	1 1 1 0	1 1 1 0	0 1 0 0	1 0 0 0	0 1 1 0

← 32-bit area →

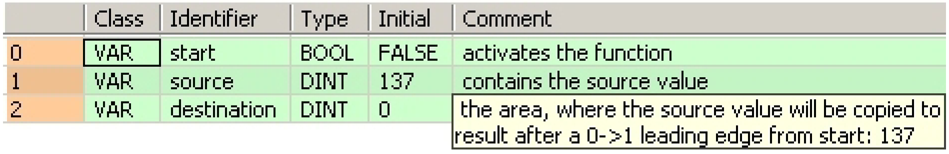
destination

bit	31 . . 28	27 . . 24	23 . . 20	19 . .16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	1 0 1 0	1 1 0 0	1 0 1 0	1 1 1 0	1 1 1 0	0 1 0 0	1 0 0 0	0 1 1 0

Destination value in this example: 16#ACAEE486

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

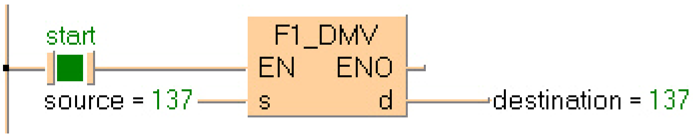
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source	DINT	137	contains the source value
2	VAR	destination	DINT	0	the area, where the source value will be copied to result after a 0->1 leading edge from start: 137

Body When the variable **start** is set to TRUE, the function is carried out.

LD



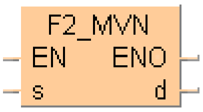
ST When programming with structured text, enter the following:

```
IF start THEN
    F1_DMV(source, destination);
END_IF;
```

F2_MVN

16-bit data inversion and move

Description The 16-bit data or 16-bit equivalent constant specified by **s** is inverted and transferred to the 16-bit area specified by **d** if the trigger **EN** is in the ON-state.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F2_MVN **(see page 1322)**

Data types

Variable	Data type	Function
s	ANY16	source 16-bit area to be inverted
d		destination 16-bit area

The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Explanation with example value 16#5555

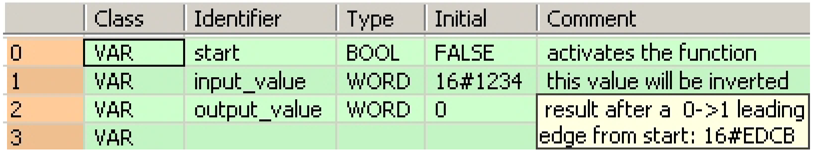
source	15	.	.	12	11	.	.	8	7	.	.	4	3	.	.	0
bit	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

dest.	15	.	.	12	11	.	.	8	7	.	.	4	3	.	.	0
bit	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Each bit is inverted, target value in this example: 16#AAAA

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

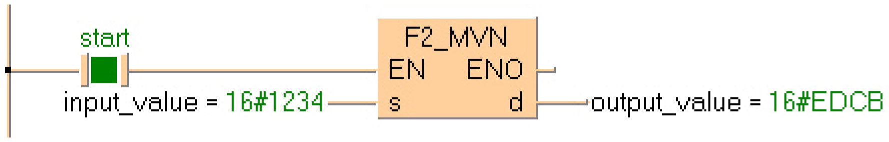
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	16#1234	this value will be inverted
2	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 16#EDCB
3	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



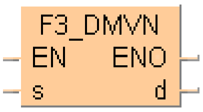
ST When programming with structured text, enter the following:

```
IF start THEN
    F2_MVN(input_value, output_value);
END_IF;
```

F3_DMVN

32-bit data inversion and move

Description The 32-bit data or 32-bit equivalent constant specified by **s** is inverted and transferred to the 32-bit area specified by **d** if the trigger **EN** is in the ON-state.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of F3_DMVN (see page 1323)**

Data types

Variable	Data type	Function
s	ANY32	source 32-bit area to be inverted
d		destination 32-bit area

The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Explanation with example value 16#75BCD15

source

bit	31 . . 28	27 . . 24	23 . . 20	19 . .16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
s	0 0 0 0	0 1 1 1	0 1 0 1	1 0 1 1	1 1 0 0	1 1 0 1	0 0 0 1	0 1 0 1

32-bit area

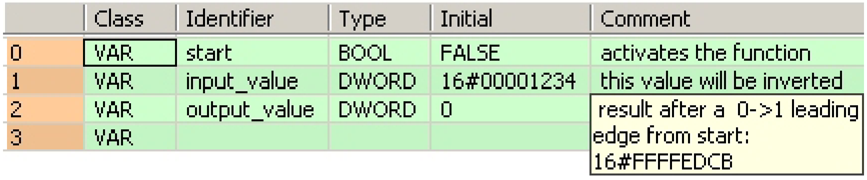
destination

bit	31 . . 28	27 . . 24	23 . . 20	19 . .16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d	1 1 1 1	1 0 0 0	1 0 1 0	0 1 0 0	0 0 1 1	0 0 1 0	1 1 1 0	1 0 1 0

Each bit is inverted, destination value in this example: 16#F8A432EA

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

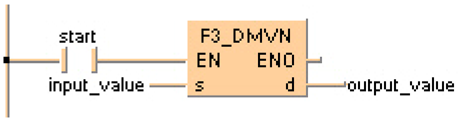
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	DWORD	16#00001234	this value will be inverted
2	VAR	output_value	DWORD	0	result after a 0->1 leading edge from start: 16#FFFFEDCB
3	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



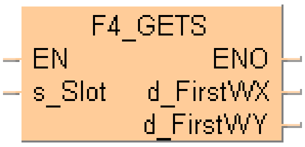
ST When programming with structured text, enter the following:

```
IF start THEN
    F3_DMVN(input_value, output_value);
END_IF;
```

F4_GETS

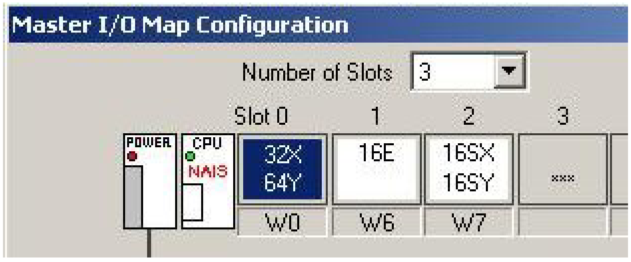
Reading of the Numbers of the First WX and the First WY of the Specified Slot

Description The head word No. of the specified slot is read.



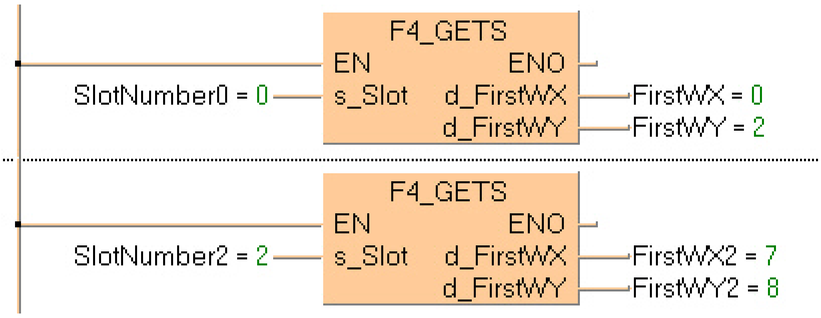
This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Double-click the I/O map configuration in the project navigator for the settings required.


Master I/O Map Configuration

Number of Slots 3

Example



PLC types **Availability of F4_GETS (see page 1325)**

Data types

Variable	Data type	Function
s_Slot	ANY16	Slot number from which the information is required
d_FirstWX		Number of the first WX of the specified slot
d_FirstWY		Number of the first WY of the specified slot

Operands

For	Relay				T/C		Register			Constant
s_Slot	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d_FirstWX	WX	WY	WR	WL	SV	EV	DT	LD	FL	
d_FirstWY	-	WY	WR	WL	SV	EV	DT	LD	FL	-

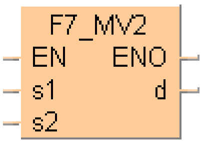
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the specified address using the index modifier exceeds a limit.
R9008	%MX0.900.8	for an instant	▪ a number other than 0 to 31 is specified for the slot number.

F7_MV2

Two 16-bit data move

Description The two 16-bit data or two 16-bit equivalent constants specified by **s1** and **s2** are copied to the 32-bit area specified by **d** when the trigger turns ON.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F7_MV2 **(see page** 1326**)**

☞ **To transfer three 16-bit data types, use either the F190_MV3 (see page 853) or P190_MV3 instruction.**

Data types

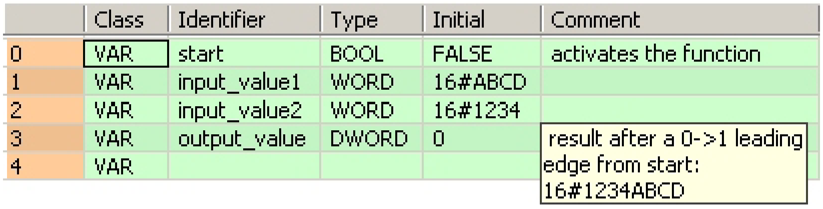
Variable	Data type	Function
s1, s2	ANY16	source 16-bit area
d	ANY32	destination 32-bit area

Operands

For	Relay				T/C		Register			Constant
s1, s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

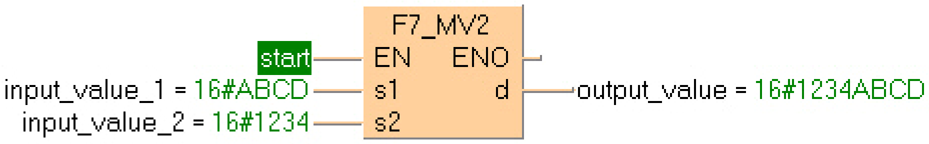


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value1	WORD	16#ABCD	
2	VAR	input_value2	WORD	16#1234	
3	VAR	output_value	DWORD	0	result after a 0->1 leading edge from start: 16#1234ABCD
4	VAR				

In this example the input variables **input_value_1** and **input_value_2** are declared. However, you can write constants directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD



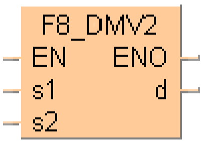
ST When programming with structured text, enter the following:

```
IF start THEN
    F7_MV2(input_value1, input_value2, output_value);
END_IF;
```

F8_DMV2

Two 32-bit data move

Description The function copies two 32-bit data areas specified at inputs **s1** and **s2** to a 32-bit ARRAY with two elements at output **d**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F8_DMV2 **(see page 1326)**

☞ **To copy three 32-bit data, use either the F191_DMV3 (see page 855) or P191_DMV3 instruction.**

Data types

Variable	Data type	Function
s1, s2	ANY32	source 32-bit area
d	ARRAY [0..1] of ANY32	destination, lower 32-bit area of 64-bit area

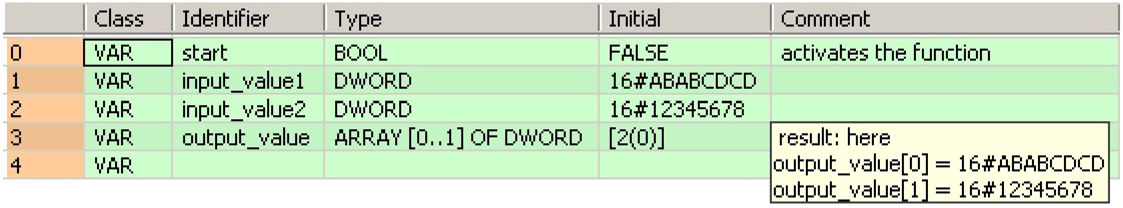
The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

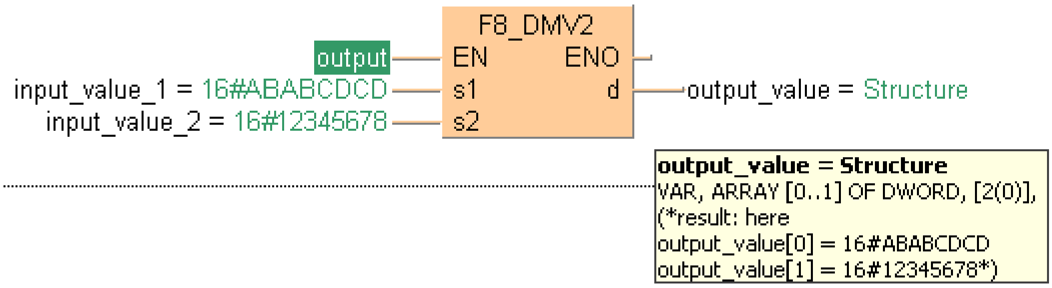


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value1	DWORD	16#ABABCDCD	
2	VAR	input_value2	DWORD	16#12345678	
3	VAR	output_value	ARRAY [0..1] OF DWORD	[2(0)]	result: here output_value[0] = 16#ABABCDCD output_value[1] = 16#12345678
4	VAR				

In this example the input variables **input_value_1** and **input_value_2** are declared. However, you can write constants directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out.

LD



output_value = Structure
VAR, ARRAY [0..1] OF DWORD, [2(0)],
(*result: here
output_value[0] = 16#ABABCDCD
output_value[1] = 16#12345678*)

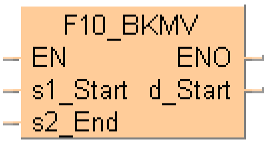
ST When programming with structured text, enter the following:

```
IF start THEN
    F8_DMV2(input_value_1, input_value_2, output_value);
END_IF;
```

F10_BKMV

Block move

Description The data block specified by the 16-bit starting area specified by **s1_Start** and the 16-bit ending area specified by **s2_End** are copied to the block starting from the 16-bit area specified by **d_Start** if the trigger **EN** is in the ON-state.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

The operands **s1_Start** and **s2_End** should be:

- in the same operand
- **s1_Start ≤ s2_End**

Whenever **s1_Start**, **s2_End** and **d_Start** are in the same data area:

- **s1_Start = d_Start**: data will be recopied to the same data area.

source	15 . 12	11 . . 8	7 . 4	3 . . 0
[0]	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1
[1]	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0
[2]	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1
[3]	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0
[4]	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 1

dest.	15 . 12	11 . . 8	7 . 4	3 . . 0
[0]	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0
[1]	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1
[2]	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0

PLC types **Availability of** F10_BKMV **(see page 1320)**

Data types

Variable	Data type	Function
s1_Start	ANY16	starting 16-bit area, source
s2_End		ending 16-bit area, source
d_Start		starting 16-bit area, destination

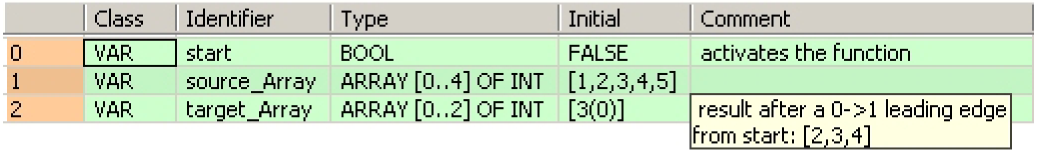
The variables **s1_Start, s2_End** and **d_Start** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1_Start, **s2_End**	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
d_Start	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

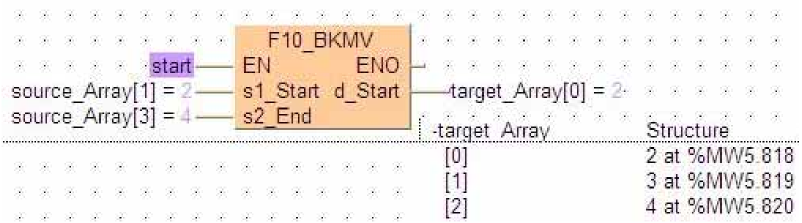
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source_Array	ARRAY [0..4] OF INT	[1,2,3,4,5]	
2	VAR	target_Array	ARRAY [0..2] OF INT	[3(0)]	result after a 0->1 leading edge from start: [2,3,4]

Body When the variable **start** changes from FALSE to TRUE, the function is carried out. It moves the data block starting at the 16-bit area specified by **s1** and ending at the 16-bit area specified by **s2** to the 16-bit area specified by **d**.

LD



target_Array	Structure
[0]	2 at %MW5.818
[1]	3 at %MW5.819
[2]	4 at %MW5.820

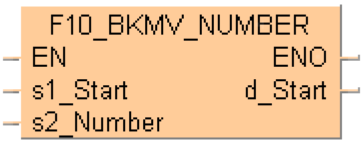
ST When programming with structured text, enter the following:

```
IF start THEN
    F10_BKMV( s1_Start:= source_Array[1],
        s2_End:= source_Array[3],
        d_Start=> target_Array[0]);
END_IF;
```

F10_BKMV_NUMBER

Block move by number

Description The data block specified by the 16-bit starting area specified by **s1_Start** and the number of WORDs specified by **s2_Number** are copied to the block starting from the 16-bit area specified by **d_Start** if the trigger EN is in the ON-state.



This instruction is a modification of the F10_BKMV (see page 819) generated by the compiler.

Whenever **s1_Start** and **d_Start** are in the same data area:

- **s1_Start = d_Start**: data will be recopied to the same data area.

PLC types **Availability of** F10_BKMV_NUMBER **(see page** 1320**)**

☞ **The value for 's2_Number' has to be greater than 0.**

Data types

Variable	Data type	Function
s1_Start	ANY16	starting 16-bit area, source
s2_Number		number of words to be copied, source
d_Start		starting 16-bit area, destination

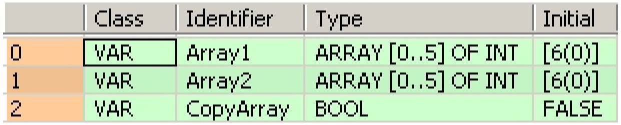
The variables **s1_Start**, **s2_Number** and **d_Start** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1_Start	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2_Number	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d_Start	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function is programmed in ladder diagram (LD). The same POU header is used for all programming languages.

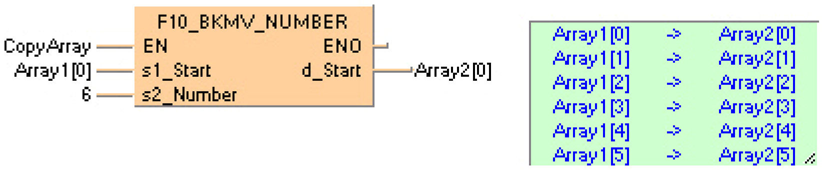
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Array1	ARRAY [0..5] OF INT	[6(0)]
1	VAR	Array2	ARRAY [0..5] OF INT	[6(0)]
2	VAR	CopyArray	BOOL	FALSE

Body When the variable **CopyArray** changes from FALSE to TRUE, the function is carried out. It copies the data block starting at the 16-bit area specified by **s1_Start** and the number of WORDs specified by **s2_Number** to the block starting from the 16-bit area specified by **d_Start**.

LD

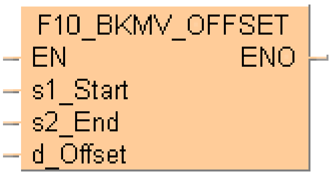


F10_BKMV_OFFSET

Block move to an offset from source

Description This instruction is a modification of the F10_BKMV (see page 819) generated by the compiler.

The data block specified by the 16-bit starting area specified by **s1_Start** and 16-bit ending area specified by **s2_End** are copied to the block starting from the 16-bit area specified by the offset **d_Offset** from **s1_Start** if the trigger EN is in the ON-state.



Whenever **s1_Start** and **s2_End** are in the same data area:

- d_Offset = 0: data will be recopied to the same data area.

PLC types **Availability of** F10_BKMV_OFFSET **(see page** 1320**)**

Data types

Variable	Data type	Function
s1_Start	ANY16	starting 16-bit area, source
s2_End		ending 16-bit area, source
d_Offset		offset from s1_Start, destination

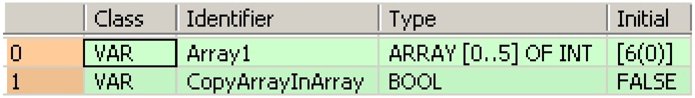
The variables **s1_Start**, **s2_End** and **d_Offset** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1_Start, **s2_End**	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
d_Offset	-	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example the function is programmed in ladder diagram (LD). The same POU header is used for all programming languages.

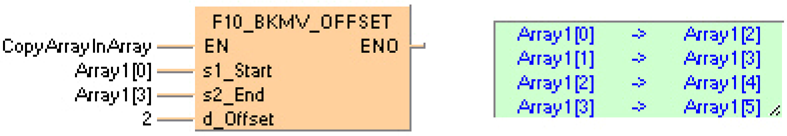
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Array1	ARRAY [0..5] OF INT	[6(0)]
1	VAR	CopyArrayInArray	BOOL	FALSE

Body When the variable **CopyArrayInArray** changes from FALSE to TRUE, the function is carried out. It copies the data block starting at the 16-bit area specified by **s1_Start** and 16-bit ending area specified by **s2_End** to the block starting from the 16-bit area specified by the offset **d_Offset** from **s1_Start**.

LD

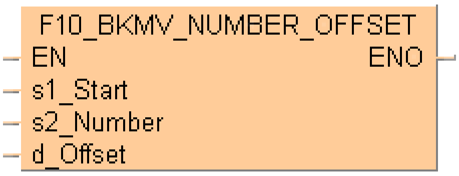


F10_BKMV_NUMBER_OFFSET

Block move by number to an offset from source

Description This instruction is a modification of the F10_BKMV (see page 819) generated by the compiler.

The data block specified by the 16-bit starting area specified by **s1_Start** and the number of WORDs specified by **s2_Number** are copied to the block starting from the 16-bit area specified by the offset **d_Offset** from **s1_Start** if the trigger EN is in the ON-state.



Whenever **d_Offset** = 0: data will be recopied to the same data area.

PLC types **Availability of** F10_BKMV_NUMBER_OFFSET **(see page 1320)**

☞ **The value for 's2_number' has to be greater than 0.**

Data types

Variable	Data type	Function
s1_Start	ANY16	starting 16-bit area, source
s2_Number		Number of words to be copied, source
d_Offset		starting 16-bit area, destination

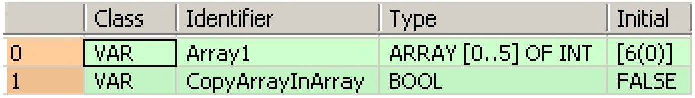
The variables **s1_Start**, **s2_Number** and **d_Offset** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1_Start	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2_Number	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d_Offset	-	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example the function is programmed in ladder diagram (LD). The same POU header is used for all programming languages.

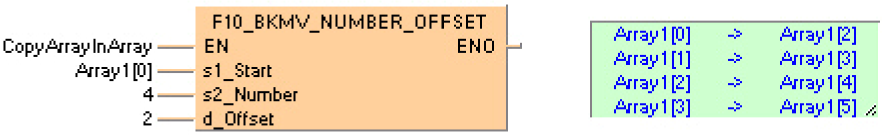
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Array1	ARRAY [0..5] OF INT	[6(0)]
1	VAR	CopyArrayInArray	BOOL	FALSE

Body When the variable **CopyArrayInArray** changes from FALSE to TRUE, the function is carried out. It copies the data block starting at the 16-bit area specified by **s1_Start** and the number of WORDs specified by **s2_Number** to the block starting from the 16-bit area specified by the offset **d_Offset** from **s1_Start**.

LD

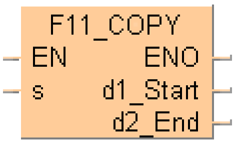


```
Array1[0]  ->  Array1[2]
Array1[1]  ->  Array1[3]
Array1[2]  ->  Array1[4]
Array1[3]  ->  Array1[5]
```

F11_COPY

Block copy

Description The 16-bit equivalent constant or 16-bit area specified by **s** is copied to all 16-bit areas of the block specified by **d1_Start** and **d2_End** if the trigger **EN** is in the ON-state.



The operands **d1_Start** and **d2_End** should be:

- in the same operand
- **d1_Start ≤ d2_End**

source	15		.	12	11	.	.	8	7	.	.	4	3	.	.	0
	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

dest.	15		.	12	11	.	.	8	7	.	.	4	3	.	.	0
[0]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
[2]	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
[3]	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
[4]	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
[5]	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F11_COPY **(see page 1320)**

Data types

Variable	**Data type**	**Function**
s	ANY16	source 16-bit area
d1_Start		starting 16-bit area, destination
d2_End		ending 16-bit area, destination

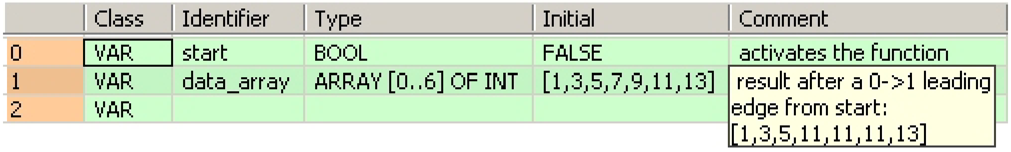
The variables **s, d1_Start** and **d2_End** have to be of the same data type.

Operands

For	**Relay**				**T/C**		**Register**			**Constant**
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1_Start, **d2_End**	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

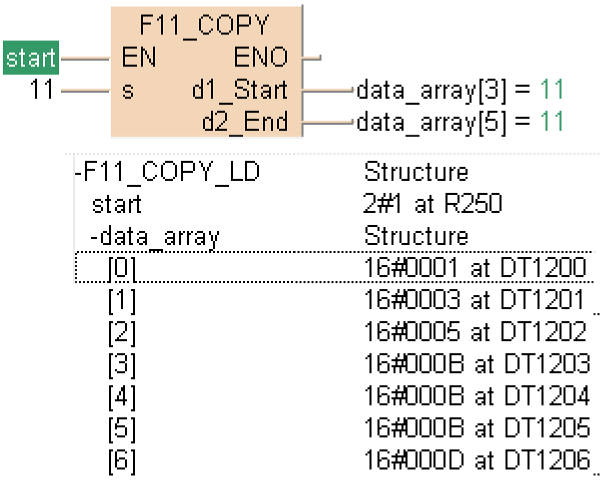
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_array	ARRAY [0..6] OF INT	[1,3,5,7,9,11,13]	result after a 0->1 leading edge from start: [1,3,5,11,11,11,13]
2	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



```
-F11_COPY_LD            Structure
  start                 2#1 at R250
 -data_array            Structure
    [0]                 16#0001 at DT1200
    [1]                 16#0003 at DT1201
    [2]                 16#0005 at DT1202
    [3]                 16#000B at DT1203
    [4]                 16#000B at DT1204
    [5]                 16#000B at DT1205
    [6]                 16#000D at DT1206
```

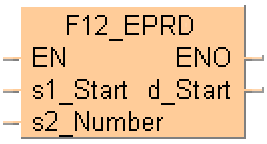
ST When programming with structured text, enter the following:

```
IF start THEN
    (* Copy the value 11 to data_array[3], *)
    (* data_array[4] and data_array[5] *)
    F11_COPY( s:= 11,
        d1_Start=> data_array[3],
        d2_End=> data_array[5]);
END_IF;
```

F12_EPRD

EEPROM read from memory

Description Using this instruction data will be copied from EEPROM/ Flash-ROM to the destination area (DT). The copy function is carried out with blocks only. Thus you can not copy single words. The block size and the number of blocks is shown in the table "PLC specific information". Also ensure that there at least 64/ 2048 free data registers (1 block = 64 words/ 2048 words (DTs)) reserved for the destination area.



PLC types **Availability of** F12_EPRD **(see page 1320)**

Data types

Variable	Data type	Function
EN	BOOL	Activation of the function (when EN has the state TRUE, the function block will be executed at every PLC scan)
s1	ANY32	EEPROM start block number
s2		Number of blocks to be read (1 block = 64 words/ 2048 words (DTs))
d	ANY16	DT start address for information to be written
ENO	BOOL	When the function was executed, ENO is set to TRUE. Helpful at cascading functions with EN-functionality

Operands

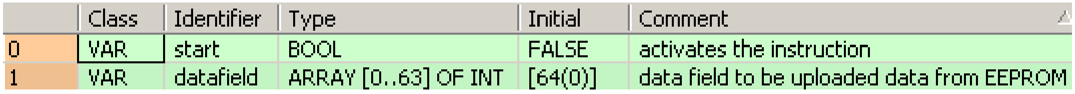
For	Relay				T/C		Register			Constant
s1, s2	DWX	DWY	DWR	-	DSV	DEV	DDT	-	-	dec. or hex.
d	-	-	-	-	-	-	DT	-	-	-

PLC specific information

PLC type	FP0 2,7k C10/C14/C16 and FP-e	FP0 5k C32	FP0 10k T32CP	FP-Sigma, FP-X, FP0R
ROM	EEPROM	EEPROM	EEPROM	Flash-ROM
Block size (1 block)	64 words (64x16bit)	64 words (64x16bit)	64 words (64x16bit)	2048 words
EEPROM start block number	0 to 9	0 to 95	0 to 255	0 to 15
Number of blocks to be read / written each execution	1 to 2	1 to 8	1 to 255	1 (writing) 1 to 16 (reading)
Write duration (Additional scan time)	< 20 ms each block	< 5 ms each block	< 5 ms each block	< 100ms each block
Read duration (Additional scan time)	Less than 1 ms each block	Less than 1 ms each block	Less than 1 ms each block	9.94µs + (1562.6*number of blocks) µs
Max number of writing events ☞ **Power down, RUN -> PROG mode changes are also counted.**	100,000	10,000	10,000	10,000
Max read times	No limit	No limit	No limit	No limit

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

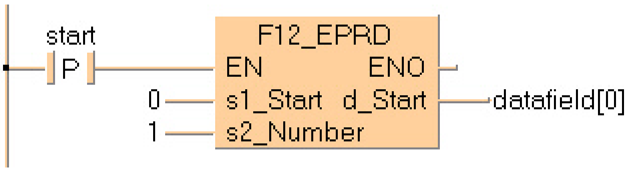
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the instruction
1	VAR	datafield	ARRAY [0..63] OF INT	[64(0)]	data field to be uploaded data from EEPROM

Body When the variable **start** changes from FALSE to TRUE, the function is carried out. The function reads the first block (= 64 words) after start block number 0 from the EEPROM and writes the information into the data fields from **datafield[0]** until **datafield[63]**.

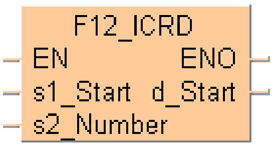
LD



F12_ICRD

IC card extended memory read

Description The data for the number of words specified by **s2_Number** are read from the address in the IC card extended memory area specified by **s1_Start** and written to the area specified by **d_Start**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F12_ICRD **(see page 1320)**

Data types

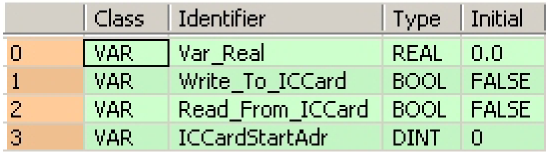
Variable	Data type	Function
s1_Start	ANY32	starting 32-bit area to be read in extended memory
s2_Number		number of words to be read
d_Start	ANY16	destination, starting 16-bit area

Operands

For	Relay				T/C		Register			Constant
s1_Start	-	-	-	-	-	-	-	-	-	dec. or hex.
s2_Number	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d_Start	-	WY	WR	WL	SV	EV	DT	LD	FL	-

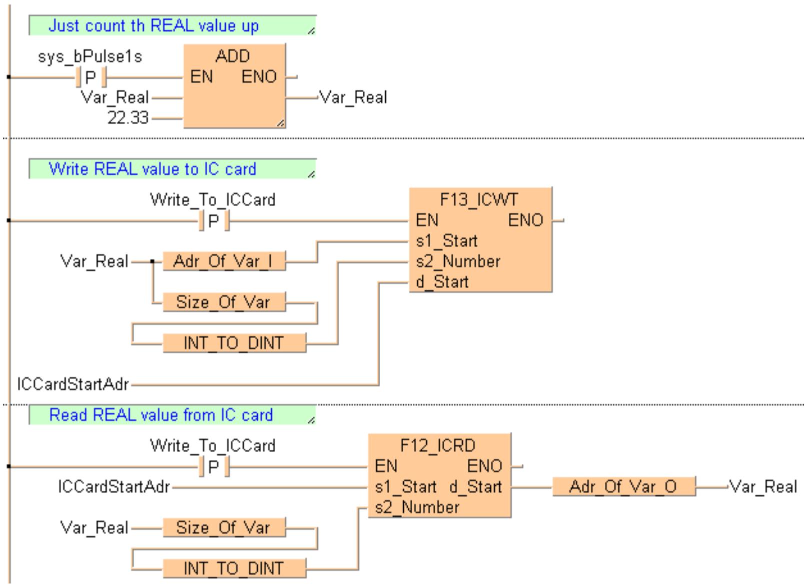
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Var_Real	REAL	0.0
1	VAR	Write_To_ICCard	BOOL	FALSE
2	VAR	Read_From_ICCard	BOOL	FALSE
3	VAR	ICCardStartAdr	DINT	0

LD

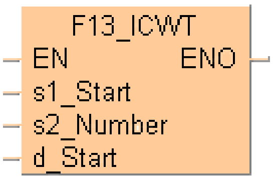


ST When programming with structured text, enter the following:

```
IF DF(R901C) THEN
    Var_Real := Var_Real + 22.33;
END_IF;

(* Write a REAL value to the IC Card *)
IF DF(Write_To_ICCard) THEN
    F13_ICWT( s1_Start:= Adr_Of_Var( Var_Real ) , s2_Number:= INT_TO_DINT(
Size_Of_Var( Var_Real ) ), d_Start:= ICCardStartAdr );
END_IF;

(*Read a REAL value from the IC Card*)
IF DF(Read_From_ICCard) THEN
    F12_ICRD( s1_Start:= ICCardStartAdr, s2_Number:=  INT_TO_DINT(
Size_Of_Var( Var_Real ) ) ,
        d_Start=> Adr_Of_Var( Var_Real ) );
END_IF;
```

F13_ICWT IC card extended memory write

Description The data for the number of words specified by **s2_Number** are read from the address specified by **s1_Start** and written to the extended memory area in the IC card specified by **d_Start**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F13_ICWT **(see page 1321)**

Data types

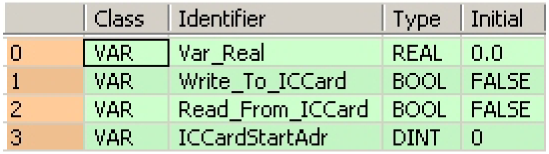
Variable	Data type	Function
s1_Start	ANY16	source data, starting 16-bit area
s2_Number	ANY32	number of words to be read then written to IC card
d_Start		destination area of IC card expansion memory

Operands

For	Relay				T/C		Register			Constant
s1_Start	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2_Number	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d_Start	-	-	-	-	-	-	-	-	-	dec. or hex.

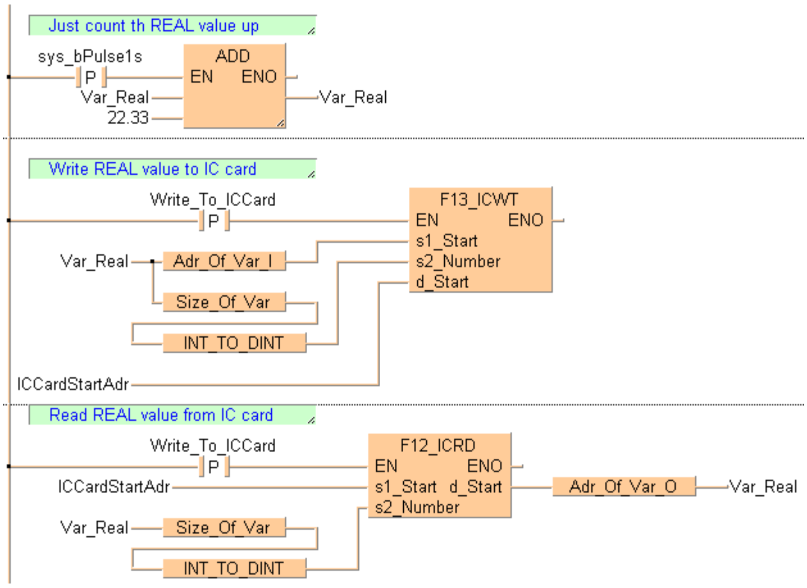
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Var_Real	REAL	0.0
1	VAR	Write_To_ICCard	BOOL	FALSE
2	VAR	Read_From_ICCard	BOOL	FALSE
3	VAR	ICCardStartAdr	DINT	0

LD



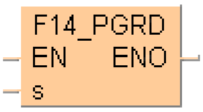
ST When programming with structured text, enter the following:

```
IF DF(R901C) THEN
    Var_Real := Var_Real + 22.33;
END_IF;

(* Write a REAL value to the IC Card *)
IF DF(Write_To_ICCard) THEN
    F13_ICWT( s1_Start:= Adr_Of_Var( Var_Real ) , s2_Number:= INT_TO_DINT(
Size_Of_Var( Var_Real ) ), d_Start:= ICCardStartAdr );
END_IF;

(*Read a REAL value from the IC Card*)
IF DF(Read_From_ICCard) THEN
    F12_ICRD( s1_Start:= ICCardStartAdr, s2_Number:=  INT_TO_DINT(
Size_Of_Var( Var_Real ) ) ,
        d_Start=> Adr_Of_Var( Var_Real ) );
END_IF;
```

F14_PGRD

Program Read from IC card

Description When the execution criterion of **F/P14_PGRD** is turned ON, the execution proceeds until the END. The program subsequently switches to the program specified by **s**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F14_PGRD **(see page 1321)**

Data types

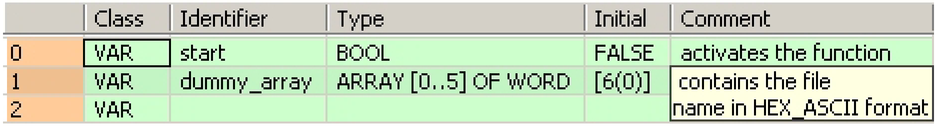
Variable	Data type	Function
s	ARRAY [0..5] of WORD	starting address of area storing program

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

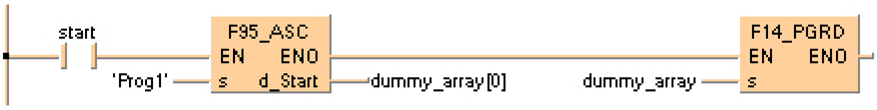
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	dummy_array	ARRAY [0..5] OF WORD	[6(0)]	contains the file name in HEX_ASCII format
2	VAR				

Body When the variable **start** is set to TRUE, the function is carried out. The instruction reads the program Prog1 from the IC card and executes it.

LD



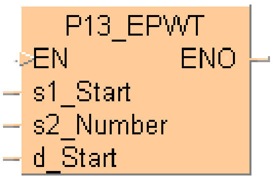
ST When programming with structured text, enter the following:

```
IF start THEN
    F95_ASC( s:= 'Prog1',
        d_Start=> dummy_array[0]);
    F14_PGRD( dummy_array );
END_IF;
```

P13_EPWT

EEPROM write to memory

Description Using this instruction data will be copied from the data area (DT) to the EEPROM/ Flash-ROM.



The EEPROM memory is not the same as the hold area. The hold area stores data in real time. Whenever the power shuts down, the hold data is stored in the EEPROM memory. The P13_EPWT instruction sends data into the EEPROM only when the instruction is executed. It also has a limitation of the number of times you can write to it (see table below). You must make sure that the P13_EPWT instruction will not be executed more often than the specified number of writes.

For example, if you execute P13_EPWT with R901A relay (pulse time 0.1s), the EEPROM will become inoperable after 100,000 * 0.1 sec=10,000 sec (2.8 hours). However if you want to hold your profile data such as positioning parameters or any other parameter values that are changed infrequently, you will find this instruction very useful.

PLC types **Availability of** P13_EPWT **(see page 1329)**

☞ **One of the two input variables 's2' or 'd' has to be assigned constant number value.**

Data types

Variable	Data type	Function
EN	BOOL	Activation of the function (when EN changes from FALSE to TRUE, the function will be executed one time)
s1	INT, WORD	DT start address of the block(s) that you want to save
s2	DINT, DWORD	Number of blocks to write (1 block = 64 words/ 2048 words (DTs))
d	DINT, DWORD	EEPROM start block number
ENO	BOOL	When the function was executed, ENO is set to TRUE. Helpful at cascading functions with EN-functionality

Operands

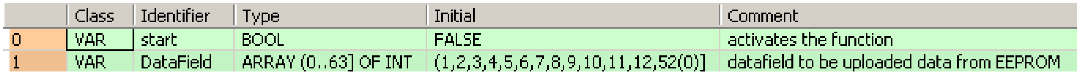
For	Relay				T/C		Register			Constant
s1	-	-	-	-	-	-	DT	-	-	-
s2, d	DWX	DWY	DWR	-	DSV	DEV	DDT	-	-	dec. or hex.

PLC specific information

PLC type	FP0 2,7k C10/C14/C16 and FP-e	FP0 5k C32	FP0 10k T32CP	FP-Sigma, FP-X, FP0R
ROM	EEPROM	EEPROM	EEPROM	Flash-ROM
Block size (1 block)	64 words (64x16bit)	64 words (64x16bit)	64 words (64x16bit)	2048 words
EEPROM start block number	0 to 9	0 to 95	0 to 255	0 to 15
Number of blocks to be read / written each execution	1 to 2	1 to 8	1 to 255	1 (writing) 1 to 16 (reading)
Write duration (Additional scan time)	< 20 ms each block	< 5 ms each block	< 5 ms each block	< 100ms each block
Read duration (Additional scan time)	Less than 1 ms each block	Less than 1 ms each block	Less than 1 ms each block	9.94µs + (1562.6*number of blocks) µs
Max number of writing events ☞ **Power down, RUN -> PROG mode changes are also counted.**	100,000	10,000	10,000	10,000
Max read times	No limit	No limit	No limit	No limit

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

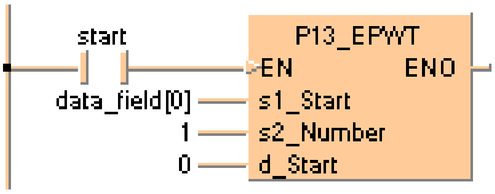
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	DataField	ARRAY (0..63] OF INT	(1,2,3,4,5,6,7,8,9,10,11,12,52(0)]	datafield to be uploaded data from EEPROM

Body When the variable **start** changes from FALSE to TRUE, the function is carried out. The function reads the contents of data field[0] until data field[63] (s2* = 1 => 1 block = 64 words) and writes the information after start block number 0 into the EEPROM.

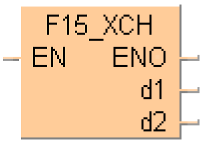
LD



F15_XCH

16-bit data exchange

Description The contents in the 16-bit areas specified by **d1** and **d2** are exchanged if the trigger **EN** is in the ON-state.



Bit	15 . . 12	11 . . 8	7 . . 4	3 . . 0
d1	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1
d2	0 0 0 0	0 0 0 0	0 0 0 1	1 0 0 0

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F15_XCH **(see page 1321)**

Data types

Variable	Data type	Function
d1	ANY16	16-bit area to be exchanged with **d2**
d2		16-bit area to be exchanged with **d1**

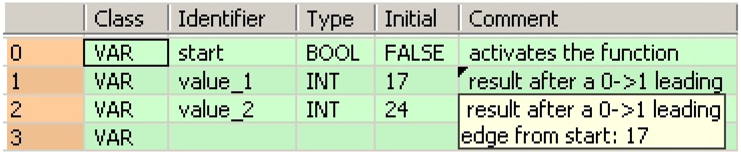
The variables **d1** and **d2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
d1, d2	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

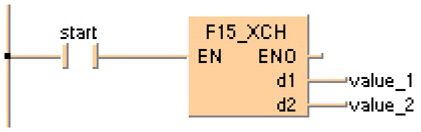
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	INT	17	result after a 0->1 leading
2	VAR	value_2	INT	24	result after a 0->1 leading edge from start: 17
3	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



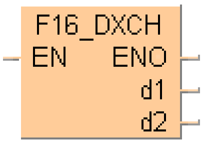
ST When programming with structured text, enter the following:

```
IF start THEN
    F15_XCH(value_1, value_2);
END_IF;
```

F16_DXCH

32-bit data exchange

Description Two 32-bit data specified by **d1** and **d2** are exchanged if the trigger **EN** is in the ON-state.



Bit	31 . . 28	27 . . 24	23 . . 20	19 . .16	15 . . 12	10 . . 8	7 . . 4	3 . . 0
d1	0 0 0 0	1 0 0 1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1
d2	0 0 0 0	0 1 1 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	1 0 0 0

32-bit area

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F16_DXCH **(see page 1321)**

Data types

Variable	Data type	Function
d1	ANY32	32-bit area to be exchanged with **d2**
d2		32-bit area to be exchanged with **d1**

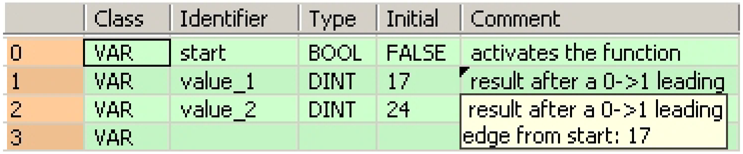
The variables **d1** and **d2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
d1, d2	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

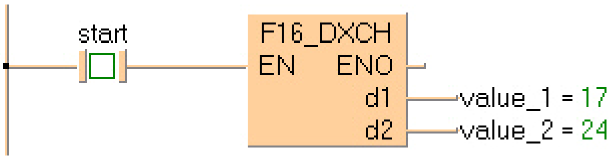


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	DINT	17	result after a 0->1 leading
2	VAR	value_2	DINT	24	result after a 0->1 leading
3	VAR				

result after a 0->1 leading edge from start: 17

Body When the variable **start** is set to TRUE, the function is carried out.

LD



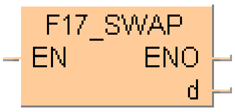
ST When programming with structured text, enter the following:

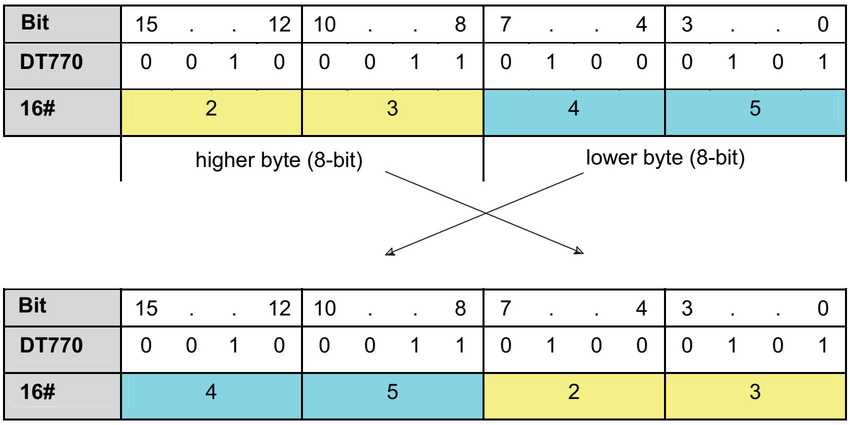
```
IF start THEN
    F16_DXCH(value_1, value_2);
END_IF;
```

F17_SWAP

Higher/lower byte in 16-bit data exchange

Description The higher byte (higher 8-bits) and lower bytes (lower 8-bits) of a 16-bit area specified by **d** are exchanged if the trigger **EN** is in the ON-state. 1 byte means 8 bit.





Bit	15	.	.	12	10	.	.	8	7	.	.	4	3	.	.	0
DT770	0	0	1	0	0	0	1	1	0	1	0	0	0	1	0	1
16#	2				3				4				5			

Bit	15	.	.	12	10	.	.	8	7	.	.	4	3	.	.	0
DT770	0	0	1	0	0	0	1	1	0	1	0	0	0	1	0	1
16#	4				5				2				3			

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F17_SWAP **(see page 1322)**

Data types

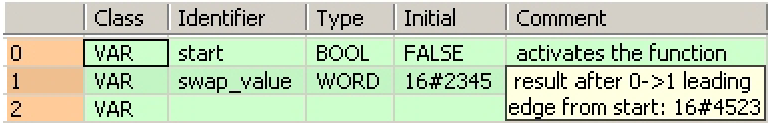
Variable	Data type	Function
d	ANY16	16-bit area in which the higher and lower bytes are swapped (exchanged)

Operands

For	Relay				T/C		Register			Constant
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

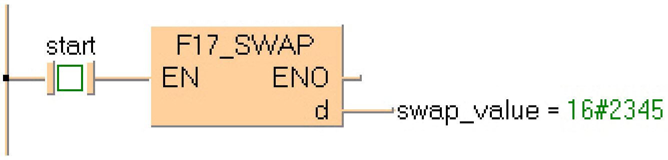
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	swap_value	WORD	16#2345	result after 0->1 leading edge from start: 16#4523
2	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



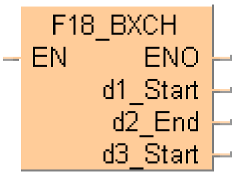
ST When programming with structured text, enter the following:

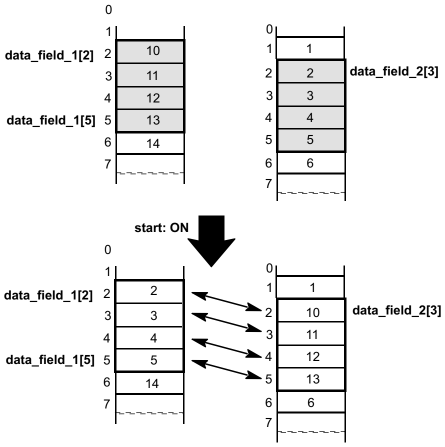
```
IF start THEN
    F17_SWAP(swap_value);
END_IF;
```

F18_BXCH

16-bit blocked data exchange

Description The function exchanges one 16-bit data block for another. The beginning of the first data block is specified at output **d1_Start** and its end at output **d2_End**. Output **d3_Start** specifies the beginning of the second data block.





This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F18_BXCH **(see page 1322)**

Data types

Variable	Data type	Function
d1_Start	ANY16	starting 16-bit area of block data 1
d2_End		ending 16-bit area of block data 1
d3_Start		starting 16-bit area of block data 2

Operands

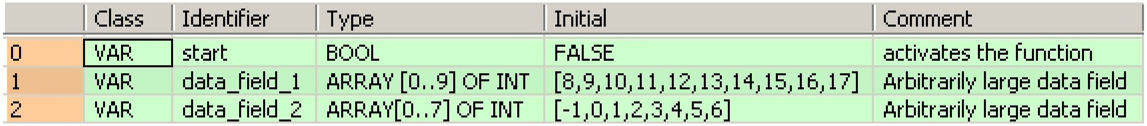
For	Relay				T/C		Register			Constant
d1_Start, d2_End, d3_Start	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the address of the variables at outputs **d1_Start** > **d2_End** ▪ the data block to be exchanged is larger than the target area.
R9008	%MX0.900.8	for an instant	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

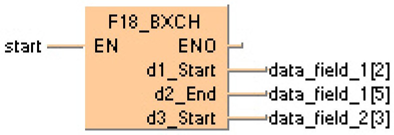
POU header All input and output variables used for programming this function have been declared in the POU header.

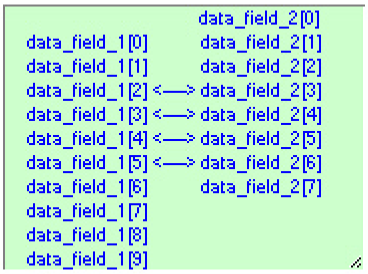


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_field_1	ARRAY [0..9] OF INT	[8,9,10,11,12,13,14,15,16,17]	Arbitrarily large data field
2	VAR	data_field_2	ARRAY[0..7] OF INT	[-1,0,1,2,3,4,5,6]	Arbitrarily large data field

Body When the variable **start** is set to TRUE, the function is carried out. It exchanges the data of ARRAY **data_field_1** (from the 2nd to the 5th element) with the data of ARRAY **data_field_2** (from the 3rd element on).

LD





```
                          data_field_2[0]
data_field_1[0]           data_field_2[1]
data_field_1[1]           data_field_2[2]
data_field_1[2] <—> data_field_2[3]
data_field_1[3] <—> data_field_2[4]
data_field_1[4] <—> data_field_2[5]
data_field_1[5] <—> data_field_2[6]
data_field_1[6]           data_field_2[7]
data_field_1[7]
data_field_1[8]
data_field_1[9]
```

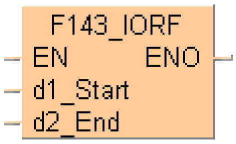
ST When programming with structured text, enter the following:

```
IF start THEN
    F18_BXCH(
        d1_Start=> d1[2], d2_End=> d1[4], d3_Start=>  d2[1]);
END_IF;
```

F143_IORF

Partial I/O update

Description Updates the inputs or outputs specified by the value of **d1_Start** (starting word address) and the value of **d2_End** (ending word address) immediately after the trigger **EN** is in the ON-state even in the program execution stage.



Using this instruction, you can update inputs or outputs without the time-lag caused by scanning.

The same type of operand should be specified for **d1_Start** and **d2_End**.

PLCs with configurable, serially numbered I/O addresses:

FP2, FP2SH, FP3 /5 /10 /10SH (PLCs **with** backplanes)

- Specify the word address as 0 ≤ **d1_Start** ≤ **d2_End** ≤ 127.
  If only WX10 (or WY10) are to be updated based on the I/O-address configuration, **d1_Start** and **d2_End** will be set as follows: **d1_Start** = 10 and **d2_End** = 10.
- Set the same word address in **d1_Start** and **d2_End** to update only 1 word.

The partial I/O update instruction is executed only for the I/O units on the master backplane or expansion backplane. It is not executed for the I/O unit in the slave station of the Remote I/O System.

PLCs whose I/O addresses cannot be configured and are not serially numbered:

FP-Σ, FP0 (PLCs **without** backplanes)

The instruction F143_IORF updates the inputs and outputs specified by **d1_Start** (starting word address) and **d2_End** (ending word address) immediately after the trigger turns ON even in the program execution stage.

☞

- **With the FP0 and FP-Σ, refreshing initiated by the IORF command is done only for the control unit.**
- **If d1_Start and d2_End are variables and not constants, then the compiler automatically accesses the variables' values via the index register.**
- **With input refreshing, WX0 should be specified for d1_Start and d2_End.**
- **With output refreshing, WY0 should be specified for d1_Start and d2_End.**

PLC types **Availability of F143_IORF (see page 1321)**

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Data types

Variable	Data type	Function
d1_Start	ANY16	starting word address
d2_End		ending word address

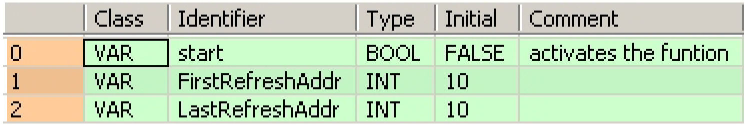
The same type of operand should be specified for **d1_Start** and **d2_End**.

Operands

For	Relay				T/C		Register			Constant
d1_Start	WX	WY	-	WL	SV	EV	DT	-	FL	dec. or hex.
d2_End	WX	WY	-	WL	SV	EV	DT	-	FL	dec. or hex.

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

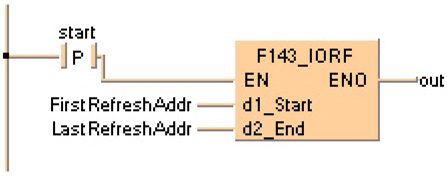
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the funtion
1	VAR	FirstRefreshAddr	INT	10	
2	VAR	LastRefreshAddr	INT	10	

Body When the variable **start** changes from FALSE to TRUE, the function is carried out. To update WX10 and WY10 based on the master I/O map configuration, set d1 = 10 and d2 = 10.

LD



ST When programming with structured text, enter the following:

```
(* PLCs with backplanes  FP-C/FP2/FP2SH/FP3/FP10SH *)
IF start THEN
    (* Updates the input/output relay of word no. 0 to 1 *)
    F143_IORF( 0, 1);
END_IF;
```

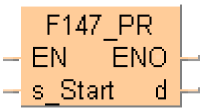
ST

```
(* PLCs without backplanes FP0, FP-Sigma *)
IF start THEN
    (* Updates the input/output relay of word no. 0 to 1 *)
    F143_IORF(WX0, WX1);
    F143_IORF(WY0, WY1);
END_IF;
```

F147_PR

Parallel printout

Description Outputs the ASCII codes for 12 characters stored in the 6-word area specified by **s** via the word external output relay specified by **d** if the trigger **EN** is in the ON-state. If a printer is connected to the output specified by **d**, a character corresponding to the output ASCII code is printed.



Only bit positions 0 to 8 of d are used in the actual printout. ASCII code is output in sequence starting with the lower byte of the starting area. Three scans are required for 1 character constant output. Therefore, 37 scans are required until all characters constants are output.

Since it is not possible to execute multiple F147_PR instructions in one scan, use print-out flag sys_bIsActive_F147_PR (ms-its:SysVars.chm::/64395.htm#o64401) to be sure they are not executed simultaneously. If the character constants convert to ASCII code, use of the F95_ASC (see page 661) instruction is recommended.

PLC types **Availability of** F147_PR **(see page 1321)**

Data types

Variable	Data type	Function
s	ANY16	starting 16-bit area for storing 12 bytes (6 words) of ASCII codes (source)
d	WORD	word external output relay used for output of ASCII codes (destination)

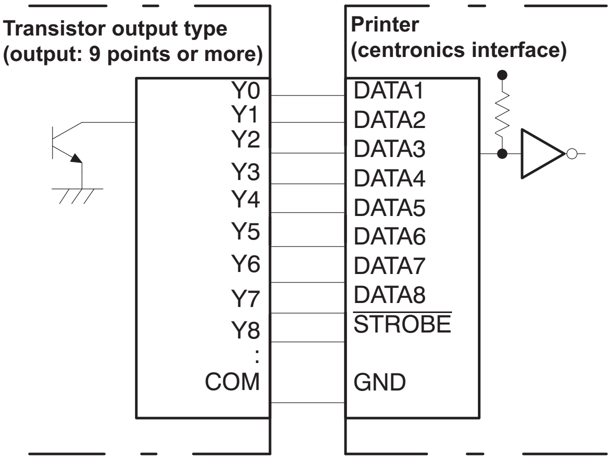
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
d	-	WY	-	-	-	-	-	-	-	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the ending area for storing ASCII codes exceeds the limit ▪ the trigger of another F147_PR instruction turns on while one F147_PR instruction is being executed
R9008	%MX0.900.8	for an instant	
R9033	%MX0.903.3	permanently	▪ a F147_PR instruction is being executed

■ Connection example

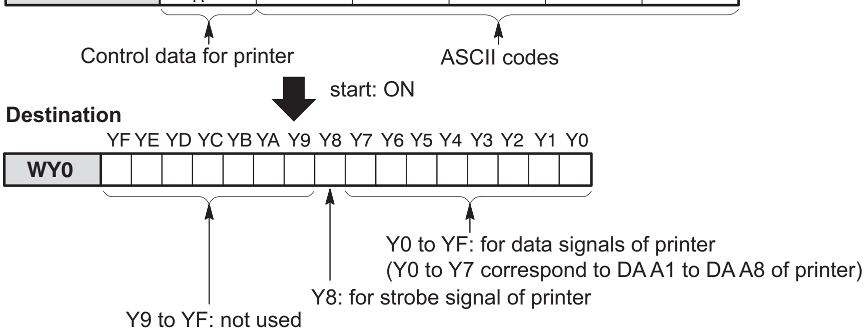


Example In this example the function is programmed in ladder diagram (LD). The same POU header is used for all programming languages.

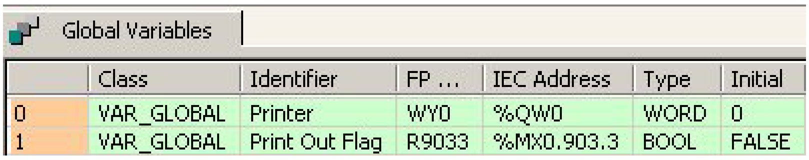
The ASCII codes stored in the string **PrintOutString** are output through word external output relay WY0 when trigger **Start** turns on.

Source: ASCII code for 12 characters A, B, C, D, E, F, G, H, I and J

	PrintOutString											
ASCII HEX code	0D	0A	4A	49	48	47	46	45	44	43	42	41
ASCII character	C_R	LF	J	I	H	G	F	E	D	C	B	A


Control data for printer
ASCII codes
start: ON
Destination
YF YE YD YC YB YA Y9 Y8 Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
WY0
Y0 to YF: for data signals of printer
(Y0 to Y7 correspond to DA A1 to DA A8 of printer)
Y8: for strobe signal of printer
Y9 to YF: not used

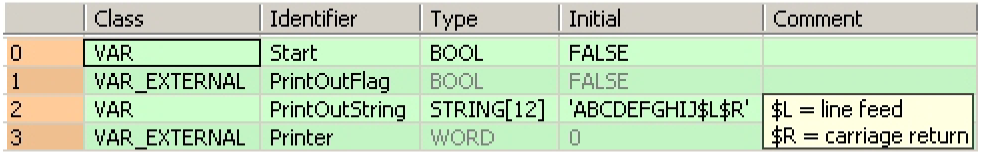
GVL In the global variable list, you define variables that can be accessed by all POUs in the project.



Global Variables

	Class	Identifier	FP ...	IEC Address	Type	Initial
0	VAR_GLOBAL	Printer	WY0	%QW0	WORD	0
1	VAR_GLOBAL	Print Out Flag	R9033	%MX0.903.3	BOOL	FALSE

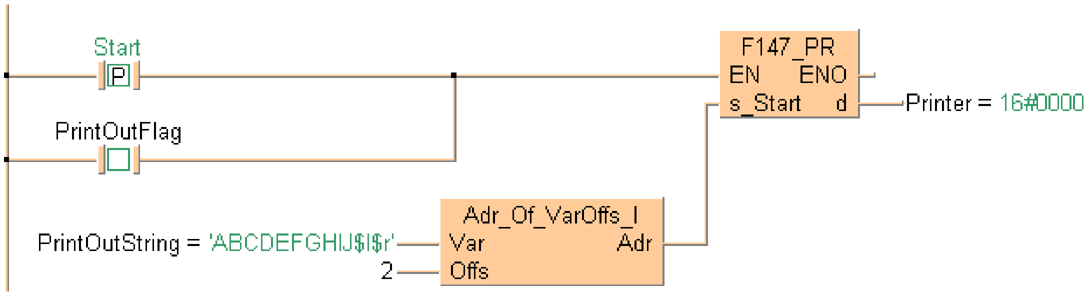
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR_EXTERNAL	PrintOutFlag	BOOL	FALSE	
2	VAR	PrintOutString	STRING[12]	'ABCDEFGHIJLR'	$L = line feed $R = carriage return
3	VAR_EXTERNAL	Printer	WORD	0	

Body

LD



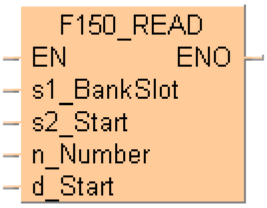
ST When programming with structured text, enter the following:

```
IF DF(start) OR PrintOutFlag THEN
    F147_PR( Adr_Of_VarOffs( PrintOutString, 2), Printer);
END_IF;
```

F150_READ

Data read from intelligent units

Description Reads data from the shared memory in an intelligent module.



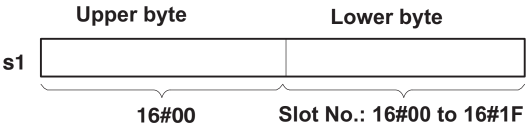
This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

The **n** words of the data stored in the shared memory of the intelligent unit/board specified by **s1** are read from the address specified by **s2**, and are stored in the area specified by **d** of the CPU.

The number of variable arguments at the inputs is limited by the available index registers of the PLC.

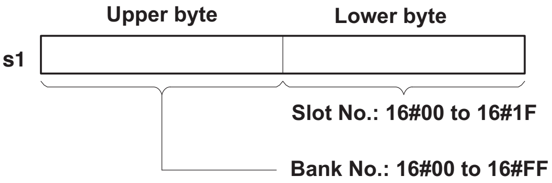
Specifying s1
Intelligent unit without bank

Specify the slot number in which the target intelligent unit has been installed.



Intelligent unit with bank

Specify the slot number (hex. constant) in which the target intelligent unit has been installed, and the bank number (hex. constant).



Reference: Intelligent unit with bank

Name	Order Number
FP3 expansion data memory unit	AFP32091 AFP32092
FPΣ expansion data memory unit	AFPG201

PLC types **Availability of** F150_READ **(see page 1321)**

Data types

Variable	Data type	Function
s1	ANY16	Specifies the bank/slot number in the shared memory of the intelligent module
s2		Specifies the starting address in the shared memory of the intelligent module (source data address)
n	INT	Specifies the number of words to be read
d	ANY16	Starting address in the CPU for storing data read (destination address)

Operands

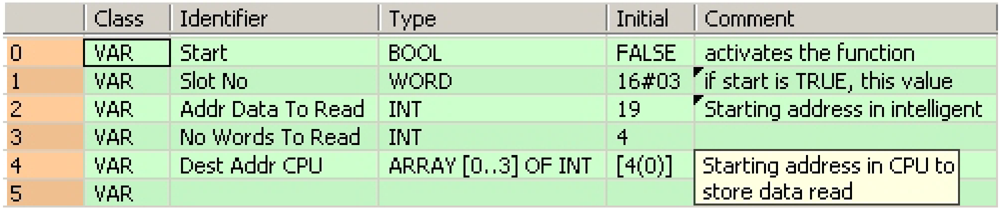
For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s2	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ **s1** exceeds the limit of specified range ▪ the data read exceeds the area of **d**
R9008	%MX0.900.8	for an instant	

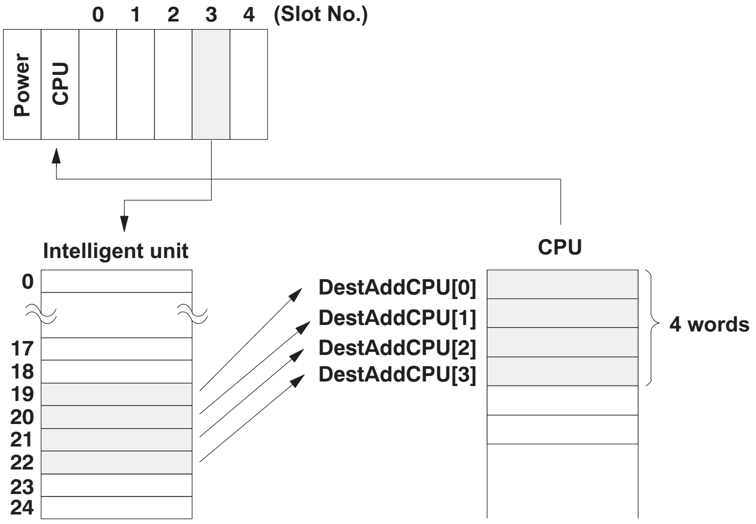
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.

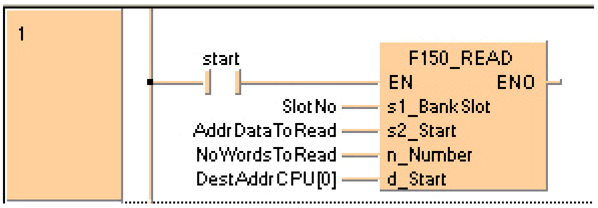


	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	activates the function
1	VAR	Slot No	WORD	16#03	if start is TRUE, this value
2	VAR	Addr Data To Read	INT	19	Starting address in intelligent
3	VAR	No Words To Read	INT	4	
4	VAR	Dest Addr CPU	ARRAY [0..3] OF INT	[4(0)]	Starting address in CPU to store data read
5	VAR				

Body Reads 4 words of data stored in the addresses starting from 19, specified in **AddrDataToRead**, of the intelligent unit's shared memory (located in slot 3). Then it stores them in the array **DestAddrCPU**, when **Start** turns on.



LD



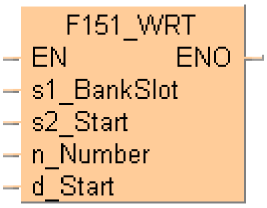
ST When programming with structured text, enter the following:

```
IF start THEN
    F150_READ( s1_BankSlot:= SlotNo,
        s2_Start:= AddrDataToRead,
        n_Number:= NoWordsToRead,
        d_Start:= DestAddrCPU[0]);
END_IF;
```

F151_WRT

Write into memory of intelligent units

Description Writes data into the shared memory of an intelligent unit.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

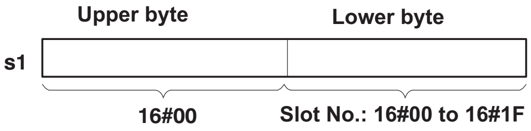
Writes **n** words of the initial data from the area specified by **s2** of the CPU to the address specified by **d** of the shared memory of the intelligent unit specified by **s1**.

The number of variable arguments at the inputs is limited by the available index registers of the PLC.

Specifying s1

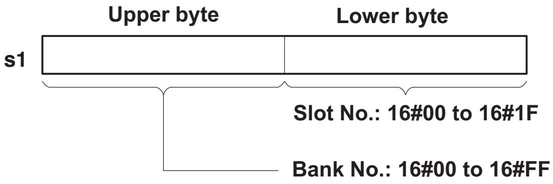
Intelligent unit without bank

Specify the slot number in which the target intelligent unit has been installed.



Intelligent unit with bank

Specify the slot number (hex. constant) in which the target intelligent unit has been installed, and the bank number (hex. constant).



Reference: Intelligent unit with bank

Name	Order Number
FP3 expansion data memory unit	AFP32091 AFP32092
FPΣ expansion data memory unit	AFPG201

PLC types **Availability of** F151_WRT **(see page 1321)**

Data types

Variable	Data type	Function
s1	ANY16	Specifies the bank/slot number in the shared memory of the intelligent module
s2		Starting address for data in the shared memory of the CPU
n	INT	Specifies the number of words to be written to the shared memory
d	ANY16	Specifies the starting address in the intelligent unit for storing data written (destination address)

Operands

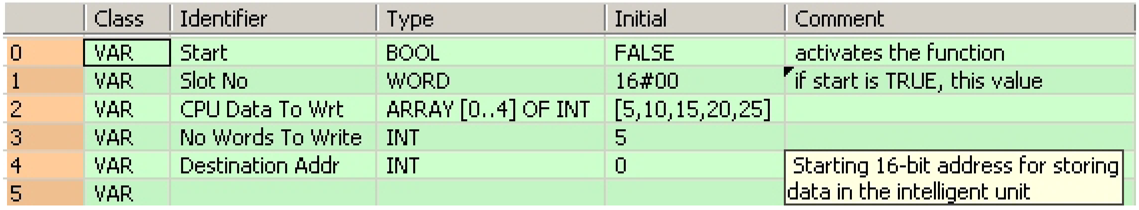
For	Relay				T/C		Register			Constant
s1	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s2	-	WY	WR	WL	SV	EV	DT	LD	FL	-
n	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ **s1** exceeds the limit of specified range ▪ the data read exceeds the area of **d**
R9008	%MX0.900.8	for an instant	

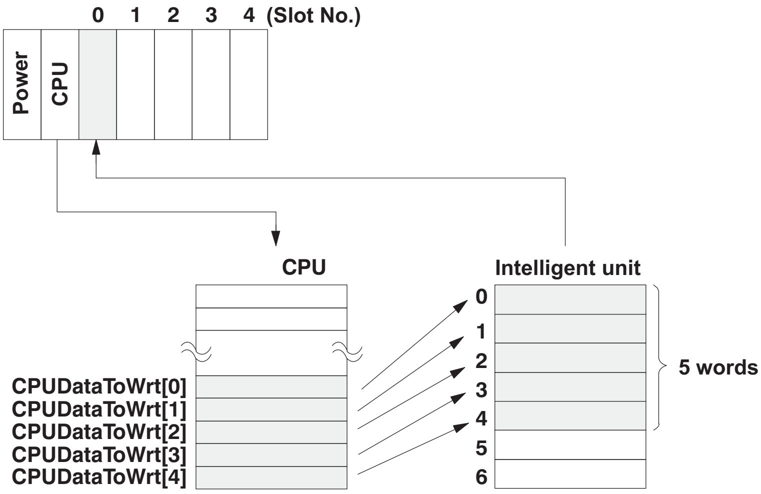
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.

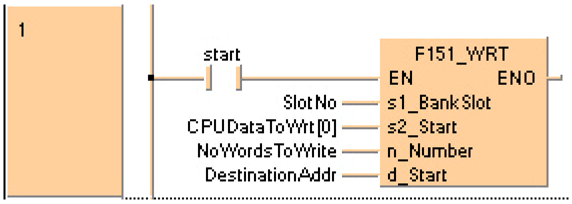


	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	activates the function
1	VAR	Slot No	WORD	16#00	if start is TRUE, this value
2	VAR	CPU Data To Wrt	ARRAY [0..4] OF INT	[5,10,15,20,25]	
3	VAR	No Words To Write	INT	5	
4	VAR	Destination Addr	INT	0	Starting 16-bit address for storing data in the intelligent unit
5	VAR				

Body Five words of data defined in **CPUDataToWrt** are written into the addresses starting from 0 to 4 of the intelligent unit's shared memory (located in slot 0) when Start turns on.



LD



ST When programming with structured text, enter the following:

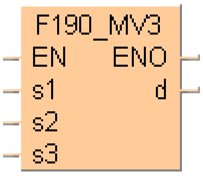
```
IF start THEN
    F151_WRT( s1_BankSlot:= SlotNo,
        s2_Start:= CPUDataToWrt[0],
        n_Number:= NoWordsToWrite,
        d_Start:= DestinationAddr);
END_IF;
```

F190_MV3

Three 16-bit data move

Description

The function copies three 16-bit data values at inputs **s1**, **s2** and **s3** to an ARRAY with three elements that is returned at output **d**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F190_MV3 **(see page 1322)**

☞ **To transfer two 16-bit data types, use either the F7_MV2 (see page 815) or P7_MV2 instruction.**

Data types

Variable	Data type	Function
s1, s2, s3	ANY16	source 16-bit area
d	ARRAY [0..2] of ANY16	destination, lower 16-bit area of 48-bit area

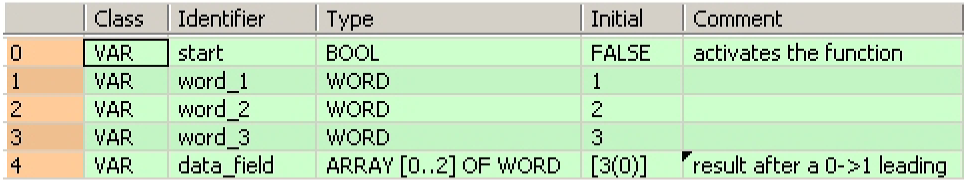
The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1,s2,s3	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

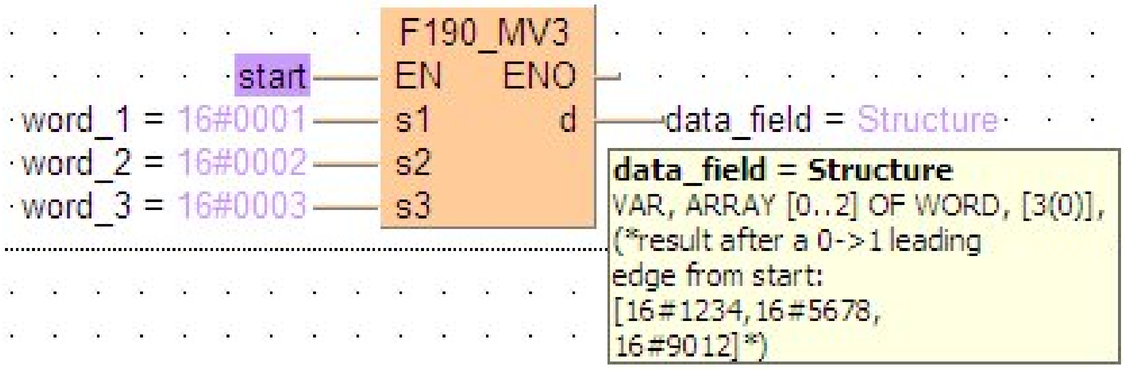
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	word_1	WORD	1	
2	VAR	word_2	WORD	2	
3	VAR	word_3	WORD	3	
4	VAR	data_field	ARRAY [0..2] OF WORD	[3(0)]	result after a 0->1 leading

Body When the variable **start** is set to TRUE, the function is carried out.

LD



data_field = Structure
VAR, ARRAY [0..2] OF WORD, [3(0)], (*result after a 0->1 leading edge from start: [16#1234, 16#5678, 16#9012] *)

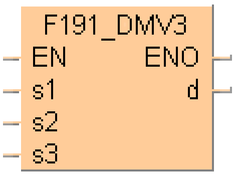
ST When programming with structured text, enter the following:

```
IF start THEN
    F190_MV3(word_1, word_2, word_3, data_field);
END_IF;
```

F191_DMV3

Three 32-bit data move

Description The function copies three 32-bit data values at inputs **s1**, **s2** and **s3** to an ARRAY with three elements that is returned at output **d**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F191_DMV3 **(see page** 1322**)**

☞ **To transfer two 32-bit data types, use either the F8_DMV2 (see page 817) or P8_DMV2 instruction.**

Data types

Variable	Data type	Function
s1, s2, s3	ANY32	source 32-bit area
d	ARRAY [0..2] ofANY32	destination, lower 32-bit area of 96-bit area

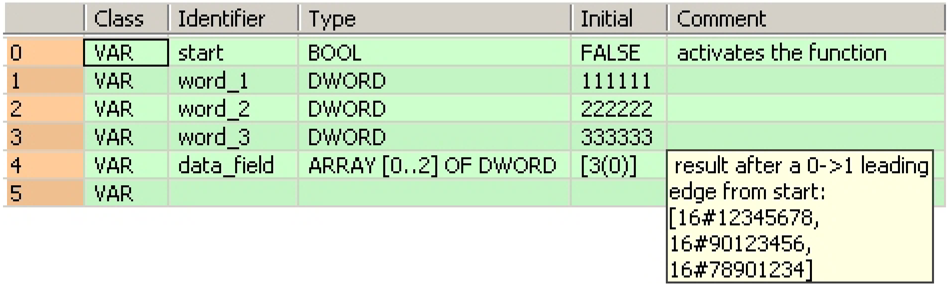
The variables **s1, s2, s3** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s1,s2,s3	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

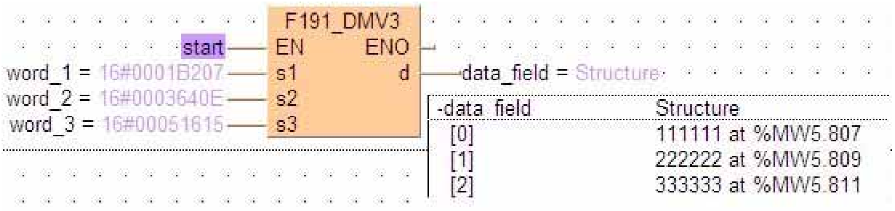
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	word_1	DWORD	111111	
2	VAR	word_2	DWORD	222222	
3	VAR	word_3	DWORD	333333	
4	VAR	data_field	ARRAY [0..2] OF DWORD	[3(0)]	result after a 0->1 leading edge from start: [16#12345678, 16#90123456, 16#78901234]
5	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



F191_DMV3

start — EN ENO

word_1 = 16#0001B207 — s1 d — data_field = Structure

word_2 = 16#0003640E — s2

word_3 = 16#00051615 — s3

-data_field	Structure
[0]	111111 at %MW5.807
[1]	222222 at %MW5.809
[2]	333333 at %MW5.811

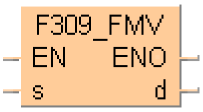
ST When programming with structured text, enter the following:

```
IF start THEN
    F191_DMV3(word_1, word_2, word_3, data_field);
END_IF;
```

F309_FMV

Floating Point Data Move

Description The floating point data (32 bits) specified by **s** is copied to the 32-bit area specified by **d** when the trigger turns on. The range of real number data which can be set is as follows:



- Positive: 0.0000001 to 9999999.0
- Negative: -9999999.0 to -0.000001

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Instead of using this FP instruction, we recommend using the related IEC instruction MOVE (see page 59). Please refer also to Advantages of the IEC instructions in the online help.

PLC types **Availability of** F309_FMV **(see page 1324)**

☞ **This instruction cannot be programmed in the interrupt program.**

Data types

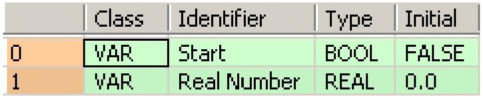
Variable	Data type	Function
s	Floating point constant	Floating point data, 32 bits (source).
d	REAL	32-bit area for result (destination).

Operands

For	Relay				T/C		Register			Constant
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

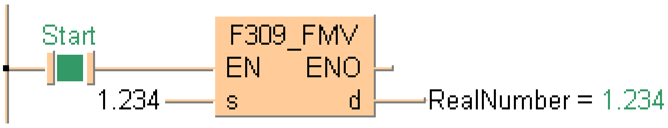
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Start	BOOL	FALSE
1	VAR	Real Number	REAL	0.0

Body When the variable **Start** is set to TRUE, the floating point data entered at s is copied to the 32-bit address assigned by the compiler for the variable **RealNumber**. The monitor value icon is activated.

LD

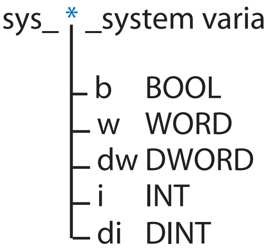


24.1 Data transfer to and from special data registers

FPWIN Pro offers three possibilities to read from or write to special relays/special data registers.

1. Via system variables (recommended from version 5.1 onwards)

   For each special data register and relay a system variable exists according to the following syntax:

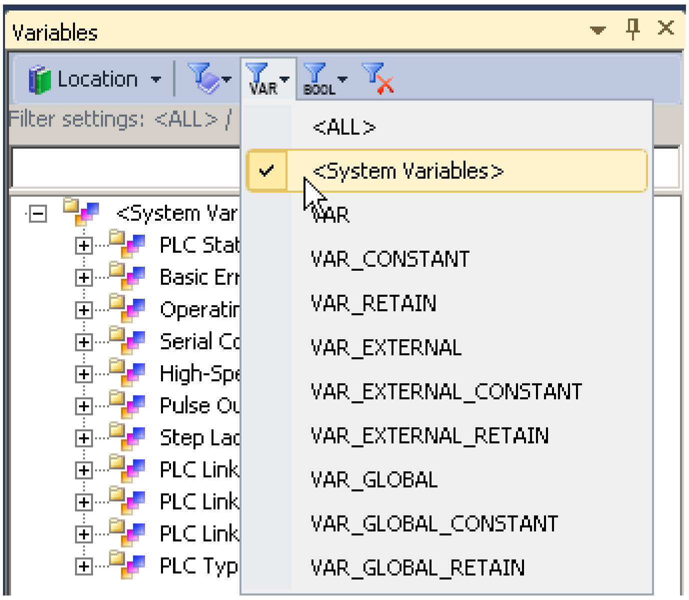


sys_ * _system variable

- b BOOL
- w WORD
- dw DWORD
- i INT
- di DINT

   You can insert these system variables into the body via the "Variables" dialog.

   Tip: Set the class filter to <System Variables> to display system variables only.



   In addition these system variables are also displayed under **Monitor** → **Special Relays and Registers** as the last entries in the comments, e.g. "**sys_w_HSC_ControlFlags**".

   Example for accessing the special data for HSC

   Example for accessing the special data for the RTC

2. via global variables
3. via direct addresses in the body

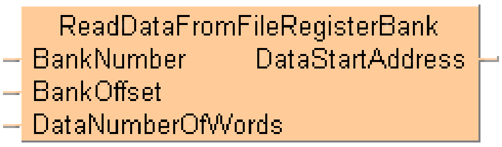
24.2 Transferring data to and from file register banks 1 or 2

In this section:

ReadDataFromFile RegisterBank

Read Data from File Register Bank 1 or 2

Description This instruction reads the number of words specified by **DataNumberOfWords** from File Register Bank 1 or 2, as specified by **BankNumber** beginning with **BankOffset**, and writes it to **DataStartAddress**.



With this function you cannot read data in the FL area (File Register Bank 0), i.e., the variable applied at **DataStartAddress** must not be located in the FL area.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** ReadDataFromFileRegisterBank **(see page 1330)**

Data types

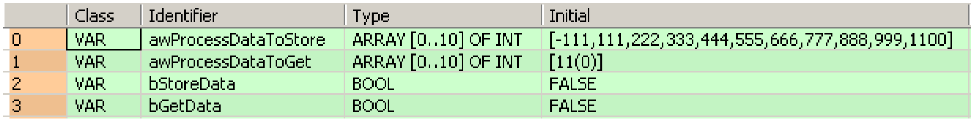
Variable	Data type	Function
BankNumber	INT	Specifies the bank number
BankOffset		Specifies the bank number offset
DataNumberOfWords		Number of word units to be read from the file register bank
DataStartAddress	ANY16	Specifies the start address of data which is read from the file register bank

Operands

For	Relay				T/C		Register			Constant
BankNumber	WX	WY	WR	WL	-	-	DT	LD	FL	dec. or hex
BankOffset	WX	WY	WR	WL	-	-	DT	LD	FL	dec. or hex.
DataNumberOfWords	WX	WY	WR	WL	-	-	DT	LD	FL	-
DataStartAddress	WX	WY	WR	WL	-	-	DT	LD	-	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

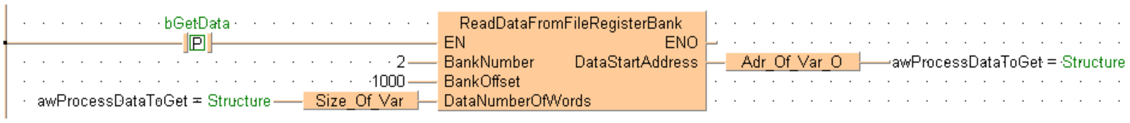
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	awProcessDataToStore	ARRAY [0..10] OF INT	[-111,111,222,333,444,555,666,777,888,999,1100]
1	VAR	awProcessDataToGet	ARRAY [0..10] OF INT	[11(0)]
2	VAR	bStoreData	BOOL	FALSE
3	VAR	bGetData	BOOL	FALSE

Body If **bGetData** changes from FALSE to TRUE, the entire data unit variable **awProcessDataToGet** (a DUT containing 11 elements) is filled with the data from File Register Bank 2 BankOffset 1000.

LD



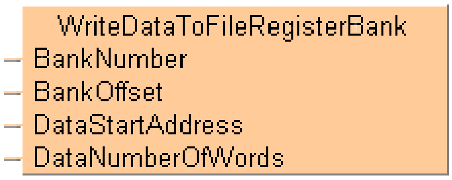
ST When programming with structured text, enter the following:

```
if (DF(bGetData)) then
      ReadDataFromFileRegisterBank(BankNumber := 2,
      BankOffset := 1000,
      DataNumberOfWords := Size_Of_Var(awProcessDataToGet),
      DataStartAddress => Adr_Of_Var(awProcessDataToGet));
end_if;
```

WriteDataToFile RegisterBank

Write Data to File Register Bank 1 or 2

Description This instruction reads the number of words specified by **DataNumberOfWords** from **DataStartAddress** and writes it to the File Register Bank 1 or 2 as specified by **BankNumber** beginning with **BankOffset**.


WriteDataToFileRegisterBank
BankNumber
BankOffset
DataStartAddress
DataNumberOfWords

With this function you cannot write data to the FL area (File Register Bank 0), i.e., the variable applied at **DataStartAddress** must not be located in the FL area.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

PLC types **Availability of** WriteDataToFileRegisterBank **(see page 1333)**

Data types

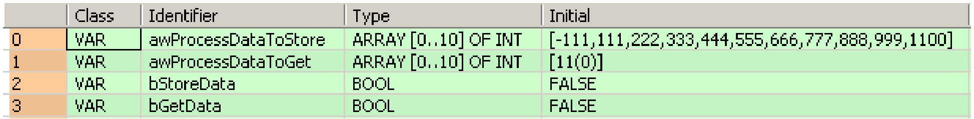
Variable	Data type	Function
BankNumber	INT	Specifies bank number
BankOffset		Specifies bank number offset
DataStartAddress	ANY16	Specifies start address of data to be written to File Register Bank
DataNumberOfWords	INT	Specifies number of word units to be written to File Register Bank

Operands

For	Relay				T/C		Register			Constant
BankNumber	WX	WY	WR	WL	-	-	DT	LD	FL	dec. or hex
BankOffset	WX	WY	WR	WL	-	-	DT	LD	FL	dec. or hex.
DataStartAddress	WX	WY	WR	WL	-	-	DT	LD	-	-
DataNumberOfWords	WX	WY	WR	WL	-	-	DT	LD	FL	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

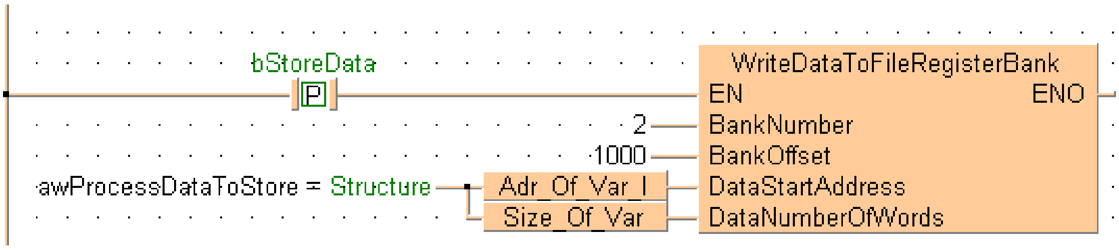
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	awProcessDataToStore	ARRAY [0..10] OF INT	[-111,111,222,333,444,555,666,777,888,999,1100]
1	VAR	awProcessDataToGet	ARRAY [0..10] OF INT	[11(0)]
2	VAR	bStoreData	BOOL	FALSE
3	VAR	bGetData	BOOL	FALSE

Body If **bStoreData** changes from FALSE to TRUE, the entire data unit variable **awProcessDataToStore** (a DUT containing 11 elements) is filled with the data from File Register Bank 2 BankOffset 1000.

LD



ST When programming with structured text, enter the following:

```
if (DF(bStoreData)) then
      WriteDataToFileRegisterBank(BankNumber := 2,
      BankOffset := 1000,
      DataStartAddress := Adr_Of_Var(awProcessDataToStore),
      DataNumberOfWords := Size_Of_Var(awProcessDataToStore));
end_if;
```

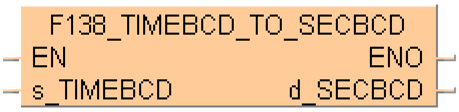
Chapter 25

Date and time instructions

F138_TIMEBCD_TO _SECBCD

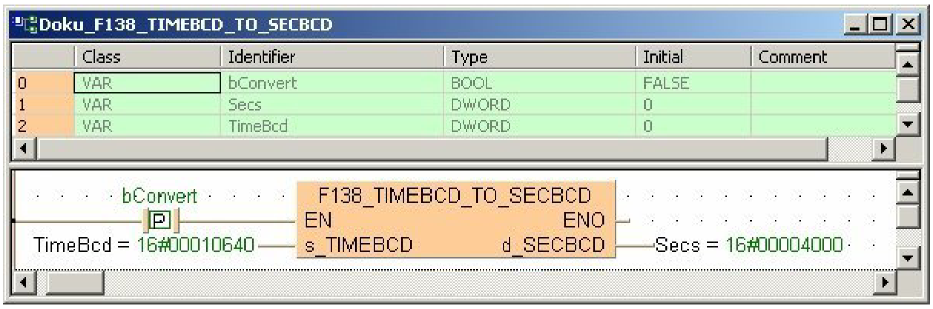
h:min:s -> s conversion

Description Converts the hours, minutes, and seconds data stored in the 32-bit area specified by **s_TIMEBCD** to seconds data if the trigger **EN** is in the ON-state.



The converted seconds data is stored in the 32-bit area specified by **d_SECBCD**. All hours, minutes, and seconds data to convert and the converted seconds data is BCD. The max. data input value is 9,999 hours, 59 minutes and 59 seconds, which will be converted to 35,999,999 seconds in BCD format.

Example



Doku_F138_TIMEBCD_TO_SECBCD

	Class	Identifier	Type	Initial	Comment
0	VAR	bConvert	BOOL	FALSE	
1	VAR	Secs	DWORD	0	
2	VAR	TimeBcd	DWORD	0	

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F138_TIMEBCD_TO_SECBCD **(see page 1321)**

Data types

Variable	Data type	Function
s_TIMEBCD	DWORD	source area for storing hours, minutes and seconds data
d_SECBCD	DWORD	destination area for storing converted seconds data

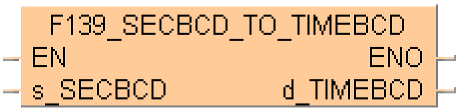
Operands

For	Relay				T/C		Register			Const.
s_TIMEBCD	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
d_SECBCD	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

F139_SECBCD_TO_TIMEBCD

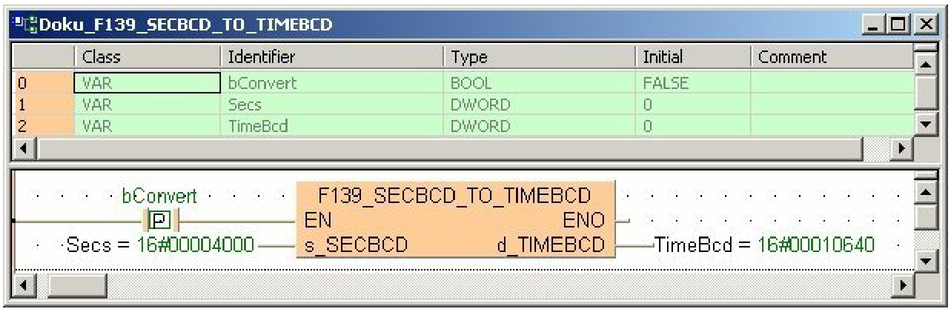
s -> h:min:s conversion

Description Converts the second data stored in the 32-bit area specified by **s** to hours, minutes, and seconds data if the trigger **EN** is in the ON-state.



The converted hours, minutes, and seconds data is stored in the 32-bit area specified by **d**. The seconds prior to conversion and the hours, minutes, and seconds after conversion are all BCD data. The maximum data input value is 35,999,999 seconds, which is converted to 9,999 hours, 59 minutes and 59 seconds.

Example


Doku_F139_SECBCD_TO_TIMEBCD

	Class	Identifier	Type	Initial	Comment
0	VAR	bConvert	BOOL	FALSE	
1	VAR	Secs	DWORD	0	
2	VAR	TimeBcd	DWORD	0	

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F139_SECBCD_TO_TIMEBCD **(see page** 1321**)**

Data types

Variable	Data type	Function
s_SECBCD	DWORD	source area for storing seconds data
d_TIME_BCD	DWORD	destination area for storing converted hours, minutes and seconds data

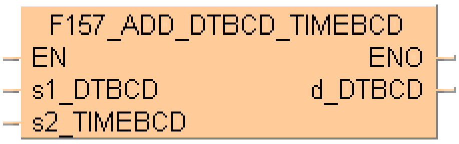
Operands

For	Relay				T/C		Register			Const.
s_SECBCD	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-
d_TIME_BCD	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

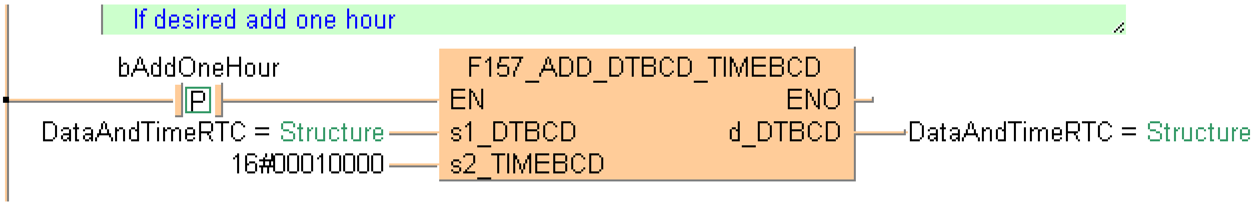
F157_ADD_DTBCD _TIMEBCD

Time addition

Description The date/clock data (3 words) specified by **s1_DTBCD** and the time data (2 words) specified by **s2_TIMEBCD** are added together if the trigger **EN** is in the ON-state. The result is stored in the area (3 words, same format as **s1_DTBCD**) specified by **d_DTBCD**. This instruction handles all data in BCD format.



Example:



☞ **You cannot specify special data registers DT9054 to DT9056 (DT90054 to DT90056 for FP2/2SH and FP10/10S/10SH) for the operand d_DTBCD. These registers are factory built-in calendar timer values. To change the built-in calendar timer value, first store the added result in other memory areas and transfer them to the special data registers using SET_RTC_DTBCD (see page 874) instruction.**

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Example 1: clock/calendar data in DTBCD format	**DUT Member**	**Result**
August 1, 1992, Time: 14:23:31 (hours:minutes:seconds)	MinSec	16#2331 (minutes/seconds)
	DayHour	16#0114 (day/hour)
	YearMon	16#9208 (year/month)
Example 2: time data in TIMEBCD format		
32 hours; 50 minutes; and 45 seconds		16#00325045 hex (hours/minutes/seconds)

PLC types **Availability of** F157_ADD_DTBCD_TIMEBCD **(see page 1321)**

Data types

Variable	Data type	Function
s1_DTBCD	DTBCD	augend, time and date, values in BCD format
s2_TIMEBCD	DWORD	addend, 32-bit area for storing time data in BCD format
d_DTBCD	DTBCD	sum in BCD format

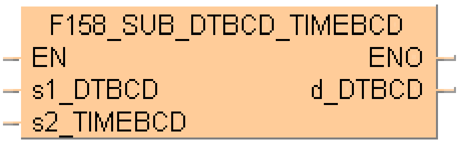
Operands

For	Relay				T/C		Register			Const.
s1_DTBCD	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2_TIMEBCD	-	WY	WR	WL	SV	EV	DT	LD	FL	-
d_DTBCD	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

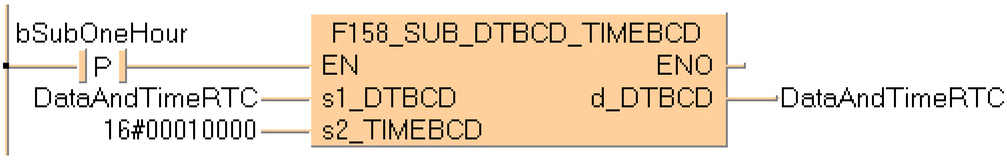
F158_SUB_DTBCD _TIMEBCD

Time subtraction

Description Subtracts time data (2 words) specified by **s2_TIMEBCD** from the date/clock data (3 words) specified by **s1_DTBCD** if the trigger **EN** is in the ON-state. The result is stored in the area (3 words, same format than **s1_DTBCD**) specified by **d_DTBCD**. All the data used in this instruction are handled in form of BCD.



Example:



☞ **You cannot specify special data registers DT9054 to DT9056 (DT90054 to DT90056 for FP2/2SH and FP10/10S/10SH) for the operand d_DTBCD. These registers are factory built-in calendar timer values. To change the built-in calendar timer value, first store the subtraction result in other memory areas and transfer them to the special data registers using SET_RTC_DTBCD (see page 874) instruction.**

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Example 1: clock/calendar data in DTBCD format	DUT Member	Result
August 1, 1992, Time: 14:23:31 (hour:minutes:seconds)	MinSec	16#2331 (minutes/seconds)
	DayHour	16#0114 (day/hour)
	YearMon	16#9208 (year/month)
Example 2: time data in TIMEBCD format		
32 hours; 50 minutes; and 45 seconds		16#00325045 hex (hours/minutes/seconds)

PLC types **Availability of** F158_SUB_DTBCD_TIMEBCD **(see page 1321)**

Data types

Variable	Data type	Function
s1_DTBCD	DTBCD	minuend, time and date, values in BCD format
s2_TIMEBCD	DWORD	subtrahend, 32-bit area for storing time data in BCD format
d_DTBCD	DTBCD	result in BCD format

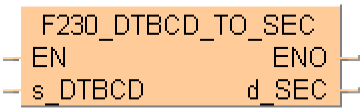
Operands

For	Relay				T/C		Register			Const.
s1_DTBCD	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
s2_TIMEBCD	-	WY	WR	WL	SV	EV	DT	LD	FL	-
d_DTBCD	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

F230_DTBCD_TO_SEC

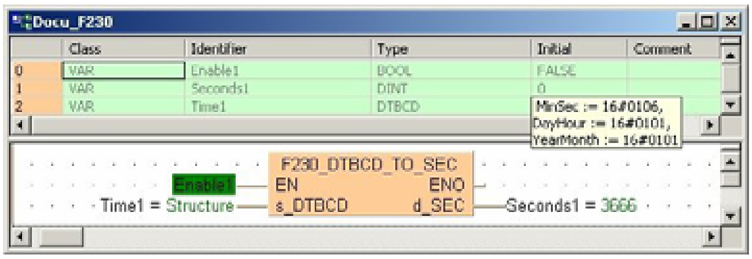
Time Data Conversion into Seconds

Description This function converts time data (date and time) into the number of seconds. It calculates the time span between the specified time date and 01/01/2001 at 00:00 hours. The time data is specified in the DUT "**DTBCD**".



For a conversion from seconds into time data, please refer to **F231_SEC_TO_DTBCD** (see page 872).

Example:



	Class	Identifier	Type	Initial	Comment
0	VAR	Enable1	BOOL	FALSE	
1	VAR	Seconds1	DINT	0	
2	VAR	Time1	DTBCD		

MinSec := 16#0106,
DayHour := 16#0101,
YearMonth := 16#0101

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F230_DTBCD_TO_SEC **(see page 1323)**

Data types

Variable	Data type	Function
s_DTBCD	DTBCD	Area in which the input time data is stored
d_SEC	ANY32	Area in which the converted second information is stored (32 bits)

Operands

For	Relay				T/C		Register			Constant
s_DTBCD	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
d_SEC	-	WY	WR	WL	SV	EV	DT	LD	FL	-

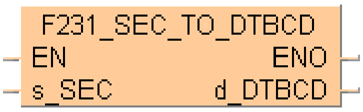
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the specified address using the index modifier exceeds a limit. ▪ values other than BCD are specified for 's'. ▪ the value which exceeds the range in the time data of 's' is specified.
R9008	%MX0.900.8	for an instant	

F231_SEC_TO_DTBCD

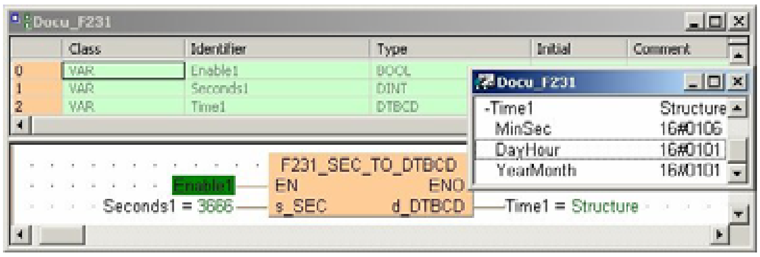
Conversion of Seconds into Time Data

Description This function converts a specified number of seconds into date and time. The time data is calculated from 01/01/2001 at 00:00 hours.



For a conversion from time data into seconds, please refer to F230_DTBCD_TO_SEC (see page 871).

Example:



	Class	Identifier	Type	Initial	Comment
0	VAR	Enable1	BOOL		
1	VAR	Seconds1	DINT		
2	VAR	Time1	DTBCD		

-Time1	Structure
MinSec	16#0106
DayHour	16#0101
YearMonth	16#0101

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of F231_SEC_TO_DTBCD (see page 1323)**

Data types

Variable	Data type	Function
s_SEC	ANY32	Area in which the number of seconds are stored (32 bits)
d_DTBCD	DTBCD	Head area in which time data is stored

Operands

For	Relay				T/C		Register			Constant
s_SEC	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
d_DTBCD	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the specified address using the index modifier exceeds a limit. ▪ the number of seconds (s) >= 16#BC191380 (valid until 31 Dec. 2100 23:59:59). ▪ the data memory of 'd' exceeds the area.
R9008	%MX0.900.8	for an instant	

GET_RTC_DTBCD

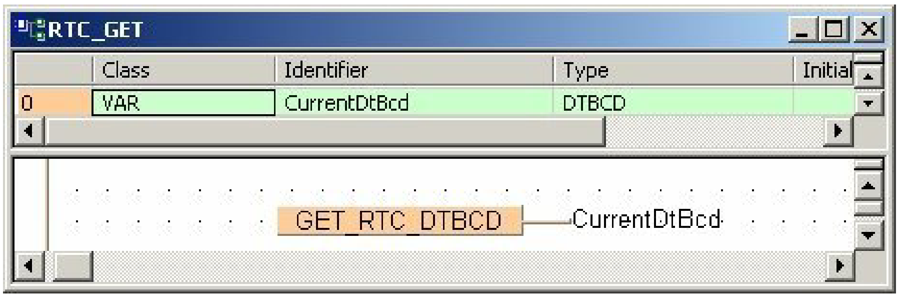
Read the Real-Time Clock

Description Use this PLC independent instruction to read the real-time clock data from the PLC. When the instruction is carried out, the values from the special data registers DT90054 to DT90056 (DT9054 to DT9056) are transferred to the data unit type DTBCD. You can also use the system variables to set the RTC. For detailed information on using system variables, please refer to data transfer to and from special data registers (see page 859).

GET_RTC_DTBCD

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Example:



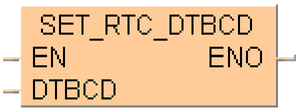
RTC_GET

	Class	Identifier	Type	Initial
0	VAR	CurrentDtBcd	DTBCD	

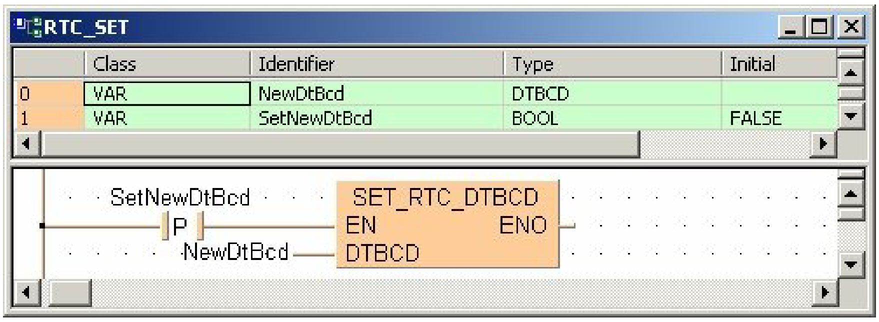
SET_RTC_DTBCD

Set the Real-Time Clock

Description Use this PLC independent instruction to write date and time data in BCD format (DTBCD) to the real-time clock. When the variable **SetNewDtBcd** is set to TRUE, the values from the data unit type DTBCD are transferred to the special data registers DT90054 to DT90056 (DT9054 to DT9056) and the value 16#8000 is written to the special data register DT90058 (DT9058) to set the real-time clock of the PLC. You can also use the system variables to set the RTC. For detailed information on using system variables, please refer to data transfer to and from special data registers (see page 859).



Example



	Class	Identifier	Type	Initial
0	VAR	NewDtBcd	DTBCD	
1	VAR	SetNewDtBcd	BOOL	FALSE

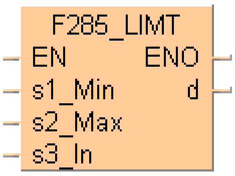
Chapter 26

Selection Instructions

F285_LIMT

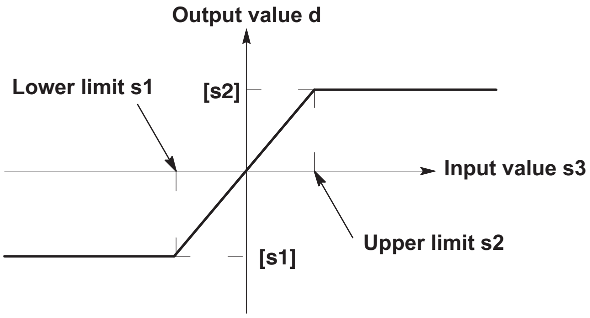
16-bit data upper and lower limit control

Description The function compares the input value at input **s3_In** with a lower and an upper limit. The lower limit is specified at input **s1_Min**, and the upper limit at input **s2_Max**. The result of the function is returned at output **d** as follows.



- If the input value at **s3_In** < **s1_Min**, the lower limit at input **s1_Min** is returned at output **d**.
- If the input value at **s3_In** > **s2_Max**, the upper limit at input **s2_Max** is returned at output **d**.
- If the input value at **s2_Max** ≥ **s3_In** ≥ **s1_Min**, the input value **s3_In** is returned unchanged at output **d**.

If you want to control the output value solely via the upper value **s2_Max**, set -32768 or 16#8000 for the lower limit **s1_Min**. To perform lower limit control only, set 32767 or 16#7FFF for the upper limit **s2_Max**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F285_LIMT **(see page 1323)**

Data types

Variable	Data type	Function
s1_Min	ANY16	the area where the lower limit is stored or the lower limit data
s2_Max		the area where the upper limit is stored or the upper limit data
s3_In		the area where the input value is stored or the input value data
d		the area where the output value data is stored

Operands

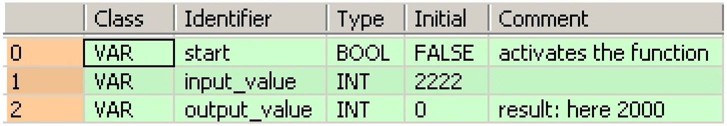
For	Relay				T/C		Register			Constant
s1_Min, s2_Max, s3_In	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the value at **s1_Min** > **s2_Max**
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	▪ the result of processing is between the upper and lower limits.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

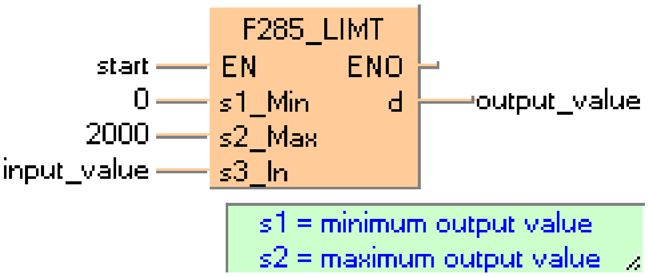


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	INT	2222	
2	VAR	output_value	INT	0	result: here 2000

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out. The constant 0 (lower limit) and 2000 (upper limit) are assigned to inputs s1 and s2. However, you can declare variables in the POU header and write them in the function in the body at the inputs.

LD



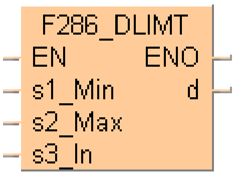
ST When programming with structured text, enter the following:

```
IF start THEN
   F285_LIMT( 0, 2000, input_value, output_value);
END_IF;    (* 0=lower limit, 2000=upper limit *)
```

F286_DLIMT

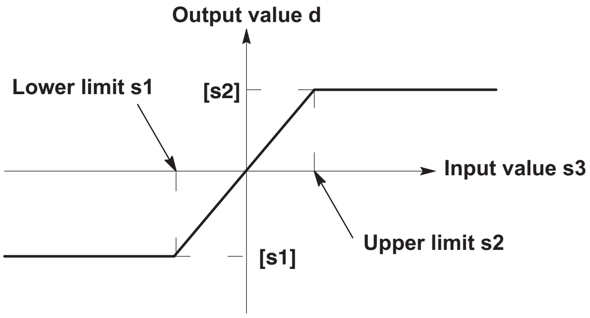
32-bit data upper and lower limit control

Description The function compares the input value at input **s3_In** with a lower and an upper limit. The lower limit is specified at input **s1_Min**, and the upper limit at input **s2_Max**. The result of the function is returned at output **d** as follows:



- If the input value at **s3_In** < **s1_Min**, the lower limit at input **s1_Min** is returned at output **d**.
- If the input value at **s3_In** > **s2_Max**, the upper limit at input **s2_Max** is returned at output **d**.
- If the input value at **s2_Max** ≥ **s3_In** ≥ **s1_Min**, the input value **s3_In** is returned unchanged at output **d**.

If you want to control the output value solely via the upper value **s2_Max**, set -2147483648 or 16#80000000 for the lower limit **s1_Min**. To perform lower limit control only, set 2147483647 or 16#7FFFFFFF the upper limit **s2_Max**.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of F286_DLIMT (see page 1323)**

Data types

Variable	Data type	Function
s1_Min	ANY32	the area where the lower limit is stored or the lower limit data
s2_Max		the area where the upper limit is stored or the upper limit data
s3_In		the area where the input value is stored or the input value data
d		the area where the output value data is stored

Operands

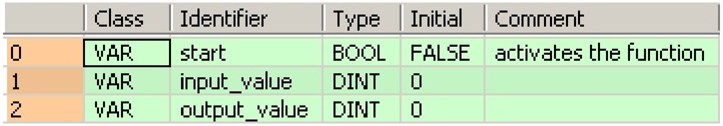
For	Relay				T/C		Register			Constant
s1_Min, s2_Max, s3_In	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	-	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the value at **s1_Min** > **s2_Max**
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	▪ the result of processing is between the upper and lower limits.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header All input and output variables used for programming this function have been declared in the POU header.

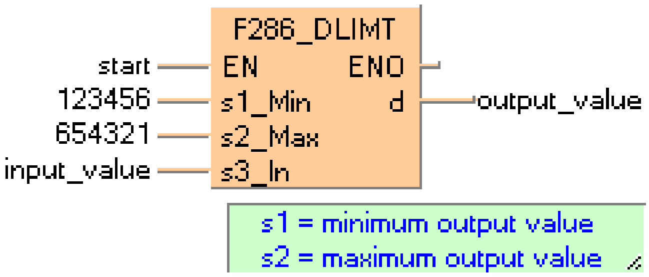


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	DINT	0	
2	VAR	output_value	DINT	0	

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

Body When the variable **start** is set to TRUE, the function is carried out. The constant -123456 (lower limit) and 654321 (upper limit) are assigned to inputs s1 and s2. However, you can declare variables in the POU header and write them in the function in the body at the inputs.

LD



ST When programming with structured text, enter the following:

```
IF start THEN
   F286_DLIMT( 123456, 654321, input_value, output_value);
END_IF;      (* 123456= lower limit, 654321=upper limit *)
```

Chapter 27

Edge detection instructions

DF

Rising edge differential

Description **DF** is a rising edge differential instruction. The **DF** instruction executes and turns ON output **o** for a singular scan duration if the trigger **i** changes from an OFF to an ON state.

PLC types **Availability of** DF **(see page 1319)**

Be careful when programming with commands that effect the order in which a program is carried out, e.g. jump or loop instructions within a sequential function chart or a function block. The order of the instructions might change depending on the time when the instruction is carried out or the input value. Specific basic JUMP and LOOP instructions are:

- MC (see page 1007) to MCE (see page 1008)
- JP (see page 1009) to LBL (see page 1013)
- F19_SJP (see page 1010) to LBL (see page 1013)
- LOOP (see page 1012) to LBL (see page 1013)

Data types

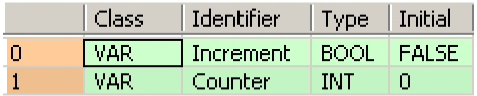
Variable	Data type
input	BOOL
output	BOOL

Operands

For	Relay				T/C		Register			Constant
i	X	Y	R	L	T	C	-	-	-	-
o	-	Y	R	L	-	-	-	-	-	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

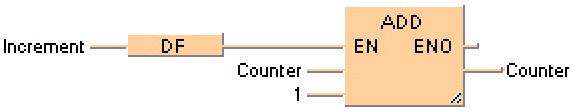
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Increment	BOOL	FALSE
1	VAR	Counter	INT	0

Body Each rising edge at the input **Increment** increments the counter.

LD



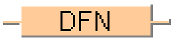
ST When programming with structured text, enter the following:

```
IF DF(Increment) THEN
    Counter:=Counter+1;
END_IF;
```

DFN

Falling edge differential

Description The **DFN** instruction executes and turns ON output **o** for a single scan duration if the trigger **i** changes from an ON to an OFF state.



PLC types **Availability of** DFN **(see page** 1319**)**

☞ **Be careful when programming with commands that effect the order in which a program is carried out, e.g. jump or loop instructions within a sequential function chart or a function block. The order of the instructions might change depending on the time when the instruction is carried out or the input value. Specific basic JUMP and LOOP instructions are:**

- MC (see page 1007) to MCE (see page 1008)
- JP (see page 1009) to LBL (see page 1013)
- F19_SJP (see page 1010) to LBL (see page 1013)
- LOOP (see page 1012) to LBL (see page 1013)

Data types

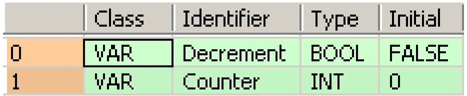
Variable	Data type
input	BOOL
output	BOOL

Operands

For	Relay				T/C		Register			Constant
i	X	Y	R	L	T	C	-	-	-	-
o	-	Y	R	L	-	-	-	-	-	-

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

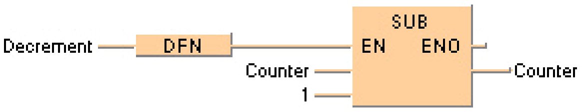
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Decrement	BOOL	FALSE
1	VAR	Counter	INT	0

Body Each falling edge at the input **Decrement** decrements the couter.

LD



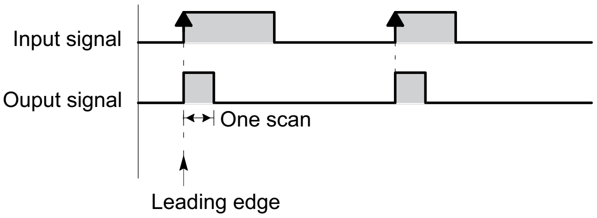
ST When programming with structured text, enter the following:

```
IF DFN(Decrement) THEN
    Counter:=Counter-1;
END_IF;
```

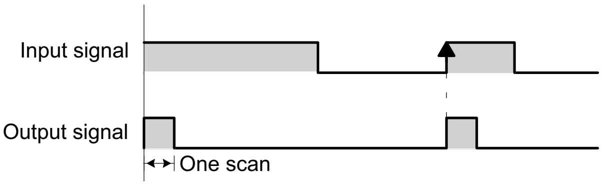
DFI

Rising edge differential (initial execution type)

Description When a rising edge of the input signal (input **i**) is detected, this function changes the status of the output signal (output **o**) to TRUE for the duration of the scan.



Detection of the input signal’s rising edge is also assured at the first scan.



You may use an unlimited number of DFI functions.

If the input signal = TRUE already when the system is turned on and this signal should not be interpreted as the first rising edge, the DF function must be used instead.

PLC types **Availability of** DFI **(see page** 1319**)**

Be careful when programming with commands that effect the order in which a program is carried out, e.g. jump or loop instructions within a sequential function chart or a function block. The order of the instructions might change depending on the time when the instruction is carried out or the input value. Specific basic JUMP and LOOP instructions are:

- MC (see page 1007) to MCE (see page 1008)
- JP (see page 1009) to LBL (see page 1013)
- F19_SJP (see page 1010) to LBL (see page 1013)
- LOOP (see page 1012) to LBL (see page 1013)

Data types

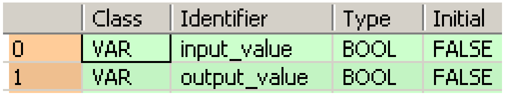
Variable	Data type
input	BOOL
output	BOOL

Operands

For	Relay				T/C		Register			Constant
i	X	Y	R	L	T	C	-	-	-	-
o	-	Y	R	L	-	-	-	-	-	-

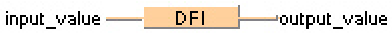
Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial
0	VAR	input_value	BOOL	FALSE
1	VAR	output_value	BOOL	FALSE

LD



ST When programming with structured text, enter the following:

```
output_value:=DFI(input_value);
```

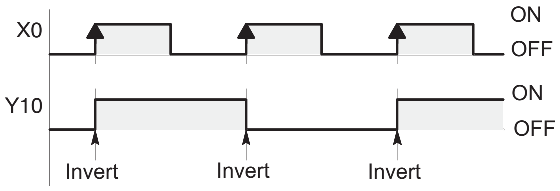
ALT

Alternative out

Description The function inverts the output condition (output **o**) each time the rising edge of the input signal (input **i**) is detected.

When the mode is changed from PROG to RUN or the power is turned on in RUN mode while the input signal is TRUE, a rising edge will not be detected for the first scan.

Time chart



PLC types **Availability of** ALT **(see page** 1318**)**

Be careful when programming with commands that effect the order in which a program is carried out, e.g. jump or loop instructions within a sequential function chart or a function block. The order of the instructions might change depending on the time when the instruction is carried out or the input value. (Specific basic JUMP and LOOP instructions are: MC to MCE instruction, JP to LBL instruction, F19_SJP to LBL instruction, LOOP to LBL instruction.

Data types

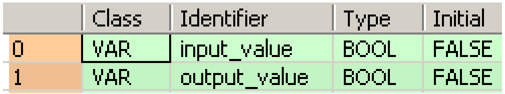
Variable	Data type
input	BOOL
output	BOOL

Operands

For	Relay				T/C		Register			Constant
i	X	Y	R	L	T	C	-	-	-	-
o	-	Y	R	L	-	-	-	-	-	-

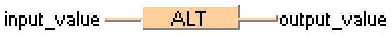
Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial
0	VAR	input_value	BOOL	FALSE
1	VAR	output_value	BOOL	FALSE

LD



ST When programming with structured text, enter the following:

```
output_value:=(ALT(input_value));
```

Chapter 28

High-speed counter instructions

28.1 Introduction

Control FPWIN Pro offers two concepts for programming with high-speed counter instructions:

- FP instructions
- Tool instructions

For users programming for different PLC types of the FP series or users who are tired of setting control code bits and looking up available channel numbers, the tool instructions offer new and comfortable features. These include information functions for evaluating status flags and settings, control functions for configuring high-speed counters and pulse outputs, PLC-independent functions and DUTs, as well as variable channel numbers. However, the FP instructions may be easier to use for beginners or users familiar with FPWIN GR.

Most of the information, which is accessible via information and control functions, is stored in special internal relays and special data registers. These relays and registers can also be accessed using PLC-independent system variables.

To take advantage of the features you prefer, the instructions of both libraries can be mixed.

◆NOTE

When programming with the tool instructions, be sure to refer to the detailed information provided via the links to the related F/P instructions.

Main features	FP instructions	Tool instructions
Pre version 6.4 support	●	
Use of inline functions	●	
Use of FPWIN GR function names	●	
Less code with constant channel numbers	●	
Control codes	●	
Control functions		●
Information functions		●
Variable channel numbers		●
Universal functions for all PLCs		●
DUT for common channel configuration for all PLCs for all pulse output instructions		●

28.2 Writing the high-speed counter control code

The special data register where the high-speed counter and pulse output control code are stored can be accessed with the system variable sys_wHscOrPulseControlCode. (The system variable sys_wHscOrPulseControlCode corresponds to special data register DT90052.)

Operations performed by the high-speed counter control code

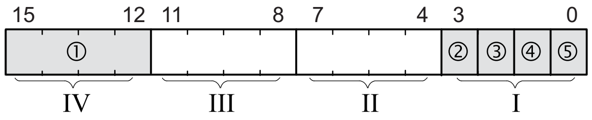
- Clearing high-speed counter instructions (bit 3)
- Enabling/disabling the reset input (hardware reset) of the high-speed counter (bit 2)
- Enabling/disabling counting operations (bit 1)
- Resetting the elapsed value (software reset) of the high-speed counter to 0 (bit 0)

The control code settings for each channel can be monitored using the system variables sys_wHscChannelxControlCode or sys_wPulseChannelxControlCode (where x=channel number).

The settings of this system variable remain unchanged until another setting operation is executed.

Description for FPΣ, FP-X, FP0R:

Bits 0–15 of the control code are allocated in groups of four. The bit setting in each group is represented by a hex number (e.g. 0002 0000 0000 1001 = 16#2009).



①	Channel number (channel n: 16#n)	
②	Clear high-speed counter instruction (bit 3)	
	0: continue	1: clear
③	Reset input (bit 2) (see note)	
	0: enabled	1: disabled
④	Count (bit 1)	
	0: permit	1: prohibit
⑤	Reset elapsed value to 0 (bit 0)	
	0: no	1: yes

Example: 16#2009

Group	Value	Description	
IV	2	Channel number: 2	
III	0	(fixed)	
II	0	(fixed)	
I	9	Hex 9 corresponds to binary 1001	
		Clear high-speed counter instruction: clear (bit 3)	1
		Reset input: enabled (bit 2)	0
		Count: permit (bit 1)	0
		Reset elapsed value to 0: yes (bit 0)	1

Description for FP0, FP-e:

Bits 0–15 of the control code are allocated in groups of four, each group containing the settings for one channel. The bit setting in each group is represented by a hex number (e.g. 0000 0000 1001 0000 = 16#90).

15			12	11			8	7			4	3			0
①	②	③	④	①	②	③	④	①	②	③	④	①	②	③	④
IV				III				II				I			

Group	IV	III	II	I
Channel	3	2	1	0

①	Clear high-speed counter instruction (bit 3)	
	0: continue	1: clear
②	Reset input (bit 2) (see note)	
	0: enabled	1: disabled
③	Count (bit 1)	
	0: permit	1: prohibit
④	Reset elapsed value to 0 (bit 0)	
	0: no	1: yes

Example: 16#90

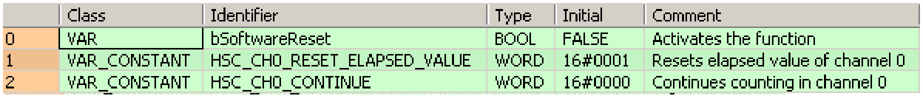
Group	Value	Description	
IV	0	–	
III	0	–	
II	9	Channel number: 1 Hex 9 corresponds to binary 1001	
		Clear high-speed counter instruction: clear (bit 3)	1
		Reset input: enabled (bit 2)	0
		Count: permit (bit 1)	0
		Reset elapsed value to 0: yes (bit 0)	1
I	0	–	

☞ **Turning the reset input to TRUE, sets the elapsed value to 0. Use the reset input setting (bit 2) to disable the reset input allocated in the system registers.**

Software reset for channel 0

Example 1 The first example shows how to perform a software reset for channel 0, and the second example shows how to perform a software reset for channel 1.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	bSoftwareReset	BOOL	FALSE	Activates the function
1	VAR_CONSTANT	HSC_CH0_RESET_ELAPSED_VALUE	WORD	16#0001	Resets elapsed value of channel 0
2	VAR_CONSTANT	HSC_CH0_CONTINUE	WORD	16#0000	Continues counting in channel 0

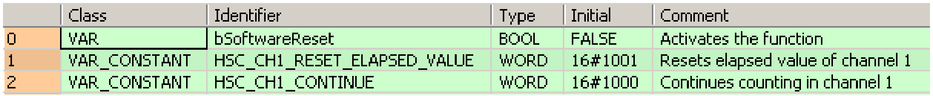
Body

The reset is performed in step 1, and 0 is entered just after that in step 2 to start counting. A reset alone does not start counting.

Software reset for channel 1

POU header (FPΣ, FP-X, FP0R)

All input and output variables used for programming this function have been declared in the POU header.

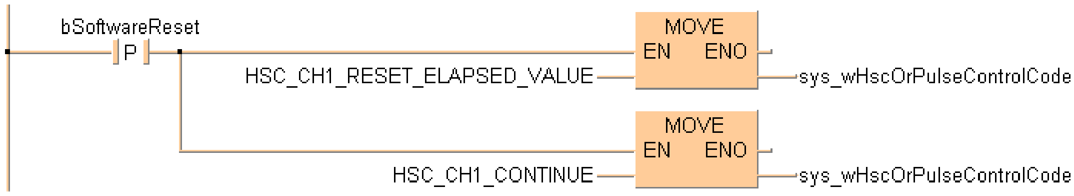


	Class	Identifier	Type	Initial	Comment
0	VAR	bSoftwareReset	BOOL	FALSE	Activates the function
1	VAR_CONSTANT	HSC_CH1_RESET_ELAPSED_VALUE	WORD	16#1001	Resets elapsed value of channel 1
2	VAR_CONSTANT	HSC_CH1_CONTINUE	WORD	16#1000	Continues counting in channel 1

Body

The reset is performed in step 1, and 0 is entered just after that in step 2 to start counting. A reset alone does not start counting.

LD


bSoftwareReset
P
MOVE
EN
ENO
HSC_CH1_RESET_ELAPSED_VALUE
sys_wHscOrPulseControlCode
MOVE
EN
ENO
HSC_CH1_CONTINUE
sys_wHscOrPulseControlCode

28.3 High-speed counter: writing and reading the elapsed value

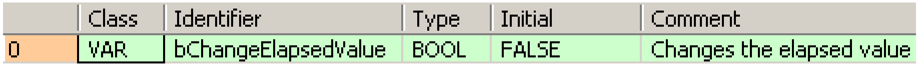
The elapsed value is stored as a double word in the special data registers. Access the special data registers using the system variable sys_diHscChannelxElapsedValue (where x=channel number).

System variables for memory areas used:

- FP-Sigma
- FP-X, Transistor types
- FP-X, Relay types
- FP0R
- FP0, FP-e

Example The first example shows how to write an initial value (elapsed value) into the high-speed counter. The second example shows how to read an elapsed value and copy it to a variable.

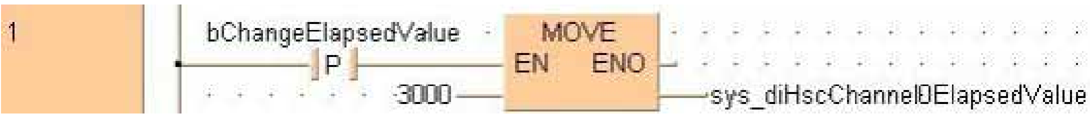
All input and output variables used for programming this function have been declared in the POU header.



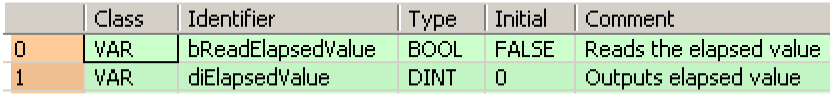
	Class	Identifier	Type	Initial	Comment
0	VAR	bChangeElapsedValue	BOOL	FALSE	Changes the elapsed value

Body An initial value of 3000 (elapsed value) is written into channel 0 of the high-speed counter.

LD



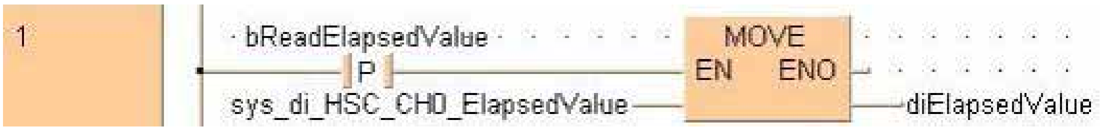
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	bReadElapsedValue	BOOL	FALSE	Reads the elapsed value
1	VAR	diElapsedValue	DINT	0	Outputs elapsed value

Body The elapsed value of the high-speed counter is read from channel 0 of the high-speed counter and copied to the variable diElapsedValue.

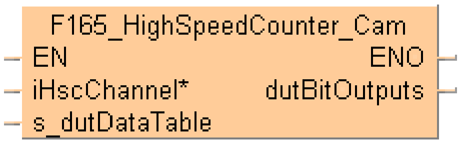
LD



F165_HighSpeedCounter_Cam

Cam control

Description This instruction performs cam control according to the parameters in the specified DUT with a maximum of 31 target values for the high-speed counter. An interrupt program can be executed whenever the elapsed value matches one of the target values.

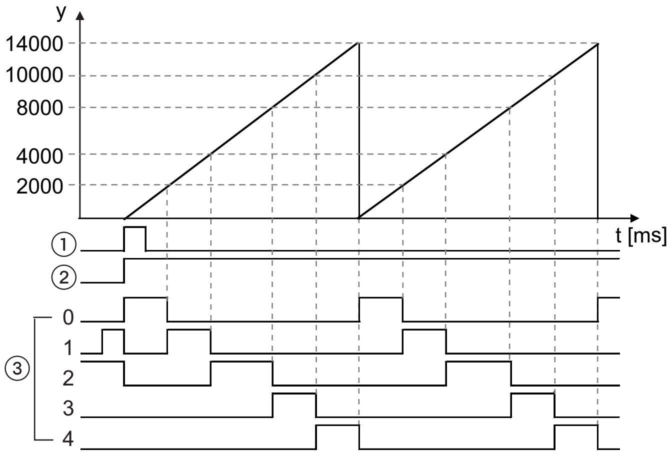


Create your own DUT using the following DUT as a sample: F165_HighSpeedCounter_Cam_8_Values_DUT

The following parameters can be specified in the DUT:

- Control code
- Word address for output relays
- Number of target values
- Target value 1
- ...
- Target value n
- Maximum target value

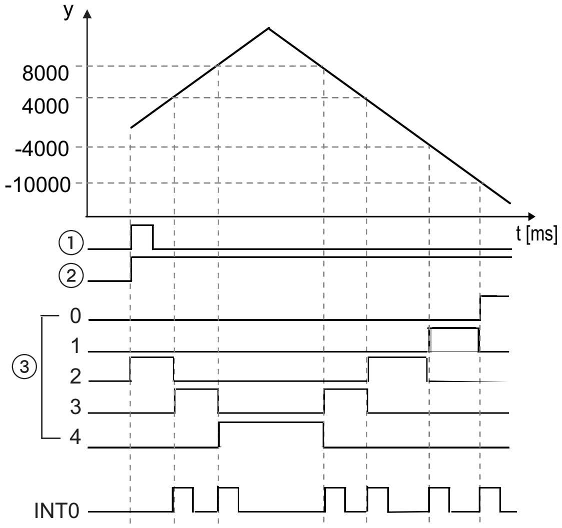
Characteristics of cam control



y	Elapsed value of high-speed counter	14000	Maximum target value
①	Execution condition	10000	Target value 4
②	High-speed counter control flag	8000	Target value 3
③	Output relay 0-4	4000	Target value 2
		2000	Target value 1

- Whenever the elapsed value is in the target value area n to n+1 (incremental counting) or n+1 to n, (decremental counting), the corresponding output relay n is TRUE.
- In the example above, maximum target value control has been enabled. When the elapsed value reaches the maximum target value, the elapsed value is reset to 0 and counting restarts.
- Specify the word address of the output relays in an overlapping DUT, e.g. BOOL32_OVERLAPPING_DUT, and apply this DUT at dutBitOutputs.

- A maximum of 31 target values can be specified.
- The target values must be arranged in ascending order. No value may be used twice.
- When the instruction starts, all output relays are FALSE, except for output relay 0, which turns to TRUE, provided that the elapsed value is smaller than target value 1. Otherwise, the output relay corresponding to the target value area turns to TRUE. Example: If the current value is between target value 2 = -4000 and target value 3 = +4000, output relay 2 is TRUE. In the following example maximum target value control has been disabled. When the elapsed value reaches the last target value, counting continues and the elapsed value is not reset to 0.



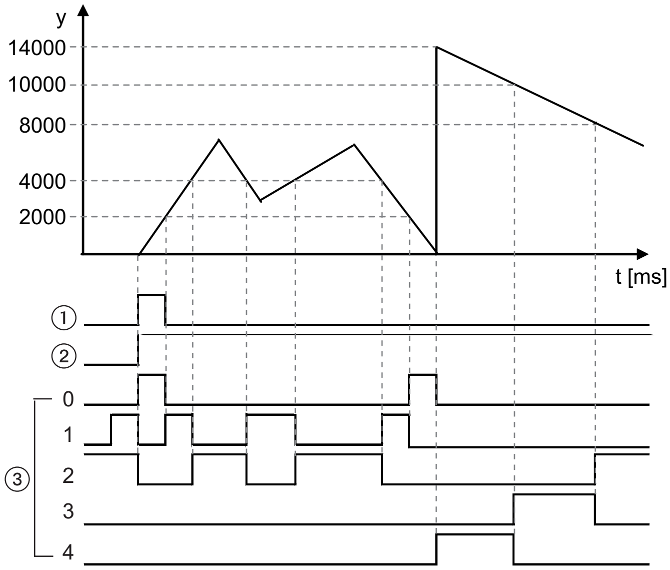
y	Elapsed value of high-speed counter	8000	Target value 4
①	Execution condition	4000	Target value 3
②	High-speed counter control flag	-4000	Target value 2
③	Output relay 0-4	-10000	Target value 1
INT0	Interrupt program 0		

Maximum target value control

The instruction can be executed using maximum target value control to reset the elapsed value to 0 when the maximum target value has been reached. Maximum target value control can be enabled in the control code of F165_HighSpeedCounter_Cam_8_Values_DUT. Instead of using maximum target value control, the elapsed value can also be reset using a reset input or a software reset (see page 1021).

To perform maximum target value control, positive integer numbers must be specified for all target values.

Incremental and decremental counting with maximum target value control:



y	Elapsed value of high-speed counter	14000	Maximum target value
①	Execution condition	10000	Target value 4
②	High-speed counter control flag	8000	Target value 3
③	Output relay 0-4	4000	Target value 2
		2000	Target value 1

Overview:

Maximum target value control:	**enabled**	**disabled (see note)**
Incremental counting: The pointer of the data table moves from target value 1 to the last target value.	When the elapsed value reaches the maximum target value: ▪ the pointer returns to target value 1 ▪ output relay 0 turns to TRUE ▪ the elapsed value is set to 0	When the elapsed value reaches the last target value: ▪ the pointer returns to target value 1 ▪ output relay 0 turns to TRUE ▪ the elapsed value continues to increment and restarts at the minimum value of the ring counter
Decremental counting: The pointer of the data table moves from the last target value to target value 1.	When the elapsed value reaches the value -1: ▪ the pointer returns to the last target value ▪ the output relay corresponding to the last target value turns to TRUE ▪ the elapsed value is set to the maximum target value	When the elapsed value reaches the value -1: ▪ the pointer returns to target value n ▪ the output relay corresponding to the last target value turns to TRUE ▪ the elapsed value continues to decrement and restarts at the maximum value of the ring counter

☞ **Provided that neither a reset input nor a software reset is being used.**

Hardware reset operation

Channel	Hardware reset input
0	X2
1	
2	X5
3	

Interrupt operation

The interrupt program will be executed when the elapsed value matches the target value. Any interrupt that has been entered into the Tasks list is automatically enabled. A special interrupt program number is assigned to each channel number.

Channel	0	1	2	3	4	5
Interrupt program	0	1	3	4	6	7

General programming information

- Select the high-speed counter input for the desired channel in the system registers.
- When a high-speed counter instruction is executed, the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) for the channel used turns to TRUE. No other high-speed counter instruction using the same channel can be executed as long as the control flag is TRUE.
- To cancel execution of an instruction, set bit 3 of the data register storing the high-speed counter control code (sys_wHscOrPulseControlCode) to TRUE. The high-speed counter control flag then changes to FALSE. To re-enable execution of the high-speed counter instruction, reset bit 3 to FALSE.
- Rewriting the elapsed value for the channel used during the execution of the instruction may cause an unexpected operation.
- Make sure the time span between adjacent target values is greater than 1ms.
- If the instruction is executed in the main program, make sure the minimum time span between adjacent target values is greater than the scan time.
- If the instruction is executed in an interrupt program, make sure the minimum time span between adjacent target values is greater than the maximum execution time of the interrupt program.
- This instruction can be executed simultaneously on a maximum of two channels.
- When using a reset input or a software reset, make sure target value 1 is an integer and ≥ 1.
- When maximum target value control is used together with a reset input or software reset, be careful not to use them at the same time.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

PLC types: **Availability of F165_HighSpeedCounter_Cam (see page 1322)**

Data types

Variable	Data type	Function
iHscChannel*	INT	High-speed counter channel: 0–5
s_dutDataTable	ANY_DUT	Starting address of area containing the data table Sample: F165_HighSpeedCounter_Cam_8_Values_DUT
dutBitOutputs	ANY_DUT	Starting address (WR) of area containing the word address for the output relays, e.g. BOOL32_OVERLAPPING_DUT. Select the size (16 or 32 bits) according to the number set with diNumberOfTargetValuesAndOutputRelays.

Operands

For	Relay				T/C		Register			Constant
iHscChannel*	-	-	-	-	-	-	DT	-	-	-

Error flags

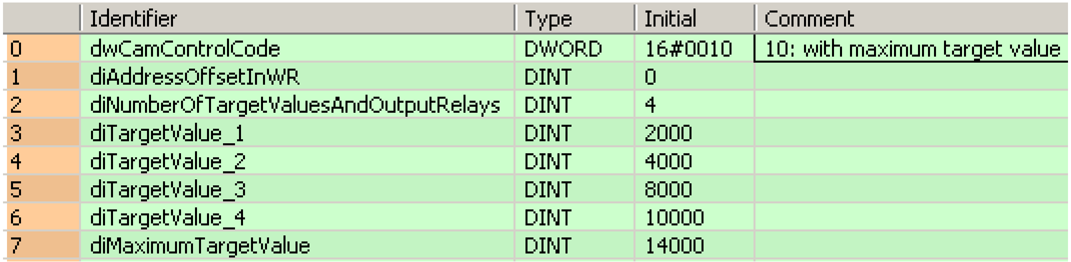
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ channel number or values of the data table are outside the permissible range ▪ high-speed counter has not been set in the system registers ▪ target value > maximum target value. ▪ target value = 0. ▪ target values are not arranged in ascending order
R9008	%MX0.900.8	for an instant	

Example

Example 1: With maximum target value control

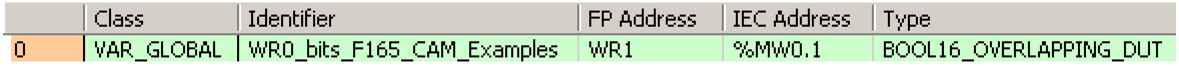
In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

DUT The DUT F165_HighSpeedCounter_Cam_8_Values_DUT is predefined in the FP Library and can be used as a sample.



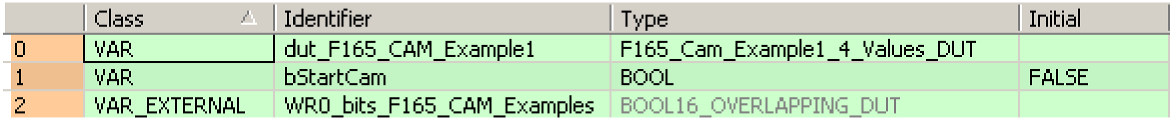
	Identifier	Type	Initial	Comment
0	dwCamControlCode	DWORD	16#0010	10: with maximum target value
1	diAddressOffsetInWR	DINT	0	
2	diNumberOfTargetValuesAndOutputRelays	DINT	4	
3	diTargetValue_1	DINT	2000	
4	diTargetValue_2	DINT	4000	
5	diTargetValue_3	DINT	8000	
6	diTargetValue_4	DINT	10000	
7	diMaximumTargetValue	DINT	14000	

GVL In the global variable list, you define variables that can be accessed by all POUs in the project.



	Class	Identifier	FP Address	IEC Address	Type
0	VAR_GLOBAL	WR0_bits_F165_CAM_Examples	WR1	%MW0.1	BOOL16_OVERLAPPING_DUT

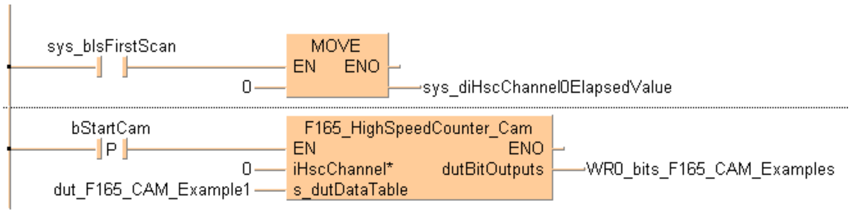
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	dut_F165_CAM_Example1	F165_Cam_Example1_4_Values_DUT	
1	VAR	bStartCam	BOOL	FALSE
2	VAR_EXTERNAL	WR0_bits_F165_CAM_Examples	BOOL16_OVERLAPPING_DUT	

Body When the variable **bStartCam** turns to TRUE, the function is carried out.

LD



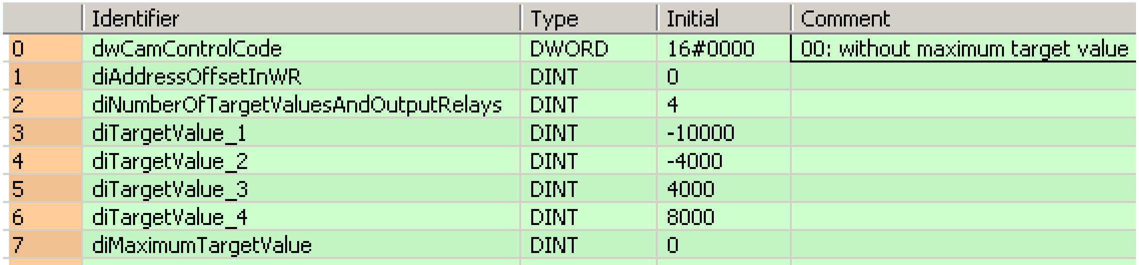
ST When programming with structured text, enter the following:

```
IF (sys_bIsFirstScan) THEN
      sys_diHscChannel0ElapsedValue:=0;
END_IF;
IF DF(bStartCam) THEN
      F165_HighSpeedCounter_Cam(iHscChannel := 0,
      s_dutDataTable := dut_F165_CAM_Example1,
      dutBitOutputs => WR0_bits_F165_CAM_Examples);
END_IF;
```

Example 2: Without maximum target value control

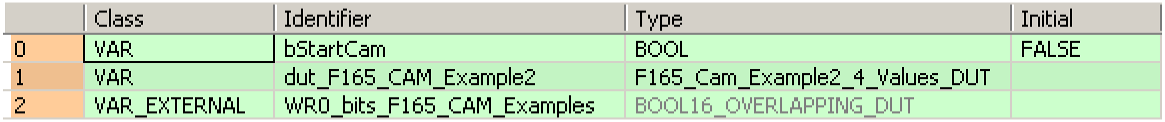
Example 2 In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

DUT The DUT F165_HighSpeedCounter_Cam_8_Values_DUT is predefined in the FP Library and can be used as a sample.



	Identifier	Type	Initial	Comment
0	dwCamControlCode	DWORD	16#0000	00: without maximum target value
1	diAddressOffsetInWR	DINT	0	
2	diNumberOfTargetValuesAndOutputRelays	DINT	4	
3	diTargetValue_1	DINT	-10000	
4	diTargetValue_2	DINT	-4000	
5	diTargetValue_3	DINT	4000	
6	diTargetValue_4	DINT	8000	
7	diMaximumTargetValue	DINT	0	

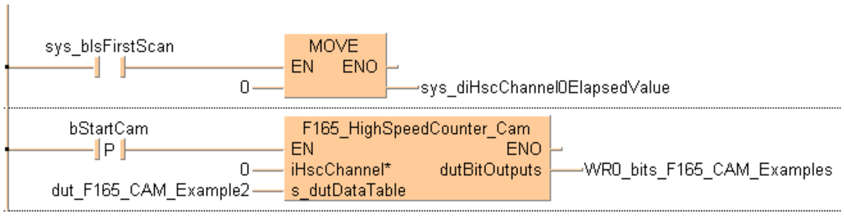
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bStartCam	BOOL	FALSE
1	VAR	dut_F165_CAM_Example2	F165_Cam_Example2_4_Values_DUT	
2	VAR_EXTERNAL	WR0_bits_F165_CAM_Examples	BOOL16_OVERLAPPING_DUT	

Body When the variable **bStartCam** is set to TRUE, the function is carried out.

LD



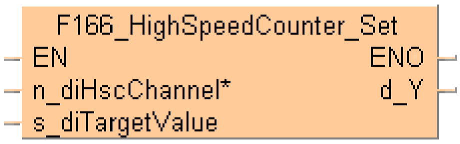
ST When programming with structured text, enter the following:

```
IF (sys_bIsFirstScan) THEN
      sys_diHscChannel0ElapsedValue:=0;
END_IF;
IF DF(bStartCam) THEN
      F165_HighSpeedCounter_Cam(iHscChannel := 0,
      s_dutDataTable := dut_F165_CAM_Example2,
      dutBitOutputs => WR0_bits_F165_CAM_Examples);
END_IF;
```

F166_HighSpeed Counter_Set

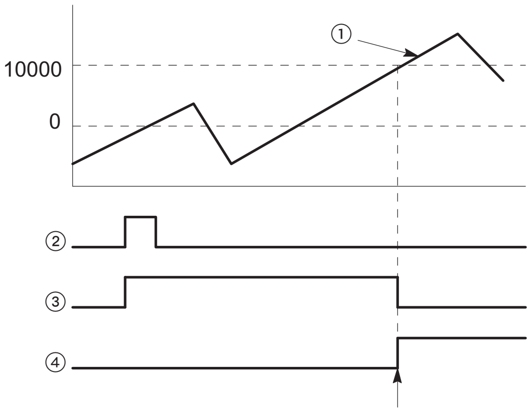
Target value match ON (high-speed counter)

Description If the elapsed value of the high-speed counter matches the target value, an interrupt process immediately turns the specified output to TRUE.



See also: Hsc_TargetValueMatch_Set (see page 1168)

Characteristics of target value match ON control



10000	Target value
①	Elapsed value of high-speed counter
②	Execution condition
③	High-speed counter control flag
④	PLC output

The PLC output turns to TRUE when the elapsed value matches the target value. In addition, the high-speed counter control flag turns to FALSE and the instruction is deactivated.

If an output is specified that has not been implemented, only the internal memory of the corresponding WY address is set or reset.

Interrupt operation

The interrupt program will be executed when the elapsed value matches the target value. Any interrupt that has been entered into the Tasks list is automatically enabled.

Channels used by interrupt programs:

PLC type	FP0, FP-e	FPΣ	FP-X (Relay types)	FP-X (Transistor types)	FP0R
Interrupt 0	Channel 0	Channel 0	Channel 0	Channel 0	Channel 0
Interrupt 1	Channel 1	Channel 1	Channel 1	Channel 1	Channel 1
Interrupt 2			Channel 2	Channel 2	
Interrupt 3	Channel 2	Channel 2	Channel 3	Channel 3	Channel 2

PLC type	FP0, FP-e	FPΣ	FP-X (Relay types)	FP-X (Transistor types)	FP0R
Interrupt 4	Channel 3	Channel 3	Channel 4	Channel 4	Channel 3
Interrupt 5			Channel 5	Channel 5	
Interrupt 6			Channel 6	Channel 6	Channel 4
Interrupt 7			Channel 7	Channel 7	Channel 5
Interrupt 8			Channel 8		
Interrupt 9			Channel 9		
Interrupt 10					
Interrupt 11			Channel A		
Interrupt 12			Channel B		

General programming information

- Select the high-speed counter input for the desired channel in the system registers.
- FP-X, FP0R: When a high-speed counter instruction is executed, the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) for the channel used turns to TRUE. No other high-speed counter instruction using the same channel can be executed as long as the control flag is TRUE.
- FP0, FP-e, FPΣ: The high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) are assigned to the same internal relay (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- To set a PLC output to FALSE that was previously set to TRUE by this instruction, use an RST or MOVE instruction.
- To cancel execution of an instruction, set bit 3 of the data register storing the high-speed counter control code (sys_wHscOrPulseControlCode) to TRUE. The high-speed counter control flag then changes to FALSE. To re-enable execution of the high-speed counter instruction, reset bit 3 to FALSE.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

PLC types **Availability of F166_HighSpeedCounter_Set (see page 1322)**

Data types

Variable	Data type	Function
n_diHscChannel	DINT	High-speed counter channel: FP-Σ: 0–3 FP-X R: 0–11 FP-X T: 0–7 FP0: 0–3 FP-e: 0–3 FP0R: 0–5
s_diTargetValue	DINT	specify a 32-bit data value for the target value within the following range: FP0, FP-e: -838808–+8388607 FPΣ, FP-X, FP0R: -2147483467–+2147483648
d_Y	BOOL	output which turns to TRUE when the elapsed value matches the target value: FP-Σ, FP0, FP-e: Y0–Y7 FP-Σ (V3.1 or higher), FP0R: Y0–Y1F FP-X: Y0–Y29F

Operands

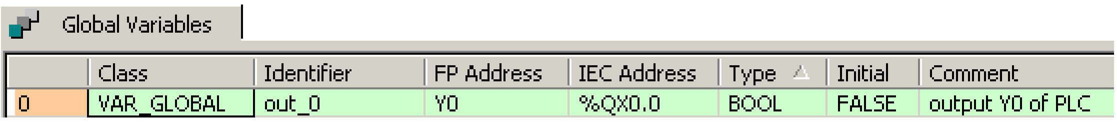
For	Relay				T/C		Register			Constant
n_diHscChannel	-	-	-	-	-	-	-	-	-	dec. or hex.
s_diTargetValue	DWX	DWY	DWR	-	DSV	DEV	DDT	-	-	-
d_Y	-	Y	-	-	-	-	-	-	-	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	ON	▪ channel number or values of the data table are outside the permissible range ▪ high-speed counter has not been set in the system registers
R9008	%MX0.900.8	ON	

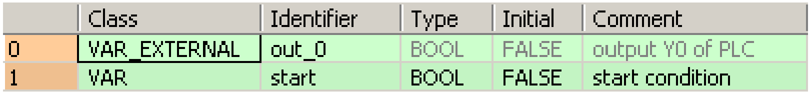
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

GVL In the global variable list, you define variables that can be accessed by all POUs in the project.


Global Variables

	Class	Identifier	FP Address	IEC Address	Type	Initial	Comment
0	VAR_GLOBAL	out_0	Y0	%QX0.0	BOOL	FALSE	output Y0 of PLC

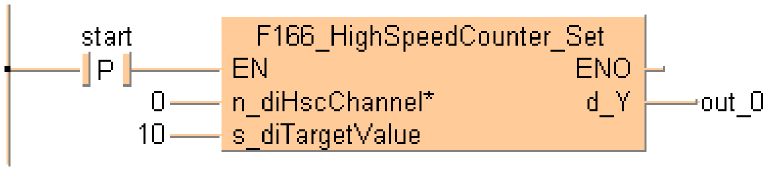
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	out_0	BOOL	FALSE	output Y0 of PLC
1	VAR	start	BOOL	FALSE	start condition

Body When the variable **start** is set to TRUE, the function is carried out.

LD



ST When programming with structured text, enter the following:

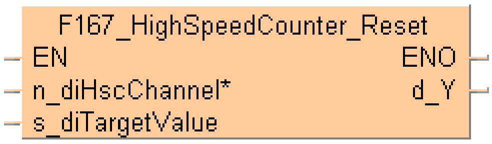
```
IF DF(start) THEN
      F166_HighSpeedCounter_Set(n_diHscChannel := 0,
      s_diTargetValue := 10,
      d_Y => out_0);

END_IF;
```

Assign a number to the input variable (e.g. Monitor → Monitor Header, click the variable, enter the value, press <Enter>), or replace the input variables with numbers.

F167_HighSpeed Counter_Reset

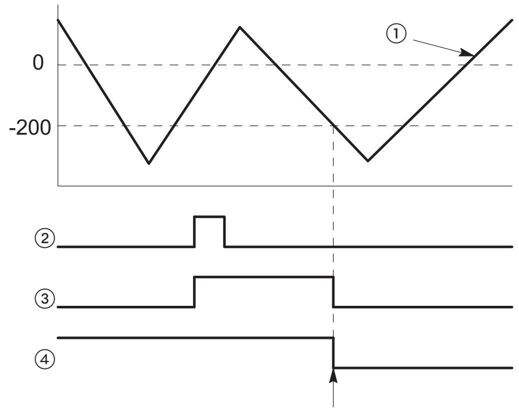
Target value match OFF (high-speed counter)

Description If the elapsed value of the high-speed counter matches the target value, an interrupt process immediately turns the specified output to FALSE.



See also: Hsc_TargetValueMatch_Reset (see page 1166)

Characteristics of target value match OFF control



-200	Target value
①	Elapsed value of high-speed counter
②	Execution condition
③	High-speed counter control flag
④	PLC output

The PLC output turns to FALSE when the elapsed value matches the target value. In addition, the high-speed counter control flag turns to FALSE and the instruction is deactivated.

If an output is specified that has not been implemented, only the internal memory of the corresponding WY address is set or reset.

Interrupt operation

The interrupt program will be executed when the elapsed value matches the target value. Any interrupt that has been entered into the Tasks list is automatically enabled. A special interrupt program number is assigned to each channel number.

Channels used by interrupt programs:

PLC type	FP0, FP-e	FPΣ	FP-X (Relay types)	FP-X (Transistor types)	FP0R
Interrupt 0	Channel 0	Channel 0	Channel 0	Channel 0	Channel 0
Interrupt 1	Channel 1	Channel 1	Channel 1	Channel 1	Channel 1
Interrupt 2			Channel 2	Channel 2	

PLC type	FP0, FP-e	FPΣ	FP-X (Relay types)	FP-X (Transistor types)	FP0R
Interrupt 3	Channel 2	Channel 2	Channel 3	Channel 3	Channel 2
Interrupt 4	Channel 3	Channel 3	Channel 4	Channel 4	Channel 3
Interrupt 5			Channel 5	Channel 5	
Interrupt 6			Channel 6	Channel 6	Channel 4
Interrupt 7			Channel 7	Channel 7	Channel 5
Interrupt 8			Channel 8		
Interrupt 9			Channel 9		
Interrupt 10					
Interrupt 11			Channel A		
Interrupt 12			Channel B		

General programming information

- Select the high-speed counter input for the desired channel in the system registers.
- FP-X, FP0R: When a high-speed counter instruction is executed, the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) for the channel used turns to TRUE. No other high-speed counter instruction using the same channel can be executed as long as the control flag is TRUE.
- FP0, FP-e, FPΣ: The high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) are assigned to the same internal relay (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- To set a PLC output to FALSE that was previously set to TRUE by this instruction, use an RST or MOVE instruction.
- To cancel execution of an instruction, set bit 3 of the data register storing the high-speed counter control code (sys_wHscOrPulseControlCode) to TRUE. The high-speed counter control flag then changes to FALSE. To re-enable execution of the high-speed counter instruction, reset bit 3 to FALSE.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

PLC types **Availability of F167_HighSpeedCounter_Reset (see page 1322)**

Data types

Variable	Data type	Function
n_diHscChannel	DINT	High-speed counter channel: FP-Σ: 0–3 FP-X R: 0–11 FP-X T: 0–7 FP0: 0–3 FP-e: 0–3 FP0R: 0–5
s_diTargetValue	DINT	specify a 32-bit data value for the target value within the following range: FP0, FP-e: -838808–+8388607 FPΣ, FP-X, FP0R: -2147483467–+2147483648
d_Y	BOOL	output which turns to FALSE when the elapsed value matches the target value: FP-Σ, FP0, FP-e: Y0–Y7 FP-Σ (V3.1 or higher), FP0R: Y0–Y1F FP-X: Y0–Y29F

Operands

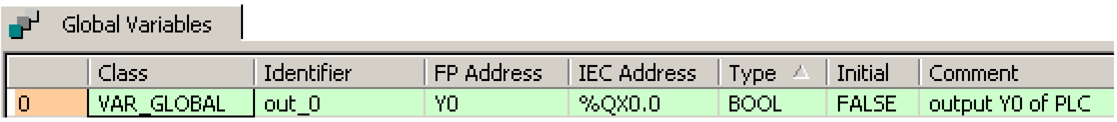
For	Relay				T/C		Register			Constant
n_diHscChannel	-	-	-	-	-	-	-	-	-	dec. or hex.
s_diTargetValue	DWX	DWY	DWR	-	DSV	DEV	DDT	-	-	-
d_Y	-	Y	-	-	-	-	-	-	-	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	ON	▪ channel number or values of the data table are outside the permissible range ▪ high-speed counter has not been set in the system registers
R9008	%MX0.900.8	ON	

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

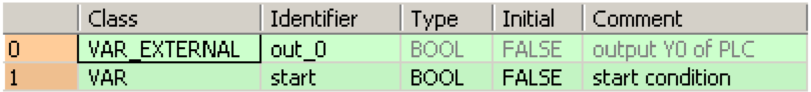
GVL In the global variable list, you define variables that can be accessed by all POUs in the project.



Global Variables

	Class	Identifier	FP Address	IEC Address	Type	Initial	Comment
0	VAR_GLOBAL	out_0	Y0	%QX0.0	BOOL	FALSE	output Y0 of PLC

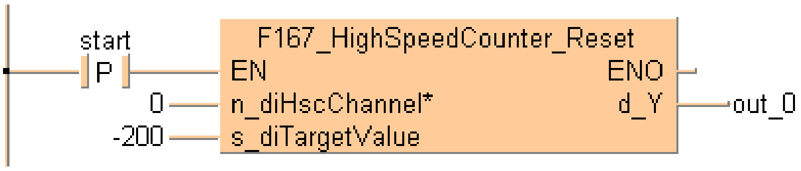
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	out_0	BOOL	FALSE	output Y0 of PLC
1	VAR	start	BOOL	FALSE	start condition

Body When the variable **start** is set to TRUE, the function is carried out.

LD



ST

When programming with structured text, enter the following:

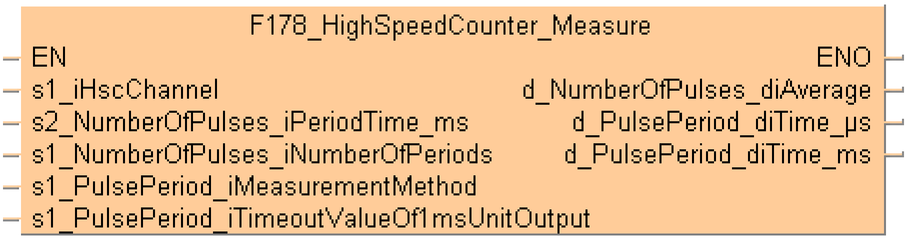
```
IF DF(start) THEN
      F167_HighSpeedCounter_Reset(n_diHscChannel := 0,
              s_diTargetValue := -200,
              d_Y => out_0);
         END_IF;
```

☞ **Assign a number to the input variable (e.g. Monitor → Monitor Header, click the variable, enter the value, press <Enter>), or replace the input variables with numbers.**

F178_HighSpeed Counter_Measure

Input pulse measurement

Description This instruction measures the number of input pulses in a specified counting period and the pulse period.



```
                    F178_HighSpeedCounter_Measure
EN                                                        ENO
s1_iHscChannel                          d_NumberOfPulses_diAverage
s2_NumberOfPulses_iPeriodTime_ms            d_PulsePeriod_diTime_μs
s1_NumberOfPulses_iNumberOfPeriods          d_PulsePeriod_diTime_ms
s1_PulsePeriod_iMeasurementMethod
s1_PulsePeriod_iTimeoutValueOf1msUnitOutput
```

Characteristics of input pulse measurement

- For input pulse measurement, the channel number, the counting period (1ms–5s) and the number of counting periods (1–5) must be specified. These parameters are used to calculate the average number of input pulses per counting period.
- The unit of pulse period measurement ([μs], [ms] or both) can be specified.
- If the measurement is in μs, the pulse period is measured and output immediately upon execution of this instruction. A maximum of approx. 174.4ms can be measured.
- If the measurement is in ms, the value of the pulse period is updated after every measurement. A maximum of approx. 49.7 days can be measured. A time-out value can be specified after which the measured pulse period is set to -1 if measurement has not been completed.
- During the first counting periods after starting the instruction, the measured pulse period is set to -1 until the specified number of counting periods has been reached.
- If the pulse period is longer than the measurable range or if measurement has not been completed, the measured pulse period is set to -1.

■ General programming information

- Select the high-speed counter input for the desired channel in the system registers.
- Keep the execution condition TRUE for pulse measurement using this instruction.
- To stop the measurement, turn the execution condition to FALSE.
- When a high-speed counter instruction is executed, the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) for the channel used turns to TRUE. No other high-speed counter instruction using the same channel can be executed as long as the control flag is TRUE.
- The instruction can be executed simultaneously on a maximum of two channels.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

PLC types **Availability of F178_HighSpeedCounter_Measure (see page 1322)**

Data types

Variable	Data type	Function
s1_iHscChannel	INT	High-speed counter channel: 0–5
s2_NumberOfPulses_iPeriodTime_ms	INT	Counting period [ms]: 1–5000 (1ms–5s).
s1_NumberOfPulses_iNumberOfPeriods	INT	Number of counting periods: 1–5
s1_PulsePeriod_iMeasurementMethod	INT	Unit of pulse period measurement 0: Pulse period is not measured 1: Pulse period is measured in µs 2: Pulse period is measured in ms 3: Pulse period is measured in µs and ms
s1_PulsePeriod_iTimeoutValueOf1msUnitOutput	INT	Time-out value of pulse period measurement [ms]: 0: no time-out 1: 100ms 6: 1s 2: 200ms 7: 2s 3: 300ms 8: 10s 4: 500ms 9: 60s
d_NumberOfPulses_diAverage	DINT	Average number of pulses per counting period (no. of pulses in counting period/number of counting periods)
d_PulsePeriod_diTime_µs	DINT	Pulse period [µs]

Operands

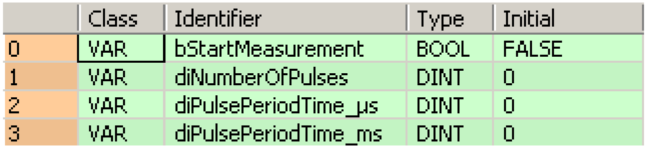
For	Relay				T/C		Register			Constant
s1_iHscChannel	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1/s2 inputs	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d outputs	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ channel number or values of the data table are outside the permissible range ▪ high-speed counter has not been set in the system registers ▪ the high-speed counter channel is already used by another high-speed counter or pulse output instruction ▪ the number of channels used is 3 or more
R9008	%MX0.900.8	for an instant	

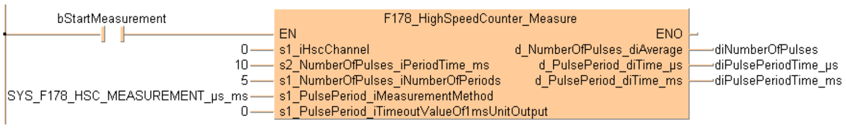
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bStartMeasurement	BOOL	FALSE
1	VAR	diNumberOfPulses	DINT	0
2	VAR	diPulsePeriodTime_µs	DINT	0
3	VAR	diPulsePeriodTime_ms	DINT	0

LD


bStartMeasurement
F178_HighSpeedCounter_Measure
EN
ENO
0
s1_iHscChannel
10
s2_NumberOfPulses_iPeriodTime_ms
5
s1_NumberOfPulses_iNumberOfPeriods
SYS_F178_HSC_MEASUREMENT_µs_ms
s1_PulsePeriod_iMeasurementMethod
0
s1_PulsePeriod_iTimeoutValueOf1msUnitOutput
d_NumberOfPulses_diAverage
diNumberOfPulses
d_PulsePeriod_diTime_µs
diPulsePeriodTime_µs
d_PulsePeriod_diTime_ms
diPulsePeriodTime_ms

ST When programming with structured text, enter the following:

```
IF (bStartMeasurement) THEN
  F178_HighSpeedCounter_Measure(s1_iHscChannel := 0,
            s2_NumberOfPulses_iPeriodTime_ms := 10,
            s1_NumberOfPulses_iNumberOfPeriods := 5,
            s1_PulsePeriod_iMeasurementMethod :=
SYS_F178_HSC_MEASUREMENT_•s_ms,
            s1_PulsePeriod_iTimeoutValueOf1msUnitOutput := 0,
            d_NumberOfPulses_diAverage => diNumberOfPulses,
            d_PulsePeriod_diTime_•s => diPulsePeriodTime_•s,
            d_PulsePeriod_diTime_ms => diPulsePeriodTime_ms);
END_IF;
```

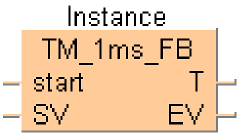
Chapter 29

Timer instructions

TM_1ms_FB

Timer for 1ms intervals (0 to 32.767s)

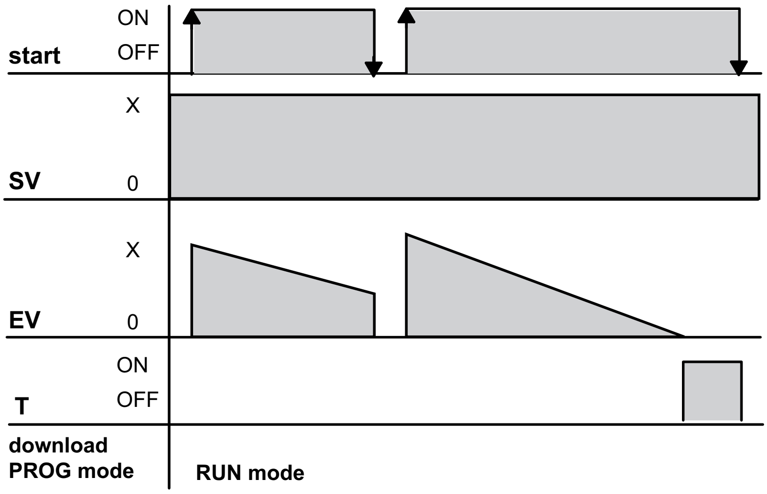
Description This timer for 0.001s units works as an ON-delay timer. If the **start** contact of the function block is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON.



For the TM_1ms_FB function block declare the following:

start	**start contact** each time a rising edge is detected, the set value SV is copied to the elapsed value EV and the timer is started
SV	**set value** the defined ON-delay time (0 to 32.767s)
T	**timer contact** is set when the time defined at SV has elapsed, this means when EV becomes 0
EV	**elapsed value** count value from which 1 is subtracted every 0.001s while the timer is running

Time chart



☞

- **The number of available timers is limited and depends on the settings in the system registers 5 and 6.**
- **The system timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the system timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**

PLC types **Availability of TM_1ms_FB (see page 1332)**

Data types

Variable	Data type	Function
start	BOOL	start contact
SV	INT, WORD	set value
T	BOOL	timer contact
EV	INT, WORD	elapsed value

Operands

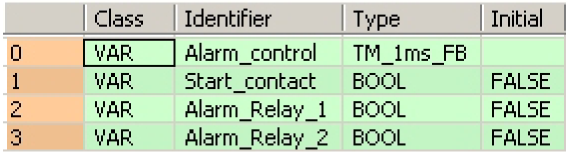
For	Relay				T/C		Register			Constant
start	X	Y	R	L	T	C	-	-	-	-
T	-	Y	R	L	-	-	-	-	-	-
SV, EV	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example

In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

All input and output variables which are used for programming the function block TM_1ms_FB are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **Alarm_Control**, and a separate data area is reserved.



	Class	Identifier	Type	Initial
0	VAR	Alarm_control	TM_1ms_FB	
1	VAR	Start_contact	BOOL	FALSE
2	VAR	Alarm_Relay_1	BOOL	FALSE
3	VAR	Alarm_Relay_2	BOOL	FALSE

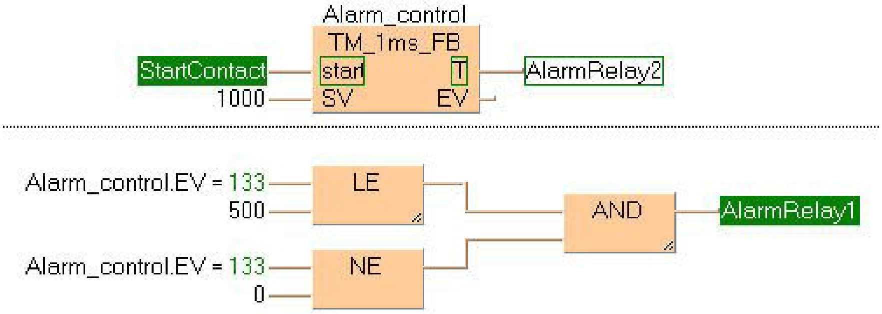
This example uses variables. You may also use constants for the input variables.

Body

As soon the variable **Start_contact** becomes TRUE, the timer **Alarm_control** will be started. The variable **EV** of the timer is set to the value of **SV**. As long as **Start_contact** is TRUE, the value 1 is subtracted from **EV** every 1ms. When **EV** reaches the value 0 (after 1 second as SV = 1000 with the timer type TM_1ms_FB), the variable **Alarm_Relay_2** becomes TRUE.

As soon as the value of the variable **EV** of the timer is smaller than or equal to 500 (after 0.5s) and **EV** is unequal 0, **Alarm_Relay_1** is set to TRUE.

LD



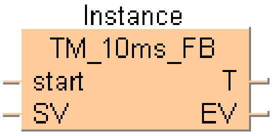
ST When programming with structured text, enter the following:

```
Alarm_Control( start:= Start_Contact ,
       SV:= 1000,
       T=> Alarm_Relay_2 ,
       EV=> Alarm_Control.EV );
(*The ON-delay time is 1000ms*)
Alarm_Relay_1:= Alarm_Control.EV <= 500 & Alarm_Control.EV <> 0;
(*Alarm_Relay_1 is set to TRUE after 500ms*)
(*Alarm_Relay_1 is set to TRUE after 500ms*)
```

TM_10ms_FB

Timer for 10ms intervals (0 to 327.67s)

Description This timer for 0.01s units works as an ON-delay timer. If the **start** contact of the function block is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON.



For the TM_10ms_FB function block declare the following:

start **start contact**

each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started

SV **set value**

the defined ON-delay time (0 to 327.67s)

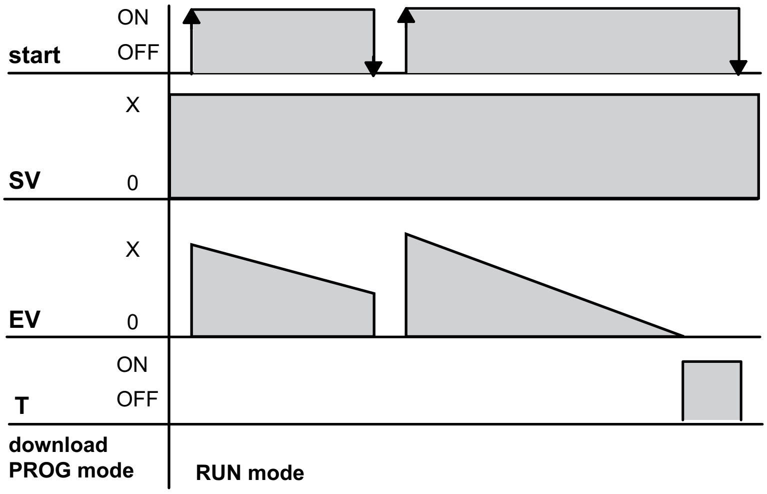
T **timer contact**

is set when the time defined at **SV** has elapsed, this means when **EV** becomes 0

EV **elapsed value**

count value from which 1 is subtracted every 0.01s while the timer is running

Time chart



☞

- **The number of available timers is limited and depends on the settings in the system registers 5 and 6.**
- **The system timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the system timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**

PLC types **Availability of TM_10ms_FB (see page 1332)**

Data types

Variable	Data type	Function
start	BOOL	start contact
SV	INT, WORD	set value
T	BOOL	timer contact
EV	INT, WORD	elapsed value

Operands

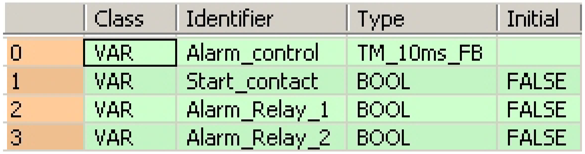
For	Relay				T/C		Register			Constant
start	X	Y	R	L	T	C	-	-	-	-
T	-	Y	R	L	-	-	-	-	-	-
SV, EV	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example

In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

All input and output variables which are used for programming the function block TM_10ms_FB are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under Alarm_Control, and a separate data area is reserved.



	Class	Identifier	Type	Initial
0	VAR	Alarm_control	TM_10ms_FB	
1	VAR	Start_contact	BOOL	FALSE
2	VAR	Alarm_Relay_1	BOOL	FALSE
3	VAR	Alarm_Relay_2	BOOL	FALSE

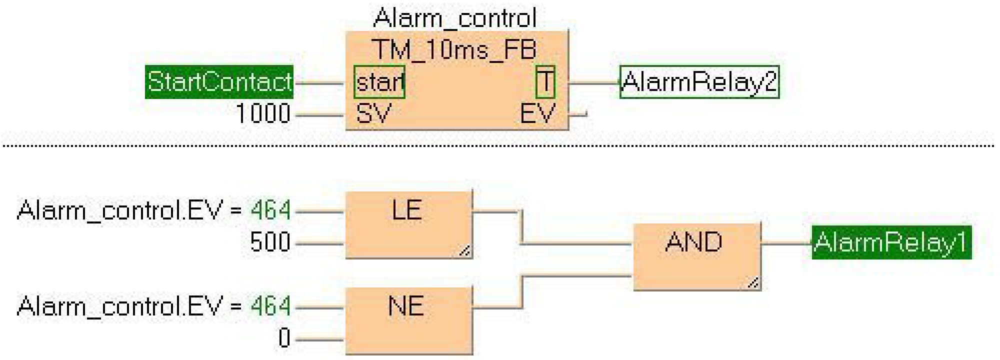
This example uses variables. You may also use constants for the input variables.

Body

As soon the variable Start_contact becomes TRUE, the timer Alarm_control will be started. The variable EV of the timer is set to the value of SV. As long as Start_contact is TRUE, the value 1 is subtracted from EV every 10ms. When EV reaches the value 0 (after 10 second as SV = 1000 with the timer type TM_10ms_FB), the variable Alarm_Relay_2 becomes TRUE.

As soon as the value of the variable EV of the timer is smaller than or equal to 500 (after 5s) and EV is unequal 0, Alarm_Relay_1 is set to TRUE.

LD



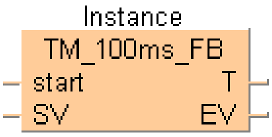
ST When programming with structured text, enter the following:

```
Alarm_Control( start:= Start_Contact ,
       SV:= 1000,
       T=> Alarm_Relay_2 ,
       EV=> Alarm_Control.EV );
(*The ON-delay time is 10s*)

Alarm_Relay_1:= Alarm_Control.EV <= 500 & Alarm_Control.EV <> 0;
(*Alarm_Relay_1 is set to TRUE after 5s*)
(*Alarm_Relay_1 is set to TRUE after 5s*)
```

TM_100ms_FB

Timer for 100ms intervals (0 to 3276.7s)

Description This timer for 0.1s units works as an ON-delay timer. If the **start** contact of the function block is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON.



For the TM_100ms_FB function block declare the following:

start **start contact**

each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started

SV **set value**

the defined ON-delay time (0 to 3276.7s)

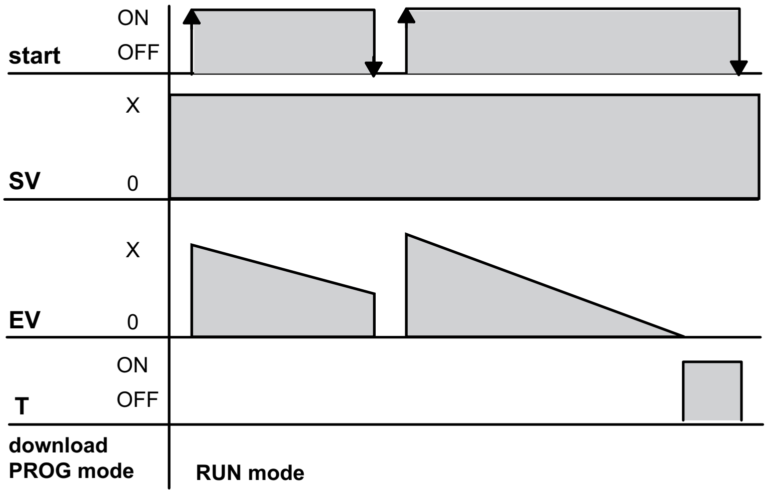
T **timer contact**

is set when the time defined at **SV** has elapsed, this means when **EV** becomes 0

EV **elapsed value**

count value from which 1 is subtracted every 0.1s while the timer is running

Time chart



- **The number of available timers is limited and depends on the settings in the system registers 5 and 6.**
- **The system timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the system timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**

PLC types **Availability of TM_100ms_FB (see page 1332)**

Data types

Variable	Data type	Function
start	BOOL	start contact
SV	INT, WORD	set value
T	BOOL	timer contact
EV	INT, WORD	elapsed value

Operands

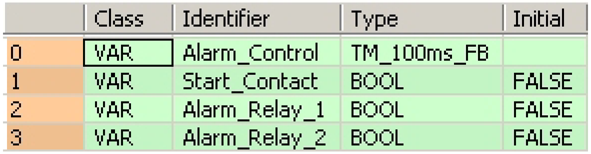
For	Relay				T/C		Register			Constant
start	X	Y	R	L	T	C	-	-	-	-
T	-	Y	R	L	-	-	-	-	-	-
SV, EV	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example

In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

All input and output variables which are used for programming the function block TM_100ms_FB are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **Alarm_Control**, and a separate data area is reserved.



	Class	Identifier	Type	Initial
0	VAR	Alarm_Control	TM_100ms_FB	
1	VAR	Start_Contact	BOOL	FALSE
2	VAR	Alarm_Relay_1	BOOL	FALSE
3	VAR	Alarm_Relay_2	BOOL	FALSE

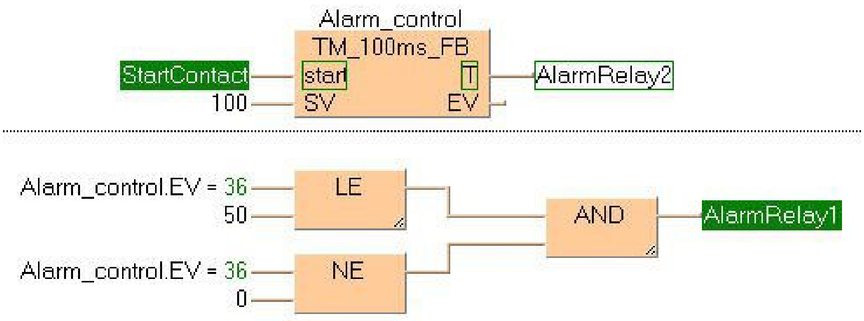
This example uses variables. You may also use constants for the input variables.

Body

As soon the variable **Start_contact** becomes TRUE, the timer **Alarm_control** will be started. The variable **EV** of the timer is set to the value of **SV**. As long as **Start_contact** is TRUE, the value 1 is subtracted from **EV** every 100ms. When **EV** reaches the value 0 (after 10 seconds as **SV** = 100 with the timer type TM_100ms_FB), the variable **Alarm_Relay_2** becomes TRUE.

As soon as the value of the variable **EV** of the timer is smaller than or equal to 50 (after 5s) and **EV** is unequal 0, **Alarm_Relay_1** is set to TRUE.

LD



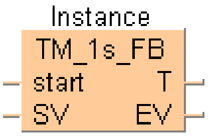
ST When programming with structured text, enter the following:

```
Alarm_Control( start:= Start_Contact ,
       SV:= 100,
       T=> Alarm_Relay_2 ,
       EV=> Alarm_Control.EV );
(*The ON-delay time is 10s*)
Alarm_Relay_1:= Alarm_Control.EV <= 50 & Alarm_Control.EV <> 0;
(*Alarm_Relay_1 is set to TRUE after 5s*)
(*Alarm_Relay_1 is set to TRUE after 5s*)
```

TM_1s_FB

Timer for 1s intervals (0 to 32767s)

Description This timer for 1s units works as an ON-delay timer. If the **start** contact of the function block is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON.



For the TM_1s_FB function block declare the following:

start **start contact**

each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started

SV **set value**

the defined ON-delay time (0 to 32767s)

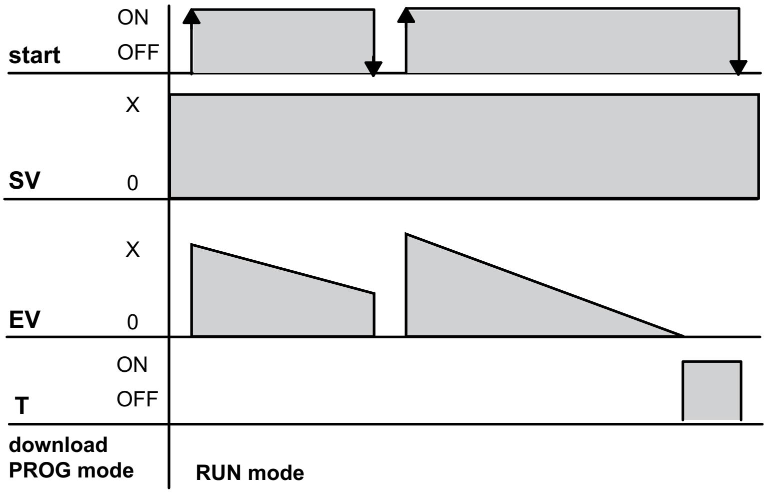
T **timer contact**

is set when the time defined at SV has elapsed, this means when **EV** becomes 0

EV **elapsed value**

count value from which 1 is subtracted every 1s while the timer is running

Time chart



☞

- **The number of available timers is limited and depends on the settings in the system registers 5 and 6.**
- **The system timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the system timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**

PLC types **Availability of TM_1s_FB (see page 1332)**

Data types

Variable	Data type	Function
start	BOOL	start contact
SV	INT, WORD	set value
T	BOOL	timer contact
EV	INT, WORD	elapsed value

Operands

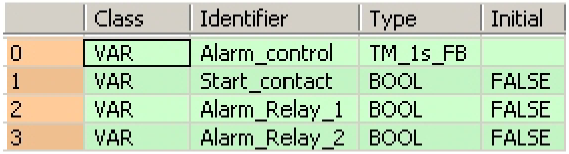
For	Relay				T/C		Register			Constant
start	X	Y	R	L	T	C	-	-	-	-
T	-	Y	R	L	-	-	-	-	-	-
SV, EV	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Example

In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header

All input and output variables which are used for programming the function block TM_1s_FB are declared in the POU header. This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under **Alarm_Control**, and a separate data area is reserved.



	Class	Identifier	Type	Initial
0	VAR	Alarm_control	TM_1s_FB	
1	VAR	Start_contact	BOOL	FALSE
2	VAR	Alarm_Relay_1	BOOL	FALSE
3	VAR	Alarm_Relay_2	BOOL	FALSE

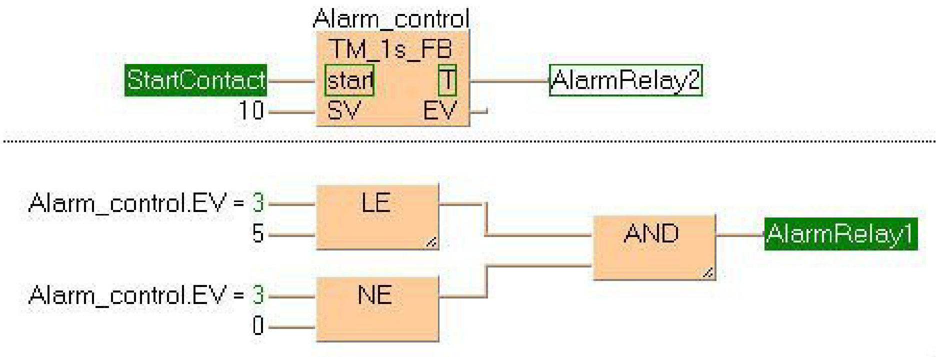
This example uses variables. You may also use constants for the input variables.

Body

As soon the variable **Start_contact** becomes TRUE, the timer **Alarm_control** will be started. The variable **EV** of the timer is set to the value of **SV**. As long as **Start_contact** is TRUE, the value 1 is subtracted from **EV** every 1s. When **EV** reaches the value 0 (after 10 seconds as **SV** = 10 with the timer type TM_1s_FB), the variable **Alarm_Relay_2** becomes TRUE.

As soon as the value of the variable **EV** of the timer is smaller than or equal to 5 (after 5s) and **EV** is unequal 0, **Alarm_Relay_1** is set to TRUE.

LD



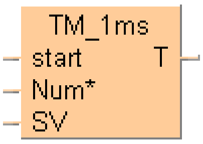
ST When programming with structured text, enter the following:

```
Alarm_Control( start:= Start_Contact ,
       SV:= 10,
       T=> Alarm_Relay_2 ,
       EV=> Alarm_Control.EV );
(*The ON-delay time is 10s*)
Alarm_Relay_1:= Alarm_Control.EV <= 5 & Alarm_Control.EV <> 0;
(*Alarm_Relay_1 is set to TRUE after 5s*)
(*Alarm_Relay_1 is set to TRUE after 5s*)
```

TM_1ms

Timer for 1ms intervals (0 to 32.767s)

Description The TM_1ms instruction sets the ON-delay timer for 0.001s units (0 to 32.767s).



Instead of using this FP instruction, we recommend using the related IEC instruction tmTM_1ms_FB (see page 913).

Please refer also to Advantages of the IEC instructions in the online help.

The areas used for the instruction are:

- Preset (Set) value area: SV
- Count (Elapsed) value area: **EV**

When the mode is set to RUN mode, the Preset (Set) value is transferred to the SV. If the trigger of the timer instruction **start** is in the ON-state, the Preset (Set) value is transferred to the **EV** from the SV.

During the timing operation, the time is subtracted from the **EV**.

The scan time is also subtracted from the **EV** in the next scan.

The timer contact **T** turns ON, when the **EV** becomes 0.

Calculation of the timing operation:

timing operation = time set value - 0 to 1/2 of units (0.5ms) + scan time

Example:

150ms time set value and 8ms PLC scan time

Upper limit = 150 - 0 + 8 = 158ms
Lower limit = 150 -0.5 +8 = 157.5ms

The result is a timing operation from 157.5ms to 158ms.

PLC types **Availability of** TM_1ms **(see page 1332)**

Data types

Variable	Data type	Function
start	BOOL	starts timer
Num*	ANY16	timer contact Must be a constant
SV		timer address in system registers 5 and 6
T	BOOL	set value

Operands

For	Relay				T/C		Register			Constant
start	X	Y	R	L	T	C	-	-	-	-
T	-	Y	R	L	-	-	-	-	-	-
Num*	-	-	-	-	-	-	-	-	-	dec. or hex.
SV	-	-	-	-	SV	-	-	-	-	dec. or hex.

☞

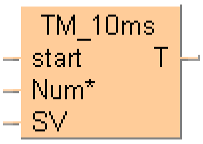
- **It is not possible to use this function in a function block POU.**
- **For correct results, timer functions and timer function blocks must be executed exactly one time in each scan. Thus it is not allowed to use timer function or timer function blocks in interrupt programs or in loops.**
- **Every used timer must have a separate constant Num*. Available Num* addresses depend on system registers 5 and 6.**
  Timer of type TM_1s, TM_100ms, TM_10ms, TM_1ms use the same Num* address range.
- **The system timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the system timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**
- **This function does not require a variable at the output "T".**

Example Please refer to the example of TM_1ms_FB (see page 913).

TM_10ms

Timer for 10ms intervals (0 to 327.67s)

Description The TM_10ms instruction sets the ON-delay timer for 0.01 s units (0 to 327.67s).



Instead of using this FP instruction, we recommend using the related IEC instruction TM_10ms_FB (see page 916).

Please refer also to Advantages of the IEC instructions in the online help.

The areas used for the instruction are:

- Preset (Set) value area: **SV**
- Count (Elapsed) value area: **EV**

When the mode is set to RUN mode, the Preset (Set) value is transferred to the **SV**. If the trigger of the timer instruction **start** is in the ON-state, the Preset (Set) value is transferred to the **EV** from the **SV**.

During the timing operation, the time is subtracted from the **EV**.

The scan time is also subtracted from the **EV** in the next scan.

The timer contact **T** turns ON, when the **EV** becomes 0.

Calculation of the timing operation:

timing operation = time set value - 0 to 1/4 of units (2.5ms) + scan time

Example:

150ms time set value and 8ms PLC scan time

Upper limit = 150 - 0 + 8 = 158ms
Lower limit = 150 -2.5 +8 = 155.5ms

The result is a timing operation from 155.5ms to 158ms.

PLC types **Availability of** TM_10ms **(see page 1332)**

Data types

Variable	Data type	Function
start	BOOL	starts timer
Num*	ANY16	timer address in system registers 5 and 6 Must be a constant
SV		set value
T	BOOL	timer contact

Operands

For	Relay				T/C		Register			Constant
start	X	Y	R	L	T	C	-	-	-	-
T	-	Y	R	L	-	-	-	-	-	-
Num*	-	-	-	-	-	-	-	-	-	dec. or hex.
SV	-	-	-	-	SV	-	-	-	-	dec. or hex.

☞

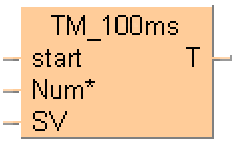
- **It is not possible to use this function in a function block POU.**
- **For correct results, timer functions and timer function blocks must be executed exactly one time in each scan. Thus it is not allowed to use timer function or timer function blocks in interrupt programs or in loops.**
- **Every used timer must have a separate constant Num*. Available Num* addresses depend on system registers 5 and 6.**
  Timer of type TM_1s, TM_100ms, TM_10ms, TM_1ms use the same Num* address range.
- **The system timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the system timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**
- **This function does not require a variable at the output "T".**

Example Please refer to the example of TM_10ms_FB (see page 916).

TM_100ms

Timer for 100ms intervals (0 to 3276.7s)

Description The TM_100ms instruction sets the ON-delay timer for 0.1s units (0 to 3276.7s).



Instead of using this FP instruction, we recommend using the related IEC instruction TM_100ms_FB (see page 919).

Please refer also to Advantages of the IEC instructions in the online help.

The **TM** instruction is a down type preset timer.

The area used for the instruction are:

- Preset (Set) value area: **SV**
- Count (Elapsed) value area: **EV**

When the mode is set to RUN mode, the Preset (Set) value is transferred to the **SV**. If the trigger of the timer instruction **start** is in the ON-state, the Preset (Set) value is transferred to the **EV** from the **SV**.

During the timing operation, the time is subtracted from the **EV**.

The scan time is also subtracted from the **EV** in the next scan.

The timer contact **T** turns ON, when the **EV** becomes 0.

Calculation of the timing operation:

timing operation = time set value - 0 to 1/4 of units (25ms) + scan time

Example:

1500ms time set value and 8ms PLC scan time

Upper limit = 1500 - 0 + 8 = 1508ms
Lower limit = 1500 -25 +8 = 1483ms

The result is a timing operation from 1483ms to 1508ms.

PLC types **Availability of** TM_100ms **(see page 1332)**

Data types

Variable	Data type	Function
start	BOOL	starts timer
Num*	ANY16	timer address in system registers 5 and 6 Must be a constant
SV		set value
T	BOOL	timer contact

Operands

For	Relay				T/C		Register			Constant
start	X	Y	R	L	T	C	-	-	-	-
T	-	Y	R	L	-	-	-	-	-	-
Num*	-	-	-	-	-	-	-	-	-	dec. or hex.
SV	-	-	-	-	SV	-	-	-	-	dec. or hex.

☞

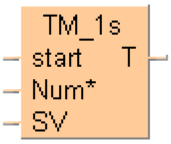
- **It is not possible to use this function in a function block POU.**
- **For correct results, timer functions and timer function blocks must be executed exactly one time in each scan. Thus it is not allowed to use timer function or timer function blocks in interrupt programs or in loops.**
- **Every used timer must have a separate constant Num*. Available Num* addresses depend on system registers 5 and 6.**
  Timer of type TM_1s, TM_100ms, TM_10ms, TM_1ms use the same Num* address range.
- **The system timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the system timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**
- **This function does not require a variable at the output "T".**

Example Please refer to the example of TM_100ms_FB (see page 919).

TM_1s

Timer for 1s intervals (0 to 32767s)

Description The TM_1s instruction sets the ON-delay timer for 1s units (0 to 32767s).



Instead of using this FP instruction, we recommend using the related IEC instruction **TM_1s_FB** (see page 922).

Please refer also to Advantages of the IEC instructions in the online help.

The area used for the instruction are:

- Preset (Set) value area: SV
- Count (Elapsed) value area: **EV**

When the mode is set to RUN mode, the Preset (Set) value is transferred to the **SV**. If the trigger of the timer instruction **start** is in the ON-state, the Preset (Set) value is transferred to the **EV** from the **SV**.

During the timing operation, the time is subtracted from the **EV**.

The scan time is also subtracted from the **EV** in the next scan.

The timer contact **T** turns ON, when the **EV** becomes 0.

Calculation of the timing operation:

timing operation = time set value - 0 to 1/4 of units (250ms) + scan time

Example:

150s time set value and 8ms PLC scan time

Upper limit = 150000 - 0 + 8 = 150008ms
Lower limit = 150000 -250 +8 = 149758ms

The result is a timing operation from 149758ms to 158ms.

PLC types **Availability of** TM_1s **(see page** 1332**)**

Data types

Variable	Data type	Function
start	BOOL	starts timer
Num*	ANY16	timer address in system registers 5 and 6 Must be a constant
SV		set value
T	BOOL	timer contact

Operands

For	Relay				T/C		Register			Constant
start	X	Y	R	L	T	C	-	-	-	-
T	-	Y	R	L	-	-	-	-	-	-
Num*	-	-	-	-	-	-	-	-	-	dec. or hex.
SV	-	-	-	-	SV	-	-	-	-	dec. or hex.

☞

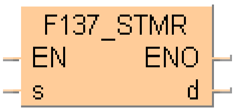
- **It is not possible to use this function in a function block POU.**
- **For correct results, timer functions and timer function blocks must be executed exactly one time in each scan. Thus it is not allowed to use timer function or timer function blocks in interrupt programs or in loops.**
- **Every used timer must have a separate constant Num*. Available Num* addresses depend on system registers 5 and 6.**
  Timer of type TM_1s, TM_100ms, TM_10ms, TM_1ms use the same Num* address range.
- **The system timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the system timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**
- **This function does not require a variable at the output "T".**

Example Please refer to the example of TM_1s_FB (see page 922).

F137_STMR

Timer 16-bit

Description The auxiliary timer instruction **F137_STMR** is a down type timer. The formula of the timer-set time is 0.01 sec. * set value **s** (time can be set from 0.01 to 327.67 sec.). If you use the special internal relay R900D as the timer contact, be sure to program it at the address immediately after the instruction.



Timer operation:

- If the trigger **EN** of the auxiliary timer instruction (STMR) is in the ON-state, the constant or value specified by **s** is transferred to the area specified by **d**.
- During the timing operation, the time is subtracted from the value in the area specified by **d**.
- The output ENO turns ON when the value in the area specified by **d** becomes 0.

PLC types **Availability of** F137_STMR **(see page** 1321**)**

Data types

Variable	Data type	Function
s	ANY16	16-bit area or equivalent constant for timer set value
d		16-bit area for timer elapsed value

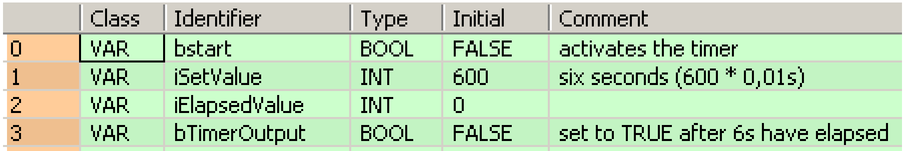
The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	-	WY	WR	WL	SV	EV	DT	LD	FL	-

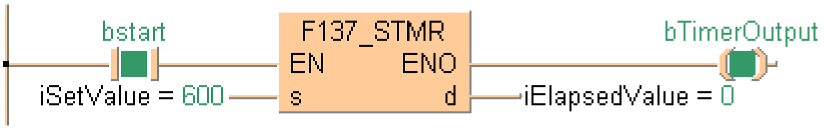
Example In this example the function is programmed in ladder diagram (LD).

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	bstart	BOOL	FALSE	activates the timer
1	VAR	iSetValue	INT	600	six seconds (600 * 0,01s)
2	VAR	iElapsedValue	INT	0	
3	VAR	bTimerOutput	BOOL	FALSE	set to TRUE after 6s have elapsed

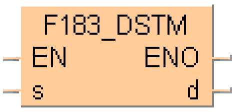
LD



F183_DSTM

Timer 32-bit

Description The F183 instruction activates an upward counting 32-bit timer which works on-delayed. The smallest counting unit is 0.01s. During execution of F183 (start = TRUE), elapsing time is added to the elapsed value **d**. The timer output will be enabled when the elapsed value **d** equals the set value **s**. If the start condition start is set to FALSE, execution will be interrupted and the elapsed value **d** will be reset to zero. The set value **s** can be changed during execution of F183.



The delay time of the timer can be calculated using the following formula: (Set Value **s**) * (0.01s) = on-delay

PLC types **Availability of** F183_DSTM **(see page 1322)**

Data types

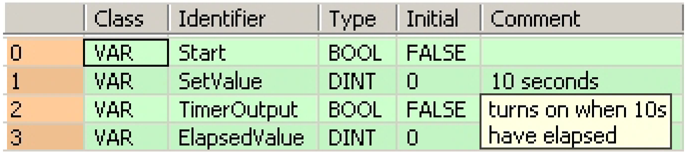
Variable	Data type	Function
s	ANY32	set value, range 0 to 2147483647
d		elapsed value, range 0 to 2147483647

Operands

For	Relay				T/C		Register			Constant
s	DWX	DWY	DWR	-	DSV	DEV	DDT	-	-	dec. or hex.
d	-	DWY	DWR	-	DSV	DEV	DDT	-	-	-

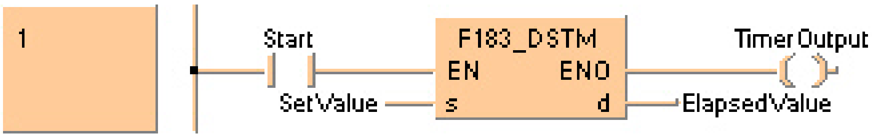
Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

POU header



	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	SetValue	DINT	0	10 seconds
2	VAR	TimerOutput	BOOL	FALSE	turns on when 10s have elapsed
3	VAR	ElapsedValue	DINT	0	

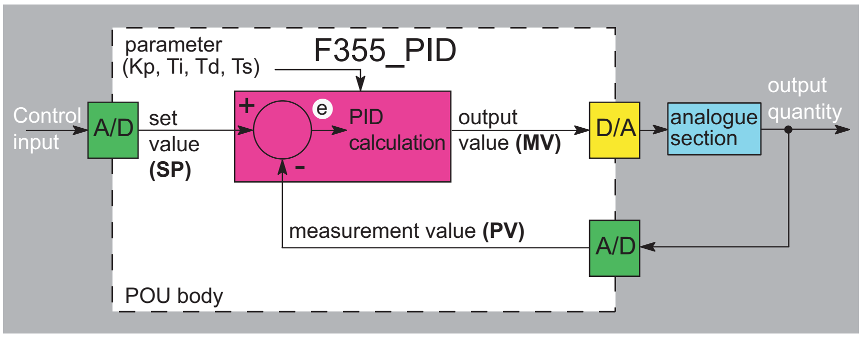
LD



Chapter 30

Process control instructions

30.1 Explanation of the operation of the PID instuctions



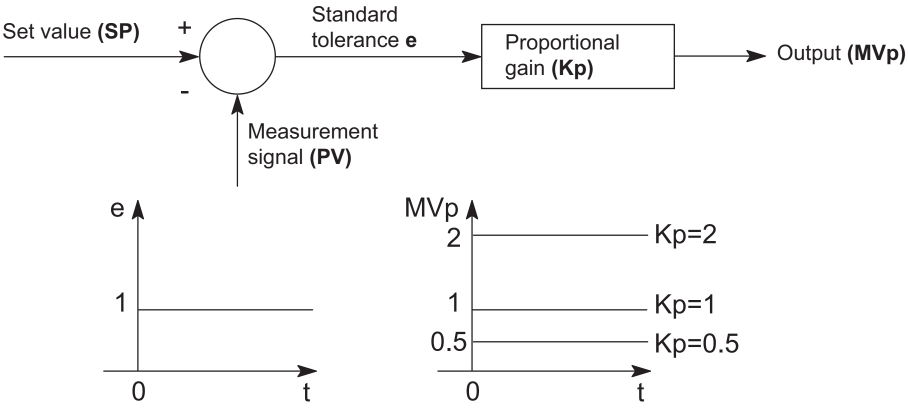
The above POU body represents the standard control loop. The control input is determined by the user (e.g. desired room temperature of 22°C). After the A/D conversion the set point value (SP) is entered as the input value for the PID processing instruction. The measured process value (PV) (e.g. current room temperature) is normally transmitted via a sensor and entered as the input value for the PID processor. F355_PID calculates the standard tolerance e from the set point value and the process value (e = set value - measured value). With the parameters given (proportional gain Kp, integral time Ti, ...) a new output value (MV) is calculated in increments set by the sampling time Ts. This result is then applied to the actuator (e.g. a fan that regulates room temperature) after the D/A conversion. The analog section represents the system's actuator, e.g. heater and temperature regulation of a room.

A PID operation consists of three components:

1. Proportional part (P part)

A proportional part generates an output that is proportional to the input. The proportional gain Kp determines by how much the input value is increased or decreased.
A proportional part can be a simple electric resistor or a linear amplifier, for example.

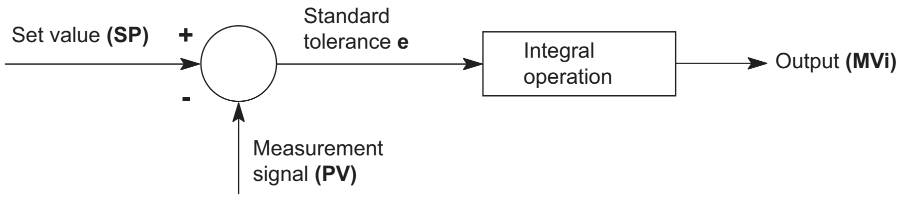
The P part displays a relatively large maximum overshot, a long settling time and a constant standard tolerance.



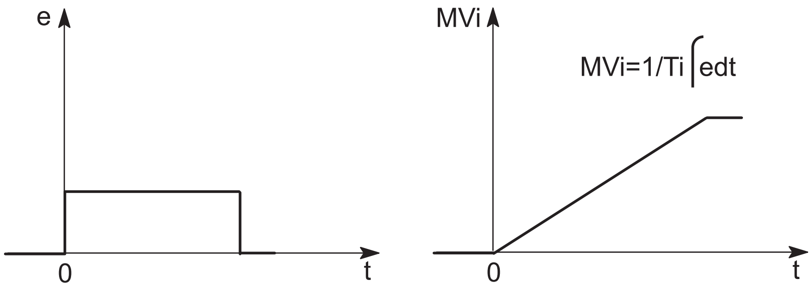
2. Integral part (I part)

An integral part produces an output quantity that corresponds to the time integral and input quantity (area of the input quantity). The integral time thus evaluates the output quantity MVi.

The integral part can be a quantity scale of a tank that is filled by a volume flow, for example. Because of the slow reaction time of the integral part, it has a larger maximum overshot than the P component, but no constant standard tolerance.



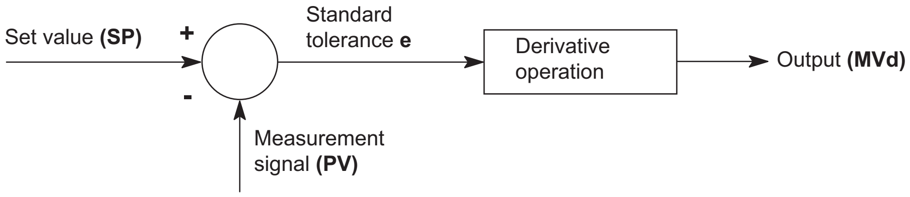
Example: Input quantity **e** and the output quantity **MVi** produced.



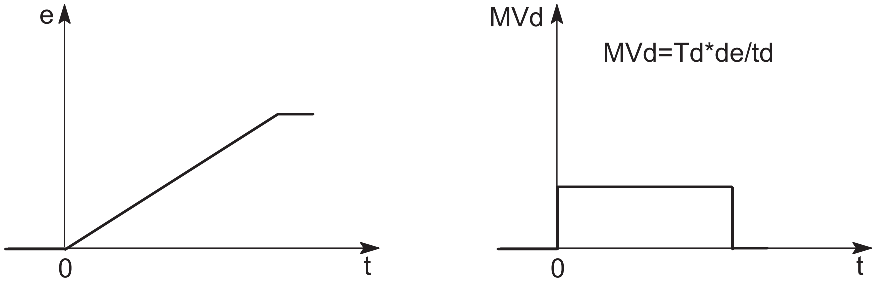
3. Derivative part (D part)

The derivative part produces an output quantity that corresponds to the time derivation of the input quantity. The derivative time corresponds to the weighting of the derived input quantity.

A derivative component can be an RC-bleeder (capacitor hooked up in series and resistance in parallel), for example.

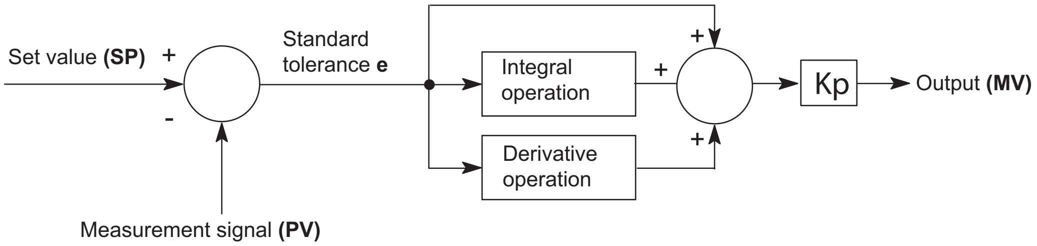


Example: Input quantity **e** and the output quantity **MVd** produced.



4. PID controller

A PID controller is a combination of a P component, an I component and a D component. When the parameters Kp, Ti and Td are optimally adjusted, a PID controller can quickly control and maintain a quantity at a predetermined set value.



Reference equations for calculating the controller output MV

The following equations are used to calculate the controller output MV under the following conditions:

In general:
The output value at time period **n** is calculated from the previous output value (n-1) and the change in the output value in this time interval.

$$MV_n = MV_{n-1} + \Delta MV$$

Reverse operation PI-D Control = 16#X000

$$\Delta MV = Kp \times \left[(e_n - e_{n-1}) + e_n \times \frac{Ts}{Ti} + \Delta D_n \right]$$

$$e_n = SP_n - PV_n$$

$$\Delta Dn = (\eta\beta - 1)D_{n-1} + \beta(PV_{n-1} - PV_n)$$

$$\eta = \frac{1}{8}(\text{constant})$$

$$\beta = \frac{Td}{(Ts + \eta Td)}$$

Forward operation PI-D Control = 16#X001

$$\Delta MV = Kp \times \left[(e_n - e_{n-1}) + e_n \times \frac{Ts}{Ti} + \Delta D_n \right]$$

$$e_n = PV_n - SP_n$$

$$\Delta Dn = (\eta\beta - 1)D_{n-1} + \beta(PV_n - PV_{n-1})$$

$$\eta = \frac{1}{8}(\text{constant})$$

$$\beta = \frac{Td}{(Ts + \eta Td)}$$

Reverse operation I-PD Control = 16#X002

$$\Delta MV = Kp \times \left[\left(PV_{n-1} - PV_n\right) + e_n \times \frac{Ts}{Ti} + \Delta D_n \right]$$

$$e_n = SP_n - PV_n$$

$$\Delta Dn = \left(\eta\,\beta - 1\right)D_{n-1} + \beta\left(PV_{n-1} - PV_n\right)$$

$$\eta = \frac{1}{8}(\text{constant})$$

$$\beta = \frac{Td}{(Ts + \eta Td)}$$

Forward operation I-PD Control = 16#X003

$$\Delta MV = Kp \times \left[\left(PV_n - PV_{n-1}\right) + e_n \times \frac{Ts}{Ti} + \Delta D_n \right]$$

$$e_n = PV_n - SP_n$$

$$\Delta Dn = \left(\eta\,\beta - 1\right)D_{n-1} + \beta\left(PV_n - PV_{n-1}\right)$$

$$\eta = \frac{1}{8}(\text{constant})$$

$$\beta = \frac{Td}{(Ts + \eta Td)}$$

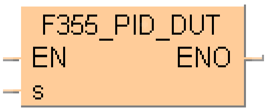
PID processing instructions:

- PID_FB_DUT (see page 953)
- PID_FB (see page 951)
- F355_PID_DUT (see page 941)

F355_PID_DUT

PID processing instruction

Description The PID processing instruction is used to regulate a process (e.g. a heater) given a measured value (e.g. temperature) and a predetermined output value (e.g. 20°C).



The function calculates a PID algorithm whose parameters are determined in a data table in the form of an ARRAY with 30 elements that is entered at input **s**.

The required data table PID_DUT_31 contains the following parameters (for details, please refer to the DUT PID_DUT_31 in the online help):

Parameter	Data type	Function		
Control	WORD	Control mode 16#X000 Inverse PI-D control 16#X001 Forward PI-D control 16#X002 Inverse I-PD control 16#X003 Forward I-PD control		
			Range	**Unit**
SP	INT	Set point value	0-10000	
PV		Process value	0-10000	
MV		Manipulated value	0-10000	
LowerLimit		MV lower limit	0-10000	
UpperLimit		MV upper limit	1-10000	
Kp		Proportional gain	1-9999	0.1
Ti		Integral time	1-30000	0.1s
Td		Derivative time	1-10000	0.1s
Ts		Sampling time	1-6000	0.01s
AT_Progress		Auto-tuning progress	0-5	
Dummies	ARRAY [11..30] OF WORD	are utilized internally by the PID controller		

PLC types **Availability of F355_PID_DUT (see page 1325)**

Data types

Variable	Data type	Function
s	PID_DUT_31	Detailed explanation of parameters

Operands

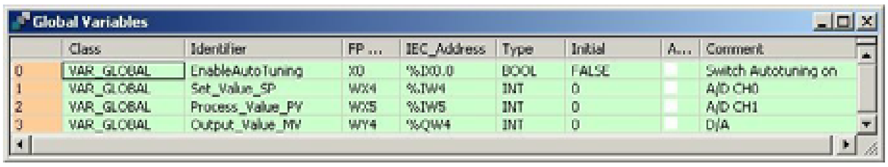
For	Relay				T/C		Register			Constant
s	-	-	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the parameter settings are outside the permissible range
R9008	%MX0.900.8	for an instant	

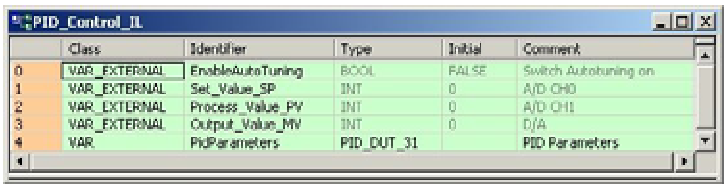
Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

GVL In the global variable list, you define variables that can be accessed by all POUs in the project.



	Class	Identifier	FP ...	IEC_Address	Type	Initial	A...	Comment
0	VAR_GLOBAL	EnableAutoTuning	X0	%IX0.0	BOOL	FALSE		Switch Autotuning on
1	VAR_GLOBAL	Set_Value_SP	WX4	%IW4	INT	0		A/D CH0
2	VAR_GLOBAL	Process_Value_PV	WX5	%IW5	INT	0		A/D CH1
3	VAR_GLOBAL	Output_Value_MV	WY4	%QW4	INT	0		D/A

POU header All input and output variables used for programming this function have been declared in the POU header.



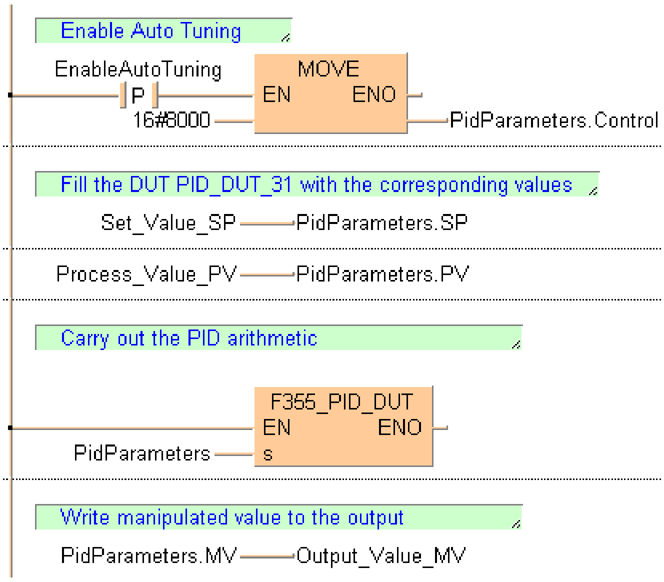
	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	EnableAutoTuning	BOOL	FALSE	Switch Autotuning on
1	VAR_EXTERNAL	Set_Value_SP	INT	0	A/D CH0
2	VAR_EXTERNAL	Process_Value_PV	INT	0	A/D CH1
3	VAR_EXTERNAL	Output_Value_MV	INT	0	D/A
4	VAR	PidParameters	PID_DUT_31		PID Parameters

In the initialization of the variable PidParameters of the data type PID_DUT_31, the MV upper limit is set to 4000. The proportional gain Kp is initially set at 80 (8), Ti and Td at 200 (20s) and the sampling time Ts at 100 (1s).

Body The standard function MOVE copies the value 16#8000 to the member Control of the DUT PidParameters when the variable EnableAutoTuning turns from FALSE to TRUE (i.e. activates the control mode auto-tuning in the function F355_PID_DUT).
The variables Set_Value_SP and Process_Value_PV are assigned to the members SP and PV of the DUT PidParameters. They receive their values from the A/D converter channel 0 and 1.

Because the F355_PID_DUT function block has an EN output connected directly to the power rail, the function is carried out when the PLC is in RUN mode. The calculated controller output stored by the member MV of the DUT PidParameters is assigned to the variable Output_Value_MV. Its value is returned via a D/A converter from the PLC to the output of the system.

LD



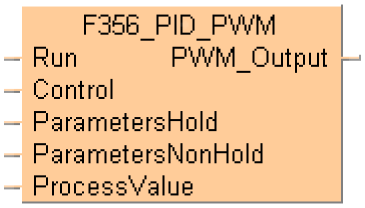
ST When programming with structured text, enter the following:

```
(* Auto Tuning: *)
if DF(EnableAutoTuning) then
      PidParameters.Control:=16#8000;
end_if;

(* Fill the DUT PidParameters with the corresponding input values: *)
PidParameters.SP:=Set_Value_SP;
PidParameters.PV:=Process_Value_PV;

(* Carry out the PID arithmetic: *)
F355_PID_DUT(PidParameters);

(* Write the manipulated value to the output: *)
Output_Value_MV:=PidParameters.MV;
```

F356_PID_PWM

PID processing with optional PWM output

Description PID processing is performed to keep the process value PV as close as possible to the set point value SP. In contrast to **F355_PID_DUT** (see page 941), this instruction enables a PWM output (on-off output). Auto-tuning is also available to automatically calculate the PID control data Kp, Ti, and Td.



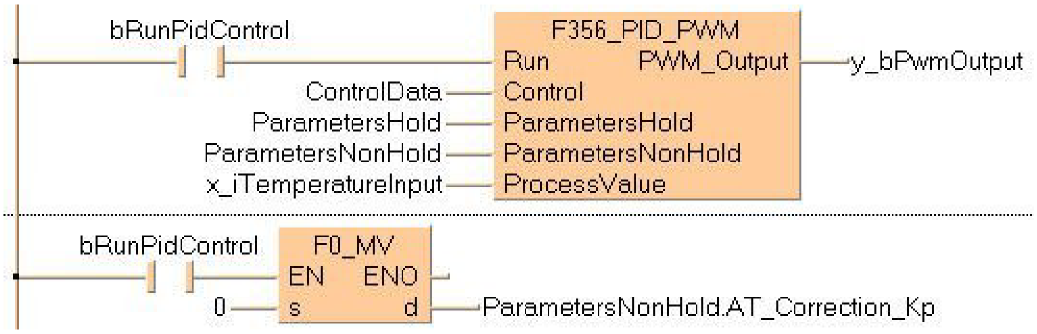
Abbreviations used when describing PID processing

Abbreviation	What it stands for	Also know as
PV	Process value	Actual value, measured value
SP	Set point value	Target value, set value
MV	Manipulated value	Output value, manipulated variable
Ts	Sampling time	Cycle time
Ti	Integral time	-
Td	Derivative time	-
Kp	Proportional gain	-
AT	Auto-tuning	-

General programming information

1. When the input at **Run** is executed, the data in the argument **ParametersNonHold** is initialized.
If you want a value in the DUT to use non-default values, write the values into the DUT using a MOVE instruction, for example, which must be triggered continuously by a TRUE condition.
2. F356_PID_PWM must be executed once and only once per scan. Therefore, do not execute F356_PID_PWM in interrupt programs or loops.
3. Do not turn the execution condition to FALSE during PID processing. Otherwise, PID processing will be disabled.
4. If you do not want parallel PWM output cycles, e.g. to enable control of multiple objects, delay the start-up times accordingly, e.g. by employing a timer instruction.

Example:



PLC types **Availability of F356_PID_PWM (see page 1325)**

☞ **The period (cycle) of the PWM output is the sampling time Ts (the frequency of the PWM output is 1/Ts) and the duty is the manipulated value MV in 0.01% units, e.g. MV = 10000 means a duty of 100%.**

Data types

Variable	Data type	Function
Run	BOOL	Start condition
Control	F356_Control_DUT (see page 948)	Control data
Parameters Hold	F356_Parameters_Hold_DUT (see page 948)	PID control parameters
Parameters NonHold	F356_Parameters_NonHold_DUT (see page 949)	Manipulated value MV, additional control mode area, auto-tuning related area and working area
ProcessValue	INT	Process value (-30000–30000)
PWM_Output (see note)	BOOL	Pulse-width modulated output (optional, instead of manipulated value output)

Operands

For	Relay				T/C		Register			Constant
Control	-	WY	WR	WL	SV	EV	DT	LD	FL	-
Parameters	WX	WY	WR	WL	SV	EV	DT	LD	FL	-
Process Values	-	WY	WR	WL	SV	EV	DT	LD	FL	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ any parameter of F356_Parameters_NonHold_DUT is out of range
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	▪ the area specified with **UpperLimit** or **LowerLimit** is out of range

Detailed information:

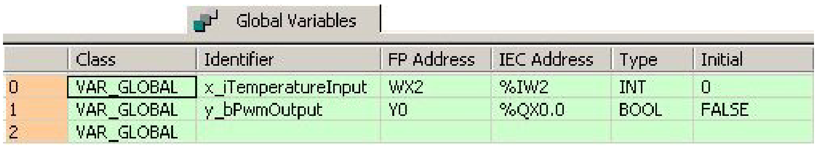
- Control conditions: F356_Parameters_Hold_DUT (see page 948)
- Set point value SP and the control parameters: F356_Parameters_NonHold_DUT (see page 949)

Additional notes on auto-tuning

- The members **AT_Progress** in F356_Parameters_NonHold_DUT (see page 949) and **b1_AT_Complete** in F356_Control_DUT (see page 948) are cleared at the rising edge of the auto-tuning signal.
- When auto-tuning has completed successfully, the element **b1_AT_Complete** of F356_Control_DUT (see page 948) is set, and the auto-tuning done code is stored in the element **AT_Progress** of F356_Parameters_NonHold_DUT (see page 949).
- When auto-tuning is aborted, the parameters of Kp, Ti, and Td are not changed.

Example In this example, the same POU header is used for all programming languages.

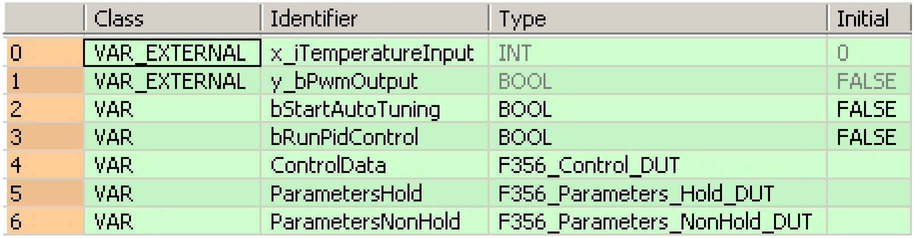
GVL In the global variable list, all values of global inputs and outputs are declared that are used for programming this function.



Global Variables

	Class	Identifier	FP Address	IEC Address	Type	Initial
0	VAR_GLOBAL	x_iTemperatureInput	WX2	%IW2	INT	0
1	VAR_GLOBAL	y_bPwmOutput	Y0	%QX0.0	BOOL	FALSE
2	VAR_GLOBAL					

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR_EXTERNAL	x_iTemperatureInput	INT	0
1	VAR_EXTERNAL	y_bPwmOutput	BOOL	FALSE
2	VAR	bStartAutoTuning	BOOL	FALSE
3	VAR	bRunPidControl	BOOL	FALSE
4	VAR	ControlData	F356_Control_DUT	
5	VAR	ParametersHold	F356_Parameters_Hold_DUT	
6	VAR	ParametersNonHold	F356_Parameters_NonHold_DUT	

Body Specify the member SP (set point value) of F356_Parameters_Hold_DUT (see page 948) before operation.

When bRunPidControl turns on, the work area specified with the F356_Parameters_NonHold_DUT (see page 949) will be initialized. However, only the member MV (manipulated value) can be held depending on the status of the flag b2_HoldMV of F356_Control_DUT (see page 948).

The default control conditions are:

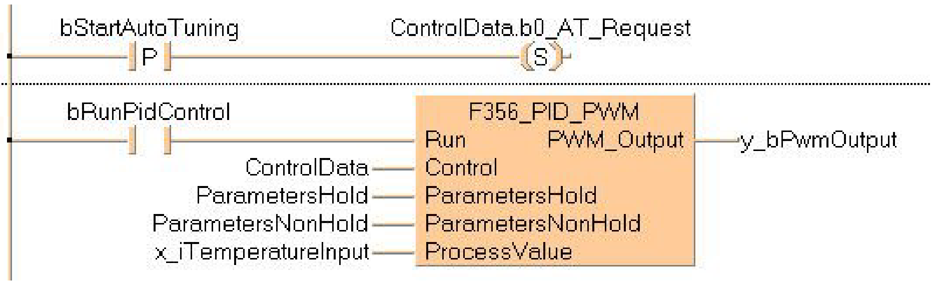
- Cycle time = 1s
- Inverse I-PD control (heating)
- PWM resolution = 1000.

PID control starts from the next scan, and PWM output is executed for **PWM_Output**.

If the member flag **b0_AT_Request** of **ControlData**, a DUT with overlapping elements, is set, auto-tuning begins. When auto-tuning has completed successfully, the member flag **b1_AT_Complete** of **ControlData** is set and Kp, Ti and Td are set for the PID control. If **bRunPidControl** is still on, it will change to PID control automatically and the PWM output will be executed.

☞ **If the execution condition bRunPidControl has turned to FALSE during PID control, PWM_Output also turns off. However, only the member MV (manipulated value) can be held depending on the status of the flag b2_HoldMV of F356_Control_DUT (see page 948).**

LD

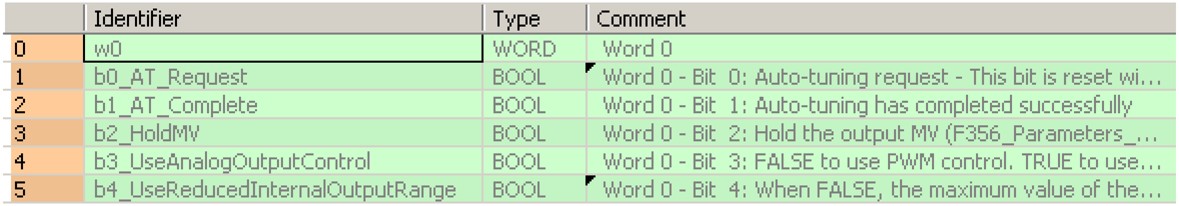


ST When programming with structured text, enter the following:

```
(* Auto Tuning: *)
if DF(bStartAutoTuning) then
       ControlData.b0_AT_Request:=TRUE;
end_if;
y_bPwmOutput:=F356_PID_PWM(    Run := bRunPidControl,
       Control := ControlData,
       ParametersHold := ParametersHold,
       ParametersNonHold := ParametersNonHold,
       ProcessValue := x_iTemperatureInput);
```

F356_Control_DUT

This data type, a DUT with overlapping elements (see page 53), is predefined in the FP Library and is used by the function F356_PID_PWM (see page 945).



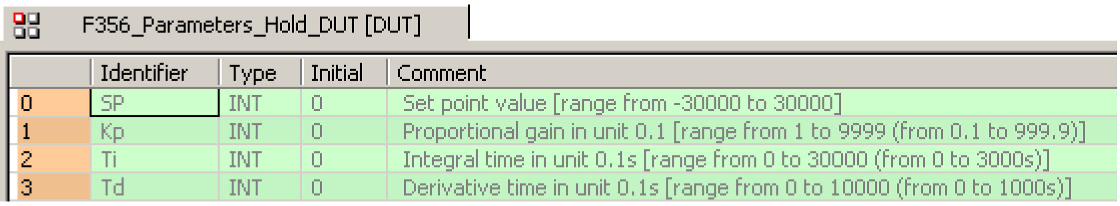
	Identifier	Type	Comment
0	w0	WORD	Word 0
1	b0_AT_Request	BOOL	Word 0 - Bit 0: Auto-tuning request - This bit is reset wi...
2	b1_AT_Complete	BOOL	Word 0 - Bit 1: Auto-tuning has completed successfully
3	b2_HoldMV	BOOL	Word 0 - Bit 2: Hold the output MV (F356_Parameters_...
4	b3_UseAnalogOutputControl	BOOL	Word 0 - Bit 3: FALSE to use PWM control. TRUE to use...
5	b4_UseReducedInternalOutputRange	BOOL	Word 0 - Bit 4: When FALSE, the maximum value of the...

We recommend specifying the non-hold type area.

Identifier	Description
w0	Since this is a DUT with overlapping elements, the BOOL members occupy the same data areas as the WORD member **w0**. Therefore by using **w0** you can simultaneously access all bits.
b0_AT_Request **(bit 0)**	When set, auto- tuning is requested. This bit is reset with the instruction F356_PID_PWM when auto-tuning is complete. Reset this bit to cancel auto-tuning. When not set, PID control will be executed.
b1_AT_Complete **(bit 1)**	When set, auto-tuning has been completed successfully.
b2_HoldMV **(bit 2)**	When set, the manipulated value output is held by switching F356_PID_PWM (see page 945) from off to on.
b3_UseAnalogOutputControl **(bit 3)**	FALSE to use PWM control. TRUE to use an analog output unit for output. In this case transmit the output value (F356_Parameters_NonHold_DUT.MV) to WY of an analog output unit.
b4_UseReducedInternalOutputRange **(bit 4)**	When FALSE, the maximum value of the internal output is the output upper limit value +20% of the output range (output upper limit value - output lower limit value), and the minimum value is the output lower limit value -20% of the output range. When TRUE, the maximum value of the internal output is the output upper limit value, and the minimum value is the output lower limit value. The output upper limit value is specified by F356_Parameters_NonHold_DUT.UpperLimit, and the output lower limit value is specified by F356_Parameters_NonHold_DUT.LowerLimit.
Bits 5–F	Are reserved and normally 0.

F356_Parameters_Hold_DUT

This data type is predefined in the FP Library and is used by the function F356_PID_PWM (see page 945).



F356_Parameters_Hold_DUT [DUT]

	Identifier	Type	Initial	Comment
0	SP	INT	0	Set point value [range from -30000 to 30000]
1	Kp	INT	0	Proportional gain in unit 0.1 [range from 1 to 9999 (from 0.1 to 999.9)]
2	Ti	INT	0	Integral time in unit 0.1s [range from 0 to 30000 (from 0 to 3000s)]
3	Td	INT	0	Derivative time in unit 0.1s [range from 0 to 10000 (from 0 to 1000s)]

This DUT specifies the control parameter (4 words). We recommend allocating the area used by this data type to the hold-type operation memory.

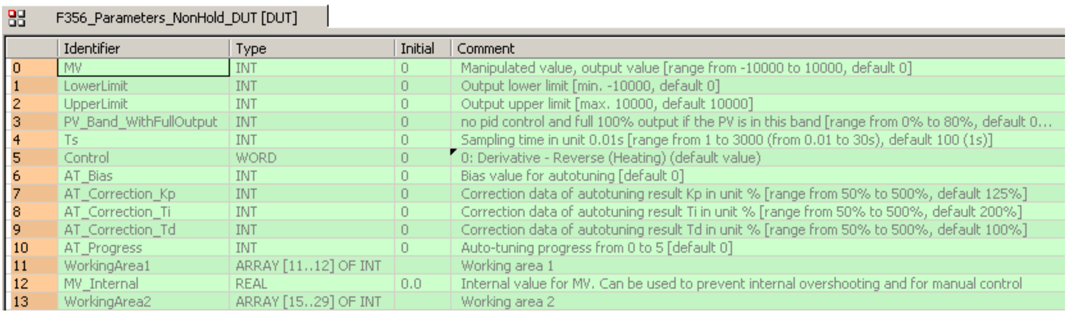
Variable	Comment	Setting range
SP	Set point value	-30000–30000
Kp	Stores the proportional gain Kp. After auto-tuning has been completed, it is automatically set.	1–9999 (0.1–999.9)
Ti	Stores the integral time Ti. This value is automatically set after auto-tuning has been completed.	0–30000 (0-3000s)
Td	Stores the derivative time Td. This value is automatically set after auto-tuning has been completed.	0–10000 (0–1000s)

If the parameters Kp, Ti, and Td are all 0 when PID operation has started, they are initialized at 1, 1, and 0, respectively, and operation continues.

If any of the parameters Kp, Ti, or Td is out of range when auto-tuning has started, they are initialized at 1, 1, and 0, respectively, and auto-tuning continues.

F356_Parameters_NonHold_DUT

This data type is predefined in the FP Library and is used by the function F356_PID_PWM (see page 945).



F356_Parameters_NonHold_DUT [DUT]

	Identifier	Type	Initial	Comment
0	MV	INT	0	Manipulated value, output value [range from -10000 to 10000, default 0]
1	LowerLimit	INT	0	Output lower limit [min. -10000, default 0]
2	UpperLimit	INT	0	Output upper limit [max. 10000, default 10000]
3	PV_Band_WithFullOutput	INT	0	no pid control and full 100% output if the PV is in this band [range from 0% to 80%, default 0...
4	Ts	INT	0	Sampling time in unit 0.01s [range from 1 to 3000 (from 0.01 to 30s), default 100 (1s)]
5	Control	WORD	0	0: Derivative - Reverse (Heating) (default value)
6	AT_Bias	INT	0	Bias value for autotuning [default 0]
7	AT_Correction_Kp	INT	0	Correction data of autotuning result Kp in unit % [range from 50% to 500%, default 125%]
8	AT_Correction_Ti	INT	0	Correction data of autotuning result Ti in unit % [range from 50% to 500%, default 200%]
9	AT_Correction_Td	INT	0	Correction data of autotuning result Td in unit % [range from 50% to 500%, default 100%]
10	AT_Progress	INT	0	Auto-tuning progress from 0 to 5 [default 0]
11	WorkingArea1	ARRAY [11..12] OF INT		Working area 1
12	MV_Internal	REAL	0.0	Internal value for MV. Can be used to prevent internal overshooting and for manual control
13	WorkingArea2	ARRAY [15..29] OF INT		Working area 2

This DUT specifies the manipulated value (MV) and the control parameters (4 words).

Variable	Comment	Default value	Setting range
MV	Stores the manipulated value (output value)	0	-10000–10000
LowerLimit	Sets the lower limit of the manipulated value MV	0	min. -10000
UpperLimit	Sets the upper limit of the manipulated value MV	10000	max. 10000
PV_Band_WithFullOutput	No PID control is performed and the output is at 100% until the defined level (0–80%) of the set point value has been reached.	0	0–80%

Ts	Sets the sampling time for updating the measured input values. Unit = 0.01s Also sets the PWM output period.	100 (1s)	1–3000 (0.01–30s)
ControlMode	0 Inverse PI-D control, e.g. heating 1 Forward PI-D control, e.g. cooling 2 Inverse I-PD control, e.g. heating 3 Forward I-PD control, e.g. cooling	0	0–3
AT_Bias	Sets a bias value for performing auto-tuning	0	min. 0
AT_Correction_Kp	Sets the correction value of the auto-tuning result for Kp	125%	50–500%
AT_Correction_Ti	Sets the correction value of the auto-tuning result for Ti	200%	50–500%
AT_Correction_Td	Sets the correction value of the auto-tuning result for Td	100%	50–500%
AT_Progress	Stores the auto-tuning progress	0	0–5
WorkingArea	Working area of up to 30 words for PID processing and auto-tuning	0	

When the execution condition has turned on, the operation work area is initialized.

◆NOTE

When the execution condition turns to TRUE, the default value is set. The manipulated value MV is only output in the range of the specified upper and lower limit.

Detailed information on the setting method:

PV_Band_WithFullOutput

Define the percentage of the set point value at which PID control should start. Below this level, output is at 100%.

For example, if PV_Band_WithFullOutput is set to 80% and the measured value (process value PV) is only at 50% of the set value, the output will be at 100%. It will remain at 100% until the measured value reaches 80% of the set value, at which point PID control will start.

The amount of the percentage determines how quickly the set value will be reached.

Fine adjustment of auto-tuning

When auto-tuning has completed, the parameters for Kp, Ti and Td are stored in the elements of F356_Parameters_Hold_DUT (see page 948). For fine adjustment, you can now correct the result of auto-tuning with the parameters AT_Correction_Kp, AT_Correction_Ti and AT_Correction_Td.

◆EXAMPLE

Set AT_Correction_Kp to 200 (i.e. 200%): perform auto-tuning to correct Kp to double its value.

Set AT_Correction_Ti to 125 (i.e. 125%): perform auto-tuning to correct Ti to 1.25 times its value.

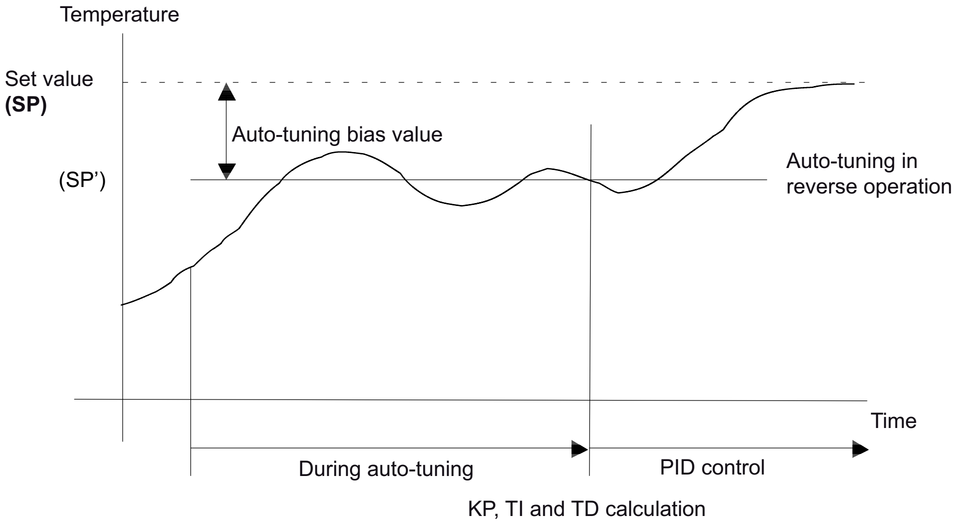
Set AT_Correction_Td to 75 (i.e. 75%): perform auto-tuning to correct Td to 0.75 times its value.

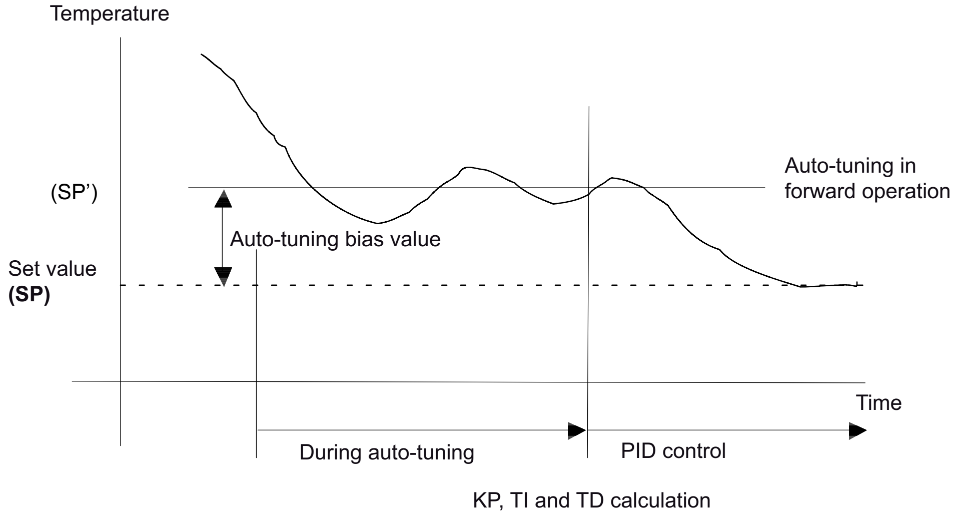
Auto-tuning bias value

If a bias value has been set, auto-tuning will be performed with a temporary set point value SP'.

In reverse operation, SP' is the difference of the set point value SP and the auto-tuning bias value. The auto-tuning bias value can be used to control excessive temperature rise during auto-tuning.

In forward operation, SP' is the sum of the set point value SP and the auto-tuning bias value.





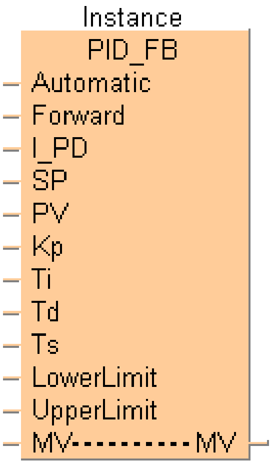
◆NOTE

Auto-tuning is peformed with SP' even if the measured process value is close to the set point value SP when auto-tuning starts.

PID_FB

PID processing instruction

Description This implementation allows you to set the parameters of F355_PID directly using arguments:



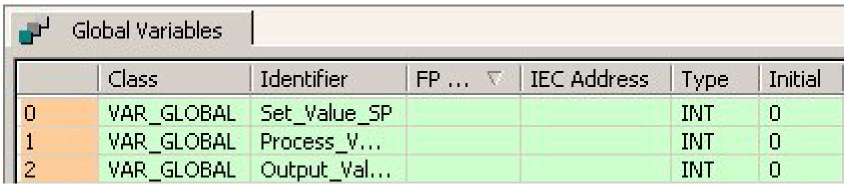
Data types

Input variables (VAR_INPUT):		
Variable	**Data type**	**Function**
Automatic	BOOL	FALSE: Manual setting of MV possible TRUE: Automatic PID controlled MV
Forward		FALSE: Inverse control (heating) TRUE: Forward control (cooling)
I_PD		FALSE: PI-D control TRUE: I-PD control
SP	INT	Set point value, range 0-10000
PV		Process value, range 0-10000
Kp		Proportional gain, range: 1-9999, unit: 0.1
Ti		Integral time, range: 1-30000, unit: 0.1s
Td		Derivative time, range: 1-10000, unit: 0.1s
Ts		Sampling time, range: 1-6000, unit: 0.01s
LowerLimit		MV lower limit, range: 0-10000
UpperLimit		MV upper limit, range: 1-10000
Input/output variable (VAR_IN_OUT):		
MV	Manipulated value	

- **Auto-tuning is not possible using PID_FB. For this, use PID_FB_DUT (see page 953).**
- **The value for MV can be assigned externally either when the program is initialized or when the value of Automatic is FALSE.**
- **In order to achieve maximum resolution and minimum dead time beyond LowerLimit and UpperLimit, their values should, if possible, cover the entire range of 0–10000.**

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

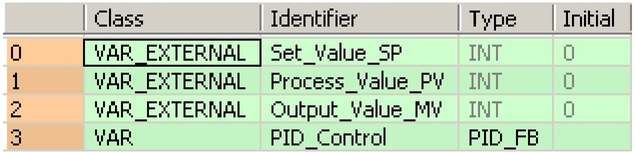
GVL In the global variable list all global input and output values are declared that are used to program the function. The addresses are depending on the respective PLC-Type.



Global Variables

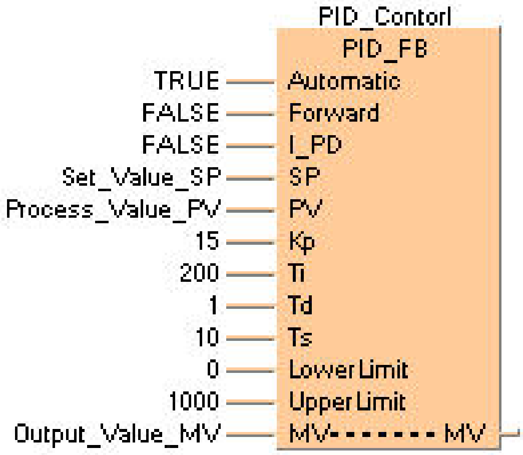
	Class	Identifier	FP ...	IEC Address	Type	Initial
0	VAR_GLOBAL	Set_Value_SP			INT	0
1	VAR_GLOBAL	Process_V...			INT	0
2	VAR_GLOBAL	Output_Val...			INT	0

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR_EXTERNAL	Set_Value_SP	INT	0
1	VAR_EXTERNAL	Process_Value_PV	INT	0
2	VAR_EXTERNAL	Output_Value_MV	INT	0
3	VAR	PID_Control	PID_FB	

LD



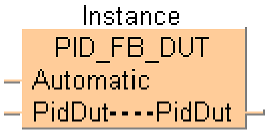
ST When programming with structured text, enter the following:

```
PID_Control( Automatic:= TRUE,
       Forward:= FALSE,
       I_PD:= FALSE,
       SP:= Set_Value_SP,
       PV:= Process_Value_PV,
       Kp:= 15,
       Ti:= 200,
       Td:= 1,
       Ts:= 10,
       LowerLimit:= 0,
       UpperLimit:= 1000,
       MV:= Output_Value_MV);
```

PID_FB_DUT

PID processing instruction

Description This implementation allows you to access the F355_PID instruction via the structure PID_DUT.



This structure defined in System Libraries / FP Library / DUTs contains the following parameters (for details, please refer to the DUT PID_DUT):

Data types

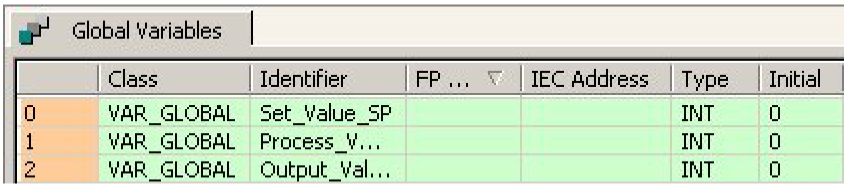
Input variables (VAR_INPUT):		
Variable	**Data type**	**Function**
Automatic	BOOL	FALSE: Manual setting of MV possible TRUE: Automatic PID controlled MV
Input/Output variable (VAR_IN_OUT):		
PidDut	PID_DUT	

☞

- **You may not enter the DUT PID_DUT a second time under DUTs of the current project.**
- **The value for MV can be assigned externally either when the program is initialized or when the value of Automatic is FALSE.**
- **In order to achieve maximum resolution and minimum dead time beyond LowerLimit and UpperLimit, these values should, if possible, cover the entire range of 0–10000.**

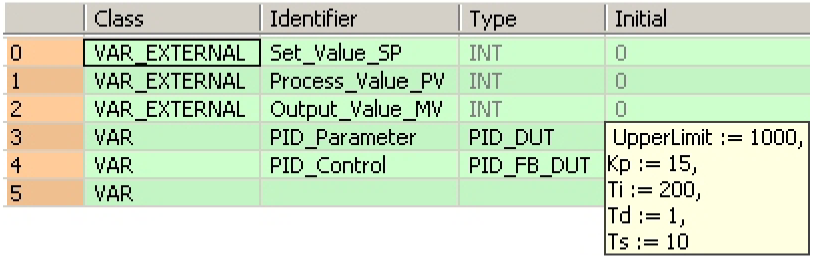
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

GVL In the global variable list all global input and output values are declared that are used to program the function. The addresses are depending on the respective PLC-Type.


Global Variables

	Class	Identifier	FP ...	IEC Address	Type	Initial
0	VAR_GLOBAL	Set_Value_SP			INT	0
1	VAR_GLOBAL	Process_V...			INT	0
2	VAR_GLOBAL	Output_Val...			INT	0

POU header All input and output variables used for programming this function have been declared in the POU header.

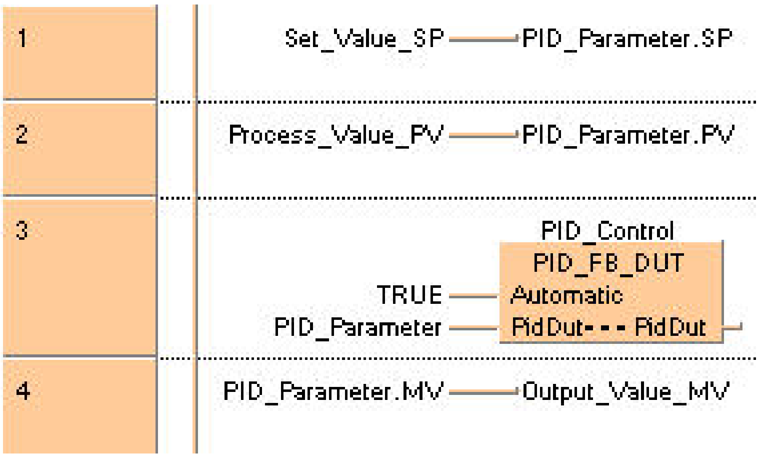


	Class	Identifier	Type	Initial
0	VAR_EXTERNAL	Set_Value_SP	INT	0
1	VAR_EXTERNAL	Process_Value_PV	INT	0
2	VAR_EXTERNAL	Output_Value_MV	INT	0

3	VAR	PID_Parameter	PID_DUT	UpperLimit := 1000,
				Kp := 15,
				Ti := 200,
				Td := 1,
				Ts := 10

4	VAR	PID_Control	PID_FB_DUT	
5	VAR			

LD



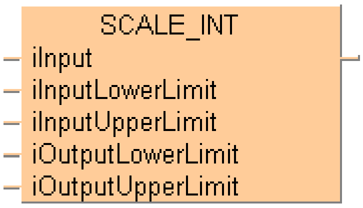
ST When programming with structured text, enter the following:

```
    PID_Parameter.SP := Set_Value_SP;
    PID_Parameter.PV := Process_Value_PV;
    PID_Control( Automatic:= TRUE,
           PidDut:= PID_Parameter);
Output_Value_MV := PID_Parameter.MV;
```

SCALE_INT

Scales INTEGER data

Description This instruction scales an INTEGER value between a lower and an upper limit to an INTEGER output value. Use WITHIN_LIMITS (see page 111) to check if the input value is within the specified limits.



See also:

- F282_SCAL (see page 468)
- F283_DSCAL (see page 471)

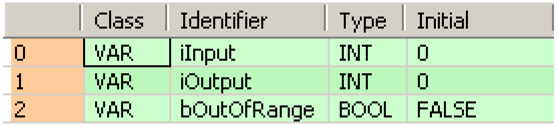
PLC types see page 1330

Data types

Variable	Data type	Function
iInput	INT	Input signal
iInputLowerLimit		Lower limit of the input range
iInputUpperLimit		Upper limit of the input range
iOutputLowerLimit		Output value assigned to the lower limit of the input range (can be higher than **iOutputUpperLimit**)
iOutputUpperLimit		Output value assigned to the upper limit of the input range (can be lower than **iOutputLowerLimit**)
Output variable		Scaled output signal

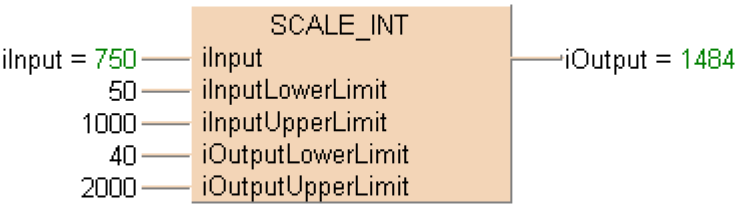
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iInput	INT	0
1	VAR	iOutput	INT	0
2	VAR	bOutOfRange	BOOL	FALSE

LD



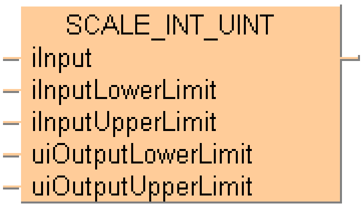
ST When programming with structured text, enter the following:

```
if (bScale) then
  SCALE_INT(iInput := iInput,
      iInputLowerLimit := 50,
      iInputUpperLimit := 1000,
      iOutputLowerLimit := 40,
      iOutputUpperLimit := 2000,
      iOutput => iOutput);
end_if;
```

SCALE_INT_UINT

Scale INTEGER data into unsigned INTEGER data

Description This instruction scales an INTEGER value between a lower and an upper limit to an unsigned INTEGER output value. Use WITHIN_LIMITS (see page 111) to check if the input value is within the specified limits.



See also:

- F282_SCAL (see page 468)
- F283_DSCAL (see page 471)

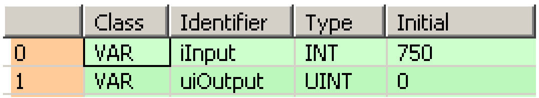
PLC types see page 1330

Data types

Variable	Data type	Function
iInput	INT	Input signal
iInputLowerLimit		Lower limit of the input range
iInputUpperLimit		Upper limit of the input range
uiOutputLowerLimit	UINT	Output value assigned to the upper limit of the input range (can be lower than **uiOutputLowerLimit**)
uiOutputUpperLimit		Output value assigned to the lower limit of the input range (can be higher than **uiOutputUpperLimit**)
Output variable		Scaled output signal

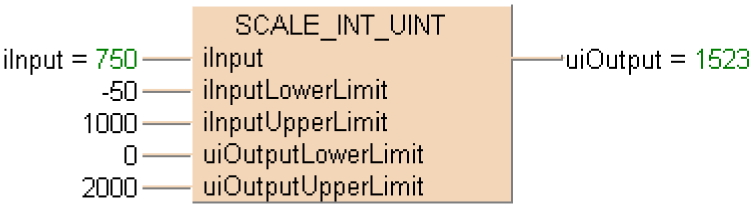
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iInput	INT	750
1	VAR	uiOutput	UINT	0

LD



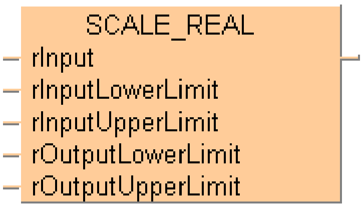
ST When programming with structured text, enter the following:

```
if (bScale) then
  SCALE_INT_UINT(iInput := iInput,
           iInputLowerLimit := -50,
           iInputUpperLimit := 1000,
           uiOutputLowerLimit := 0,
           uiOutputUpperLimit := 2000,
           uiOutput => uiOutput);
end_if;
```

SCALE_REAL

Scale REAL data

Description This instruction scales a REAL value between a lower and an upper limit to a REAL output value. Use WITHIN_LIMITS (see page 111) to check if the input value is within the specified limits.



See also:

- F354_FSCAL (see page 478)

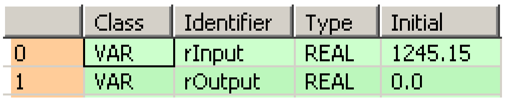
PLC types see page 1330

Data types

Variable	Data type	Function
rInput	REAL	Input signal
rInputLowerLimit		Lower limit of the input range
rInputUpperLimit		Upper limit of the input range
rOutputLowerLimit		Output value assigned to the upper limit of the input range (can be lower than **rOutputLowerLimit**)
rOutputUpperLimit		Output value assigned to the lower limit of the input range (can be higher than **rOutputUpperLimit**)
Output variable		Scaled output signal

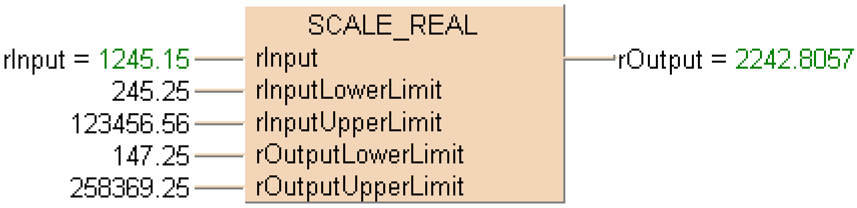
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	rInput	REAL	1245.15
1	VAR	rOutput	REAL	0.0

LD



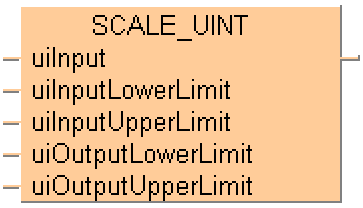
ST When programming with structured text, enter the following:

```
if (bScale) then
  SCALE_REAL(rInput := rInput,
            rInputLowerLimit := 245.25,
            rInputUpperLimit := 123456.56,
            rOutputLowerLimit := 147.25,
            rOutputUpperLimit := 258369.25,
            rOutput => rOutput);
end_if;
```

SCALE_UINT

Scale UINT data

Description This instruction scales an unsigned INTEGER value between a lower and an upper limit to an unsigned INTEGER output value. Use WITHIN_LIMITS (see page 111) to check if the input value is within the specified limits.



See also:

- F282_SCAL (see page 468)
- F283_DSCAL (see page 471)

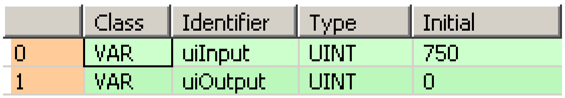
PLC types see page 1330

Data types

Variable	Data type	Function
uiInput	UINT	Input signal
uiInputLowerLimit		Lower limit of the input range
uiInputUpperLimit		Upper limit of the input range
uiOutputLowerLimit		Output value assigned to the lower limit of the input range (can be higher than **uiOutputUpperLimit**)
uiOutputUpperLimit		Output value assigned to the upper limit of the input range (can be lower than **uiOutputLowerLimit**)
Output variable		Scaled output signal

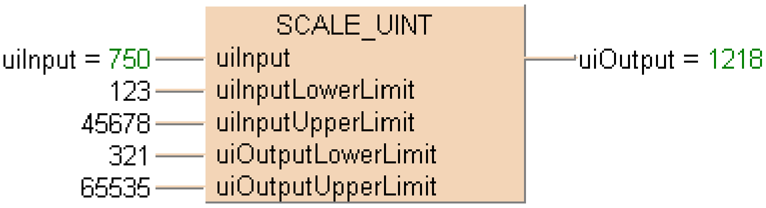
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	uiInput	UINT	750
1	VAR	uiOutput	UINT	0

LD



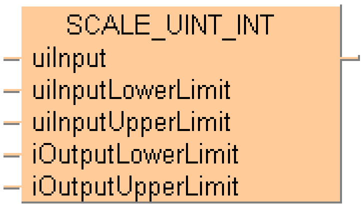
ST When programming with structured text, enter the following:

```
if (bScale) then
  SCALE_UINT(uiInput := uiInput,
        uiInputLowerLimit := 123,
        uiInputUpperLimit := 45678,
        uiOutputLowerLimit := 321,
        uiOutputUpperLimit := 65535,
        uiOutput => uiOutput);
end_if;
```

SCALE_UINT_INT

Scales UINT input data to INT output data

Description This instruction scales an unsigned INTEGER value between a lower and an upper limit to an INTEGER output value. Use WITHIN_LIMITS (see page 111) to check if the input value is within the specified limits.



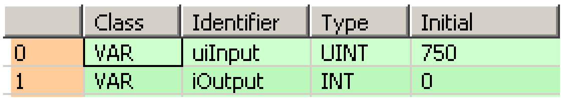
PLC types see page 1330

Data types

Variable	Data type	Function
uiInput	UINT	Input signal
uiInputLowerLimit		Lower limit of the input range
uiInputUpperLimit		Upper limit of the input range
iOutputLowerLimit	INT	Output value assigned to the upper limit of the input range (can be lower than **iOutputLowerLimit**)
iOutputUpperLimit		Output value assigned to the lower limit of the input range (can be higher than **iOutputUpperLimit**)
Output variable		Scaled output signal

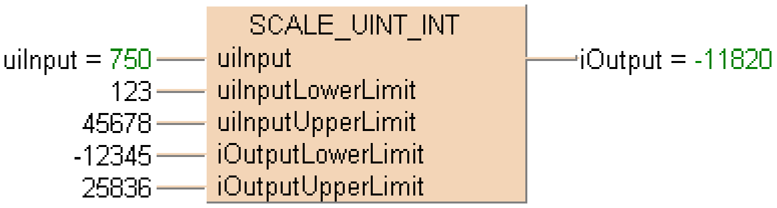
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	uiInput	UINT	750
1	VAR	iOutput	INT	0

LD



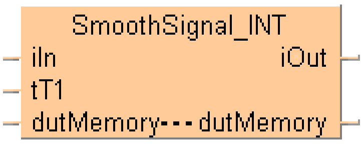
ST When programming with structured text, enter the following:

```
if (bScale) then
  SCALE_UINT_INT(uiInput := uiInput,
            uiInputLowerLimit := 123,
            uiInputUpperLimit := 45678,
            iOutputLowerLimit := -123,
            iOutputUpperLimit := 25836,
            iOutput => iOutput);
end_if;
```

SmoothSignal_INT

Smooth INT signals

Description This instructions uses a 1st order delay time **tT1** to smooth the INTEGER input value at **iIN**.



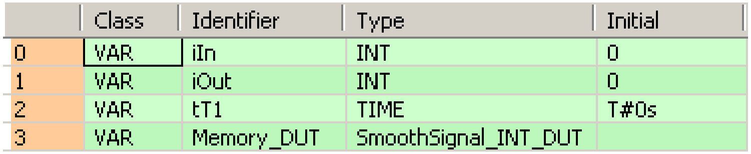
PLC types see page 1330

Data types

Input variable	Data type	Function
iIn	INT	Input signal
tT1	TIME	Time constant of the 1st order low-pass filter
Input/output variable		
dutMemory	dutMemory	Instance-dependent data memory structure, which serves as the internal memory of the function. As with the instance name of a function block, it may be neither initialized nor written in the body!
Output variable		
iOut	INT	Output signal

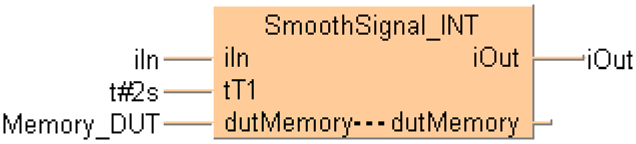
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iIn	INT	0
1	VAR	iOut	INT	0
2	VAR	tT1	TIME	T#0s
3	VAR	Memory_DUT	SmoothSignal_INT_DUT	

LD

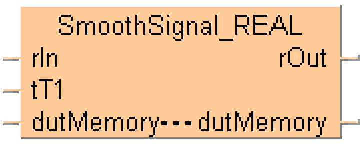


ST When programming with structured text, enter the following:

```
SmoothSignal_INT(iIn := iIn,
        tT1 := t#2s,
                dutMemory := Memory_DUT,
                iOut => iOut);
```

SmoothSignal_REAL Smooth REAL signals

Description This instructions uses a 1st order delay time **tT1** to smooth the REAL input value at **iIN**.



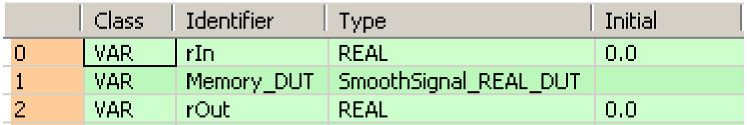
PLC types see page 1331

Data types

Input variable	Data type	Function
rIn	REAL	Input signal
tT1	TIME	Time constant of the 1st order low-pass filter
Input/output variable		
dutMemory	dutMemory	Instance-dependent data memory structure, which serves as the internal memory of the function. As with the instance name of a function block, it may be neither initialized nor written in the body!
Output variable		
rOut	REAL	Output signal

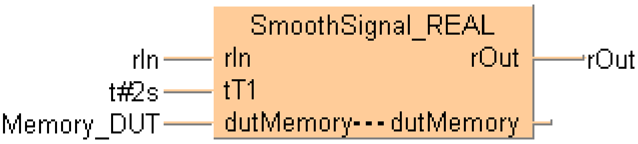
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	rIn	REAL	0.0
1	VAR	Memory_DUT	SmoothSignal_REAL_DUT	
2	VAR	rOut	REAL	0.0

LD



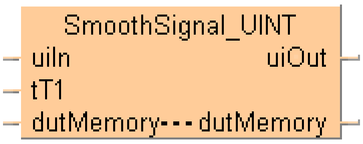
ST When programming with structured text, enter the following:

```
SmoothSignal_REAL(rIn := rIn,
     tT1 := t#2s,
           dutMemory := Memory_DUT,
           rOut => rOut);
```

SmoothSignal_UINT

Smooth UINT signals

Description This instructions uses a 1st order delay time **tT1** to smooth the unsigned INTEGER input value at **iIN**.



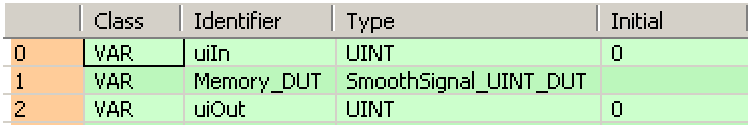
PLC types see page 1331

Data types

Input variable	Data type	Function
uiIn	UINT	Input signal
tT1	TIME	Time constant of the 1st order low-pass filter
Input/output variable		
dutMemory	dutMemory	Instance-dependent data memory structure, which serves as the internal memory of the function. As with the instance name of a function block, it may be neither initialized nor written in the body!
Output variable		
uiOut	UINT	Output signal

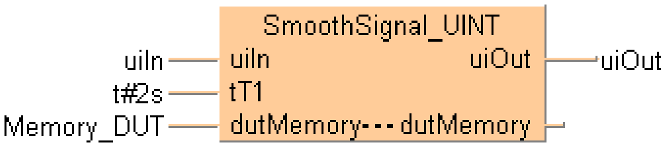
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	uiIn	UINT	0
1	VAR	Memory_DUT	SmoothSignal_UINT_DUT	
2	VAR	uiOut	UINT	0

LD



ST When programming with structured text, enter the following:

```
SmoothSignal_UINT(uiIn := uiIn,
     tT1 := t#2s,
           dutMemory := Memory_DUT,
           uiOut => uiOut);
```

Chapter 31

FP-e display instructions

F180_SCR

Screen display instruction

Description This instruction sets up the screen display in the normal mode (N) and switch mode (S) of the FP-e unit.

PLC types **Availability of** F180_SCR **(see page** 1335**)**

Data types

Variable	Data type	Function
s1	ANY16	Specifies "s1" registration screen
s2	ARRAY [0..2] OF INT, WORD	Specifies the head of the screen display control data (3 words).
s3	ANY16	Specifies the data displayed in the upper section.
s4		Specifies the data displayed in the lower section.

Operands

For	Relay				T/C		Register			Constant
s1	WX	WY	WR	-	SV	EV	DT	IX	IY	dec. or hex.
s2	WX	WY	WR	-	SV	EV	DT	IX	IY	
s3	WX	WY	WR	-	SV	EV	DT	-	-	
s4	-	WY	WR	-	SV	EV	DT	IX	IY	-

Error flags

No.	IEC address	Set	If
R9007 R9008	%MX0.900.7 %MX0.900.8	permanently for an instant	▪ the area specified using the Index modifier exceeds the limit. ▪ the value for "s1" or "s2" is outside of the range specified.

☞

- **Special register "DT9***" cannot be specified for the lower section display data "s4."**
- **This instruction cannot be used in an interrupt program.**

Detailed information, please refer to the online help:

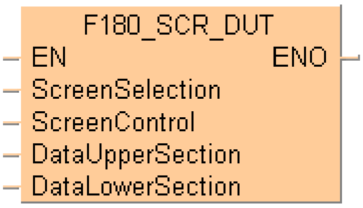
Examples of control register

ASCII code and its display

7-segment data and its display

F180_SCR_DUT Configuring the Display of the FP-e

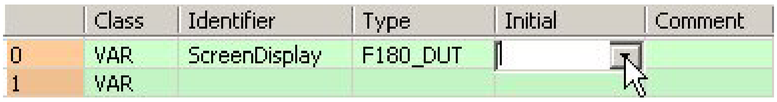
Description This instruction allows you to configure the screen display of the FP-e for N mode (normal mode) and S mode (switch mode).



Using a convenient dialog, the control code for the screen display is configured.

Procedure

1. **Assigning a DUT**
2. **Select F180_DUT in the header of the declaration under "Type"**
3. **Click ▼ in the “Initial” field**



	Class	Identifier	Type	Initial	Comment
0	VAR	ScreenDisplay	F180_DUT		
1	VAR				

The configuration dialog opens.

4. **Make desired settings**
5. **[OK]**

PLC types **Availability of F180_SCR_DUT (see page 1322)**

Data types

Variable	Data type	Function
ScreenSelection	ANY16	Display mode
ScreenControl	F180_DUT	Data unit type for the control data of the screen display.
DataUpperSection	ANY16	Value in the upper display area
DataLowerSection		Value in the lower display area

Operands

For	Relays				T/C		Registers			Constant
Screen Selection	WX	WY	WR	-	SV	EV	DT	IX	IY	dec. or hex.
Screen Control	-	-	-	-	-	-	-	-	-	-
DataUpper Section	WX	WY	WR	-	SV	EV	DT	IX	IY	-
DataLower Section	-	WY	WR	-	SV	EV	DT	IX	IY	-

Error flags

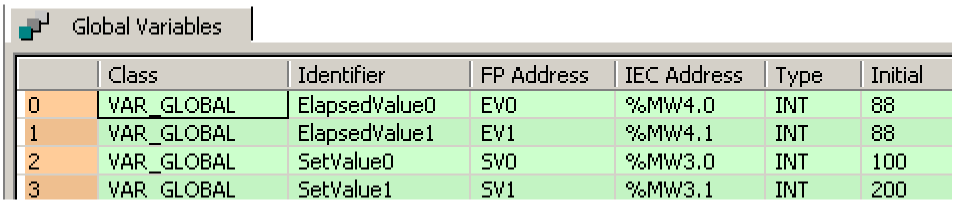
No.	IEC Address	Set	If
R9007	%MX0.900.7	permanently	▪ when the area defined by index modifiers is greater than the area allowed ▪ the value for **s1** or **s2** is invalid
R9008	%MX0.900.8	temporarily	

☞

- **You cannot enter the special data register "DT9***" for the lower display area.**
- **You cannot use this instruction in an interrupt program.**

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

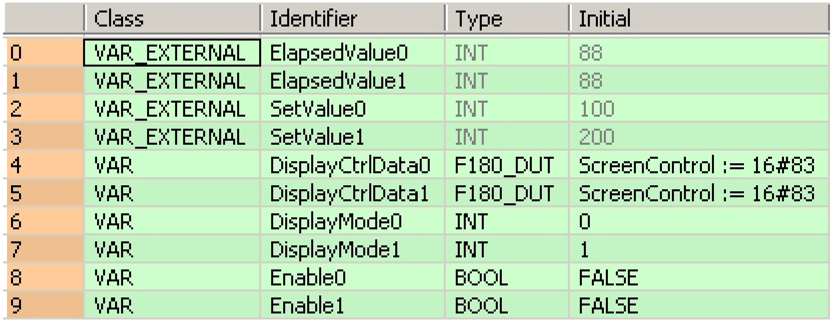
GVL In the global variable list, you define variables that can be accessed by all POUs in the project.



Global Variables

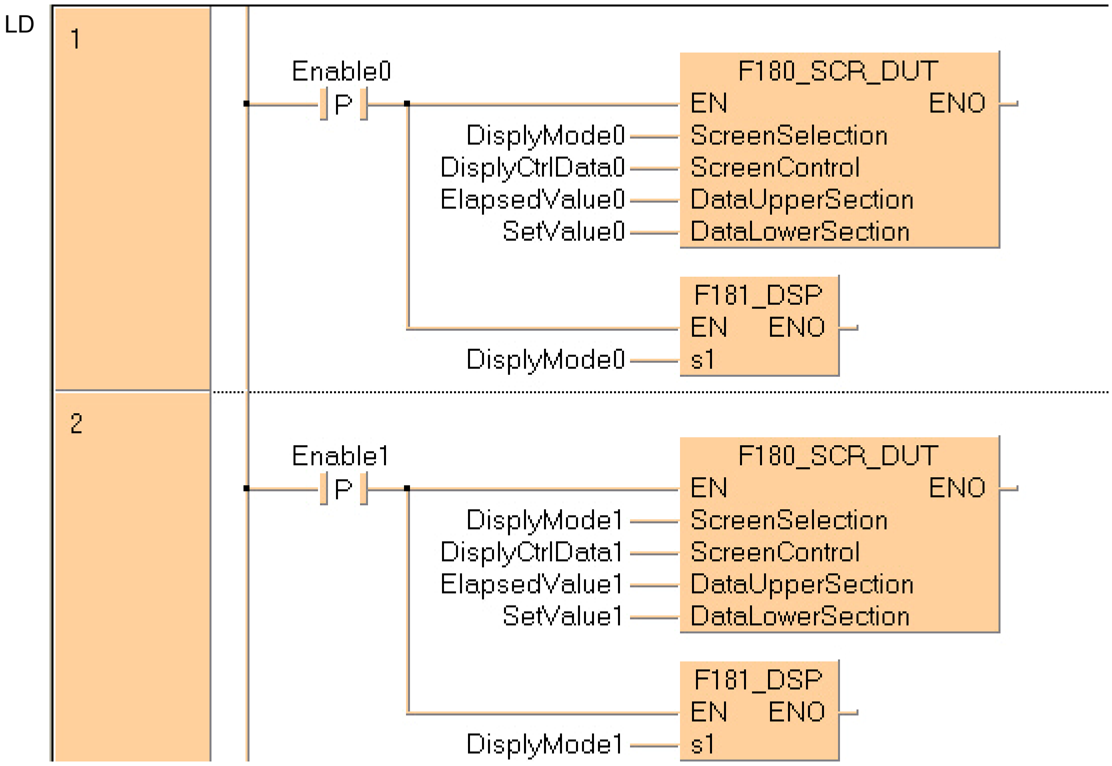
	Class	Identifier	FP Address	IEC Address	Type	Initial
0	VAR_GLOBAL	ElapsedValue0	EV0	%MW4.0	INT	88
1	VAR_GLOBAL	ElapsedValue1	EV1	%MW4.1	INT	88
2	VAR_GLOBAL	SetValue0	SV0	%MW3.0	INT	100
3	VAR_GLOBAL	SetValue1	SV1	%MW3.1	INT	200

POU header All input and output variables used for programming this function have been declared in the POU header.



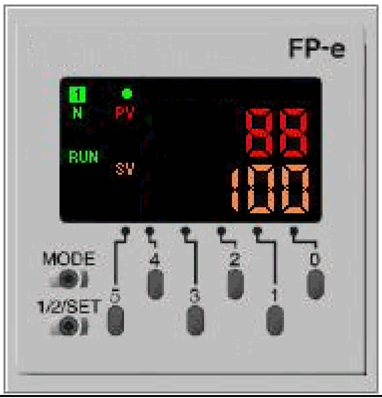
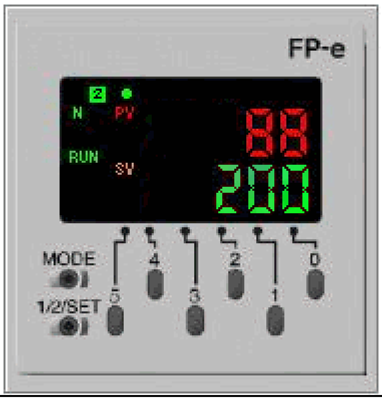
	Class	Identifier	Type	Initial
0	VAR_EXTERNAL	ElapsedValue0	INT	88
1	VAR_EXTERNAL	ElapsedValue1	INT	88
2	VAR_EXTERNAL	SetValue0	INT	100
3	VAR_EXTERNAL	SetValue1	INT	200
4	VAR	DisplayCtrlData0	F180_DUT	ScreenControl := 16#83
5	VAR	DisplayCtrlData1	F180_DUT	ScreenControl := 16#83
6	VAR	DisplayMode0	INT	0
7	VAR	DisplayMode1	INT	1
8	VAR	Enable0	BOOL	FALSE
9	VAR	Enable1	BOOL	FALSE

Body When the variable **Enable0** is set to TRUE, the function is executed and the FP-e is switched to N mode, 1st screen. **ProcessValue0** and **SetValue0** are displayed in the upper and lower sections in red and orange. When the variable **Enable1** is set to TRUE, the function is executed and the FP-e is switched to N mode, 2nd screen. **ProcessValue1** and **SetValue1** are displayed in the upper and lower sections in red and green. The monitor value icon is activated for both LD bodies. Use the instruction F181_DSP (see page 974) to change the display of the FP-e.


LD
1
Enable0
P
F180_SCR_DUT
EN
ENO
DisplyMode0 — ScreenSelection
DisplyCtrlData0 — ScreenControl
ElapsedValue0 — DataUpperSection
SetValue0 — DataLowerSection
F181_DSP
EN ENO
DisplyMode0 — s1
2
Enable1
P
F180_SCR_DUT
EN
ENO
DisplyMode1 — ScreenSelection
DisplyCtrlData1 — ScreenControl
ElapsedValue1 — DataUpperSection
SetValue1 — DataLowerSection
F181_DSP
EN ENO
DisplyMode1 — s1

DisplayMode0	**DisplayMode1**

DisplayControlData0
ScreenControl := 16#83,
UpperDisplayControl := 16#4000,
LowerDisplayControl := 16#2000
	DisplayControlData1
	ScreenControl := 16#83,
	UpperDisplayControl := 16#4000,
	LowerDisplayControl := 16#6000

 FP-e 1 N PV RUN SV 88 100 MODE 1/2/SET 5 4 3 2 1 0	 FP-e 2 N PV RUN SV 88 200 MODE 1/2/SET 5 4 3 2 1 0
For detailed information please refer to the technical manual of the FP-e (file ARCT1F369E.PDF on your installation CD of Control FPWIN Pro).	

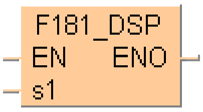
ST When programming with structured text, enter the following:

```
IF DF (Enable0) THEN
    F180_SCR_DUT(ScreenSelection:=DisplayMode0,
            ScreenControl:=DisplayCtrlData0,
            DataUpperSection:=ElapsedValue0,
            DataLowerSection:=SetValue0);
  F181_DSP (DisplayMode0);
END_IF;

IF DF (Enable1) THEN
    F180_SCR_DUT(ScreenSelection:=DisplayMode1,
            ScreenControl:=DisplayCtrlData1,
            DataUpperSection:=ElapsedValue1,
            DataLowerSection:=SetValue1);
  F181_DSP (DisplayMode1);
END_IF;
```

F181_DSP

Screen change instruction

Description The FP-e display mode is changed to the one specified using **s1**.



PLC types **Availability of** F181_DSP **(see page 1322)**

Data types

Variable	Data type	Function
s1	ANY16	Display mode and No. (0 to 7 can be specified).

Operands

For	Relay				T/C		Register			Constant
s1	WX	WY	WR	-	SV	EV	DT	IX	IY	dec. or hex.

Specifying "s1" registration display

Values for "s1"	Display type
0	N mode 1st screen
1	N mode 2nd screen
2	S mode 1 st screen
3	S mode 2nd screen
4	R mode 1st screen
5	R mode 2nd screen
6	I mode 1st screen
7	I mode 2nd screen

(N=normal mode, S=switch mode, R=register mode, I=I/O monitor mode).

Error flags

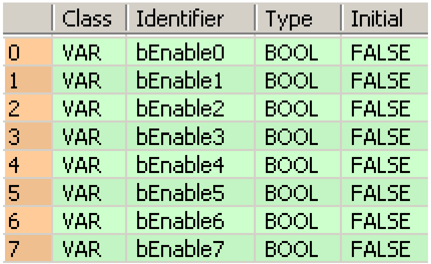
No.	IEC address	Set	If
R9007 R9008	%MX0.900.7 %MX0.900.8	permanently for an instant	▪ the area specified using the Index modifier exceeds the limit. ▪ the value "s1" is not "0" to "7".

- **If a value other than "0" to "7" is specified for "s1", an operation error will occur.**
- **This instruction cannot be used during an interrupt program.**

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header

All input and output variables used for programming this function have been declared in the POU header.

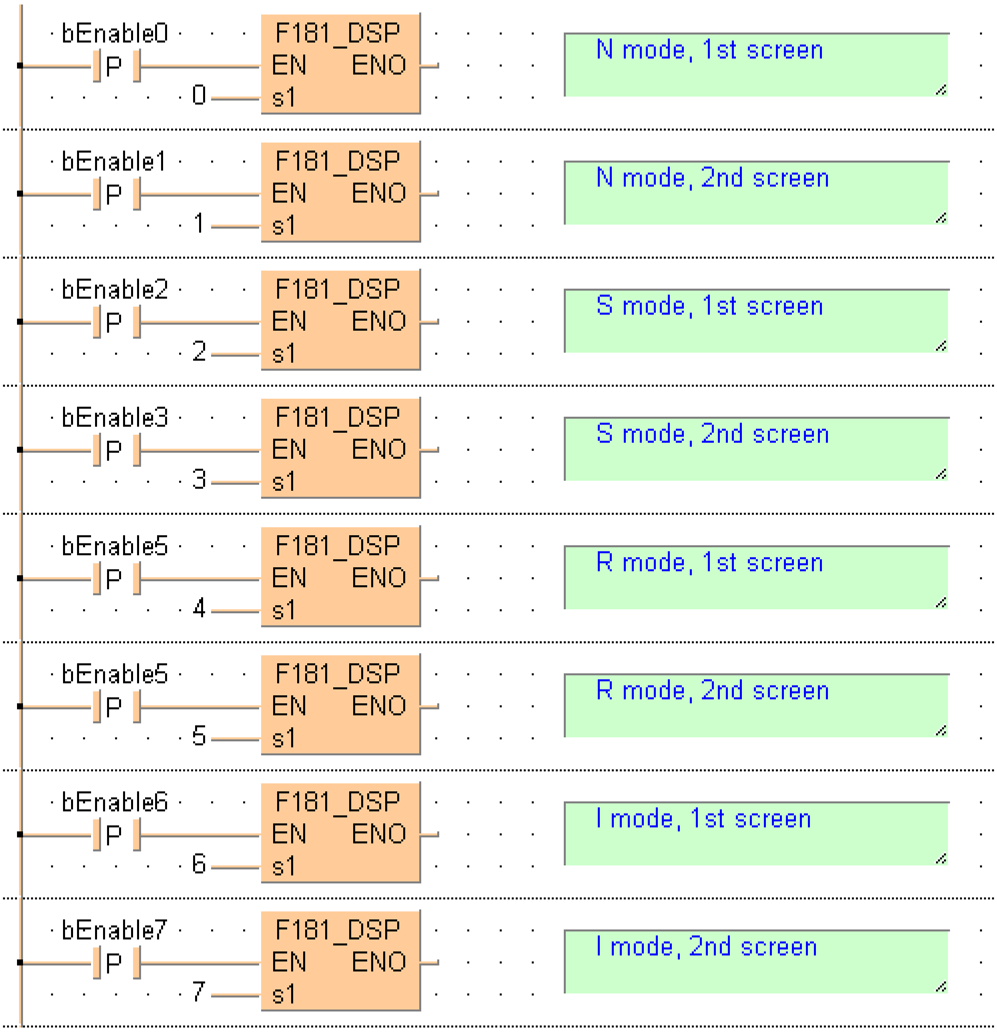


	Class	Identifier	Type	Initial
0	VAR	bEnable0	BOOL	FALSE
1	VAR	bEnable1	BOOL	FALSE
2	VAR	bEnable2	BOOL	FALSE
3	VAR	bEnable3	BOOL	FALSE
4	VAR	bEnable4	BOOL	FALSE
5	VAR	bEnable5	BOOL	FALSE
6	VAR	bEnable6	BOOL	FALSE
7	VAR	bEnable7	BOOL	FALSE

Body

According to the variable **Enable0** to **Enable7** that is set to TRUE, the function is executed and the FP-e is switched to the corresponding mode and the corresponding screen. (N=normal mode, S=switch mode, R=register mode, I=I/O monitor mode).

LD



ST When programming with structured text, enter the following:

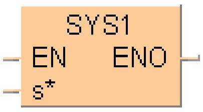
```
IF DF(bEnable0) THEN
    (* N mode, 1st screen *)
    F181_DSP(s1:=0);
END_IF;
IF DF(bEnable1) THEN
    (* N mode, 2nd screen *)
    F181_DSP(s1:=1);
END_IF;
IF DF(bEnable2) THEN
    (* S mode, 1st screen *)
    F181_DSP(s1:=2);
END_IF;
IF DF(bEnable3) THEN
    (* S mode, 2nd screen *)
    F181_DSP(s1:=3);
END_IF;
IF DF(bEnable4) THEN
    (* R mode, 1st screen *)
    F181_DSP(s1:=4);
END_IF;
IF DF(bEnable5) THEN
    (* R mode, 2nd screen *)
    F181_DSP(s1:=5);
END_IF;
IF DF(bEnable6) THEN
    (* I mode, 1st screen *)
    F181_DSP(s1:=6);
END_IF;
IF DF(bEnable7) THEN
    (* I mode, 2nd screen *)
    F181_DSP(s1:=7);
END_IF;
```

Chapter 32

System register instructions

SYS1 Change PLC system setting

Description

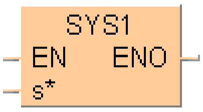


The description for SYS1 is divided into the following sections:

- Communication condition setting (see page 980)
- Password setting (see page 984)
- Interrupt setting (see page 986)
- PLC link time setting (see page 988)
- Change high-speed counter operation mode (see page 989)
- RS485 response time control (see page 991)

PLC types **Availability of** SYS1 **(see page** 1331**)**

Communication condition setting for the COM ports of the CPU



This changes the communication conditions for the COM port or Tool port based on the contents specified by the character constant.

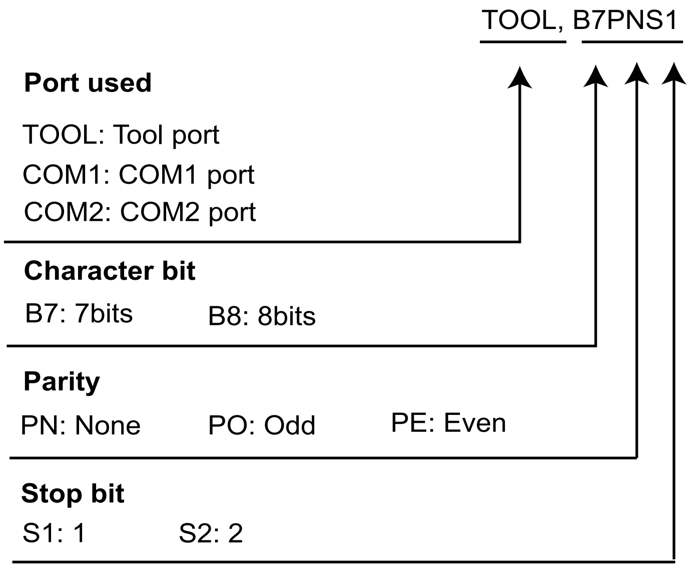
The communication conditions for the port specified by the first keyword are changed to the contents specified by the second keyword. The first and second keywords are separated by a comma.

Contents that can be changed include the following:

1. Communication format
2. Baud rate
3. Unit No.
4. Header and Terminator
5. RS (Request to Send) control

Keyword setting

1. Communication format (Shared by the Tool, COM 1 and COM 2 ports)



TOOL, B7PNS1

Port used

TOOL: Tool port
COM1: COM1 port
COM2: COM2 port

Character bit

B7: 7bits B8: 8bits

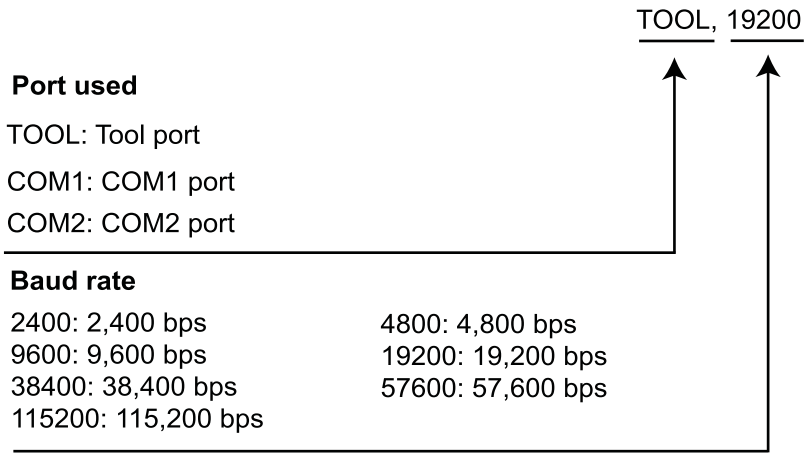
Parity

PN: None PO: Odd PE: Even

Stop bit

S1: 1 S2: 2

2. Baud rate (Shared by the Tool, COM 1 and COM 2 ports)



TOOL, 19200

Port used

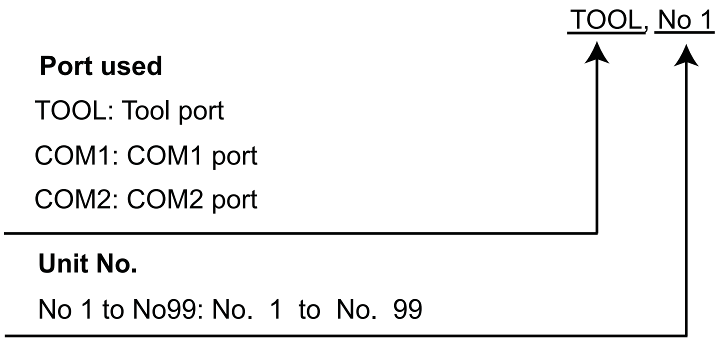
TOOL: Tool port
COM1: COM1 port
COM2: COM2 port

Baud rate

2400: 2,400 bps	4800: 4,800 bps
9600: 9,600 bps	19200: 19,200 bps
38400: 38,400 bps	57600: 57,600 bps
115200: 115,200 bps	

Lower baud rates of 300, 600, and 1200bit/s can be specified for FP-X V2.0 or later and FPΣ V3.1 or later. These baud rates cannot be set in the system registers.

3. Unit No. (Shared by the Tool, COM 1 and COM 2 ports)



TOOL, No 1

Port used

TOOL: Tool port
COM1: COM1 port
COM2: COM2 port

Unit No.

No 1 to No99: No. 1 to No. 99

With the FP0R, use the keywords 'COM1No' and 'TOOLNo' to read the unit number from a data register (DT0–DT9999) containing the unit number 1–99. The data register has to be specified with exactly five characters: For example, D0815 indicates DT815. Leading zeros must be entered. The keyword is case sensitive, hence COM1NO, Com1No or … d0815 would be invalid.

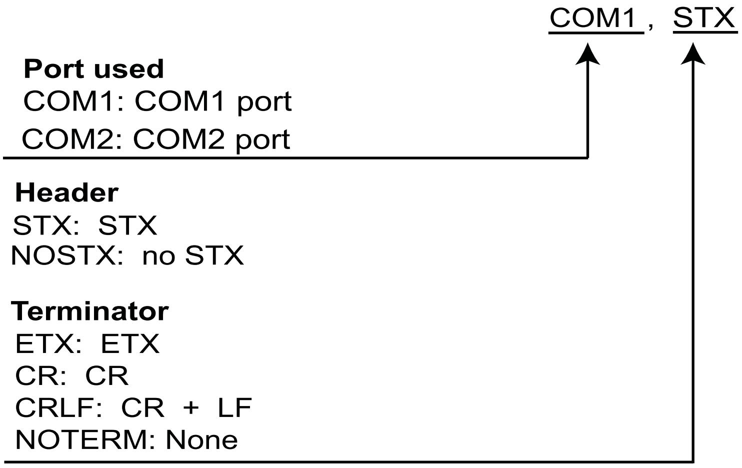
Example:

SYS1 'COM1No,D9999' indicates DT9999

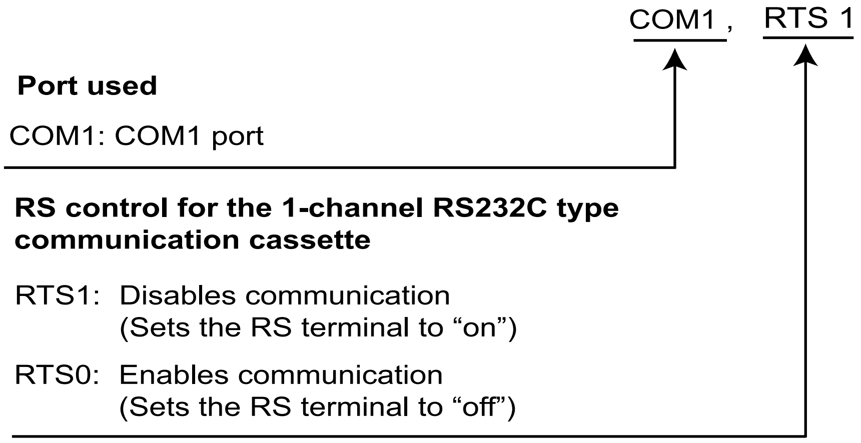
SYS1 'COM1No,D0000' indicates DT0

A calculation error occurs if any value except 1–99 is assigned to the DT memory.

4. Header and Terminator (Shared by the COM 1 and COM 2 ports)



5. RS (Request to Send) control (COM 1 port only)



COM1 , RTS 1

Port used

COM1: COM1 port

RS control for the 1-channel RS232C type communication cassette

RTS1: Disables communication
(Sets the RS terminal to “on”)

RTS0: Enables communication
(Sets the RS terminal to “off”)

Precautions during programming

- Executing this instruction does not rewrite the contents of the system ROM in the control unit. As a result, turning the power supply off and then on again rewrites the contents of the system registers specified by the tool software.
- We recommend using differential execution with this instruction.
- Because the system register settings are changed, a verification error may occur in some cases if verification is carried out with the tools.
- Separate first and second keywords with a comma "," and do not use spaces.

Error flags

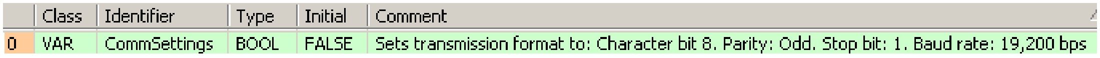
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ any character other than a keyword is specified ▪ no comma is between the first and second keywords ▪ small letters of the alphabet are used to specify the keyword (except for numbers used to specify unit no.) ▪ no communication cassette has been installed when COM1 or COM2 has been set
R9008	%MX0.900.8	for an instant	▪ the setting of the unit no. setting switch is anything other than 0 when COM1 or COM2 has been set and the unit no. is being changed ▪ the unit no. set using this instruction is anything other than a value between 1 and 99 ▪ the baud rate or transmission format for COM1 has been changed when the PLC link mode is specified for COM1 ▪ the baud rate or transmission format is changed while the Tool port, COM port 1, or COM port 2 is being initialized using MODEM ▪ the communication mode is set to anything other than the general communication mode when header and terminator have been set ▪ any communication cassette other than the 1-channel RS232C type communication cassette is installed when using RS control ▪ the specified unit no. is larger than the largest unit no. specified by the system register when the COM 1 port is in the PLC link mode

Example

In this example the function SYS1 is programmed in ladder diagram (LD).

POU header

The same POU header is used for all programming languages.

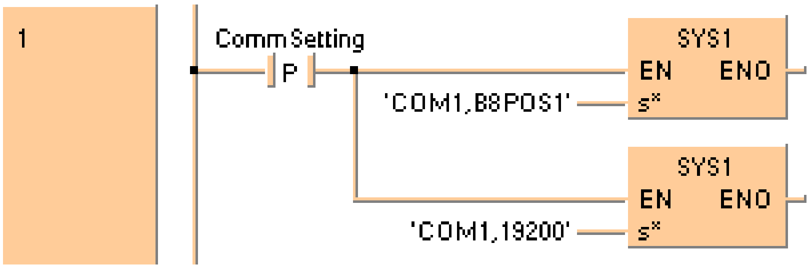


	Class	Identifier	Type	Initial	Comment
0	VAR	CommSettings	BOOL	FALSE	Sets transmission format to: Character bit 8. Parity: Odd. Stop bit: 1. Baud rate: 19,200 bps

Body

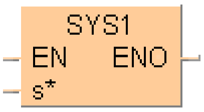
When **CommSettings** turns on, the transmission format and baud rate for the COM1 port are set as follows: Character bit: 8, Parity: Odd; Stop bit: 1; Baud rate: 19,200 bps.

LD



☞ **The values entered at s* will be right aligned automatically by the compiler.**

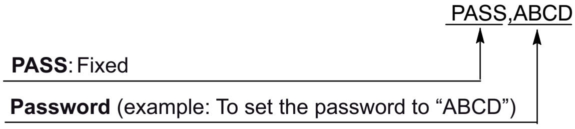
Password Setting



This changes the password specified by the controller, based on the contents specified by the character constant.

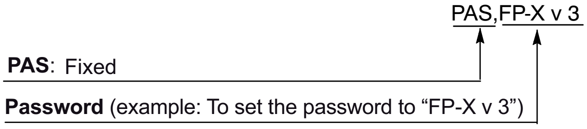
This changes the password specified by the controller to the contents specified by the second keyword. The first and second keywords are separated by a comma.

Keyword setting for 4-digit hexadecimal password



Keyword setting for 8-digit alphanumeric password

Enter for example 'PAS,FP-X␣v␣3'. Spaces at the end of the password are not significant.



Precautions during programming

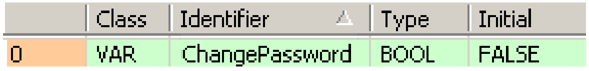
- When this instruction is executed, writing to the internal F-ROM takes approximately 100ms.
- If the specified password is the same as the password that has already been written, the password is not written to the F-ROM.
- We recommend using differential execution with this instruction.
- Separate first and second keywords with a comma "," and do not use spaces.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ any character other than a keyword is specified ▪ no comma is between the first and second keywords ▪ small letters of the alphabet are used to specify the keyword ▪ the data specified for the password setting is any character other than 0 to 9 or A to F, or the specified data consists of other than four digits.
R9008	%MX0.900.8	for an instant	

Example In this example the function SYS1 is programmed in ladder diagram (LD).

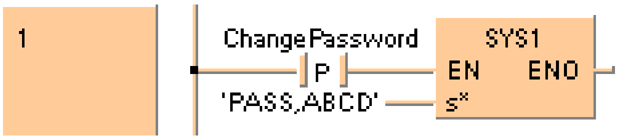
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial
0	VAR	ChangePassword	BOOL	FALSE

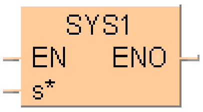
Body When **ChangePassword** turns on, the controller password is changed to "ABCD".

LD



☞ **The values entered at s* will be right aligned automatically by the compiler.**

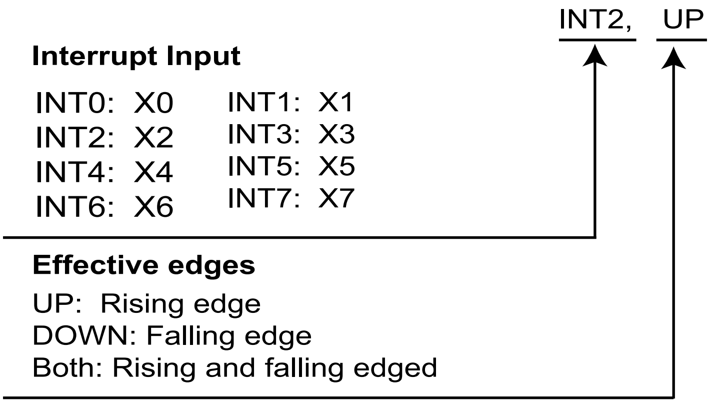
Interrupt Setting



This sets the interrupt input based on the contents specified by the character constant.

This sets the input specified by the first keyword as the interrupt input, and changes the input conditions to the contents specified by the second keyword. The first and second keywords are separated by a comma.

Keyword setting



INT2, UP

Interrupt Input

INT0: X0 INT1: X1
INT2: X2 INT3: X3
INT4: X4 INT5: X5
INT6: X6 INT7: X7

Effective edges

UP: Rising edge
DOWN: Falling edge
Both: Rising and falling edged

For the FP-X you can set INT0 to INT13.

Precautions during programming

- Executing this instruction does not rewrite the contents of the system ROM in the control unit. As a result, turning the power supply off and then on again rewrites the contents of the system registers specified by the tool software.
- We recommend using differential execution with this instruction.
- When UP or DOWN has been specified, the contents of the system registers change in accordance with the specification, so a verification error may occur in some cases, when the program is verified. When BOTH has been specified, the contents of the system registers do not change.
- Separate first and second keywords with a comma "," and do not use spaces.

Error flags

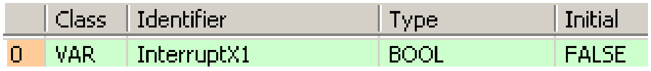
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ any character other than a keyword is specified ▪ no comma is between the first and second keywords ▪ small letters of the alphabet are used to specify the keyword
R9008	%MX0.900.8	for an instant	

Example

In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header

All input and output variables used for programming this function have been declared in the POU header.

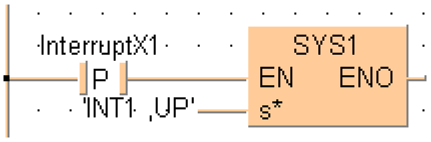


	Class	Identifier	Type	Initial
0	VAR	InterruptX1	BOOL	FALSE

Body

When **InterruptX1** turns on, the input condition of interrupt input X1 is changed to "Rising Edge".

LD



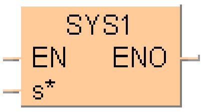
ST

When programming with structured text, enter the following:

```
if (DF(InterruptX1)) then
      SYS1('INT1, UP');
end_if;
```

☞ **The values entered at s* will be right aligned automatically by the compiler.**

PLC Link Time Setting



This sets the system setting time when a PLC link is used, based on the contents specified by the character constant.

The conditions specified by the first keyword are set as the time specified by the second keyword. The first and second keywords are separated by a comma.

The setting for the link entry waiting time is set if the transmission cycle time is shortened when there are stations that have not joined the link. (Stations that have not joined the link: Stations that have not been connected between the first station and the station with the largest number, or stations for which the power supply has not been turned on.)

The error detection time setting for the transmission assurance relay is set if the time between the power supply being turned off at one station and the transmission assurance relay being turned off at a different station is to be shortened.

Keyword setting

1. Link entry wait time

PCLK1T0, 100

PCLK1T0: Fixed

Specified range: 10 to 400 (10 ms to 400 ms)

2. Error detection time for transmission assurance relay

PCLK1T1, 100

PCLK1T1: Fixed

Specified range: 100 to 6400 (100 ms to 6400 ms)

Precautions during programming

- The program should be placed at the beginning of all PLCs being linked, and the same values specified.
- This instruction should be specified in order to set special internal relay R9014 as the differential execution condition.
- The setting contents of the system registers are not affected by this instruction being executed.
- Separate first and second keywords with a comma "," and do not use spaces.

Precautions when setting the link entry wait time

- This should be specified such that the value is at least twice that of the largest scan time of all the PLCs that are linked.
- If a short value has been specified, there may be some PLCs that are not able to join the link even though the power supply for that PLC has been turned on.
- If there are any stations that have not joined the link, the setting should not be changed, even if the link transmission cycle time is longer as a result. (The

default value is 400 ms.)

Precautions when setting the error detection time for the transmission assurance relay

- This should be specified such that the value is at least twice that of the largest transmission cycle time of all the PLCs that are linked.
- If a short value has been specified, there is a possibility that the transmission assurance relay will malfunction.
- The setting should not be changed, even if the detection time for the transmission assurance relay is longer than the result. (The default value is 6400ms.)

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ any character other than a keyword is specified ▪ no comma is between the first and second keywords ▪ small letters of the alphabet are used to specify the keyword ▪ the specified value is outside the specified range
R9008	%MX0.900.8	for an instant	

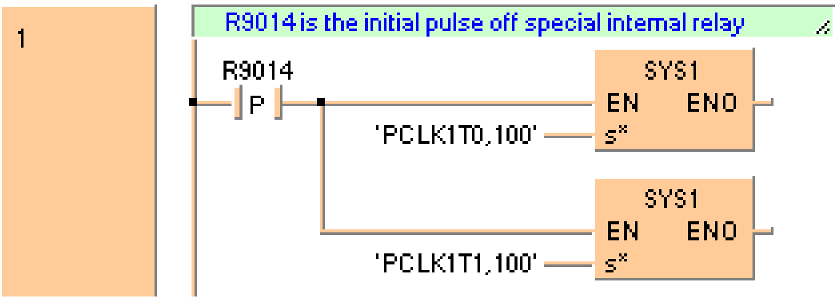
Example Below is an example of a ladder diagram (LD) body for the instruction. Because FP addresses and strings are entered directly instead of using variables, no POU header is required.

Body When R9014 turns on when a PLC link is being used, the link entry wait time and the error detection times for transmission assurance relay are set as follows:

Link entry wait time: 100ms

Error detection time for transmission assurance relay: 100ms.

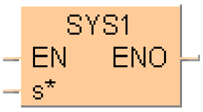
LD



R9014 is the initial pulse off special internal relay

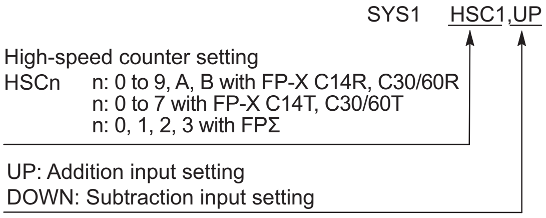
☞ **The values entered at s* will be right aligned automatically by the compiler.**

Change high-speed counter operation mode



This changes the operation mode of the high-speed counter based on the contents specified by the character constant.

Keyword setting


SYS1 HSC1,UP
High-speed counter setting
HSCn n: 0 to 9, A, B with FP-X C14R, C30/60R
n: 0 to 7 with FP-X C14T, C30/60T
n: 0, 1, 2, 3 with FPΣ
UP: Addition input setting
DOWN: Subtraction input setting

Precautions during programming

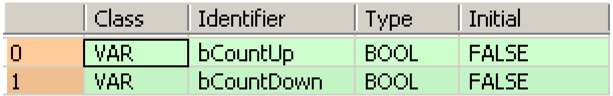
- If the corresponding HSC system register is set to Unused, an operation error occurs. Set the system register to Incremental input or Decremental input in advance.
- Executing this instruction does not rewrite the contents of the system ROM in the control unit. As a result, turning the power supply off and then on again rewrites the contents of the system registers specified by the software tool.
- We recommend to execute this instruction only once, e.g. in dependency of a rising or falling edge of an execution condition.
- When UP or DOWN has been specified, the contents of the system registers change in accordance with the specification, so a verification error may occur in some cases when checking or compiling the program. When BOTH have been specified, the contents of the system registers do not change. Separate the first and the second keyword with a comma "," e.g. **HSCB,UP**; do not use spaces. Otherwise an operation error will occur.

Error flags

No.	IEC address	Set	If
R9007 R9008	%MX0.900.7 %MX0.900.8	permanently for an instant	▪ any character other than a keyword is specified ▪ no comma is between the first and second keyword ▪ the letters used to specify the keyword are not capitalized ▪ the HSC system register is set to items other than the addition input or subtraction input

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

POU header

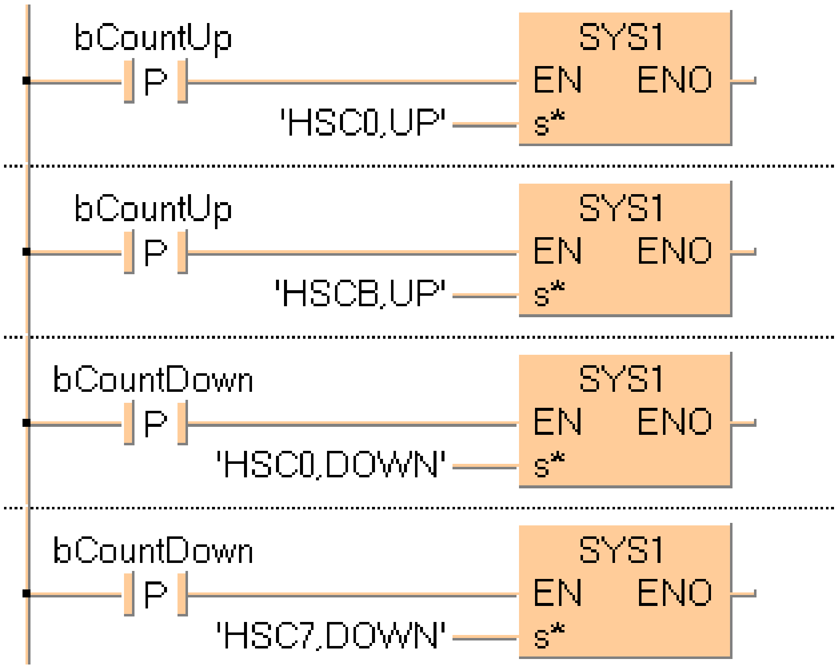


	Class	Identifier	Type	Initial
0	VAR	bCountUp	BOOL	FALSE
1	VAR	bCountDown	BOOL	FALSE

Body

When **bCountUp** is set to TRUE, the function is carried out. The system register for the specified channel is set to count up. When **bCountDown** is set to TRUE, the specified channel is set to count down.

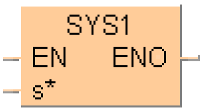
LD



ST When programming with structured text, enter the following:

```
          IF DF(bCountUp) Then
      SYS1('HSC0,UP');
      SYS1('HSCB,UP');
End_If;

IF DF(bCountDown) Then
      SYS1('HSC0,DOWN');
      SYS1('HSC7,DOWN');
          End_If;
```

RS485 Response Time Control



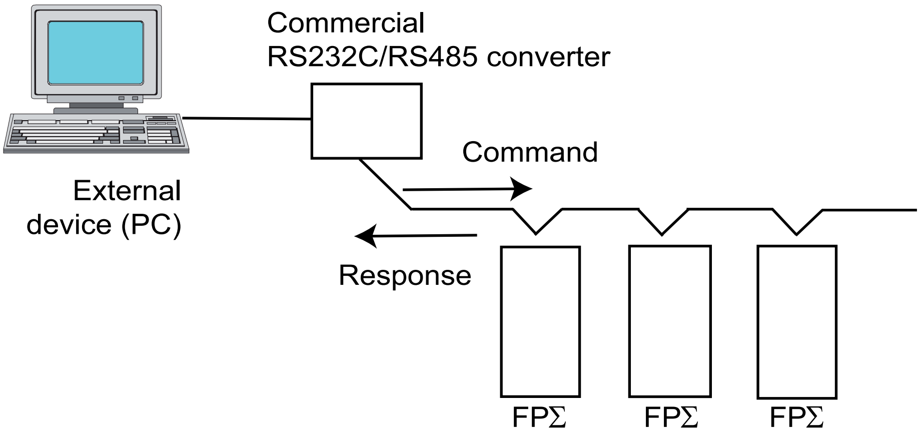
This changes the communication conditions based on the RS485 of the COM port or Tool port, in response to the contents specified by the character constant.

The port response time specified by the first keyword is delayed based on the contents specified by the second keyword. This instruction is used to delay the response time on the PLC side until the state is reached in which commands can be sent by an external device and responses can be received from the PLC.

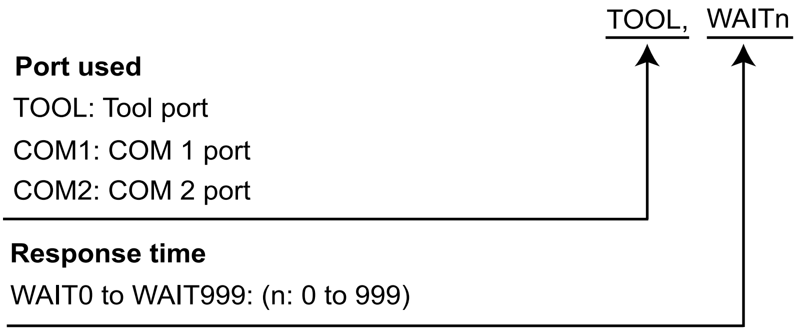
The first and second keywords are separated by a comma.

Usage Example

When a commercial RS232C/RS485 converter is being used to carry out communication between a personal computer and the FP-Σ, this instruction is used to return the PLC response after switching of the enable signal has been completed on the converter side.



Keyword setting



TOOL, WAITn

Port used

TOOL: Tool port

COM1: COM 1 port

COM2: COM 2 port

Response time

WAIT0 to WAIT999: (n: 0 to 999)

If the communication mode has been set to the computer link mode, the set time is the scan time x n (n: 0 to 999).

If the communication mode has been set to the PLC link mode, the set time is n µs (n: 0 to 999).

If n = 0, the delay time set by this instruction will be set to "None".

Precautions during programming

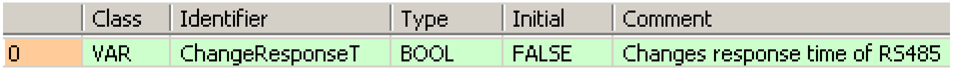
- This instruction is valid only if the setting on the controller side has been set to the computer link mode or the PLC link mode. It is invalid in the general communication mode.
- Executing this instruction does not change the settings in the system registers.
- We recommend using differential execution with this instruction.
- When the power supply to the PLC is off, the settings set by this instruction are cleared. (The set value will become 0.) If the mode is switched to the PROG. mode after the instruction has been executed, however, the settings will be retained.
- If a commercial RS232C/RS485 converter is being used in the PLC link mode, this instruction should be programmed in all of the stations (PLCs) connected to the link.
- Separate first and second keywords with a comma "," and do not use spaces.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ any character other than a keyword is specified ▪ no comma is between the first and second keywords ▪ small letters of the alphabet are used to specify the keyword ▪ no communication cassette has been installed when COM1 or COM2 has been set
R9008	%MX0.900.8	for an instant	

Example In this example the function SYS1 is programmed in ladder diagram (LD).

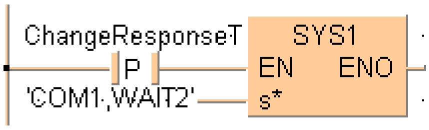
POU header The same POU header is used for all programming languages.



	Class	Identifier	Type	Initial	Comment
0	VAR	ChangeResponseT	BOOL	FALSE	Changes response time of RS485

Body When **ChangeResponseT** turns on, the response time for COM port 1 is delayed by 2µs.

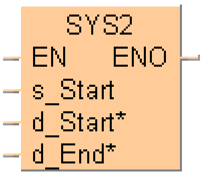
LD



☞ **The values entered at s* will be right aligned automatically by the compiler.**

SYS2 Change System Register Settings for PC Link Area

Description While the PLC is in RUN mode, SYS2 changes the settings for the specified system registers. **s_Start** contains the new values for those system registers defined between **d_Start*** and **d_End***.



You can change the values in system registers 40 - 47 (with the FP0R, FP-Σ 32k, FP-X also 50 - 57), PC link area.

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

Precautions during programming

- Executing this instruction does not rewrite the contents of the system ROM in the control unit. As a result, turning the power supply off and then on again rewrites the contents of the system registers specified by the tool software.
- A value between 40 and 47 should be specified for **d_Start*** or **d_End***. Also, the values should always be specified in such a way that **d_Start*** ≤ **d_End***.
- The values of the system registers change, so a verification error may occur when the program is verified.

PLC types **Availability of** SYS2 **(see page 1331)**

Data types

Variable	Data type	Function
s_Start	ANY16	Contains new values for the system registers defined by remaining two variables.
d_Start*		First system register (between 40-47) to receive new value. Must be a constant
d_End*		Last system register (between 40-47) to receive new value. Must be a constant

Operands

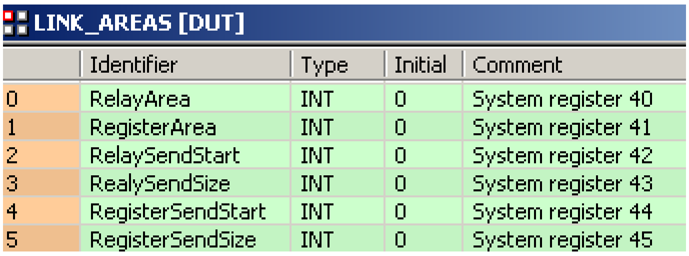
For	Relay				T/C		Register			Constant
s_Start	-	-	-	-	-	-	DT	-	-	-
d_Start*	-	-	-	-	-	-	-	-	-	dec. or hex.
d_End*	-	-	-	-	-	-	-	-	-	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ d1 > d2 ▪ the specified value is outside the ranges specified for the various system registers setting values
R9008	%MX0.900.8	for an instant	

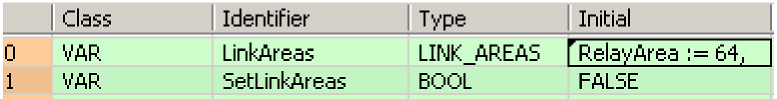
Example In this example the function SYS2 is programmed in ladder diagram (LD).

DUT A Data Unit Type (DUT) can be composed of several data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.


LINK_AREAS [DUT]

	Identifier	Type	Initial	Comment
0	RelayArea	INT	0	System register 40
1	RegisterArea	INT	0	System register 41
2	RelaySendStart	INT	0	System register 42
3	RealySendSize	INT	0	System register 43
4	RegisterSendStart	INT	0	System register 44
5	RegisterSendSize	INT	0	System register 45

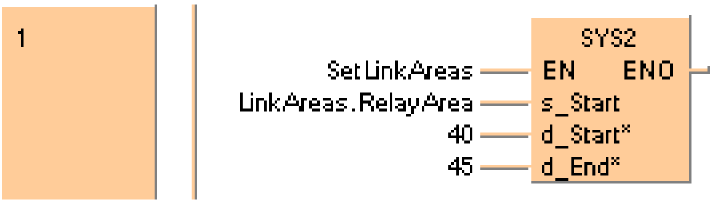
POU header The same POU header is used for all programming languages.



	Class	Identifier	Type	Initial
0	VAR	LinkAreas	LINK_AREAS	RelayArea := 64,
1	VAR	SetLinkAreas	BOOL	FALSE

Body Changes the values for the PC link area system registers 40 through 45 as defined in **LinkAreas** when **SetLinkAreas** turns on.

LD



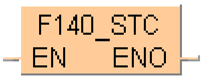
Chapter 33

Special instructions

F140_STC

Carry-flag set

Description Special internal relay R9009 (carry-flag) goes ON if the trigger **EN** is in the ON-state. This instruction can be used to control data using carry-flag R9009 (e.g. **F122_RCR** (see page 589) and **F123_RCL** (see page 591) instructions).

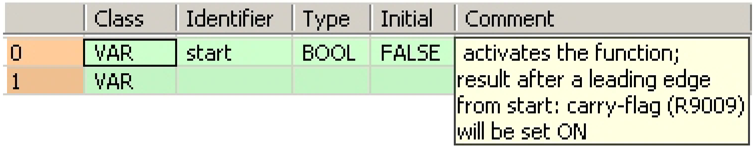


This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F140_STC **(see page 1321)**

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

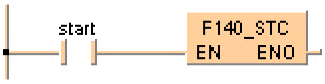
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function; result after a leading edge from start: carry-flag (R9009) will be set ON
1	VAR				

Body When the variable **start** is set to TRUE, the function is carried out.

LD



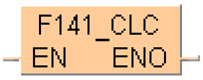
ST When programming with structured text, enter the following:

```
IF start THEN
    F140_STC();
END_IF;
```

F141_CLC

Carry-flag reset

Description Special internal relay R9009 (carry-flag) goes OFF if the trigger **EN** is in the ON-state. This instruction can be used to control data using carry-flag R9009 (e.g. F122_RCR (see page 589) and F123_RCL (see page 591) instructions).

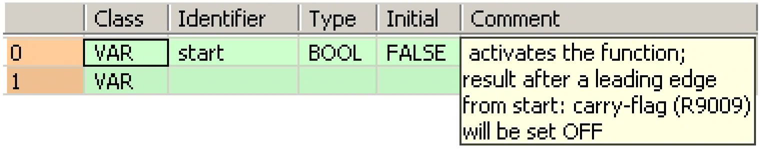


This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F141_CLC **(see page 1321)**

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

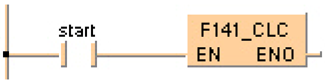
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function;
1	VAR				result after a leading edge from start: carry-flag (R9009) will be set OFF

Body When the variable **start** is set to TRUE, the function is carried out.

LD

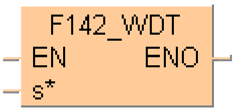


ST When programming with structured text, enter the following:

```
IF start THEN
    F141_CLC();
END_IF;
```

F142_WDT Watchdog timer update

Description The scan ‘check watchdog timer’ is preset with the constant specified by **s*** if the trigger **EN** is in the ON-state. The value specified by **s*** is 1 to 255 and the preset time becomes 2.5 ms * **s*** (637.5 ms).



The scan ‘check watchdog timer’ is automatically set at the start of a scan with the value of the system register (No. 30). To monitor the transit of a processing block, set the watchdog timer with this instruction immediately before transition and set again immediately after that.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

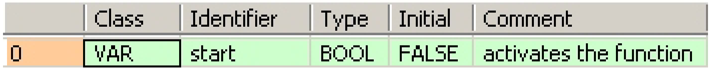
PLC types **Availability of** F142_WDT **(see page 1321)**

Data types

Variable	Data type	Function
s	INT	specifies watchdog timer value

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

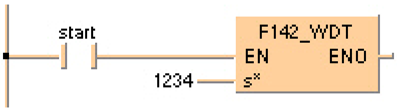
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function

Body When the variable **start** is set to TRUE, the function is carried out.

LD



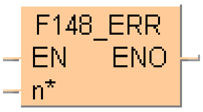
ST When programming with structured text, enter the following:

```
IF start THEN
    (* Watchdog timer value is changed to 123.4ms *)
    F142_WDT(1234);
END_IF;
```

F148_ERR

Self-diagnostic error set/reset

Description The error no. specified by **n*** is copied into the system variable sys_iSelfDiagnosticErrorCode that reads the corresponding special data register. Setting **n***=0, all error numbers greater than 43 are cleared and the error LED turns off.



At the same time, the self-diagnostic error-flag R9000 is set and ERROR LED on the CPU is turned ON.

The contents of the error flag R9000 and the error no. can be read and checked using Control FPWIN Pro (**Monitor** → **Display special relays and registers** → **Basic error messages**) or the corresponding system variables.

Error number areas:

When **n*** = 100 to 199, the operation is halted.

When **n*** = 200 to 299, the operation is continued.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F148_ERR **(see page 1321)**

Data types

Variable	Data type	Function
n*	ANY16	▪ Must be a constant ▪ self-diagnostic error code number, range: 0 and 100 to 299 ▪ See also: PLC status in the online help

Operands

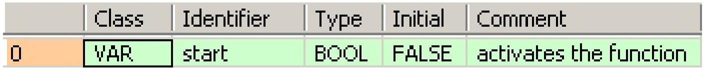
For	Relay				T/C		Register			Constant
n*	-	-	-	-	-	-	-	-	-	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ **n** exceeds the limit.
R9008	%MX0.900.8	permanently	

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

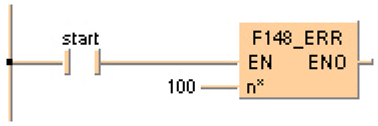
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function

Body When the variable **start** is set to TRUE, the function is carried out.

LD



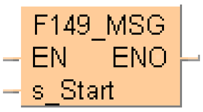
ST When programming with structured text, enter the following:

```
IF start THEN
    (* Sets the self-diagnostic error 100 *)
    (* The ERROR/ALARM LED of the PLC is on,
    and operation stops. *)
    F148_ERR(100);
END_IF;
```

F149_MSG

Message display

Description This instruction is used for displaying the message on the FP Programmer II screen. After executing the **F149_MSG** instruction, you can see the message specified by **s** on the FP Programmer II screen.



When the **F149_MSG** instruction is executed, the message-flag R9026 is set and the message specified by **s** is set in special data registers DT9030 to DT9035 (DT90030 to DT90035 for FP0 T32CP, FP2/2SH, FP10/10S/10SH). Once the message is set in special data registers, the message cannot be changed even if the **F149_MSG** instruction is executed again. You can clear the message with the FP Programmer II.

This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

PLC types **Availability of** F149_MSG **(see page 1321)**

Data types

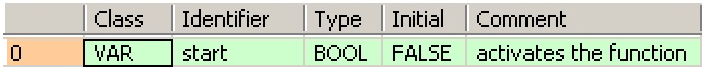
Variable	Data type	Function
s	STRING(12)	message to be displayed

Operands

For	Relay				T/C		Register			Constant
s	-	-	-	-	-	-	-	-	-	character

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

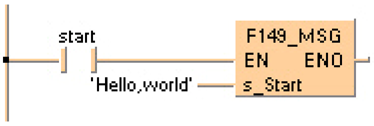
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function

Body When the variable **start** is set to TRUE, the function is carried out.

LD



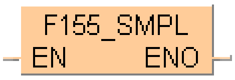
ST When programming with structured text, enter the following:

```
IF start THEN
   F149_MSG('Hello, world');
END_IF;
```

F155_SMPL Transfer sampling data

Description This instruction transfers the sampling data specified by the sampling trace editor into the sampling memory.

F155_SMPL can only be used with the sampling mode "per Scan".



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Specify the sampling points using Control FPWIN Pro:

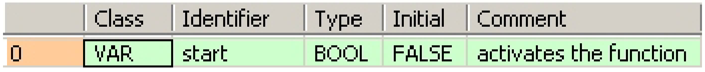
Relay contacts: 16 points
Available for (FP format): X, Y, R, L, T, C

Data: 3 words
Available for (FP format): WX, WY, WR, WL, SV, EV, DT, LD, FL

PLC types **Availability of F155_SMPL (see page 1321)**

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

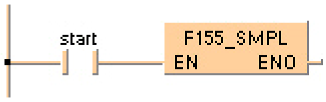
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function

Body When the variable **start** is set to TRUE, the function is carried out.

LD



ST When programming with structured text, enter the following:

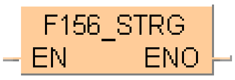
```
IF start THEN
    F155_SMPL();
END_IF;
```

F156_STRG

Set sampling trigger

Description This instruction sets the sampling trigger that stopps the sampling after the delay specified by the sampling trace parameters.

F156_STRG can be used with both sampling modes, "per Scan" and "per Time Interval".



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

Specify the sampling points using Control FPWIN Pro:

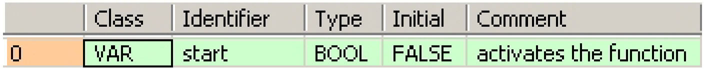
Relay contacts: 16 points
Available for (FP format): X, Y, R, L, T, C

Data: 3 words
Available for (FP format): WX, WY, WR, WL, SV, EV, DT, LD, FL

PLC types **Availability of** F156_STRG **(see page** 1321**)**

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

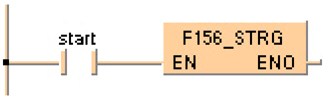
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function

Body When the variable **start** is set to TRUE, the function is carried out.

LD



ST When programming with structured text, enter the following:

```
IF start THEN
    F156_STRG();
END_IF;
```

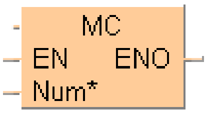
Chapter 34

Program execution control instructions

MC

Master control relay

Description Executes the program between the master control relay **MC** and master control relay end **MCE** (see page 1008) instructions of the same number **Num*** only if the trigger **EN** is in the ON-state



When the predetermined trigger **EN** is in the OFF state, the program between the master control relay **MC** and master control relay end **MCE** instructions is not executed.

A master control instruction (**MC** and **MCE**) pair may also be programmed in between another pair of master control instructions. This construction is called "nesting".

The constant number **Num*** that must correspond to **MC** number, both of which delimit a "nested" program that is not executed.

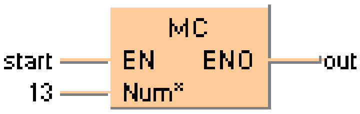
- **It is not possible to use this function in a function block POU.**
- **The maximum possible value that can be assigned to Num* depends on the PLC type.**

PLC types **Availability of** MC **(see page 1328)**

Data types

Variable	Data type	Function
Num*	constant	Constant number that must correspond to MCE number, both of which delimit a "nested" program that is not executed

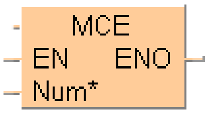
Example LD



MCE

Master control relay end

Description Executes the program between the master control relay MC (see page 1007) and master control relay end **MCE** instructions of the same number **Num*** only if the trigger **EN** is in the ON-state.



When the predetermined trigger **EN** is in the OFF state, the program between the master control relay **MC** and master control relay end **MCE** instructions is not executed.

A master control instruction (**MC** and **MCE**) pair may also be programmed in between another pair of master control instructions. This construction is called "nesting".

The constant number **Num*** that must correspond to **MC** number, both of which delimit a "nested" program that is not executed.

☞

- **It is not possible to use this function in a function block POU.**
- **The maximum possible value that can be assigned to Num* depends on the PLC type.**

PLC types **Availability of** MCE **(see page 1328)**

Data types

Variable	Data type	Function
Num*	constant	Constant number that must correspond to **MC** number, both of which delimit a "nested" program that is not executed

Example In this example, the progamming language Instruction List (IL) is used.

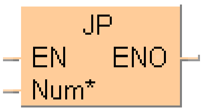
IL

```
LD      start   (* EN = start; Starting signal for the MC/MCE function.
                *)
MC      1       (* 1 = Num* *)
                (* ... *)
                (* Execute or execute not this program part. *)
                (* ... *)
MCE     1       (* 1 = Num* *)
```

JP

Jump to label

Description The **JP** (Jump to Label) instruction skips to the Label (LBL (see page 1013)) function that has the same number **Num*** as the **JP** function when the predetermined trigger **EN** is in the ON-state.



The **JP** function will skip all instructions between a **JP** and an **LBL** of the same number. When the **JP** instruction is executed, the execution time of the skipped instructions is not included in the scan time. Two or more **JP** functions with the same number **Num*** can be used in a program. However, no two **LBL** instructions may be identically numbered. **LBL** instructions are specified as destinations of **JP**, LOOP (see page 1012) and F19_SJP (see page 1010) instructions.

One **JP** and LBL instruction pair can be programmed between another pair. This construction is called nesting.

☞

- **It is not possible to use this function in a function block POU.**
- **The maximum possible value that can be assigned to Num* depends on the PLC type.**

PLC types **Availability of** JP **(see page 1328)**

Data types

Variable	Data type	Function
Num*	constant	Constant number that must correspond to LBL number, this "nested" program is jumped over

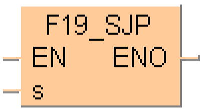
Example In this example, the programming language Instruction List (IL) is used.

```
IL  LD      start       (* EN = start; Starting signal for the JP function. *)
    JP      1           (* Num* = 1 (Address of Label) *)
```

F19_SJP

Indirect jump to label

Description Jumps to the label LBL (see page 1013) **s** with the same number as the data stored in the area specified by **s** if the trigger **EN** is in the ON-state.



This instruction also exists as a P instruction (for FP2/2SH, FP3/5, FP10/10SH PLC types), which is only executed at the rising edge of the EN trigger. Select **[Insert P instruction]** from the "Instructions" pane if you require a P instruction. To facilitate reuse, the instruction then appears under "Recently used" in the pop-up menu. Press **<Ctrl>+<Shift>+<v>** within the programming area to open the list of recently used elements.

The range of the number **s** can be between 0 and 255.

PLC types **Availability of** F19_SJP **(see page 1322)**

Data types

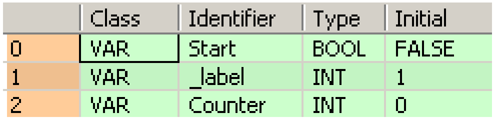
Variable	Data type	Function
s	ANY16	Stores label number (0 to 255)

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	-

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

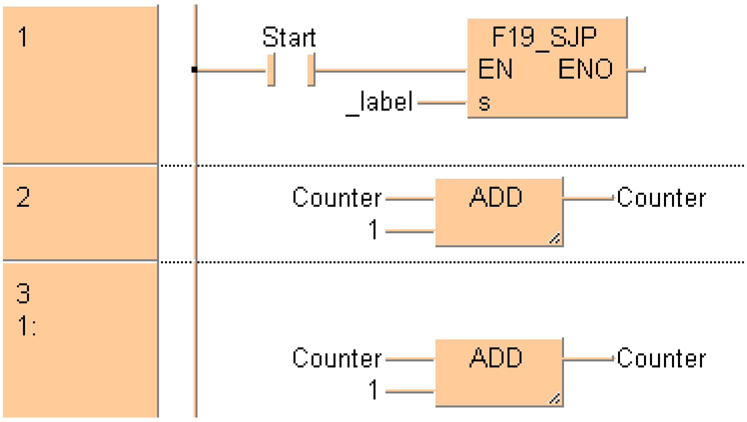
POU header The same POU header is used for all programming languages.



	Class	Identifier	Type	Initial
0	VAR	Start	BOOL	FALSE
1	VAR	_label	INT	1
2	VAR	Counter	INT	0

Body When the variable **start** is set to TRUE, the function is carried out.

LD



ST

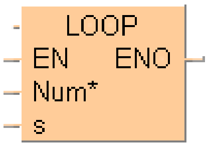
When programming with structured text, enter the following:

```
(* if Start is true Counter will be incremented by 1, else by 2 *)
IF Start THEN
    F19_SJP(_label);
END_IF;
Counter:=Counter+1;

LBL(1);
Counter:=Counter+1;
```

LOOP

Loop to label

Description LOOP (Loop to Label) instruction skips to the **LBL** (see page 1013) instruction with the same number **Num*** as the **LOOP** instruction and repeats execution of what follows until the data of a specified operand becomes "0".



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

The **LBL** instructions are specified as destination of the **LOOP** instruction. It is not possible to specify two or more **LBL** instructions with the same number **Num*** within a program. If the set value **s** in the data area is "0" from the beginning, the **LOOP** instruction is not executed (ignored).

☞

- **It is not possible to use this function in a function block POU.**
- **The maximum possible value that can be assigned to Num* depends on the PLC type.**

PLC types **Availability of** LOOP **(see page 1328)**

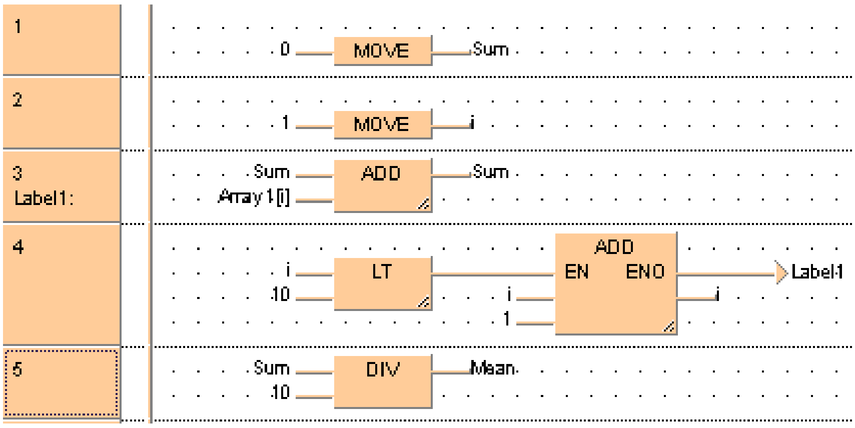
Data types

Variable	Data type	Function
s	INT, WORD	Set value
Num*	constant	Constant number that must correspond to LBL number, this "nested" program is looped until the variable at **s** reaches 0

Operands

For	Relay				T/C		Register			Constant
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	-

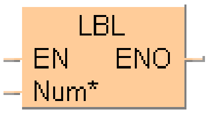
Example In this example the function is programmed in ladder diagram (LD).



LBL

Label for the JP- and LOOP-instruction

Description The **LBL** (Label for the JP and LOOP) instruction skips to the **LBL** instruction with the same number **Num*** as the JUMP (see page 1009) instruction if the predetermined trigger **EN** is in the ON-state.



Skips to the **LBL** instruction with the same number **Num*** as the **LOOP** (see page 1012) instruction and repeats execution of what follows until the data of a specified operand becomes "0".

☞

- **It is not possible to use this function in a function block POU.**
- **The maximum possible value that can be assigned to Num* depends on the PLC type.**

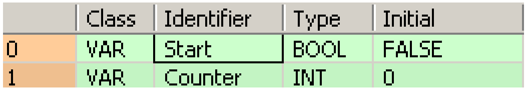
PLC types **Availability of** LBL **(see page 1328)**

Data types

Variable	Data type	Function
Num*	constant	Constant number that must correspond to JP, LOOP or F19 label number

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

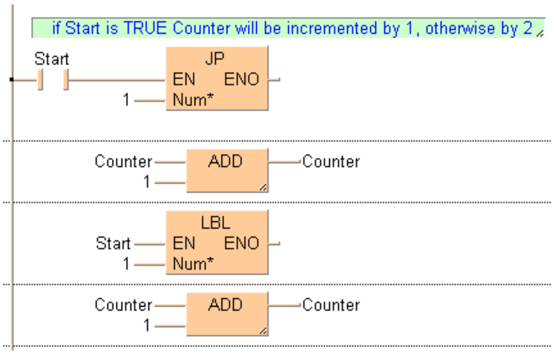
POU header The same POU header is used for all programming languages.



	Class	Identifier	Type	Initial
0	VAR	Start	BOOL	FALSE
1	VAR	Counter	INT	0

Body When the variable **start** is set to TRUE, the function is carried out.

LD



if Start is TRUE Counter will be incremented by 1, otherwise by 2

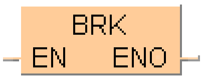
ST When programming with structured text, enter the following:

```
(* if Start is true Counter will be incremented by 1, else by 2 *)
IF Start THEN
    JP(1);
END_IF;

Counter:=Counter+1;

LBL(1);
Counter:=Counter+1;
```

BRK

Break

Description The **BRK** (Breakpoint) instruction stops the execution at the address of this instruction during the test run mode if the trigger **EN** is in the ON-state.

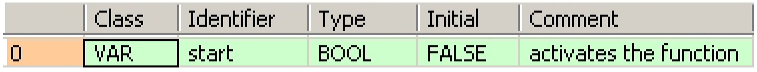


Once this instruction is executed, the program halts. To continue the program, the mode in the test run (continuous run / step run) should be selected. In the step run mode, the program is executed instruction by instruction regardless of the instructions and in the continuous run mode, the program is executed until it is stopped by the next break instruction (BRK) or the end of the program (end instruction ED).

The test run mode is executed, when the mode selector switch on the PLC is set to RUN mode with setting the INITIALIZE/TEST switch to the TEST mode.

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

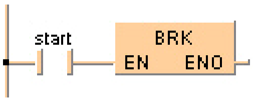
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function

Body When the variable **start** is set to TRUE, the function is carried out.

LD



ST When programming with structured text, enter the following:

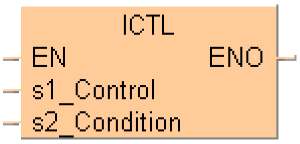
```
IF start THEN
    BRK();
END_IF;
```

ICTL

Interrupt Control

Description The **ICTL** instruction sets all interrupts to enable or disable. Each time the **ICTL** instruction is executed, it is possible to set parameters like the type and validity of interrupt programs. Settings can be specified by **s1_Control** and **s2_Condition**.

- **s1_Control** 16-bit equivalent constant or 16-bit area for interrupt control setting
- **s2_Condition** 16-bit equivalent constant or 16-bit area for interrupt trigger condition setting



The number of interrupt programs available is:

- **16** interrupt module initiated interrupt programs (INT 0–INT 15)
- **8** advanced module (special modules, like positioning,...) initiated interrupt programs (INT 16–INT 23)
- **1** periodic interrupt program (INT 24) (Time base 0.5ms selectable for FP2/2SH, FP10SH)

Be sure to use ICTL instructions so that they are executed once at the rising edge of the ICTL trigger using the DF instruction.

Two or more ICTL instructions can have the same trigger.

Bit	15 .. 8	7 .. 0
s1_Control **16#**	**Selection of control function** **00**: Interrupt "enable/disable" control **01**: Interrupt trigger reset control	**Interrupt type selection** **00**: Interrupt module (INT 0–15) **01**: Advanced module (INT 16–23) **02**: Periodic interrupt (INT 24)
s2_Condition **2#**	Bit 0: **0** Interrupt program 0 disabled Bit 0: **1** Interrupt program 0 enabled Bit 1: **0** Interrupt program 1 disabled ... Bit 15: **1** Interrupt program 15 enabled Example: s2 = 2#0000000000001010	

- **The current enable/disable status of each interrupt module initiated interrupt can be checked by monitoring the special data register (see page 1254) DT90025.**
- **The current enable/disable status of each non-interrupt module initiated interrupt can be checked by monitoring the special data register DT90026.**
- **The current interrupt interval of the periodic interrupt can be checked by monitoring the special data register DT90027.**
- **If a program is written into an interrupt task, the interrupt concerned will be enabled automatically during the initialization routine when starting the program.**
- **With the ICTL instruction an interrupt task can be enabled or disabled by the program.**

PLC types **Availability of** ICTL **(see page** 1327**)**

Data types

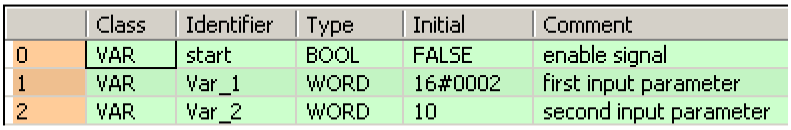
Variable	Data type	Function
s1	ANY16	Interrupt control data setting
s2		Interrupt condition setting

Operands

For	Relay				T/C		Register			Constant
s1, s2	-	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.

Example In this example, the same POU header is used for all programming languages. For an example using IL (instruction list), please refer to the online help.

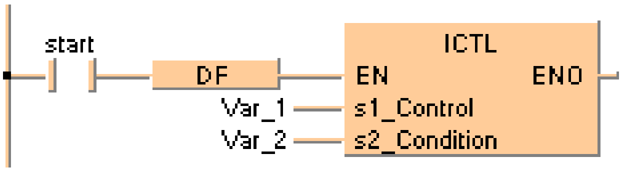
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	enable signal
1	VAR	Var_1	WORD	16#0002	first input parameter
2	VAR	Var_2	WORD	10	second input parameter

Body The interval for executing the periodic interrupt is specified as 100ms (10ms time base selected) when the rising edge of start is detected.

LD



Chapter 35

Pulse output instructions

35.1 Introduction

Control FPWIN Pro offers two concepts for programming with pulse output instructions:

- FP instructions
- Tool instructions

For users programming for different PLC types of the FP series or users who are tired of setting control code bits and looking up available channel numbers, the tool instructions offer new and comfortable features. These include information functions for evaluating status flags and settings, control functions for configuring high-speed counters and pulse outputs, PLC-independent functions and DUTs, as well as variable channel numbers. However, the FP instructions may be easier to use for beginners or users familiar with FPWIN GR.

Most of the information, which is accessible via information and control functions, is stored in special internal relays and special data registers. These relays and registers can also be accessed using PLC-independent system variables.

To take advantage of the features you prefer, the instructions of both libraries can be mixed.

◆**NOTE**

When programming with the tool instructions, be sure to refer to the detailed information provided via the links to the related F/P instructions.

Main features	FP instructions	Tool instructions
Pre version 6.4 support	●	
Use of inline functions	●	
Use of FPWIN GR function names	●	
Less code with constant channel numbers	●	
Control codes	●	
Control functions		●
Information functions		●
Variable channel numbers		●
Universal functions for all PLCs		●
DUT for common channel configuration for all PLCs for all pulse output instructions		●

35.2 Writing the pulse output control code

The special data register where the high-speed counter and pulse output control code are stored can be accessed with the system variable sys_wHscOrPulseControlCode. (The system variable sys_wHscOrPulseControlCode corresponds to special data register DT90052.)

Operations performed by the pulse output control code

- Setting/resetting near home input
- Continuing/stopping pulse output (forced stop)
- Enabling/disabling counting operations
- Resetting the elapsed value (software reset) of the high-speed counter
- Clearing high-speed counter and position control instructions (FP0R only)

The control code settings for each channel can be monitored using the system variables sys_wHscChannelxControlCode or sys_wPulseChannelxControlCode (where x=channel number).

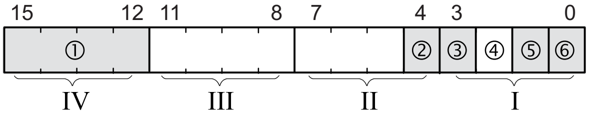
The settings of this system variable remain unchanged until another setting operation is executed.

☞

- **Performing a forced stop may cause the elapsed value at the PLC output side to differ from the elapsed value at the motor input side. Therefore, you must execute a home return after pulse output has stopped.**
- **Setting the near home input is not possible if counting is prohibited or if a software reset is performed.**

Description for FPΣ:

Bits 0–15 of the control code are allocated in groups of four. The bit setting in each group is represented by a hex number (e.g. 0002 0000 0000 1001 = 16#2009).



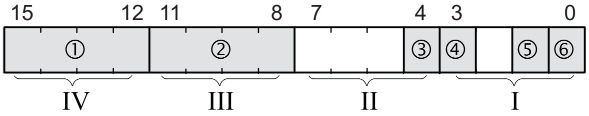
Group IV	①	Channel number (channel n: 16#n)	
Group III		0 (fixed)	
Group II	②	Near home input (bit 4)	
		0: FALSE	1: TRUE
Group I	③	Pulse output (bit 3)	
		0: continue	1: stop
	④	0 (bit 2, fixed)	
	⑤	Count (bit 1)	
		0: permit	1: prohibit
	⑥	Reset elapsed value to 0 (bit 0)	
		0: no	1: yes

Example: 16#2009

Group	Value	Description	
IV	2	Channel number: 2	
III	0	(fixed)	
II	0	Near home input: FALSE	
I	9	Hex 9 corresponds to binary 1001	
		Pulse output: stop (bit 3)	1
		(bit 2, fixed)	0
		Count: permit (bit 1)	0
		Reset elapsed value to 0: yes (bit 0)	1

Description for FP-X:

Bits 0–15 of the control code are allocated in groups of four. The bit setting in each group is represented by a hex number (e.g. 0002 0001 0000 1001 = 16#2109).



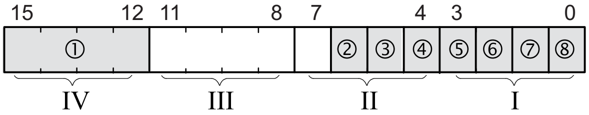
Group IV	①	Channel number (channel n: 16#n)	
Group III	②	1 (fixed)	
Group II	③	Near home input (bit 4) (see note)	
		0: FALSE	1: TRUE
Group I	④	Pulse output (bit 3)	
		0: continue	1: stop
	⑤	Count (bit 1)	
		0: permit	1: prohibit
	⑥	Reset elapsed value to 0 (bit 0)	
		0: no	1: yes

Example: 16#2109

Group	Value	Description	
IV	2	Channel number: 2	
III	1	(fixed)	
II	0	Near home input: FALSE	
I	9	Hex 9 corresponds to binary 1001	
		Pulse output: stop (bit 3)	1
		(Bit 2 fixed)	0
		Count: permit (bit 1)	0
		Reset elapsed value to 0: yes (bit 0)	1

Description for FP0R:

Bits 0–15 of the control code are allocated in groups of four. The bit setting in each group is represented by a hex number (e.g. 0002 0001 0000 1001 = 16#2109).



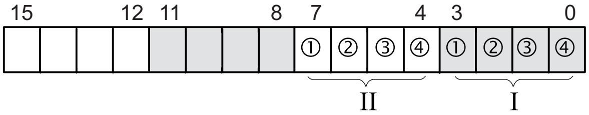
Group IV	①	Channel number (channel n: 16#n)	
Group III		1 (fixed)	
Group II	②	Position control start request	
		0: disabled	1: enabled
	③	Decelerated stop request	
		0: disabled	1: enabled
	④	Near home input (bit 4) (see note)	
		0: FALSE	1: TRUE
Group I	⑤	Pulse output (bit 3)	
		0: continue	1: stop
	⑥	Clear pulse output control (bit 2)	
		0: continue	1: stop
	⑦	Count (bit 1)	
		0: permit	1: prohibit
	⑧	Reset elapsed value to 0 (bit 0)	
		0: no	1: yes

Example: 16#2109

Group	Value	Description	
IV	2	Channel number: 2	
III	1	(fixed)	
II	0	Position control start request: disabled	
		Decelerated stop request: disabled	
		Near home input: FALSE	
I	9	Hex 9 corresponds to binary 1001	
		Pulse output: stop (bit 3)	1
		Clear pulse output control (bit 2)	0
		Count: permit (bit 1)	0
		Reset elapsed value to 0: yes (bit 0)	1

Description for FP0, FP-e:

Bits 0–15 of the control code are allocated in groups of four, each group containing the settings for one channel. The bit setting in each group is represented by a hex number (e.g. 0000 0000 1001 0000 = 16#90).



Group	II	I
Channel	1	0

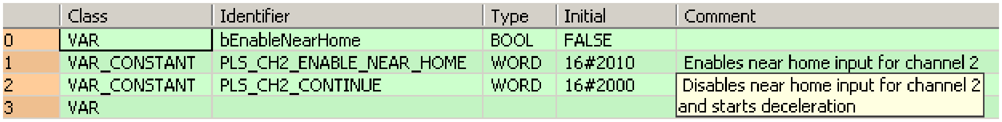
①	Pulse output (bit 3)	
	0: continue	1: stop
②	Near home input (bit 2) (see note)	
	0: FALSE	1: TRUE
③	Count (bit 1)	
	0: permit	1: prohibit
④	Reset elapsed value to 0 (bit 0)	
	0: no	1: yes

Example: 16#90

Group	Value	Description	
II	9	Channel number: 1 Hex 9 corresponds to binary 1001	
		Pulse output: stop (bit 3)	1
		Near home input: FALSE (bit 2)	0
		Count: permit (bit 1)	0
		Reset elapsed value to 0: yes (bit 0)	1
I	0	–	

Example The first example shows how to enable the near home input for channel 2, and the second example shows how to perform pulse output stop for channel 0.

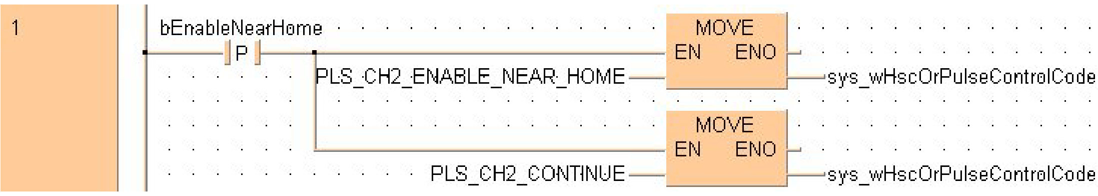
All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	bEnableNearHome	BOOL	FALSE	
1	VAR_CONSTANT	PLS_CH2_ENABLE_NEAR_HOME	WORD	16#2010	Enables near home input for channel 2
2	VAR_CONSTANT	PLS_CH2_CONTINUE	WORD	16#2000	Disables near home input for channel 2 and starts deceleration
3	VAR				

Body The near home input is enabled for channel 2 during home return operations.

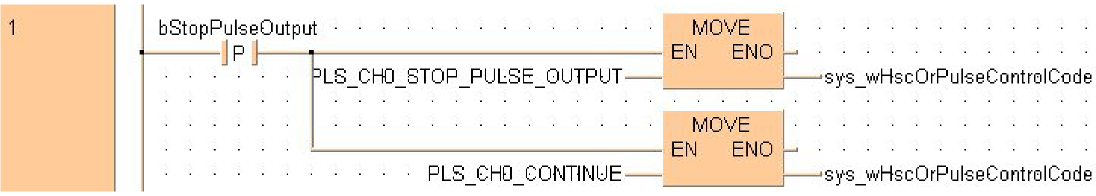
LD



Performing a forced stop for channel 0 (FP0, FP-e, FPΣ)

Body A forced stop of the pulse output is performed for channel 0.

LD


1
bStopPulseOutput
P
MOVE
EN
ENO
PLS_CH0_STOP_PULSE_OUTPUT
sys_wHscOrPulseControlCode
MOVE
EN
ENO
PLS_CH0_CONTINUE
sys_wHscOrPulseControlCode

☞ **Performing a forced stop may cause the elapsed value at the PLC output side to differ from the elapsed value at the motor input side. Therefore, you must execute a home return after pulse output has stopped.**

35.3 Pulse output: writing and reading the elapsed value

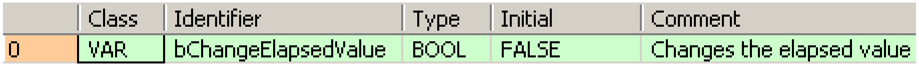
The elapsed value is stored as a double word in the special data registers. Access the special data registers using the system variable sys_diPulseChannelxElapsedValue (where x=channel number).

System variables for memory areas used:

- FP-Sigma
- FP-X, Transistor types
- FP-X, Relay types
- FP0R
- FP0

Example The first example shows how to write an initial value (elapsed value) into the high-speed counter. The second example shows how to read an elapsed value and copy it to a variable.

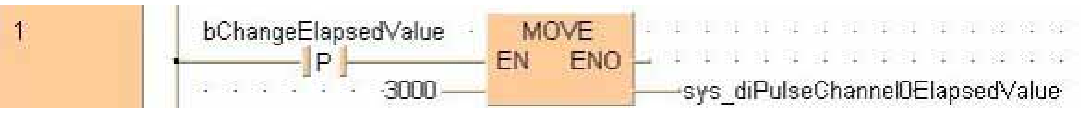
All input and output variables used for programming this function have been declared in the POU header.



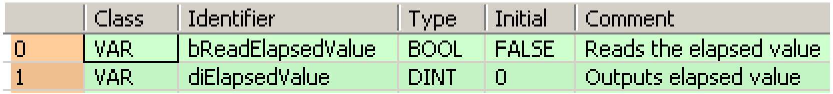
	Class	Identifier	Type	Initial	Comment
0	VAR	bChangeElapsedValue	BOOL	FALSE	Changes the elapsed value

Body An initial value of 3000 (elapsed value) is written into channel 0 of the high-speed counter.

LD



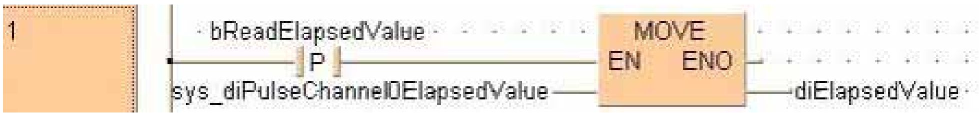
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	bReadElapsedValue	BOOL	FALSE	Reads the elapsed value
1	VAR	diElapsedValue	DINT	0	Outputs elapsed value

Body The elapsed value of the high-speed counter is read from channel 0 of the high-speed counter and copied to the variable diElapsedValue.

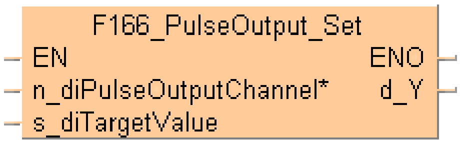
LD



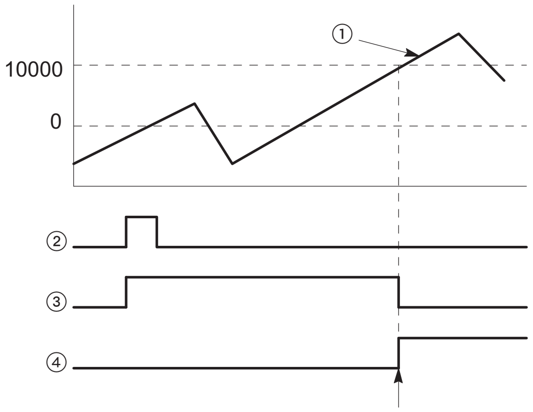
F166_PulseOutput_Set

Target value match ON (pulse output)

Description If the elapsed value matches the target value of the selected pulse output channel, the specified output immediately turns to TRUE.



Pulse output characteristics



10000	Target value
①	Elapsed value of pulse output
②	Execution condition
③	"Output control active" flag
④	PLC output

The PLC output turns to TRUE when the elapsed value matches the target value. In addition, the "Output control active" flag turns to FALSE and the instruction is deactivated.

If an output is specified that has not been implemented, only the internal memory of the corresponding WY address is set or reset.

Interrupt operation

The interrupt program will be executed when the elapsed value matches the target value. Any interrupt that has been entered into the Tasks list is automatically enabled. A special interrupt program number is assigned to each channel number.

Channels used by interrupt programs:

Interrupt 8	Channel 0
Interrupt 9	Channel 1
Interrupt 10	Channel 2
Interrupt 11	Channel 3

■ General programming information

- Set "Pulse output" for the desired channel in the system registers.

- When this instruction is executed, the "Output control active" flag (e.g. sys_bIsPulseChannel0ControlActive) for the channel used turns to TRUE. No other high-speed counter instruction with output control (F166_PulseOutput_Set or F167_PulseOutput_Reset) using the same channel can be executed as long as this flag is TRUE.
- This instruction is available for all pulse output instructions except F173_PulseOutput_PWM (see page 1066) and can be executed before or after execution of a pulse output instruction.
- The duplicate use of an external output relay in other instructions (OUT, SET, RST, KEEP and other F instructions) is not verified by FPWIN Pro and will not be detected.
- To set a PLC output to FALSE that was previously set to TRUE by this instruction, use an RST or MOVE instruction.
- To cancel execution of a pulse output instruction, set bit 2 of the data register storing the pulse output control code (sys_wHscOrPulseControlCode) to TRUE. The pulse output control flag will then change to FALSE. To reenable execution of the instruction, reset bit 2 to FALSE. However, pulse output will continue.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- We strongly recommend that you incorporate a forced stop (see page 1021) option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

PLC types **Availability of F166_PulseOutput_Set (see page 1322)**

Data types

Variable	Data type	Function
n_diPulseOutputChannel	DINT	Pulse output channel: 0–3
s_diTargetValue	DINT	specify a 32-bit data value for the target value within the following range: -2147483467–+2147483648
d_Y	BOOL	output which turns to TRUE when the elapsed value matches the target value: Y0–Y1F

Operands

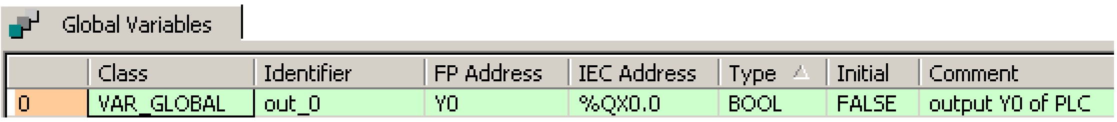
For	Relay				T/C		Register			Constant
n_diPulseOutputChannel	-	-	-	-	-	-	-	-	-	dec. or hex.
s_diTargetValue	DWX	DWY	DWR	-	DSV	DEV	DDT	-	-	-
d_Y	-	Y	-	-	-	-	-	-	-	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	ON	▪ channel number or values of the data table are outside the permissible range ▪ pulse output has not been set in the system registers
R9008	%MX0.900.8	ON	

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

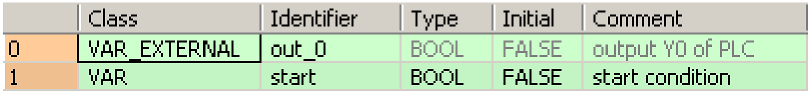
GVL In the global variable list, you define variables that can be accessed by all POUs in the project.



Global Variables

	Class	Identifier	FP Address	IEC Address	Type	Initial	Comment
0	VAR_GLOBAL	out_0	Y0	%QX0.0	BOOL	FALSE	output Y0 of PLC

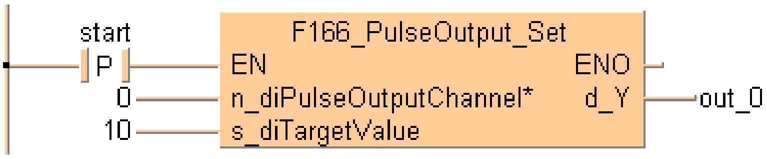
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	out_0	BOOL	FALSE	output Y0 of PLC
1	VAR	start	BOOL	FALSE	start condition

Body When the variable **start** is set to TRUE, the function is carried out.

LD



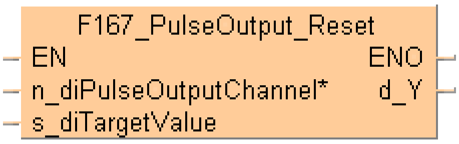
ST When programming with structured text, enter the following:

```
IF DF(start) THEN
      F166_PulseOutput_Set(n_diPulseOutputChannel := 0,
        s_diTargetValue := 10,
        d_Y => out_0);
END_IF;
```

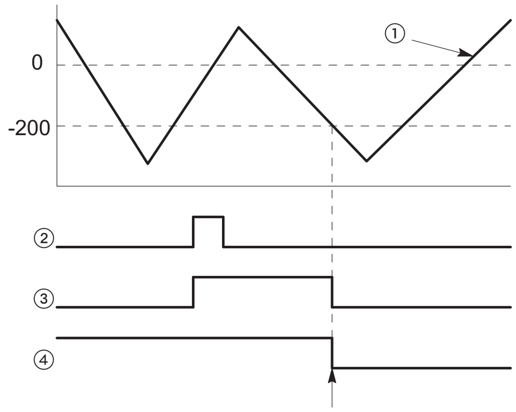
F167_PulseOutput_Reset

Target value match OFF (pulse output)

Description If the elapsed value matches the target value of the pulse output channel, the specified output immediately turns to FALSE.



Pulse output characteristics



-200	Target value
①	Elapsed value of pulse output
②	Execution condition
③	"Output control active" flag
④	PLC output

The PLC output turns to FALSE when the elapsed value matches the target value. In addition, the "Output control active" flag turns to FALSE and the instruction is deactivated.

If an output is specified that has not been implemented, only the internal memory of the corresponding WY address is set or reset.

Interrupt operation

The interrupt program will be executed when the elapsed value matches the target value. Any interrupt that has been entered into the Tasks list is automatically enabled. A special interrupt program number is assigned to each channel number.

Interrupt 8	Channel 0
Interrupt 9	Channel 1
Interrupt 10	Channel 2
Interrupt 11	Channel 3

■ General programming information

- Set "Pulse output" for the desired channel in the system registers.
- When this instruction is executed, the "Output control active" flag (e.g. sys_bIsPulseChannel0ControlActive) for the channel used turns to TRUE. No

other high-speed counter instruction with output control (F166_PulseOutput_Set or F167_PulseOutput_Reset) using the same channel can be executed as long as this flag is TRUE.

- This instruction is available for all pulse output instructions except F173_PulseOutput_PWM (see page 1066) and can be executed before or after execution of a pulse output instruction.
- The duplicate use of an external output relay in other instructions (OUT, SET, RST, KEEP and other F instructions) is not verified by FPWIN Pro and will not be detected.
- To cancel execution of a pulse output instruction, set bit 2 of the data register storing the pulse output control code (sys_wHscOrPulseControlCode) to TRUE. The pulse output control flag will then change to FALSE. To reenable execution of the instruction, reset bit 2 to FALSE. However, pulse output will continue.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- We strongly recommend that you incorporate a forced stop (see page 1021) option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

PLC types **Availability of F167_PulseOutput_Reset (see page 1322)**

Data types

Variable	Data type	Function
n_diPulseOutputChannel	DINT	Pulse output channel: 0–3
s_diTargetValue	DINT	specify a 32-bit data value for the target value within the following range: -2147483467–+2147483648
d_Y	BOOL	output which turns to FALSE when the elapsed value matches the target value: Y0–Y1F

Operands

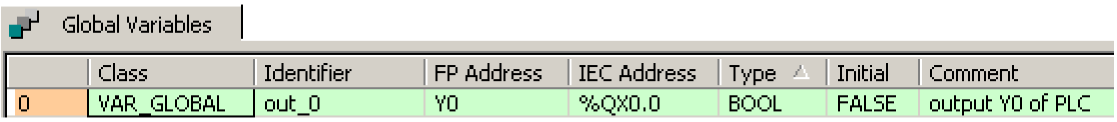
For	Relay				T/C		Register			Constant
n_diPulseOutputChannel	-	-	-	-	-	-	-	-	-	dec. or hex.
s_diTargetValue	DWX	DWY	DWR	-	DSV	DEV	DDT	-	-	-
d_Y	-	Y	-	-	-	-	-	-	-	-

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	ON	▪ channel number or values of the data table are outside the permissible range ▪ pulse output has not been set in the system registers
R9008	%MX0.900.8	ON	

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

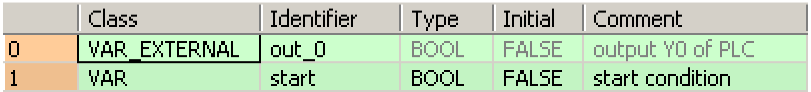
GVL In the global variable list, you define variables that can be accessed by all POUs in the project.



Global Variables

	Class	Identifier	FP Address	IEC Address	Type	Initial	Comment
0	VAR_GLOBAL	out_0	Y0	%QX0.0	BOOL	FALSE	output Y0 of PLC

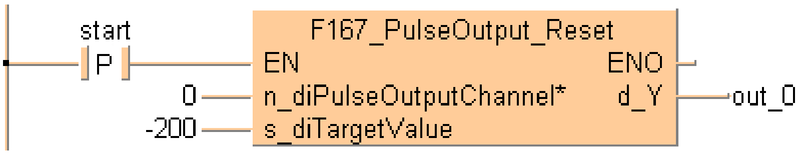
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	out_0	BOOL	FALSE	output Y0 of PLC
1	VAR	start	BOOL	FALSE	start condition

Body When the variable **start** is set to TRUE, the function is carried out.

LD



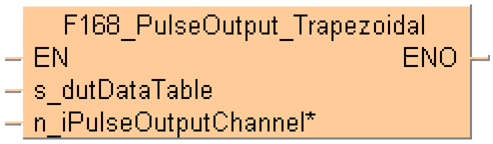
ST When programming with structured text, enter the following:

```
IF DF(start) THEN
      F167_PulseOutput_Reset(n_diPulseOutputChannel := 0,
        s_diTargetValue := -200,
        d_Y => out_0);
END_IF;
```

F168_PulseOutput_Trapezoidal

Trapezoidal control

Description This instruction automatically performs trapezoidal control according to the parameters in the specified DUT. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



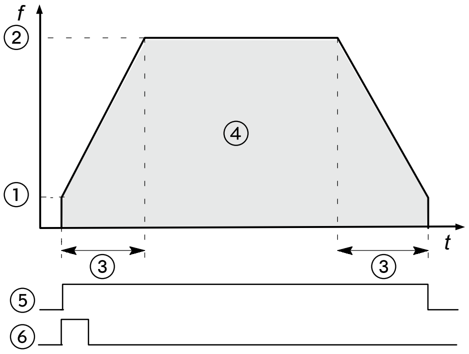
See also: PulseOutput_Trapezoidal_FB (see page 1194)

Use the following predefined DUT:

F168_PulseOutput_Trapezoidal_DUT

- Control code
- Initial and final speed
- Target speed
- Acceleration/deceleration time
- Target value
- Pulse stop (fixed)

Pulse output characteristics



①	Initial and final speed	④	Target value
②	Target speed	⑤	Pulse output control flag
③	Acceleration/deceleration time	⑥	Execution condition

The pulse output frequency changes according to the specified acceleration/deceleration time.

The difference between target and initial speed determines the slope of the ramps.

General programming information

- Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- When programs are being edited in RUN mode, pulse output stops but resumes after the program changes have been downloaded.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- The high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and

the pulse output control flag (e.g. sys_bIsPulseChannel0Active) are assigned to the same internal relay (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.

- Pulse output stops when the upper limit of the internal elapsed value is exceeded if rotation is in one direction only. As a countermeasure, reset the elapsed value to 0 before executing this instruction. Pulse output does not stop when the FP0R is used in FP0 compatibility mode because the data range for the elapsed value is a signed 32-bit value.
- We strongly recommend that you incorporate a forced stop (see page 1021) option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

■ Running the FP0R in FP0 compatibility mode

To run the FP0R in FP0 compatibility mode, you can download an FP0 program to the FP0R. Please note the following restrictions:

- The FP0R supports signed 32-bit data for elapsed value and target value; the FP0 supports signed 24-bit data. In FP0 compatibility mode, counting and pulse output continue even if data exceeds the FP0 range.
- The duty ratio is always 25% regardless of the settings in the instructions. With the pulse output method "pulse/direction", pulses are output approx. 300μs after the direction signal has been output; the motor driver characteristics are simultaneously taken into consideration.
- The FP0R does not support the "no counting" setting. Instead, incremental counting is performed with the FP0 pulse output instructions set to "no counting".
- The maximum pulse output frequency is 10000Hz.
- Make sure the pulse output instruction does not use an output that is also being used as a normal output.
- For an FP0 program to be able to run in FP0 compatibility mode, the PLC types (C10, C14, C16, C32, and T32) must match exactly. FP0 compatibility mode is not available for the F32 type FP0R.

PLC types **Availability of F168_PulseOutput_Trapezoidal (see page 1322)**

Data types

Variable	Data type	Function
s_dutDataTable	F168_PulseOutput_Trapezoidal_DUT	Starting address of area containing the data table
n_iPulseOutputChannel	decimal constant	Pulse output: 0 or 1

Operands

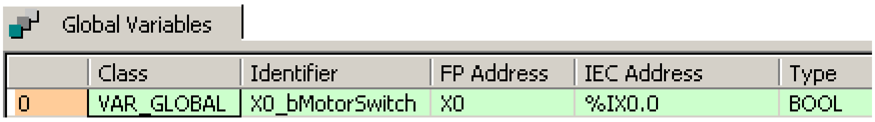
For	Relay				T/C		Register			Constant
s_dutDataTable	-	-	-	-	-	-	DT	-	-	-
n_iPulseOutputChannel	-	-	-	-	-	-	-	-	-	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ channel number or values of the data table are outside the permissible range ▪ initial speed < 40 ▪ initial speed > maximum speed
R9008	%MX0.900.8	for an instant	

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

GVL In the global variable list, you define variables that can be accessed by all POUs in the project.

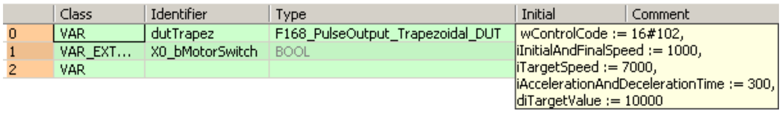


Global Variables

	Class	Identifier	FP Address	IEC Address	Type
0	VAR_GLOBAL	X0_bMotorSwitch	X0	%IX0.0	BOOL

DUT The DUT F168_PulseOutput_Trapezoidal_DUT is predefined in the FP Library.

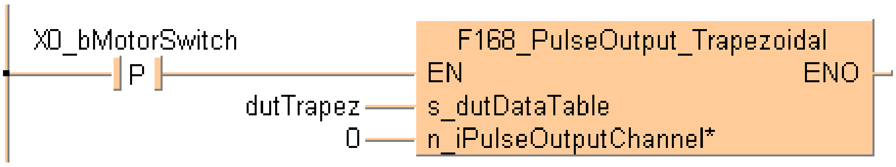
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	dutTrapez	F168_PulseOutput_Trapezoidal_DUT	wControlCode := 16#102, iInitialAndFinalSpeed := 1000, iTargetSpeed := 7000, iAccelerationAndDecelerationTime := 300, diTargetValue := 10000	
1	VAR_EXT...	X0_bMotorSwitch	BOOL		
2	VAR				

Body When **X0_bMotorSwitch** turns to TRUE the function is executed.

LD



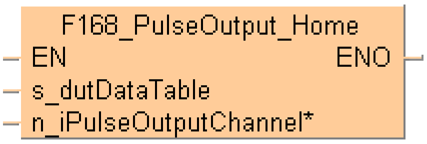
ST When programming with structured text, enter the following:

```
IF DF(X0_bMotorSwitch) THEN
      F168_PulseOutput_Trapezoidal(s_dutDataTable := dutTrapez,
      n_iPulseOutputChannel := 0);
END_IF;
```

F168_PulseOutput_Home

Home return

Description This instruction performs a home return according to the parameters in the specified DUT. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



See also:

- PulseOutput_Home_FB (see page 1176)
- PulseControl_NearHome (see page 1209)

After a drive system has been switched on, there is a difference between the internal position value (elapsed value) and the mechanical position of the axis; this difference cannot be predetermined. The internal value must be synchronized with the actual position value of the axis. This is done by means of a home return, during which a position value is registered at a known reference point (home).

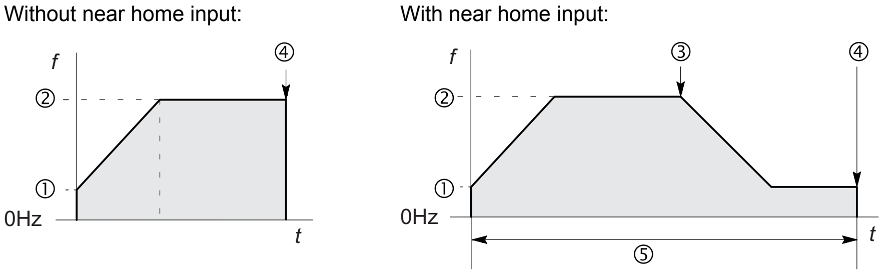
During execution of a home return instruction, pulses are continuously output until the home input is enabled. The I/O allocation is determined by the channel used.

To decelerate movement when near the home position, designate a near home input and set bit 4 of the special data register storing the pulse output control code (sys_wHscOrPulseControlCode) to TRUE and back to FALSE again.

The value in the elapsed value area during a home return differs from the current value. When the return is completed, the elapsed value changes to 0.

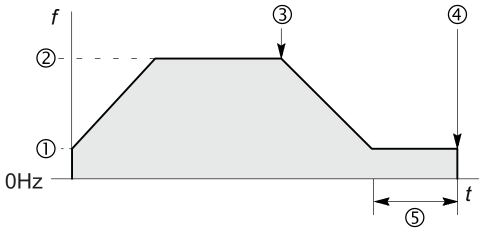
Select one of two different operation modes:

- Type 1: The home input is effective regardless of whether or not there is a near home input, whether deceleration is taking place, or whether deceleration has been completed.



①	Initial and final speed	③	Near home input: TRUE
②	Target speed	④	Home input: TRUE
⑤	Home input is effective at any time.		

- Type 2: The home input is effective only after deceleration (started by near home input) has been completed.



①	Initial and final speed	③	Near home input: TRUE
②	Target speed	④	Home input: TRUE
⑤	Home input is effective only after deceleration		

Use the following predefined DUT: F168_PulseOutput_Home_DUT

The following parameters can be specified in the DUT:

- Control code
- Initial and final speed
- Target speed
- Acceleration/deceleration time
- Pulse stop (fixed)

Pulse output characteristics

- The pulse output frequency changes according to the specified acceleration/deceleration time.
- The difference between target and initial speed determines the slope of the ramps.

General programming information

- Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- When programs are being edited in RUN mode, pulse output stops but resumes after the program changes have been downloaded.
- The high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) are assigned to the same internal relay (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- Even when home input has occurred, executing this instruction causes pulse output to begin.
- If the near home input is enabled while acceleration is in progress, deceleration will start.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- We strongly recommend that you incorporate a forced stop (see page 1021) option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more

than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

■ Running the FP0R in FP0 compatibility mode

To run the FP0R in FP0 compatibility mode, you can download an FP0 program to the FP0R. Please note the following restrictions:

- The FP0R supports signed 32-bit data for elapsed value and target value; the FP0 supports signed 24-bit data. In FP0 compatibility mode, counting and pulse output continue even if data exceeds the FP0 range.
- The duty ratio is always 25% regardless of the settings in the instructions. With the pulse output method "pulse/direction", pulses are output approx. 300µs after the direction signal has been output; the motor driver characteristics are simultaneously taken into consideration.
- The FP0R does not support the "no counting" setting. Instead, incremental counting is performed with the FP0 pulse output instructions set to "no counting".
- The maximum pulse output frequency is 10000Hz.
- Make sure the pulse output instruction does not use an output that is also being used as a normal output.
- For an FP0 program to be able to run in FP0 compatibility mode, the PLC types (C10, C14, C16, C32, and T32) must match exactly. FP0 compatibility mode is not available for the F32 type FP0R.

PLC types **Availability of F168_PulseOutput_Home (see page 1322)**

Data types

Variable	Data type	Function
s_dutDataTable	F168_PulseOutput_Home_DUT	Starting address of area containing the data table
n_iPulseOutputChannel	decimal constant	Pulse output: 0 or 1

Operands

For	Relay				T/C		Register			Constant
s_dutDataTable	-	-	-	-	-	-	DT	-	-	-
n_iPulseOutputChannel	-	-	-	-	-	-	-	-	-	dec. or hex.

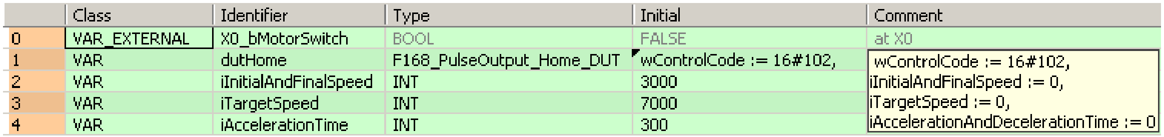
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ channel number or values of the data table are outside the permissible range ▪ initial speed < 40 ▪ initial speed > maximum speed
R9008	%MX0.900.8	for an instant	

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

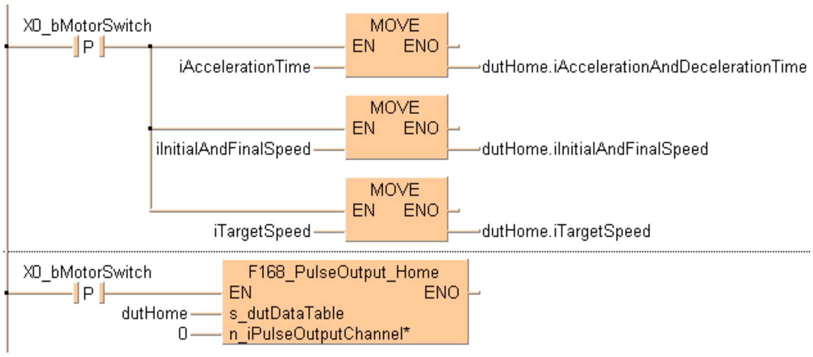
DUT The DUT F168_PulseOutput_Home_DUT is predefined in the FP Library.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	X0_bMotorSwitch	BOOL	FALSE	at X0
1	VAR	dutHome	F168_PulseOutput_Home_DUT	wControlCode := 16#102,	wControlCode := 16#102, iInitialAndFinalSpeed := 0, iTargetSpeed := 0, iAccelerationAndDecelerationTime := 0
2	VAR	iInitialAndFinalSpeed	INT	3000	
3	VAR	iTargetSpeed	INT	7000	
4	VAR	iAccelerationTime	INT	300	

LD



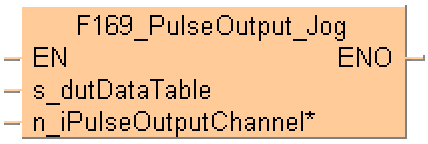
ST When programming with structured text, enter the following:

```
IF DF(X0_bMotorSwitch) THEN
    dutHome.iInitialAndFinalSpeed:=iInitialAndFinalSpeed
    dutHome.iTargetSpeed:=iTargetSpeed
    dutHome.iAccelerationAndDecelerationTime:=iAccelerationTime
END_IF;

IF DF(X0_bMotorSwitch) THEN
F168_PulseOutput_Home(s_dutDataTable := dutHome,
    n_iPulseOutputChannel := 0);
END_IF;
```

F169_PulseOutput_Jog

JOG operation

Description This instruction is used for JOG operation. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



Use the following predefined DUT: **F169_PulseOutput_Jog_DUT**

The following parameters can be specified in the DUT:

- Control code
- Speed

Pulse output characteristics

The frequency and the duty can be changed in each scan. (The change becomes effective with the next pulse output.)

General programming information

Warning!

As soon as you begin editing a program online (i.e., in RUN mode) using this instruction, pulse output will stop.

- Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- The high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) are assigned to the same internal relay (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- Pulse output stops when the upper limit of the internal elapsed value is exceeded if rotation is in one direction only. As a countermeasure, reset the elapsed value to 0 before executing this instruction. Pulse output does not stop when the FP0R is used in FP0 compatibility mode because the data range for the elapsed value is a signed 32-bit value.
- When using incremental counting, pulse output stops when the elapsed value exceeds 2147483647.
- When using decremental counting, pulse output stops when the elapsed value exceeds -2147483648.
- We strongly recommend that you incorporate a forced stop (see page 1021)

option in your positioning program.

- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

■ Running the FP0R in FP0 compatibility mode

To run the FP0R in FP0 compatibility mode, you can download an FP0 program to the FP0R. Please note the following restrictions:

- The FP0R supports signed 32-bit data for elapsed value and target value; the FP0 supports signed 24-bit data. In FP0 compatibility mode, counting and pulse output continue even if data exceeds the FP0 range.
- The duty ratio is always 25% regardless of the settings in the instructions. With the pulse output method "pulse/direction", pulses are output approx. 300μs after the direction signal has been output; the motor driver characteristics are simultaneously taken into consideration.
- The FP0R does not support the "no counting" setting. Instead, incremental counting is performed with the FP0 pulse output instructions set to "no counting".
- The maximum pulse output frequency is 10000Hz.
- Make sure the pulse output instruction does not use an output that is also being used as a normal output.
- For an FP0 program to be able to run in FP0 compatibility mode, the PLC types (C10, C14, C16, C32, and T32) must match exactly. FP0 compatibility mode is not available for the F32 type FP0R.

PLC types **Availability of F169_PulseOutput_Jog (see page 1322)**

Data types

Variable	Data type	Function
s_dutDataTable	F169_PulseOutput_Jog_DUT	Starting address of area containing the data table
n_iPulseOutputChannel	INT	Pulse output: 0 or 1

Operands

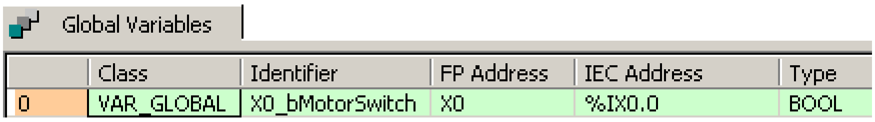
For	Relay				T/C		Register			Constant
s_dutDataTable	-	-	-	-	-	-	DT	-	-	-
n_iPulseOutputChannel	-	-	-	-	-	-	-	-	-	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	channel number or values of the data table are outside the permissible range
R9008	%MX0.900.8	for an instant	

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

GVL In the global variable list you define variables that can be accessed by all POUs in the project.



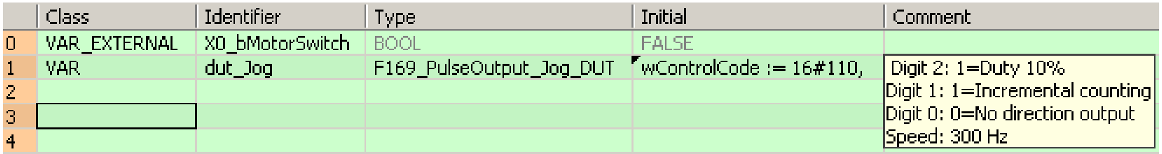
Global Variables

	Class	Identifier	FP Address	IEC Address	Type
0	VAR_GLOBAL	X0_bMotorSwitch	X0	%IX0.0	BOOL

DUT

The DUT F169_PulseOutput_Jog_DUT is predefined in the FP Library.

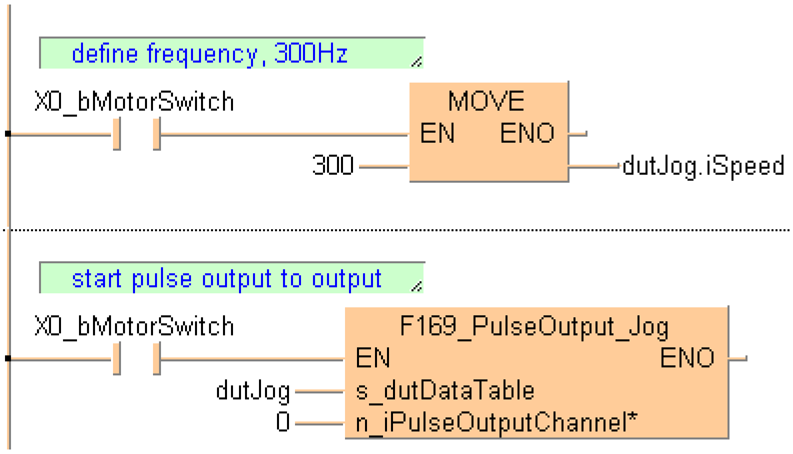
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	X0_bMotorSwitch	BOOL	FALSE	
1	VAR	dut_Jog	F169_PulseOutput_Jog_DUT	wControlCode := 16#110,	Digit 2: 1=Duty 10% Digit 1: 1=Incremental counting Digit 0: 0=No direction output Speed: 300 Hz
2					
3					
4					

Body The comment fields explain the function of this example.

LD



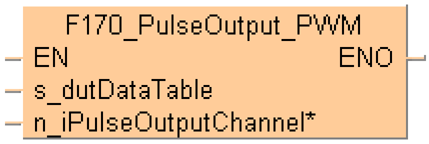
ST When programming with structured text, enter the following:

```
IF (X0_bMotorSwitch) THEN
      dutJog.ispeed := 300;
END_IF;
IF (X0_bMotorSwitch) THEN
      F169_PulseOutput_Jog(s_dutDataTable := dutJog,
  n_iPulseOutputChannel := 0);
END_IF;
```

F170_PulseOutput_PWM

PWM output

Description This instruction delivers a pulse width modulated output signal according to the specified DUT. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



Use the following predefined DUT: F170_PulseOutput_PWM_DUT

The following parameters can be specified in the DUT:

- Approximate frequency
- Duty ratio (for pulse duration and period)

General programming information

Warning!

As soon as you begin editing a program online (i.e., in RUN mode) using this instruction, pulse output will stop.

- Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- The high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) are assigned to the same internal relay (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- At a point close to the minimum or maximum duty ratio, the output is delayed, which may cause the duty ratio to differ from the specified value.
- The duty ratio can be changed for each scan. The change becomes effective with the next pulse output. The frequency setting is only effective at the start of execution of an instruction.
- We strongly recommend that you incorporate a forced stop (see page 1021) option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

■ Using the FP0 compatibility mode of the FP0R

To run the FP0R in FP0 compatibility mode, you can download an FP0 program to the FP0R.

PLC types **Availability of F170_PulseOutput_PWM (see page 1322)**

Data types

Variable	Data type	Function
s_dutDataTable	F170_PulseOutput_PWM_DUT	Starting address of area containing the data table
n_iPulseOutputChannel	INT	Pulse output channel:: 0 or 1

Operands

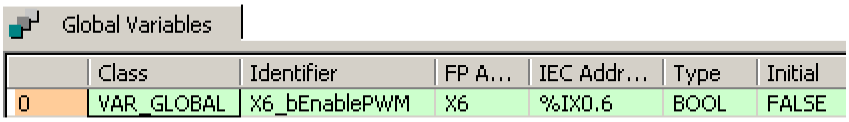
For	Relay				T/C		Register			Constant
s_dutDataTable	-	-	-	-	-	-	DT	-	-	-
n_iPulseOutputChannel	-	-	-	-	-	-	-	-	-	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ channel number or values of the data table are outside the permissible range
R9008	%MX0.900.8	for an instant	

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).

GVL In the global variable list, you define variables that can be accessed by all POUs in the project.

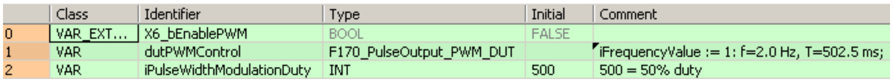


Global Variables

	Class	Identifier	FP A...	IEC Addr...	Type	Initial
0	VAR_GLOBAL	X6_bEnablePWM	X6	%IX0.6	BOOL	FALSE

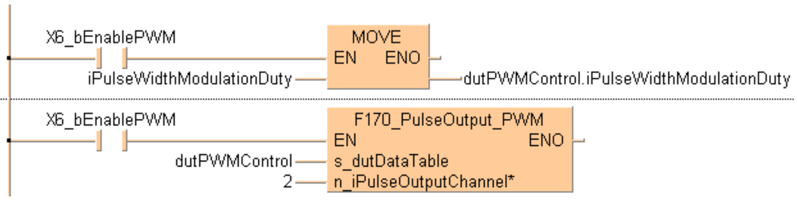
DUT The DUT F170_PulseOutput_PWM_DUT is predefined in the FP Library.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR_EXT...	X6_bEnablePWM	BOOL	FALSE	
1	VAR	dutPWMControl	F170_PulseOutput_PWM_DUT		iFrequencyValue := 1: f=2.0 Hz, T=502.5 ms;
2	VAR	iPulseWidthModulationDuty	INT	500	500 = 50% duty

LD



ST

When programming with structured text, enter the following:

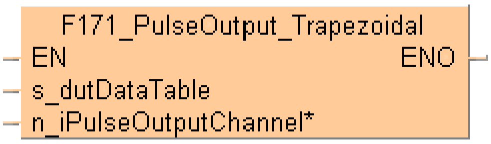
```
IF (X6_bEnablePWM) THEN
      dutPWMControl.iPulseWidthModulationDuty:=iPulseWidthModulationDuty;
END_IF;
IF (X6_bEnablePWM) THEN
      F170_PulseOutput_PWM(s_dutDataTable := dutPWMControl,
    n_iPulseOutputChannel := 2);
END_IF;
```

F171_PulseOutput_ Trapezoidal

Trapezoidal control

Description

This instruction automatically performs trapezoidal control according to the parameters in the specified DUT. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



See also: PulseOutput_Trapezoidal_FB (see page 1194)

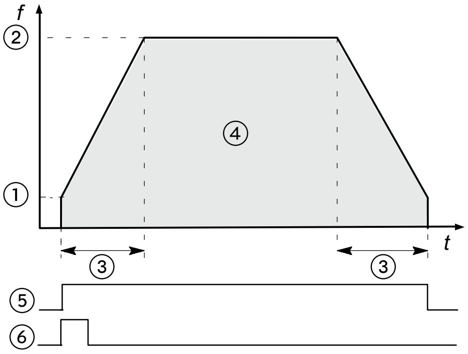
■ Description for FP-Sigma, FP-X (for the FP0R, please see on page 1047)

Use the following predefined DUT: F171_PulseOutput_Trapezoidal_DUT.

The following parameters can be specified in the DUT:

- Control code
- Initial and final speed
- Target speed
- Acceleration/deceleration time
- Target value
- Pulse stop

Pulse output characteristics



①	Initial and final speed	④	Target value
②	Target speed	⑤	Pulse output control flag
③	Acceleration/deceleration time	⑥	Execution condition

The pulse output frequency changes according to the specified acceleration/deceleration time.

The difference between target and initial speed determines the slope of the ramps.

General programming information

- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.

- FP-X: When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. sys_bIsPulseChannel0Active) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: The high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) are assigned to the same internal relay (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: Executing the circular interpolation control instruction **F176** sets the circular interpolation control flag (sys_bIsCircularInterpolationActive) to TRUE. The status of this flag is maintained until the target value is reached (even if the execution condition is no longer TRUE). During this time, other pulse output instructions cannot be executed.
- When programs are being edited in RUN mode, pulse output stops but resumes after the program changes have been downloaded.
- FPΣ: Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- FP-X: Set "Pulse output" for the desired channel in the system registers.
- We strongly recommend that you incorporate a forced stop (see page 1021) option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

◆REFERENCE

Please refer to the FPWIN Pro online help for detailed information on using system variables.

- Output relays and system variables for FP-Sigma
- Output relays and system variables for FP-X relay types
- Output relays and system variables for FP-X transistor types

■ Description for FP0R

Use the following predefined DUT:
F171_PulseOutput_Trapezoidal_Type0_DUT (maximum speed = first target speed) or
F171_PulseOutput_Trapezoidal_Type1_DUT (maximum speed = 50kHz).

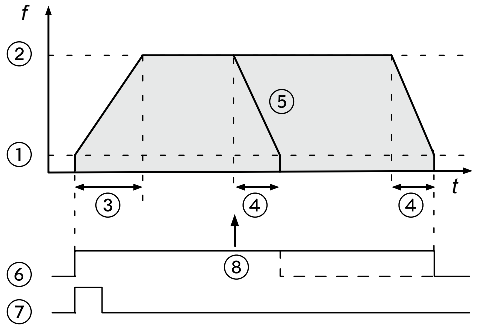
The target speed can be changed during pulse output. Two control methods are available:

- Type 0: The speed can be changed within the range of the target speed specified first.
- Type 1: The speed can be changed within the range of the maximum speed (50kHz).

The following parameters can be specified in the DUT:

- Control code
- Initial and final speed
- Target speed
- Acceleration time
- Deceleration time
- Target value

Pulse output characteristics



①	Initial and final speed	⑤	Target value
②	Target speed	⑥	Pulse output control flag
③	Acceleration time	⑦	Execution condition
④	Deceleration time	⑧	Decelerated stop request

Type 0: The difference between target speed and initial speed determines the slope of the acceleration ramp. The difference between target speed and final speed determines the slope of the deceleration ramp.

Type 1: The difference between the maximum speed of 50kHz and the initial speed determines the slope of the acceleration ramp. The difference between the maximum speed of 50kHz and the final speed determines the slope of the deceleration ramp.

Pulses are output using a duty of 25%.

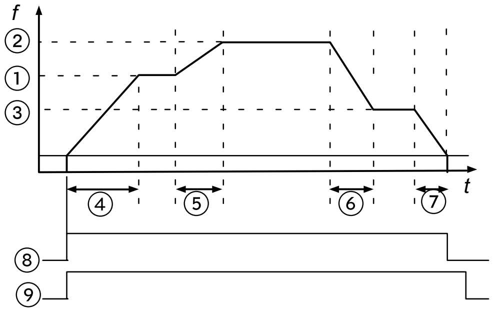
With the pulse output method "pulse/direction", pulses are output approx. 300µs after the direction signal has been output; the motor driver characteristics are simultaneously taken into consideration.

Decelerated stop

To perform a decelerated stop, set bit 5 of the data register storing the pulse output control code from FALSE to TRUE (e.g. `MOVE(16#120, sys_wHscOrPulseControlCode);`).

When a decelerated stop is requested during acceleration, deceleration is performed with the same slope as deceleration from the target speed.

Changing the target speed during pulse output



Type 1: The speed can be changed within the range of the maximum speed (50kHz).

①	Target speed	⑥	Deceleration
②	1st change of target speed	⑦	Deceleration time
③	2nd change of target speed	⑧	Pulse output control flag
④	Acceleration time	⑨	Execution condition
⑤	Acceleration		

To change the speed, keep the execution condition TRUE.

Type 0: If a value larger than the target speed at start-up is specified, it will be corrected to the target speed at start-up.
Type 1: If the target speed is set to a value larger than 50kHz, it will be corrected to 50kHz.

If the elapsed value crosses over the acceleration forbidden area starting position (e.g. sys_diPulseChannel0AccelerationForbiddenAreaStartingPosition) during acceleration, acceleration cannot be performed.

The deceleration speed cannot be lower than the corrected final speed.

General programming information

Warning!

As soon as you begin editing a program online (i.e., in RUN mode) using this instruction, pulse output will stop.

- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. sys_bIsPulseChannel0Active) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- The instruction cannot be started when a decelerated stop has been requested.
- To restart after stopping the operation, turn the execution condition to FALSE and then to TRUE again.
- The execution of the instruction is faster the second time it is started if the positioning parameters remain unchanged. Changing the setting of the output operation (pulse output or calculation only) does not effect this behavior.
- We strongly recommend that you incorporate a forced stop (see page 1021) option in your positioning program.

- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

◆REFERENCE

Please refer to the FPWIN Pro online help for detailed information on using system variables.

Output relays and system variables for FP0R

PLC types **Availability of F171_PulseOutput_Trapezoidal (see page 1322)**

Data types

Variable	Data type	Function
s_dutDataTable	**FP-Σ, FP-X:** F171_PulseOutput_Trapezoidal_DUT **FP0R:** F171_PulseOutput_Trapezoidal_Type0_DUT F171_PulseOutput_Trapezoidal_Type1_DUT	Starting address of area containing the data table
n_iPulseOutputChannel	decimal constant	Pulse output channel: FP-Σ: 0, 2 FP-X R: 0, 1 FP-X C14T: 0, 1, 2 FP-X C30T/C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3

Operands

For	Relay				T/C		Register			Constant
s_dutDataTable	-	-	-	-	-	-	DT	-	-	-
n_iPulseOutputChannel	-	-	-	-	-	-	-	-	-	dec. or hex.

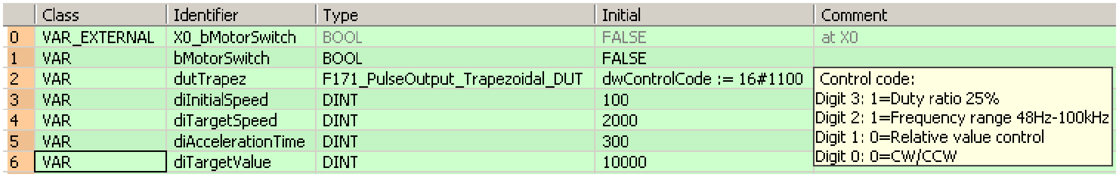
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ channel number or values of the data table are outside the permissible range ▪ initial speed > target speed ▪ FP0R/FP-X:pulse output has not been set in the system registers
R9008	%MX0.900.8	for an instant	

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

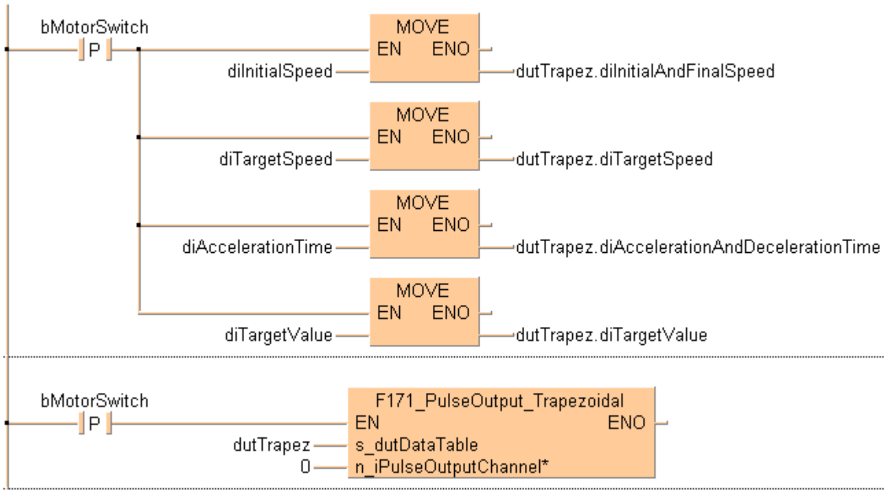
DUT The DUT F171_PulseOutput_Trapezoidal_DUT is predefined in the FP Library.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	X0_bMotorSwitch	BOOL	FALSE	at X0
1	VAR	bMotorSwitch	BOOL	FALSE	
2	VAR	dutTrapez	F171_PulseOutput_Trapezoidal_DUT	dwControlCode := 16#1100	Control code: Digit 3: 1=Duty ratio 25% Digit 2: 1=Frequency range 48Hz-100kHz Digit 1: 0=Relative value control Digit 0: 0=CW/CCW
3	VAR	diInitialSpeed	DINT	100	
4	VAR	diTargetSpeed	DINT	2000	
5	VAR	diAccelerationTime	DINT	300	
6	VAR	diTargetValue	DINT	10000	

LD



ST When programming with structured text, enter the following:

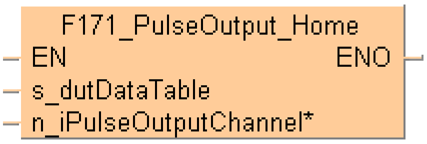
```
IF DF(bMotorSwitch) THEN
  dutTrapez.diInitialAndFinalSpeed:=diInitialSpeed;
  dutTrapez.diTargetSpeed:=diTargetSpeed;
  dutTrapez.diAccelerationDecelerationTime:=diAccelerationTime;
  dutTrapez.diDeviationCounterClearSignalOutputTime:=10;
END_IF;

IF DF(bMotorSwitch) THEN
  F171_PulseOutput_Trapezoidal(s_dutDataTable := dutTrapez,
        n_iPulseOutputChannel := 0);
END_IF;
```

F171_PulseOutput_ Home

Home return

Description

This instruction performs a home return according to the parameters in the specified DUT. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



```
F171_PulseOutput_Home
EN                      ENO
s_dutDataTable
n_iPulseOutputChannel*
```

See also:

- PulseOutput_Home_FB (see page 1176)
- PulseControl_NearHome (see page 1209)

After a drive system has been switched on, there is a difference between the internal position value (elapsed value) and the mechanical position of the axis; this difference cannot be predetermined. The internal value must be synchronized with the actual position value of the axis. This is done by means of a home return, during which a position value is registered at a known reference point (home).

During execution of a home return instruction, pulses are continuously output until the home input is enabled. The I/O allocation is determined by the channel used.

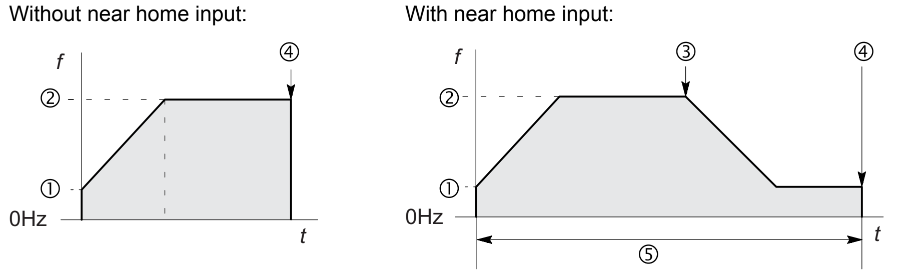
To decelerate movement when near the home position, designate a near home input and set bit 4 of the special data register storing the pulse output control code (sys_wHscOrPulseControlCode) to TRUE and back to FALSE again.

The deviation counter clear output can be set to TRUE when home return has been completed.

The value in the elapsed value area during a home return differs from the current value. When the return is completed, the elapsed value changes to 0.

Select one of two different operation modes:

- Type 1: The home input is effective regardless of whether or not there is a near home input, whether deceleration is taking place, or whether deceleration has been completed.

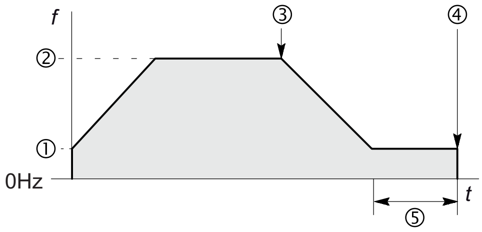


Without near home input:

With near home input:

①	Initial and final speed	③	Near home input: TRUE
②	Target speed	④	Home input: TRUE
⑤	Home input is effective at any time.		

- Type 2: The home input is effective only after deceleration (started by near home input) has been completed.



①	Initial and final speed	③	Near home input: TRUE
②	Target speed	④	Home input: TRUE
⑤	Home input is effective only after deceleration		

Use the following predefined DUT: F171_PulseOutput_Home_DUT

The following parameters can be specified in the DUT:

- Control code
- Initial and final speed
- Target speed
- Acceleration/deceleration time
- Deviation counter clear signal

Pulse output characteristics

- The pulse output frequency changes according to the specified acceleration/deceleration time.
- The difference between target and initial speed determines the slope of the ramps.

General programming information

- Even when home input has occurred, executing this instruction causes pulse output to begin.
- If the near home input is enabled while acceleration is in progress, deceleration will start.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- The deviation counter clear signal is allocated to dedicated output numbers specific to each PLC type.
- When programs are being edited in RUN mode, pulse output stops but resumes after the program changes have been downloaded.
- FP-X: When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. sys_bIsPulseChannel0Active) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: The high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) are assigned to the same internal relay (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.

- FPΣ: Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- FP-X: Set "Pulse output" for the desired channel in the system registers.
- FPΣ: Executing the circular interpolation control instruction **F176** sets the circular interpolation control flag (sys_bIsCircularInterpolationActive) to TRUE. The status of this flag is maintained until the target value is reached (even if the execution condition is no longer TRUE). During this time, other pulse output instructions cannot be executed.
- We strongly recommend that you incorporate a forced stop (see page 1021) option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

PLC types **Availability of F171_PulseOutput_Home (see page 1322)**

Data types

Variable	Data type	Function
s_dutDataTable	F171_PulseOutput_Home_DUT	Starting address of area containing the data table
n_iPulseOutputChannel	decimal constant	Pulse output channel: FP-Σ: 0, 2 FP-X R: 0, 1 FP-X C14T: 0, 1, 2 FP-X C30T/C60T: 0, 1, 2, 3

Operands

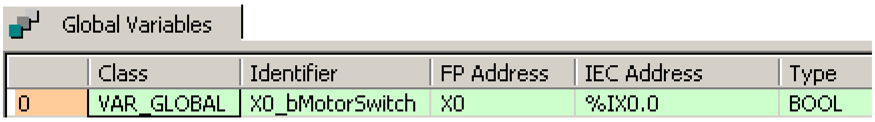
For	Relay				T/C		Register			Constant
s_dutDataTable	-	-	-	-	-	-	DT	-	-	-
n_iPulseOutputChannel	-	-	-	-	-	-	-	-	-	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ channel number or values of the data table are outside the permissible range ▪ initial speed > target speed ▪ FP-X: pulse output has not been set in the system registers
R9008	%MX0.900.8	for an instant	

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

GVL In the global variable list, you define variables that can be accessed by all POUs in the project.

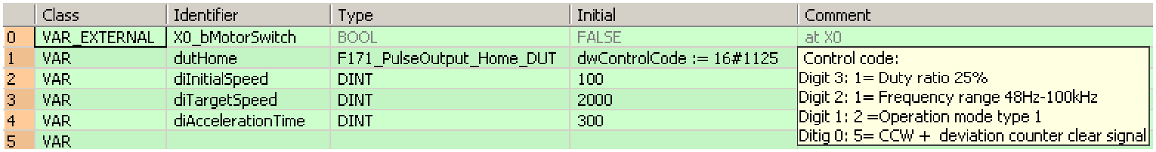


Global Variables

	Class	Identifier	FP Address	IEC Address	Type
0	VAR_GLOBAL	X0_bMotorSwitch	X0	%IX0.0	BOOL

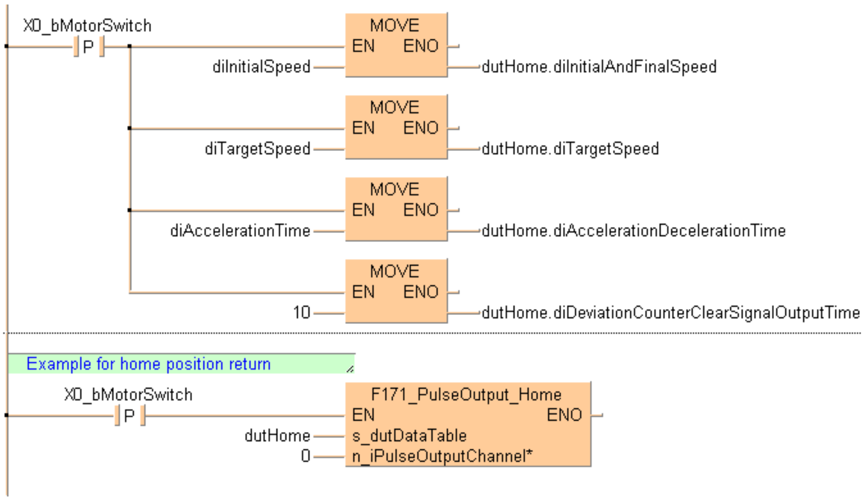
DUT The DUT F171_PulseOutput_Home_DUT is predefined in the FP Library.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	X0_bMotorSwitch	BOOL	FALSE	at X0
1	VAR	dutHome	F171_PulseOutput_Home_DUT	dwControlCode := 16#1125	Control code: Digit 3: 1= Duty ratio 25% Digit 2: 1= Frequency range 48Hz-100kHz Digit 1: 2 =Operation mode type 1 Ditig 0: 5= CCW + deviation counter clear signal
2	VAR	diInitialSpeed	DINT	100	
3	VAR	diTargetSpeed	DINT	2000	
4	VAR	diAccelerationTime	DINT	300	
5	VAR				

LD



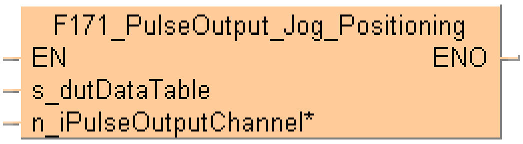
ST When programming with structured text, enter the following:

```
IF DF(X0_bMotorSwitch) THEN
      dutHome.diInitialAndFinalSpeed:=diInitialSpeed;
      dutHome.diTargetSpeed:=diTargetSpeed;
      dutHome.diAccelerationDecelerationTime:=diAccelerationTime;
      dutHome.diDeviationCounterClearSignalOutputTime:=10;
END_IF;
(*Example for home position return*)
IF DF(X0_bMotorSwitch) THEN
    F171_PulseOutput_Home(s_dutDataTable := dutHome,
      n_iPulseOutputChannel := 0);
END_IF;
```

F171_PulseOutput_ Jog_Positioning

JOG operation and positioning

Description The specified number of pulses is output after the position control trigger input has turned to TRUE. A deceleration is performed before the target value is reached and pulse output stops. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



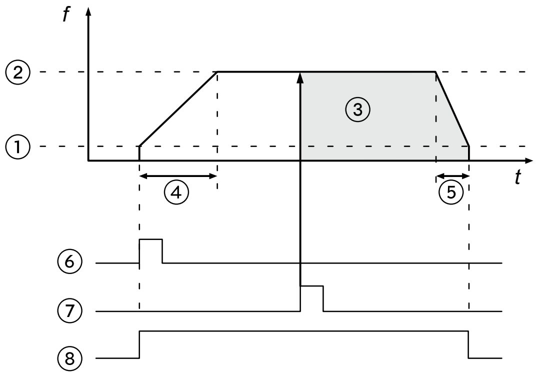
See also:

- PulseOutput_Jog_Positioning0_FB (see page 1181)
- PulseOutput_Jog_Positioning1_FB (see page 1184)
- PulseControl_JogPositionControl (see page 1208)

Select one of two different operation modes:

Type 0: The speed can be changed within the range of the specified target speed.
Type 1: The target speed can be changed once when the position control trigger input turns to TRUE.

Pulse output characteristics



①	Initial and final speed	⑤	Deceleration time
②	Target speed	⑥	Execution condition
③	Target value	⑦	Position control trigger input
④	Acceleration time	⑧	Pulse output control flag

- The pulse output frequency changes according to the specified acceleration time and the specified deceleration time.
- The difference between target speed and initial speed determines the slope of the acceleration ramp.
- The difference between target speed and final speed determines the slope of the deceleration ramp.
- After the position control trigger input has turned to TRUE, pulse output continues, then decelerates and stops when the target value is reached.

Stopping pulse output

Pulse output can be stopped by one of the following operations:

- Turning the position control trigger to TRUE (pulse output continues until the target value has been reached and deceleration has completed): The position control trigger can be started by turning a position control trigger input to TRUE or by setting bit 6 of the data register storing the pulse output control code from FALSE to TRUE (e.g. `MOVE(16#140, sys_wHscOrPulseControlCode);`).
- Requesting a decelerated stop: To perform a decelerated stop, set bit 5 of the data register storing the pulse output control code from FALSE to TRUE (e.g. `MOVE(16#120, sys_wHscOrPulseControlCode);`). When a decelerated stop is requested during acceleration, deceleration is performed with the same slope as deceleration from the target speed.
- Executing an emergency stop: To perform an emergency stop, set bit 3 of the data register storing the pulse output control code from FALSE to TRUE (e.g. `MOVE(16#108, sys_wHscOrPulseControlCode);`).

Note: When stopping, disable all pulse output functions for the channel used in the program.

■ JOG Operation Type 0

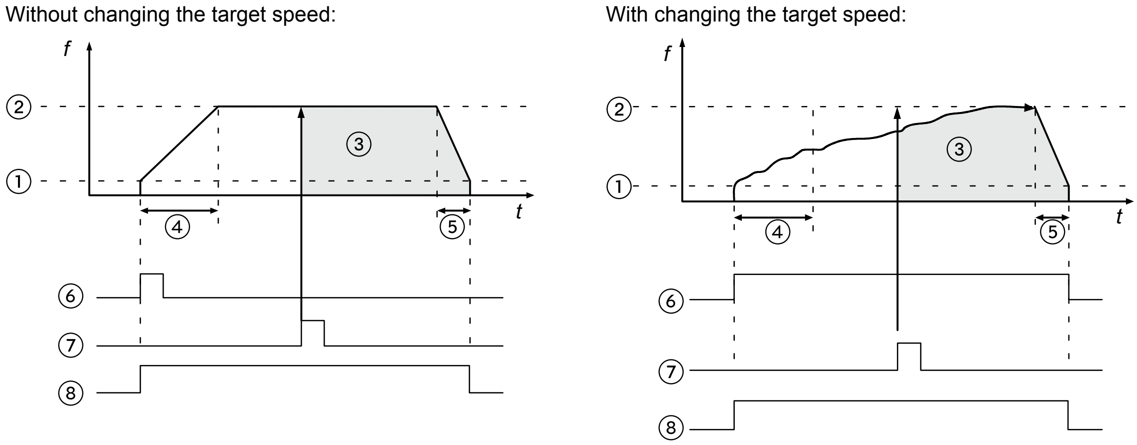
Use the following predefined DUT:
F171_PulseOutput_Jog_Positioning_Type0_DUT

The following parameters can be specified in the DUT:

- Control code
- Initial and final speed
- Target speed
- Acceleration time
- Deceleration time
- Target value

The target speed can be changed during pulse output.

Changing the target speed during pulse output


Without changing the target speed:
With changing the target speed:
f
t
①
②
③
④
⑤
⑥
⑦
⑧

①	Initial and final speed	⑤	Deceleration time
②	Target speed	⑥	Execution condition
③	Target value	⑦	Position control trigger input
④	Acceleration time	⑧	Pulse output control flag

- To change the speed, keep the execution condition TRUE.
- If the target speed is set to a value larger than 50kHz, it will be corrected to 50kHz.
- If the elapsed value crosses over the acceleration forbidden area starting position (e.g. sys_diPulseChannel0AccelerationForbiddenAreaStartingPosition) during acceleration, acceleration cannot be performed.
- The deceleration speed cannot be lower than the corrected final speed.
- Changing the target speed is not possible if the instruction is executed in an interrupt program.

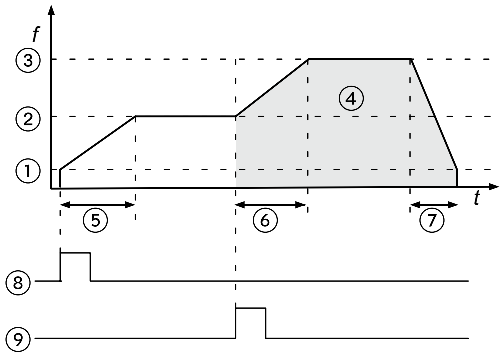
JOG Operation Type 1

Use the following predefined DUT:
F171_PulseOutput_Jog_Positioning_Type1_DUT

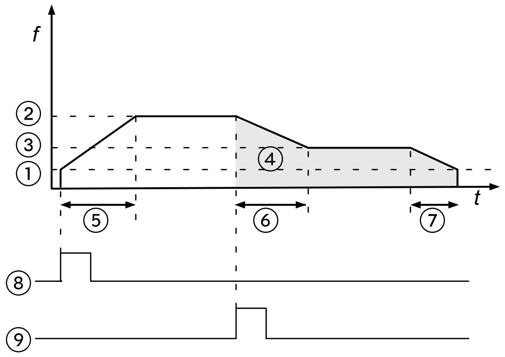
The following parameters can be specified in the DUT:

- Control code
- Initial and final speed
- Target speed 1
- Acceleration time
- Target speed 2
- Change time
- Deceleration time
- Target value

Target speed 1 < target speed 2:



Target speed 1 > target speed 2:



①	Initial and final speed	⑥	Change time
②	Target speed 1	⑦	Deceleration time
③	Target speed 2	⑧	Execution condition
④	Target value	⑨	Position control trigger input
⑤	Acceleration time		

After the position control trigger input has turned to TRUE, the pulse output frequency will change using the change time to accelerate or decelerate to target speed 2. Further target speed changes are not possible. The position control trigger input will be disregarded if it is turned on during acceleration.

General programming information

Warning!

As soon as you begin editing a program online (i.e., in RUN mode) using this instruction, pulse output will stop.

- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. sys_bIsPulseChannel0Active) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- Set the position control trigger input (X0, X1, X2, X3) in system register 402.
- For the position control trigger input, only the rising edge (TRUE) is detected.
- The instruction cannot be started when a decelerated stop has been requested.
- To restart after stopping the operation, turn the execution condition to FALSE and then to TRUE again.
- We strongly recommend that you incorporate a forced stop (see page 1021) option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

PLC types **Availability of F171_PulseOutput_Jog_Positioning (see page 1322)**

Data types

Variable	Data type	Function
s_DUT_DataTable	F171_PulseOutput_Jog_Positioning_Type0_DUT or F171_PulseOutput_Jog_Positioning_Type1_DUT	Starting address of area containing the data table
n_iPulseOutputChannel	decimal constant	Pulse output channel: 0–3

Operands

For	Relay				T/C		Register			Constant
s_dutDataTable	-	-	-	-	-	-	DT	-	-	-
n_iPulseOutputChannel	-	-	-	-	-	-	-	-	-	dec. or hex.

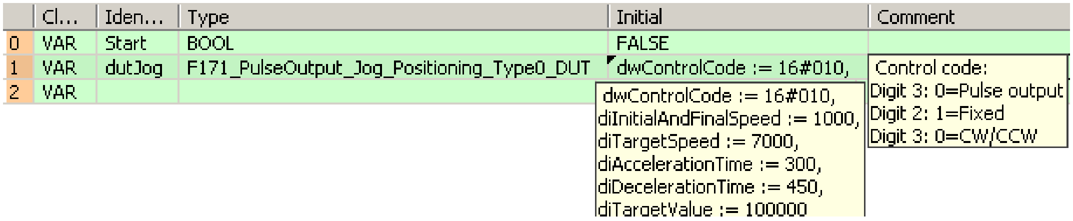
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ the same channel is started twice ▪ channel number or values of the data table are outside the permissible range ▪ initial speed > target speed ▪ pulse output has not been set in the system registers
R9008	%MX0.900.8	for an instant	

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

DUT The DUT F171_PulseOutput_Jog_Positioning_Type0_DUT is predefined in the FP Library.

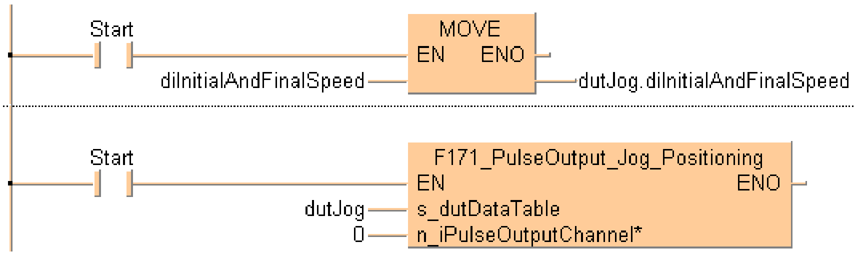
POU header All input and output variables used for programming this function have been declared in the POU header.



	Cl...	Iden...	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	dutJog	F171_PulseOutput_Jog_Positioning_Type0_DUT	dwControlCode := 16#010,	Control code: Digit 3: 0=Pulse output Digit 2: 1=Fixed Digit 3: 0=CW/CCW
2	VAR				

```
dwControlCode := 16#010,
diInitialAndFinalSpeed := 1000,
diTargetSpeed := 7000,
diAccelerationTime := 300,
diDecelerationTime := 450,
diTargetValue := 100000
```

LD



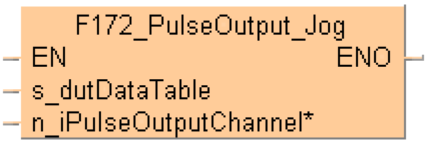
ST When programming with structured text, enter the following:

```
IF (Start) THEN
      dutJog.diInitialAndFinalSpeed:=diInitialAndFinalSpeed;
END_IF;
IF (Start) THEN
  F171_PulseOutput_Jog_Positioning(s_dutDataTable := dutJog, 0);
END_IF;
```

F172_PulseOutput_Jog

JOG operation

Description This instruction is used for JOG operation. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



See also:

- PulseOutput_Jog_FB (see page 1179)
- PulseOutput_Jog_TargetValue_FB (see page 1186)

■ Description for FP-Sigma, FP-X

Use the following predefined DUT:
F172_PulseOutput_Jog_Type0_DUT_0 (Mode with no target value) or
F172_PulseOutput_Jog_Type1_DUT_0 (Target value match stop mode)

The following parameters can be specified in the DUT:

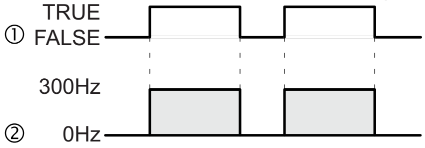
- Control code
- Frequency
- Target value

Pulse output characteristics

The frequency and the target value can be changed in each scan. The control code, however, cannot be changed during execution of the instruction.

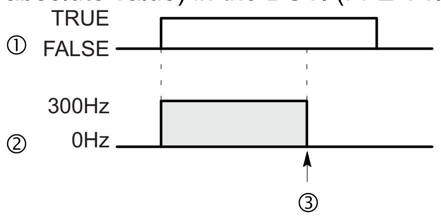
Select one of two different operation modes:

- Mode with no target value (type 0): Pulses are output in accordance with the conditions set in the DUT as long as the execution condition is TRUE.



①	Execution condition
②	CW pulse output

- Target value match stop mode (type 1): Output stops when the target value is reached. Set this mode in the control code, and specify the target value (an absolute value) in the DUT. (FPΣ V1.4 or higher, FP-X)



①	Execution condition
②	CW pulse output
③	Target value reached (pulse output stops)

General programming information

Warning!

As soon as you begin editing a program online (i.e., in RUN mode) using this instruction, pulse output will stop.

- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- FP-X: When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. sys_bIsPulseChannel0Active) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: The high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) are assigned to the same internal relay (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: Executing the circular interpolation control instruction **F176** sets the circular interpolation control flag (sys_bIsCircularInterpolationActive) to TRUE. The status of this flag is maintained until the target value is reached (even if the execution condition is no longer TRUE). During this time, other pulse output instructions cannot be executed.
- FPΣ: Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- FP-X: Set "Pulse output" for the desired channel in the system registers.
- If the execution of the instruction is started with an invalid frequency value, an operation error occurs. If the frequency is changed to an invalid value during execution of the instruction, the frequency output will be adjusted to either the minimum or the maximum value of the permissible range.
- Changing the control code during execution of the instruction will have no effect.
- We strongly recommend that you incorporate a forced stop (see page 1021) option in your positioning program.

- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

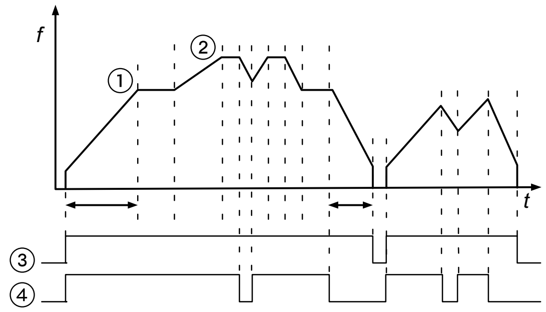
Description for FP0R

Use the following predefined DUT:
F172_PulseOutput_Jog_Type0_DUT_1 (Mode with no target value) or
F172_PulseOutput_Jog_Type1_DUT_1 (Target value match stop mode)

The following parameters can be specified in the DUT:

- Control code
- Initial and final speed
- Target speed
- Acceleration time
- Deceleration time
- Target value

Pulse output characteristics



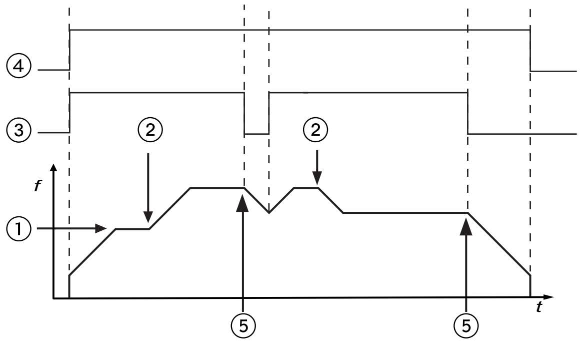
①	Target speed 1	③	Pulse output control flag
②	Target speed 2	④	Execution condition

- The pulse output frequency changes according to the specified acceleration time and the specified deceleration time.
- The difference between the maximum speed of 50kHz and the initial speed determines the slope of the acceleration ramp.
- The difference between the maximum speed of 50kHz and the final speed determines the slope of the deceleration ramp.
- When the execution condition turns to FALSE after starting the instruction, a decelerated stop is performed.
- When the execution condition turns to TRUE during deceleration, acceleration is performed again.
- The target speed can be changed during pulse output.
- Pulses are output using a duty of 25%.
- With the pulse output method "pulse/direction", pulses are output approx. 300μs after the direction signal has been output; the motor driver characteristics are simultaneously taken into consideration.
- When a decelerated stop is requested during acceleration, deceleration is performed with the same slope as deceleration from the target speed.
- Acceleration time and deceleration time have priority over initial speed and final

speed. This means that the values for acceleration time and deceleration time will not be changed whereas the values for initial speed and final speed may be corrected by the pulse output instruction to enable acceleration and deceleration within the specified time. The modified values are written to data registers which can be accessed using the system variables sys_iPulseChannelxCorrectedInitialSpeed and sys_iPulseChannelxCorrectedFinalSpeed (where x=channel number).

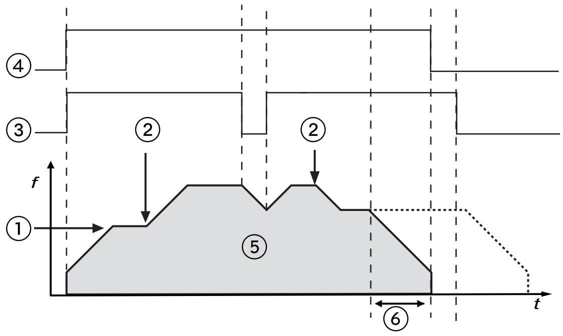
Select one of two different operation modes:

- Mode with no target value (type 0): Pulses are output in accordance with the conditions set in the DUT as long as the execution condition is TRUE. A decelerated stop begins whenever the execution condition is FALSE.



①	Initial and final speed	④	Pulse output control flag
②	Change of target speed	⑤	Decelerated stop
③	Execution condition		

- Target value match stop mode (type 1): Output stops when the target value is reached. Set this mode in the control code, and specify the target value (an absolute value) in the DUT. A decelerated stop is performed when the target value has been reached. Deceleration is performed within the specified deceleration time.



①	Initial and final speed	④	Pulse output control flag
②	Change of target speed	⑤	Target value
③	Execution condition	⑥	Deceleration time

Changing the target speed during pulse output

- If the elapsed value crosses over the acceleration forbidden area starting position (e.g. sys_diPulseChannel0AccelerationForbiddenAreaStartingPosition) during acceleration, acceleration cannot be performed.
- The deceleration speed cannot be lower than the corrected final speed.

General programming information

Warning!

As soon as you begin editing a program online (i.e., in RUN mode) using this instruction, pulse output will stop.

- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. sys_bIsPulseChannel0Active) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- Changing the control code during execution of the instruction will have no effect.
- We strongly recommend that you incorporate a forced stop (see page 1021) option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

REFERENCE

Please refer to the FPWIN Pro online help for detailed information on using system variables.

Output relays and system variables for FP0R

PLC types **Availability of F172_PulseOutput_Jog (see page 1322)**

Data types

Variable	Data type	Function
s_dutDataTable	**FP-Σ, FP-X:** F172_PulseOutput_Jog_Type0_DUT_0 F172_PulseOutput_Jog_Type1_DUT_0 **FP0R:** F172_PulseOutput_Jog_Type0_DUT_1 F172_PulseOutput_Jog_Type1_DUT_1	Starting address of area containing the data table
n_iPulseOutputChannel	decimal constant	Pulse output channel: FP-Σ: 0, 2 FP-X R: 0, 1 FP-X C14T: 0, 1, 2 FP-X C30T/C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3

Operands

For	Relay				T/C		Register			Constant
s_dutDataTable	-	-	-	-	-	-	DT	-	-	-
n_iPulseOutputChannel	-	-	-	-	-	-	-	-	-	dec. or hex.

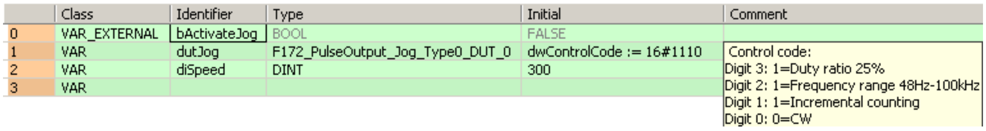
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ channel number or values of the data table are outside the permissible range ▪ FP0R/FP-X: pulse output has not been set in the system registers
R9008	%MX0.900.8	for an instant	

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

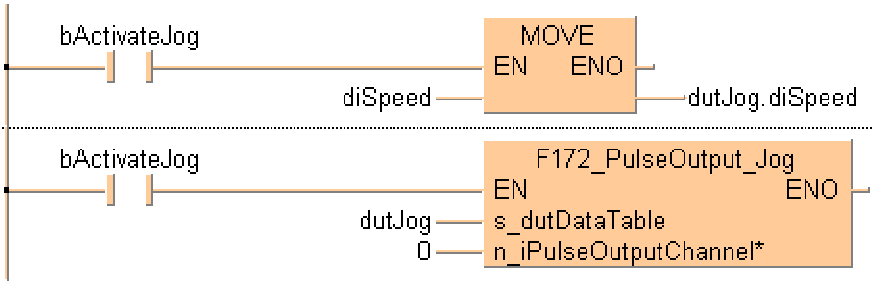
DUT The DUT F172_PulseOutput_Jog_Type0_DUT_0 is predefined in the FP Library.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	bActivateJog	BOOL	FALSE	
1	VAR	dutJog	F172_PulseOutput_Jog_Type0_DUT_0	dwControlCode := 16#1110	Control code: Digit 3: 1=Duty ratio 25% Digit 2: 1=Frequency range 48Hz-100kHz Digit 1: 1=Incremental counting Digit 0: 0=CW
2	VAR	diSpeed	DINT	300	
3	VAR				

LD



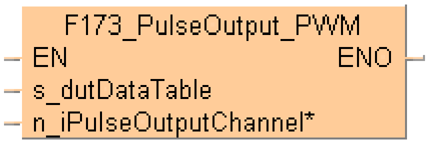
ST When programming with structured text, enter the following:

```
IF (bActivateJog) THEN
  dutJog.diSpeed:=diSpeed;
END_IF;
IF (bActivateJog) THEN
  F172_PulseOutput_Jog(s_dutDataTable := dutJog, 0);
END_IF;
```

F173_PulseOutput_PWM

PWM output

Description This instruction delivers a pulse width modulated output signal according to the specified DUT. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



Use the following predefined DUT: F173_PulseOutput_PWM_DUT

The following parameters can be specified in the DUT:

- Approximate frequency
- Duty ratio (for pulse duration and period)

General programming information

Warning!

As soon as you begin editing a program online (i.e., in RUN mode) using this instruction, pulse output will stop.

- The duty ratio, particularly when it is close to the minimum or maximum value, may differ from the specified duty ratio, depending on the load voltage and the load current.
- The duty ratio can be changed for each scan.
- The frequency constant K cannot be changed during execution of the instruction. If it is changed, it will have no effect on the frequency but on the resolution of the duty ratio.
- If the duty ratio is changed to a value outside the permissible range while the instruction is being executed, the duty ratio is adjusted to the maximum value. When execution of the instruction begins, an operation error is displayed.
- If the frequency is changed to a value outside the permissible range while the instruction is being executed, the resolution is adjusted to 100. When execution of the instruction begins, no operation error is displayed.
- If the duty is changed to 100% or higher while the instruction is being executed, the frequency is adjusted to the maximum value at the specified resolution. When execution of the instruction begins, no operation error is displayed.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- FP-X, FP0R: When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. sys_bIsPulseChannel0Active) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: The high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) are assigned to the same internal relay (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output

instruction is executed, both the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.

- FPΣ: Executing the circular interpolation control instruction **F176** sets the circular interpolation control flag (sys_bIsCircularInterpolationActive) to TRUE. The status of this flag is maintained until the target value is reached (even if the execution condition is no longer TRUE). During this time, other pulse output instructions cannot be executed.
- FPΣ: Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- FP-X, FP0R: Set "PWM output" for the desired channel in the system registers.
- We strongly recommend that you incorporate a forced stop (see page 1021) option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

PLC types **Availability of F173_PulseOutput_PWM (see page 1322)**

Data types

Variable	Data type	Function
s_dutDataTable	F173_PulseOutput_PWM_DUT	Starting address of area containing the data table
n_iPulseOutputChannel	decimal constant	Pulse output channel: FP-Σ: 0, 2 FP-X R: 0, 1 FP-X C14T: 0, 1, 2 FP-X C30T/C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3

Operands

For	Relay				T/C		Register			Constant
s_dutDataTable	-	-	-	-	-	-	DT	-	-	-
n_iPulseOutputChannel	-	-	-	-	-	-	-	-	-	dec. or hex.

Error flags

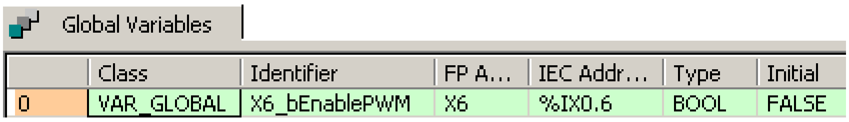
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ channel number or values of the data table are outside the permissible range (when executing the instruction for the first time) ▪ FP0R/FP-X: pulse output has not been set in the system registers
R9008	%MX0.900.8	for an instant	

Example

In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

GVL

In the global variable list, you define variables that can be accessed by all POUs in the project.



Global Variables

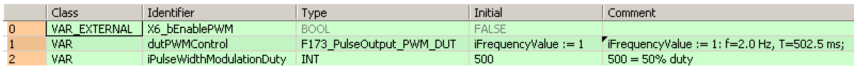
	Class	Identifier	FP A...	IEC Addr...	Type	Initial
0	VAR_GLOBAL	X6_bEnablePWM	X6	%IX0.6	BOOL	FALSE

DUT

The DUT F173_PulseOutput_PWM_DUT is predefined in the FP Library.

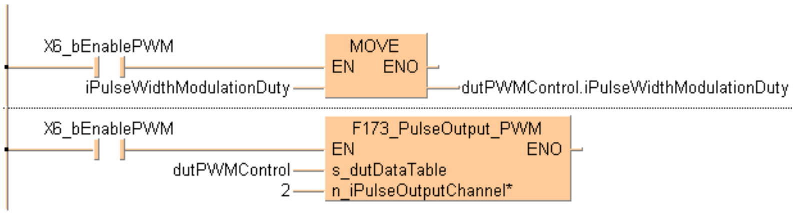
POU header

All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	X6_bEnablePWM	BOOL	FALSE	
1	VAR	dutPWMControl	F173_PulseOutput_PWM_DUT	iFrequencyValue := 1	iFrequencyValue := 1: f=2.0 Hz, T=502.5 ms;
2	VAR	iPulseWidthModulationDuty	INT	500	500 = 50% duty

LD



ST

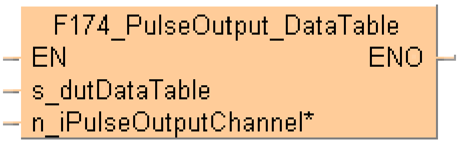
When programming with structured text, enter the following:

```
IF (X6_bEnablePWM) THEN
      dutPWMControl.iPulseWidthModulationDuty:=iPulseWidthModulationDuty;
END_IF;
IF (X6_bEnablePWM) THEN
      F173_PulseOutput_PWM(s_dutDataTable := dutPWMControl,
       n_iPulseOutputChannel := 2);
END_IF;
```

F174_PulseOutput_DataTable

Data table control

Description This instruction performs rectangular control according to the parameters in the specified DUT with an arbitrary number of different speeds and target values. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.

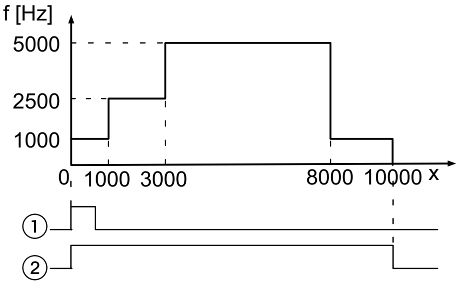


Create your own DUT using the following DUT as a sample: F174_PulseOutput_DataTable_8_Values_DUT

The following parameters can be specified in the DUT:

- Control code
- Frequency 1
- Target value 1
- Frequency 2
- Target value 2
- ...
- Frequency n
- Target value n
- Pulse stop

Pulse output characteristics



x	Elapsed value of high-speed counter (amount of travel)
①	Execution condition
②	Pulse output control flag

- Pulses are output at the specified frequency until the target value is reached. Then the frequency changes to the second frequency value and pulse output continues until the second target value is reached, and so forth.
- Pulse output stops when the last target value is reached.
- A frequency of 0 signifies the final frequency and stops pulse output.

General programming information

- FP-X, FP0R: When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. sys_bIsPulseChannel0Active) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.

- FPΣ: The high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) are assigned to the same internal relay (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- If the value of the first frequency specified is out of range, an operation error occurs. (If the value of the first frequency is 0, operation stops without any pulses having been output.)
- If the value of the second frequency specified is out of range or 0, pulse output stops.
- If the target value is out of range, the number of pulses output may be different from the specified value.
- FPΣ: Executing the circular interpolation control instruction **F176** sets the circular interpolation control flag (sys_bIsCircularInterpolationActive) to TRUE. The status of this flag is maintained until the target value is reached (even if the execution condition is no longer TRUE). During this time, other pulse output instructions cannot be executed.
- FPΣ: Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- FPΣ, FP-X: If a periodic interrupt or high-speed counter interrupt program is run, or the PLC link function is used at the same time, a frequency of 80kHz or less should be used.
- FP-X: Set "Pulse output" for the desired channel in the system registers.
- When programs are being edited in RUN mode, pulse output stops but resumes after the program changes have been downloaded.
- We strongly recommend that you incorporate a forced stop (see page 1021) option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

PLC types

Availability of F174_PulseOutput_DataTable (see page 1322)

Data types

Variable	Data type	Function
s_dutDataTable	ANY_DUT	Starting address of area containing the data table Sample: F174_PulseOutput_DataTable_8_Values_DUT
n_iPulseOutputChannel	decimal constant	Pulse output channel: FP-Σ: 0, 2 FP-X R: 0, 1 FP-X C14T: 0, 1, 2 FP-X C30T/C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3

Operands

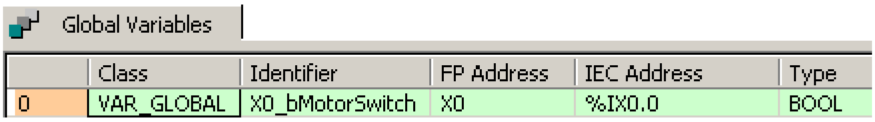
For	Relay				T/C		Register			Constant
s_dutDataTable	-	-	-	-	-	-	DT	-	-	-
n_iPulseOutputChannel	-	-	-	-	-	-	-	-	-	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ channel number or values of the data table are outside the permissible range ▪ frequency 1 is outside the permissible range ▪ FP0R/FP-X: pulse output has not been set in the system registers
R9008	%MX0.900.8	for an instant	

Example In this example the function is programmed in ladder diagram (LD). The same POU header is used for all programming languages.

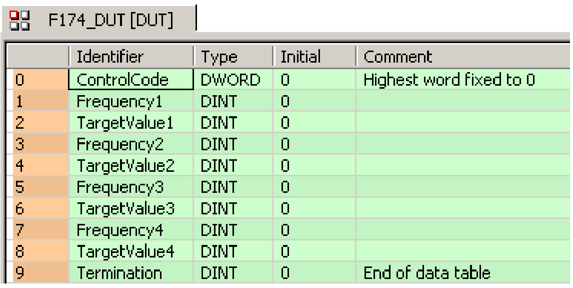
GVL In the global variable list, you define variables that can be accessed by all POUs in the project.



Global Variables

	Class	Identifier	FP Address	IEC Address	Type
0	VAR_GLOBAL	X0_bMotorSwitch	X0	%IX0.0	BOOL

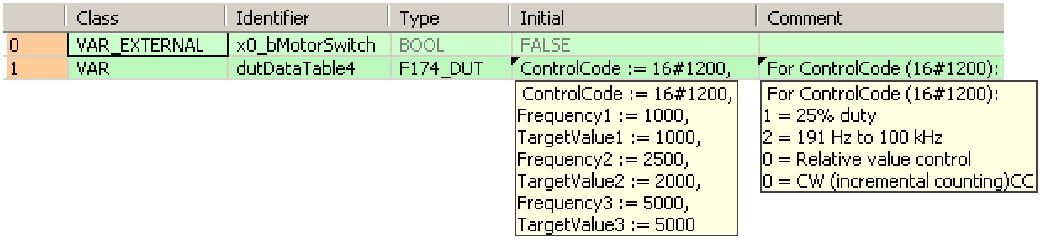
DUT The DUT F174_PulseOutput_DataTable_8_Values_DUT is predefined in the FP Library and can be used as a sample.



F174_DUT [DUT]

	Identifier	Type	Initial	Comment
0	ControlCode	DWORD	0	Highest word fixed to 0
1	Frequency1	DINT	0	
2	TargetValue1	DINT	0	
3	Frequency2	DINT	0	
4	TargetValue2	DINT	0	
5	Frequency3	DINT	0	
6	TargetValue3	DINT	0	
7	Frequency4	DINT	0	
8	TargetValue4	DINT	0	
9	Termination	DINT	0	End of data table

POU header All input and output variables used for programming this function have been declared in the POU header.

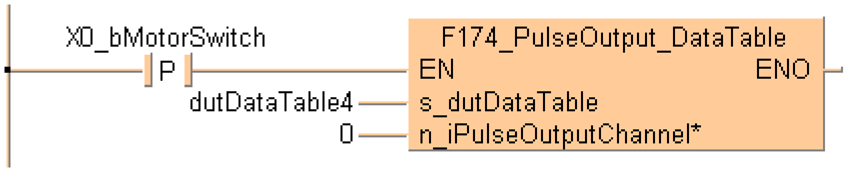


	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	x0_bMotorSwitch	BOOL	FALSE	
1	VAR	dutDataTable4	F174_DUT	ControlCode := 16#1200,	For ControlCode (16#1200):

```
ControlCode := 16#1200,
Frequency1 := 1000,
TargetValue1 := 1000,
Frequency2 := 2500,
TargetValue2 := 2000,
Frequency3 := 5000,
TargetValue3 := 5000
```

```
For ControlCode (16#1200):
1 = 25% duty
2 = 191 Hz to 100 kHz
0 = Relative value control
0 = CW (incremental counting)CC
```

LD



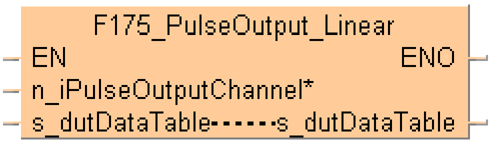
ST When programming with structured text, enter the following:

```
IF DF(X0_bMotorSwitch) THEN
  F174_PulseOutput_DataTable(s_dutDataTable := dutDataTable4, 4);

END_IF;
```

F175_PulseOutput_Linear

Linear interpolation

Description Pulses are output from two channels in accordance with the parameters in the specified DUT, so that the path to the target position forms a straight line. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



See also: PulseOutput_Linear_FB (see page 1189)

■ Description for FP-Sigma, FP-X (for the FP0R, please see on page 1074)

Use the following predefined DUT: F175_PulseOutput_Linear_DUT_0

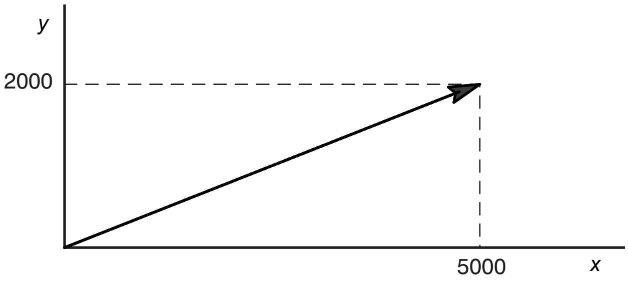
The following parameters can be specified in the DUT:

- Control code
- Initial and final speed
- Target speed
- Acceleration/deceleration time
- X-axis target value
- Y-axis target value

The following parameters for each axis are calculated upon execution of the instruction and stored in the operation result area of the DUT.

- X-axis initial and final speed
- X-axis target speed
- Y-axis initial and final speed
- Y-axis target speed
- X-axis frequency range
- Y-axis frequency range
- X-axis number of acceleration/deceleration steps
- Y-axis number of acceleration/deceleration steps

Pulse output characteristics



5000	X-axis target value (channel 0)
2000	Y-axis target value (channel 2) (FP-X: channel 1)

The two axes are controlled so that a linear path is followed to the target position.

General programming information

- The target value for each axis must be within the range of -8388608–8388607. When this instruction is used in combination with other pulse output instructions, e.g. F171_PulseOutput_Trapezoidal (see page 1045), the target value in these instructions must be within the same range.
- When using in applications requiring precision, test runs with the actual machine are necessary.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- FP-X: When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. sys_bIsPulseChannel0Active) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: The high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) are assigned to the same internal relay (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: Executing the circular interpolation control instruction **F176** sets the circular interpolation control flag (sys_bIsCircularInterpolationActive) to TRUE. The status of this flag is maintained until the target value is reached (even if the execution condition is no longer TRUE). During this time, other pulse output instructions cannot be executed.
- FPΣ: Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- FP-X: Set "Pulse output" for the desired channel in the system registers.
- When programs are being edited in RUN mode, pulse output stops but resumes after the program changes have been downloaded.
- We strongly recommend that you incorporate a forced stop (see page 1021) option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

◆**REFERENCE**

Please refer to the FPWIN Pro online help for detailed information on using system variables.

■ Description for FP0R

Use the following predefined DUT: F175_PulseOutput_Linear_DUT_1

The following parameters can be specified in the DUT:

General programming information

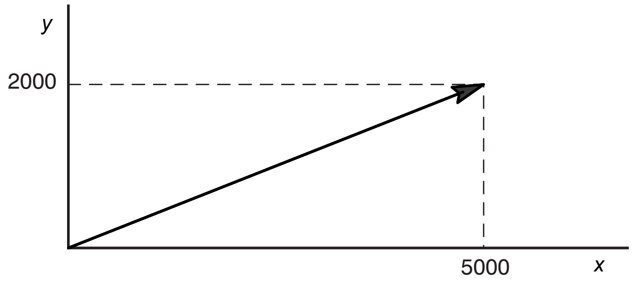
- Control code

- Initial and final speed
- Target speed
- Acceleration time
- Deceleration time
- X-axis target value
- Y-axis target value

The following parameters for each axis are calculated upon execution of the instruction and stored in the operation result area of the DUT.

- X-axis initial and final speed
- X-axis target speed
- Y-axis initial and final speed
- Y-axis target speed

Pulse output characteristics



5000	X-axis target value (channel 0)
2000	Y-axis target value (channel 1)

Pulses are output from the X-axis (channel 0) and the Y-axis (channel 1), so that the initial speed is 500Hz, the target speed is 5kHz, and the acceleration time and deceleration time is 300ms. The two axes are controlled so that a linear path is followed to the target position.

Pulses are output using a duty of 25%.

With the pulse output method "pulse/direction", pulses are output approx. 300µs after the direction signal has been output; the motor driver characteristics are simultaneously taken into consideration.

General programming information

- Pulse output stops when the target value is reached.
- The target value for each axis must be within the range of -8388608–8388607. When this instruction is used in combination with other pulse output instructions, e.g. F171_PulseOutput_Trapezoidal (see page 1045), the target value in these instructions must be within the same range.
- When using in applications requiring precision, test runs with the actual machine are necessary.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. sys_bIsPulseChannel0Active) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- When programs are being edited in RUN mode, pulse output stops but resumes after the program changes have been downloaded.

- We strongly recommend that you incorporate a forced stop (see page 1021) option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

◆REFERENCE

Please refer to the FPWIN Pro online help for detailed information on using system variables.

Output relays and system variables for FP0R

PLC types **Availability of F175_PulseOutput_Linear (see page 1322)**

Data types

Variable	Data type	Function
n_iPulseOutputChannel	Constant	Pulse output channel: FP-Σ: 0, 2 FP-X R: 0, 1 FP-X C14T: 0, 1, 2 FP-X C30T/C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 For interpolation, channel 0 and 1 or channel 2 and 3 are used as pairs. You may only specify 0 or 2 (for C14T: 0 only).
s_dutDataTable	**FP-Σ, FP-X:** F175_PulseOutput_Linear_DUT_0 **FP0R:** F175_PulseOutput_Linear_DUT_1	Starting address of area containing the data table

Operands

For	Relay				T/C		Register			Constant
s_dutDataTable	-	-	-	-	-	-	DT	-	-	-
n_iPulseOutputChannel	-	-	-	-	-	-	-	-	-	dec. or hex.

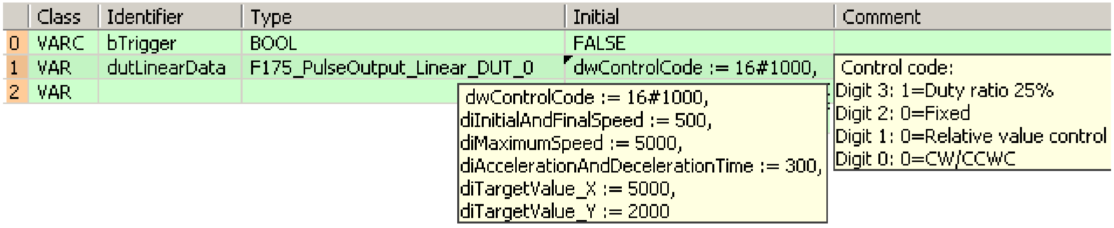
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ channel number or values of the data table are outside the permissible range ▪ Fmin > Fmax ▪ Fmax > 100kHz ▪ FP-X C14T, C30/60T (using channel 2 and 3): Fmax > 20kHz ▪ Relative value control: [elapsed value + target value] is outside the range of -8388608 to +8388607 ▪ Absolute value control: target value is outside the range of-8388608 to +8388607 ▪ FP-X: pulse output has not been set in the system registers
R9008	%MX0.900.8	for an instant	

Example In this example the function is programmed in ladder diagram (LD). The same POU header is used for all programming languages. This example is programmed for the FP-Σ. The parameters for the FP0R are only slightly different.

DUT The DUT F175_PulseOutput_Linear_DUT_0 is predefined in the FP Library.

POU header All input and output variables used for programming this function have been declared in the POU header.

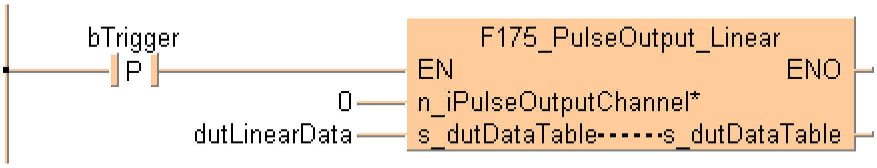


	Class	Identifier	Type	Initial	Comment
0	VARC	bTrigger	BOOL	FALSE	
1	VAR	dutLinearData	F175_PulseOutput_Linear_DUT_0	dwControlCode := 16#1000,	Control code:
2	VAR				

```
dwControlCode := 16#1000,
diInitialAndFinalSpeed := 500,
diMaximumSpeed := 5000,
diAccelerationAndDecelerationTime := 300,
diTargetValue_X := 5000,
diTargetValue_Y := 2000
```

Control code:
Digit 3: 1=Duty ratio 25%
Digit 2: 0=Fixed
Digit 1: 0=Relative value control
Digit 0: 0=CW/CCWC

LD



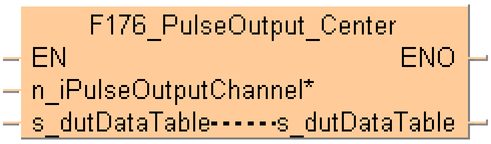
ST When programming with structured text, enter the following:

```
IF DF(bTrigger) THEN
      F175_PulseOutput_Linear(n_iPulseOutputChannel := 0,
      s_dutDataTable := dutLinearData);
END_IF;
```

F176_PulseOutput_Center

Circular interpolation (center position)

Description Pulses are output from two channels in accordance with the parameters in the specified DUT, so that the path to the target position forms an arc. The radius of the circle is calculated by specifying the center position and the end position. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



See also: PulseOutput_Center_FB (see page 1173)

Use the following predefined DUT: **F176_PulseOutput_Center_DUT**

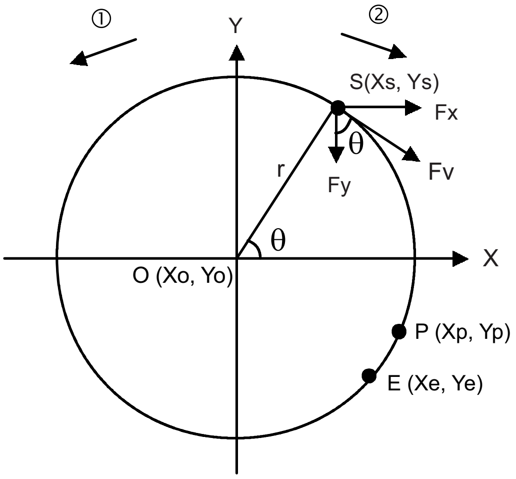
The following parameters can be specified in the DUT:

- Control code
- Composite speed
- X-axis target value
- Y-axis target value
- X-axis center value
- Y-axis center value

The following parameters for each axis are calculated upon execution of the instruction and stored in the operation result area of the DUT.

- Radius

Pulse output characteristics



①	Rotation direction: Reverse	②	Rotation direction: Forward
F_v:	Composite speed	**O (Xo,Yo):**	Center position
F_x:	X-axis speed	**S (Xs,Ys):**	Current position (Start)
F_y:	Y-axis speed	**P (Xp,Yp)**	Pass position
r:	Radius	**E (Xe,Ye)**	Target position (End)

$Fx = Fv\sin\theta = Fv\frac{|Ye-Yo|}{r}$ $Fy = Fv\cos\theta = Fv\frac{|Xe-Xo|}{r}$

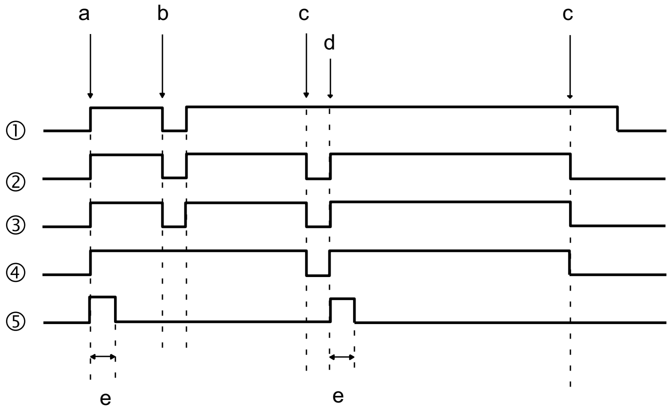
Example: Let channel 0 be the X-axis and channel 2 be the Y-axis. The position control mode is absolute value control.

The current position is (θ=60°, Xs=5000, Ys=8660). The center position O (Xo=0, Yo=0) is used as a reference point. Pulses are output from the X-axis (channel 0) and the Y-axis (channel 2) at a speed of Fv=2000Hz until the target position (θ=-30°, Xe=8660, Ye=-5000) is reached.

General programming information

- The execution condition for this instruction must be continually TRUE. When the execution condition is FALSE, pulse output stops.
- The high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) are assigned to the same internal relay (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- Executing the circular interpolation control instruction **F176** sets the circular interpolation control flag (sys_bIsCircularInterpolationActive) to TRUE. The status of this flag is maintained until the target value is reached (even if the execution condition is no longer TRUE). During this time, other pulse output instructions cannot be executed. To restart circular interpolation, perform a forced stop (stop pulse output (see page 1021)) to set the circular interpolation control flag (sys_bIsCircularInterpolationActive) to FALSE.
- If "Continue" has been selected for the operation connection mode, use a special flag (sys_bIsCircularInterpolationOverwritingPossible) to permit overwriting of the target value. The relay is TRUE for one scan when the circular interpolation instruction is executed.
- The target value for each axis must be within the range of -8388608–8388607. When this instruction is used in combination with other pulse output instructions, e.g. F171_PulseOutput_Trapezoidal (see page 1045), the target value in these instructions must be within the same range.
- The accuracy of circular interpolation may degrade if the scan time is too long.
- Online editing during RUN mode is not available for this instruction.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- If you specify the same value for the current position and the target position, a circle drawing operation will result.
- As there is no interpolation function for the home return, the home return should be executed for each channel.
- When using in applications requiring precision, test runs with the actual machine are necessary.
- Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- We strongly recommend that you incorporate a forced stop (see page 1021) option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

Flag setting during command execution



①	Execution condition X0
②	Pulse output control flag, channel 0 (sys_bIsPulseChannel0Active)
③	Pulse output control flag, channel 2 (sys_bIsPulseChannel2Active)
④	Circular interpolation control flag (sys_bIsCircularInterpolationActive)
⑤	Target value overwriting possible flag (sys_bIsCircularInterpolationOverwritingPossible)
a	Start
b	Execution condition FALSE
c	Target value reached
d	Start continue mode
e	1 scan

PLC types Availability of F176_PulseOutput_Center (see page 1322)

Data types

Variable	Data type	Function
n_iPulseOutputChannel	decimal constant	Pulse output channel: 0, 2
s_dutDataTable	F176_PulseOutput_Center_DUT	Starting address of area containing the data table

Operands

For	Relay				T/C		Register			Constant
s_dutDataTable	-	-	-	-	-	-	DT	-	-	-
n_iPulseOutputChannel	-	-	-	-	-	-	-	-	-	dec. or hex.

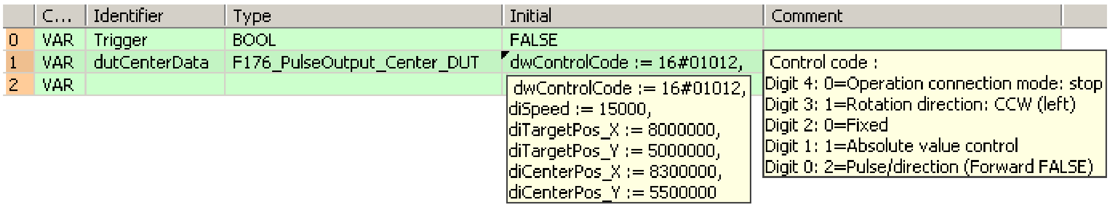
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ channel number or values of the data table are outside the permissible range ▪ Relative value control: [elapsed value + target value] is outside the range of -8388608 to +8388607 ▪ Absolute value control: target value is outside the range of-8388608 to +8388607 ▪ center position **O** = end position **E** ▪ center position **O** = start position **S**
R9008	%MX0.900.8	for an instant	

Example In this example the function is programmed in ladder diagram (LD). The same POU header is used for all programming languages.

DUT The DUT F176_PulseOutput_Center_DUT is predefined in the FP Library.

POU header All input and output variables used for programming this function have been declared in the POU header.

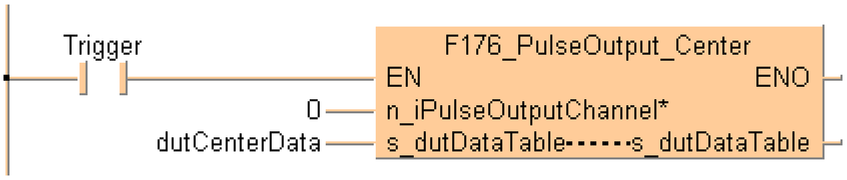


	C...	Identifier	Type	Initial	Comment
0	VAR	Trigger	BOOL	FALSE	
1	VAR	dutCenterData	F176_PulseOutput_Center_DUT	dwControlCode := 16#01012,	
2	VAR				

dwControlCode := 16#01012,
diSpeed := 15000,
diTargetPos_X := 8000000,
diTargetPos_Y := 5000000,
diCenterPos_X := 8300000,
diCenterPos_Y := 5500000

Control code :
Digit 4: 0=Operation connection mode: stop
Digit 3: 1=Rotation direction: CCW (left)
Digit 2: 0=Fixed
Digit 1: 1=Absolute value control
Digit 0: 2=Pulse/direction (Forward FALSE)

LD



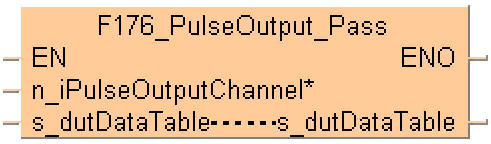
ST When programming with structured text, enter the following:

```
IF DF(Trigger) THEN
      F176_PulseOutput_Center(n_iPulseOutputChannel := 0,
      s_dutDataTable := dutCenterData);
END_IF;
```

F176_PulseOutput_ Pass

Circular interpolation (pass position)

Description Pulses are output from two channels in accordance with the parameters in the specified DUT, so that the path to the target position forms an arc. The center position and radius of the arc are calculated by specifying the pass position and the end position. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



```
        F176_PulseOutput_Pass
EN                             ENO
n_iPulseOutputChannel*
s_dutDataTable------s_dutDataTable
```

See also: PulseOutput_Pass_FB (see page 1191)

Use the following predefined DUT: **F176_PulseOutput_Pass_DUT**

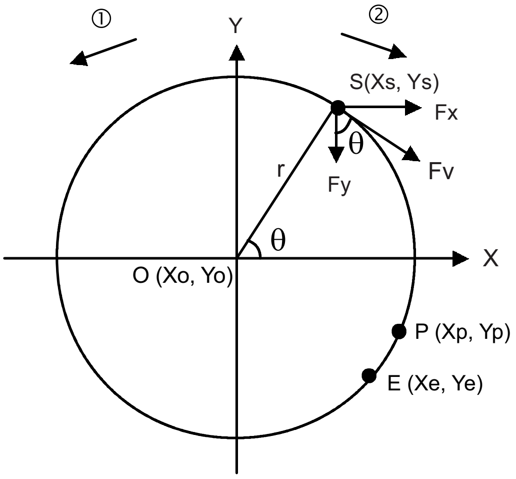
The following parameters can be specified in the DUT:

- Control code
- Composite speed
- X-axis target value
- Y-axis target value
- X-axis pass value
- Y-axis pass value

The following parameters for each axis are calculated upon execution of the instruction and stored in the operation result area of the DUT.

- Radius
- X-axis center value
- Y-axis center value

Pulse output characteristics



①	Rotation direction: Reverse	②	Rotation direction: Forward
F_v:	Composite speed	**O (Xo,Yo):**	Center position
F_x:	X-axis speed	**S (Xs,Ys):**	Current position (start)
F_y:	Y-axis speed	**P (Xp,Yp)**	Pass position

r:	Radius	**E (Xe,Ye)**	Target position (End)

$Fx = Fv\sin\theta = Fv\frac{|Ye - Yo|}{r}$ $Fy = Fv\cos\theta = Fv\frac{|Xe - Xo|}{r}$

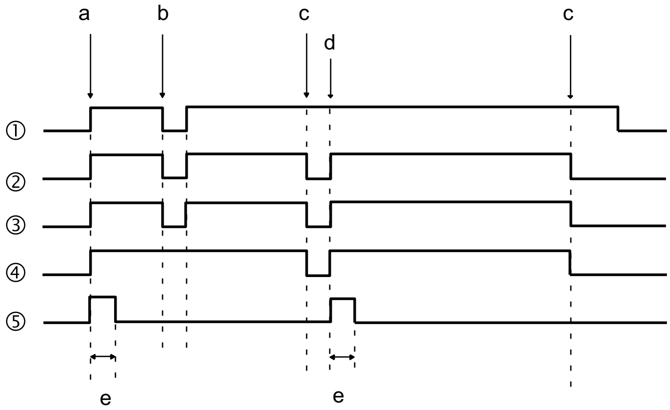
Example: Let channel 0 be the X-axis and channel 2 be the Y-axis. The position control mode is absolute value control.

The current position is (θ=60°, Xs=5000, Ys=8660). Pulses are output from the X-axis (channel 0) and the Y-axis (channel 2) at a speed of Fv=2000Hz. When the pass position (θ=-20°, Xp=9396, Yp=3420) has been passed and the target position has been reached, pulse output stops (θ=-30°, Xe=8660, Ye=-5000).

General programming information

- The execution condition for this instruction must be continually TRUE. When the execution condition is FALSE, pulse output stops.
- The high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) are assigned to the same internal relay (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. sys_bIsHscChannel0ControlActive) and the pulse output control flag (e.g. sys_bIsPulseChannel0Active) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- Executing the circular interpolation control instruction **F176** sets the circular interpolation control flag (sys_bIsCircularInterpolationActive) to TRUE. The status of this flag is maintained until the target value is reached (even if the execution condition is no longer TRUE). During this time, other pulse output instructions cannot be executed. To restart circular interpolation, perform a forced stop (stop pulse output (see page 1021)) to set the circular interpolation control flag (sys_bIsCircularInterpolationActive) to FALSE.
- If "Continue" has been selected for the operation connection mode, use a special flag (sys_bIsCircularInterpolationOverwritingPossible) to permit overwriting of the target value. The relay is TRUE for one scan when the circular interpolation instruction is executed.
- The target value for each axis must be within the range of -8388608–8388607. When this instruction is used in combination with other pulse output instructions, e.g. F171_PulseOutput_Trapezoidal (see page 1045), the target value in these instructions must be within the same range.
- The accuracy of circular interpolation may degrade if the scan time is too long.
- Online editing during RUN mode is not available for this instruction.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- As there is no interpolation function for the home return, the home return should be executed for each channel.
- When using in applications requiring precision, test runs with the actual machine are necessary.
- Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- We strongly recommend that you incorporate a forced stop (see page 1021) option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

Flag setting during command execution



①	Execution condition X0
②	Pulse output control flag, channel 0 (sys_bIsPulseChannel0Active)
③	Pulse output control flag, channel 2 (sys_bIsPulseChannel2Active)
④	Circular interpolation control flag (sys_bIsCircularInterpolationActive)
⑤	Target value overwriting possible flag (sys_bIsCircularInterpolationOverwritingPossible)
a	Start
b	Execution condition FALSE
c	Target value reached
d	Start continue mode
e	1 scan

PLC types **Availability of F176_PulseOutput_Pass (see page 1322)**

Data types

Variable	Data type	Function
n_iPulseOutputChannel	decimal constant	Pulse output channel: 0, 2
s_dutDataTable	F176_PulseOutput_Pass_DUT	Starting address of area containing the data table

Operands

For	Relay				T/C		Register			Constant
s_dutDataTable	-	-	-	-	-	-	DT	-	-	-
n_iPulseOutputChannel	-	-	-	-	-	-	-	-	-	dec. or hex.

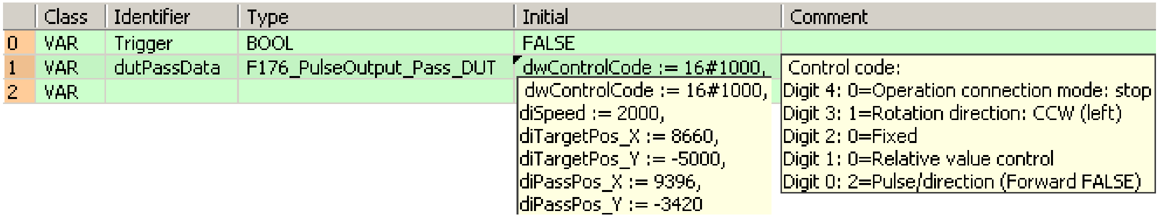
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ channel number or values of the data table are outside the permissible range ▪ Relative value control: [elapsed value + target value] is outside the range of -8388608 to +8388607 ▪ Absolute value control: target value is outside the range of-8388608 to +8388607 ▪ start position S = end position E ▪ start position S = pass position P ▪ pass position P = end position E ▪ start position S, pass position P, and end position E approximate a straight line.
R9008	%MX0.900.8	for an instant	

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

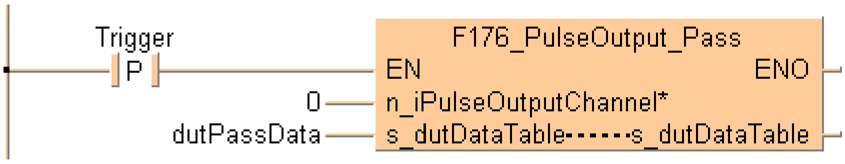
DUT The DUT F176_PulseOutput_Pass_DUT is predefined in the FP Library.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR	Trigger	BOOL	FALSE	
1	VAR	dutPassData	F176_PulseOutput_Pass_DUT	dwControlCode := 16#1000, dwControlCode := 16#1000, diSpeed := 2000, diTargetPos_X := 8660, diTargetPos_Y := -5000, diPassPos_X := 9396, diPassPos_Y := -3420	Control code: Digit 4: 0=Operation connection mode: stop Digit 3: 1=Rotation direction: CCW (left) Digit 2: 0=Fixed Digit 1: 0=Relative value control Digit 0: 2=Pulse/direction (Forward FALSE)
2	VAR				

LD



ST When programming with structured text, enter the following:

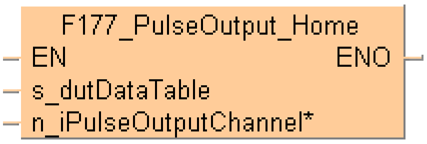
```
IF DF(Trigger) THEN
      F176_PulseOutput_Pass(n_iPulseOutputChannel := 0,
      s_dutDataTable := dutPassData);
END_IF;
```

F177_PulseOutput_Home

Home Return

Description

This instruction performs a home return according to the parameters in the specified DUT. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



See also:

- PulseOutput_Home_FB (see page 1176)
- PulseControl_NearHome (see page 1209)

After a drive system has been switched on, there is a difference between the internal position value (elapsed value) and the mechanical position of the axis; this difference cannot be predetermined. The internal value must be synchronized with the actual position value of the axis. This is done by means of a home return, during which a position value is registered at a known reference point (home).

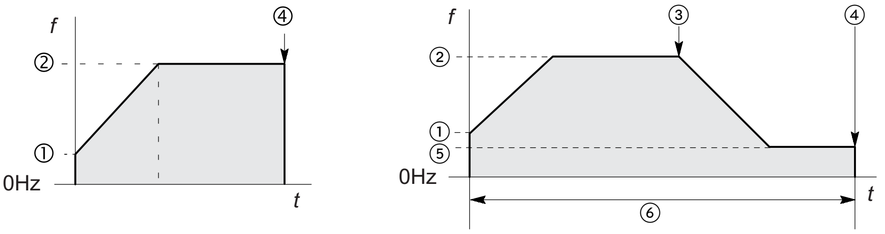
During execution of a home return instruction, pulses are continuously output until the home input is enabled. The I/O allocation is determined by the channel used.

To decelerate movement when near the home position, designate a near home input and set bit 4 of the special data register storing the pulse output control code (sys_wHscOrPulseControlCode) to TRUE and back to FALSE again.

The deviation counter clear output can be set to TRUE when home return has been completed.

Select one of two different operation modes:

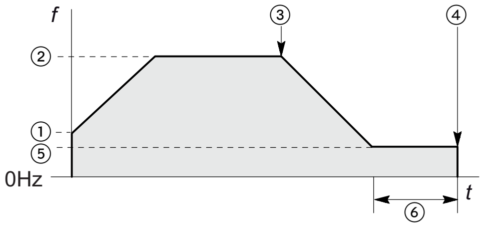
- Type 0: The home input is effective regardless of whether or not there is a near home input, whether deceleration is taking place, or whether deceleration has been completed.


Without near home input:

With near home input:

①	Initial speed	④	Home input: TRUE
②	Target speed	⑤	Creep speed
③	Near home input: TRUE	⑥	Home input is effective at any time.

- Type 1: The home input is effective only after deceleration (started by near home input) has been completed.



①	Initial speed	④	Home input: TRUE
②	Target speed	⑤	Creep speed
③	Near home input: TRUE	⑥	Home input is effective only after deceleration

Use the following predefined DUT: F177_PulseOutput_Home_Type0_DUT or F177_PulseOutput_Home_Type1_DUT

The following parameters can be specified in the DUT:

- Control code
- Initial speed
- Target speed
- Acceleration time
- Deceleration time
- Creep speed
- Deviation counter clear signal (output time)

Pulse output characteristics

- The pulse output frequency changes according to the specified acceleration time and the specified deceleration time.
- The difference between target and initial speed determines the slope of the ramps.
- Pulses are output using a duty of 25%.
- With the pulse output method "pulse/direction", pulses are output approx. 300μs after the direction signal has been output; the motor driver characteristics are simultaneously taken into consideration.

■ General programming information

- Set "Pulse output" for the desired channel in the system registers.
- Even when home input has occurred, executing this instruction causes pulse output to begin.
- If the near home input is enabled while acceleration is in progress, deceleration will start.
- The deviation counter clear signal is allocated to dedicated output numbers specific to each PLC type.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. sys_bIsPulseChannel0Active) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- When programs are being edited in RUN mode, pulse output stops but resumes after the program changes have been downloaded.
- We strongly recommend that you incorporate a forced stop (see page 1021)

option in your positioning program.

- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the flag is used more than once as an input condition, different statuses may exist within one scan. To ensure proper execution of the program, the status of the special internal relay should be copied to a variable at the beginning of the program.

PLC types **Availability of F177_PulseOutput_Home (see page 1322)**

Data types

Variable	Data type	Function
s_dutDataTable	F177_PulseOutput_Home_Type0_DUT or F177_PulseOutput_Home_Type1_DUT	Starting address of area containing the data table
n_iPulseOutputChannel	decimal constant	Pulse output channel: 0–3

Operands

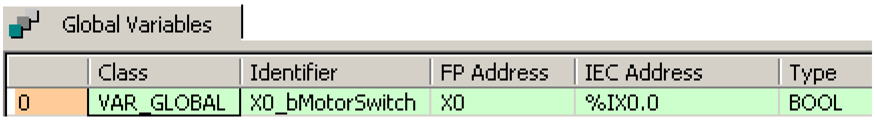
For	Relay				T/C		Register			Constant
s_dutDataTable	-	-	-	-	-	-	DT	-	-	-
n_iPulseOutputChannel	-	-	-	-	-	-	-	-	-	dec. or hex.

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	▪ channel number or values of the data table are outside the permissible range ▪ initial speed > target speed
R9008	%MX0.900.8	for an instant	

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

GVL In the global variable list, you define variables that can be accessed by all POUs in the project.

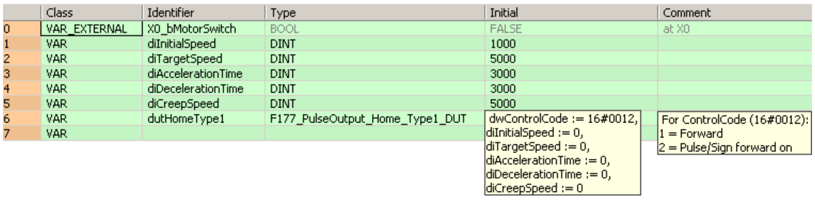


Global Variables

	Class	Identifier	FP Address	IEC Address	Type
0	VAR_GLOBAL	X0_bMotorSwitch	X0	%IX0.0	BOOL

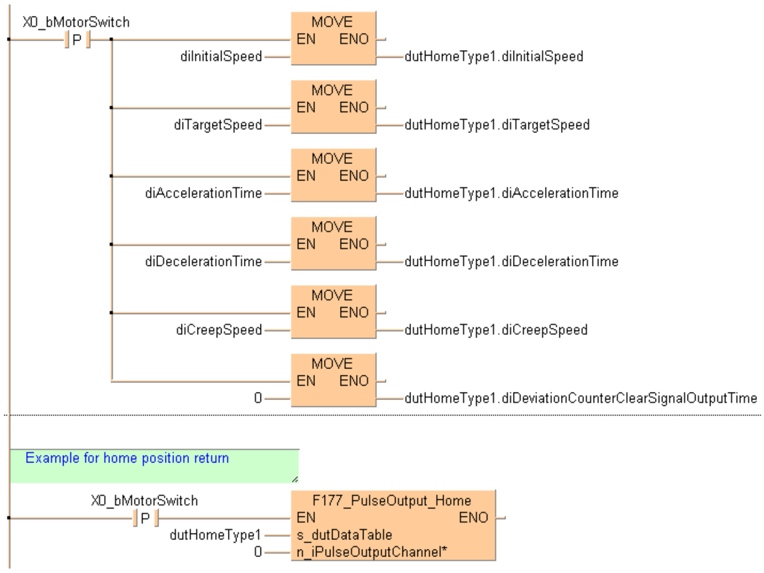
DUT The DUT F177_PulseOutput_Home_Type1_DUT is predefined in the FP Library.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	X0_bMotorSwitch	BOOL	FALSE	at X0
1	VAR	diInitialSpeed	DINT	1000	
2	VAR	diTargetSpeed	DINT	5000	
3	VAR	diAccelerationTime	DINT	3000	
4	VAR	diDecelerationTime	DINT	3000	
5	VAR	diCreepSpeed	DINT	5000	
6	VAR	dutHomeType1	F177_PulseOutput_Home_Type1_DUT	dwControlCode := 16#0012, diInitialSpeed := 0, diTargetSpeed := 0, diAccelerationTime := 0, diDecelerationTime := 0, diCreepSpeed := 0	For ControlCode (16#0012): 1 = Forward 2 = Pulse/Sign forward on
7	VAR				

LD



ST When programming with structured text, enter the following:

```
IF DF(X0_bMotorSwitch) THEN
  dutHomeType1.diInitialSpeed:=diInitialSpeed;
  dutHomeType1.diTargetSpeed:=diTargetSpeed;
  dutHomeType1.diAccelerationTime:=diAccelerationTime;
  dutHomeType1.diDecelerationTime:=diDecelerationTime;
  dutHomeType1.diCreepSpeed:=diCreepSpeed;
  dutHomeType1.diDeviationCounterClearSignalOutputTime:=0;
END_IF;

(*Example for home position return*)
IF DF(X0_bMotorSwitch) THEN
  F177_PulseOutput_Home(s_dutDataTable := dutHomeType1,
    n_iPulseOutputChannel := 0);
END_IF;
```

Chapter 36

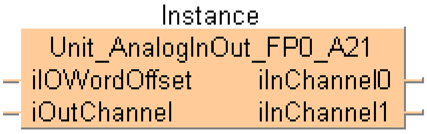
Analog unit instructions

Unit_AnalogInOut_ FP0_A21

Reads data from the FP0-A21 unit

Description

This function block reads data from the FP0-A21 unit.

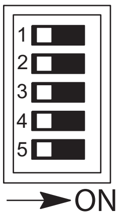


REFERENCE

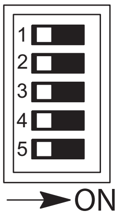
The online help only provides a short overview of DIP switch settings and wiring. For technical information, please refer to the manual FP0A21AnalogIOUnitTechnicalManual_ARCT1F390 on your FPWIN Pro installation CD.

PLC types see page 1332

Input channel setting by DIP switches 1,2,3 and 5



	0 – 5 V, 0 – 20 mA		-10 – +10 V	
	averaging			
	no averaging, see note 1	with averaging, see note 2	no averaging, see note 1	with averaging, see note 2
	Off On	Off On	Off On	Off On
1	□■	□■	■□	■□
2	□■	□■	□■	□■
3	□■	□■	□■	□■
4	▭	▭	▭	▭
5	□■	■□	□■	■□



	Thermocouples, see notes 3, 4											
	K Type				J Type				T Type			
	Temperature of terminal to 1000 °C		-100 °C to Temperature of terminal		Temperature of terminal to 750 °C		-100 °C to Temperature of terminal		Temperature of terminal to 350 °C		-100 °C to Temperature of terminal	
	Off	On	Off	On	Off	On	Off	On	Off	On	Off	On
1	■		■		■		■			■		■
2		■		■	■		■			■		■
3	■		■			■		■	■		■	
4	□		□		□		□		□		□	
5	■			■	■			■	■			■

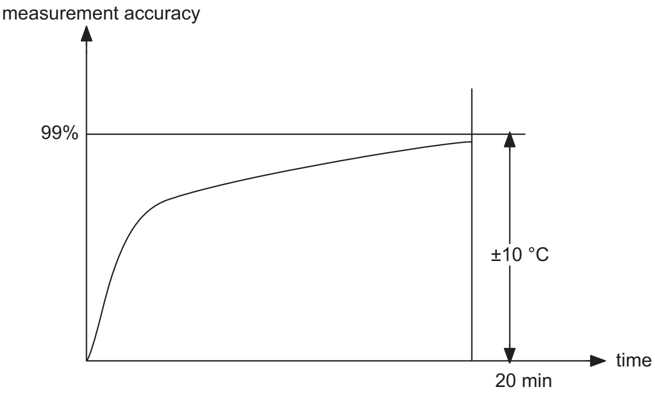
☞

1. **No averaging: Conversion data is set for the specified input contact point area for each A/D conversion, on each channel.**

2. **With averaging: on each channel, for each A/D conversion, the maximum and minimum values from the data of the last ten times are excluded, and the data from the other eight times is averaged, and the result set.**

3. **If a thermocouple setting is used, averaging is carried out, regardless of the switch settings.**

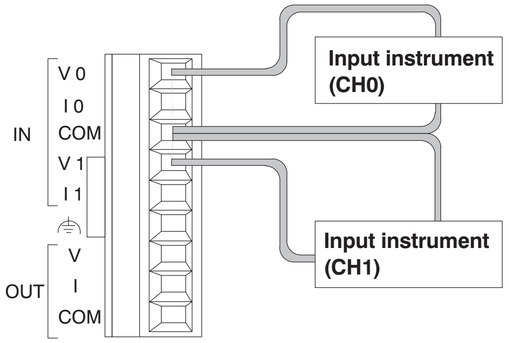
4. **After turning on the analog unit, 20 minutes are required for the transient state to reach a measurement accuracy of 99%. During this time, deviations of ±10°C can occur.**



5. **The DIP switch settings are read once at switching the CPU power to ON. Changes of the DIP switches are not recognized until the next reset of the CPU (power OFF→ ON).**

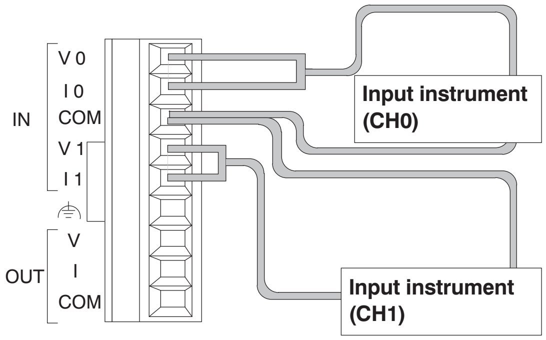
Input wiring

Voltage input



Connect input instrument between IN/V and IN/COM terminal.

Current input

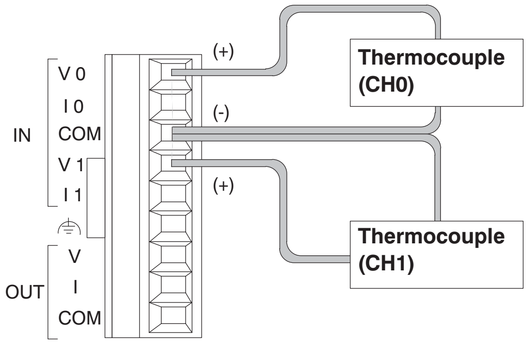


First, connect both IN/V terminal and IN/I terminal. And then connect input instrument between it and

IN/COM terminal.

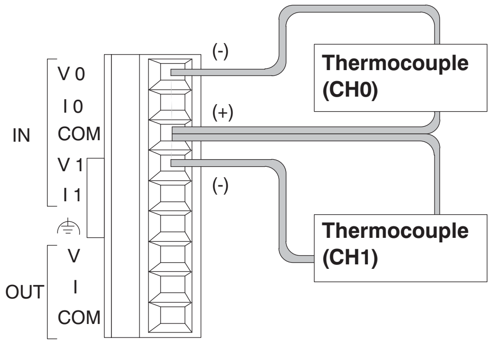
Thermocouple input

when measured at temperature higher than the temperature of the terminal



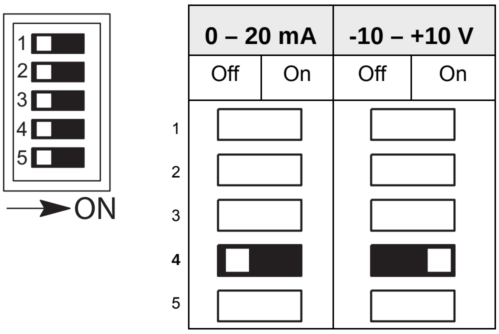
Connect IN/V terminal to the (+) side of the thermocouple, and connect IN/COM terminal to the (-) side of the thermocouple.

when measured at temperature lower than the temperature of the terminal



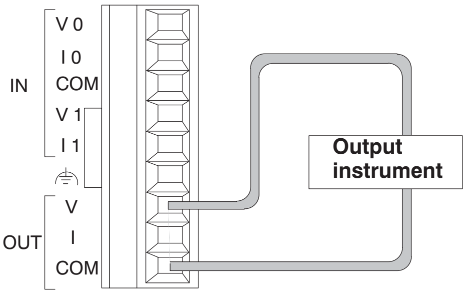
Connect IN/V terminal to the (-) side of the thermocouple, and connect IN/COM terminal to the (+) side of the thermocouple.

Output channel setting by DIP switch 4



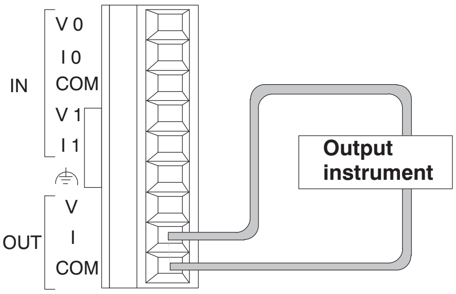
Output wiring

Voltage output



Connect output instrument between OUT/V and OUT/COM terminal.

Current output



Connect output instrument between OUT/I and OUT/COM terminal.

D/A conversion values

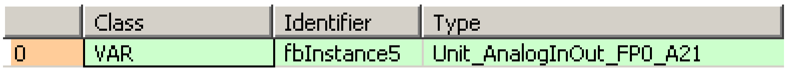
	Value of WY	Digital value	Analog output
Voltage output (V)	-2000	0	-10 V
	0	2047	0 V
	2000	4095	+10 V
Current output (mA)	0	0	0 mA
	2000	2047	10 mA
	4000	4095	20 mA

Data types

<table>
<tr><th>Input variable</th><th>Data type</th><th>Function</th></tr>
<tr><td><b>iIOWordOffset</b></td><td rowspan="2">INT</td><td>The offset of the first WX/WY address of the RTD unit according to its position.
<b>FP0R</b>, <b>FP0</b>, <b>FP-Sigma</b>: (use ExpansionUnitToIOWordOffset_FP0 (see page 1128)) or
unit 1 => address 2, unit 2 => address 4, unit 3 => address 6
<b>FP-X</b>: (use ExpansionUnotToIOWordOffset_FPX_FP0 (see page 1129)) or
<table>
<tr><td>FP0 adapter</td><td>address of unit 1</td><td>address of unit 2</td><td>address of unit 3</td></tr>
<tr><td>1st unit</td><td>30</td><td>32</td><td>34</td></tr>
<tr><td>2nd unit</td><td>40</td><td>42</td><td>44</td></tr>
<tr><td>3rd unit</td><td>50</td><td>52</td><td>54</td></tr>
<tr><td>4th unit</td><td>60</td><td>62</td><td>64</td></tr>
<tr><td>5th unit</td><td>70</td><td>72</td><td>72</td></tr>
<tr><td>6th unit</td><td>80</td><td>82</td><td>84</td></tr>
<tr><td>7th unit</td><td>90</td><td>92</td><td>94</td></tr>
<tr><td>8th unit</td><td>100</td><td>102</td><td>104</td></tr>
</table></td></tr>
<tr><td><b>iOutChannel</b></td><td>Output value channel
-20002–000->-10V–10V.
0–4000->4mA-20mA</td></tr>
<tr><th colspan="3">Output variable</th></tr>
<tr><td><b>iInChannel0–iInChannel1</b></td><td>INT</td><td>Input value at corresponding output channel 0–1
0V–5V, 0mA–20mA -> 0–4000.
-10V–10V, -100mV–100mV -> -2000–2000</td></tr>
</table>

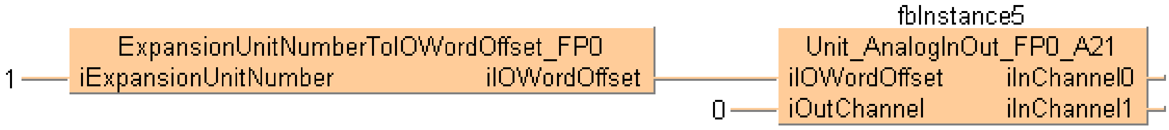
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type
0	VAR	fbInstance5	Unit_AnalogInOut_FP0_A21

LD



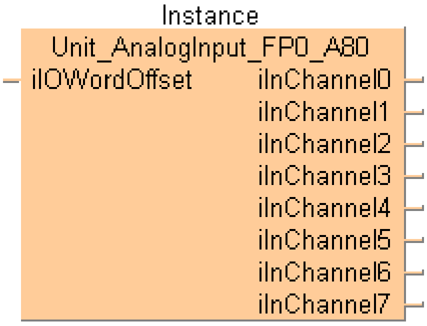
ST When programming with structured text, enter the following:

```
Unit_AnalogInOut_FP0_A21(iOutChannel := iOutChannel,
        iModuleOffsetWX := iModuleOffsetWX,
                iInChannel0 => iInChannel0,
    iInChannel1 => iInChannel1);
```

Unit_AnalogInput_ FP0_A80

Reads data from the FP0-A80 unit

Description This function block reads data from the input channels of the FP0-A80 unit. The result is stored as 16-bit words in the output variables **iInChannel0–iInChannel7**. The unit has eight channels and supports D/A conversion.



REFERENCE

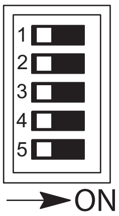
The online help only provides a short overview of DIP switch settings and wiring. For technical information, please refer to the manual FP0 A/D Converter Unit ARCT1F321 manual on your FPWIN Pro installation CD.

PLC types see see page 1333

Analog mode switch setting

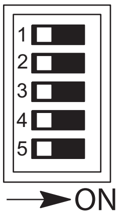
Use the DIP switches at the front of the unit to set the analog channels:

Input channels, configured by DIP switches 1 and 2:



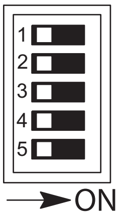
	0 – 5 V, 0 – 20 mA see note 1		-10 – +10 V		-100 – +100 mV				
	Off	On	Off	On	Off	On		Off	On
1	■			■	■		or		■
2	■		■			■			■
3									
4									
5									

Number of channels, configured by DIP switches 3 and 4:



	Off	On	Off	On	Off	On	Off	On
1								
2								
3	■			■	■			■
4	■		■			■		■
5								
channel for converted data	0 and 1		0–3		0–5		0–7	
number of input channel	2		4		6		8	

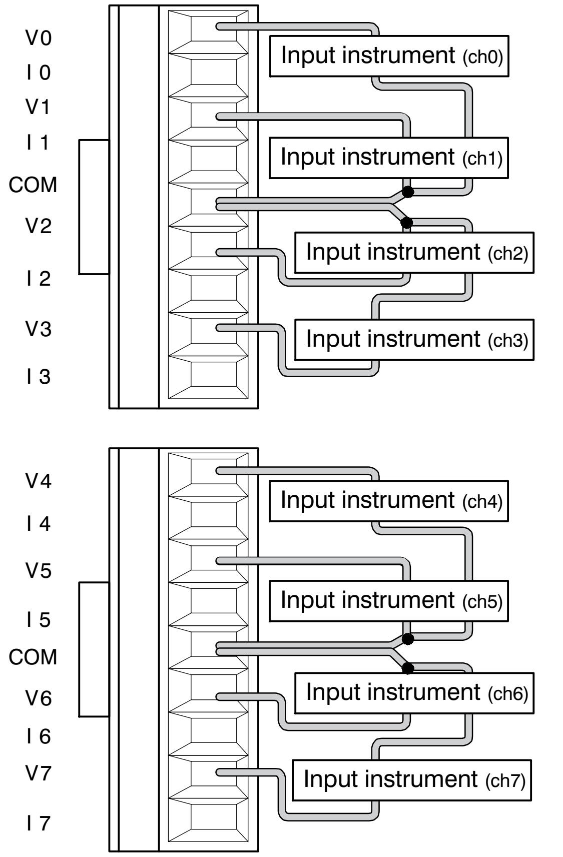
Averaging, configured by DIP switch 5:



	without averaging, see note 3		with averaging, see note 3	
	Off	On	Off	On
1				
2				
3				
4				
5	■			■

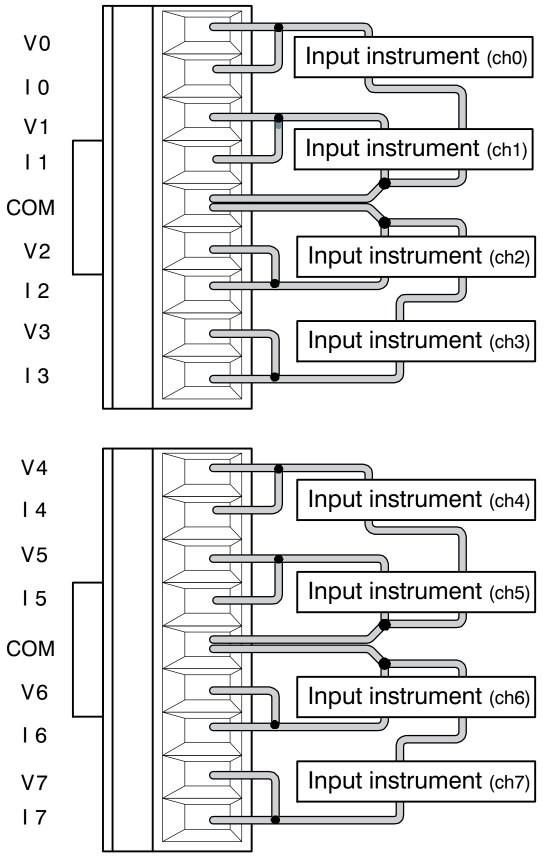
Input wiring

Voltage input



Connect input instrument between V and COM terminal.

Current input



First, connect both V terminal and I terminal. And then connect input instrument between it and COM terminal.

A/D conversion values

Input current (mA)	A/D conversion value
0.0	0
2.5	500
5.0	1000
7.5	1500
10.0	2000
12.5	2500
15.0	3000
17.5	3500
20.0	4000
Processing if the range is exceeded	
0mA or less (including negative value)	0
20mA or more	4000
Input voltage (V)	**A/D conversion value**
0.0	0
0.5	400
1.0	800
1.5	1200
2.0	1600

Input current (mA)	A/D conversion value
2.5	2000
3.0	2400
3.5	2800
4.0	3200
4.5	3600
5.0	4000
Processing if the range is exceeded	
0V or less (including negative value)	0
5V or more	4000

Data types

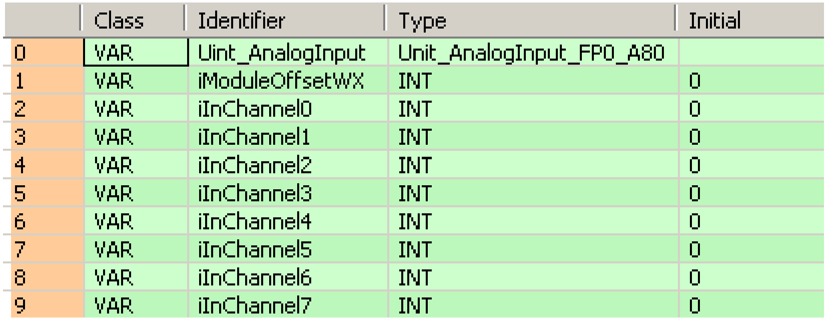
<table>
<tr><th>Input variable</th><th>Data type</th><th>Function</th></tr>
<tr><td><b>iIOWordOffset</b></td><td>INT</td><td>The offset of the first WX/WY address of the RTD unit according to its position.
<b>FP0R</b>, <b>FP0</b>, <b>FP-Sigma</b>: (use ExpansionUnitToIOWordOffset_FP0 (see page 1128)) or
unit 1 => address 2, unit 2 => address 4, unit 3 => address 6
<b>FP-X</b>: (use ExpansionUnotToIOWordOffset_FPX_FP0 (see page 1129)) or
<table>
<tr><td>FP0 adapter</td><td>address of unit 1</td><td>address of unit 2</td><td>address of unit 3</td></tr>
<tr><td>1st unit</td><td>30</td><td>32</td><td>34</td></tr>
<tr><td>2nd unit</td><td>40</td><td>42</td><td>44</td></tr>
<tr><td>3rd unit</td><td>50</td><td>52</td><td>54</td></tr>
<tr><td>4th unit</td><td>60</td><td>62</td><td>64</td></tr>
<tr><td>5th unit</td><td>70</td><td>72</td><td>72</td></tr>
<tr><td>6th unit</td><td>80</td><td>82</td><td>84</td></tr>
<tr><td>7th unit</td><td>90</td><td>92</td><td>94</td></tr>
<tr><td>8th unit</td><td>100</td><td>102</td><td>104</td></tr>
</table>
</td></tr>
<tr><th colspan="3">Output variable</th></tr>
<tr><td><b>iInChannel0–iInChannel7</b></td><td>INT</td><td>input values on the corresponding output channel 0–7:
0V–5V, 0mA–20mA -> 0–4000
-10V–10V, -100mV–100mV -> -2000–2000</td></tr>
</table>

Example

In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

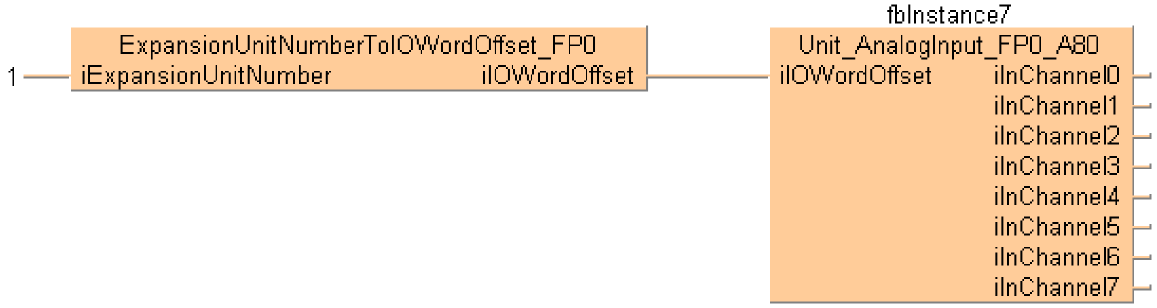
POU header

All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	Uint_AnalogInput	Unit_AnalogInput_FP0_A80	
1	VAR	iModuleOffsetWX	INT	0
2	VAR	iInChannel0	INT	0
3	VAR	iInChannel1	INT	0
4	VAR	iInChannel2	INT	0
5	VAR	iInChannel3	INT	0
6	VAR	iInChannel4	INT	0
7	VAR	iInChannel5	INT	0
8	VAR	iInChannel6	INT	0
9	VAR	iInChannel7	INT	0

LD



ST When programming with structured text, enter the following:

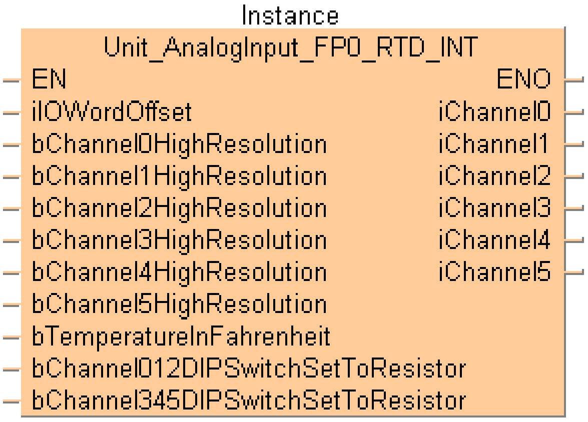
```
Uint_AnalogInput(iModuleOffsetWX := iModuleOffsetWX,
         iInChannel0 => iInChannel0,
         iInChannel1 => iInChannel1,
         iInChannel2 => iInChannel2,
         iInChannel3 => iInChannel3,
         iInChannel4 => iInChannel4,
         iInChannel5 => iInChannel5,
         iInChannel6 => iInChannel6,
iInChannel7 => iInChannel7);
```

Unit_AnalogInput_ FP0_RTD_INT

Reads analog data from the FP0-RTD6 unit

Description This function block reads RTD (Resistance Temperature Detector) data on the input channels of the FP0-RTD6 unit. The RTD unit converts the data to digital data transferred to the output channels as INTEGER values.

For the RTD input data you can use the following devices: Pt100 (according to IEC751), Pt1000 (according to IEC751), Ni1000 (according to DIN43760) or a resitor.



☞

- **Between power ON and the first valid conversion data, the digital value will be 8191 or 16383. When programming, be sure not to use the data obtained during this period.**
- **When the RTD is broken, the digital value will change to 8191 or 16383. When programming avoid any risks resulting from a broken RTD. A broken RTD needs to be replaced.**

◆ **REFERENCE**

The online help only provides a short overview of DIP switch settings and wiring. For technical information, please refer to the manual FP0 RTD Unit ACGM0159 on your FPWIN Pro installation CD.

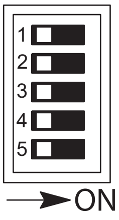
PLC types see 1333

Input range setting by DIP switches

The DIP switches configure the analog channels. You set the measurement range (type of sensor or resistor) and the sampling cycle.

The following switch settings are read once when the control unit is turned ON. Changes will not be reflected if they are performed while the control unit is turned ON.

Measurement range (switch 1 to 4):



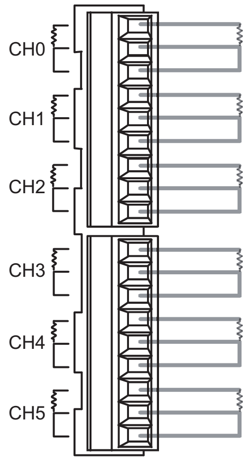
Channel	switch	Off On	Off On	Off On	Off On
0, 1, 2	1	□■	■□	□■	■□
	2	□■	□■	■□	■□
3, 4, 5	3	□■	■□	□■	■□
	4	□■	□■	■□	■□
Measurement		Pt100	Pt1000	Ni1000	Resistor

Sampling cycle (switch 5)

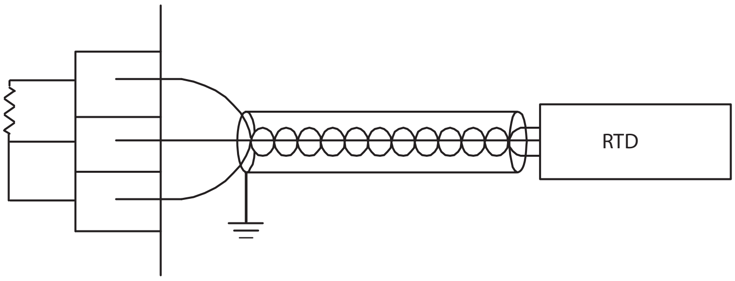
switch	Off On	Off On
1	▭	▭
2	▭	▭
3	▭	▭
4	▭	▭
5	□■	■□
Sampling cycle	0.1 s	1 s

Input wiring

Wiring method



Input line wiring



RTD = resistance temperature detector

☞ **Keep a distance of more than 100mm between the input line and the power line/high-voltage line.**

A/D conversion values

Type		temperature unit °C		temperature unit °F	
Pt100	range:	-200.0°C–+500.0°C		-328.0°F–+800.0°F	
	resolution:	0.1		0.1°F	
		analog input value °C	digital output value	analog input value °F	digital output value
		-200.0	-2000	-328.0	-3280
		+500.0	+5000	+800.0	+8000
	if input value exceeds the range	≤ -200.1 ≥ +500.1 RTD broken	8191	≤ -328.1 ≥ +800.1 RTD broken	8191
	range:	-80.00°C–+80.00°C		-80.00°F–+80.00°F	
	resolution:	0.01		0.01°F	
		analog input value °C	digital output value	analog input value °F	digital output value
		-80.00	-8000	-80.00	-8000
		+80.00	+8000	+80.00	+8000
	if input value exceeds the range	≤ -80.01 ≥ +80.01 RTD broken	8191	≤ -80.01 ≥ +80.01 RTD broken	8191
Pt1000	range:	-200.0°C–+300.0°C		-328.0°F–+572.0°F	
	resolution:	0.1		0.1°F	
		analog input value °C	digital output value	analog input value °F	digital output value
		-200.0	-2000	-328.0	-3280
		+300.0	+3000	+572.0	+5720
	if input value exceeds the range	≤ -200.1 ≥ +300.0 RTD broken	8191	≤ -328.0 ≥ +572.0 RTD broken	8191
	range:	-80.00°C–+80.00°C		-80.00°F–+80.00°F	
	resolution:	0.01		0.01°F	
		analog input value °C	digital output value	analog input value °F	digital output value
		-80.00	-8000	-80.00	-8000
		+80.00	+8000	+80.00	+8000
	if input value exceeds the range	≤ -80.01 ≥ +80.01 RTD broken	8191	≤ -80.01 ≥ +80.01 RTD broken	8191
Ni1000	range:	-30.0°C–150.0°C		-22.0°F–302.0°F	
	resolution:	0.1		0.1°F	
		analog input value °C	digital output value	analog input value °F	digital output value
		-30.0	-300	-22.0	-220

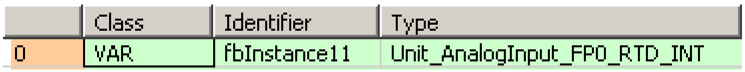
		+150.0	+1500	+302.0	+3020
	if input value exceeds the range	≤ -30.1		≤ -22.1	
		≥ +150.1	8191	≥ +302.1	8191
		RTD broken		RTD broken	
	range:	-30.00°C–+80.00°C		-22.00°F–+80.00°F	
	resolution:	0.01		0.01°F	
		analog input value °C	digital output value	analog input value °F	digital output value
		-30.00	-3000	-22.00	-2200
		+80.00	+8000	+80.00	+8000
	if input value exceeds the range	≤ -30.01		≤ -22.01	
		≥ +80.01	8191	≥ +80.01	8191
		RTD broken		RTD broken	
Resistor	range:	20Ω–2200Ω			
	resolution:	1Ω			
		+20	+20		
		+2200	+2200		
	if input value exceeds the range	≤ +19			
		≥ +2201	16383		
		resistor broken			
	range:	20.0Ω–163.0Ω			
		+20.0	+200		
		+1630.0	+16300		
	if input value exceeds the range	≤ +19.9			
		≥ +1630.1	16383		
		resistor broken			

Data types

<table>
<tr><th>Input variable</th><th>Data type</th><th>Function</th></tr>
<tr><td><b>iIOWordOffset</b></td><td>INT</td><td>The offset of the first WX/WY address of the RTD unit according to its position.
<b>FP0R</b>, <b>FP0</b>, <b>FP-Sigma</b>: (use ExpansionUnitToIOWordOffset_FP0 (see page 1128)) or
unit 1 => address 2, unit 2 => address 4, unit 3 => address 6
<b>FP-X</b>: (use ExpansionUnotToIOWordOffset_FPX_FP0 (see page 1129)) or
<table>
<tr><td>FP0 adapter</td><td>address of unit 1</td><td>address of unit 2</td><td>address of unit 3</td></tr>
<tr><td>1st unit</td><td>30</td><td>32</td><td>34</td></tr>
<tr><td>2nd unit</td><td>40</td><td>42</td><td>44</td></tr>
<tr><td>3rd unit</td><td>50</td><td>52</td><td>54</td></tr>
<tr><td>4th unit</td><td>60</td><td>62</td><td>64</td></tr>
<tr><td>5th unit</td><td>70</td><td>72</td><td>72</td></tr>
<tr><td>6th unit</td><td>80</td><td>82</td><td>84</td></tr>
<tr><td>7th unit</td><td>90</td><td>92</td><td>94</td></tr>
<tr><td>8th unit</td><td>100</td><td>102</td><td>104</td></tr>
</table></td></tr>
<tr><td><b>bChannel0HighResolution–bChannel5HighResolution</b></td><td>BOOL</td><td>sets the resolution at the corresponding channel:
▪ FALSE: low resolution
▪ TRUE: high resolution
Do not change this value during runtime.</td></tr>
<tr><td><b>bTemperatureInFahrenheit</b></td><td></td><td>sets the temperature measurement
▪ FALSE: °C
▪ TRUE: °F</td></tr>
<tr><td><b>bChannel012DIPSwitchSetToResistor</b>
<b>bChannel345DIPSwitchSetToResistor</b></td><td></td><td>settings according to the RTD device
▪ FALSE if you have set the DIP switch to Pt100, Pt1000, Ni1000
▪ TRUE if you have set the DIP switch to resistor</td></tr>
<tr><th colspan="3">Output variable</th></tr>
<tr><td><b>iInChannel0–iInChannel5</b></td><td>INT</td><td>stores the digital data from the corresponding input channels of the FP0-RTD6 unit
Temperature low resolution 0.1 °C or °F according to settings: e.g. 20.12°C -> channel value 201 (outside range 8191)
Temperature high resolution 0.01 °C or °F according to settings: e.g. 20.12°C -> channel value 2012 (outside range 8191)
Resistor low resolution 1Ω: 20Ω-2200Ω->chanel value 20-2200 (outside range 16383)
Resistor high resolution 0.1Ω: 20Ω-1630Ω->channel value 200-16300 (outside range 16383)</td></tr>
</table>

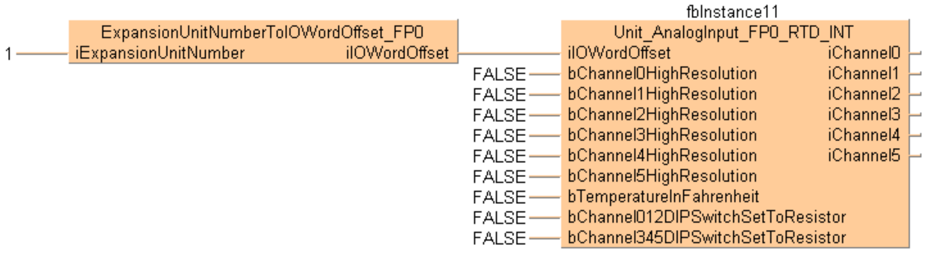
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type
0	VAR	fbInstance11	Unit_AnalogInput_FP0_RTD_INT

LD Use ExpansionUnitNumberToIOWordOffset_FP0 (see page 1125) to calculate the word offset of the analog unit connected to an FP0.


fbInstance11
ExpansionUnitNumberToIOWordOffset_FP0
1
iExpansionUnitNumber
iIOWordOffset
Unit_AnalogInput_FP0_RTD_INT
iIOWordOffset
FALSE bChannel0HighResolution
FALSE bChannel1HighResolution
FALSE bChannel2HighResolution
FALSE bChannel3HighResolution
FALSE bChannel4HighResolution
FALSE bChannel5HighResolution
FALSE bTemperatureInFahrenheit
FALSE bChannel012DIPSwitchSetToResistor
FALSE bChannel345DIPSwitchSetToResistor
iChannel0
iChannel1
iChannel2
iChannel3
iChannel4
iChannel5

ST When programming with structured text, enter the following:

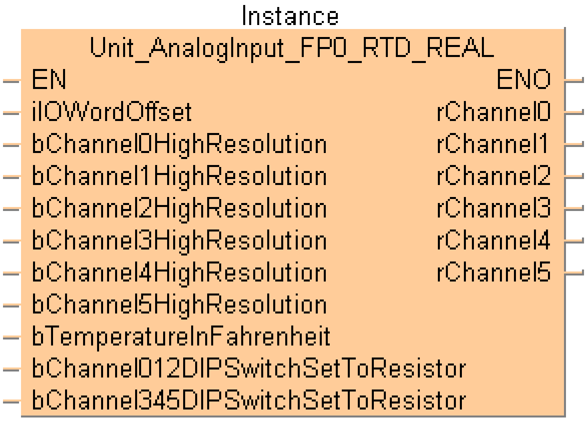
```
fbInstance11(iIOWordOffset := iIOOffsetFP0,
                  bChannel0HighResolution := bHighResolutionChannel0,
                  bChannel1HighResolution := bHighResolutionChannel1,
                  bChannel2HighResolution := bHighResolutionChannel2,
                  bChannel3HighResolution := bHighResolutionChannel3,
                  bChannel4HighResolution := bHighResolutionChannel4,
                  bChannel5HighResolution := bHighResolutionChannel5,
                  bTemperatureInFahrenheit := bHighResolutionChannel6,
                  bChannel012DIPSwitchSetToResistor := bSetToResistor012,
                  bChannel345DIPSwitchSetToResistor := bSetToResistor345,
                  iChannel0 => iIn1,
                  iChannel1 => iIn2,
                  iChannel2 => iIn3,
                  iChannel3 => iIn4,
                  iChannel4 => iIn5,
                  iChannel5 => iIn6);
```

Unit_AnalogInput_ FP0_RTD_REAL

Reads analog data from the FP0-RTD6 unit

Description This function block reads RTD (Resistance Temperature Detector) data on the input channels of the FP0-RTD6 unit. The RTD unit converts the data to digital data transferred to the output channels as REAL values.

For the RTD input data you can use the following devices: Pt100 (according to IEC751), Pt1000 (according to IEC751), Ni1000 (according to DIN43760) or a resitor.



☞

- **Between power ON and the first valid conversion data, the digital value will be 8191 or 16383. When programming, be sure not to use the data obtained during this period.**
- **When the RTD is broken, the digital value will change to 8191 or 16383. When programming avoid any risks resulting from a broken RTD. A broken RTD needs to be replaced.**

REFERENCE

The online help only provides a short overview of DIP switch settings and wiring. For technical information, please refer to the manual FP0 RTD Unit ACGM0159 on your FPWIN Pro installation CD.

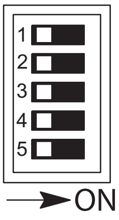
PLC types see 1333

Input range setting by DIP switches

The DIP switches configure the analog channels. You set the measurement range (type of sensor or resistor) and the sampling cycle.

The following switch settings are read once when the control unit is turned ON. Changes will not be reflected if they are performed while the control unit is turned ON.

Measurement range (switch 1 to 4):



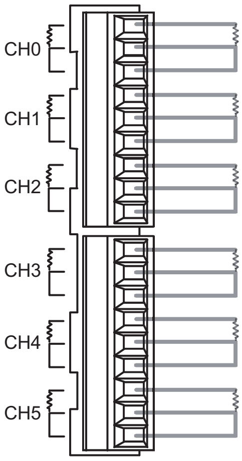
Channel	switch	Off On	Off On	Off On	Off On
0, 1, 2	1	□■	■□	□■	■□
	2	□■	□■	■□	■□
3, 4, 5	3	□■	■□	□■	■□
	4	□■	□■	■□	■□
Measurement		**Pt100**	**Pt1000**	**Ni1000**	**Resistor**

Sampling cycle (switch 5)

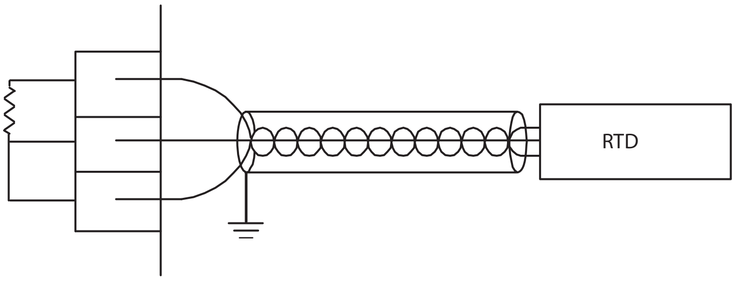
switch	Off On	Off On
1	□	□
2	□	□
3	□	□
4	□	□
5	□■	■□
Sampling cycle	0.1 s	1 s

Input wiring

Wiring method



Input line wiring



RTD = resistance temperature detector

☞ **Keep a distance of more than 100mm between the input line and the power line/high-voltage line.**

A/D conversion values

Type		temperature unit °C		temperature unit °F	
Pt100	range:	-200.0°C–+500.0°C		-328.0°F–+800.0°F	
	resolution:	0.1		0.1°F	
		analog input value °C	digital output value	analog input value °F	digital output value
		-200.0	-2000	-328.0	-3280
		+500.0	+5000	+800.0	+8000
	if input value exceeds the range	≤ -200.1 ≥ +500.1 RTD broken	8191	≤ -328.1 ≥ +800.1 RTD broken	8191
	range:	-80.00°C–+80.00°C		-80.00°F–+80.00°F	
	resolution:	0.01		0.01°F	
		analog input value °C	digital output value	analog input value °F	digital output value
		-80.00	-8000	-80.00	-8000
		+80.00	+8000	+80.00	+8000
	if input value exceeds the range	≤ -80.01 ≥ +80.01 RTD broken	8191	≤ -80.01 ≥ +80.01 RTD broken	8191
Pt1000	range:	-200.0°C–+300.0°C		-328.0°F–+572.0°F	
	resolution:	0.1		0.1°F	
		analog input value °C	digital output value	analog input value °F	digital output value
		-200.0	-2000	-328.0	-3280
		+300.0	+3000	+572.0	+5720
	if input value exceeds the range	≤ -200.1 ≥ +300.0 RTD broken	8191	≤ -328.0 ≥ +572.0 RTD broken	8191
	range:	-80.00°C–+80.00°C		-80.00°F–+80.00°F	
	resolution:	0.01		0.01°F	
		analog input value °C	digital output value	analog input value °F	digital output value
		-80.00	-8000	-80.00	-8000
		+80.00	+8000	+80.00	+8000
	if input value exceeds the range	≤ -80.01 ≥ +80.01 RTD broken	8191	≤ -80.01 ≥ +80.01 RTD broken	8191
Ni1000	range:	-30.0°C–150.0°C		-22.0°F–302.0°F	
	resolution:	0.1		0.1°F	
		analog input value °C	digital output value	analog input value °F	digital output value
		-30.0	-300	-22.0	-220

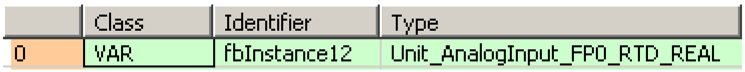
		+150.0	+1500	+302.0	+3020
	if input value exceeds the range	≤ -30.1		≤ -22.1	
		≥ +150.1	8191	≥ +302.1	8191
		RTD broken		RTD broken	
	range:	-30.00°C–+80.00°C		-22.00°F–+80.00°F	
	resolution:	0.01		0.01°F	
		analog input value °C	digital output value	analog input value °F	digital output value
		-30.00	-3000	-22.00	-2200
		+80.00	+8000	+80.00	+8000
	if input value exceeds the range	≤ -30.01		≤ -22.01	
		≥ +80.01	8191	≥ +80.01	8191
		RTD broken		RTD broken	
Resistor	range:	20Ω–2200Ω			
	resolution:	1Ω			
		+20	+20		
		+2200	+2200		
	if input value exceeds the range	≤ +19			
		≥ +2201	16383		
		resistor broken			
	range:	20.0Ω–163.0Ω			
		+20.0	+200		
		+1630.0	+16300		
	if input value exceeds the range	≤ +19.9			
		≥ +1630.1	16383		
		resistor broken			

Data types

<table>
<tr><th>Input variable</th><th>Data type</th><th>Function</th></tr>
<tr><td><b>iIOWordOffset</b></td><td>INT</td><td>The offset of the first WX/WY address of the RTD unit according to its position.
<b>FP0R</b>, <b>FP0</b>, <b>FP-Sigma</b>: (use ExpansionUnitToIOWordOffset_FP0 (see page 1128)) or
unit 1 => address 2, unit 2 => address 4, unit 3 => address 6
<b>FP-X</b>: (use ExpansionUnotToIOWordOffset_FPX_FP0 (see page 1129)) or
<table>
<tr><td>FP0 adapter</td><td>address of unit 1</td><td>address of unit 2</td><td>address of unit 3</td></tr>
<tr><td>1st unit</td><td>30</td><td>32</td><td>34</td></tr>
<tr><td>2nd unit</td><td>40</td><td>42</td><td>44</td></tr>
<tr><td>3rd unit</td><td>50</td><td>52</td><td>54</td></tr>
<tr><td>4th unit</td><td>60</td><td>62</td><td>64</td></tr>
<tr><td>5th unit</td><td>70</td><td>72</td><td>72</td></tr>
<tr><td>6th unit</td><td>80</td><td>82</td><td>84</td></tr>
<tr><td>7th unit</td><td>90</td><td>92</td><td>94</td></tr>
<tr><td>8th unit</td><td>100</td><td>102</td><td>104</td></tr>
</table></td></tr>
<tr><td><b>bChannel0HighResolution–bChannel5HighResolution</b></td><td rowspan="4">BOOL</td><td>sets the resolution at the corresponding channel:
▪ FALSE: low resolution
▪ TRUE: high resolution
Do not change this value during runtime. If you change this value during runtime the channel value will be wrong for about one second.</td></tr>
<tr><td><b>bTemperatureInFahrenheit</b></td><td>sets the temperature measurement
▪ FALSE: °C
▪ TRUE: °F</td></tr>
<tr><td><b>bChannel012DIPSwitchSetToResistor</b></td><td rowspan="2">settings according to the RTD device
▪ FALSE if you have set the DIP switch to Pt100, Pt1000, Ni1000
▪ TRUE if you have set the DIP switch to resistor</td></tr>
<tr><td><b>bChannel345DIPSwitchSetToResistor</b></td></tr>
<tr><th colspan="3">Output variable</th></tr>
<tr><td><b>rInChannel0–rInChannel5</b></td><td>REAL</td><td>stores the digital data from the corresponding input channels of the FP0-RTD6 unit as REAL values
Temperature low resolution 0.1 °C or °F according to settings: e.g. 20.12°C -> channel value 20.1 (outside range 819.1)
Temperature high resolution 0.01 °C or °F according to settings e.g. 20.12°C -> channel value 20.12 (outside range 81.91)
Resistor low resolution 1Ω: 20Ω-2200Ω->channel value 20-2200 (outside range 16383)
Resistor high resolution 0.1Ω: 20Ω-1630Ω->channel value 20.0-1630.0 (outside range 1638.3)</td></tr>
</table>

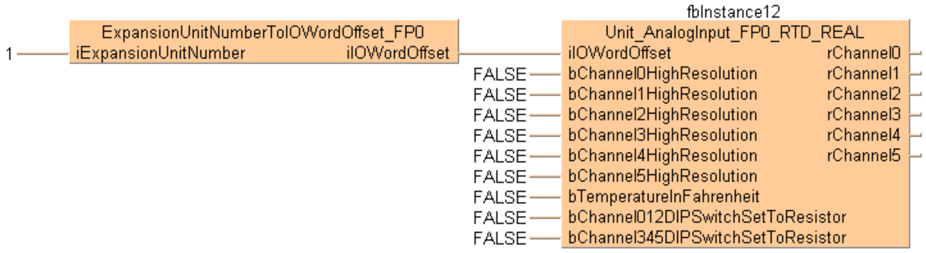
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type
0	VAR	fbInstance12	Unit_AnalogInput_FP0_RTD_REAL

LD Use ExpansionUnitNumberToIOWordOffset_FP0 (see page 1125) to calculate the word offset of the analog unit connected to an FP0.



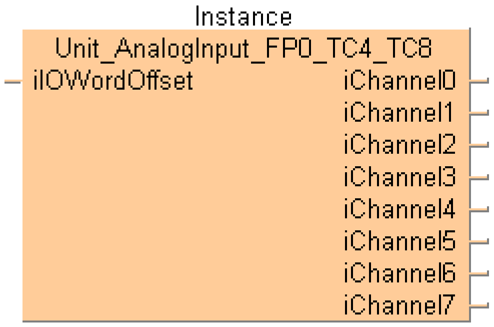
ST When programming with structured text, enter the following:

```
fbInstance12(iIOWordOffset := iIOOffsetFP0,
                  bChannel0HighResolution := bHighResolutionChannel0,
                  bChannel1HighResolution := bHighResolutionChannel1,
                  bChannel2HighResolution := bHighResolutionChannel2,
                  bChannel3HighResolution := bHighResolutionChannel3,
                  bChannel4HighResolution := bHighResolutionChannel4,
                  bChannel5HighResolution := bHighResolutionChannel5,
                  bTemperatureInFahrenheit := bHighResolutionChannel6,
                  bChannel012DIPSwitchSetToResistor := bSetToResistor012,
                  bChannel345DIPSwitchSetToResistor := bSetToResistor345);

rReal0 := fbInstance7.rChannel0;
rReal1 := fbInstance7.rChannel1;
rReal2 := fbInstance7.rChannel2;
rReal3 := fbInstance7.rChannel3;
rReal4 := fbInstance7.rChannel4;
```

Unit_AnalogInput_ FP0_TC4_TC8

Reads data from the FP0-TC4/FP0-TC8 unit

Description This function block reads the analog input data of the analog unit FP0-TC4 (four analog input channels) or FP0-TC8 (eight analog input channels). The result is stored as 16-bit words in the output variables **iChannel0–iChannel3** for FP0-TC4 and **iChannel0–iChannel7** for FP0-TC8. The function block supports the thermo couple types K, J, T and R. Furthermore it supports averaging and detects if the thermocouple is broken.



◆ **REFERENCE**

For technical information, please refer to the manual FP0 Analog unit manual on your FPWIN Pro installation CD.

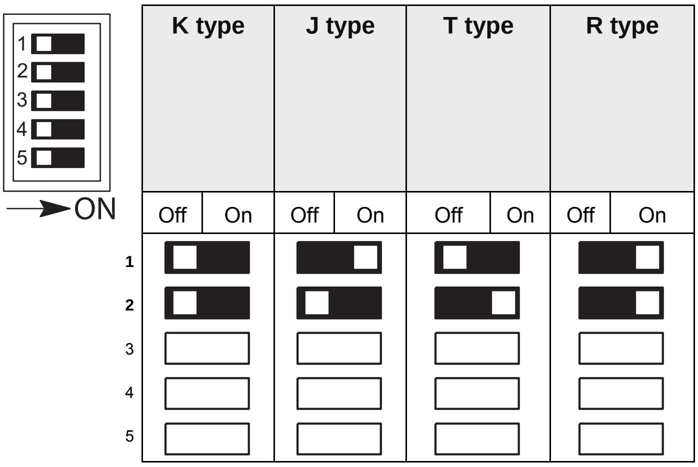
PLC types see see page 1333

DIP switch settings

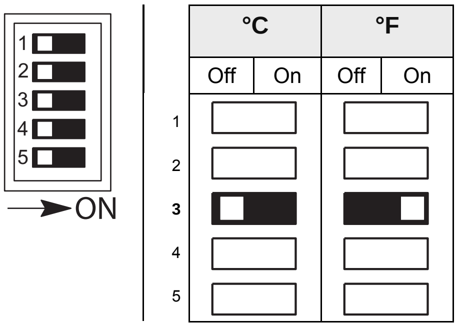
Use the DIP switches at the front side to set the thermocouple type, the temperature unit (°C, °F) and the number of input channels.

- **The DIP switch settings are read once at switching the CPU power to ON. Changes of the DIP switches are not recognized until the next reset of the CPU (power OFF→ ON).**

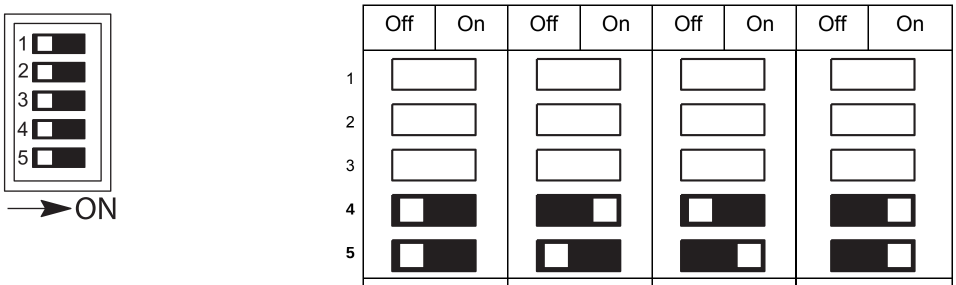
Type of thermocouple, configured by DIP switch 1 and 2



Temperature unit configured by DIP switch 3

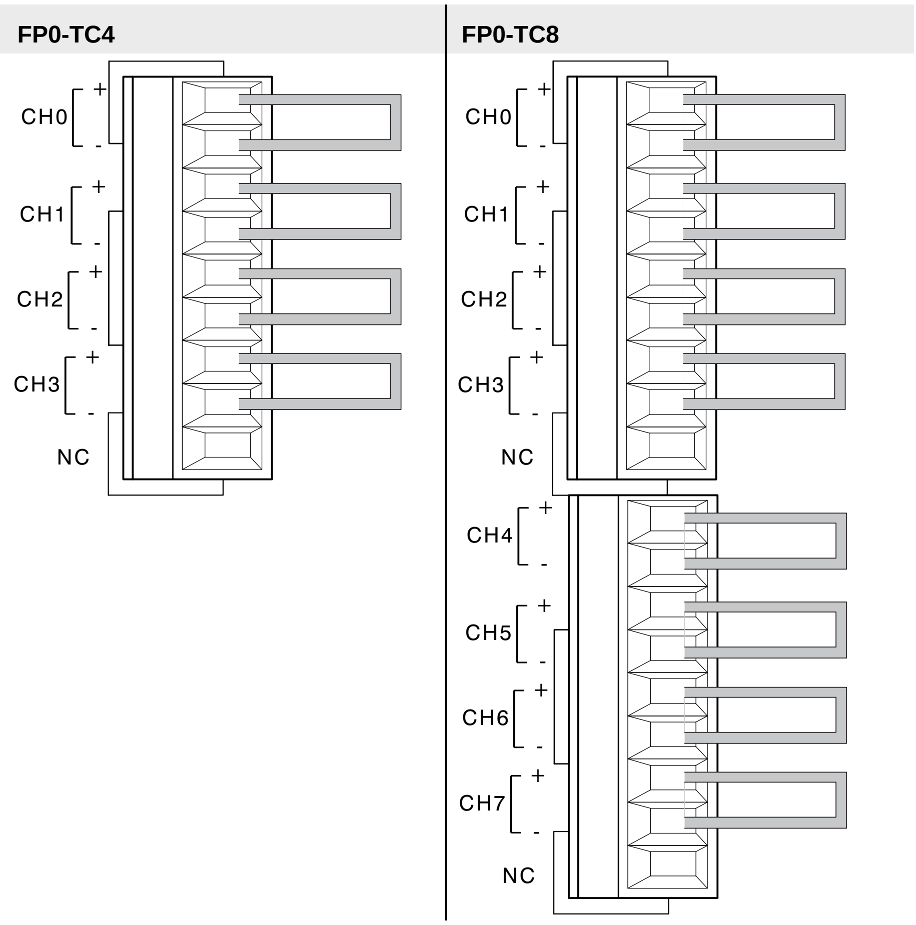


Input channels configured by DIP switches 4 and 5



channel for converted data	0 and 1	0–3	0–5	0–7
number of input channel	2	4	6	8

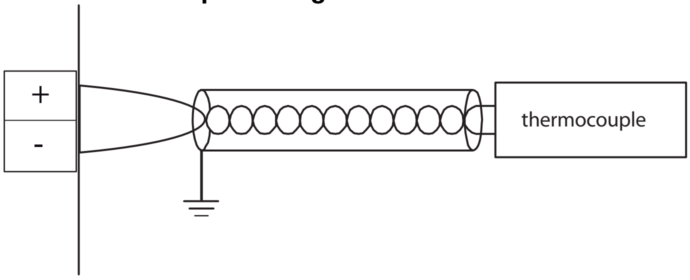
Input wiring


FP0-TC4

FP0-TC8

☞

- **Keep a distance of more than 100mm between the input line and the power line/high-voltage line.**
- **A shielded compensating lead wire is recommended.**



A/D conversion values

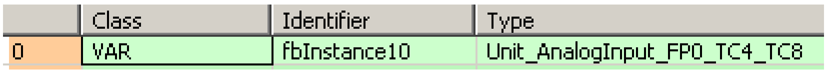
Type		input temperature °C	digial value
K,J	range:	-100.0°C–500.0°C	
		-100.0	-1000
		+500.0	+5000
	if input value exceeds the range	≤ -100.1	-1001
		≥ +500.1	5001 or 8000
		broken thermocouple	8000
T	range:	-100.0°C–400.0°C	
		-100.0	-1000
		+400.0	+4000
	if input value exceeds the range	≤ -100.1	-1001
		≥ +400.1	4001 or 8000
		broken thermocouple	8000
R	range:	-100.0°C–1500.0°C	
		0	0
		+1500.0	+15000
	if input value exceeds the range	≤ -0.0	0
		≥ +1500.1	15001 or 16000
		broken thermocouple	16000

Data types

<table>
<tr><th>Input variable</th><th>Data type</th><th>Function</th></tr>
<tr><td><b>iIOWordOffset</b></td><td>INT</td><td>The offset of the first WX/WY address of the RTD unit according to its position.
<b>FP0R</b>, <b>FP0</b>, <b>FP-Sigma</b>: (use ExpansionUnitToIOWordOffset_FP0 (see page 1128)) or
unit 1 => address 2, unit 2 => address 4, unit 3 => address 6
<b>FP-X</b>: (use ExpansionUnotToIOWordOffset_FPX_FP0 (see page 1129)) or
<table>
<tr><th>FP0 adapter</th><th>address of unit 1</th><th>address of unit 2</th><th>address of unit 3</th></tr>
<tr><td>1st unit</td><td>30</td><td>32</td><td>34</td></tr>
<tr><td>2nd unit</td><td>40</td><td>42</td><td>44</td></tr>
<tr><td>3rd unit</td><td>50</td><td>52</td><td>54</td></tr>
<tr><td>4th unit</td><td>60</td><td>62</td><td>64</td></tr>
<tr><td>5th unit</td><td>70</td><td>72</td><td>72</td></tr>
<tr><td>6th unit</td><td>80</td><td>82</td><td>84</td></tr>
<tr><td>7th unit</td><td>90</td><td>92</td><td>94</td></tr>
<tr><td>8th unit</td><td>100</td><td>102</td><td>104</td></tr>
</table></td></tr>
<tr><th colspan="3">Output variable</th></tr>
<tr><td><b>iChannel0–iChannel7</b></td><td></td><td>input values on the corresponding output channel (0–3 for FP0-TC4, channel0–7 for FP0-TC8):
<b>Range K, J type</b> (-100,1°C to 500,1°C->-1001 to 5001 or -148,1°F to 790,1°F -> -1481 to 7901)
<b>Range T type</b>: (-100,1°C to 400,1°C -> -1001 to 4001 or -148,1°F to 752,1°F -> -1481 to 7521)
<b>Range R type</b>: (0°C to 1500,1°C -> 0 to 15001 or 32°F to 1590,1°F -> 320 to 15901)
8000 (when the thermocouple is broken)</td></tr>
</table>

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



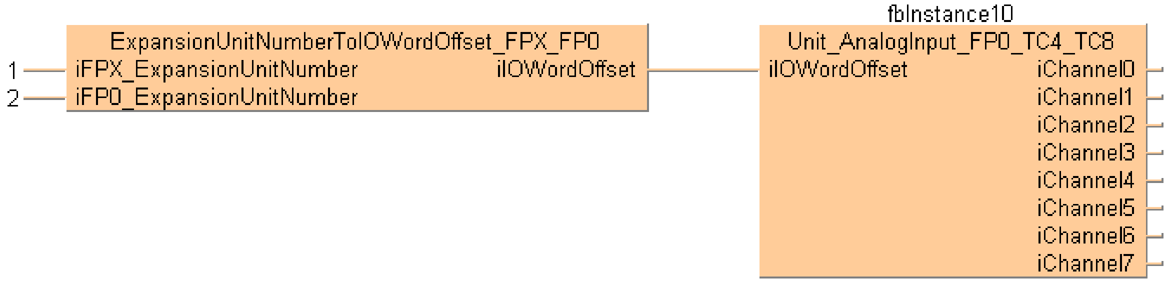
	Class	Identifier	Type
0	VAR	fbInstance10	Unit_AnalogInput_FP0_TC4_TC8

Body Use ExpansionUnitNumberToIOWordOffset_FP0 (see page 1125) to calculate the word offset of the analog unit connected to an FP0.

This function block reads the analog input data from the FP0-TC4 (TC8) unit and stores the digital values in the corresponding ínput channels:

Range	°C values	digital values	°F values	digital values
K, J Type	-100.1–500.1	-1001–5001	-148.1–790.1	-1481–7901
T Type	-100.1–400.1	-1001–4001	-148.1–752.1	-1481–7521
R Type	0–1500.1	0–15001	32–1590.1	320–15901
8000	when the thermocouple is broken			

LD

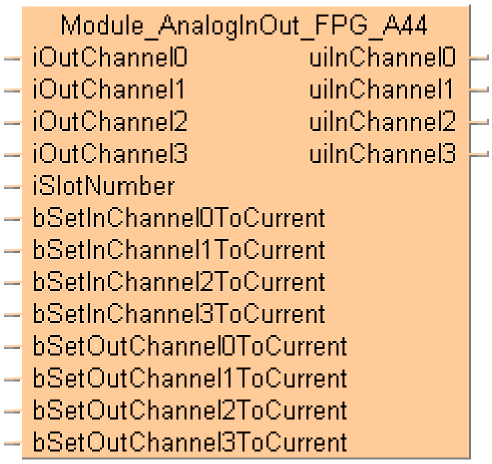


```
ST fbInstance10(iIOWordOffset := iIOOffsetFP0);

   iIn1 := fbInstance10.iChannel0;
   iIn2 := fbInstance10.iChannel1;
   iIn3 := fbInstance10.iChannel2;
   iIn4 := fbInstance10.iChannel3;
   iIn5 := fbInstance10.iChannel4;
   iIn6 := fbInstance10.iChannel5;
   iIn7 := fbInstance10.iChannel6;
   iIn8 := fbInstance10.iChannel7;
```

Unit_AnalogInOut_FPG_A44

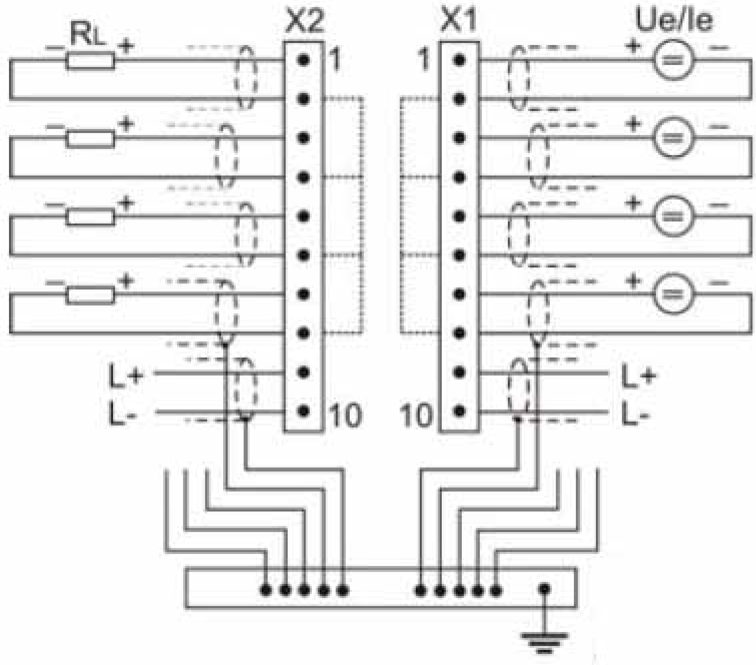
Reads data from the FPG-A44 unit

Description This function block reads data from the FPG-A44 unit. The unit converts analog input data (0-10 V DC and 0-20 mA DC) into 16-bit word digital output values.



PLC types see page 1335

Wiring diagram



EMC shield bar in proximity to the plug module.

A/D conversion values

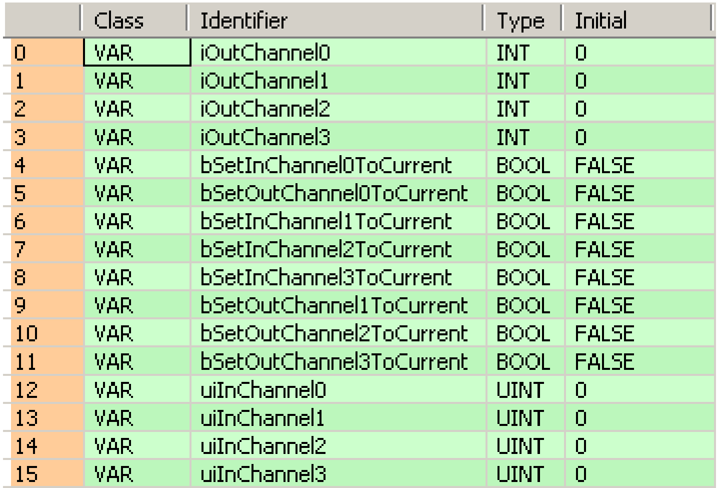
Input value (mA) range: 0–20mA	**output value**
20.00	65535
19.00	62258
18.00	58982
17.00	55705
16.00	52428
15.00	49151
14.00	45875
13.00	42598
12.00	39321
11.00	36044
10.00	32768
9.00	29491
8.00	26214
7.00	22937
6.00	19661
5.00	16384
4.00	13107
3.00	9830
2.00	6554
1.00	3277
0.00	0
Input voltage (V) range: 0.00–10.00 V	**output value**
10.00	65535
9.50	62258
9.00	58982
8.50	55705
8.00	52428
7.50	49151
7.00	45875
6.50	42598
6.00	39321
5.50	36044
5.00	32768
4.50	29491
4.00	26214
3.50	22937
3.00	19661
2.50	16384
2.00	13107
3.00	9830
2.00	6554
1.00	3277
0.00	0

Data types

Input variable	Data type	Function
iOutChannel0– **iOutChannel3**	INT	Stores the value from the corresponding channel number of the FPG-A44 unit.
iSlotNumber		Specifies the slot number
bSetInChannel0ToCurrent– **bSetInChannel3ToCurrent**	BOOL	If TRUE, the operation mode is set to the current output type for the corresponding input channel number. Otherwise, the voltage output type is set.
bSetOutChannel0ToCurrent– **bSetOutChannel3ToCurrent**		If TRUE, the operation mode is set to the current output type for the corresponding output channel number. Otherwise, voltage output type is set.
Output variable		
uiInChannel0– **uiInChannel3**	UINT	Stores the converted values from the corresponding channels

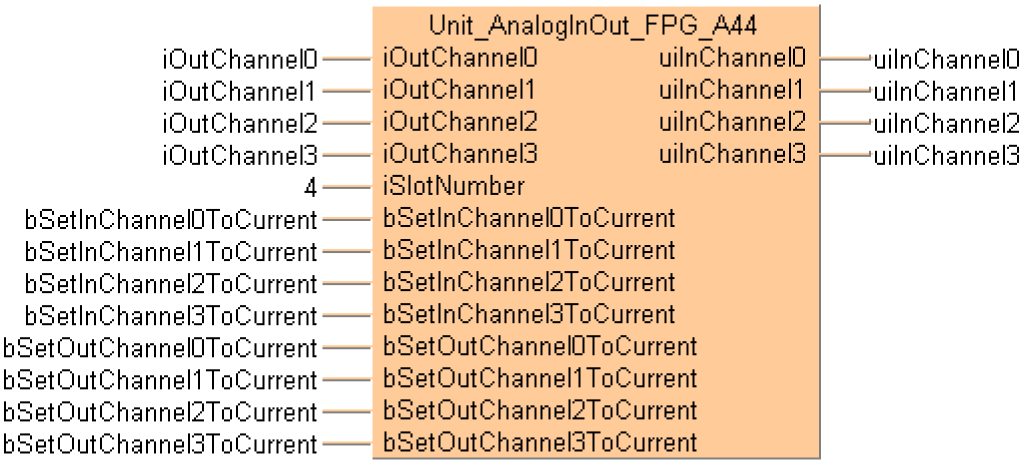
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iOutChannel0	INT	0
1	VAR	iOutChannel1	INT	0
2	VAR	iOutChannel2	INT	0
3	VAR	iOutChannel3	INT	0
4	VAR	bSetInChannel0ToCurrent	BOOL	FALSE
5	VAR	bSetOutChannel0ToCurrent	BOOL	FALSE
6	VAR	bSetInChannel1ToCurrent	BOOL	FALSE
7	VAR	bSetInChannel2ToCurrent	BOOL	FALSE
8	VAR	bSetInChannel3ToCurrent	BOOL	FALSE
9	VAR	bSetOutChannel1ToCurrent	BOOL	FALSE
10	VAR	bSetOutChannel2ToCurrent	BOOL	FALSE
11	VAR	bSetOutChannel3ToCurrent	BOOL	FALSE
12	VAR	uiInChannel0	UINT	0
13	VAR	uiInChannel1	UINT	0
14	VAR	uiInChannel2	UINT	0
15	VAR	uiInChannel3	UINT	0

LD



ST When programming with structured text, enter the following:

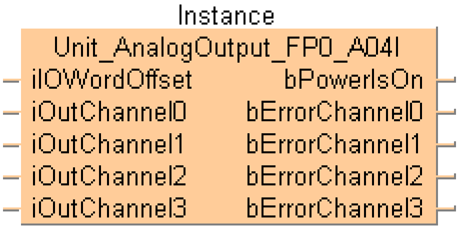
```
Unit_AnalogInOut_FPG_A44(iOutChannel0 := iOutChannel0,
             iOutChannel1 := iOutChannel1,
             iOutChannel2 := iOutChannel2,
             iOutChannel3 := iOutChannel3,
             iSlotNumber := 4,
             bSetInChannel0ToCurrent := bSetInChannel0ToCurrent,
             bSetInChannel1ToCurrent := bSetInChannel1ToCurrent,
             bSetInChannel2ToCurrent := bSetInChannel2ToCurrent,
             bSetInChannel3ToCurrent := bSetInChannel3ToCurrent,
             bSetOutChannel0ToCurrent := bSetOutChannel0ToCurrent,
             bSetOutChannel1ToCurrent := bSetOutChannel1ToCurrent,
             bSetOutChannel2ToCurrent := bSetOutChannel2ToCurrent,
             bSetOutChannel3ToCurrent := bSetOutChannel3ToCurrent,
             uiInChannel0 => uiInChannel0,
             uiInChannel1 => uiInChannel1,
             uiInChannel2 => uiInChannel2,
    uiInChannel3 => uiInChannel3);
```

Unit_AnalogOutput _FP0_A04I

Reads data from the FP0-A04 unit

Description

This function block reads data from the FP0-A04 current output type from the output channels 0–3 and stores the digital data in the input channels **iOutChannel0–iOutChannel3**. The valid range is from 4–20 mA (0–4000).

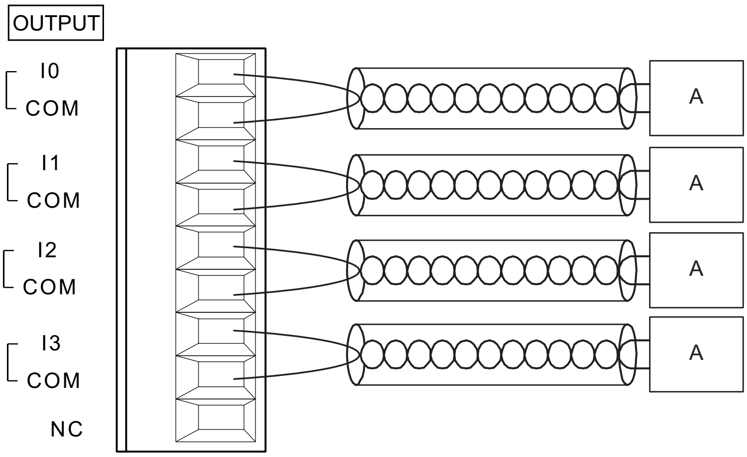


REFERENCE

For technical information, please refer to the manual FP0 Analog unit manual on your FPWIN Pro installation CD.

PLC types see see page 1333

Wiring of analog outputs



A: Analog device

- **The COM contacts are connected internally.**

- **The COM contacts are connected internally.**

- **Keep a distance of more than 100mm between the output line and the power line/high-voltage line.**
- **For wiring the analog outputs shielded twisted pair cables are recommended. Connect the shield with the frame ground of the analog unit.**

- **For wiring the analog outputs shielded twisted pair cables are recommended. Connect the shield with the frame ground of the analog unit.**

D/A conversion values

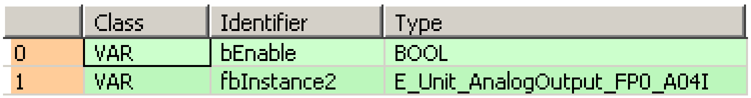
digital input value range: 0–4000	**current output (mA)** range: 4–20 mA
0	4.0
500	6.0
1000	8.0
1500	10.0
2000	12.0
2500	14.5
3000	16.0
3500	18.5
4000	20.0
values outside of range	
≤ -1	constant, the converted value exactly is based on the latest valid input value
≥ +4001	

Data types

<table>
<tr><th>Input variable</th><th>Data type</th><th>Function</th></tr>
<tr><td><b>iIOWordOffset</b></td><td>INT</td><td>The offset of the first WX/WY address of the RTD unit according to its position.
<b>FP0R</b>, <b>FP0</b>, <b>FP-Sigma</b>: (use ExpansionUnitToIOWordOffset_FP0 (see page 1128)) or
unit 1 => address 2, unit 2 => address 4, unit 3 => address 6
<b>FP-X</b>: (use ExpansionUnotToIOWordOffset_FPX_FP0 (see page 1129)) or
<table>
<tr><th>FP0 adapter</th><th>address of unit 1</th><th>address of unit 2</th><th>address of unit 3</th></tr>
<tr><td>1st unit</td><td>30</td><td>32</td><td>34</td></tr>
<tr><td>2nd unit</td><td>40</td><td>42</td><td>44</td></tr>
<tr><td>3rd unit</td><td>50</td><td>52</td><td>54</td></tr>
<tr><td>4th unit</td><td>60</td><td>62</td><td>64</td></tr>
<tr><td>5th unit</td><td>70</td><td>72</td><td>72</td></tr>
<tr><td>6th unit</td><td>80</td><td>82</td><td>84</td></tr>
<tr><td>7th unit</td><td>90</td><td>92</td><td>94</td></tr>
<tr><td>8th unit</td><td>100</td><td>102</td><td>104</td></tr>
</table>
</td></tr>
<tr><td><b>iOutChannel0–iOutChannel3</b></td><td>INT</td><td>0–4000 -> 4mA–20mA on the corresponding channel</td></tr>
<tr><th>Output variable</th><th>Data type</th><th>Function</th></tr>
<tr><td><b>bPowerIsOn</b></td><td>BOOL</td><td>Status data of unit (1: ON, 0: OFF)</td></tr>
<tr><td><b>bErrorChannel0–bErrorChannel3</b></td><td></td><td>Status data channel (1: error, 0: normal)</td></tr>
</table>

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

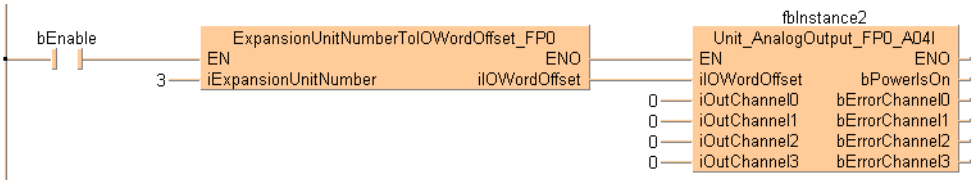
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type
0	VAR	bEnable	BOOL
1	VAR	fbInstance2	E_Unit_AnalogOutput_FP0_A04I

Body If **bEnable** is set to TRUE, the function block converts the digital data (0–4000) of the analog unit FP0-A04I with current output into analog data at the corresponding output channels (4–20mA).

LD



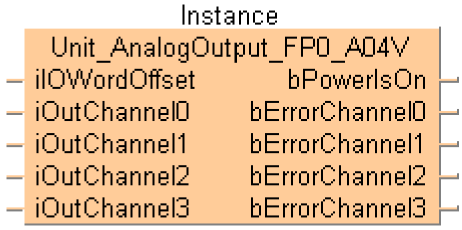
ST
```
IF bEnable THEN
    fbInstance2(iIOWordOffset := iIOOffsetFP0,
                iOutChannel0 := iOut1,
                iOutChannel1 := iOut2,
                iOutChannel2 := iOut3,
                iOutChannel3 := iOut4,
                bPowerIsOn => bOutPower2,
                bErrorChannel0 => bOutError1,
                bErrorChannel1 => bOutError2,
                bErrorChannel2 => bOutError3,
                bErrorChannel3 => bOutError4);
END_IF;
```

Unit_AnalogOutput_FP0_A04V

Reads data from the FP0-A04 unit

Description

This function block reads digital data from the FP0-A04 unit voltage output type from the output channels 0–3 and stores the analog data in the input channels **iOutChannel0–iOutChannel3**. The valid range is from -10–+10 V (-2000–+2000).



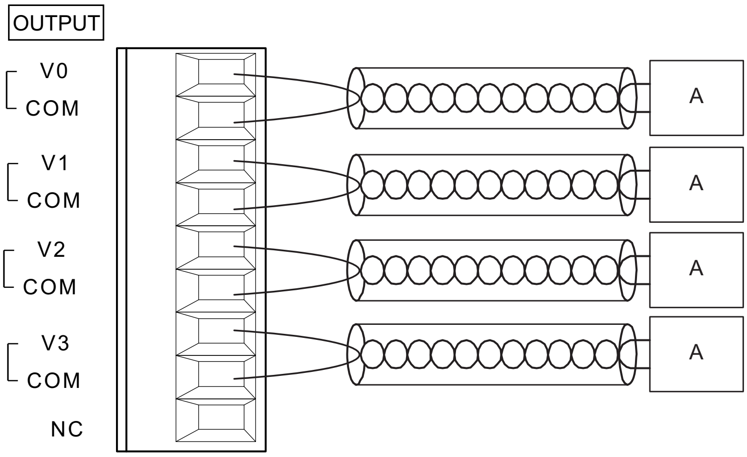
REFERENCE

For technical information, please refer to the manual FP0 Analog unit manual on your FPWIN Pro installation CD.

PLC types see on page 1333

The function block needs two PLC cycle scans to write all four channels into the FP0-A04V Unit. Do not use pulse relay at EN input.

Wiring of analog outputs



A: Analog device

☞

- **The COM contacts are connected internally.**
- **Keep a distance of more than 100mm between the output line and the power line/high-voltage line.**
- **For wiring the analog outputs shielded twisted pair cables are recommended. Connect the shield with the frame ground of the analog unit.**

D/A conversion values

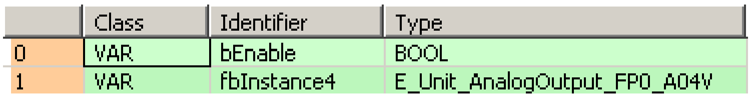
digital input value range: -2000–+2000	**output voltage (V)** range: -10–+10 V
-2000	-10.0
-1500	-7.5
1000	-5.0
-500	-2.5
0	0.0
+500	+2.5
+1000	+5.0
+1500	+7.5
+2000	+10.0
values outside of range	
≤ -2001	constant, the converted value exactly is based on the latest valid input value
≥ +2001	

Data types

<table>
<tr><th>Input variable</th><th>Data type</th><th>Function</th></tr>
<tr><td><b>iIOWordOffset</b></td><td>INT</td><td>The offset of the first WX/WY address of the RTD unit according to its position.
<b>FP0R</b>, <b>FP0</b>, <b>FP-Sigma</b>: (use ExpansionUnitToIOWordOffset_FP0 (see page 1128)) or
unit 1 => address 2, unit 2 => address 4, unit 3 => address 6
<b>FP-X</b>: (use ExpansionUnotToIOWordOffset_FPX_FP0 (see page 1129)) or
<table>
<tr><td>FP0 adapter</td><td>address of unit 1</td><td>address of unit 2</td><td>address of unit 3</td></tr>
<tr><td>1st unit</td><td>30</td><td>32</td><td>34</td></tr>
<tr><td>2nd unit</td><td>40</td><td>42</td><td>44</td></tr>
<tr><td>3rd unit</td><td>50</td><td>52</td><td>54</td></tr>
<tr><td>4th unit</td><td>60</td><td>62</td><td>64</td></tr>
<tr><td>5th unit</td><td>70</td><td>72</td><td>72</td></tr>
<tr><td>6th unit</td><td>80</td><td>82</td><td>84</td></tr>
<tr><td>7th unit</td><td>90</td><td>92</td><td>94</td></tr>
<tr><td>8th unit</td><td>100</td><td>102</td><td>104</td></tr>
</table></td></tr>
<tr><td><b>iOutChannel0–iOutChannel3</b></td><td>INT</td><td>-20002–+000 -> -10V–+10V on the corresponding channel</td></tr>
<tr><th>Output variable</th><th>Data type</th><th>Function</th></tr>
<tr><td><b>bPowerIsOn</b></td><td>BOOL</td><td>Status data of unit (1: ON, 0: OFF)</td></tr>
<tr><td><b>bErrorChannel0–bErrorChannel3</b></td><td></td><td>Status data channel (1: error, 0: normal)</td></tr>
</table>

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

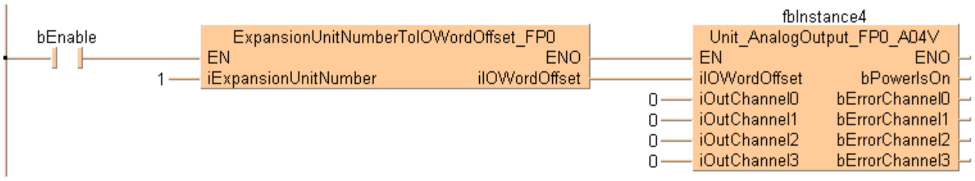
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type
0	VAR	bEnable	BOOL
1	VAR	fbInstance4	E_Unit_AnalogOutput_FP0_A04V

Body If **bEnable** is set to TRUE, the function block converts the digital data (-2000–+2000) of the analog unit FP0-A04V with current output into analog data at the corresponding output channels (-10–+10V).

LD



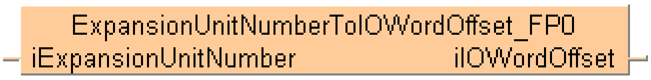
ST When programming with structured text, enter the following:

```
IF bEnable THEN
fbInstance4(iIOWordOffset := iIOOffsetFP0,
                  iOutChannel0 := iOut1,
                  iOutChannel1 := iOut2,
                  iOutChannel2 := iOut3,
                  iOutChannel3 := iOut4,
                  bPowerIsOn => bOutPower2,
                  bErrorChannel0 => bOutError1,
                  bErrorChannel1 => bOutError2,
                  bErrorChannel2 => bOutError3,
                  bErrorChannel3 => bOutError4);
END_IF;
```

ExpansionUnitNumberToIOWordOffset_FP0

Calculate the IO offset of analog units for FP0

Description This instruction calculates the word offset using an FP0 analog unit connected to an FP0.



REFERENCE

For technical information, please refer to the manual FP0 RTD Unit ACGM0159 on your FPWIN Pro installation CD.

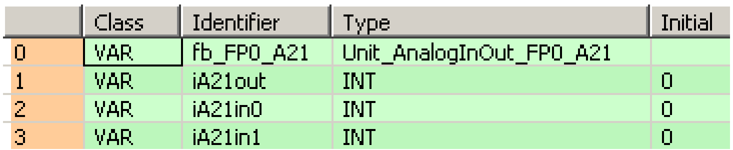
PLC types see see page 1320

Data types

Input variable	Data type	Function
iExpansionUnitNumber	INT	FP0 expansion unit number 1–3
Output variable		
iIOWordOffset	INT	Offset of the I/O word (WX/WY 2/4/6)

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). (ST).The same POU header is used for all programming languages.

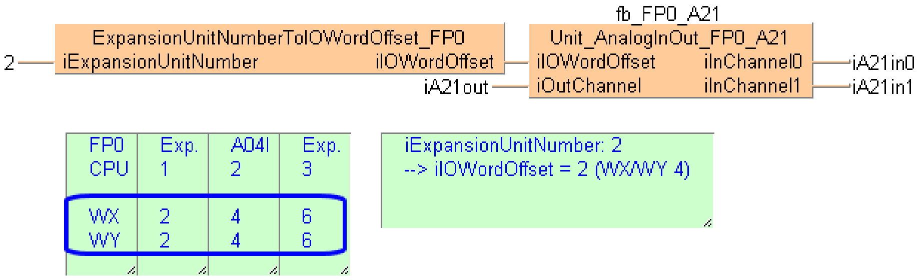
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	fb_FP0_A21	Unit_AnalogInOut_FP0_A21	
1	VAR	iA21out	INT	0
2	VAR	iA21in0	INT	0
3	VAR	iA21in1	INT	0

Body The function block converts the FP0 expansion unit position number 1–3, where 0 is the FP0 or FPΣ CPU, to the corresponding I/O word offset 2, 4 and 6.

LD



FP0 CPU	Exp. 1	A04I 2	Exp. 3
WX WY	2 2	4 4	6 6

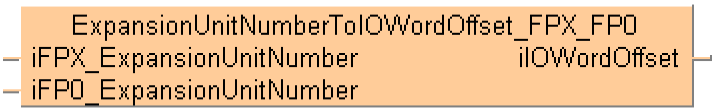
iExpansionUnitNumber: 2
--> iIOWordOffset = 2 (WX/WY 4)

ST
```
Unit_AnalogInOut_FP0_A21(iIOWordOffset :=
ExpansionUnitNumberToIOWordOffset_FP0(2),
                        iOutChannel := iA21out,
                        iInChannel0 => iA21in0,
                        iInChannel1 => iA21in1);
```

ExpansionUnitNumberToIOWordOffset_FPX_FP0

Calculates the IO offset of analog units for FP-X

Description This instruction calculates the word offset using an FP0 analog unit connected to an FP-X or FP-X0.



REFERENCE

For technical information, please refer to the manual FP0 RTD Unit ACGM0159 on your FPWIN Pro installation CD.

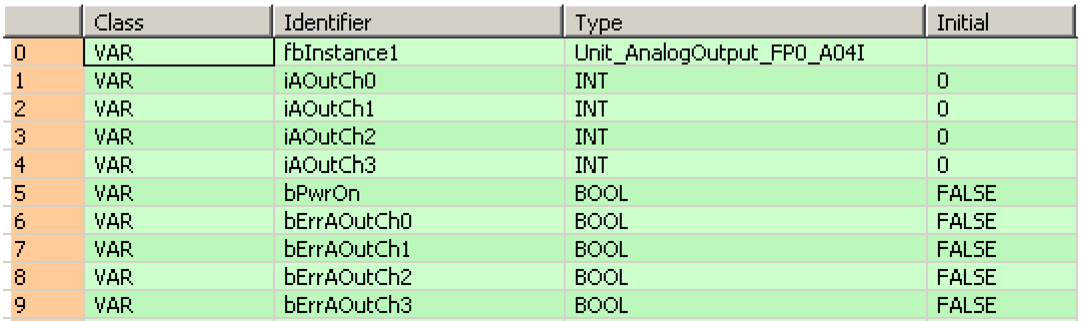
PLC types see see page 1320

Data types

Input variable	Data type	Function
iFPX_ExpansionUnitNumber	INT	FP-X expansion unit number 1–8
iFP0_ExpansionUnitNumber		FP0 expansion unit number 1–3
Output variable		
iIOWordOffset	INT	Offset of the I/O word (WX/WY)

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

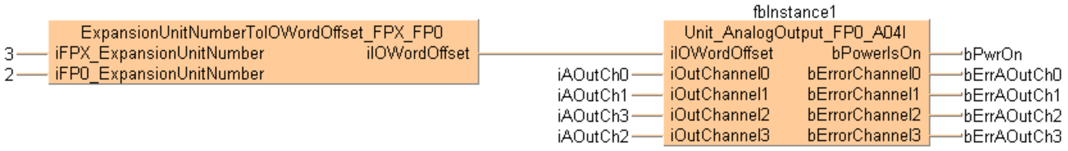
POU header All input and output variables used for programming this function have been declared in the POU header.

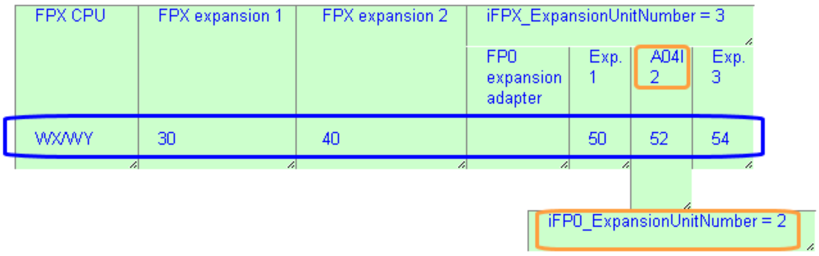


	Class	Identifier	Type	Initial
0	VAR	fbInstance1	Unit_AnalogOutput_FP0_A04I	
1	VAR	iAOutCh0	INT	0
2	VAR	iAOutCh1	INT	0
3	VAR	iAOutCh2	INT	0
4	VAR	iAOutCh3	INT	0
5	VAR	bPwrOn	BOOL	FALSE
6	VAR	bErrAOutCh0	BOOL	FALSE
7	VAR	bErrAOutCh1	BOOL	FALSE
8	VAR	bErrAOutCh2	BOOL	FALSE
9	VAR	bErrAOutCh3	BOOL	FALSE

Body This instruction evaluates the IO word offset for an FP0 analog unit connected to an FP-X CPU. The input **iFPX_ExpansionUnitNumber** is the FP-X unit position number (1–8) to which the FP0 expansion adapter is connected. The input **iFP0_ExpansionUnitNumber** is the FP0 analog unit position number (1–3) connected next to the FP0 expansion adapter.

LD





FPX CPU	FPX expansion 1	FPX expansion 2	iFPX_ExpansionUnitNumber = 3			
			FP0 expansion adapter	Exp. 1	A04I 2	Exp. 3
WX/WY	30	40		50	52	54

iFP0_ExpansionUnitNumber = 2

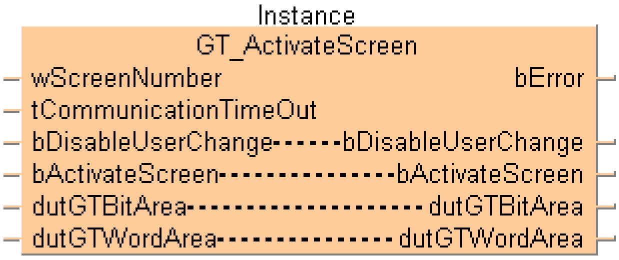
```
ST Unit_AnalogOutput_FP0_A04I(iIOWordOffset :=
   ExpansionUnitNumberToIOWordOffset_FPX_FP0(3, 2),
                           iOutChannel0 := iAOutCh0,
                           iOutChannel1 := iAOutCh1,
                           iOutChannel2 := iAOutCh2,
                           iOutChannel3 := iAOutCh2,
                           bPowerIsOn => bPwrOn,
                           bErrorChannel0 => bErrAOutCh0,
                           bErrorChannel1 => bErrAOutCh1,
                           bErrorChannel2 => bErrAOutCh2,
                           bErrorChannel3 => bErrAOutCh3);
```

Chapter 37

GT panel instructions

GT_ActivateScreen Control the GT panel screen

Description Function block to activate or change a specified GT screen from the PLC using the variables described in the table for data types.



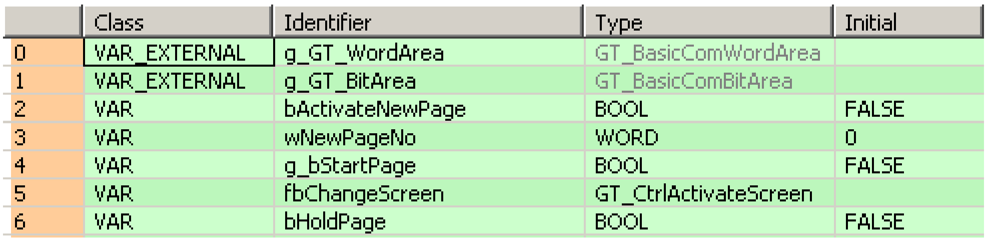
PLC types see page 1327

Data types

Input variable	Data type	Function
wScreenNumber	WORD	Screen number
tComTimeOut	TIME	Communication timeout
Input/output variable	**Data type**	**Function**
bDisableUserChange	BOOL	Disable screen change by touch operation on GT
bActivateScreen		Activate new screen
dutGTBitArea	GT_BasicComBitArea	GT basic communication bit area
dutGTWordArea	GT_BasicComWordArea	GT basic communication word area
Output variable	**Data type**	**Function**
bError	BOOL	Turns on when the screen is not switched within the communication timeout

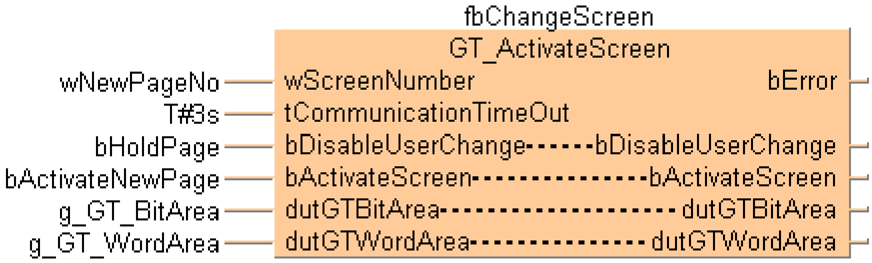
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR_EXTERNAL	g_GT_WordArea	GT_BasicComWordArea	
1	VAR_EXTERNAL	g_GT_BitArea	GT_BasicComBitArea	
2	VAR	bActivateNewPage	BOOL	FALSE
3	VAR	wNewPageNo	WORD	0
4	VAR	g_bStartPage	BOOL	FALSE
5	VAR	fbChangeScreen	GT_CtrlActivateScreen	
6	VAR	bHoldPage	BOOL	FALSE

LD

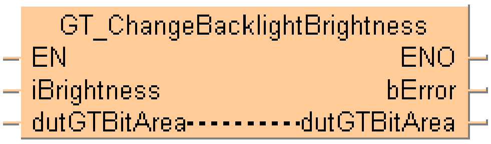


```
ST fb_GT_ActivateScreen(wScreenNum := wNewPageNo,
                                        tComTimeOut := T#3s,
                                        bDisableUserChange := bHoldPage,
                                        bActivateScreen := bActivateNewPage,
                                        dutGTBitArea := g_GT_BitArea,
                                        dutGTWordArea := g_GT_WordArea,
                                        bErrorActivateScreen =>
   bErrorActivateScreen);
```

GT_ChangeBacklight Brightness

Changes the backlight brightness of a GT panel

Description This instruction changes the backlight brightness of the GT Panel using the variables described in the table for data types.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

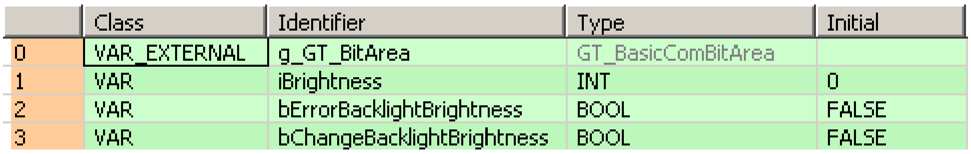
PLC types see page 1327

Data types

Input variable	Data type	Function
iBrightness	INT	Brightness value 0–15
Input/output variable		
dutGTBitArea	GT_BasicComBitArea	GT basic communication bit area
Output variable		
bError	BOOL	Turns on if the brightness value is out of range

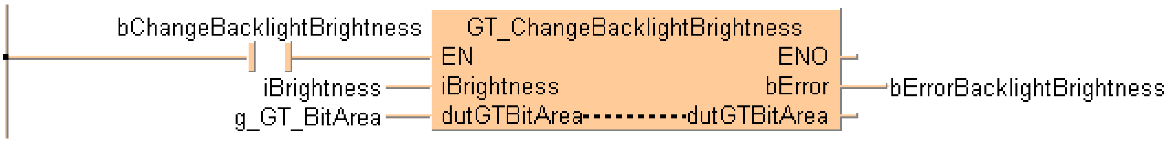
Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST).The same POU header is used for all programming languages.

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR_EXTERNAL	g_GT_BitArea	GT_BasicComBitArea	
1	VAR	iBrightness	INT	0
2	VAR	bErrorBacklightBrightness	BOOL	FALSE
3	VAR	bChangeBacklightBrightness	BOOL	FALSE

LD


bChangeBacklightBrightness
GT_ChangeBacklightBrightness
EN
ENO
iBrightness
iBrightness
bError
bErrorBacklightBrightness
g_GT_BitArea
dutGTBitArea
dutGTBitArea

ST
```
fb_GT_ChangeBacklightBrightness((* EN := TRUE, *)
                              iBrightness := iBrightness,
                              dutGTBitArea := g_GT_BitArea,
                              bError => bErrorBacklightBrightness
                               (* , ENO => ?BOOL? *));
```

Chapter 38

High-speed counter instructions

38.1 Introduction

Control FPWIN Pro offers two concepts for programming with high-speed counter instructions:

- FP instructions
- Tool instructions

For users programming for different PLC types of the FP series or users who are tired of setting control code bits and looking up available channel numbers, the tool instructions offer new and comfortable features. These include information functions for evaluating status flags and settings, control functions for configuring high-speed counters and pulse outputs, PLC-independent functions and DUTs, as well as variable channel numbers. However, the FP instructions may be easier to use for beginners or users familiar with FPWIN GR.

Most of the information, which is accessible via information and control functions, is stored in special internal relays and special data registers. These relays and registers can also be accessed using PLC-independent system variables.

To take advantage of the features you prefer, the instructions of both libraries can be mixed.

☞ ◆NOTE

When programming with the tool instructions, be sure to refer to the detailed information provided via the links to the related F/P instructions.

Main features	FP instructions	Tool instructions
Pre version 6.4 support	●	
Use of inline functions	●	
Use of FPWIN GR function names	●	
Less code with constant channel numbers	●	
Control codes	●	
Control functions		●
Information functions		●
Variable channel numbers		●
Universal functions for all PLCs		●
DUT for common channel configuration for all PLCs for all pulse output instructions		●

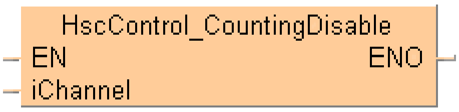
38.2 High-speed counter control instructions

In this section:

HscControl_Counting Disable

Disables counting on a high-speed counter channel

Description This instruction disables counting on the high-speed counter channel specified by **iChannel**. Bit 1 of the high-speed counter control code (see page 891) is set to TRUE.



See also:

- Tool instructions: overview of high-speed counter instructions
- HscControl_CountingEnable (see page 1141)
- HscInfo_IsCountingDisabled (see page 1159)

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

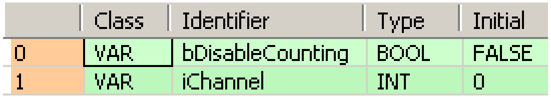
PLC types see page 1327

Data types

Variable	Data type	Function
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

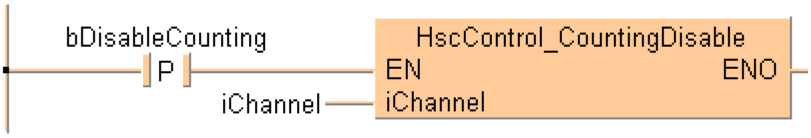
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bDisableCounting	BOOL	FALSE
1	VAR	iChannel	INT	0

Body When the variable **bDisableCounting** changes from FALSE to TRUE, counting on the channel specified by **iChannel** is disabled.

LD



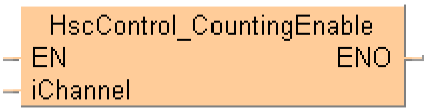
ST When programming with structured text, enter the following:

```
if DF(bDisableCounting) then
    HscControl_CountingDisable(iChannel := iChannel);
end_if;
```

HscControl_Counting Enable

Enables counting on a high-speed counter channel

Description This instruction enables counting on the high-speed counter channel specified by **iChannel** after counting has been disabled with HscControl_CountingDisable (see page 1139). Bit 1 of the high-speed counter control code (see page 891) is set to FALSE.



See also:

- Tool instructions: overview of high-speed counter instructions
- HscControl_CountingDisable (see page 1139)
- HscInfo_IsCountingDisabled (see page 1159)

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

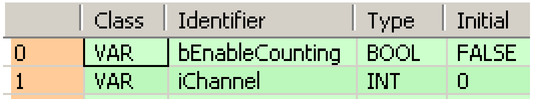
PLC types see page 1327

Data types

Variable	Data type	Function
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

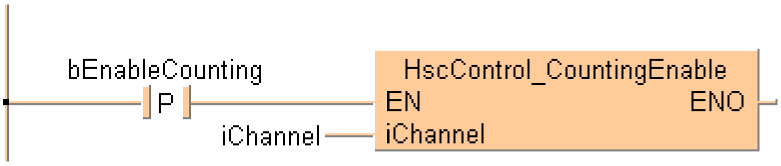
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bEnableCounting	BOOL	FALSE
1	VAR	iChannel	INT	0

Body When the variable **bEnableCounting** changes from FALSE to TRUE, counting on the channel specified by **iChannel** is enabled.

LD



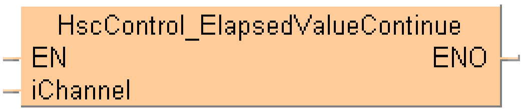
ST When programming with structured text, enter the following:

```
if DF(bEnableCounting) then
    HscControl_CountingEnable(iChannel := iChannel);
end_if;
```

HscControl_ElapsedValueContinue

Continues counting after reset

Description This instruction resumes counting on the channel specified by **iChannel** after a reset of the elapsed value using HscControl_ElapsedValueReset (see page 1145). Bit 0 of the high-speed counter control code (see page 891) is set to FALSE.



See also:

- Tool instructions: overview of high-speed counter instructions
- HscControl_WriteElapsedValue (see page 1152)
- HscInfo_ReadElapsedValue (see page 1163)
- HscInfo_IsElapsedValueReset (see page 1160)

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

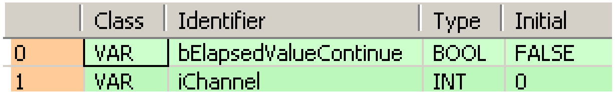
PLC types see page 1327

Data types

Variable	Data type	Function
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

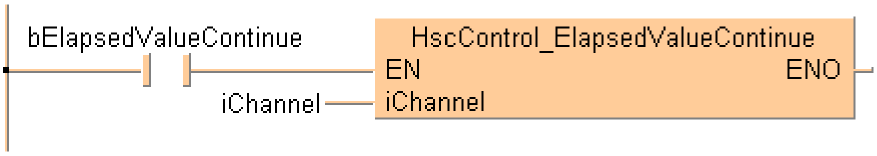
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bElapsedValueContinue	BOOL	FALSE
1	VAR	iChannel	INT	0

Body When the variable **bElapsedValueContinue** is set to TRUE, counting resumes on the channel specified by **iChannel**.

LD



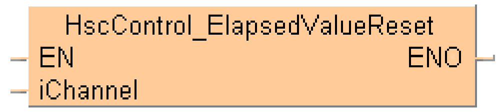
ST When programming with structured text, enter the following:

```
if (bElapsedValueContinue) then
    HscControl_ElapsedValueContinue(iChannel := iChannel);
end_if;
```

HscControl_Elapsed ValueReset

Sets elapsed value to 0

Description This instruction sets the elapsed value of the high-speed counter channel specified by **iChannel** to 0. Bit 0 of the high-speed counter control code (see page 891) is set to TRUE.



See also:

- Tool instructions: overview of high-speed counter instructions
- HscControl_WriteElapsedValue (see page 1152)
- HscControl_ElapsedValueContinue (see page 1143)
- HscInfo_IsElapsedValueReset (see page 1160)

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

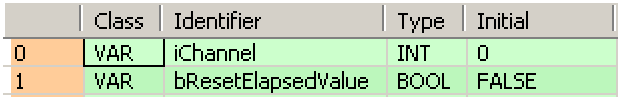
PLC types see page 1327

Data types

Variable	Data type	Function
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

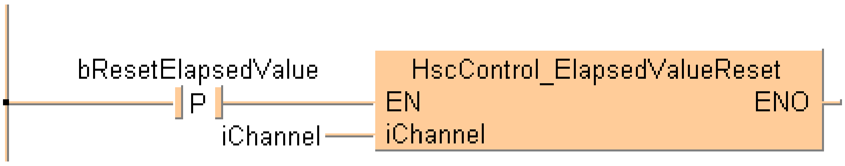
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	bResetElapsedValue	BOOL	FALSE

Body When the variable **bResetElapsedValue** changes from FALSE to TRUE, the elapsed value on the channel specified by **iChannel** is set to 0.

LD



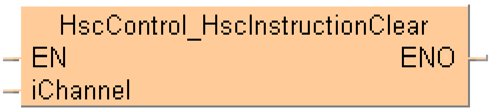
ST When programming with structured text, enter the following:

```
if (bResetElapsedValue) then
    HscControl_ElapsedValueReset(iChannel := iChannel);
end_if;
```

HscControl_HscInstruction Clear

Clears high-speed counter instruction

Description This instruction cancels the execution of a high-speed counter instruction on the channel specified by **iChannel**. Bit 3 of the high-speed counter control code (see page 891) is set to TRUE and subsequently reset to FALSE.



See also:

- Tool instructions: overview of high-speed counter instructions

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

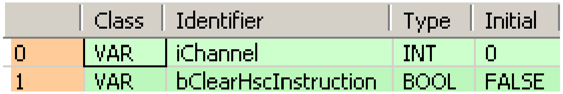
PLC types see page 1327

Data types

Variable	Data type	Function
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

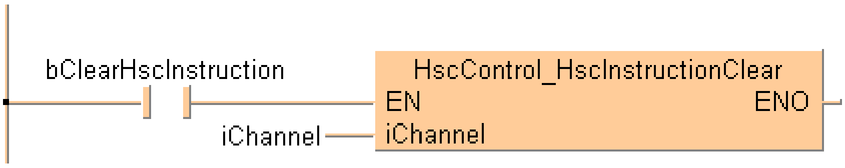
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	bClearHscInstruction	BOOL	FALSE

Body When the variable **bClearHscInstruction** is set to TRUE, the execution of a high-speed counter instruction on the channel specified by **iChannel** is canceled.

LD



ST

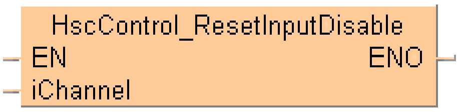
When programming with structured text, enter the following:

```
if (bClearHscInstruction) then
    HscControl_HscInstructionClear(iChannel := iChannel);
end_if;
```

HscControl_ResetInputDisable

Disables reset input

Description This instruction disables the reset input of the high-speed counter channel specified by **iChannel**. Bit 2 of the high-speed counter control code (see page 891) is set to TRUE.



See also:

- Tool instructions: overview of high-speed counter instructions
- HscInfo_IsResetInputDisabled (see page 1161)

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

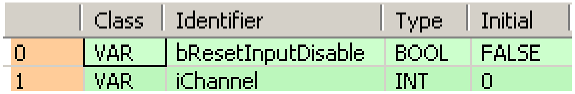
PLC types see page 1327

Data types

Variable	Data type	Function
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

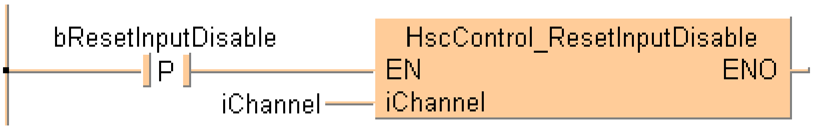
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bResetInputDisable	BOOL	FALSE
1	VAR	iChannel	INT	0

Body When the variable **bResetInputDisable** changes from FALSE to TRUE, the reset input of the channel specified by **iChannel** is disabled.

LD



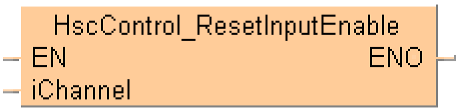
ST When programming with structured text, enter the following:

```
if DF(bResetInputDisable) then
    HscControl_ResetInputDisable(iChannel := iChannel);
end_if;
```

HscControl_ResetInputEnable

Enables reset input

Description This instruction enables the reset input of the channel specified by **iChannel**. Bit 2 of the high-speed counter control code (see page 891) is set to FALSE.



See also:

- Tool instructions: overview of high-speed counter instructions
- HscInfo_IsResetInputDisabled (see page 1161)

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

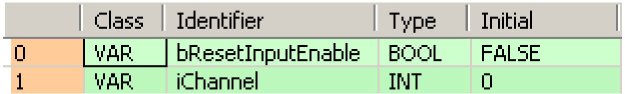
PLC types see page 1327

Data types

Variable	Data type	Function
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

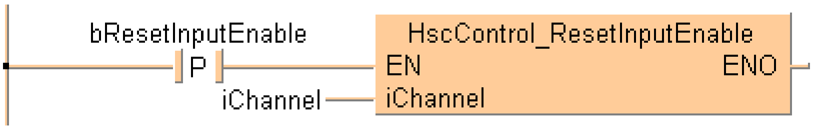
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bResetInputEnable	BOOL	FALSE
1	VAR	iChannel	INT	0

Body When the variable **bResetInputEnable** changes from FALSE to TRUE, the reset input of the channel specified by **iChannel** is enabled.

LD



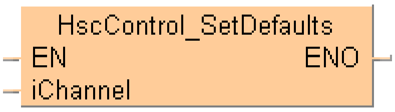
ST When programming with structured text, enter the following:

```
if DF(bResetInputEnable) then
    HscControl_ResetInputEnable(iChannel := iChannel);
end_if;
```

HscControl_SetDefaults

Sets defaults for high-speed counter channel

Description This instruction sets all bits of the high-speed counter control code (see page 891) of the channel specified by **iChannel** to 0. 0 is the default setting.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

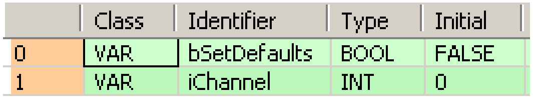
PLC types see page 1327

Data types

Variable	Data type	Function
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

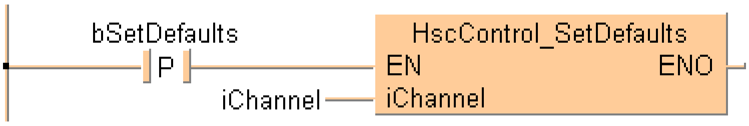
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bSetDefaults	BOOL	FALSE
1	VAR	iChannel	INT	0

Body When the variable **bSetDefaults** changes from FALSE to TRUE, all settings of the channel specified by **iChannel** are set to their default values.

LD



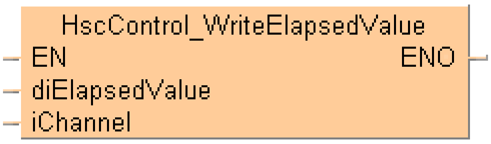
ST When programming with structured text, enter the following:

```
if DF(bSetDefaults) then
    HscControl_SetDefaults(iChannel := iChannel);
end_if;
```

HscControl_Write ElapsedValue

Writes elapsed value into high-speed counter channel

Description This instruction writes an elapsed value into the high-speed counter channel specified by **iChannel**.



HscControl_WriteElapsedValue
EN ENO
diElapsedValue
iChannel

See also:

- Tool instructions: overview of high-speed counter instructions
- HscControl_ElapsedValueReset (see page 1145)
- HscInfo_ReadElapsedValue (see page 1163)
- HscInfo_IsElapsedValueReset (see page 1160)
- FP instructions Writing and reading the elapsed value (see page 894)

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

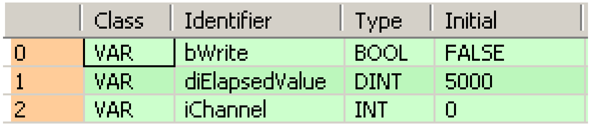
PLC types see page 1327

Data types

Variable	Data type	Function
diElapsedValue	DINT	Elapsed value to be written into the channel specified by **iChannel**
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

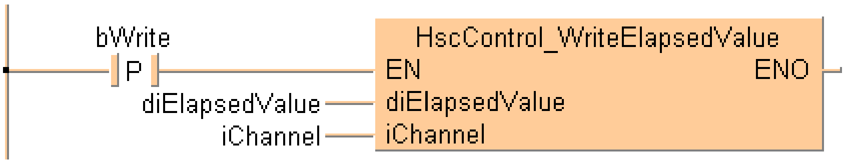
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bWrite	BOOL	FALSE
1	VAR	diElapsedValue	DINT	5000
2	VAR	iChannel	INT	0

Body When the variable **bWrite** changes from FALSE to TRUE, the elapsed value specified by **diElapsedValue** is written into the channel specified by **iChannel**.

LD



ST When programming with structured text, enter the following:

```
if DF(bWrite) then
    HscControl_WriteElapsedValue(diElapsedValue := diElapsedValue,
    iChannel := iChannel);
end_if;
```

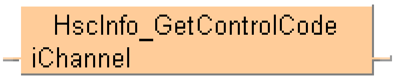
38.3 High-speed counter information instructions

In this section:

HscInfo_GetControlCode

Returns control code of high-speed counter channel

Description This instruction returns the control code (see page 891) of the high-speed counter channel specified by **iChannel**.



See also:

Tool instructions: overview of high-speed counter instructions

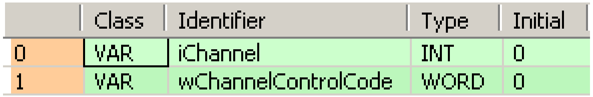
PLC types see page 1327

Data types

Variable	Data type	Function
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	WORD	Stores the control code

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

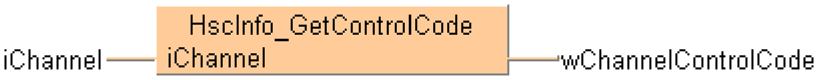
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	wChannelControlCode	WORD	0

Body

LD



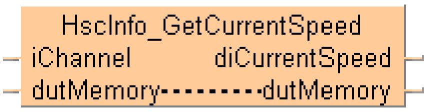
ST When programming with structured text, enter the following:

```
if (bGetControlCode) then
    wChannelControlCode := HscInfo_GetControlCode(iChannel := iChannel);
end_if;
```

HscInfo_GetCurrentSpeed

Returns current speed of high-speed counter channel

Description This instruction returns the current speed in Hz of the high-speed counter channel specified by **iChannel**.



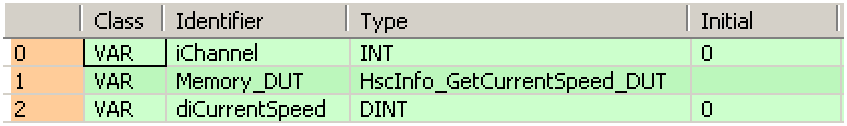
PLC types see page 1327

Data types

Input variable	Data type	Function
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
dutMemory	DUT (see page 51)	HscInfo_GetCurrentSpeed_DUT
Output variable		
diCurrentSpeed	DINT	Stores the current speed of the channel specified by **iChannel**

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

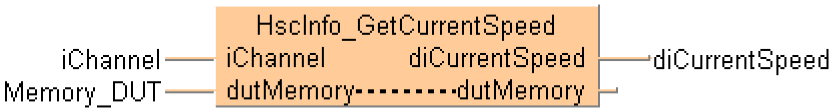
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	Memory_DUT	HscInfo_GetCurrentSpeed_DUT	
2	VAR	diCurrentSpeed	DINT	0

Body

LD



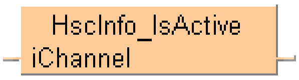
ST When programming with structured text, enter the following:

```
if (bGetCurrentSpeed) then
    HscInfo_GetCurrentSpeed(iChannel := iChannel,
    dutMemory := Memory_DUT,
    diCurrentSpeed => diCurrentSpeed);
end_if;
```

HscInfo_IsActive

Checks if high-speed counter is active

Description This instruction evaluates the high-speed counter control flag and returns TRUE if the high-speed counter channel specified by **iChannel** is active.



See also:

- Tool instructions: overview of high-speed counter instructions
- HscControl_HscInstructionClear (see page 1147)

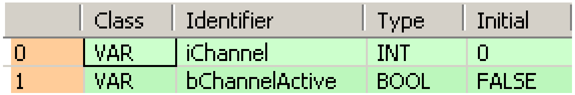
PLC types see page 1327

Data types

Variable	Data type	Function
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	BOOL	TRUE if the high-speed counter channel specified by **iChannel** is active

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

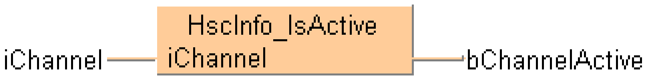
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	bChannelActive	BOOL	FALSE

Body

LD



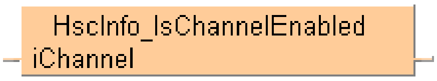
ST When programming with structured text, enter the following:

```
bChannelActive := HscInfo_IsActive(iChannel := iChannel);
```

HscInfo_IsChannel Enabled

Checks if high-speed counter channel is enabled

Description This instruction returns TRUE if the high-speed counter channel specified by **iChannel** has been enabled in the system registers and is supported by the selected PLC type.



See also:

- Tool instructions: overview of high-speed counter instructions
- Required system register settings

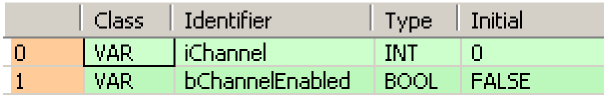
PLC types see page 1327

Data types

Variable	Data type	Function
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	BOOL	TRUE if the channel specified by **iChannel** is enabled

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

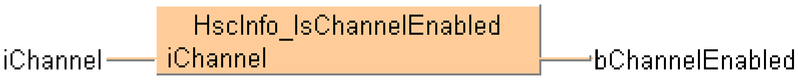
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	bChannelEnabled	BOOL	FALSE

Body

LD



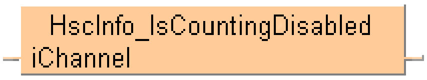
ST When programming with structured text, enter the following:

```
if (bChannelEnabled_Check) then
  bChannelEnabled := HscInfo_IsChannelEnabled(iChannel := iChannel);
end_if;
```

HscInfo_IsCountingDisabled

Checks if counting is disabled

Description This instruction returns TRUE if counting on the channel specified by **iChannel** has been disabled.



See also:

- Tool instructions: overview of high-speed counter instructions
- HscControl_CountingDisable (see page 1139)
- HscControl_CountingEnable (see page 1141)
- FP instructions Enabling/disabling counting operations (see page 891)

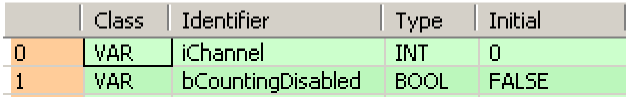
PLC types see page 1327

Data types

Variable	Data type	Function
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	BOOL	TRUE if counting on the channel specified by **iChannel** is disabled

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

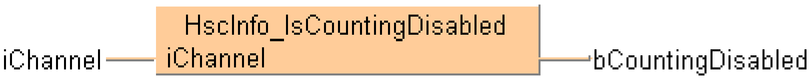
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	bCountingDisabled	BOOL	FALSE

Body

LD



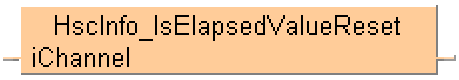
ST When programming with structured text, enter the following:

```
if (bCountingDisabled_Check) then
  bCountingDisabled := HscInfo_IsCountingDisabled(iChannel := iChannel);
end_if;
```

HscInfo_IsElapsedValueReset

Checks if elapsed value is set to 0

Description This instruction returns TRUE if the elapsed value of the high-speed counter channel specified by **iChannel** has been reset to 0.



See also:

- Tool instructions: overview of high-speed counter instructions
- HscControl_ElapsedValueReset (see page 1145)
- HscControl_ElapsedValueSet (see page 1143)
- FP instructions Resetting the elapsed value (software reset) (see page 891)

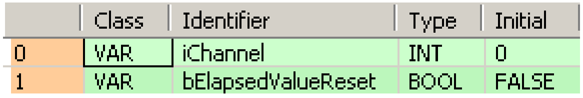
PLC types see page 1327

Data types

Variable	Data type	Function
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	BOOL	TRUE if the channel specified by **iChannel** has been reset

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

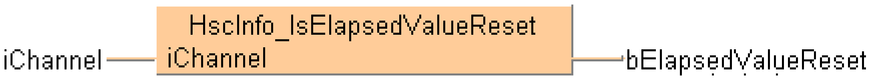
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	bElapsedValueReset	BOOL	FALSE

Body

LD



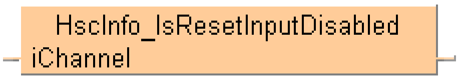
ST When programming with structured text, enter the following:

```
if (bElapsedValueReset_Check) then
 bElapsedValueReset := HscInfo_IsElapsedValueReset(iChannel := iChannel);
end_if;
```

HscInfo_IsResetInputDisabled

Checks if reset input is disabled

Description This instruction returns TRUE if the reset input of the channel specified by **iChannel** has been disabled.



See also:

- Tool instructions: overview of high-speed counter instructions
- HscControl_ResetInputEnable (see page 1151)
- HscControl_ResetInputDisable (see page 1149)
- FP instructions Enabling/disabling the reset input (hardware reset) (see page 891)

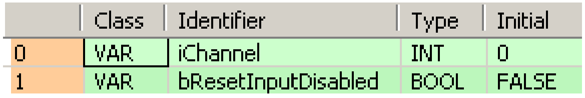
PLC types see page 1327

Data types

Variable	Data type	Function
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	BOOL	TRUE if the reset input of the channel specified by **iChannel** is disabled

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

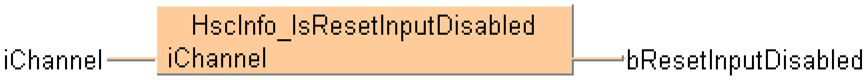
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	bResetInputDisabled	BOOL	FALSE

Body

LD



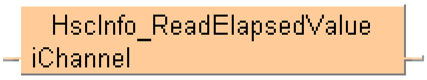
ST When programming with structured text, enter the following:

```
if (bResetInput_Check) then
    bResetInputDisabled := HscInfo_IsResetInputDisabled(iChannel :=
iChannel);
end_if;
```

HscInfo_ReadElapsedValue

Reads elapsed value from high-speed counter channel

Description This instruction reads the elapsed value from the high-speed counter channel specified by **iChannel**.



See also:

- Tool instructions: overview of high-speed counter instructions
- HscControl_WriteElapsedValue (see page 1152)
- HscControl_ElapsedValueReset (see page 1145)
- HscControl_ElapsedValueContinue (see page 1143)
- FP instructions Writing and reading the elapsed value (see page 894)

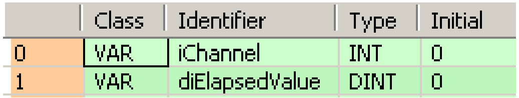
PLC types see page 1327

Data types

Variable	Data type	Function
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	DINT	Stores the elapsed value from the channel specified by **iChannel**

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

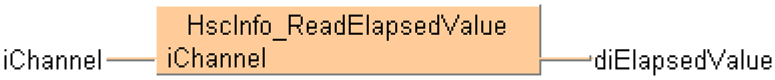
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	diElapsedValue	DINT	0

Body

LD



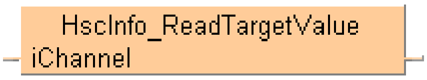
ST When programming with structured text, enter the following:

```
if (bReadElapsedValue) then
    diElapsedValue := HscInfo_ReadElapsedValue(iChannel := iChannel);
end_if;
```

HscInfo_ReadTarget Value

Reads target value from high-speed counter channel

Description This instruction reads the target value from the high-speed counter channel specified by **iChannel**.



See also:

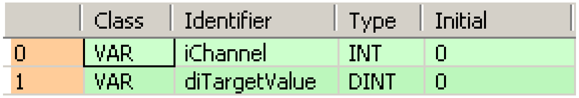
Tool instructions: overview of high-speed counter instructions

Data types

Variable	Data type	Function
iChannel	INT	High-speed counter channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	DINT	Stores the target value of the channel specified by **iChannel**

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

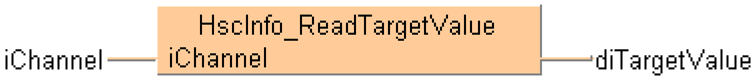
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	diTargetValue	DINT	0

Body

LD



ST When programming with structured text, enter the following:

```
if (bReadTargetValue) then
    diTargetValue := HscInfo_ReadTargetValue(iChannel := iChannel);
end_if;
```

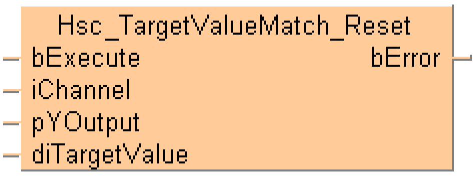
38.4 High-speed counter target value match control

In this section:

Hsc_TargetValue Match_Reset

Target value match OFF (high-speed counter)

Description If the elapsed value matches the target value **diTargetValue** of the high-speed counter channel specified by **iChannel**, the output relay specified by **pYOutput** immediately turns to FALSE.



This non-inline instruction is part of the tool instructions for high-speed counters. For a detailed description of the instruction(s) used internally, please refer to the online help: F167_HighSpeedCounter_Reset (see page 904)

To validate the combination of channel and Y output, the compiler requires the following name pattern for global variables:

%sHsc_TargetValueMatch_Channel%d_Y%d%s

Always use this pattern for global variables in target value match control.

- Channel%d must be a high-speed counter channel number enabled in the system registers
- Y%d must be an explicit output address supported by the PLC

FP-Σ, FP0, FP-e:	Y0–Y7
FP-Σ (V3.1 or higher), FP0R:	Y0–Y1F
FP-X:	Y0–Y29F

- %s is an optional descriptor at the beginning and the end of the pattern

Optiona l	Fixed pattern	Optional
g_b	Hsc_TargetValueMatch_ChannelA_Y11F	_MotorOn

This global variable generates the code for channel **A** and output **Y11F**.

PLC types See 1327

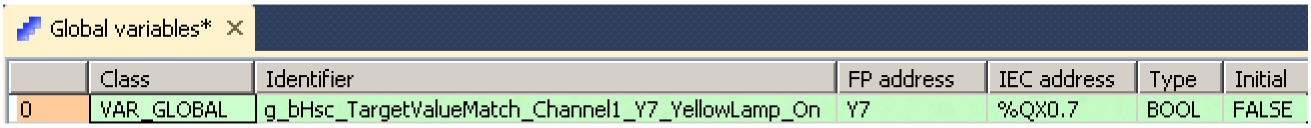
Data types

Input variable	Data type	Function
bExecute	BOOL	A rising edge activates the function; evaluate the high-speed counter control flag using HscInfo_IsActive (see page 1157)
iChannel	INT	FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0, FP-e: 0, 1
pYOutput	POINTER	Pointer result obtained by GetPointer from a global variable that supplies the channel number and output relay
diTargetValue	DINT	Specify a 32-bit data value for the target value within the following range: FP0, FP-e: -838808–+8388607 FPΣ, FP-X, FP0R: -2147483467–+2147483648

Output variable	Data type	Function
iError	BOOL	TRUE if the combination of **Channel%d** and **pYOuput.iOffset** does not match a valid combination of channel number and output relay as determined by the global variable

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). (ST).The same POU header is used for all programming languages.

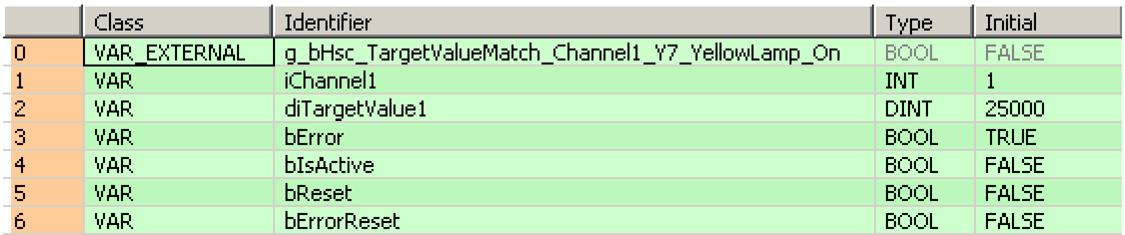
GVL In the global variable list you define variables that can be accessed by all POUs in the project.



Global variables*

	Class	Identifier	FP address	IEC address	Type	Initial
0	VAR_GLOBAL	g_bHsc_TargetValueMatch_Channel1_Y7_YellowLamp_On	Y7	%QX0.7	BOOL	FALSE

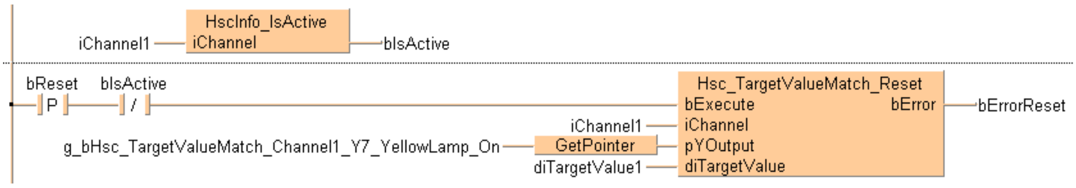
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR_EXTERNAL	g_bHsc_TargetValueMatch_Channel1_Y7_YellowLamp_On	BOOL	FALSE
1	VAR	iChannel1	INT	1
2	VAR	diTargetValue1	DINT	25000
3	VAR	bError	BOOL	TRUE
4	VAR	bIsActive	BOOL	FALSE
5	VAR	bReset	BOOL	FALSE
6	VAR	bErrorReset	BOOL	FALSE

Body Use HscInfo_IsActive (see page 1157) to evaluate the channel specified by **iChannel1**. If a rising edge is detected at **bReset** and if **bIsActive** is **not** TRUE, the instruction is executed. The combination of channel number and output contact is validated in the global variable **g_bHsc_TargetValueMatch_Channel1_Y7_YellowLamp_On**. When the high-speed counter on channel 1 reaches the target value **diTargetValue1**, output **Y7** is set to FALSE.

LD

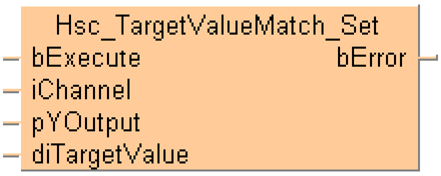


ST
```
bIsActive:=HscInfo_IsActive(iChannel1);

Hsc_TargetValueMatch_Reset(bExecute := DF(bReset) AND NOT bIsActive,
                iChannel := iChannel1,
                pYOutput :=
GetPointer(g_bHsc_TargetValueMatch_Channel1_Y7_YellowLamp_On),
                diTargetValue := diTargetValue1,
                bError => bErrorReset);
```

Hsc_TargetValue Match_Set

Target value match ON (high-speed couter)

Description If the elapsed value matches the target value **diTargetValue** of the high-speed counter channel specified by **iChannel**, the output relay specified by **pYOutput** immediately turns to TRUE.



This non-inline instruction is part of the tool instructions for high-speed counters. For a detailed description of the instruction(s) used internally, please refer to the online help: F166_HighSpeedCounter_Set (see page 900)

To validate the combination of channel and Y output, the compiler requires the following name pattern for global variables:
%sHsc_TargetValueMatch_Channel%d_Y%d_%s

Always use this pattern for global variables in target value match control.

- Channel%d must be a high-speed counter channel number enabled in the system registers
- Y%d must be an explicit output address supported by the PLC

FP-Σ, FP0, FP-e:	Y0–Y7
FP-Σ (V3.1 or higher), FP0R:	Y0–Y1F
FP-X:	Y0–Y29F

- _%s is an optional descriptor at the beginning and the end of the pattern

Optional	Fixed pattern	Optional
g_b	Hsc_TargetValueMatch_ChannelA_Y11F	_MotorOn

This global variable generates the code for channel **A** and output **Y11F**.

PLC types See 1327

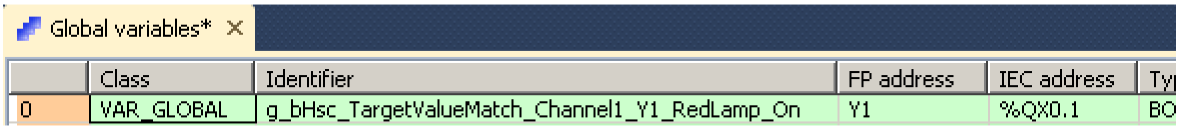
Data types

Input variable	Data type	Function
bExecute	BOOL	A rising edge activates the function; evaluate the high-speed counter control flag using HscInfo_IsActive (see page 1157)
iChannel	INT	FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
pYOutput	POINTER	Pointer result obtained by GetPointer from a global variable that supplies the channel number and output relay
diTargetValue	DINT	Specify a 32-bit data value for the target value within the following range: FP0, FP-e: -838808–+8388607 FPΣ, FP-X, FP0R: -2147483467–+2147483648

Output variable	Data type	Function
bError	BOOL	TRUE if the combination of **Channel%d** and **pYOuput.iOffset** does not match a valid combination of channel number and output relay as determined by the global variable

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). (ST).The same POU header is used for all programming languages.

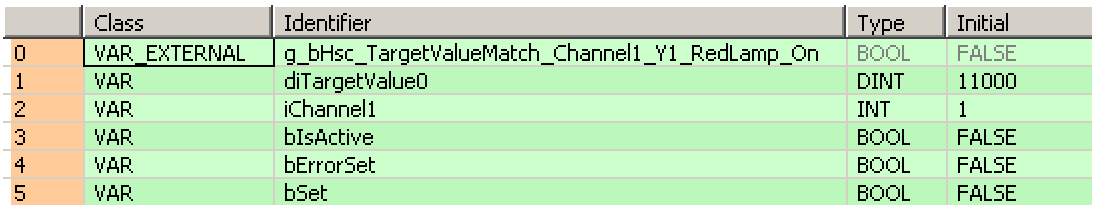
GVL In the global variable list you define variables that can be accessed by all POUs in the project.



Global variables* ×

	Class	Identifier	FP address	IEC address	Ty
0	VAR_GLOBAL	g_bHsc_TargetValueMatch_Channel1_Y1_RedLamp_On	Y1	%QX0.1	BO

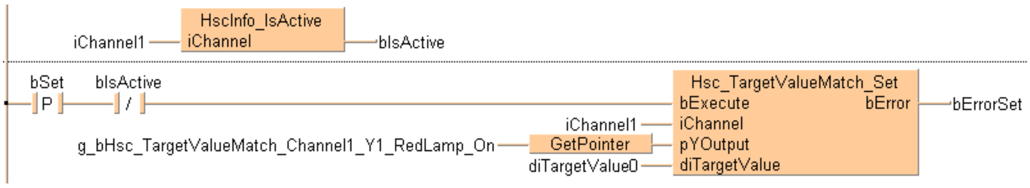
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR_EXTERNAL	g_bHsc_TargetValueMatch_Channel1_Y1_RedLamp_On	BOOL	FALSE
1	VAR	diTargetValue0	DINT	11000
2	VAR	iChannel1	INT	1
3	VAR	bIsActive	BOOL	FALSE
4	VAR	bErrorSet	BOOL	FALSE
5	VAR	bSet	BOOL	FALSE

Body Use HscInfo_IsActive (see page 1157) to evaluate the channel specified by **iChannel1**. If a rising edge is detected at **bSet** and if **bIsActive** is **not** TRUE, the instruction is executed. The combination of channel number and output contact is validated in the global variable **g_bHsc_TargetValueMatch_Channel1_Y1_RedLamp_On**. When the high-speed counter on channel 1 reaches the target value **diTargetValue0**, output **Y1** is set to TRUE.

LD



ST

```
bIsActive:=HscInfo_IsActive(iChannel1);

Hsc_TargetValueMatch_Set(bExecute := DF(bSet) AND NOT bIsActive,
                iChannel := iChannel1,
                pYOutput :=
GetPointer(g_bHsc_TargetValueMatch_Channel1_Y1_RedLamp_On),
                diTargetValue := diTargetValue0,
                bError => bErrorSet);
```

Chapter 39

Pulse output instructions

39.1 Introduction

Control FPWIN Pro offers two concepts for programming with pulse output instructions:

- FP instructions
- Tool instructions

For users programming for different PLC types of the FP series or users who are tired of setting control code bits and looking up available channel numbers, the tool instructions offer new and comfortable features. These include information functions for evaluating status flags and settings, control functions for configuring high-speed counters and pulse outputs, PLC-independent functions and DUTs, as well as variable channel numbers. However, the FP instructions may be easier to use for beginners or users familiar with FPWIN GR.

Most of the information, which is accessible via information and control functions, is stored in special internal relays and special data registers. These relays and registers can also be accessed using PLC-independent system variables.

To take advantage of the features you prefer, the instructions of both libraries can be mixed.

☞ ◆**NOTE**

When programming with the tool instructions, be sure to refer to the detailed information provided via the links to the related F/P instructions.

Main features	FP instructions	Tool instructions
Pre version 6.4 support	●	
Use of inline functions	●	
Use of FPWIN GR function names	●	
Less code with constant channel numbers	●	
Control codes	●	
Control functions		●
Information functions		●
Variable channel numbers		●
Universal functions for all PLCs		●
DUT for common channel configuration for all PLCs for all pulse output instructions		●

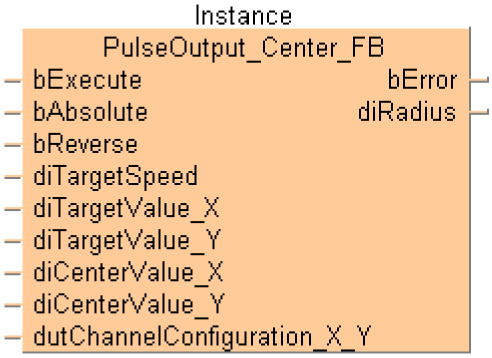
39.2 Pulse output function blocks

In this section:

PulseOutput_Center_FB

Circular interpolation (center position)

Description Pulses are output from two channels in accordance with the parameters in the function block and in the specified DUT, so that the path to the target position forms an arc. The radius of the circle is calculated by specifying the center position and the end position. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help: F176_PulseOutput_Center (see page 1077)

Use PulseInfo_IsActive (see page 1223) to check if the control flag for the selected channel is FALSE.

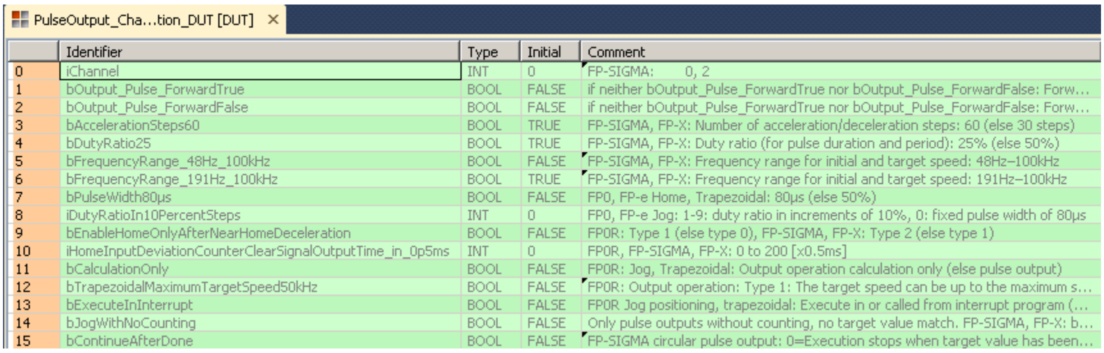
PLC types see see page 1329

Data types

Input variable	Data type	Function
bExecute	BOOL	Activates the function block (with permanent trigger)
bAbsolute		Absolute value control = TRUE, Relative value control = FALSE
bCounterclockwise		Rotation direction: Reverse = TRUE, Forward = FALSE
diTargetSpeed	DINT	Target speed: Composite speed of both axes = 100–20000 (100Hz–20kHz)
diTargetValue_X		Target value [pulses]: -8388608–8388607
diTargetValue_Y		
diCenterValue_X		
diCenterValue_Y		
dutChannelConfiguration_X_Y	Predefined system DUT for channel configuration: PulseOutput_Channel_Configuration_DUT Channel: 0, 2	
Output variable	**Data type**	**Function**
bError	BOOL	Refers to an internal mismatch of input values to avoid a PLC error.
diRadius	DINT	Radius [pulses]

Example In this example the function has been programmed in ladder diagram (LD). Please refer to the online help for a structured text (ST) example. The same POU header is used for all programming languages.

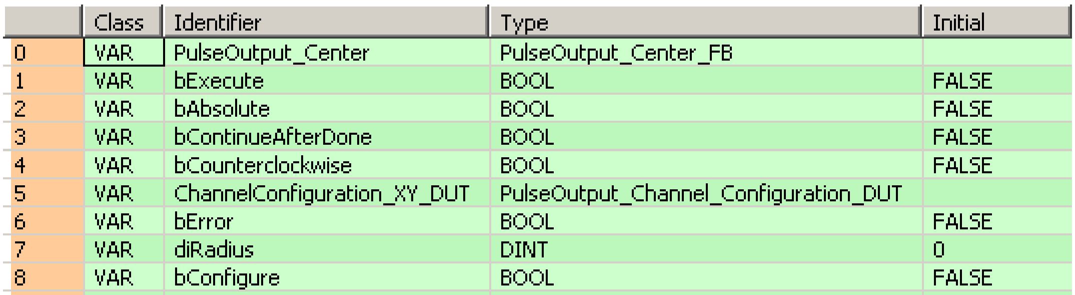
DUT Use the following predefined DUT: PulseOutput_Channel_Configuration_DUT



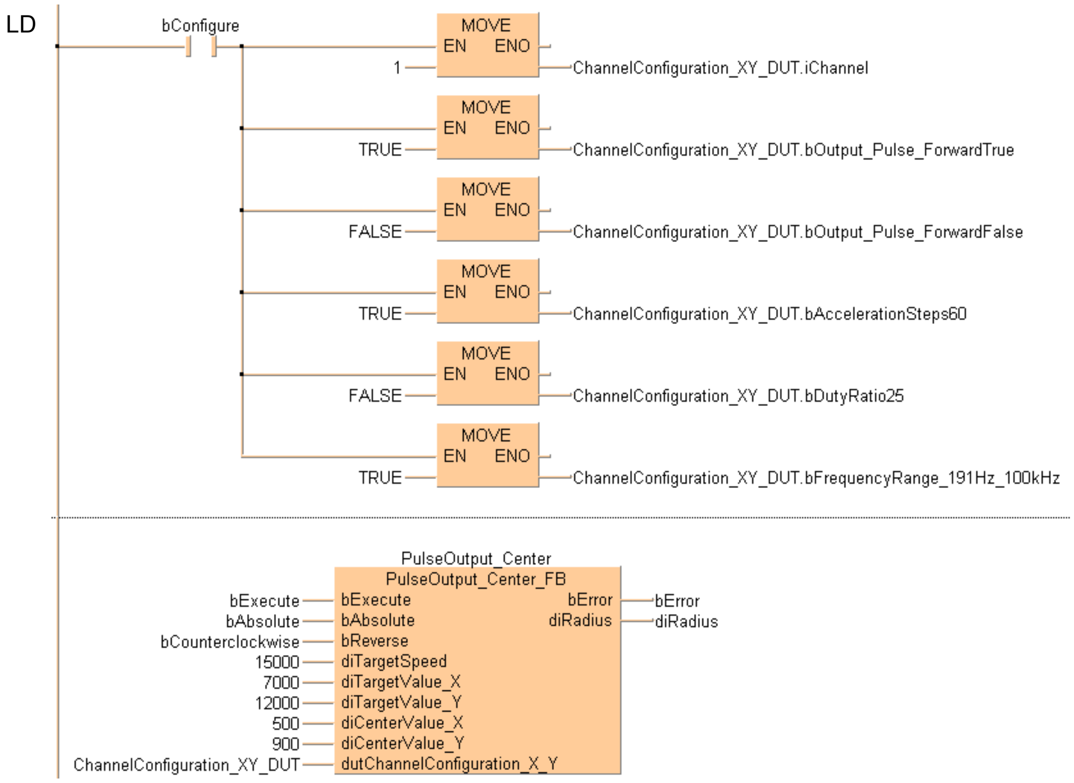
PulseOutput_Cha...tion_DUT [DUT]

	Identifier	Type	Initial	Comment
0	iChannel	INT	0	FP-SIGMA: 0, 2
1	bOutput_Pulse_ForwardTrue	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
2	bOutput_Pulse_ForwardFalse	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
3	bAccelerationSteps60	BOOL	TRUE	FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps)
4	bDutyRatio25	BOOL	TRUE	FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%)
5	bFrequencyRange_48Hz_100kHz	BOOL	FALSE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz
6	bFrequencyRange_191Hz_100kHz	BOOL	TRUE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz
7	bPulseWidth80µs	BOOL	FALSE	FP0, FP-e Home, Trapezoidal: 80µs (else 50%)
8	iDutyRatioIn10PercentSteps	INT	0	FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs
9	bEnableHomeOnlyAfterNearHomeDeceleration	BOOL	FALSE	FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1)
10	iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms	INT	0	FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms]
11	bCalculationOnly	BOOL	FALSE	FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output)
12	bTrapezoidalMaximumTargetSpeed50kHz	BOOL	FALSE	FP0R: Output operation: Type 1: The target speed can be up to the maximum s...
13	bExecuteInInterrupt	BOOL	FALSE	FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (...
14	bJogWithNoCounting	BOOL	FALSE	Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b...
15	bContinueAfterDone	BOOL	FALSE	FP-SIGMA circular pulse output: 0=Execution stops when target value has been...

POU header All input and output variables used for programming this function have been declared in the POU header.



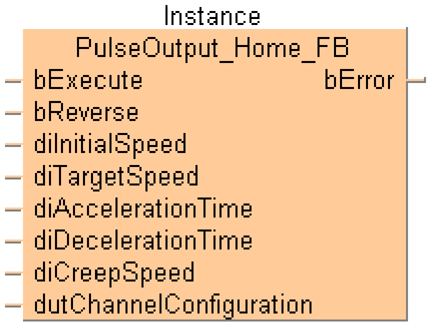
	Class	Identifier	Type	Initial
0	VAR	PulseOutput_Center	PulseOutput_Center_FB	
1	VAR	bExecute	BOOL	FALSE
2	VAR	bAbsolute	BOOL	FALSE
3	VAR	bContinueAfterDone	BOOL	FALSE
4	VAR	bCounterclockwise	BOOL	FALSE
5	VAR	ChannelConfiguration_XY_DUT	PulseOutput_Channel_Configuration_DUT	
6	VAR	bError	BOOL	FALSE
7	VAR	diRadius	DINT	0
8	VAR	bConfigure	BOOL	FALSE


LD
bConfigure
MOVE
EN ENO
1
ChannelConfiguration_XY_DUT.iChannel
MOVE
EN ENO
TRUE
ChannelConfiguration_XY_DUT.bOutput_Pulse_ForwardTrue
MOVE
EN ENO
FALSE
ChannelConfiguration_XY_DUT.bOutput_Pulse_ForwardFalse
MOVE
EN ENO
TRUE
ChannelConfiguration_XY_DUT.bAccelerationSteps60
MOVE
EN ENO
FALSE
ChannelConfiguration_XY_DUT.bDutyRatio25
MOVE
EN ENO
TRUE
ChannelConfiguration_XY_DUT.bFrequencyRange_191Hz_100kHz
PulseOutput_Center
PulseOutput_Center_FB
bExecute
bExecute
bError
bError
bAbsolute
bAbsolute
diRadius
diRadius
bCounterclockwise
bReverse
15000
diTargetSpeed
7000
diTargetValue_X
12000
diTargetValue_Y
500
diCenterValue_X
900
diCenterValue_Y
ChannelConfiguration_XY_DUT
dutChannelConfiguration_X_Y

PulseOutput_Home_FB

Home return

Description This instruction performs a home return according to the parameters in the function block and in the specified DUT. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help:

- FPΣ, FP-X: F171_PulseOutput_Home (see page 1051)
- FP0R: F177_PulseOutput_Home (see page 1085)
- FP-e, FP0: F168_PulseOutput_Home (see page 1035)

Use PulseInfo_IsActive (see page 1223) to check if the control flag for the selected channel is FALSE.

Use PulseInfo_IsHomeInputTrue (see page 1227) to check if the home input is TRUE.

☞ **Note the following to avoid malfunctions or an operation error:**

- Ensure to set the system register to pulse output mode with home input.
- The home input may not be occupied by other instructions like pulse-catch input, interrupt input or high-speed counter.

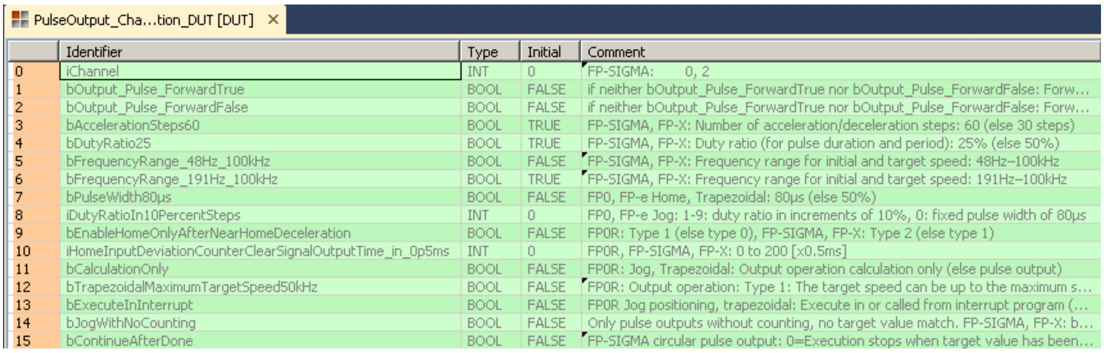
PLC types see page 1330

Data types

Input variable	Data type	Function
bExecute	BOOL	A rising edge activates the function block
bReverse		Movement direction: Forward = FALSE, Reverse = TRUE
diInitialSpeed	DINT	Initial speed/Target speed: Set this value according to the frequency range selected in PulseOutput_Channel_Configuration_DUT: FPΣ, FP-X: 1 to 9800 (1.5Hz–9.8kHz) 48 to 100000 (48Hz–100kHz) 191 to 100000 (191–100kHz) FP0R: 1 to 50000 (1Hz–50kHz) FP0, FP-e: 40 to 5000 (40Hz–5kHz)
diTargetSpeed		
diAccelerationTime		Acceleration/deceleration time (FPΣ, FP-X): With 30 steps: 30ms–32760ms (specify in steps of 30) With 60 steps: 60ms–32760ms (specify in steps of 60) Acceleration/deceleration time (FP0, FP-e): 30ms–32760ms Acceleration time (FP0R): 1ms–32760ms
diDecelerationTime		Deceleration time (FP0R): 1ms–32760ms
diCreepSpeed		Creep speed (FP0R): 1 to 50000 (1Hz–50kHz)
dutChannelConfiguration	Predefined system DUT for channel configuration: PulseOutput_Channel_Configuration_DUT	
Output variable	**Data type**	**Function**
bError	BOOL	Refers to an internal mismatch of input values to avoid a PLC error.

Example In this example the function has been programmed in ladder diagram (LD). Please refer to the online help for a structured text (ST) example. The same POU header is used for all programming languages.

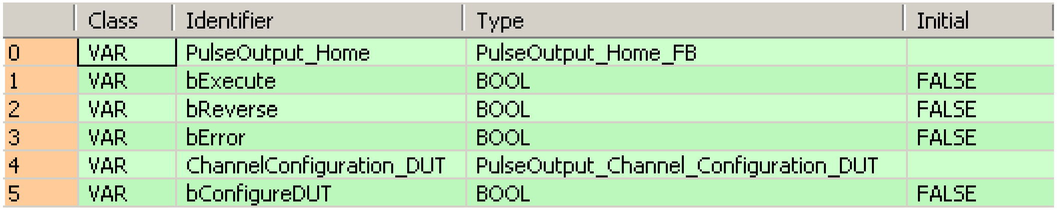
DUT Use the following predefined DUT: PulseOutput_Channel_Configuration_DUT



PulseOutput_Cha...tion_DUT [DUT]

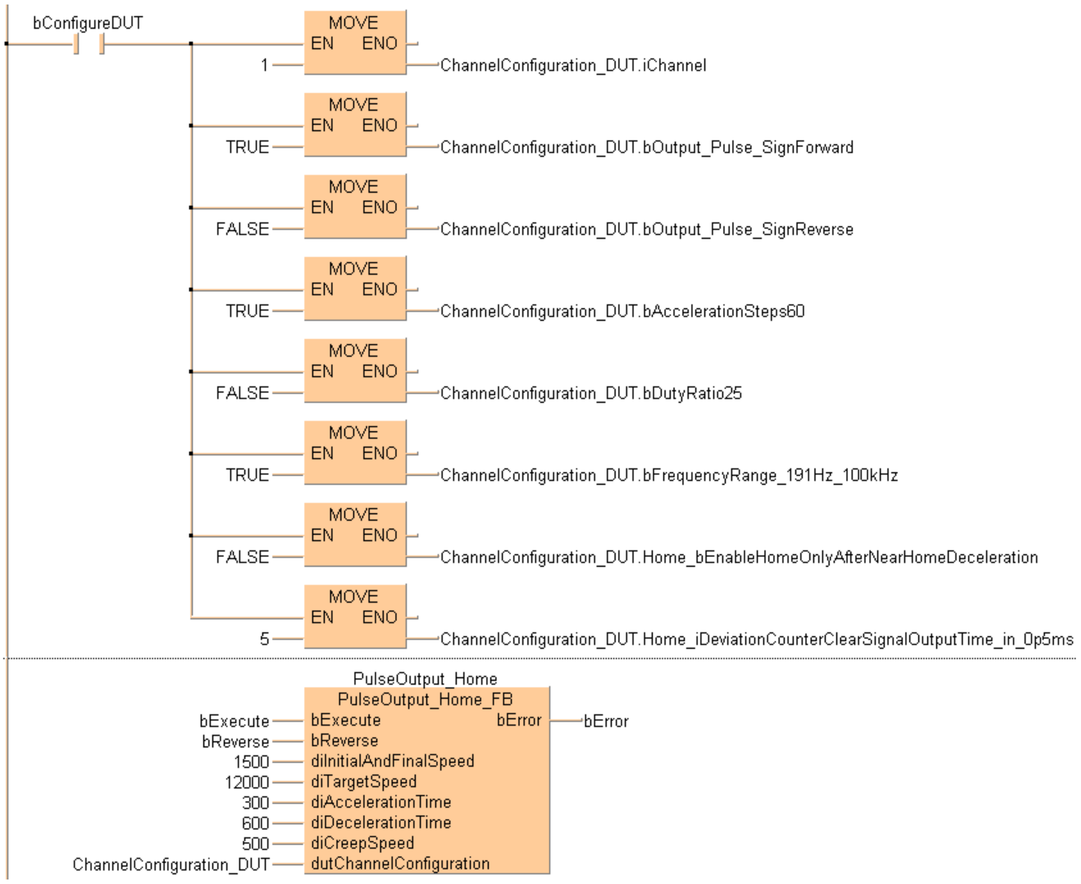
	Identifier	Type	Initial	Comment
0	iChannel	INT	0	FP-SIGMA: 0, 2
1	bOutput_Pulse_ForwardTrue	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
2	bOutput_Pulse_ForwardFalse	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
3	bAccelerationSteps60	BOOL	TRUE	FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps)
4	bDutyRatio25	BOOL	TRUE	FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%)
5	bFrequencyRange_48Hz_100kHz	BOOL	FALSE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz
6	bFrequencyRange_191Hz_100kHz	BOOL	TRUE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz
7	bPulseWidth80µs	BOOL	FALSE	FP0, FP-e Home, Trapezoidal: 80µs (else 50%)
8	iDutyRatioIn10PercentSteps	INT	0	FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs
9	bEnableHomeOnlyAfterNearHomeDeceleration	BOOL	FALSE	FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1)
10	iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms	INT	0	FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms]
11	bCalculationOnly	BOOL	FALSE	FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output)
12	bTrapezoidalMaximumTargetSpeed50kHz	BOOL	FALSE	FP0R: Output operation: Type 1: The target speed can be up to the maximum s...
13	bExecuteInInterrupt	BOOL	FALSE	FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (...
14	bJogWithNoCounting	BOOL	FALSE	Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b...
15	bContinueAfterDone	BOOL	FALSE	FP-SIGMA circular pulse output: 0=Execution stops when target value has been...

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	PulseOutput_Home	PulseOutput_Home_FB	
1	VAR	bExecute	BOOL	FALSE
2	VAR	bReverse	BOOL	FALSE
3	VAR	bError	BOOL	FALSE
4	VAR	ChannelConfiguration_DUT	PulseOutput_Channel_Configuration_DUT	
5	VAR	bConfigureDUT	BOOL	FALSE

LD

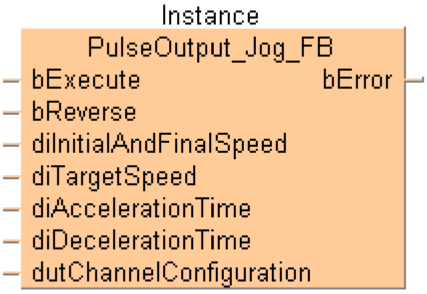


```
bConfigureDUT
MOVE EN ENO      1 → ChannelConfiguration_DUT.iChannel
MOVE EN ENO   TRUE → ChannelConfiguration_DUT.bOutput_Pulse_SignForward
MOVE EN ENO  FALSE → ChannelConfiguration_DUT.bOutput_Pulse_SignReverse
MOVE EN ENO   TRUE → ChannelConfiguration_DUT.bAccelerationSteps60
MOVE EN ENO  FALSE → ChannelConfiguration_DUT.bDutyRatio25
MOVE EN ENO   TRUE → ChannelConfiguration_DUT.bFrequencyRange_191Hz_100kHz
MOVE EN ENO  FALSE → ChannelConfiguration_DUT.Home_bEnableHomeOnlyAfterNearHomeDeceleration
MOVE EN ENO      5 → ChannelConfiguration_DUT.Home_iDeviationCounterClearSignalOutputTime_in_0p5ms

PulseOutput_Home
PulseOutput_Home_FB
bExecute               bExecute                  bError  bError
bReverse               bReverse
1500                   diInitialAndFinalSpeed
12000                  diTargetSpeed
300                    diAccelerationTime
600                    diDecelerationTime
500                    diCreepSpeed
ChannelConfiguration_DUT  dutChannelConfiguration
```

PulseOutput_Jog_FB

JOG operation

Description This instruction is used for JOG operation. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help: F172_PulseOutput_Jog (see page 1060). Use PulseInfo_IsActive (see page 1223) to check if the control flag for the selected channel is FALSE.

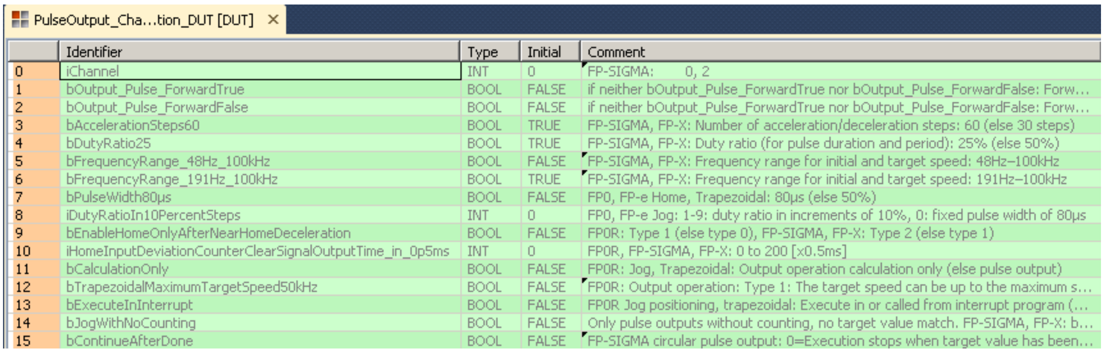
PLC types see see page 1329

Data types

Input variable	Data type	Function
bExecute	BOOL	With edge or permanent if change of speed required
bReverse		Movement direction: Forward = FALSE, Reverse = TRUE
diInitialAndFinalSpeed	DINT	Initial and final speed (FP0R): 1 to 50000 (1Hz–50kHz)
diTargetSpeed		Target speed: Set this value according to the frequency range selected in PulseOutput_Channel_Configuration_DUT: FPΣ, FP-X: 1 to 9800 (1.5Hz–9.8kHz) 48 to 100000 (48Hz–100kHz) 191 to 100000 (191–100kHz) FP0R: 1 to 50000 (1Hz–50kHz) FP0, FP-e: 40 to 5000 (40Hz–5kHz)
diAccelerationTime		Acceleration time (FP0R): 1ms–32760ms (up to the maximum speed)
diDecelerationTime		Deceleration time (FP0R): 1ms–32760ms (from the maximum speed)
dutChannelConfiguration	Predefined system DUT for channel configuration: PulseOutput_Channel_Configuration_DUT	
Output variable	**Data type**	**Function**
bError	BOOL	Refers to an internal mismatch of input values to avoid a PLC error.

Example In this example the function has been programmed in ladder diagram (LD). Please refer to the online help for a structured text (ST) example. The same POU header is used for all programming languages.

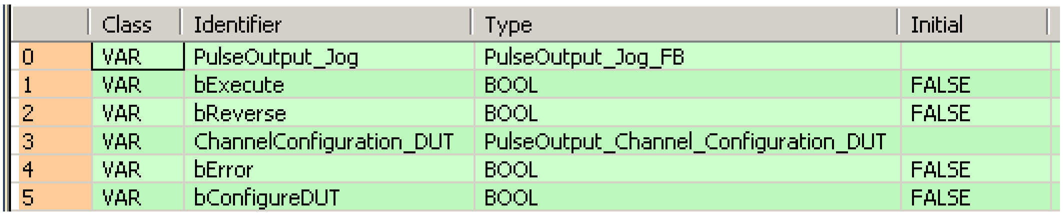
DUT Use the following predefined DUT: PulseOutput_Channel_Configuration_DUT



PulseOutput_Cha...tion_DUT [DUT]

	Identifier	Type	Initial	Comment
0	iChannel	INT	0	FP-SIGMA: 0, 2
1	bOutput_Pulse_ForwardTrue	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
2	bOutput_Pulse_ForwardFalse	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
3	bAccelerationSteps60	BOOL	TRUE	FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps)
4	bDutyRatio25	BOOL	TRUE	FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%)
5	bFrequencyRange_48Hz_100kHz	BOOL	FALSE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz
6	bFrequencyRange_191Hz_100kHz	BOOL	TRUE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz
7	bPulseWidth80µs	BOOL	FALSE	FP0, FP-e Home, Trapezoidal: 80µs (else 50%)
8	iDutyRatioIn10PercentSteps	INT	0	FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs
9	bEnableHomeOnlyAfterNearHomeDeceleration	BOOL	FALSE	FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1)
10	iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms	INT	0	FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms]
11	bCalculationOnly	BOOL	FALSE	FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output)
12	bTrapezoidalMaximumTargetSpeed50kHz	BOOL	FALSE	FP0R: Output operation: Type 1: The target speed can be up to the maximum s...
13	bExecuteInInterrupt	BOOL	FALSE	FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (...
14	bJogWithNoCounting	BOOL	FALSE	Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b...
15	bContinueAfterDone	BOOL	FALSE	FP-SIGMA circular pulse output: 0=Execution stops when target value has been...

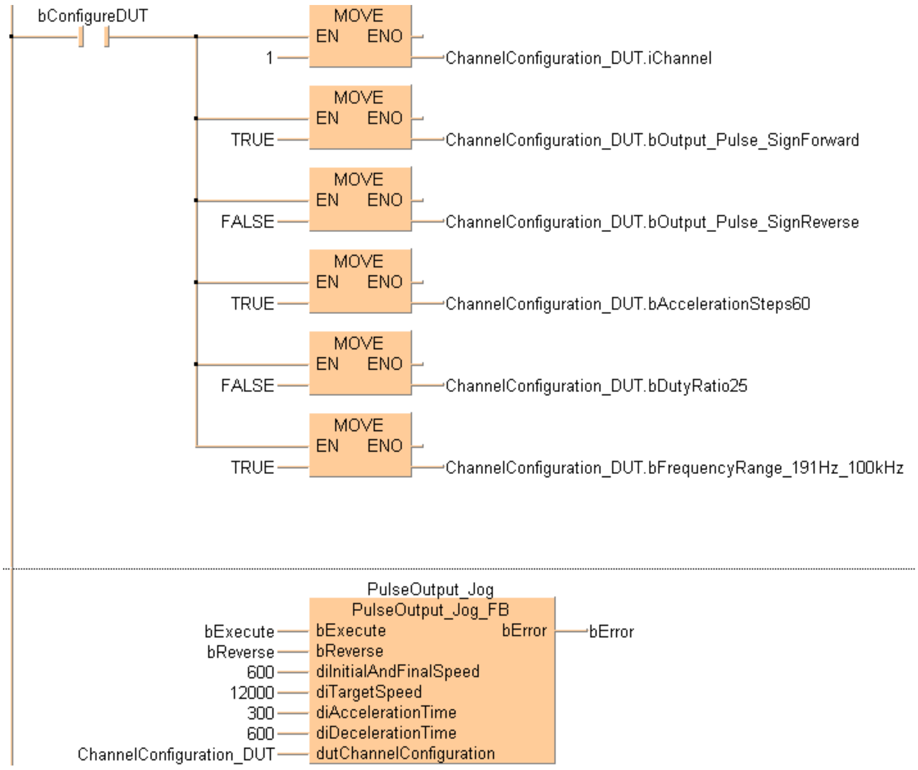
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	PulseOutput_Jog	PulseOutput_Jog_FB	
1	VAR	bExecute	BOOL	FALSE
2	VAR	bReverse	BOOL	FALSE
3	VAR	ChannelConfiguration_DUT	PulseOutput_Channel_Configuration_DUT	
4	VAR	bError	BOOL	FALSE
5	VAR	bConfigureDUT	BOOL	FALSE

Body

LD

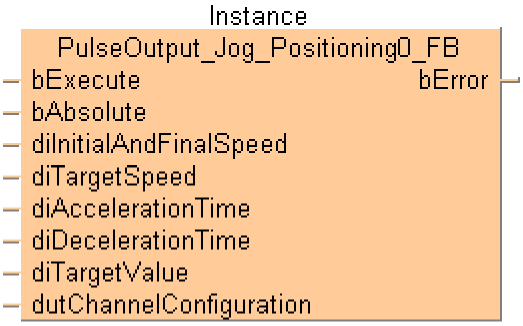


PulseOutput_Jog_Positioning0_FB

JOG operation and positioning

Description This instruction is used for JOG operation. The specified number of pulses is output after the position control trigger input has turned to TRUE. A deceleration is performed before the target value is reached and pulse output stops. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.

The speed can be changed within the range of the specified target speed.



This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help: F171_PulseOutput_Jog_Positioning (see page 1055). Use PulseInfo_IsActive (see page 1223) to check if the control flag for the selected channel is FALSE. Use PulseControl_PulseOutputStop (see page 1213) to stop pulse output on a specified channel. Use PulseControl_DeceleratedStop (see page 1202) to perform a decelerated stop.

PLC types see see page 1329

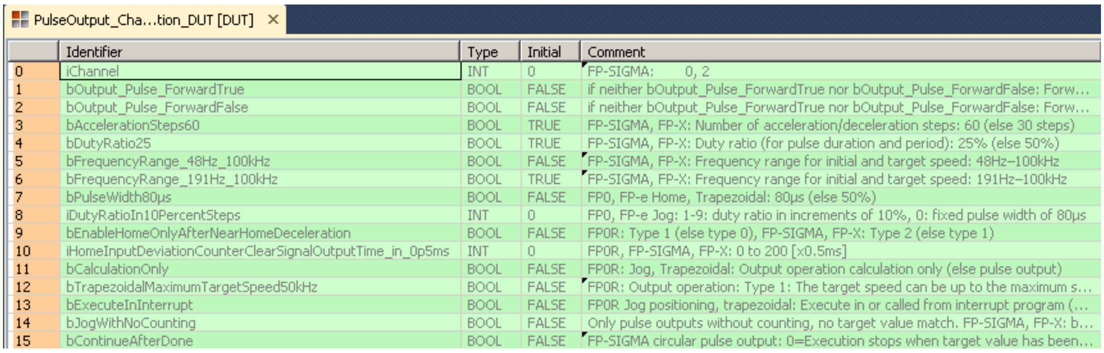
Data types

Input variable	Data type	Function
bExecute	BOOL	With edge or permanent if change of speed required
bAbsolute	BOOL:=FALSE	Only incremental mode supported, must be FALSE always, otherwise an error is output.
diInitialAndFinalSpeed	DINT	Initial and final speed (FP0R): 1 to 50000 (1Hz–50kHz)
diTargetSpeed		Target speed: Set this value according to the frequency range selected in PulseOutput_Channel_Configuration_DUT: FPΣ, FP-X: 1 to 9800 (1.5Hz–9.8kHz) 48 to 100000 (48Hz–100kHz) 191 to 100000 (191–100kHz) FP0R: 1 to 50000 (1Hz–50kHz) FP0, FP-e: 40 to 5000 (40Hz–5kHz)
diAccelerationTime		Acceleration time (FP0R): 1ms–32760ms (up to the maximum speed)
diDecelerationTime		Deceleration time (FP0R): 1ms–32760ms (from the maximum speed)
diTargetValue		Target value [pulses]: -2147483648–2147483647
dutChannelConfiguration	Predefined system DUT for channel configuration: PulseOutput_Channel_Configuration_DUT	

Output variable	Data type	Function
bError	BOOL	Refers to an internal mismatch of input values to avoid a PLC error. TRUE if the applied channel is not enabled in the system registers or if **bAbsolute** is TRUE

Example In this example the function has been programmed in ladder diagram (LD). Please refer to the online help for a structured text (ST) example. The same POU header is used for all programming languages.

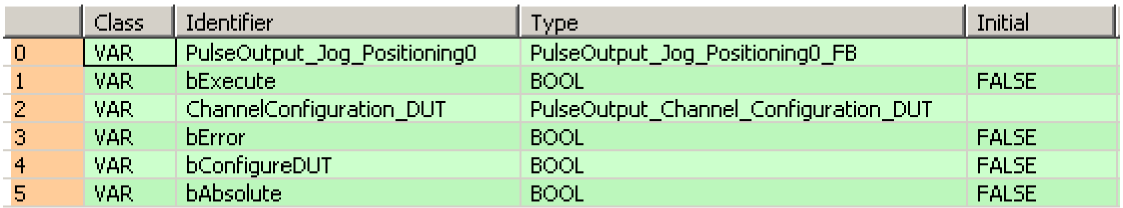
DUT Use the following predefined DUT: PulseOutput_Channel_Configuration_DUT



PulseOutput_Cha...tion_DUT [DUT]

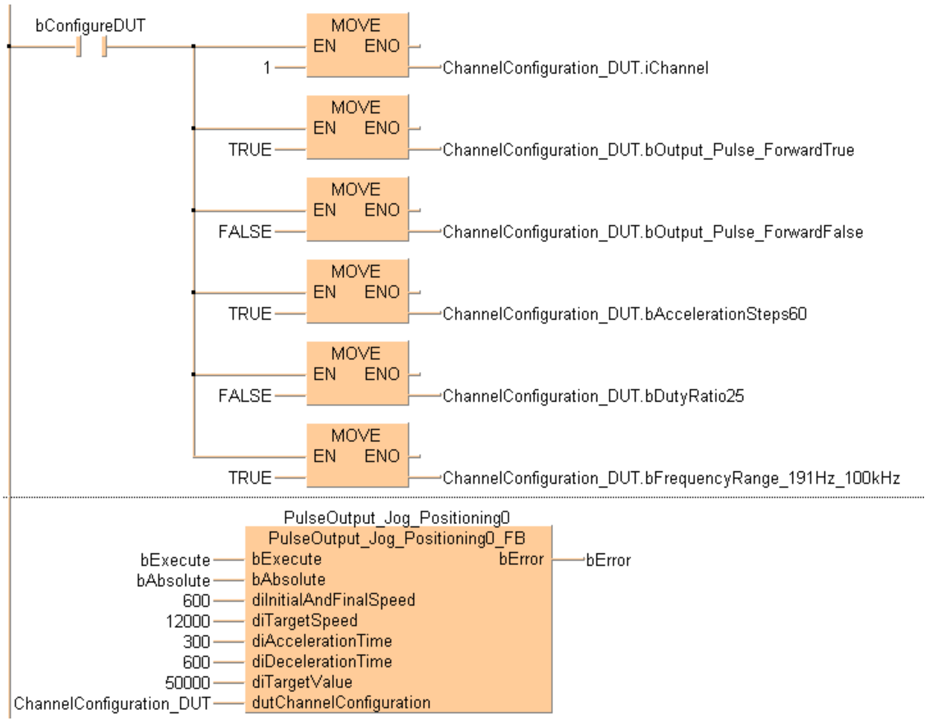
	Identifier	Type	Initial	Comment
0	iChannel	INT	0	FP-SIGMA: 0, 2
1	bOutput_Pulse_ForwardTrue	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
2	bOutput_Pulse_ForwardFalse	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
3	bAccelerationSteps60	BOOL	TRUE	FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps)
4	bDutyRatio25	BOOL	TRUE	FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%)
5	bFrequencyRange_48Hz_100kHz	BOOL	FALSE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz
6	bFrequencyRange_191Hz_100kHz	BOOL	TRUE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz
7	bPulseWidth80µs	BOOL	FALSE	FP0, FP-e Home, Trapezoidal: 80µs (else 50%)
8	iDutyRatioIn10PercentSteps	INT	0	FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs
9	bEnableHomeOnlyAfterNearHomeDeceleration	BOOL	FALSE	FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1)
10	iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms	INT	0	FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms]
11	bCalculationOnly	BOOL	FALSE	FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output)
12	bTrapezoidalMaximumTargetSpeed50kHz	BOOL	FALSE	FP0R: Output operation: Type 1: The target speed can be up to the maximum s...
13	bExecuteInInterrupt	BOOL	FALSE	FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (...
14	bJogWithNoCounting	BOOL	FALSE	Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b...
15	bContinueAfterDone	BOOL	FALSE	FP-SIGMA circular pulse output: 0=Execution stops when target value has been...

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	PulseOutput_Jog_Positioning0	PulseOutput_Jog_Positioning0_FB	
1	VAR	bExecute	BOOL	FALSE
2	VAR	ChannelConfiguration_DUT	PulseOutput_Channel_Configuration_DUT	
3	VAR	bError	BOOL	FALSE
4	VAR	bConfigureDUT	BOOL	FALSE
5	VAR	bAbsolute	BOOL	FALSE

LD


bConfigureDUT
MOVE
EN ENO
1
ChannelConfiguration_DUT.iChannel
MOVE
EN ENO
TRUE
ChannelConfiguration_DUT.bOutput_Pulse_ForwardTrue
MOVE
EN ENO
FALSE
ChannelConfiguration_DUT.bOutput_Pulse_ForwardFalse
MOVE
EN ENO
TRUE
ChannelConfiguration_DUT.bAccelerationSteps60
MOVE
EN ENO
FALSE
ChannelConfiguration_DUT.bDutyRatio25
MOVE
EN ENO
TRUE
ChannelConfiguration_DUT.bFrequencyRange_191Hz_100kHz
PulseOutput_Jog_Positioning0
PulseOutput_Jog_Positioning0_FB
bExecute
bExecute
bError
bError
bAbsolute
bAbsolute
600
diInitialAndFinalSpeed
12000
diTargetSpeed
300
diAccelerationTime
600
diDecelerationTime
50000
diTargetValue
ChannelConfiguration_DUT
dutChannelConfiguration

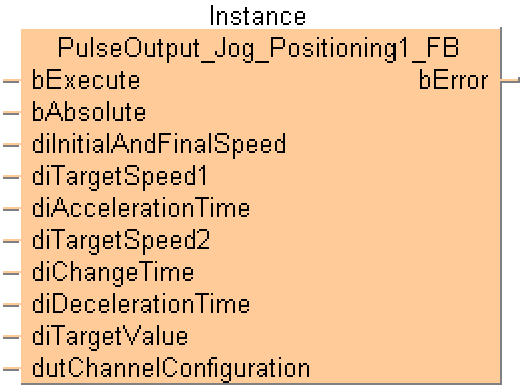
PulseOutput_Jog_Positioning1_FB

JOG operation and positioning

Description

This instruction is used for JOG operation. The specified number of pulses is output after the position control trigger input has turned to TRUE. A deceleration is performed before the target value is reached and pulse output stops. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.

The target speed can be changed once when the position control trigger input turns to TRUE.



This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help:
F171_PulseOutput_Jog_Positioning (see page 1055)

Use PulseInfo_IsActive (see page 1223) to check if the control flag for the selected channel is FALSE.

Use PulseControl_PulseOutputStop (see page 1213) to stop pulse output on a specified channel. Use PulseControl_DeceleratedStop (see page 1202) to perform a decelerated stop.

PLC types

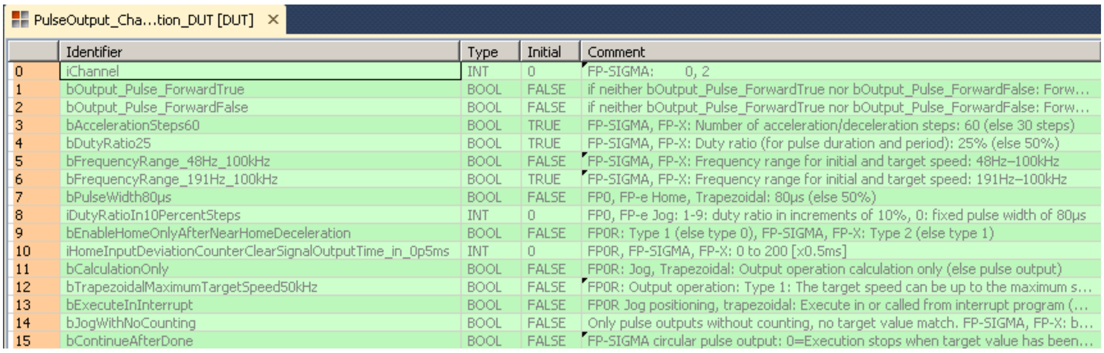
see see page 1329

Data types

Input variable	Data type	Function
bExecute	BOOL	With edge or permanent if change of speed required
bAbsolute	BOOL:=FALSE	Only incremental mode supported, must be FALSE always, otherwise an error is output.
diInitialAndFinalSpeed	DINT	Initial and final speed = 1 to 50000 (1Hz–50kHz)
diTargetSpeed1		Target speed = 1 to 50000 (1Hz–50kHz)
diAccelerationTime		Acceleration time = 1ms–32760ms
diTargetSpeed2		Target speed = 1 to 50000 (1Hz–50kHz)
diChangeTime		Change time = 1ms–32760ms
diDecelerationTime		Deceleration time = 1ms–32760ms
diTargetValue		Target value [pulses]: -2147483648–2147483647
dutChannelConfiguration	Predefined system DUT for channel configuration: PulseOutput_Channel_Configuration_DUT	
Output variable	**Data type**	**Function**
bError	BOOL	Refers to an internal mismatch of input values to avoid a PLC error. TRUE if the applied channel is not enabled in the system registers or if **bAbsolute** is TRUE

Example In this example the function has been programmed in ladder diagram (LD). Please refer to the online help for a structured text (ST) example. The same POU header is used for all programming languages.

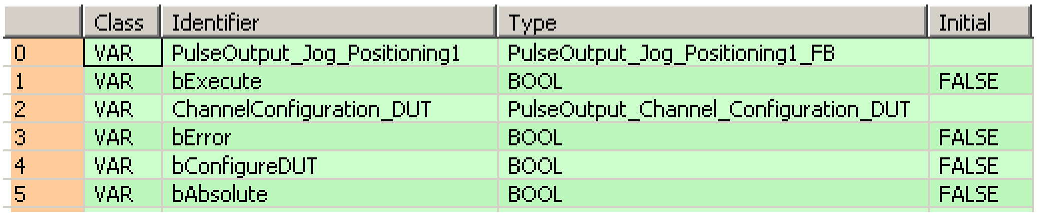
DUT Use the following predefined DUT: PulseOutput_Channel_Configuration_DUT



PulseOutput_Cha...tion_DUT [DUT]

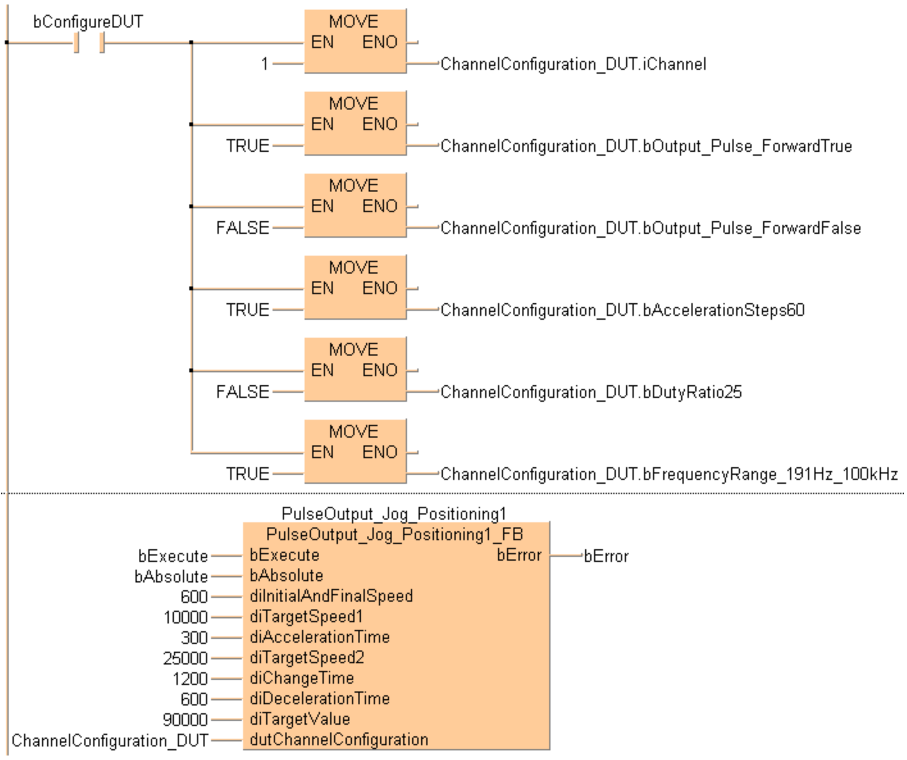
	Identifier	Type	Initial	Comment
0	iChannel	INT	0	FP-SIGMA: 0, 2
1	bOutput_Pulse_ForwardTrue	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
2	bOutput_Pulse_ForwardFalse	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
3	bAccelerationSteps60	BOOL	TRUE	FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps)
4	bDutyRatio25	BOOL	TRUE	FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%)
5	bFrequencyRange_48Hz_100kHz	BOOL	FALSE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz
6	bFrequencyRange_191Hz_100kHz	BOOL	TRUE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz
7	bPulseWidth80µs	BOOL	FALSE	FP0, FP-e Home, Trapezoidal: 80µs (else 50%)
8	iDutyRatioIn10PercentSteps	INT	0	FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs
9	bEnableHomeOnlyAfterNearHomeDeceleration	BOOL	FALSE	FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1)
10	iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms	INT	0	FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms]
11	bCalculationOnly	BOOL	FALSE	FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output)
12	bTrapezoidalMaximumTargetSpeed50kHz	BOOL	FALSE	FP0R: Output operation: Type 1: The target speed can be up to the maximum s...
13	bExecuteInInterrupt	BOOL	FALSE	FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (...
14	bJogWithNoCounting	BOOL	FALSE	Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b...
15	bContinueAfterDone	BOOL	FALSE	FP-SIGMA circular pulse output: 0=Execution stops when target value has been...

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	PulseOutput_Jog_Positioning1	PulseOutput_Jog_Positioning1_FB	
1	VAR	bExecute	BOOL	FALSE
2	VAR	ChannelConfiguration_DUT	PulseOutput_Channel_Configuration_DUT	
3	VAR	bError	BOOL	FALSE
4	VAR	bConfigureDUT	BOOL	FALSE
5	VAR	bAbsolute	BOOL	FALSE

LD

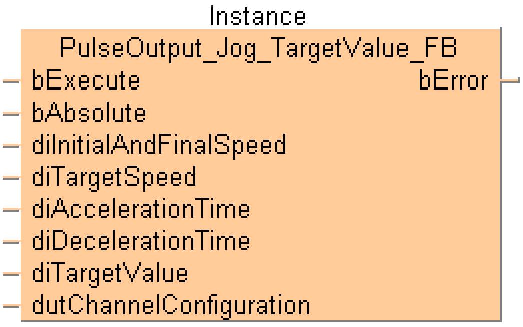

bConfigureDUT
MOVE
EN ENO
1
ChannelConfiguration_DUT.iChannel
MOVE
EN ENO
TRUE
ChannelConfiguration_DUT.bOutput_Pulse_ForwardTrue
MOVE
EN ENO
FALSE
ChannelConfiguration_DUT.bOutput_Pulse_ForwardFalse
MOVE
EN ENO
TRUE
ChannelConfiguration_DUT.bAccelerationSteps60
MOVE
EN ENO
FALSE
ChannelConfiguration_DUT.bDutyRatio25
MOVE
EN ENO
TRUE
ChannelConfiguration_DUT.bFrequencyRange_191Hz_100kHz
PulseOutput_Jog_Positioning1
PulseOutput_Jog_Positioning1_FB
bExecute
bExecute
bError
bError
bAbsolute
bAbsolute
600
diInitialAndFinalSpeed
10000
diTargetSpeed1
300
diAccelerationTime
25000
diTargetSpeed2
1200
diChangeTime
600
diDecelerationTime
90000
diTargetValue
ChannelConfiguration_DUT
dutChannelConfiguration

PulseOutput_Jog_TargetValue_FB

JOG operation with target value

Description

This instruction is used for JOG operation. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE. Pulse output stops when the target value is reached.



This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help: F172_PulseOutput_Jog (see page 1060). Use PulseInfo_IsActive (see page 1223) to check if the control flag for the selected channel is FALSE.

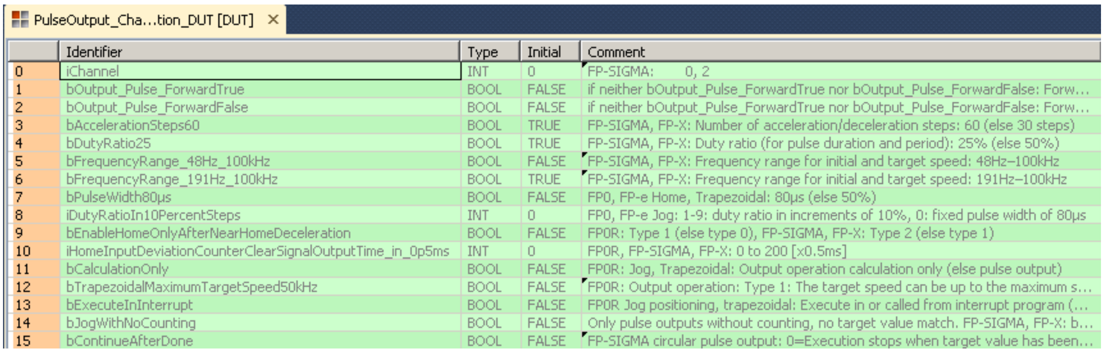
PLC types see page 1329

Data types

Input variable	Data type	Function
bExecute	BOOL	With edge or permanent if change of speed requiredWith edge or permanent if change of speed required
bAbsolute		FP0R: Absolute value control = TRUE, Relative value control = FALSE
diInitialAndFinalSpeed	DINT	FP0R: Initial and final speed = 1 to 50000 (1Hz–50kHz)
diTargetSpeed		Target speed: Set this value according to the frequency range selected in PulseOutput_Channel_Configuration_DUT: FPΣ, FP-X: 1 to 9800 (1.5Hz–9.8kHz) 48 to 100000 (48Hz–100kHz) 191 to 100000 (191–100kHz) FP0R: 1 to 50000 (1Hz–50kHz) FP0, FP-e: 40 to 5000 (40Hz–5kHz)
diAccelerationTime		Acceleration time (FP0R): 1ms–32760ms (up to the maximum speed)
diDecelerationTime		Deceleration time (FP0R): 1ms–32760ms (from the maximum speed)
diTargetValue		Target value [pulses]: -2147483648–2147483647
dutChannelConfiguration	Predefined system DUT for channel configuration: PulseOutput_Channel_Configuration_DUT	
Output variable	**Data type**	**Function**
bError	BOOL	Refers to an internal mismatch of input values to avoid a PLC error. Additional error condition for FPΣ, FP-X : TRUE if the applied channel is not enabled in the system registers or if **bAbsolute** is TRUE

Example In this example the function has been programmed in ladder diagram (LD). Please refer to the online help for a structured text (ST) example. The same POU header is used for all programming languages.

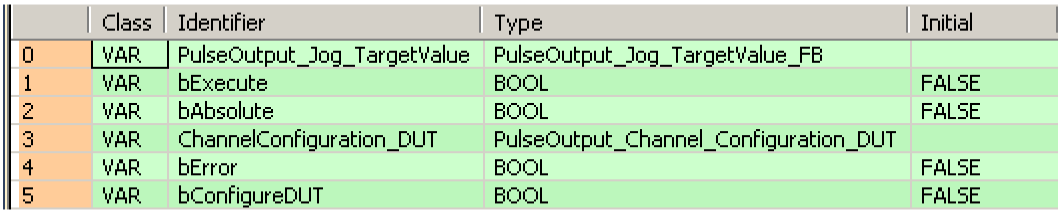
DUT Use the following predefined DUT: PulseOutput_Channel_Configuration_DUT



PulseOutput_Cha...tion_DUT [DUT]

	Identifier	Type	Initial	Comment
0	iChannel	INT	0	FP-SIGMA: 0, 2
1	bOutput_Pulse_ForwardTrue	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
2	bOutput_Pulse_ForwardFalse	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
3	bAccelerationSteps60	BOOL	TRUE	FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps)
4	bDutyRatio25	BOOL	TRUE	FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%)
5	bFrequencyRange_48Hz_100kHz	BOOL	FALSE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz
6	bFrequencyRange_191Hz_100kHz	BOOL	TRUE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz
7	bPulseWidth80µs	BOOL	FALSE	FP0, FP-e Home, Trapezoidal: 80µs (else 50%)
8	iDutyRatioIn10PercentSteps	INT	0	FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs
9	bEnableHomeOnlyAfterNearHomeDeceleration	BOOL	FALSE	FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1)
10	iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms	INT	0	FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms]
11	bCalculationOnly	BOOL	FALSE	FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output)
12	bTrapezoidalMaximumTargetSpeed50kHz	BOOL	FALSE	FP0R: Output operation: Type 1: The target speed can be up to the maximum s...
13	bExecuteInInterrupt	BOOL	FALSE	FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (...
14	bJogWithNoCounting	BOOL	FALSE	Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b...
15	bContinueAfterDone	BOOL	FALSE	FP-SIGMA circular pulse output: 0=Execution stops when target value has been...

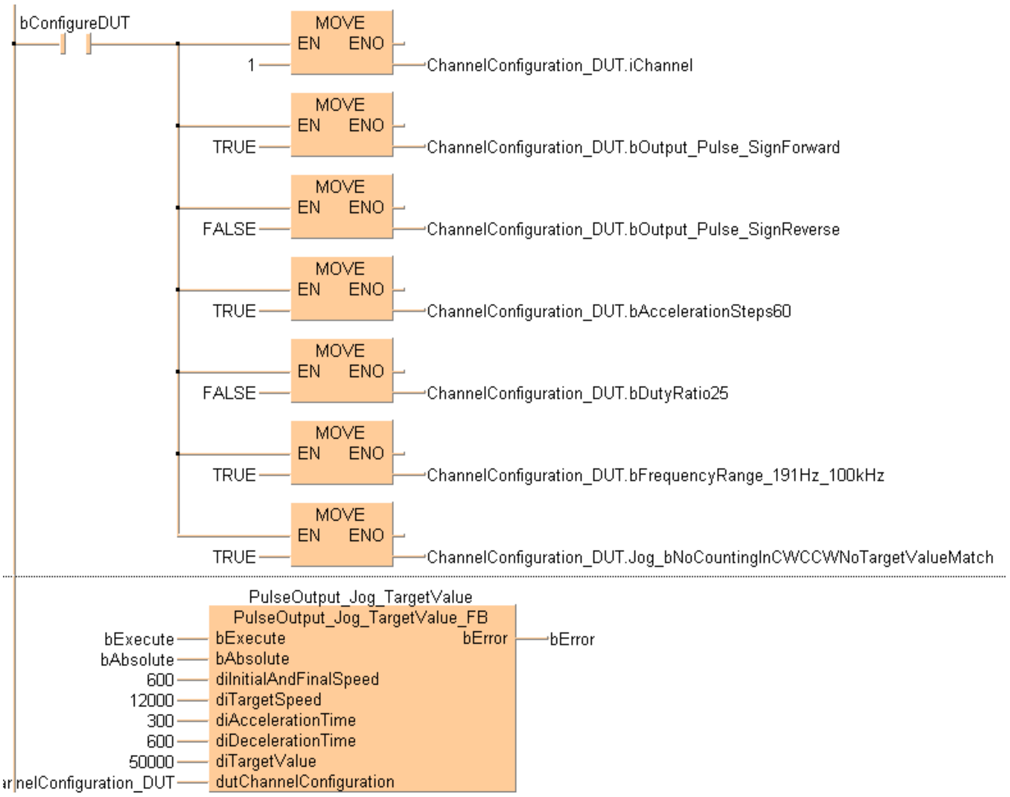
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	PulseOutput_Jog_TargetValue	PulseOutput_Jog_TargetValue_FB	
1	VAR	bExecute	BOOL	FALSE
2	VAR	bAbsolute	BOOL	FALSE
3	VAR	ChannelConfiguration_DUT	PulseOutput_Channel_Configuration_DUT	
4	VAR	bError	BOOL	FALSE
5	VAR	bConfigureDUT	BOOL	FALSE

Body

LD

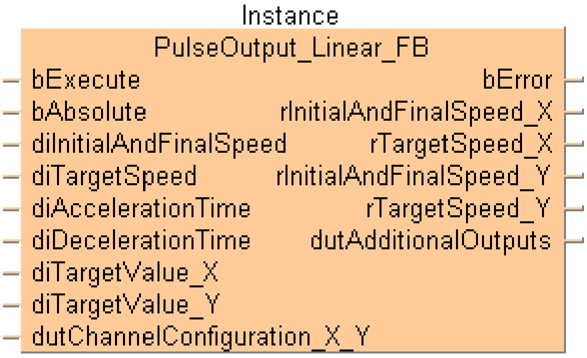


bConfigureDUT
MOVE
EN ENO
1
ChannelConfiguration_DUT.iChannel
MOVE
EN ENO
TRUE
ChannelConfiguration_DUT.bOutput_Pulse_SignForward
MOVE
EN ENO
FALSE
ChannelConfiguration_DUT.bOutput_Pulse_SignReverse
MOVE
EN ENO
TRUE
ChannelConfiguration_DUT.bAccelerationSteps60
MOVE
EN ENO
FALSE
ChannelConfiguration_DUT.bDutyRatio25
MOVE
EN ENO
TRUE
ChannelConfiguration_DUT.bFrequencyRange_191Hz_100kHz
MOVE
EN ENO
TRUE
ChannelConfiguration_DUT.Jog_bNoCountingInCWCCWNoTargetValueMatch
PulseOutput_Jog_TargetValue
PulseOutput_Jog_TargetValue_FB
bExecute
bExecute
bError
bError
bAbsolute
bAbsolute
600
diInitialAndFinalSpeed
12000
diTargetSpeed
300
diAccelerationTime
600
diDecelerationTime
50000
diTargetValue
nelConfiguration_DUT
dutChannelConfiguration

PulseOutput_Linear_FB

Linear interpolation

Description Pulses are output from two channels in accordance with the parameters in the function block and in the specified DUT, so that the path to the target position forms a straight line. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help: F175_PulseOutput_Linear (see page 1072). Use PulseInfo_IsActive (see page 1223) to check if the control flag for the selected channel is FALSE.

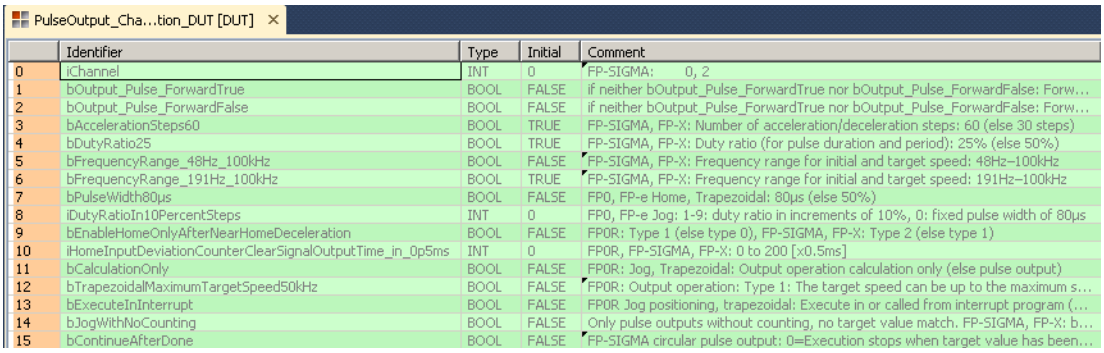
PLC types see page 1330

Data types

Input variable	Data type	Function
bExecute	BOOL	With edge or permanent if change of speed required
bAbsolute		Absolute value control = TRUE, Relative value control = FALSE
diInitialAndFinalSpeed	DINT	Initial and final speed: Composite speed = 1 to 50000 (1Hz–50kHz)
diTargetSpeed		Target speed: Composite speed = 1 to 50000 (1Hz–50kHz)
diAccelerationTime		Acceleration/deceleration time (FPΣ, FP-X): 0ms–32767ms Acceleration time (FP0R): 0ms–32767ms
diDecelerationTime		Deceleration time (FP0R): 0ms–32767ms
diTargetValue_X		X-axis target value [pulses] -8388608–8388607
diTargetValue_Y		Y-axis target value [pulses] -8388608–8388607
dutChannelConfiguration_X_Y	Predefined system DUT for channel configuration: PulseOutput_Channel_Configuration_DUT For interpolation, channel 0 and 1 or channel 2 and 3 are used as pairs. You may only specify 0 or 2 (for C14T: 0 only).	
Output variable	**Data type**	**Function**
bError	BOOL	Refers to an internal mismatch of input values to avoid a PLC error. Is set only if global constant MC_PulseOutput_Library_Basic_bCheckInputs is set to TRUE.
rInitialAndFinalSpeed_X	REAL	X-axis initial and final speed [Hz]
riTargetSpeed_X		X-axis target speed [Hz]
rInitialAndFinalSpeed_Y		Y-axis initial and final speed [Hz]
riTargetSpeed_Y		Y-axis target speed [Hz]
dutAdditionalOutputs	FPΣ, FP-X: PulseOutput_Linear_AdditionalOutputs_DUT	

Example In this example the function has been programmed in ladder diagram (LD). Please refer to the online help for a structured text (ST) example. The same POU header is used for all programming languages.

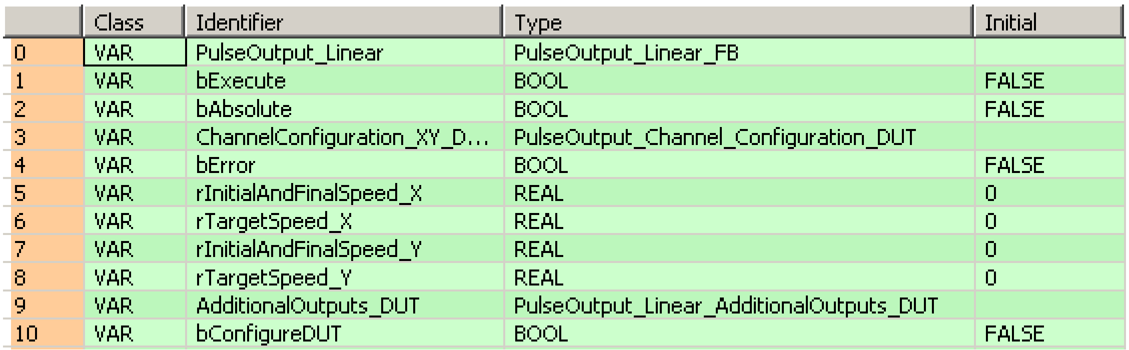
DUT Use the following predefined DUT: PulseOutput_Channel_Configuration_DUT



PulseOutput_Cha...tion_DUT [DUT]

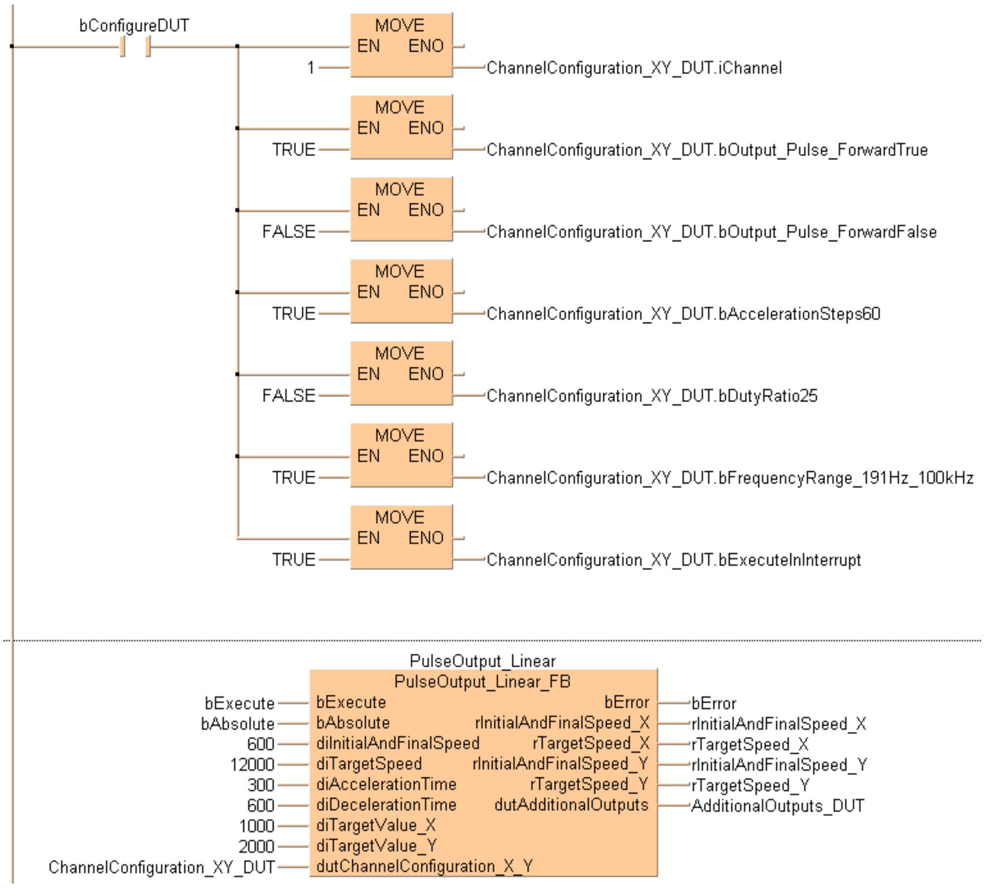
	Identifier	Type	Initial	Comment
0	iChannel	INT	0	FP-SIGMA: 0, 2
1	bOutput_Pulse_ForwardTrue	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
2	bOutput_Pulse_ForwardFalse	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
3	bAccelerationSteps60	BOOL	TRUE	FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps)
4	bDutyRatio25	BOOL	TRUE	FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%)
5	bFrequencyRange_48Hz_100kHz	BOOL	FALSE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz
6	bFrequencyRange_191Hz_100kHz	BOOL	TRUE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz
7	bPulseWidth80µs	BOOL	FALSE	FP0, FP-e Home, Trapezoidal: 80µs (else 50%)
8	iDutyRatioIn10PercentSteps	INT	0	FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs
9	bEnableHomeOnlyAfterNearHomeDeceleration	BOOL	FALSE	FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1)
10	iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms	INT	0	FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms]
11	bCalculationOnly	BOOL	FALSE	FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output)
12	bTrapezoidalMaximumTargetSpeed50kHz	BOOL	FALSE	FP0R: Output operation: Type 1: The target speed can be up to the maximum s...
13	bExecuteInInterrupt	BOOL	FALSE	FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (...
14	bJogWithNoCounting	BOOL	FALSE	Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b...
15	bContinueAfterDone	BOOL	FALSE	FP-SIGMA circular pulse output: 0=Execution stops when target value has been...

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	PulseOutput_Linear	PulseOutput_Linear_FB	
1	VAR	bExecute	BOOL	FALSE
2	VAR	bAbsolute	BOOL	FALSE
3	VAR	ChannelConfiguration_XY_D...	PulseOutput_Channel_Configuration_DUT	
4	VAR	bError	BOOL	FALSE
5	VAR	rInitialAndFinalSpeed_X	REAL	0
6	VAR	rTargetSpeed_X	REAL	0
7	VAR	rInitialAndFinalSpeed_Y	REAL	0
8	VAR	rTargetSpeed_Y	REAL	0
9	VAR	AdditionalOutputs_DUT	PulseOutput_Linear_AdditionalOutputs_DUT	
10	VAR	bConfigureDUT	BOOL	FALSE

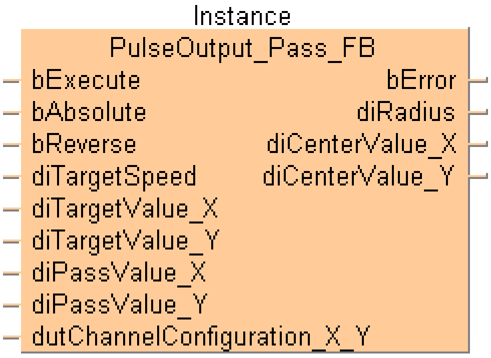
LD


bConfigureDUT
MOVE
EN ENO
1
ChannelConfiguration_XY_DUT.iChannel
MOVE
EN ENO
TRUE
ChannelConfiguration_XY_DUT.bOutput_Pulse_ForwardTrue
MOVE
EN ENO
FALSE
ChannelConfiguration_XY_DUT.bOutput_Pulse_ForwardFalse
MOVE
EN ENO
TRUE
ChannelConfiguration_XY_DUT.bAccelerationSteps60
MOVE
EN ENO
FALSE
ChannelConfiguration_XY_DUT.bDutyRatio25
MOVE
EN ENO
TRUE
ChannelConfiguration_XY_DUT.bFrequencyRange_191Hz_100kHz
MOVE
EN ENO
TRUE
ChannelConfiguration_XY_DUT.bExecuteInInterrupt
PulseOutput_Linear
PulseOutput_Linear_FB
bExecute
bExecute
bError
bError
bAbsolute
bAbsolute
rInitialAndFinalSpeed_X
rInitialAndFinalSpeed_X
600
diInitialAndFinalSpeed
rTargetSpeed_X
rTargetSpeed_X
12000
diTargetSpeed
rInitialAndFinalSpeed_Y
rInitialAndFinalSpeed_Y
300
diAccelerationTime
rTargetSpeed_Y
rTargetSpeed_Y
600
diDecelerationTime
dutAdditionalOutputs
AdditionalOutputs_DUT
1000
diTargetValue_X
2000
diTargetValue_Y
ChannelConfiguration_XY_DUT
dutChannelConfiguration_X_Y

PulseOutput_Pass_FB

Circular interpolation (pass position)

Description Pulses are output from two channels in accordance with the parameters in the function block and in the specified DUT, so that the path to the target position forms an arc. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help: F176_PulseOutput_Pass (see page 1081). Use PulseInfo_IsActive (see page 1223) to check if the control flag for the selected channel is FALSE.

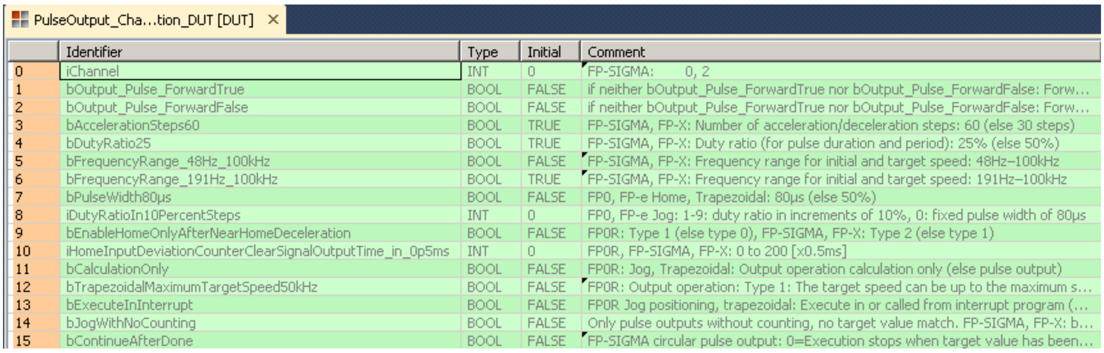
PLC types see page 1330

Data types

Input variable	Data type	Function
bExecute	BOOL	Activates the function block
bAbsolute		Absolute value control = TRUE, Relative value control = FALSE
bCounterclockwise		Operation connection mode: Reverse = TRUE, Forward = FALSE
diTargetSpeed	DINT	Target speed: Composite speed of both axes = 100–20000 (100Hz–20kHz)
diTargetValue_X		Target value [pulses]: -8388608–8388607
diTargetValue_Y		
diPassValue_X		
diPassValue_Y		
dutChannelConfiguration_X_Y	Predefined system DUT for channel configuration: PulseOutput_Channel_Configuration_DUT Channel: 0, 2	
Output variable	**Data type**	**Function**
bError	BOOL	Refers to an internal mismatch of input values to avoid a PLC error.
diRadius	DINT	Radius [pulses]
diCenterValue_X		X-axis center value [pulses] = -8388608–8388607
diCenterValue_Y		Y-axis center value [pulses] = -8388608–8388607

Example In this example the function has been programmed in ladder diagram (LD). Please refer to the online help for a structured text (ST) example. The same POU header is used for all programming languages.

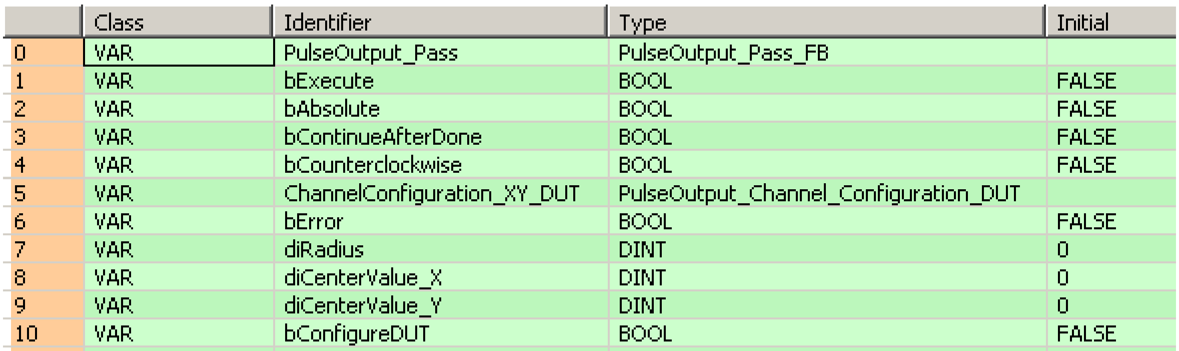
DUT Use the following predefined DUT: PulseOutput_Channel_Configuration_DUT



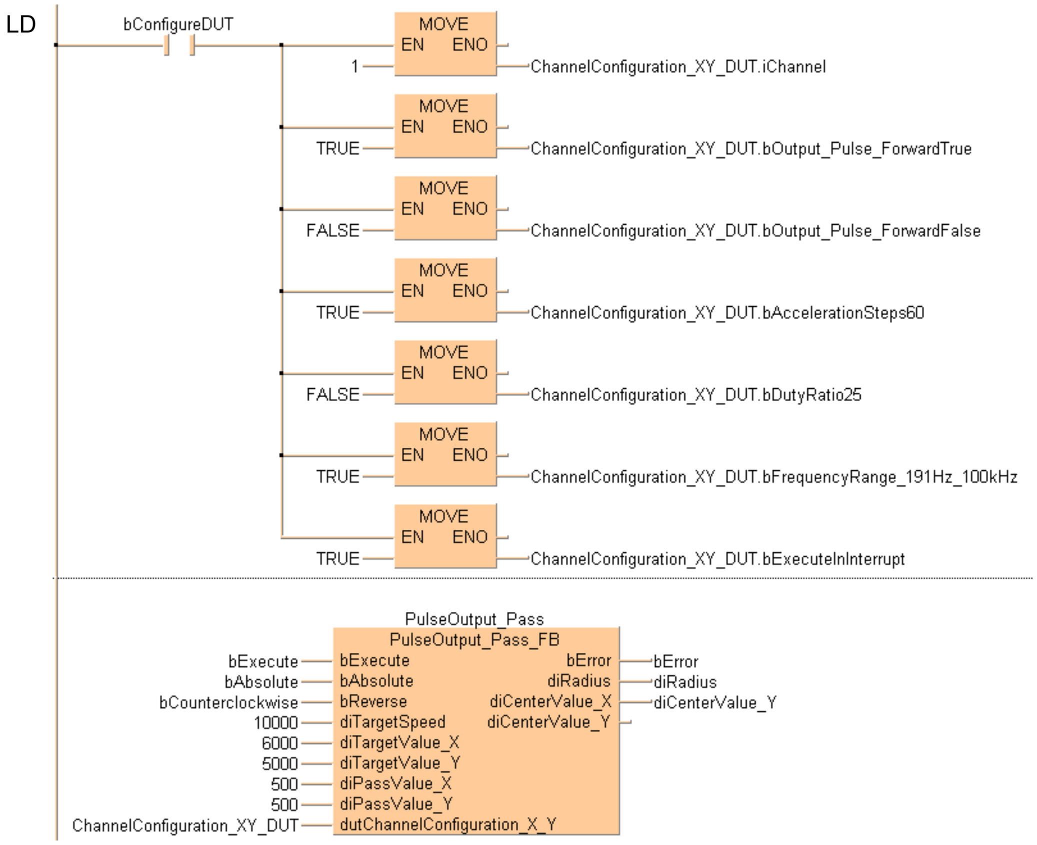
PulseOutput_Cha...tion_DUT [DUT]

	Identifier	Type	Initial	Comment
0	iChannel	INT	0	FP-SIGMA: 0, 2
1	bOutput_Pulse_ForwardTrue	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
2	bOutput_Pulse_ForwardFalse	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
3	bAccelerationSteps60	BOOL	TRUE	FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps)
4	bDutyRatio25	BOOL	TRUE	FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%)
5	bFrequencyRange_48Hz_100kHz	BOOL	FALSE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz
6	bFrequencyRange_191Hz_100kHz	BOOL	TRUE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz
7	bPulseWidth80µs	BOOL	FALSE	FP0, FP-e Home, Trapezoidal: 80µs (else 50%)
8	iDutyRatioIn10PercentSteps	INT	0	FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs
9	bEnableHomeOnlyAfterNearHomeDeceleration	BOOL	FALSE	FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1)
10	iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms	INT	0	FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms]
11	bCalculationOnly	BOOL	FALSE	FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output)
12	bTrapezoidalMaximumTargetSpeed50kHz	BOOL	FALSE	FP0R: Output operation: Type 1: The target speed can be up to the maximum s...
13	bExecuteInInterrupt	BOOL	FALSE	FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (...
14	bJogWithNoCounting	BOOL	FALSE	Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b...
15	bContinueAfterDone	BOOL	FALSE	FP-SIGMA circular pulse output: 0=Execution stops when target value has been...

POU header All input and output variables used for programming this function have been declared in the POU header.



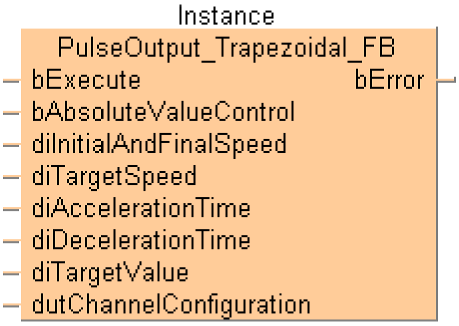
	Class	Identifier	Type	Initial
0	VAR	PulseOutput_Pass	PulseOutput_Pass_FB	
1	VAR	bExecute	BOOL	FALSE
2	VAR	bAbsolute	BOOL	FALSE
3	VAR	bContinueAfterDone	BOOL	FALSE
4	VAR	bCounterclockwise	BOOL	FALSE
5	VAR	ChannelConfiguration_XY_DUT	PulseOutput_Channel_Configuration_DUT	
6	VAR	bError	BOOL	FALSE
7	VAR	diRadius	DINT	0
8	VAR	diCenterValue_X	DINT	0
9	VAR	diCenterValue_Y	DINT	0
10	VAR	bConfigureDUT	BOOL	FALSE


LD
bConfigureDUT
MOVE
EN ENO
1
ChannelConfiguration_XY_DUT.iChannel
MOVE
EN ENO
TRUE
ChannelConfiguration_XY_DUT.bOutput_Pulse_ForwardTrue
MOVE
EN ENO
FALSE
ChannelConfiguration_XY_DUT.bOutput_Pulse_ForwardFalse
MOVE
EN ENO
TRUE
ChannelConfiguration_XY_DUT.bAccelerationSteps60
MOVE
EN ENO
FALSE
ChannelConfiguration_XY_DUT.bDutyRatio25
MOVE
EN ENO
TRUE
ChannelConfiguration_XY_DUT.bFrequencyRange_191Hz_100kHz
MOVE
EN ENO
TRUE
ChannelConfiguration_XY_DUT.bExecuteInInterrupt
PulseOutput_Pass
PulseOutput_Pass_FB
bExecute
bExecute
bError
bError
bAbsolute
bAbsolute
diRadius
diRadius
bCounterclockwise
bReverse
diCenterValue_X
diCenterValue_Y
10000
diTargetSpeed
diCenterValue_Y
6000
diTargetValue_X
5000
diTargetValue_Y
500
diPassValue_X
500
diPassValue_Y
ChannelConfiguration_XY_DUT
dutChannelConfiguration_X_Y

PulseOutput_Trapezoidal_FB

Trapezoidal control

Description This instruction automatically performs trapezoidal control according to the parameters in the function block and in the specified DUT. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.



This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help:

- FPΣ, FP-X, FP0R: F171_PulseOutput_Trapezoidal (see page 1045)
- FP0, FP-e: F168_PulseOutput_Trapezoidal (see page 1032)

Use PulseInfo_IsActive (see page 1223) to check if the control flag for the selected channel is FALSE.

PLC types see page 1330

Data types

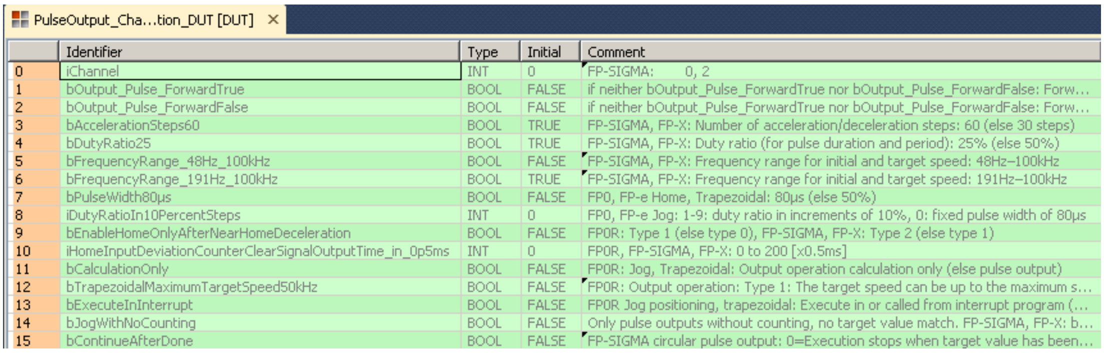
Input variable	Data type	Function
bExecute	BOOL	FP-SIGMA, FP-X, FP0, FP-e: Only with edge trigger FP0R: With edge or permanent if change of speed required
bAbsoluteValueControl		Absolute value control = TRUE, Relative value control = FALSE
diInitialAndFinalSpeed	DINT	Initial and final speed: Set this value according to the frequency range selected in PulseOutput_Channel_Configuration_DUT: FPΣ, FP-X: 1 to 9800 (1.5Hz–9.8kHz) 48 to 100000 (48Hz–100kHz) 191 to 100000 (191–100kHz) FP0R: 1 to 50000 (1Hz–50kHz) FP0, FP-e: 40 to 5000 (40Hz–5kHz)
diTargetSpeed		Target speed: Set this value according to the frequency range selected in PulseOutput_Channel_Configuration_DUT: FPΣ, FP-X: 1 to 9800 (1.5Hz–9.8kHz) 48 to 100000 (48Hz–100kHz) 191 to 100000 (191–100kHz) FP0R: 1 to 50000 (1Hz–50kHz) FP0, FP-e: 40 to 5000 (40Hz–5kHz)
diAccelerationTime		Acceleration/deceleration time (FPΣ, FP-X): With 30 steps: 30ms–32760ms (specify in steps of 30) With 60 steps: 60ms–32760ms (specify in steps of 60) Acceleration/deceleration time (FP0, FP-e): 30ms–32760ms Acceleration time (FP0R): 1ms–32760ms
diDecelerationTime		Deceleration time (FP0R): 1ms–32760ms
diTargetValue		Target value [pulses]: -2147483648–2147483647

Data types

Input variable	Data type	Function
dutChannelConfiguration	Predefined system DUT for channel configuration: PulseOutput_Channel_Configuration_DUT	
Output variable	**Data type**	**Function**
bError	BOOL	Refers to an internal mismatch of input values to avoid a PLC error.

Example In this example the function has been programmed in ladder diagram (LD). Please refer to the online help for a structured text (ST) example. The same POU header is used for all programming languages.

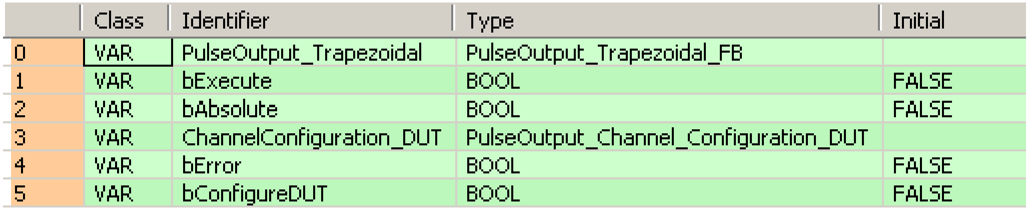
DUT Use the following predefined DUT: PulseOutput_Channel_Configuration_DUT



PulseOutput_Cha...tion_DUT [DUT]

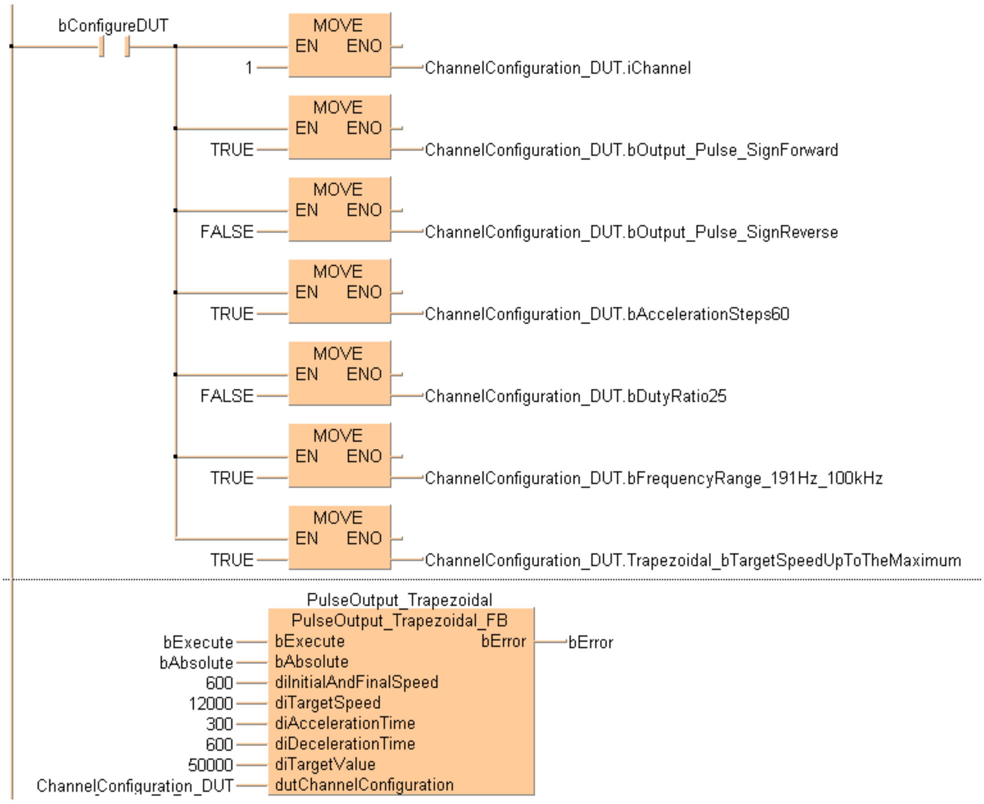
	Identifier	Type	Initial	Comment
0	iChannel	INT	0	FP-SIGMA: 0, 2
1	bOutput_Pulse_ForwardTrue	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
2	bOutput_Pulse_ForwardFalse	BOOL	FALSE	if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw...
3	bAccelerationSteps60	BOOL	TRUE	FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps)
4	bDutyRatio25	BOOL	TRUE	FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%)
5	bFrequencyRange_48Hz_100kHz	BOOL	FALSE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz
6	bFrequencyRange_191Hz_100kHz	BOOL	TRUE	FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz
7	bPulseWidth80µs	BOOL	FALSE	FP0, FP-e Home, Trapezoidal: 80µs (else 50%)
8	iDutyRatioIn10PercentSteps	INT	0	FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs
9	bEnableHomeOnlyAfterNearHomeDeceleration	BOOL	FALSE	FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1)
10	iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms	INT	0	FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms]
11	bCalculationOnly	BOOL	FALSE	FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output)
12	bTrapezoidalMaximumTargetSpeed50kHz	BOOL	FALSE	FP0R: Output operation: Type 1: The target speed can be up to the maximum s...
13	bExecuteInInterrupt	BOOL	FALSE	FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (...
14	bJogWithNoCounting	BOOL	FALSE	Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b...
15	bContinueAfterDone	BOOL	FALSE	FP-SIGMA circular pulse output: 0=Execution stops when target value has been...

POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	PulseOutput_Trapezoidal	PulseOutput_Trapezoidal_FB	
1	VAR	bExecute	BOOL	FALSE
2	VAR	bAbsolute	BOOL	FALSE
3	VAR	ChannelConfiguration_DUT	PulseOutput_Channel_Configuration_DUT	
4	VAR	bError	BOOL	FALSE
5	VAR	bConfigureDUT	BOOL	FALSE

LD


bConfigureDUT
MOVE
EN ENO
1
ChannelConfiguration_DUT.iChannel
MOVE
EN ENO
TRUE
ChannelConfiguration_DUT.bOutput_Pulse_SignForward
MOVE
EN ENO
FALSE
ChannelConfiguration_DUT.bOutput_Pulse_SignReverse
MOVE
EN ENO
TRUE
ChannelConfiguration_DUT.bAccelerationSteps60
MOVE
EN ENO
FALSE
ChannelConfiguration_DUT.bDutyRatio25
MOVE
EN ENO
TRUE
ChannelConfiguration_DUT.bFrequencyRange_191Hz_100kHz
MOVE
EN ENO
TRUE
ChannelConfiguration_DUT.Trapezoidal_bTargetSpeedUpToTheMaximum
PulseOutput_Trapezoidal
PulseOutput_Trapezoidal_FB
bExecute
bExecute
bError
bError
bAbsolute
bAbsolute
600
diInitialAndFinalSpeed
12000
diTargetSpeed
300
diAccelerationTime
600
diDecelerationTime
50000
diTargetValue
ChannelConfiguration_DUT
dutChannelConfiguration

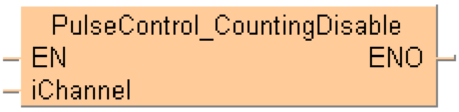
39.3 Pulse control instructions

In this section:

PulseControl_Counting Disable

Disables counting on a pulse output channel

Description This instruction disables counting on the channel specified by **iChannel**. Bit 1 of the pulse output control code (see page 1021) is set to TRUE.



See also:

- Pulse output tool instructions in the online help
- PulseInfo_IsCountingDisabled (see page 1225)
- PulseControl_CountingEnable (see page 1200)

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

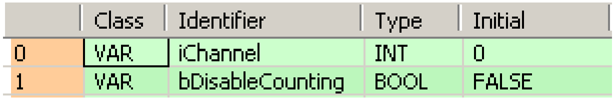
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	BOOL	TRUE if pulse counting has been disabled

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

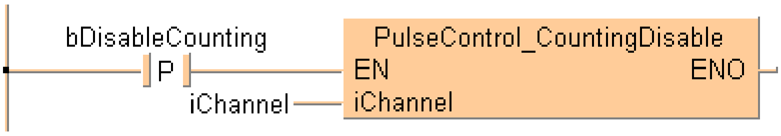
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	bDisableCounting	BOOL	FALSE

Body

LD



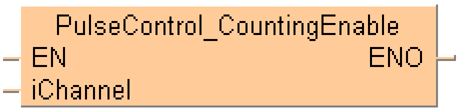
ST When programming with structured text, enter the following:

```
if DF(bDisableCounting) then
    PulseControl_CountingDisable(iChannel := iChannel);
end_if;
```

PulseControl_CountingEnable

Enables counting on a pulse output channel

Description This instruction enables counting on the pulse output channel specified by **iChannel** after counting has been disabled with PulseControl_CountingDisable (see page 1198). Bit 1 of the pulse output control code (see page 1021) is set to FALSE.



See also:

- Pulse output tool instructions in the online help
- PulseControl_CountingDisable (see page 1198)
- PulseInfo_IsCountingDisabled (see page 1225)

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

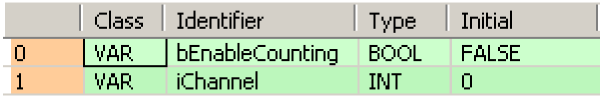
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

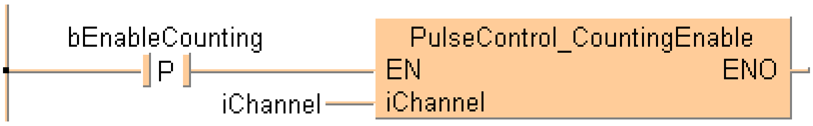
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bEnableCounting	BOOL	FALSE
1	VAR	iChannel	INT	0

Body

LD



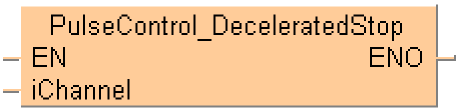
ST When programming with structured text, enter the following:

```
if DF(bEnableCounting) then
    PulseControl_CountingEnable(iChannel := iChannel);
end_if;
```

PulseControl_DeceleratedStop

Performs a decelerated stop

Description This instruction performs a decelerated stop on the channel specified by **iChannel**. When a decelerated stop is requested during acceleration, deceleration is performed with the same slope as deceleration from the target speed.



See also:

- Pulse output tool instructions in the online help
- PulseControl_PulseOutputStop (see page 1213)
- PulseInfo_IsPulseOutputStopped (see page 1228)
- FP instructions Writing the pulse output control code (see page 1021) (FP0R)

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

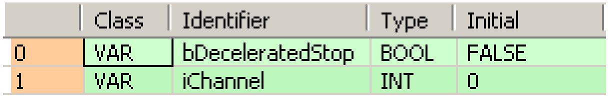
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

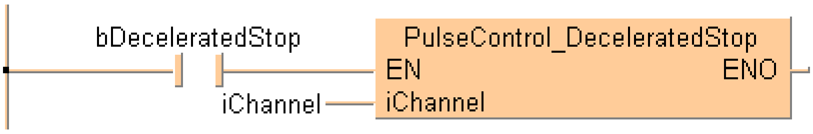
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bDeceleratedStop	BOOL	FALSE
1	VAR	iChannel	INT	0

Body

LD



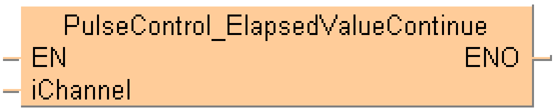
ST When programming with structured text, enter the following:

```
if DF(bDeceleratedStop) then
    PulseControl_DeceleratedStop(iChannel := iChannel);
end_if;
```

PulseControl_ElapsedValueContinue

Continues pulse counting after reset

Description This instruction resumes pulse counting on the channel specified by **iChannel** after a reset of the elapsed value using PulseControl_ElapsedValueReset (see page 1206). Bit 0 of the pulse output control code (see page 1021) is set to FALSE.



See also:

- Pulse output tool instructions in the online help
- PulseInfo_IsElapsedValueReset (see page 1226)
- PulseInfo_ReadElapsedValue (see page 1233)
- PulseControl_ElapsedValueContinue (see page 1206)
- PulseControl_WriteElapsedValue (see page 1216)

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

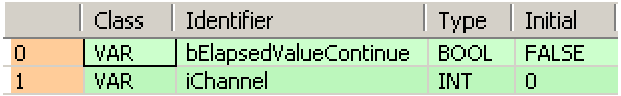
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

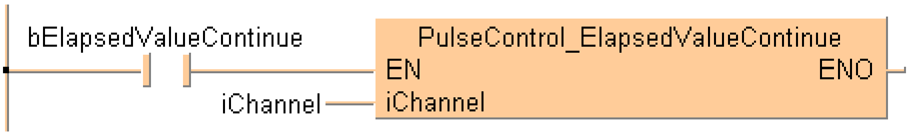
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bElapsedValueContinue	BOOL	FALSE
1	VAR	iChannel	INT	0

Body

LD



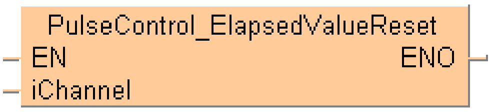
ST When programming with structured text, enter the following:

```
if (bElapsedValueContinue) then
    PulseControl_ElapsedValueContinue(iChannel := iChannel);
end_if;
```

PulseControl_ElapsedValueReset

Sets elapsed value to 0

Description This instruction sets the elapsed value of the pulse output channel specified by iChannel to 0. Bit 0 of the pulse output control code (see page 1021) is set to TRUE. Use PulseControl_ElapsedValueContinue (see page 1204) to continue counting on the pulse output channel. Use PulseInfo_IsElapsedValueReset (see page 1226) to check the current state. Pulse output continues when resetting the elapsed value.



See also:

- Pulse output tool instructions in the online help
- PulseInfo_IsElapsedValueReset (see page 1226)
- PulseControl_ElapsedValueContinue (see page 1204)
- PulseControl_WriteElapsedValue (see page 1216)

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

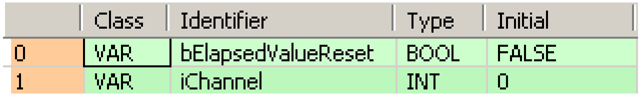
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

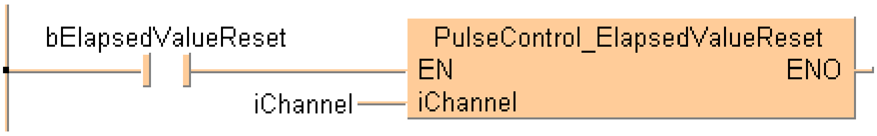
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bElapsedValueReset	BOOL	FALSE
1	VAR	iChannel	INT	0

Body

LD



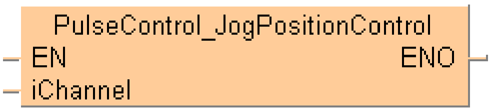
ST When programming with structured text, enter the following:

```
if (bElapsedValueReset) then
    PulseControl_ElapsedValueReset(iChannel := iChannel);
end_if;
```

PulseControl_JogPositionControl

Starts position control

Description This instruction sets and resets bit 6 of the pulse output control code (see page 1021) to start position control on the channel specified by **iChannel**. The position control trigger is used with the JOG operation instructions PulseOutput_Jog_Positioning0_FB (see page 1181) and PulseOutput_Jog_Positioning1_FB (see page 1184).



See also:

- Pulse output tool instructions in the online help
- FP instructions F171_PulseOutput_Jog_Positioning (see page 1055)

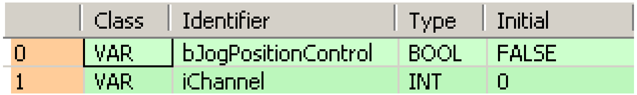
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

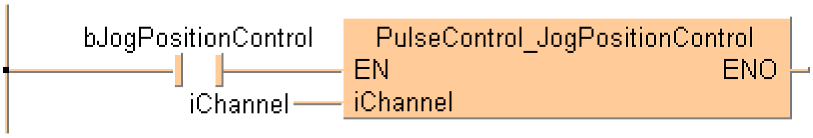
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bJogPositionControl	BOOL	FALSE
1	VAR	iChannel	INT	0

Body

LD



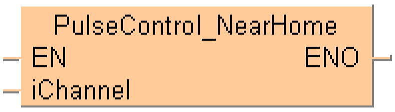
ST When programming with structured text, enter the following:

```
if (bJogPositionControl) then
    PulseControl_JogPositionControl(iChannel := iChannel);
end_if;
```

PulseControl_NearHome

Starts deceleration when near home

Description This instruction starts deceleration on the channel specified by **iChannel** when near the home input by setting bit 4 of the pulse output control code (see page 1021) to TRUE and back to FALSE again. Use PulseInfo_IsHomeInputTrue (see page 1227) to check if the home input is TRUE.



See also:

- Tool instructions PulseOutput_Home_FB (see page 1176), Pulse output tool instructions in the online help
- FP instructions

F168_PulseOutput_Home (see page 1035) (FP0, FP-e)

F171_PulseOutput_Home (see page 1051) (FP$_{\Sigma}$, FP-X)

F177_PulseOutput_Home (see page 1085) (FP0R)

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

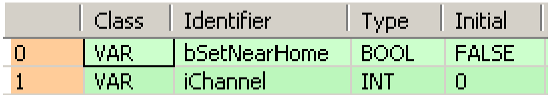
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

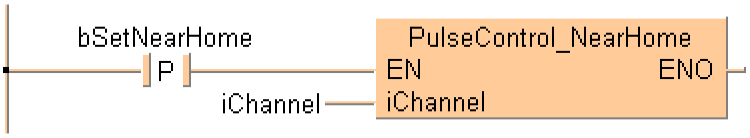
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bSetNearHome	BOOL	FALSE
1	VAR	iChannel	INT	0

Body

LD



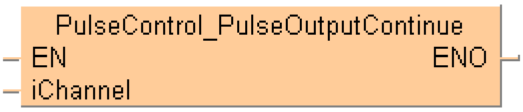
ST When programming with structured text, enter the following:

```
if DF(bSetNearHome) then
    PulseControl_NearHome(iChannel := iChannel);
end_if;
```

PulseControl_PulseOutputContinue

Continues pulse output

Description This instruction continues pulse output at the channel specified by **iChannel** after pulse output has been stopped using PulseControl_PulseOutputStop (see page 1213). Bit 3 of the pulse output control code (see page 1021) is set to FALSE.



See also:

- Pulse output tool instructions in the online help
- PulseControl_PulseOutputStop (see page 1213)
- PulseInfo_IsPulseOutputStopped (see page 1228)

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

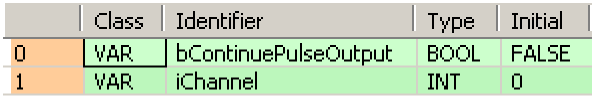
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

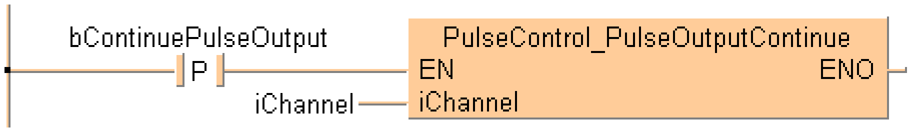
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bContinuePulseOutput	BOOL	FALSE
1	VAR	iChannel	INT	0

Body

LD



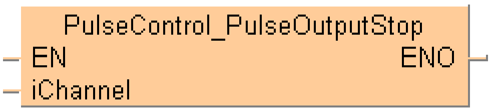
ST When programming with structured text, enter the following:

```
if DF(bContinuePulseOutput) then
    PulseControl_PulseOutputContinue(iChannel := iChannel);
end_if;
```

PulseControl_PulseOutputStop

Stops pulse output

Description This instruction stops the pulse output on the channel specified by **iChannel** by setting bit 3 of the pulse output control code (see page 1021) to TRUE. Use PulseControl_PulseOutputContinue (see page 1211) to continue pulse output after the interrupt.



See also:

- PulseInfo_IsPulseOutputStopped (see page 1228)
- PulseControl_PulseOutputContinue (see page 1211)
- Stopping pulse output

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

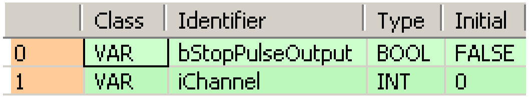
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

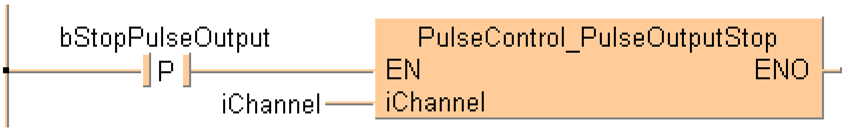
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bStopPulseOutput	BOOL	FALSE
1	VAR	iChannel	INT	0

Body

LD



ST

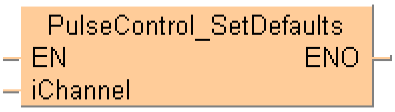
When programming with structured text, enter the following:

```
if DF(bStopPulseOutput) then
    PulseControl_PulseOutputStop(iChannel := iChannel);
end_if;
```

PulseControl_SetDefaults

Sets defaults for pulse output channel

Description This instruction sets all bits of the pulse output control code (see page 1021) of the channel specified by **iChannel** to 0. 0 is the default setting.



To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

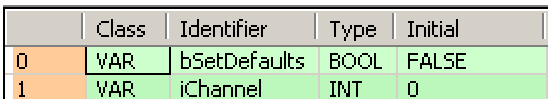
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

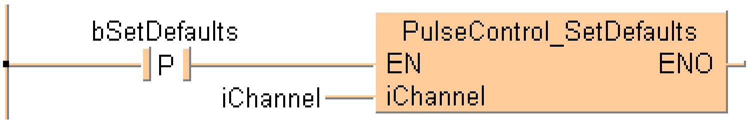
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bSetDefaults	BOOL	FALSE
1	VAR	iChannel	INT	0

Body

LD



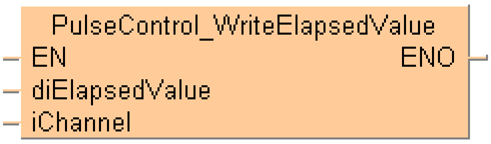
ST When programming with structured text, enter the following:

```
if DF(bSetDefaults) then
    PulseControl_SetDefaults(iChannel := iChannel);
end_if;
```

PulseControl_WriteElapsedValue

Writes elapsed value into a pulse output channel

Description This instruction writes an elapsed value into pulse output channel specified by **iChannel**.



See also:

- Pulse output tool instructions in the online help
- PulseInfo_IsElapsedValueReset (see page 1226)
- PulseControl_ElapsedValueContinue (see page 1204)
- PulseInfo_ReadElapsedValue (see page 1233)
- Writing and reading the elapsed value (see page 1026)

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

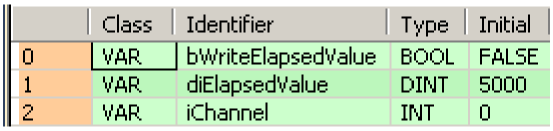
PLC types see page 1329

Data types

Variable	Data type	Function
diElapsedValue	DINT	Elapsed value to be written into the channel specified by **iChannel**
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

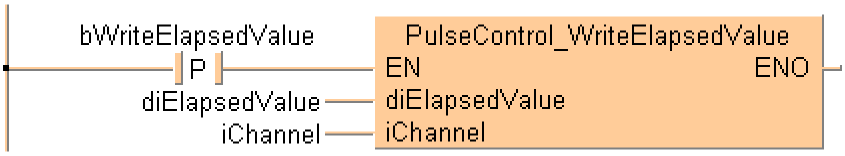
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	bWriteElapsedValue	BOOL	FALSE
1	VAR	diElapsedValue	DINT	5000
2	VAR	iChannel	INT	0

Body

LD



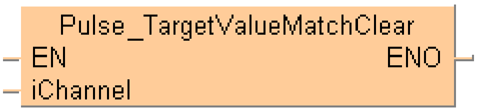
ST When programming with structured text, enter the following:

```
if DF(bWriteElapsedValue) then
    PulseControl_WriteElapsedValue(ElapsedValue := diElapsedValue,
    iChannel := iChannel);
end_if;
```

Pulse_TargetValue MatchClear

Clears target value match control

Description This instruction clears the target value match control on the channel specified by **iChannel**.



See also:

- Pulse output tool instructions in the online help
- PulseInfo_IsTargetValueMatchActive (see page 1229)
- PulseInfo_ReadTargetValueMatchValue (see page 1235)

To add an enable input and enable output to the instruction, select [With EN/ENO] from the "Instructions" pane (LD, FBD or IL editor). To reuse an instruction select "Recently used" from the context menu or press <Ctrl>+<Shift>+<v> in the programming window.

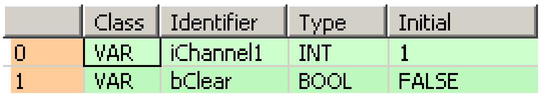
PLC types see page 1335

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

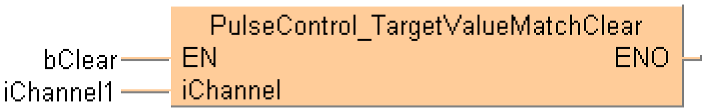
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel1	INT	1
1	VAR	bClear	BOOL	FALSE

Body

LD



ST When programming with structured text, enter the following:

```
if DF(bClearTargetValueMatch) then
    Pulse_TargetValueMatchClear(iChannel := iChannel);
end_if;
```

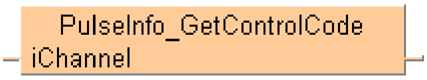
39.4 Pulse information instructions

In this section:

PulseInfo_GetControlCode

Returns control code of pulse output channel

Description This instruction returns the control code (see page 1021) of the pulse output channel specified by **iChannel**.



See also: Pulse output tool instructions in the online help

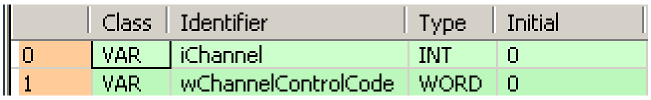
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	WORD	Stores the control code

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

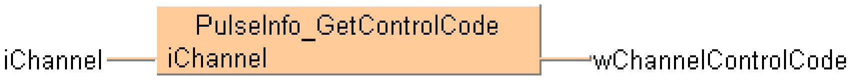
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	wChannelControlCode	WORD	0

Body

LD



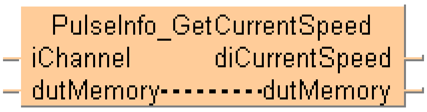
ST When programming with structured text, enter the following:

```
if (bReadControlCode) then
  wChannelControlCode := PulseInfo_GetControlCode(iChannel := iChannel);
end_if;
```

PulseInfo_GetCurrentSpeed

Returns current speed on pulse output channel

Description This instruction returns the current speed in Hz of the pulse output channel specified by **iChannel**.



See also: Pulse output tool instructions in the online help

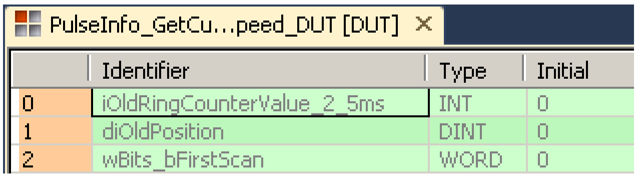
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
dutMemory	PulseInfo_GetCurrentSpeed_DUT	
diCurrentSpeed	DINT	Current speed in Hz

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

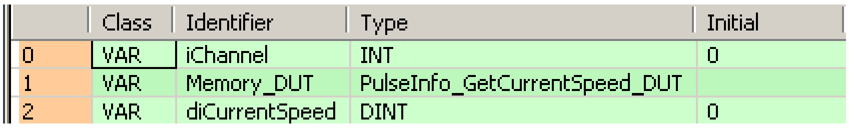
DUT Use the following predefined DUT: PulseInfo_GetCurrentSpeed_DUT



PulseInfo_GetCu...peed_DUT [DUT]

	Identifier	Type	Initial
0	iOldRingCounterValue_2_5ms	INT	0
1	diOldPosition	DINT	0
2	wBits_bFirstScan	WORD	0

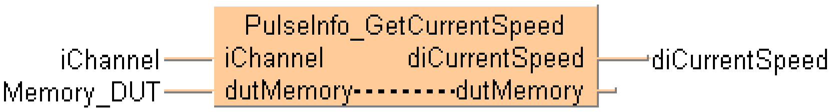
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	Memory_DUT	PulseInfo_GetCurrentSpeed_DUT	
2	VAR	diCurrentSpeed	DINT	0

Body

LD



ST When programming with structured text, enter the following:

```
if (bGetCurrentSpeed) then
    PulseInfo_GetCurrentSpeed(iChannel := iChannel,
        dutMemory := Memory_DUT,
        diCurrentSpeed => diCurrentSpeed);
end_if;
```

PulseInfo_IsActive

Check if pulse output is active

Description This instruction evaluates the pulse output control flag and returns TRUE if the pulse output channel specified by **iChannel** is active.

PulseInfo_IsActive
iChannel

See also: Pulse output tool instructions in the online help

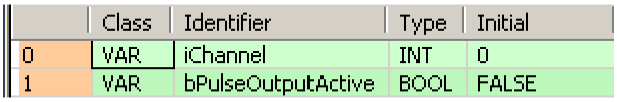
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	BOOL	TRUE if pulse output is active

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

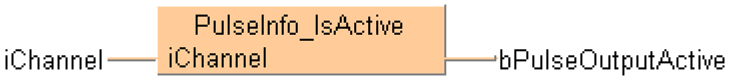
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	bPulseOutputActive	BOOL	FALSE

Body

LD



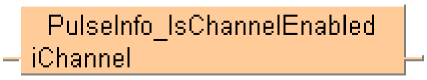
ST When programming with structured text, enter the following:

```
if (bPulseOutput_Check) then
  bPulseOutputActive := PulseInfo_IsActive(iChannel := iChannel);
end_if;
```

PulseInfo_IsChannel Enabled

Checks if pulse output channel is enabled

Description This instruction returns TRUE if the pulse output channel specified by **iChannel** has been enabled in the system registers and is supported by the selected PLC type.



See also:

- Pulse output tool instructions in the online help
- Required system register settings

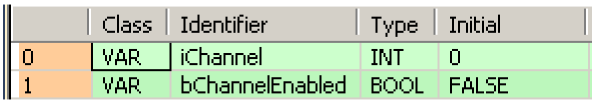
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	BOOL	TRUE if the pulse output channel specified by **iChannel** is enabled

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

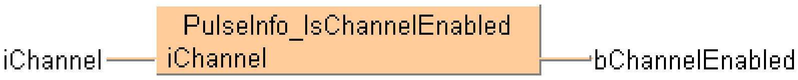
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	bChannelEnabled	BOOL	FALSE

Body

LD



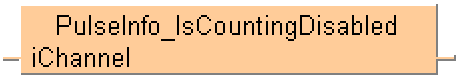
ST When programming with structured text, enter the following:

```
if (bPulseChannel_Check) then
    bChannelEnabled := PulseInfo_IsChannelEnabled(iChannel := iChannel);
end_if;
```

PulseInfo_IsCountingDisabled

Checks if pulse counting is disabled

Description This instruction returns TRUE if counting on the channel specified by **iChannel** has been disabled.



See also:

- Pulse output tool instructions in the online help
- PulseControl_CountingDisable (see page 1198)
- PulseControl_CountingEnable (see page 1200)
- FP instructions Enabling/disabling counting operations (see page 1021)

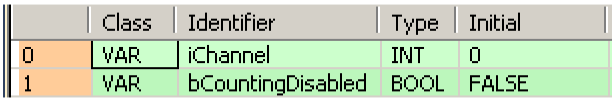
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

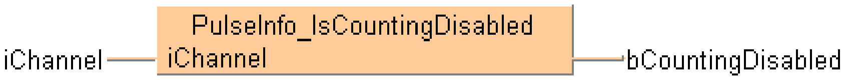
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	bCountingDisabled	BOOL	FALSE

Body

LD



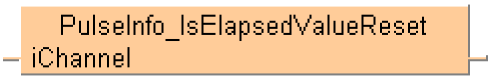
ST When programming with structured text, enter the following:

```
if (bChannelCounting_Check) then
    bCountingDisabled := PulseInfo_IsCountingDisabled(iChannel :=
iChannel);
end_if;
```

PulseInfo_IsElapsedValueReset

Checks if elapsed value is set to 0

Description This instruction returns TRUE if the elapsed value of the pulse output channel specified by **iChannel** has been reset to 0.



See also:

- Pulse output tool instructions in the online help
- PulseInfo_ReadElapsedValue (see page 1233)
- PulseControl_ElapsedValueReset (see page 1206)
- PulseControl_ElapsedValueContinue (see page 1204)

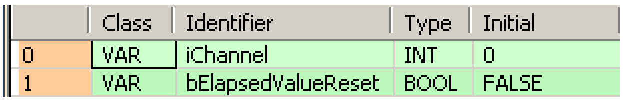
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	BOOL	TRUE if the channel specified by **iChannel** has been reset

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

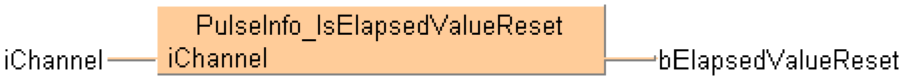
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	bElapsedValueReset	BOOL	FALSE

Body

LD



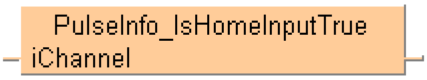
ST When programming with structured text, enter the following:

```
if (bIsElapsedValueReset) then
    bElapsedValueReset := PulseInfo_IsElapsedValueReset(iChannel :=
iChannel);
end_if;
```

PulseInfo_IsHomeInputTrue

Checks if home input is TRUE

Description This instruction returns TRUE if the home input of the channel specified by **iChannel** is TRUE.



See also:

- Pulse output tool instructions in the online help
- PulseOutput_Home_FB (see page 1176)

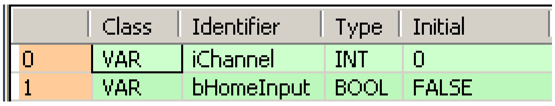
PLC types see page 1329

Data types

Variable	Data type	Function
Output variable	BOOL	TRUE if the home input has been reached

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

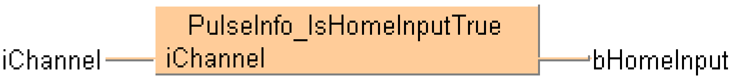
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	bHomeInput	BOOL	FALSE

Body

LD



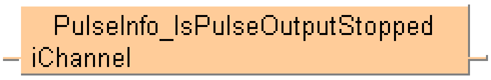
ST When programming with structured text, enter the following:

```
if (bHomeInput_Check) then
    bHomeInput := PulseInfo_IsHomeInputTrue(iChannel := iChannel);
end_if;
```

PulseInfo_IsPulseOutputStopped

Check if pulse output has stopped

Description This instruction returns TRUE if pulse output has been stopped, e.g. with PulseControl_DeceleratedStop (see page 1202) or PulseControl_PulseOutputStop (see page 1213). Use PulseControl_PulseOutputContinue (see page 1211) to resume pulse output.



See also:

- Pulse output tool instructions in the online help
- Stopping pulse output

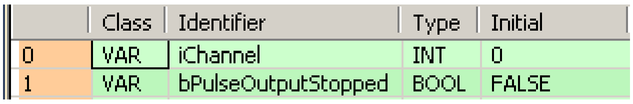
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	BOOL	TRUE if pulse output has been stopped

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

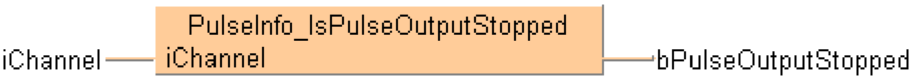
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	bPulseOutputStopped	BOOL	FALSE

Body

LD



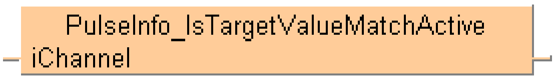
ST When programming with structured text, enter the following:

```
if (bPulseOutput_Check) then
    bPulseOutputStopped := PulseInfo_IsPulseOutputStopped(iChannel :=
iChannel);
end_if;
```

PulseInfo_IsTargetValueMatchActive

Checks if target value match control is active

Description This instruction returns TRUE if target value match control (see page 1237) is active on the channel specified by **iChannel**.



See also:

- Pulse output tool instructions in the online help
- PulseControl_TargetValueMatchClear (see page 1218)
- PulseInfo_ReadTargetValueMatchValue (see page 1235)

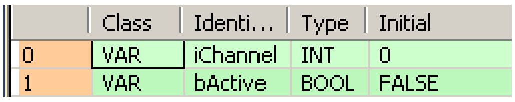
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	BOOL	TRUE if target value match control is active

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

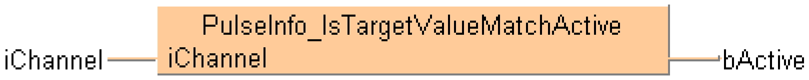
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identi...	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	bActive	BOOL	FALSE

Body

LD



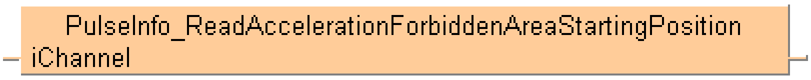
ST When programming with structured text, enter the following:

```
if (bTargetValueMatch_Check) then
    bActive := PulseInfo_IsTargetValueMatchActive(iChannel := iChannel);
end_if;
```

PulseInfo_ReadAccelerationForbiddenAreaStartingPosition

Read acceleration forbidden area starting position

Description This instruction reads the starting position of an acceleration forbidden area. If the elapsed value crosses over this position when the speed is being changed, acceleration cannot be continued any more.



See also: Pulse output tool instructions in the online help

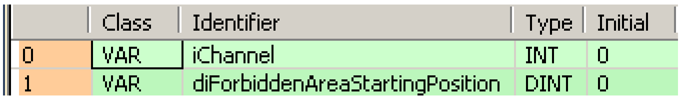
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	DINT	Stores the start position of the acceleration forbidden area

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

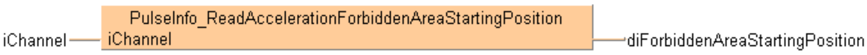
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	diForbiddenAreaStartingPosition	DINT	0

Body

LD


PulseInfo_ReadAccelerationForbiddenAreaStartingPosition
iChannel
iChannel
diForbiddenAreaStartingPosition

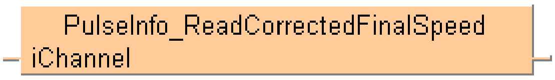
ST When programming with structured text, enter the following:

```
diForbiddenAreaStartingPosition :=
PulseInfo_ReadAccelerationForbiddenAreaStartingPosition(iChannel :=
iChannel);
```

PulseInfo_ReadCorrectedFinalSpeed

Reads corrected value of final speed

Description This instruction returns the value of the corrected final speed on the channel specified by **iChannel**.


PulseInfo_ReadCorrectedFinalSpeed
iChannel

See also: Pulse output tool instructions in the online help

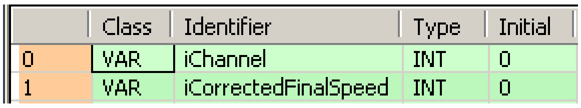
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable		Stores the value of the corrected final speed

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

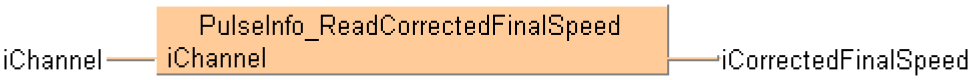
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	iCorrectedFinalSpeed	INT	0

Body

LD


iChannel
PulseInfo_ReadCorrectedFinalSpeed
iChannel
iCorrectedFinalSpeed

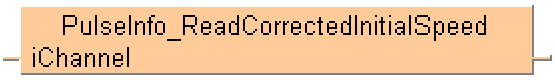
ST When programming with structured text, enter the following:

```
iCorrectedFinalSpeed:= PulseInfo_ReadCorrectedFinalSpeed(iChannel :=
iChannel);
```

PulseInfo_ReadCorrectedInitialSpeed

Reads corrected value of initial speed

Description This instruction returns the value of the corrected initial speed on the channel specified by **iChannel**.



See also: Pulse output tool instructions in the online help

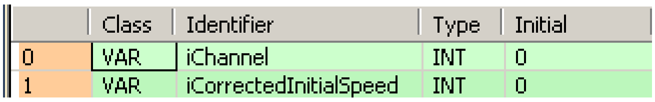
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable		Stores the value of the corrected initial speed

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

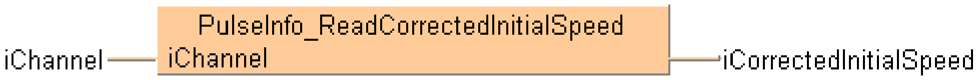
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	iCorrectedInitialSpeed	INT	0

Body

LD



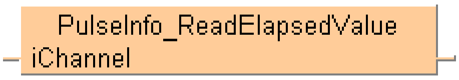
ST When programming with structured text, enter the following:

```
iCorrectedInitialSpeed := PulseInfo_ReadCorrectedInitialSpeed(iChannel :=
iChannel);
```

PulseInfo_ReadElapsedValue

Reads elapsed value from pulse output channel

Description This instruction reads the elapsed value from the pulse output channel specified by **iChannel**. Use PulseControl_WriteElapsedValue (see page 1216) to modify the elapsed value and PulseControl_ElapsedValueReset (see page 1206) to set the elapsed value to 0.



See also:

- Pulse output tool instructions in the online help
- PulseInfo_IsElapsedValueReset (see page 1226)
- PulseControl_ElapsedValueContinue (see page 1204)
- PulseControl_WriteElapsedValue (see page 1216)

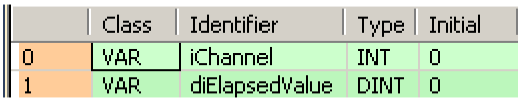
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	DINT	Stores the elapsed value from the channel specified by **iChannel**

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

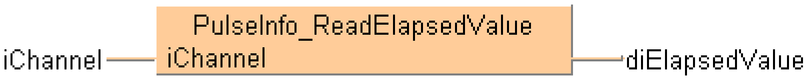
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	diElapsedValue	DINT	0

Body

LD



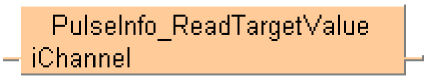
ST When programming with structured text, enter the following:

```
if (bRead) then
    diElapsedValue := PulseInfo_ReadElapsedValue(iChannel := iChannel);
end_if;
```

PulseInfo_ReadTarget Value

Reads target value from pulse output channel

Description This instruction reads the target value from the pulse output channel specified by **iChannel**.



See also: Tool instructions: overview of high-speed counter instructions

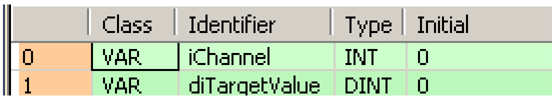
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	DINT	Stores the target value of the channel specified by **iChannel**

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

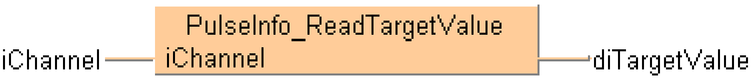
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	diTargetValue	DINT	0

Body

LD



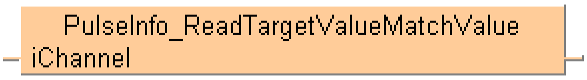
ST When programming with structured text, enter the following:

```
if (bRead) then
    diTargetValue := PulseInfo_ReadTargetValue(iChannel := iChannel);
end_if;
```

PulseInfo_ReadTargetValueMatchValue

Reads output control target value from pulse output channel

Description This instruction returns the output control target value of the pulse output channel specified by **iChannel**. The output control target value is used by the target value match instructions.



See also:

- Pulse output tool instructions in the online help
- Pulse_TargetValueMatch_Set (see page 1240)
- Pulse_TargetValueMatch_Reset (see page 1237)
- Pulse_TargetValueMatch_Clear (see page 1218)
- Info_IsTargetValueMatch_Active (see page 1229)

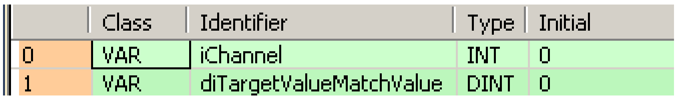
PLC types see page 1329

Data types

Variable	Data type	Function
iChannel	INT	Pulse output channel: FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
Output variable	DINT	Stores the output control target value

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for all programming languages.

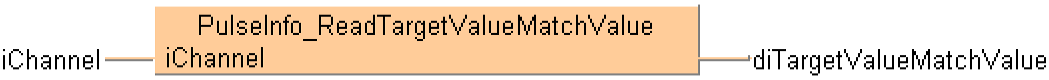
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR	iChannel	INT	0
1	VAR	diTargetValueMatchValue	DINT	0

Body

LD



ST When programming with structured text, enter the following:

```
if (bRead) then
  diTargetValueMatchValue := PulseInfo_ReadTargetValueMatchValue(iChannel
:= iChannel);
end_if;
```

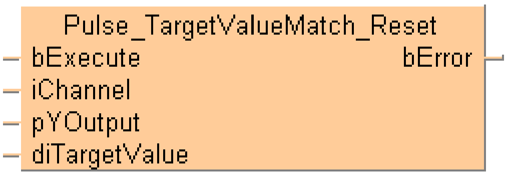
39.5 Pulse output target value match control

In this section:

Pulse_TargetValue Match_Reset

Target value match OFF (pulse output)

Description If the elapsed value matches the target value **diTargetValue** of the pulse output channel specified by **iChannel**, the output relay specified by **pYOutput** immediately turns to FALSE.



See also:

Pulse output tool instructions in the online help

This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help: F167_PulseOutput_Reset (see page 1029)

To validate the combination of channel and Y output, the compiler requires the following name pattern for global variables: %sPulse_TargetValueMatch_Channel%d_Y%d%s

Always use this pattern for global variables in target value match control.

- Channel%d must be a pulse output channel number enabled in the system registers
- Y%d must be an explicit output address supported by the PLC

FP-Σ, FP0, FP-e:	Y0–Y7
FP-Σ (V3.1 or higher), FP0R:	Y0–Y1F
FP-X:	Y0–Y29F

- %s is an optional descriptor at the beginning and the end of the pattern
- Y%d must be an explicit output address supported by the PLC

FP-Σ, FP0, FP-e:	Y0–Y7
FP-Σ (V3.1 or higher), FP0R:	Y0–Y1F
FP-X:	Y0–Y29F

- %s is an optional descriptor at the beginning and the end of the pattern

Optional	Fixed pattern	Optional
g_b	Pulse_TargetValueMatch_ChannelA_Y11F	_MotorOn

This global variable generates the code for channel **A** and output **Y11F**.

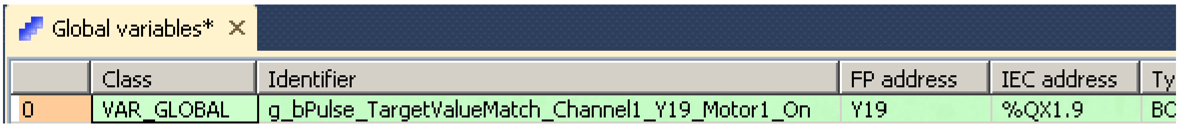
PLC types see see page 1329

Data types

Input variable	Data type	Function
bExecute	BOOL	A rising edge activates the function; evaluate the pulse output channel control flag using PulseInfo_IsTargetValueMatchActive (see page 1229)
iChannel	INT	FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
pYOutput	POINTER	Pointer result obtained by GetPointer from a global variable that supplies the channel number and output relay
diTargetValue	DINT	Specify a 32-bit data value for the target value within the following range: FP0, FP-e: -838808–+8388607 FPΣ, FP-X, FP0R: -2147483467–+2147483648
Output variable	**Data type**	**Function**
bError	BOOL	TRUE if the combination of **Channel%d** and **pYOuput.iOffset** does not match a valid combination of channel number and output relay as determined by the global variable

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). (ST).The same POU header is used for all programming languages.

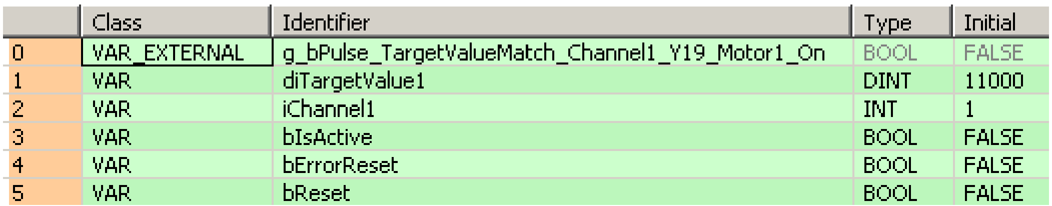
GVL In the global variable list you define variables that can be accessed by all POUs in the project.



Global variables*

	Class	Identifier	FP address	IEC address	Ty
0	VAR_GLOBAL	g_bPulse_TargetValueMatch_Channel1_Y19_Motor1_On	Y19	%QX1.9	BC

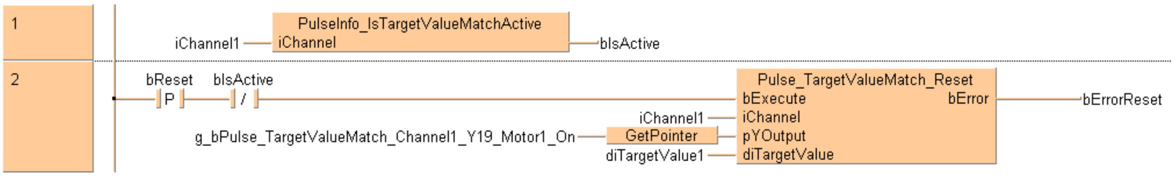
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR_EXTERNAL	g_bPulse_TargetValueMatch_Channel1_Y19_Motor1_On	BOOL	FALSE
1	VAR	diTargetValue1	DINT	11000
2	VAR	iChannel1	INT	1
3	VAR	bIsActive	BOOL	FALSE
4	VAR	bErrorReset	BOOL	FALSE
5	VAR	bReset	BOOL	FALSE

Body Use PulseInfo_IsTargetValueMatchActive (see page 1229) to evaluate the channel **iChannel1** is active. If a rising edge is detected at **bReset** and if **bIsActive** is **not** TRUE, the instruction is executed. The combination of channel number and output contact is validated in the global variable **g_bPulse_TargetValueMatch_Channel1_Y19_Motor1_On**. When pulse output on channel 1 reaches the target value **diTargetValue1**, output **Y19** is set to FALSE.

LD

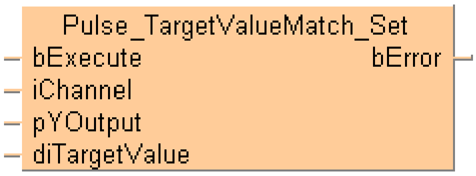


```
ST bIsActive:=PulseInfo_IsTargetValueMatchActive(iChannel1);
   Pulse_TargetValueMatch_Reset(bExecute := DF(bReset) AND NOT bIsActive,
                iChannel := iChannel1,
                pYOutput :=
   GetPointer(g_bPulse_TargetValueMatch_Channel1_Y19_Motor1_On),
                diTargetValue := diTargetValue1,
                bError => bErrorReset);
```

Pulse_TargetValue Match_Set

Target value match ON (pulse output)

Description If the elapsed value matches the target value **diTargetValue** of the pulse output channel specified by **iChannel**, the output relay specified by **pYOutput** immediately turns to TRUE.



See also:

Pulse output tool instructions in the online help

This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help: F166_PulseOutput_Set (see page 1026)

To validate the combination of channel and Y output, the compiler requires the following name pattern for global variables: %sPulse_TargetValueMatch_Channel%d_Y%d%s

Always use this pattern for global variables in target value match control.

- Channel%d must be a pulse output channel number enabled in the system registers
- Y%d must be an explicit output address supported by the PLC

FP-Σ, FP0, FP-e:	Y0–Y7
FP-Σ (V3.1 or higher), FP0R:	Y0–Y1F
FP-X:	Y0–Y29F

- %s is an optional descriptor at the beginning and the end of the pattern
- Y%d must be an explicit output address supported by the PLC

FP-Σ, FP0, FP-e:	Y0–Y7
FP-Σ (V3.1 or higher), FP0R:	Y0–Y1F
FP-X:	Y0–Y29F

- %s is an optional descriptor at the beginning and the end of the pattern

Optional	Fixed pattern	Optional
g_b	Pulse_TargetValueMatch_ChannelA_Y11F	_MotorOn

This global variable generates the code for channel **A** and output **Y11F**.

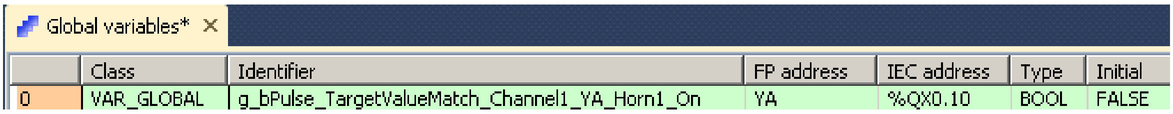
PLC types see see page 1329

Data types

Input variable	Data type	Function
bExecute	BOOL	A rising edge activates the function; evaluate the pulse output channel control flag using PulseInfo_IsTargetValueMatchActive (see page 1229)
iChannel	INT	FPΣ: 0, 2 FP-X R: 0, 1 FP-X 16K C14T: 0, 1, 2 FP-X 32K C30T, C60T: 0, 1, 2, 3 FP0R: 0, 1, 2, 3 FP0: 0, 1 FP-e: 0, 1
pYOutput	POINTER	Pointer result obtained by GetPointer from a global variable that supplies the channel number and output relay
diTargetValue	DINT	Specify a 32-bit data value for the target value within the following range: FP0, FP-e: -838808–+8388607 FPΣ, FP-X, FP0R: -2147483467–+2147483648
Output variable	**Data type**	**Function**
bError	BOOL	TRUE if the combination of **Channel%d** and **pYOuput.iOffset** does not match a valid combination of channel number and output relay as determined by the global variable

Example In this example the function has been programmed in ladder diagram (LD) and structured text (ST). (ST).The same POU header is used for all programming languages.

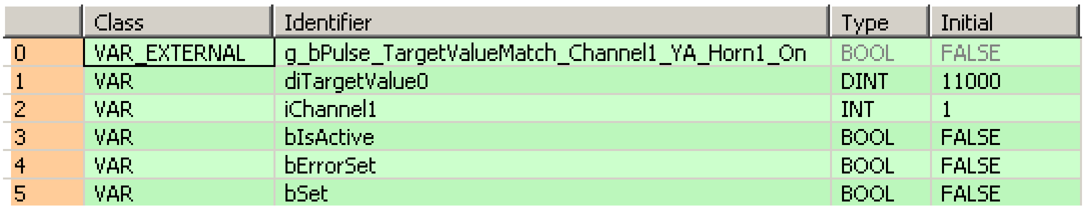
GVL In the global variable list you define variables that can be accessed by all POUs in the project.



Global variables*

	Class	Identifier	FP address	IEC address	Type	Initial
0	VAR_GLOBAL	g_bPulse_TargetValueMatch_Channel1_YA_Horn1_On	YA	%QX0.10	BOOL	FALSE

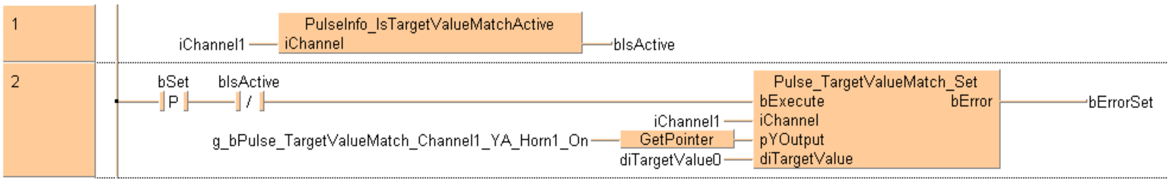
POU header All input and output variables used for programming this function have been declared in the POU header.



	Class	Identifier	Type	Initial
0	VAR_EXTERNAL	g_bPulse_TargetValueMatch_Channel1_YA_Horn1_On	BOOL	FALSE
1	VAR	diTargetValue0	DINT	11000
2	VAR	iChannel1	INT	1
3	VAR	bIsActive	BOOL	FALSE
4	VAR	bErrorSet	BOOL	FALSE
5	VAR	bSet	BOOL	FALSE

Body Use PulseInfo_IsTargetValueMatchActive (see page 1229) to evaluate the channel **iChannel1** is active. If a rising edge is detected at **bSet** and if **bIsActive** is **not** TRUE, the instruction is executed. The combination of channel number and output contact is validated in the global variable **g_bPulse_TargetValueMatch_Channel1_YA_Horn1_On**. When pulse output on channel 1 reaches the target value **diTargetValue0**, output **YA** is set to TRUE.

LD



```
ST bIsActive:=PulseInfo_IsTargetValueMatchActive(iChannel1);
   Pulse_TargetValueMatch_Set(bExecute := DF(bSet) AND NOT bIsActive,
                iChannel := iChannel1,
                pYOutput :=
   GetPointer(g_bPulse_TargetValueMatch_Channel1_YA_Horn1_On),
                diTargetValue := diTargetValue0,
                bError => bErrorSet);
```

Chapter 40

Appendix Programming Information

40.1 FP TOOL Library

The FP TOOL Library contains advanced address, information and copy functions available for all PLCs to make programming easier. Below please find a selection of these functions. For more detailed information and examples, see Online help.

Program can be adversely effected!
These functions can cause substantial problems by accessing incorrect memory areas if they are not used in the sense they were meant for. Especially other parts of the program can be adversely effected.

Name	Function
Addresses Instructions	
Adr_Of_Var	Address of a variable at the input/output of a FP function
AdrLast_Of_Var	Address of a variable at the input/output of a FP function
Adr_Of_VarOffs	Address of a variable with offset at the input/output of a FP function
Size Information Instructions	
Size_Of_Var	Yields the size of a variable in words (with Enable)
Elem_OfArray1D	Yields the number of elements in an array (with Enable)
Elem_OfArray2D	Yields the number of elements of the 1st and 2nd dimension of an array (with Enable)
Elem_OfArray3D	Yields the number of elements of the 1st, 2nd and 3rd dimension of an array (with Enable)
Pointer Instructions	
GetPointer	Provides pointer information
AreaOffs_ToVar	Copies the content of an address specified by memory area and address offset to a variable (with Enable)
Var_ToAreaOffs	Copies the value of a variable to an address specified by memory area and address offset to a variable (with Enable)
Is_AreaDT	Yields TRUE if the memory area of a variable is a DT area (with Enable)
Is_AreaFL	Yields TRUE if the memory area of a variable is a FL area (with Enable)
AdrDT_Of_Offs	DT address from the address offset for the input/output of a FP function
AdrFL_Of_Offs	FL address from the address offset for the input/output of a FP function
Additional Copy Instructions	
☞	**This functions are allowed to be compiled because of the down-compatibility to lower versions but cannot be selected in the "Instructions" dialog anymore.**
Any16_ToBool16	Replaced from version 5 onwards by the function INT_TO_BOOL16 or WORD_TO_BOOL16.
Bool16_ToAny16	Replaced from version 5 onwards by the function BOOL16_TO_INT or BOOL16_TO_WORD.
Any32_ToBool32	Replaced from version 5 onwards by the function DINT_TO_BOOL32 or DWORD_TO_BOOL32.
Bool32_ToAny32	Replaced from version 5 onwards by the function BOOL32_TO_DINT or BOOL32_TO_DWORD.
Any16_ToSpecDT	Replaced from version 5 onwards by the function INT_TO_SDT or WORD_TO_SDT.
SpecDT_ToAny16	Replaced from version 5 onwards by the function SDT_TO_INT or SDT_TO_WORD.

Name	Function
Any32_ToSpecDT	Replaced from version 5 onwards by the function DINT_TO_SDDT or DWORD_TO_SDDT.
SpecDT_ToAny32	Replaced from version 5 onwards by the function SDDT_TO_DINT or SDDT_TO_DWORD.
SFC Control Instructions	
Instructions that control all SFC programs simultaneously	
StartStopAllSfcs	Stops and restarts all Sequential Function Chart (SFC) programs
StartStopAllSfcsAndInitData	
A function that reveals the status of all SFCs	
AllSfcsStopped	Indicates whether all Sequential Function Chart (SFC) programs were stopped
Instructions that control a specific SFC	
StartStopSfc	Stops and restarts a specific Sequential Function Chart (SFC) program
StartStopSfcAndInitData	
ControlSfc	Controls a specific Sequential Function Chart (SFC) program
ControlSfcAndData	
ActivateStepsOfStoppedSfc	Continues a Sequential Function Chart (SFC) program that has been stopped
Instructions that reveal the statuses of a specific SFC	
SfcStopped	Indicates whether a specific Sequential Function Chart (SFC) program was stopped
SfcTransitionsInhibited	Indicates whether the transitions of a specific Sequential Function Chart (SFC) program are locked
SfcRunning	Indicates whether a certain Sequential Function Chart (SFC) program is running
SfcOutputsReset	Indicates whether the inputs of a specific Sequential Function Chart (SFC) program have been reset

40.2 Floating Point Instructions

The floating point F/P instructions are designed specifically for applications that require variables of the data type REAL. Most of these can be replaced by the more flexible IEC commands. By doing so you will reduce the number of commands with which you need to be familiar.

The following floating point instructions are described in detail in this manual because they are not easily duplicated with IEC instructions: F327_INT (see page 686), F328_DINT (see page 688), F333_FINT (see page 690), F334_FRINT (see page 692), F335_FSIGN (see page 694), F337_RAD (see page 696) and F338_DEG (see page 698).

For details and examples on the other floating point instructions, see Online help. For quick reference, please refer to the table below.

Name	Function	Equivalent IEC function used with EN/ENO
F309_FMV	Constant floating point data move	MOVE
F310_FADD	Floating point data add	ADD
F311_FSUB	Floating point data subtract	SUB
F312_FMUL	Floating point data multiply	MUL
F313_FDIV	Floating point data divide	DIV
F314_FSIN	Floating point Sine operation	SIN
F315_FCOS	Floating point Cosine operation	COS
F316_FTAN	Floating point Tangent operation	TAN
F317_ASIN	Floating point Arcsine operation	ASIN
F318_ACOS	Floating point Arccosine operation	ACOS
F319_ATAN	Floating point Arctangent operation	ATAN
F320_LN	Floating point data natural logarithm	LN
F321_EXP	Floating point data exponent	EXP
F322_LOG	Floating point data logarithm	LOG
F323_PWR	Floating point data power	EXPT
F324_FSQR	Floating point data square root	SQRT
F325_FLT	16-bit integer → Floating point data	INT_TO_REAL
F326_DFLT	32-bit integer → Floating point data	DINT_TO_REAL
F329_FIX	Floating point data → 16-bit integer Rounding the first decimal point down	TRUNC_TO_INT
F330_DFIX	Floating point data → 32-bit integer Rounding the first decimal point down	TRUNC_TO_DINT
F331_ROFF	Floating point data → 16-bit integer Rounding the first decimal point off	REAL_TO_INT
F332_DROFF	Floating point data → 32-bit integer Rounding the first decimal point off	REAL_TO_DINT
F336_FABS	Floating point data absolute	ABS

Name	Function	Equivalent IEC function used with EN/ENO
F345_FCMP	Floating point data compare	GE, GT, EQ, LE, LT, NE
F347_FLIMT	Floating point data upper and lower limit control	LIMIT

40.3 Index Registers

Like other registers, index registers are used to read and write 16-bit data. There are seven 16-bit registers (IX, IY, IZ to ID). Use index registers to indirectly specify a memory area number. Changing an address using an index register value is called “index modification”.

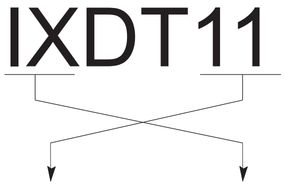
In FPWIN Pro the user has only access to IX, IY. Indexes IZ through ID are used by the system for array calculation, for fb indexing, or special implementations of certain FP instructions. (see note)

Possible index modifications include:

- Memory areas in addition to data registers (DT).
- An index register cannot modify another index register.
- In FPWIN Pro, an index register cannot modify a constant value.

Example

Modifying a memory area address.
Address = Base address + value in IX, IY, IZ through ID
Modifying DT11



Base address		IX value		Target address
11	+	0	=	DT11
11	+	10	=	DT21
11	+	-10	=	DT1

When the index register is used as an address modifier, be sure to check that the shifted address does not exceed its last address. If the shifted address exceeds its last address, an operation error occurs.

When a 32-bit constant is modified, the specified register number and the following register number are used in combination to handle the data as a 32-bit data. (When modifying a 32-bit constant, do not specify ID.)

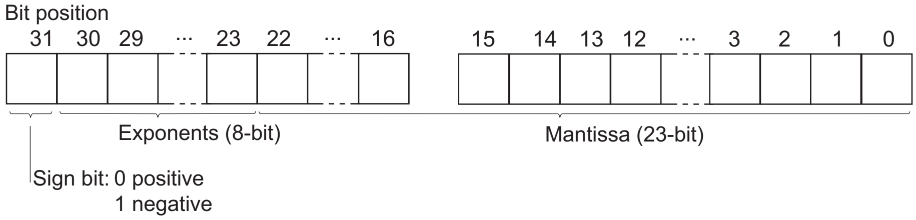
☞ **It is strongly suggested that you use arrays instead of using index registers to modify memory areas because a conflict could arise from the user and system using the same index register. For more detailed information on using index registers, see “Programmable Controller FP10SH Programming” Manual (ACG-M0081-1).**

40.4 Real Numbers

Instructions used with the FP10SH and the FP2 series allow the use of real numbers for calculation. Real number types available are floating point constants and BCD constants.

40.4.1 Floating Point Constant (f)

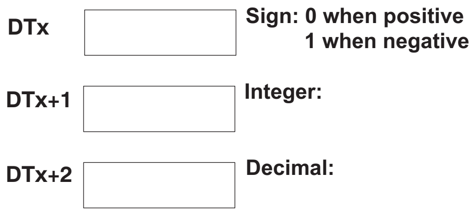
Floating point constants consist of two words processed by single precision floating point logic. There are up to seven effective digits. The mantissa is 23 bits and the exponent is 8 bits. (Based on IEEE754)



Numbers which can be used are: ±(1.175494 x 10^{-38} to 3.402823 x 10^{38}).

40.4.2 BCD Type Constant

BCD-type floating-point constants are processed as three words as shown below.



Example

Numbers which can be used are as follows: -9999.9999 to 9999.9999

The principal instructions which allow use of BCD constants are:

• **F300**	BSIN	BCD type	sine operation
• **F301**	BCOS	BCD type	cosine operation
• **F302**	BTAN	BCD type	tangent operation
• **F303**	BASIN	BCD type	arcsine operation
• **F304**	BACOS	BCD type	arccosine operation
• **F305**	BATAN	BCD type	arctangent operation

In FPWIN Pro use 16# to specify a BCD constant. This data type is implemented as ARRAY [0..2] OF WORD.

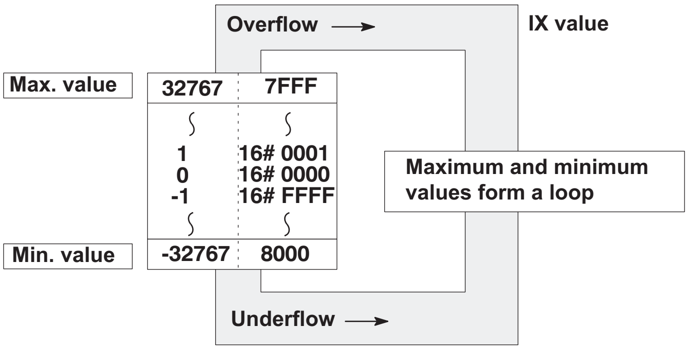
40.5 Overflow and Underflow

During execution of a processing instruction, it sometimes happens that the allowed value range is exceeded. Exceeding the maximum value is called “overflow”, and falling short of the minimum value is called “underflow.” If overflow or underflow occurs, the R9009 carry flag CY will turn ON.

40.5.1 Values When Overflow/Underflow Occurs

All of the maximum and minimum values handled by FP series programmable controller's form a loop as shown in the diagram.

Binary 16-bit processing



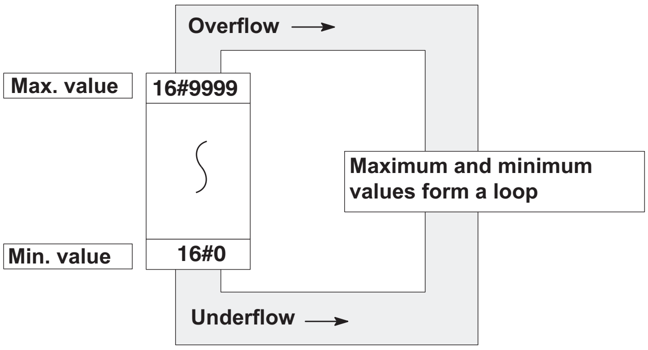
Example 1: 32767 + 1 (overflow)

The result of processing will be K-32768 and the carry flag will turn ON.

Example 2: -32768 - 1 (underflow)

The result of processing will be 32767 and the carry flag will turn ON.

BCD 4-digit processing



Example 1: 16#9999+ 16#1 (overflow)

The result of processing will be 16#0 and the carry flag will turn ON.

Example 2: 16#0 - 16#1 (underflow)

The result of processing will be 16#9999 and the carry flag will turn ON.

40.5.2 Decimal to binary/BCD/gray code table

Decimal number	Binary data	BCD data (Binary Coded Decimal)	Gray code
0	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000
1	0000 0000 0000 0001	0000 0000 0000 0001	0000 0000 0000 0001
2	0000 0000 0000 0010	0000 0000 0000 0010	0000 0000 0000 0011
3	0000 0000 0000 0011	0000 0000 0000 0011	0000 0000 0000 0010
4	0000 0000 0000 0100	0000 0000 0000 0100	0000 0000 0000 0110
5	0000 0000 0000 0101	0000 0000 0000 0101	0000 0000 0000 0111
6	0000 0000 0000 0110	0000 0000 0000 0110	0000 0000 0000 0101
7	0000 0000 0000 0111	0000 0000 0000 0111	0000 0000 0000 0100
8	0000 0000 0000 1000	0000 0000 0000 1000	0000 0000 0000 1100
9	0000 0000 0000 1001	0000 0000 0000 1001	0000 0000 0000 1101
10	0000 0000 0000 1010	0000 0000 0001 0000	0000 0000 0000 1111
11	0000 0000 0000 1011	0000 0000 0001 0001	0000 0000 0000 1110
12	0000 0000 0000 1100	0000 0000 0001 0010	0000 0000 0000 1010
13	0000 0000 0000 1101	0000 0000 0001 0011	0000 0000 0000 1011
14	0000 0000 0000 1110	0000 0000 0001 0100	0000 0000 0000 1001
15	0000 0000 0000 1111	0000 0000 0001 0101	0000 0000 0000 1000
16	0000 0000 0001 0000	0000 0000 0001 0110	0000 0000 0001 1000
17	0000 0000 0001 0001	0000 0000 0001 0111	0000 0000 0001 1001
18	0000 0000 0001 0010	0000 0000 0001 1000	0000 0000 0001 1011
19	0000 0000 0001 0011	0000 0000 0001 1001	0000 0000 0001 1010
20	0000 0000 0001 0100	0000 0000 0010 0000	0000 0000 0001 1110
21	0000 0000 0001 0101	0000 0000 0010 0001	0000 0000 0001 1111
22	0000 0000 0001 0110	0000 0000 0010 0010	0000 0000 0001 1101
23	0000 0000 0001 0111	0000 0000 0010 0011	0000 0000 0001 1100
24	0000 0000 0001 1000	0000 0000 0010 0100	0000 0000 0001 0100
25	0000 0000 0001 1001	0000 0000 0010 0101	0000 0000 0001 0101
26	0000 0000 0001 1010	0000 0000 0010 0110	0000 0000 0001 0111
27	0000 0000 0001 1011	0000 0000 0010 0111	0000 0000 0001 0110
28	0000 0000 0001 1100	0000 0000 0010 1000	0000 0000 0001 0010
29	0000 0000 0001 1101	0000 0000 0010 1001	0000 0000 0001 0011
30	0000 0000 0001 1110	0000 0000 0011 0000	0000 0000 0001 0001
31	0000 0000 0001 1111	0000 0000 0011 0001	0000 0000 0001 0000
32	0000 0000 0010 0000	0000 0000 0011 0010	0000 0000 0011 0000
...	...	...	...
63	0000 0000 0011 1111	0000 0000 0110 0011	0000 0000 0010 0000
64	0000 0000 0100 0000	0000 0000 0110 0100	0000 0000 0110 0000
...	...	...	...
255	0000 0000 1111 1111	0000 0010 0101 0101	0000 0000 1000 0000

40.6 Special data registers

To access special data registers and special internal relays, use the PLC-independent system variables.You can insert system variables directly into the POU body: Use the "Variables" dialog without entering a declaration in the POU header.

◆ **REFERENCE**

Please refer to the FPWIN Pro online help for detailed information on using system variables.

40.7 Relays and memory areas

40.7.1 Relays and memory areas for FP0

Relays [bits]

Type	Memory size	Available address area		Function
		F/P	IEC	
External input relays	208	X0–X12F	%IX0.0– %IX12.15	Turn on or off based on external input.
External output relays	208	Y0–Y12F	%QX0.0– %QX12.15	Turn on or off external outputs based on the operation result.
Internal relays 1)	1008	R0–R62F	%MX0.0.0– %MX0.62.15	Used internally by the PLC program to store bit information.
Timer relays 1) 2)	144	T0–T99/ C100–C143	%MX1.0– %MX1.99/ %MX2.100– %MX2.143	Turn on when the value set with a TM instruction for the timer with the same number has reached 0.
Counter relays 1) 2)	144	C100–C143/ T0–T99	%MX2.100– %MX2.143/ %MX1.0– %MX1.99	Turn on when the value set with a CT instruction for the counter with the same number has reached 0.
Special internal relays	64	R9000–R903F	%MX0.900.0– %MX0.903.15	Turn on or off based on specific conditions. Used internally as a flag.

Memory area [words]

Type		Memory size	Available address area		Function
			F/P	IEC	
External input relays		13	WX0–WX12	%IW0– %IW12	Code for specifying 16 external input points as one word (16 bits) of data.
External output relays		13	WY0–WY12	%QW0– %QW12	Code for specifying 16 external output points as one word (16 bits) of data.
Internal relays 1)		63	WR0–WR62	%MW0.0– %MW0.62	Code for specifying 16 internal relays as one word (16 bits) of data.
Data registers 1)	**C10/C14/C16**	1660	DT0–DT1659	%MW5.0– %MW5.1659	Data memory used in a program. Data is handled in 16-bit units (one word).
	C32/SL1	6144	DT0–DT6143	%MW5.0– %MW5.6143	
	T32C	16384	DT0–DT16383	%MW5.0– %MW5.16383	
Timer/counter set value area 1) 2)		144	SV0–SV143	%MW3.0– %MW3.143	Data memory for storing the set values of timers or counters. The values are stored by timer/counter number.
Timer/counter elapsed value area 1) 2)		144	EV0–EV143	%MW4.0– %MW4.143	Data memory for storing the elapsed values during operation of timers or counters. The values are stored by timer/counter number.
Special data registers		112	DT90000– DT90111	%MW5.90000– %MW5.90111	Data memory for storing settings and error codes.

Type	Memory size	Available address area		Function
		F/P	IEC	
Index registers	2	IX, IY	%MW6.0– %MW6.1	Data memory used to modify constants and memory area addresses.

Memory area [double words]

Type		Memory size	Available address area		Function
			F/P	IEC	
External input relays		6	DWX0–DWX11	%ID0– %ID11	Code for specifying 32 external input points as a double word (32 bits) of data.
External output relays		6	DWY0–DWY11	%QD0– %QD11	Code for specifying 32 external output points as a double word (32 bits) of data.
Internal relays 1)		31	DWR0–DWR61	%MD0.0– %MD0.61	Code for specifying 32 internal relay points as a double word (32 bits) of data.
Data registers 1)	**C10/C14/C16**	830	DDT0– DDT1658	%MD5.0– %MD5.1658	Data memory used in a program. Data is handled in 32-bit units (double word).
	C32/SL1	3072	DDT0– DDT6142	%MD5.0– %MD5.6142	
	T32C	8192	DDT0– DDT16382	%MD5.0– %MD5.16382	
Timer/counter set value area 1) 2)		72	DSV0–DSV142	%MD3.0– %MD3.142	Data memory for storing the set values of timers or counters. The values are stored by timer/counter number.
Timer/counter elapsed value area 1) 2)		72	DEV0–DEV142	%MD4.0– %MD4.142	Data memory for storing the elapsed values during operation of timers or counters. The values are stored by timer/counter number.
Special data registers		56	DDT90000– DDT90110	%MD5.90000– %MD5.90110	Data memory for storing settings and error codes.
Index registers		1	DI0	%MD6.0	Data memory used to modify constants and memory area addresses.

1) There are two memory types, the hold type that saves the conditions that exist just before turning the power off or changing from RUN to PROG mode, and the non−hold type that resets them. FP0 T32C: The hold and non−hold type memory areas can be changed by setting the system registers. All other CPU types: The hold and non-hold type memory area is fixed and allotted the numbers as shown below.

2) The number of points for timer and counter relays can be changed using system register 5. The numbers in the table are the default settings.

Hold and non-hold type memory areas

Memory area		C10/C14/C16	C32
Timer relays		Non-hold type: All points	
Counter relays	**Non-hold type**	From specified value to C139	From the specified value to C127
	Hold type	4 points (elapsed values) (C140–C143)	16 points (elapsed values) (C128–C143)

Internal relays	**Non-hold type**	976 points (R0–R60F) 61 words (WR0–WR60)	880 points (R0–R54F) 55 words (WR0–WR54)
	Hold type	32 points (R610–R62F) 2 words (WR61–WR62)	128 points (R550–R62F) 8 words (WR55–WR62)
Data registers	**Non-hold type**	1652 words (DT0–DT1651)	6112 words (DT0–DT6111)
	Hold type	8 words (DT1652–DT1659)	32 words (DT6112–DT6143)

40.7.2 Relays and memory areas for FP0R

Relays [bits]

Type	Memory size	Available address area		Function
		FP	IEC	
External input relays 1)	1760	X0–X109F	%IX0.0–%IX109.15	Turn on or off based on external input.
External output relays 1)	1760	Y0–Y109F	%QX0.0–%QX109.15	Turn on or off external outputs based on the operation result.
Internal relays 2)	4096	R0–R255F	%MX0.0.0–%MX0.255.15	Used internally by the PLC program to store bit information.
Link relays 2)	2048	L0–L127F	%MX7.0.0–%MX7.127.15	Shared by multiple PLCs connected using PLC link.
Timer relays 2) 3)	1024	T0–T1007/ C1008-C1023	%MX1.0–%MX1.1007/ %MX2.1008–%MX2.1023	Turn on when the value set with a TM instruction for the timer with the same number has reached 0.
Counter relays 2) 3)	1024	C1008–C1023/ T0–T1007	%MX2.1008–%MX2.1023/ %MX1.0–%MX1.1007	Turn on when the value set with a CT instruction for the counter with the same number has reached 0.
Special internal relays	224	R9000–R913F	%MX0.900.0–%MX0.913.15	Turn on or off based on specific conditions. Used internally as a flag.

Memory area [words]

Type		Memory size	Available address area		Function
			FP	IEC	
External input relays 1)		110	WX0–WX109	%IW0–%IW109	Code for specifying 16 external input points as one word (16 bits) of data.
External output relays 1)		110	WY0–WY109	%QW0–%QW109	Code for specifying 16 external output points as one word (16 bits) of data.
Internal relays 2)		256	WR0–WR255	%MW0.0–%MW0.255	Code for specifying 16 internal relays as one word (16 bits) of data.
Link relays		128	WL0–WL127	%MW7.0–%MW7.127	Code for specifying 16 link relays as one word (16 bits) of data.
Data registers 2)	**C10, C14, C16**	12315	DT0–DT12312	%MW5.0–%MW5.12312	Data memory used in a program. Data is handled in 16-bit units (one word).
	C32, T32, F32	32763	DT0–DT32762	%MW5.0–%MW5.32762	

Type	Memory size	Available address area		Function
		FP	IEC	
Link registers 2)	256	LD0–LD255	%MW8.0– %MW8.255	Data memory shared by multiple PLCs connected using PLC link. Data is handled in 16-bit units (one word).
Timer/counter set value area 2)	1024	SV0–SV1023	%MW3.0– %MW3.1023	Data memory for storing the set values of timers or counters. The values are stored by timer/counter number.
Timer/counter elapsed value area 2)	1024	EV0–EV1023	%MW4.0– %MW4.1023	Data memory for storing the elapsed values during operation of timers or counters. The values are stored by timer/counter number.
Special data registers	440	DT90000– DT90439	%MW5.90000– %MW5.90439	Data memory for storing settings and error codes.

Memory area [double words]

Type		Memory size	Available address area		Function
			FP	IEC	
External input relays 1)		55	DWX0–DWX108	%ID0– %ID108	Code for specifying 32 external input points as a double word (32 bits) of data.
External output relays 1)		55	DWY0–DWY108	%QD0– %QD108	Code for specifying 32 external output points as a double word (32 bits) of data.
Internal relays 2)		128	DWR0–DWR254	%MD0.0– %MD0.254	Code for specifying 32 internal relay points as a double word (32 bits) of data.
Link relays		64	DWL0–DWL126	%MD7.0– %MD7.126	Code for specifying 32 link relay points as a double word (32 bits) of data.
Data registers 2)	**C10, C14, C16**	6157	DDT0– DDT12311	%MD5.0– %MD5.12311	Data memory used in a program. Data is handled in 32-bit units (double word).
	C32, T32, F32	16382	DDT0– DDT32761	%MD5.0– %MD5.32761	
Link registers 2)		128	DLD0–DLD126	%MD8.0– %MD8.126	Data memory shared by multiple PLCs connected using PLC link. Data is handled in 32-bit units (double word).
Timer/counter set value area 2)		512	DSV0–DSV1022	%MD3.0– %MD3.1022	Data memory for storing the set values of timers or counters. The values are stored by timer/counter number.
Timer/counter elapsed value area 2)		512	DEV0–DEV1022	%MD4.0– %MD4.1022	Data memory for storing the elapsed values during operation of timers or counters. The values are stored by timer/counter number.
Special data registers		220	DDT90000– DDT90438	%MD5.90000– %MD5.90438	Data memory for storing settings and error codes.

1) The number of points noted above is the number reserved as the calculation memory. The actual number of points available for use is determined by the hardware configuration.

2) There are hold and non-hold type memory areas. When the power supply turns off or the mode is changed from RUN to PROG mode, hold type areas are stored and non-hold type areas are reset.

C10/C14/C16/C32:
The hold type and non-hold type areas are fixed. For information on the size of each area, refer to the performance specifications.

T32/F32:
The settings of the hold type areas and non-hold type areas can be changed using the system registers.

T32:
If the battery is empty and additional hold areas have been defined, the hold/non-hold operation becomes unstable. The data value will become indefinite. It is cleared to 0 the next time the power is turned on. See.

3) The number of points for timer and counter relays can be changed using system register 5. The numbers in the table are the default settings.

40.7.3 Relays and memory areas for FP-Sigma

Relays [bits]

Type	Memory size	Available address area		Function
		FP	IEC	
External input relays 1)	1184	X0–X73F	%IX0.0– %IX73.15	Turn on or off based on external input.
External output relays 1)	1184	Y0–Y73F	%QX0.0– %QX73.15	Turn on or off external outputs based on the operation result.
Internal relays 2)	4096	R0–R255F	%MX0.0– %MX0.255.15	Used internally by the PLC program to store bit information.
Link relays 2)	2048	L0–L127F	%MX7.0.0– %MX7.63.15	Shared by multiple PLCs connected using PLC link.
Timer relays 2) 3)	1024	T0–T1007/ C1008–C1023	%MX1.0– %MX1.1007/ %MX2.1008– %MX2.1023	Turn on when the value set with a TM instruction for the timer with the same number has reached 0.
Counter relays 2) 3)	1024	C1008–C1023/ T0–T1007	%MX2.1008– %MX2.1023/ %MX1.0– %MX1.1007	Turn on when the value set with a CT instruction for the counter with the same number has reached 0.
Special internal relays	176	R9000–R910F	%MX0.900.0– %MX0.910.15	Turn on or off based on specific conditions. Used internally as a flag.

Memory area [words]

Type	Memory size	Available address area		Function
		FP	IEC	
External input relays 1)	74	WX0–WX73	%IW0– %IW73	Code for specifying 16 external input points as one word (16 bits) of data.
External output relays 1)	74	WY0–WY73	%QW0– %QW73	Code for specifying 16 external output points as one word (16 bits) of data.
Internal relays 2)	256	WR0–WR255	%MW0.0– %MW0.255	Code for specifying 16 internal relays as one word (16 bits) of data.
Link relays	128	WL0–WL127	%MW7.0– %MW7.127	Code for specifying 16 link relays as one word (16 bits) of data.

Type	Memory size	Available address area		Function
		FP	IEC	
Data registers 2)	32763	DT0–DT32762	%MW5.0– %MW5.32762	Data memory used in a program. Data is handled in 16-bit units (one word).
Link registers 2)	256	LD0–LD255	%MW8.0– %MW8.255	Data memory shared by multiple PLCs connected using PLC link. Data is handled in 16-bit units (one word).
Timer/counter set value area 2)	1024	SV0–SV1023	%MW3.0– %MW3.1023	Data memory for storing the set values of timers or counters. The values are stored by timer/counter number.
Timer/counter elapsed value area 2)	1024	EV0–EV1023	%MW4.0– %MW4.1023	Data memory for storing the elapsed values during operation of timers or counters. The values are stored by timer/counter number.
Special data registers	260	DT90000– DT90259	%MW5.90000– %MW5.90259	Data memory for storing settings and error codes.

Memory area [double words]

Type	Memory size	Available address area		Function
		FP	IEC	
External input relays 1)	37	DWX0–DWX72	%ID0– %ID72	Code for specifying 32 external input points as a double word (32 bits) of data.
External output relays 1)	37	DWY0–DWY72	%QD0– %QD72	Code for specifying 32 external output points as a double word (32 bits) of data.
Internal relays 2)	128	DWR0–DWR254	%MD0.0– %MD0.254	Code for specifying 32 internal relay points as a double word (32 bits) of data.
Link relays	64	DWL0–DWL126	%MD7.0– %MD7.126	Code for specifying 32 link relay points as a double word (32 bits) of data.
Data registers 2)	16382	DDT0– DDT32763	%MD5.0– %MD5.32763	Data memory used in a program. Data is handled in 32-bit units (double word).
Link registers 2)	128	DLD0–DLD254	%MD8.0– %MD8.254	Data memory shared by multiple PLCs connected using PLC link. Data is handled in 32-bit units (double word).
Timer/counter set value area 2)	512	DSV0–DSV1022	%MD3.0– %MD3.1022	Data memory for storing the set values of timers or counters. The values are stored by timer/counter number.
Timer/counter elapsed value area 2)	512	DEV0–DEV1022	%MD4.0– %MD4.1022	Data memory for storing the elapsed values during operation of timers or counters. The values are stored by timer/counter number.
Special data registers	130	DDT90000– DDT90258	%MD5.90000– %MD5.90258	Data memory for storing settings and error codes.

1) The number of points noted above is the number reserved as the calculation memory. The actual number of points available for use is determined by the hardware configuration.

2) If no battery is used, only the fixed area is backed up.

Counter relays: 16 (C1008–C1023)
Internal relays: 128 (R900–R97F)
Data registers: DT32710–DT32764.

If the optional battery is used, the data in the hold and non-hold areas specified in the system registers will be backed up.

If the battery is empty or no battery is present and additional hold areas have been defined, the hold/non-hold operation becomes unstable. The data value will become indefinite. It is not cleared to 0 the next time the power is turned on. Do not forget to monitor the battery status or to reset the hold areas to the default values if no battery is used. See.

3) The number of points for timer and counter relays can be changed using system register 5. The numbers in the table are the default settings.

40.7.4 Relays and memory areas for FP-X

Relays [bits]

Type	Memory size	Available address area		Function
		F/P	IEC	
External input relays 1)	1760	X0–X109F	%IX0.0– %IX109.15	Turn on or off based on external input.
External output relays 1)	1760	Y0–Y109F	%QX0.0– %QX109.15	Turn on or off external outputs based on the operation result.
Internal relays 2)	4096	R0–R255F	%MX0.0.0– %MX0.255.15	Used internally by the PLC program to store bit information.
Link relays 2)	2048	L0–L127F	%MX7.0.0– %MX7.127.15	Shared by multiple PLCs connected using PLC link.
Timer relays 2) 3)	1024	T0–T1007/ C1008–C1023	%MX1.0– %MX1.1007/ %MX2.1008– %MX2.1023	Turn on when the value set with a TM instruction for the timer with the same number has reached 0.
Counter relays 2) 3)	1024	C1008–C1023/ T0–T1007	%MX2.1008– %MX2.1023/ %MX1.0– %MX1.1007	Turn on when the value set with a CT instruction for the counter with the same number has reached 0.
Special internal relays	192	R9000–R911F	%MX0.900.0– %MX0.911.15	Turn on or off based on specific conditions. Used internally as a flag.

Memory area [words]

Type		Memory size	Available address area		Function
			F/P	IEC	
External input relays 1)		110	WX0–WX109	%IW0– %IW109	Code for specifying 16 external input points as one word (16 bits) of data.
External output relays 1)		110	WY0–WY109	%QW0– %QW109	Code for specifying 16 external output points as one word (16 bits) of data.
Internal relays 2)		256	WR0–WR255	%MW0.0– %MW0.255	Code for specifying 16 internal relays as one word (16 bits) of data.
Link relays		128	WL0–WL127	%MW7.0– %MW7.127	Code for specifying 16 link relays as one word (16 bits) of data.
Data registers 2)	**C14**	12285	DT0–DT12284	%MW5.0– %MW5.12284	Data memory used in a program. Data is handled in 16-bit units

Type		Memory size	Available address area		Function
			F/P	IEC	
	C30, C60	32765	DT0–DT32764	%MW5.0–%MW5.32764	
Link registers 2)		256	LD0–LD255	%MW8.0–%MW8.255	Data memory shared by multiple PLCs connected using PLC link. Data is handled in 16-bit units (one word)..
Timer/counter set value area 2)		1024	SV0–SV1023	%MW3.0–%MW3.1023	Data memory for storing the set values of timers or counters. The values are stored by timer/counter number.
Timer/counter elapsed value area 2)		1024	EV0–EV1023	%MW4.0–%MW4.1023	Data memory for storing the elapsed values during operation of timers or counters. The values are stored by timer/counter number.
Special data registers		374	DT90000–DT90373	%MW5.90000–%MW5.90373	Data memory for storing settings and error codes.
Index registers		14	I0–ID	%MW6.0–%MW6.13	Data memory used to modify constants and memory area addresses.

Memory area [double words]

Type		Memory size	Available address area		Function
			F/P	IEC	
External input relays 1)		55	DWX0–DWX108	%ID0–%ID108	Code for specifying 32 external input points as a double word (32 bits) of data.
External output relays 1)		55	DWY0–DWY108	%QD0–%QD108	Code for specifying 32 external output points as a double word (32 bits) of data.
Internal relays 2)		128	DWR0–DWR254	%MD0.0–%MD0.254	Code for specifying 32 internal relay points as a double word (32 bits) of data.
Link relays		64	DWL0–DWL126	%MD7.0–%MD7.126	Code for specifying 32 link relay points as a double word (32 bits) of data.
Data registers 2)	C14	6142	DDT0–DDT12283	%MD5.0–%MD5.12283	Data memory used in a program. Data is handled in 32-bit units (double word).
	C30, C60	16382	DDT0–DDT32763	%MD5.0–%MD5.32763	
Link registers 2)		128	DLD0–DLD126	%MD8.0–%MD8.126	Data memory shared by multiple PLCs connected using PLC link. Data is handled in 32-bit units (double word).
Timer/counter set value area 2)		512	DSV0–DSV1022	%MD3.0–%MD3.1022	Data memory for storing the set values of timers or counters. The values are stored by timer/counter number.
Timer/counter elapsed value area 2)		512	DEV0–DEV1022	%MD4.0–%MD4.1022	Data memory for storing the elapsed values during operation of timers or counters. The values are stored by timer/counter number.

Type	Memory size	Available address area		Function
		F/P	IEC	
Special data registers	187	DDT90000– DDT90438	%MD5.90000– %MD5.90438	Data memory for storing settings and error codes.
Index registers	7	DI0–DIC	%MD6.0– %MD6.12	Data memory used to modify constants and memory area addresses.

[1] The number of points noted above is the number reserved as the calculation memory. The actual number of points available for use is determined by the hardware configuration.

[2] If no battery is used, only the fixed area is backed up.

If the optional battery is used, the data in the hold and non-hold areas specified in the system registers will be backed up.

If the battery is empty or no battery is present and additional hold areas have been defined, the hold/non-hold operation becomes unstable. The data value will become indefinite. It is not cleared to 0 the next time the power is turned on. Do not forget to monitor the battery status or to reset the hold areas to the default values if no battery is used. See.

[3] The number of points for timer and counter relays can be changed using system register 5. The numbers in the table are the default settings.

40.7.5 Relays and memory areas for FP-e

Relays [bits]

Type	Memory size	Available address area		Function
		F/P	IEC	
External input relays [1]	208	X0–X12F	%IX0.0– %IX12.15	Turn on or off based on external input.
External output relays [1]	208	Y0–Y12F	%QX0.0– %QX12.15	Turn on or off external outputs based on the operation result.
Internal relays [2]	1008	R0–R62F	%MX0.0.0– %MX0.62.15	Used internally by the PLC program to store bit information.
Timer relays [2] [3]	100	T0–T99/ C100–C143	%MX1.0– %MX1.99/ %MX2.100– %MX2.143	Turn on when the value set with a TM instruction for the timer with the same number has reached 0.
Counter relays [2] [3]	44	C100–C143/ T0–T99	%MX2.100– %MX2.143/ %MX1.0– %MX1.99	Turn on when the value set with a CT instruction for the counter with the same number has reached 0.
Special internal relays	64	R9000–R903F	%MX0.900.0– %MX0.903.15	Turn on or off based on specific conditions. Used internally as a flag.

Memory area [words]

Type	Memory size	Available address area		Function
		F/P	IEC	
External input relays [1]	13	WX0–WX12	%IW0– %IW12	Code for specifying 16 external input points as one word (16 bits) of data.

Type	Memory size	Available address area		Function
		F/P	IEC	
External output relays 1)	13	WY0–WY12	%QW0– %QW12	Code for specifying 16 external output points as one word (16 bits) of data.
Internal relays 2)	63	WR0–WR62	%MW0.0– %MW0.62	Code for specifying 16 internal relays as one word (16 bits) of data.
Data registers 2)	1660	DT0–DT1659	%MW5.0– %MW5.1659	Data memory used in a program. Data is handled in 16-bit units (one word).
Timer/counter set value area 2) 3)	144	SV0–SV143	%MW3.0– %MW3.143	Data memory for storing the set values of timers or counters. The values are stored by timer/counter number.
Timer/counter elapsed value area 2) 3)	144	EV0–EV143	%MW4.0– %MW4.143	Data memory for storing the elapsed values during operation of timers or counters. The values are stored by timer/counter number.
Special data registers	112	DT90000– DT90111	%MW5.90000– %MW5.90111	Data memory for storing settings and error codes.
Index registers	2	IX, IY	%MW6.0– %MW6.1	Data memory used to modify constants and memory area addresses.

Memory area [double words]

Type	Memory size	Available address area		Function
		F/P	IEC	
External input relays 1)	6	DWX0–DWX11	%ID0– %ID11	Code for specifying 32 external input points as a double word (32 bits) of data.
External output relays 1)	6	DWY0–DWY11	%QD0– %QD11	Code for specifying 32 external output points as a double word (32 bits) of data.
Internal relays 2)	31	DWR0–DWR61	%MD0.0– %MD0.61	Code for specifying 32 internal relay points as a double word (32 bits) of data.
Data registers 2)	830	DDT0– DDT1658	%MD5.0– %MD5.1658	Data is handled in 32-bit units (double word).
Timer/counter set value area 2) 3)	72	DSV0–DSV142	%MD3.0– %MD3.142	Data memory for storing the set values of timers or counters. The values are stored by timer/counter number.
Timer/counter elapsed value area 2) 3)	72	DEV0–DEV142	%MD4.0– %MD4.142	Data memory for storing the elapsed values during operation of timers or counters. The values are stored by timer/counter number.
Special data registers	56	DDT90000– DDT90110	%MD5.90000– %MD5.90110	Data memory for storing settings and error codes.

Type	Memory size	Available address area		Function
		F/P	IEC	
Index registers	1	DI0	%MD6.0	Data memory used to modify constants and memory area addresses.

1) The number of points noted above is the number reserved as the calculation memory. The actual number of points available for use is determined by the hardware configuration.

2) There are two memory types, the hold type that saves the conditions that exist just before turning the power off or changing from RUN to PROG mode, and the non−hold type that resets them. Standard type CPU: The hold and non-hold type memory area is fixed and allotted the numbers as shown below. CPU types with clock/calendar function: The hold and non−hold type memory areas can be changed by setting the system registers.

3) The number of points for timer and counter relays can be changed using system register 5. The numbers in the table are the default settings.

Hold and non-hold type memory areas

Memory area		Standard type CPU	CPU types with clock/calendar function
Timer relays		Non-hold type	
Counter relays	**Non-hold type**	From specified value to C139	
	Hold type	Non-hold type	4 points (elapsed values) (C140–C143) 1)
Internal relays	**Non-hold type**	976 points (R0–R60F) 61 words (WR0–WR60)	
	Hold type	32 points (R610–R62F) 2 words (WR61–WR62)	
Data registers	**Non-hold type**	1652 words (DT0–DT1651)	
	Hold type	8 words (DT1652–DT1659)	

1) A battery must be installed.

40.7.6 Relays and memory areas for FP2

Relays [bits]

Type	Memory size	Available address area		Function
		F/P	IEC	
External input relays	2048	X0–X127F	%IX0.0– %IX127.15	Turn on or off based on external input.
External output relays	2048	Y0–Y127F	%QX0.0– %QX127.15	Turn on or off external outputs based on the operation result.
Internal relays 1)	4048	R0–R252F	%MX0.0– %MX0.252.15	Used internally by the PLC program to store bit information.
Link relays 1)	2048	L0–L127F	%MX7.0.0– %MX7.127.15	Shared by multiple PLCs connected using PLC link.
Timer relays 1) 2)	1024	T0–T999/ C1000–C1023	%MX1.0– %MX1.999/ %MX2.1000– %MX2.1023	Turn on when the value set with a TM instruction for the timer with the same number has reached 0.
Counter relays 1) 2)	1024	C1000–C1023/ T0–T999	%MX2.1000– %MX2.1023/ %MX1.0– %MX1.999	Turn on when the value set with a CT instruction for the counter with the same number has reached 0.

Type	Memory size	Available address area		Function
		F/P	IEC	
External input relays	2048	X0–X127F	%IX0.0– %IX127.15	Turn on or off based on external input.
Pulse relays	1024	P0–P63F	%MX11.0.0– %MX11.63.15	Turn on for one scan only. Used internally by the PLC program.
Special internal relays	176	R9000–R910F	%MX0.900.0– %MX0.910.15	Turn on or off based on specific conditions. Used internally as a flag.

Memory area [words]

Type		Memory size	Available address area		Function
			F/P	IEC	
External input relays		128	WX0–WX127	%IW0– %IW127	Code for specifying 16 external input points as one word (16 bits) of data.
External output relays		128	WY0–WY127	%QW0– %QW127	Code for specifying 16 external output points as one word (16 bits) of data.
Internal relays		253	WR0–WR252	%MW0.0– %MW0.252	Code for specifying 16 internal relays as one word (16 bits) of data.
Link relays		128	WL0–WL127	%MW7.0– %MW7.127	Code for specifying 16 link relays as one word (16 bits) of data.
Data registers 1)		6000	DT0–DT5999	%MW5.0– %MW5.5999	Data memory used in a program. Data is handled in 16-bit units (one word).
Link registers 1)		256	LD0–LD255	%MW8.0– %MW8.255	Data memory shared by multiple PLCs connected using PLC link. Data is handled in 16-bit units (one word)..
Timer/counter set value area 1)		1024	SV0–SV1023	%MW3.0– %MW3.1023	Data memory for storing the set values of timers or counters. The values are stored by timer/counter number.
Timer/counter elapsed value area 1)		1024	EV0–EV1023	%MW4.0– %MW4.1023	Data memory for storing the elapsed values during operation of timers or counters. The values are stored by timer/counter number.
File registers 1) 3)	**12k type**	14333	FL0–FL14332	%MW9.0– %MW9.14332	Data memory used in a program. Data is handled in 16-bit units (one word).
	32k type (expanded)	30717	FL0–FL30716	%MW9.0– %MW9.30716	
Special data registers		256	DT90000– DT90255	%MW5.90000– %MW5.90255	Data memory for storing settings and error codes.
Index registers		14	I0–ID	%MW6.0– %MW6.13	Data memory used to modify constants and memory area addresses.

Memory area [double words]

Type	Memory size	Available address area		Function
		F/P	IEC	
External input relays 1)	64	DWX0–DWX72	%ID0– %ID72	Code for specifying 32 external input points as a double word (32 bits) of data.
External output relays 1)	64	DWY0–DWY72	%QD0– %QD72	Code for specifying 32 external output points as a double word (32 bits) of data.

Type		Memory size	Available address area		Function
			F/P	IEC	
Internal relays 2)		126	DWR0–DWR254	%MD0.0– %MD0.254	Code for specifying 32 internal relay points as a double word (32 bits) of data.
Link relays		64	DWL0–DWL126	%MD7.0– %MD7.126	Code for specifying 32 link relay points as a double word (32 bits) of data.
Data registers 2)		3000	DDT0– DDT32763	%MD5.0– %MD5.32763	Data memory used in a program. Data is handled in 32-bit units (double word).
Link registers 2)		128	DLD0–DLD254	%MD8.0– %MD8.254	Data memory shared by multiple PLCs connected using PLC link. Data is handled in 32-bit units (double word).
Timer/counter set value area 2)		512	DSV0–DSV1022	%MD3.0– %MD3.1022	Data memory for storing the set values of timers or counters. The values are stored by timer/counter number.
Timer/counter elapsed value area 2)		512	DEV0–DEV1022	%MD4.0– %MD4.1022	Data memory for storing the elapsed values during operation of timers or counters. The values are stored by timer/counter number.
File registers 1) 3)	**12k type**	7166	DFL0–DFL14331	%MD9.0– %MD9.14331	Data memory used in a program. Data is handled in 16-bit units (one word).
	32k type (expanded)	15358	DFL0–DFL30715	%MD9.0– %MD9.30715	
Special data registers		128	DDT90000– DDT90254	%MD5.90000– %MD5.90254	Data memory for storing settings and error codes.
Index registers		7	DI0–DID	%MD6.0– %MD6.13	Data memory used to modify constants and memory area addresses.

1) There are two memory types, the hold type that saves the conditions that exist just before turning the power off or changing from RUN to PROG mode, and the non−hold type that resets them. The hold and non−hold type memory areas can be changed by setting the system registers.

2) The number of points for timer and counter relays can be changed using system register 5. The numbers in the table are the default settings.

3) The size of the file registers varies depending on the settings of system registers 0, 1, and 2.

40.7.7 Relays and memory areas for FP2SH

Relays [bits]

Type	Memory size	Available address area		Function
		F/P	IEC	
External input relays	8192	X0–X511F	%IX0.0– %IX511.15	Turn on or off based on external input.
External output relays	8192	Y0–Y511F	%QX0.0– %QX511.15	Turn on or off external outputs based on the operation result.
Internal relays 1)	14192	R0–R886F	%MX0.0.0– %MX0.886.15	Used internally by the PLC program to store bit information.
Link relays 1)	10240	L0–L639F	%MX7.0.0– %MX7.639.15	Shared by multiple PLCs connected using PLC link.
Timer relays 1) 2)	3072	T0–T2999/ C3000–C3071	%MX1.0– %MX1.2999/ %MX2.3000– %MX2.3071	Turn on when the value set with a TM instruction for the timer with the same number has reached 0.
Counter relays 1) 2)	3072	C3000–C3071/ T0–T2999	%MX2.3000– %MX2.3071/ %MX1.0– %MX1.2999	Turn on when the value set with a CT instruction for the counter with the same number has reached 0.
Pulse relays	2048	P0–P127F	%MX11.0.0– %MX11.127.15	Turn on for one scan only. Used internally by the PLC program.
Error alarm relays	2048	E0–E127F	%MX10.0.0– %MX10.127.15	Turns on in the event of error. The error history is stored in dedicated data registers.
Special internal relays	176	R9000–R910F	%MX0.900.0– %MX0.910.15	Turn on or off based on specific conditions. Used internally as a flag.

Memory area [words]

Type	Memory size	Available address area		Function
		F/P	IEC	
External input relays	512	WX0–WX127	%IW0– %IW127	Code for specifying 16 external input points as one word (16 bits) of data.
External output relays	512	WY0–WY127	%QW0– %QW127	Code for specifying 16 external output points as one word (16 bits) of data.
Internal relays	887	WR0–WR252	%MW0.0– %MW0.252	Code for specifying 16 internal relays as one word (16 bits) of data.
Link relays	640	WL0–WL127	%MW7.0– %MW7.127	Code for specifying 16 link relays as one word (16 bits) of data.
Data registers 1)	10240	DT0–DT5999	%MW5.0– %MW5.5999	Data memory used in a program. Data is handled in 16-bit units (one word).
Link registers 1)	8448	LD0–LD255	%MW8.0– %MW8.255	Data memory shared by multiple PLCs connected using PLC link. Data is handled in 16-bit units (one word)..
Timer/counter set value area 1)	3072	SV0–SV1023	%MW3.0– %MW3.1023	Data memory for storing the set values of timers or counters. The values are stored by timer/counter number.

Type	Memory size	Available address area		Function
		F/P	IEC	
Timer/counter elapsed value area 1)	3072	EV0–EV1023	%MW4.0– %MW4.1023	Data memory for storing the elapsed values during operation of timers or counters. The values are stored by timer/counter number.
File registers 1)	98295 (32765 × 3 banks)	FL0–FL32764	%MW9.0– %MW9.32764	Data memory used in a program. Data is handled in 16-bit units (one word).
Special data registers	512	DT90000– DT90511	%MW5.90000– %MW5.90511	Data memory for storing settings and error codes.
Index registers	14	I0–ID	%MW6.0– %MW6.13	Data memory used to modify constants and memory area addresses.

Memory area [double words]

Type	Memory size	Available address area		Function
		F/P	IEC	
External input relays 1)	256	DWX0–DWX510	%ID0– %ID510	Code for specifying 32 external input points as a double word (32 bits) of data.
External output relays 1)	256	DWY0–DWY510	%QD0– %QD510	Code for specifying 32 external output points as a double word (32 bits) of data.
Internal relays 2)	443	DWR0–DWR885	%MD0.0– %MD0.885	Code for specifying 32 internal relay points as a double word (32 bits) of data.
Link relays	320	DWL0–DWL638	%MD7.0– %MD7.638	Code for specifying 32 link relay points as a double word (32 bits) of data.
Data registers 2)	5120	DDT0– DDT10238	%MD5.0– %MD5.10238	Data memory used in a program. Data is handled in 32-bit units (double word).
Link registers 2)	4224	DLD0–DLD8446	%MD8.0– %MD8.8446	Data memory shared by multiple PLCs connected using PLC link. Data is handled in 32-bit units (double word).
Timer/counter set value area 2)	1513	DSV0–DSV3070	%MD3.0– %MD3.3070	Data memory for storing the set values of timers or counters. The values are stored by timer/counter number.
Timer/counter elapsed value area 2)	1513	DEV0–DEV3070	%MD4.0– %MD4.3070	Data memory for storing the elapsed values during operation of timers or counters. The values are stored by timer/counter number.
File registers	16382	DFL0–DFL32763	%MD9.0– %MD9.32763	Data memory used in a program. Data is handled in 16-bit units (one word).
Special data registers	265	DDT90000– DDT90510	%MD5.90000– %MD5.90510	Data memory for storing settings and error codes.

Type	Memory size	Available address area		Function
		F/P	IEC	
Index registers	7	DI0–DID	%MD6.0– %MD6.13	Data memory used to modify constants and memory area addresses.

[1] There are two memory types, the hold type that saves the conditions that exist just before turning the power off or changing from RUN to PROG mode, and the non−hold type that resets them. The hold and non−hold type memory areas can be changed by setting the system registers.

[2] The number of points for timer and counter relays can be changed using system register 5. The numbers in the table are the default settings.

40.7.8 Relays and memory areas for FP10SH

Relays [bits]

Type	Memory size	Available address area		Function
		F/P	IEC	
External input relays	8192	X0–X511F	%IX0.0– %IX511.15	Turn on or off based on external input.
External output relays	8192	Y0–Y511F	%QX0.0– %QX511.15	Turn on or off external outputs based on the operation result.
Internal relays 1)	14192	R0–R886F	%MX0.0.0– %MX0.886.15	Used internally by the PLC program to store bit information.
Link relays 1)	10240	L0–L639F	%MX7.0.0– %MX7.639.15	Shared by multiple PLCs connected using PLC link.
Timer relays 1) 2)	3072	T0–T2999/ C3000–C3071	%MX1.0– %MX1.2999/ %MX2.3000– %MX2.3071	Turn on when the value set with a TM instruction for the timer with the same number has reached 0.
Counter relays 1) 2)	3072	C3000–C3071/ T0–T2999	%MX2.3000– %MX2.3071/ %MX1.0– %MX1.2999	Turn on when the value set with a CT instruction for the counter with the same number has reached 0.
Pulse relays	2048	P0–P127F	%MX11.0.0– %MX11.127.15	Turn on for one scan only. Used internally by the PLC program.
Error alarm relays	2048	E0–E127F	%MX10.0.0– %MX10.127.15	Turns on in the event of error. The error history is stored in dedicated data registers.
Special internal relays	176	R9000–R910F	%MX0.900.0– %MX0.910.15	Turn on or off based on specific conditions. Used internally as a flag.

Memory area [words]

Type	Memory size	Available address area		Function
		F/P	IEC	
External input relays	512	WX0–WX127	%IW0– %IW127	Code for specifying 16 external input points as one word (16 bits) of data.
External output relays	512	WY0–WY127	%QW0– %QW127	Code for specifying 16 external output points as one word (16 bits) of data.

Type	Memory size	Available address area		Function
		F/P	IEC	
Internal relays	887	WR0–WR252	%MW0.0– %MW0.252	Code for specifying 16 internal relays as one word (16 bits) of data.
Link relays	640	WL0–WL127	%MW7.0– %MW7.127	Code for specifying 16 link relays as one word (16 bits) of data.
Data registers 1)	10240	DT0–DT5999	%MW5.0– %MW5.5999	Data memory used in a program. Data is handled in 16-bit units (one word).
Link registers 1)	8448	LD0–LD255	%MW8.0– %MW8.255	Data memory shared by multiple PLCs connected using PLC link. Data is handled in 16-bit units (one word)..
Timer/counter set value area 1)	3072	SV0–SV1023	%MW3.0– %MW3.1023	Data memory for storing the set values of timers or counters. The values are stored by timer/counter number.
Timer/counter elapsed value area 1)	3072	EV0–EV1023	%MW4.0– %MW4.1023	Data memory for storing the elapsed values during operation of timers or counters. The values are stored by timer/counter number.
File registers 1)	32765	FL0–FL32764	%MW9.0– %MW9.32764	Data is handled in 16-bit units (one word). Data memory used in a program.
Special data registers	512	DT90000– DT90511	%MW5.90000– %MW5.90511	Data memory for storing settings and error codes.
Index registers	14	I0–ID	%MW6.0– %MW6.13	Data memory used to modify constants and memory area addresses.

Memory area [double words]

Type	Memory size	Available address area		Function
		F/P	IEC	
External input relays 1)	256	DWX0–DWX510	%ID0– %ID510	Code for specifying 32 external input points as a double word (32 bits) of data.
External output relays 1)	256	DWY0–DWY510	%QD0– %QD510	Code for specifying 32 external output points as a double word (32 bits) of data.
Internal relays 2)	443	DWR0–DWR885	%MD0.0– %MD0.885	Code for specifying 32 internal relay points as a double word (32 bits) of data.
Link relays	320	DWL0–DWL638	%MD7.0– %MD7.638	Code for specifying 32 link relay points as a double word (32 bits) of data.
Data registers 2)	5120	DDT0– DDT10238	%MD5.0– %MD5.10238	Data memory used in a program. Data is handled in 32-bit units (double word).
Link registers 2)	4224	DLD0–DLD8446	%MD8.0– %MD8.8446	Data memory shared by multiple PLCs connected using PLC link. Data is handled in 32-bit units (double word).
Timer/counter set value area 2)	1513	DSV0–DSV3070	%MD3.0– %MD3.3070	Data memory for storing the set values of timers or counters. The values are stored by timer/counter number.

Type	Memory size	Available address area		Function
		F/P	IEC	
Timer/counter elapsed value area [2]	1513	DEV0–DEV3070	%MD4.0– %MD4.3070	Data memory for storing the elapsed values during operation of timers or counters. The values are stored by timer/counter number.
File registers	16382	DFL0–DFL32763	%MD9.0– %MD9.32763	Data memory used in a program. Data is handled in 16-bit units (one word).
Special data registers	265	DDT90000– DDT90510	%MD5.90000– %MD5.90510	Data memory for storing settings and error codes.
Index registers	7	DI0–DID	%MD6.0– %MD6.13	Data memory used to modify constants and memory area addresses.

[1] There are two memory types, the hold type that saves the conditions that exist just before turning the power off or changing from RUN to PROG mode, and the non−hold type that resets them. The hold and non−hold type memory areas can be changed by setting the system registers.

[2] The number of points for timer and counter relays can be changed using system register 5. The numbers in the table are the default settings.

40.8 System registers

System registers are memory areas reserved for setting hold and non-hold areas for timers, counters, flags and data registers.

In the system registers you can also define parameters for PLC interfaces as to how they should react when errors occur.

◆ **NOTE**

The size of the memory depends on the PLC type used. The sum of all memory sizes for system registers, user program and machine program may not be larger than the entire PLC memory.

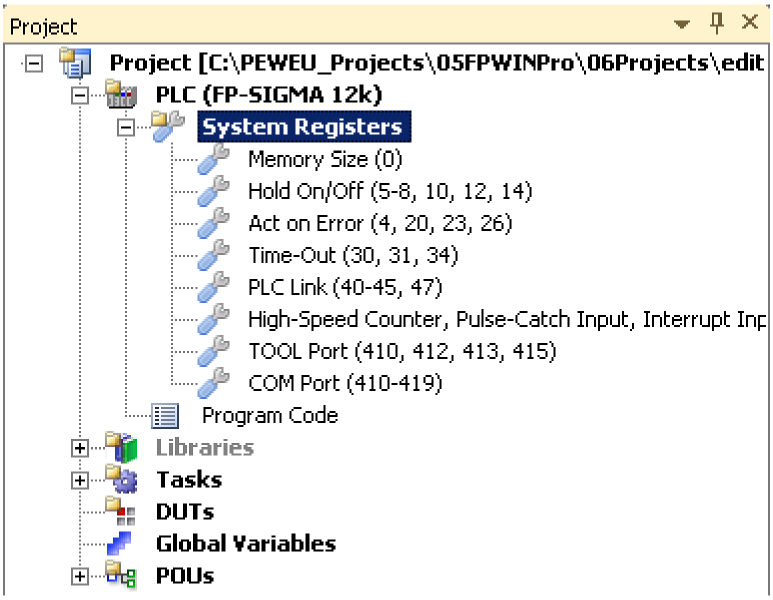
The 2 highest data registers (4 in PLCs with a second task) are at the user's disposal, since they are always in the hold area and used by the compiler.

◆ **Procedure**

1. **Double-click "PLC"**

2. **Double-click "System Registers"**

   A list with all system registers will be displayed. The number indicated in parentheses is identical to the system register number. In "Memory Size (0-3)", you define the memory sizes for machine programs, for example. You will find a list with all system registers and the memory size of your PLC in your hardware description.



Project
Project [C:\PEWEU_Projects\05FPWINPro\06Projects\edit
PLC (FP-SIGMA 12k)
System Registers
Memory Size (0)
Hold On/Off (5-8, 10, 12, 14)
Act on Error (4, 20, 23, 26)
Time-Out (30, 31, 34)
PLC Link (40-45, 47)
High-Speed Counter, Pulse-Catch Input, Interrupt Inp
TOOL Port (410, 412, 413, 415)
COM Port (410-419)
Program Code
Libraries
Tasks
DUTs
Global Variables
POUs

3. **Double-click desired set of system registers**

4. **Enter your settings**

40.8.1 Types of system registers

System registers are used to set values (parameters) which determine operation ranges and functions used. Set values based on the use and specifications of your program. There is no need to set system registers for functions which will not be used.

Not all system registers are available for all PLC types. Please see the system register tables for a list of system registers for each PLC type.

Memory size

Set the size of the memory area for the user program.

Hold on/off

Use these system registers to specify the hold area start addresses for relays and registers. Hold areas are not cleared to 0 when the PLC is switched to PROG mode or when the power is turned off.

The memory area for timer relays and counter relays is partitioned using system register no. 5. Specify the start address for the counter relays.

Act on Error

Set the operation mode to be chosen after errors such as an operation error, a battery error, or an I/O verification error.

Time-Out

Set the waiting time before an error is output. You can also specify a constant scan time.

MEWNET-F

Set the PLC start mode and timeout when MEWNET-F slave stations are connected.

PLC Link

These settings are for using link relays and link registers in MEWNET-W0 PLC link communication. Note that PLC Link is not the default setting.

PROFIBUS/MEWNET-H

Set the data size to be processed during one scan.

High-Speed Counter, Pulse-Catch Input, Interrupt Input

When using the high-speed counter function, pulse catch function or interrupt function, set the operation mode and the input number to be used for the function.

Time Constants

Set a time constant for the CPU inputs. These time constants can be useful to negate the effects of noise or bouncing, e.g. for a switching device.

TOOL Port, COM Port

Set these registers when the TOOL port and COM ports 1 and 2 ports are to be used for MEWTOCOL-COM Master/Slave connections, program controlled communication, PLC link, and modem communication. Note that the default setting is MEWTOCOL-COM Master/Slave.

40.8.2 System registers for FP-X

Memory size

No.	Name	Default	Values
0	Sequence program area size	12/16/32 kwords[1]	Fixed

[1] Depending on PLC type (12k, 16k, or 32k type)

Hold on/off

No.	Name	Default	Values
5[1)]	Counter start address	1008	0–1024
6[1)]	Timer/Counter hold area start address	1008	0–1024
7[1)]	Internal relay hold area start address (in word units)	248	0–256
8[1)]	Data register hold area start address	12230/ 32710[2)]	0–12283/ 0–32763[2)]
10	Link relay hold area start address for PLC Link 0 (in word units)	64	0–64
11	Link relay hold area start address for PLC Link 1 (in word units)	128	64–128
12	Link register hold area start address for PLC Link 0	128	0–128
13	Link register hold area start address for PLC Link 1	256	128–256
14[1)]	Step ladder hold/non-hold	Non-hold	Hold/Non-hold

[1)]
- These settings are effective if the optional backup battery is installed
- If no backup battery is used, do not change the default settings. Otherwise proper functioning of hold/non-hold values cannot be guaranteed.

[2)] Depending on PLC type (16k/32k type)

Act on Error

No.	Name	Default	Values
4	Battery error indication	Disable	Disable: When a battery error occurs, a self-diagnostic error is not issued and the ERROR LED will not flash. Enable: When a battery error occurs, a self-diagnostic error is issued and the ERROR will LED flash.
4	DF-, P-function leading/falling edge detection	Holds result	Holds result/disregards result
20	Duplicate output	Enable	Fixed
23	I/O verification error	Stop	Stop/Continue
26	Operation error	Stop	Stop/Continue

Time-Out

No.	Name	Default	Values
31	Multi-frame communication time	6500.0ms	10.0–81900.0ms
32	Timeout value for the communication functions based on F145, F146, F152, F153	10000.0ms	10.0–81900.0ms
34	Constant scan time	0.0ms	0.0–350.0ms 0.0: Normal scan (non-constant)
36	Expansion unit recognition time	0.0s	0.0–10.0s

PLC Link

No.	Name	Default	Values
46	PLC Link 0 and 1 allocation setting	Normal	Normal/Reverse
47	PLC link 0 - Highest station number in network	16	1–16
40	PLC link 0 - Link relays - Send/receive area - Number of words shared by all linked PLCs	0	0–64 words

No.	Name	Default	Values
42	PLC link 0 - Link relays - Send area - Start sending from this word address	0	0–63
43	PLC link 0 - Link relays - Send area - Number of words to send	0	0–64 words
41	PLC link 0 - Link registers - Send/receive area - Number of words shared by all linked PLCs	0	0–128 words
44	PLC link 0 - Link registers - Send area - Start sending from this word address	0	0–127
45	PLC link 0 - Link registers - Send area - Number of words to send	0	0–127 words
57	PLC link 1 - Highest station number in network	16	1–16
50	PLC link 1 - Link relays - Send/receive area - Number of words shared by all linked PLCs	0	0–64 words
52	PLC link 1 - Link relays - Send area - Start sending from this word address	64	64–127
53	PLC link 1 - Link relays - Send area - Number of words to send	0	0–64 words
51	PLC link 1 - Link registers - Send/receive area - Number of words shared by all linked PLCs	0	0–128 words
54	PLC link 1 - Link registers - Send area - Start sending from this word address	128	128–255
55	PLC link 1 - Link registers - Send area - Number of words to send	0	0–127 words

High-Speed Counter, Pulse-Catch Input, Interrupt Input (Transistor types)

No.	Name	Default	Values
400/ 401	High-speed counter: Channel 0	Unused	▪ Incremental input (X0) ▪ Incremental input (X0), Reset input (X6) ▪ Decremental input (X0) ▪ Decremental input (X0), Reset input (X6) ▪ Two-phase input (X0, X1) ▪ Two-phase input (X0, X1), Reset input (X6) ▪ Incremental input (X0), Decremental input (X1) ▪ Incremental input (X0), Decremental input (X1), Reset input (X6) ▪ Counter input (X0), Incremental/decremental control input (X1) ▪ Counter input (X0), Incremental/decremental control input (X1), Reset input (X6)
400	High-speed counter: Channel 1	Unused	▪ Incremental input (X1) ▪ Decremental input (X1)
400/ 401	High-speed counter: Channel 2	Unused	▪ Incremental input (X2) ▪ Incremental input (X2), Reset input (X7) ▪ Decremental input (X2) ▪ Decremental input (X2), Reset input (X7) ▪ Two-phase input (X2, X3) ▪ Two-phase input (X2, X3), Reset input (X7) ▪ Incremental input (X2), Decremental input (X3) ▪ Incremental input (X2), Decremental input (X3), Reset input (X7) ▪ Counter input (X2), Incremental/decremental control input (X3) ▪ Counter input (X2), Incremental/decremental control input (X3), Reset input (X7)
400	High-speed counter: Channel 3	Unused	▪ Incremental input (X3) ▪ Decremental input (X3)

No.	Name	Default	Values
401	High-speed counter: Channel 4	Unused	▪ Incremental input (X4) ▪ Decremental input (X4) ▪ Two-phase input (X4, X5) ▪ Incremental input (X4), Decremental input (X5) ▪ Counter input (X4), Incremental/decremental control input (X5)
401	High-speed counter: Channel 5	Unused	▪ Incremental input (X5) ▪ Decremental input (X5)
401	High-speed counter: Channel 6	Unused	▪ Incremental input (X6) ▪ Decremental input (X6) ▪ Two-phase input (X6, X7) ▪ Incremental input (X6), Decremental input (X7) ▪ Counter input (X6), Incremental/decremental control input (X7)
401	High-speed counter: Channel 7	Unused	▪ Incremental input (X7) ▪ Decremental input (X7)
402/ 401	Pulse output: Channel 0	Unused	▪ Pulse output (Y0, Y1) ▪ Pulse output (Y0, Y1), Home input (X4) ▪ PWM output (Y0)
402/ 401	Pulse output: Channel 1	Unused	▪ Pulse output (Y2, Y3) ▪ Pulse output (Y2, Y3), Home input (X5) ▪ PWM output (Y2)
402/ 401	Pulse output: Channel 2	Unused	▪ Pulse output (Y4, Y5) ▪ Pulse output (Y4, Y5), Home input (X6) ▪ PWM output (Y4)
402/ 401	Pulse output: Channel 3 (32k type only)	Unused	▪ Pulse output (Y6, Y7) ▪ Pulse output, (Y6, Y7) Home input (X7) ▪ PWM output (Y6)
403	Pulse catch input: X0	Disable	Disable/Enable
403	Pulse catch input: X1	Disable	Disable/Enable
403	Pulse catch input: X2	Disable	Disable/Enable
403	Pulse catch input: X3	Disable	Disable/Enable
403	Pulse catch input: X4	Disable	Disable/Enable
403	Pulse catch input: X5	Disable	Disable/Enable
403	Pulse catch input: X6	Disable	Disable/Enable
403	Pulse catch input: X7	Disable	Disable/Enable
404/ 405	Interrupt input: X0→Interrupt 0	Unused	Rising edge/Falling edge
404/ 405	Interrupt input: X1→Interrupt 1	Unused	Rising edge/Falling edge
404/ 405	Interrupt input: X2→Interrupt 2	Unused	Rising edge/Falling edge
404/ 405	Interrupt input: X3→Interrupt 3	Unused	Rising edge/Falling edge
404/ 405	Interrupt input: X4→Interrupt 4	Unused	Rising edge/Falling edge
404/ 405	Interrupt input: X5→Interrupt 5	Unused	Rising edge/Falling edge

No.	Name	Default	Values
404/ 405	Interrupt input: X6→Interrupt 6	Unused	Rising edge/Falling edge
404/ 405	Interrupt input: X7→Interrupt 7	Unused	Rising edge/Falling edge

☞ **The input modes two-phase, incremental/decremental, or incremental/decremental control require a second channel. If channel 0, channel 2, channel 4, or channel 6 have been set to one of these modes, the settings for channel 1, channel 3, channel 5, and channel 7, respectively, will be invalid.**

Only channel 0 and channel 2 are available for the reset input of the high-speed counter.

Input numbers X4 to X7 can be used as home input of pulse output channels 0 to 3. When using the home return function, always set the home input. In this case, X4 to X7 cannot be used as high-speed counter inputs.

CPU outputs which have been specified as pulse output or PWM output cannot be used as normal outputs.

If the same input has been set as high-speed counter input, pulse catch input or interrupt input, the following order of precedence is effective: High-speed counter → Pulse catch → Interrupt.

High-Speed Counter, Pulse-Catch Input, Interrupt Input (Relay types)

No.	Name	Default	Values
402	High-speed counter: Channel 0	Unused	▪ Incremental input (X0) ▪ Decremental input (X0) ▪ Two-phase input (X0, X1)
402	High-speed counter: Channel 1	Unused	▪ Incremental input (X1) ▪ Decremental input (X1) ▪ Two-phase input (X0, X1)
402	High-speed counter: Channel 2	Unused	▪ Incremental input (X2) ▪ Decremental input (X2) ▪ Two-phase input (X2, X3)
402	High-speed counter: Channel 3	Unused	▪ Incremental input (X3) ▪ Decremental input (X3) ▪ Two-phase input (X2, X3)
402	High-speed counter: Channel 4	Unused	▪ Incremental input (X4) ▪ Decremental input (X4) ▪ Two-phase input (X4, X5)
402	High-speed counter: Channel 5	Unused	▪ Incremental input (X5) ▪ Decremental input (X5) ▪ Two-phase input (X4, X5)
402	High-speed counter: Channel 6	Unused	▪ Incremental input (X6) ▪ Decremental input (X6) ▪ Two-phase input (X6, X7)
402	High-speed counter: Channel 7	Unused	▪ Incremental input (X7) ▪ Decremental input (X7) ▪ Two-phase input (X6, X7)

No.	Name	Default	Values
400	High-speed counter: Channel 8 (with pulse I/O cassette)	Unused	▪ Two-phase input (X100, X101) ▪ Two-phase input (X100, X101), Reset input (X102) ▪ Incremental input (X100) ▪ Incremental input (X100), Reset input (X102) ▪ Decremental input (X100) ▪ Decremental input (X100), Reset input (X102) ▪ Incremental input (X100), Decremental input (X101) ▪ Incremental input (X100), Decremental input (X101), Reset input (X102) ▪ Counter input (X100), Incremental/decremental control input (X101) ▪ Counter input (X100), Incremental/decremental control input (X101), Reset input (X102)
400	High-speed counter: Channel 9 (with pulse I/O cassette)	Unused	▪ Incremental input (X101) ▪ Incremental input (X101), Reset input (X102) ▪ Decremental input (X101) ▪ Decremental input (X101), Reset input (X102)
401	High-speed counter: Channel A (with pulse I/O cassette)	Unused	▪ Two-phase input (X200, X201) ▪ Two-phase input (X200, X201), Reset input (X202) ▪ Incremental input (X200) ▪ Incremental input (X200), Reset input (X202) ▪ Decremental input (X200) ▪ Decremental input (X200), Reset input (X202) ▪ Incremental input (X200), Decremental input (X201) ▪ Incremental input (X200), Decremental input (X201), Reset input (X202) ▪ Counter input (X200), Incremental/decremental control input (X201) ▪ Counter input (X200), Incremental/decremental control input (X201), Reset input (X202)
401	High-speed counter: Channel B (with pulse I/O cassette)	Unused	▪ Incremental input (X201) ▪ Incremental input (X201), Reset input (X202) ▪ Decremental input (X201) ▪ Decremental input (X201), Reset input (X202)
400	Pulse output: Channel 0 (with pulse I/O cassette)	Unused	▪ Pulse output (Y100, Y101), Home input (X102) ▪ PWM output (Y100)
401	Pulse output: Channel 1 (with pulse I/O cassette)	Unused	▪ Pulse output (Y200, Y201), Home input (X202) ▪ PWM output (Y200)
403	Pulse catch input: X0	Disable	Disable/Enable
403	Pulse catch input: X1	Disable	Disable/Enable
403	Pulse catch input: X2	Disable	Disable/Enable
403	Pulse catch input: X3	Disable	Disable/Enable
403	Pulse catch input: X4	Disable	Disable/Enable
403	Pulse catch input: X5	Disable	Disable/Enable
403	Pulse catch input: X6	Disable	Disable/Enable
403	Pulse catch input: X7	Disable	Disable/Enable
403	Pulse catch input: X100	Disable	Disable/Enable

No.	Name	Default	Values
403	Pulse catch input: X101	Disable	Disable/Enable
403	Pulse catch input: X102	Disable	Disable/Enable
403	Pulse catch input: X200	Disable	Disable/Enable
403	Pulse catch input: X201	Disable	Disable/Enable
403	Pulse catch input: X202	Disable	Disable/Enable
404/ 405	Interrupt input: X0→Interrupt 0	Unused	Rising edge/Falling edge
404/ 405	Interrupt input: X1→Interrupt 1	Unused	Rising edge/Falling edge
404/ 405	Interrupt input: X2→Interrupt 2	Unused	Rising edge/Falling edge
404/ 405	Interrupt input: X3→Interrupt 3	Unused	Rising edge/Falling edge
404/ 405	Interrupt input: X4→Interrupt 4	Unused	Rising edge/Falling edge
404/ 405	Interrupt input: X5→Interrupt 5	Unused	Rising edge/Falling edge
404/ 405	Interrupt input: X6→Interrupt 6	Unused	Rising edge/Falling edge
404/ 405	Interrupt input: X7→Interrupt 7	Unused	Rising edge/Falling edge
404/ 406	Interrupt input X100→Interrupt 8	Unused	Rising edge/Falling edge
404/ 406	Interrupt input X101→Interrupt 9	Unused	Rising edge/Falling edge
404/ 406	Interrupt input X102→Interrupt 10	Unused	Rising edge/Falling edge
404/ 406	Interrupt input X200→Interrupt 11	Unused	Rising edge/Falling edge
404/ 406	Interrupt input X201→Interrupt 12	Unused	Rising edge/Falling edge
404/ 406	Interrupt input X202→Interrupt 13	Unused	Rising edge/Falling edge

☞ **If the same input has been set as high-speed counter input, pulse catch input or interrupt input, the following order of precedence is effective: High-speed counter → Pulse catch → Interrupt.**

The two-phase input mode requires a second channel. If channels 0, 2, 4, or 6 have been set to two-phase mode, channels 1, 3, 5, or 7, respectively, must also be set to this mode.

The settings for pulse catch inputs and interrupt inputs can only be specified in the system registers.

Using the pulse I/O cassette:

☞ **The input modes two-phase, incremental/decremental, or incremental/decremental control require a second channel. If channel 8, or channel A have been set to one of these modes, the settings for channel 9, and channel B, respectively, will be invalid.**

If reset input settings overlap for channel 8 and channel 9, the channel 9 setting takes precedence. If reset input settings overlap for channel A and channel B, the channel B setting takes precedence.

System registers 400 applies when the pulse I/O cassette is installed in cassette mounting part 0. System register 401 applies when the pulse I/O cassette is installed in cassette mounting part 1.

Input numbers X102 and X202 can be used as home input of pulse output channels 0 and 1. When using the home return function, always set the home input. In this case, X102 and X202 cannot be used as reset inputs for channels 8 to B.

System registers 404/406 apply when the pulse I/O cassette is used.

Time Constants

No.	Name	Default	Values
430	Time constant of input X0	Unused	1.0ms 2.0ms 4.0ms 8.0ms 16.0ms 32.0ms 64.0ms 128.0ms 256.0ms
430	Time constant of input X1		
430	Time constant of input X2		
430	Time constant of input X3		
431	Time constant of input X4		
431	Time constant of input X5		
431	Time constant of input X6		
431	Time constant of input X7		
432	Time constant of input X8		
432	Time constant of input X9		
432	Time constant of input XA		
432	Time constant of input XB		
433	Time constant of input XC		
433	Time constant of input XD		
433	Time constant of input XE		
433	Time constant of input XF		
434	Time constant of input X10		
434	Time constant of input X11		
434	Time constant of input X12		
434	Time constant of input X13		
435	Time constant of input X14		
435	Time constant of input X15		
435	Time constant of input X16		
435	Time constant of input X17		
436	Time constant of input X18		
436	Time constant of input X19		
436	Time constant of input X1A		
436	Time constant of input X1B		
437	Time constant of input X1C		

No.	Name	Default	Values
437	Time constant of input X1D		
437	Time constant of input X1E		
437	Time constant of input X1F		

TOOL Port

No.	Name	Default	Values
412	TOOL port - communication mode	MEWTOCOL-COM Slave	MEWTOCOL-COM Slave/Program controlled
410	TOOL port - station number	1	1–99
415	TOOL port - baud rate	115200 baud	115200/57600/38400/19200/9600/4800/2400 baud
413	TOOL port - sending data length	8 bits	7 bits/8 bits
413	TOOL port - sending parity check	With-Odd	None/With-Odd/With-Even
413	TOOL port - sending stop bit	1 bit	1 bit/2 bits
413	TOOL port - sending start code	No-STX	No-STX/STX
413	TOOL port - sending end code/reception done condition	CR	CR/CR+LF/ETX/None
420	TOOL port - receive buffer starting address	4096	0–12282 (16k type) 0–32762 (32k type)
421	TOOL port - receive buffer capacity	0 words	0-2048 words
412	TOOL port - modem connection	Disable	Disable/Enable

COM Port

No.	Name	Default	Values
412	COM port 1 - communication mode	MEWTOCOL-COM Master/Slave	MEWTOCOL-COM Master/Slave/Program controlled/PLC Link/Modbus RTU Master/Slave
410	COM port 1 - station number	1	1–99
415	COM port 1 - baud rate[1)]	9600 baud	115200/57600/38400/19200/9600/4800/2400 baud
413	COM port 1 - sending data length[1)]	8 bits	7 bits/8 bits
413	COM port 1 - sending parity check[1)]	With-Odd	None/With-Odd/With-Even
413	COM port 1 - sending stop bit	1 bit	1 bit/2 bits
413	COM port 1 - sending start code[1)]	No-STX	No-STX/STX
413	COM port 1 - sending end code/reception done condition[1)]	CR	CR/CR+LF/ETX/None

No.	Name	Default	Values
416	COM port 1 - receive buffer starting address	0	0–12282 (16k type) 0–32762 (32k type)
417	COM port 1 - receive buffer capacity	0 words	0-2048 words
412	COM port 1 - modem connection	Disable	Disable/Enable
412	COM port 2 - port selection[2]	Internal USB port (32k type only)	Internal USB port/Communication cassette
412	COM port 2 - communication mode	MEWTOCOL-COM Master/Slave	MEWTOCOL-COM Master/Slave/Program controlled/Modbus RTU Master/Slave
411	COM port 2 - station number	1	1–99
415	COM port 2 - baud rate[1]	9600 baud	115200/57600/38400/19200/9600/4800/2400 baud
414	COM port 2 - sending data length[1]	8 bits	7 bits/8 bits
414	COM port 2 - sending parity check[1]	With-Odd	None/With-Odd/With-Even
414	COM port 2 - sending stop bit[1]	1 bit	1 bit/2 bits
414	COM port 2 - sending start code[1]	No-STX	No-STX/STX
414	COM port 2 - sending end code/reception done condition[1]	CR	CR/CR+LF/ETX/None
418	COM port 2 - receive buffer starting address	2048	0–12282 (16k type) 0–32762 (32k type)
419	COM port 2 - receive buffer capacity	0 words	0–2048 words
412	COM port 2 - modem connection	Disable	Disable/Enable

[1] For PLC Link, the communication format and baud rate settings are fixed:

Data length:	8 bits
Parity:	Odd
Stop bit:	1 bit
End code:	CR
Start code:	No STX

Other system register settings will be ignored.

[2] CPU types C30 and C60 offer a USB port. To use this port, COM port 2 must be set to "Internal USB port". In this case, COM port 2 of the communication cassette cannot be used. Vice versa, if COM port 2 has been set to "Communication cassette", the USB port cannot be used.

For C14, COM port 2 is set to "Communication cassette". This setting is fixed.

40.8.3 System registers for FP-Sigma

Memory size

No.	Name	Default	Values
0	Sequence program area size	12/16/32 kwords[1]	Fixed

[1] Depending on PLC type (12k, 16k, or 32k type)

Hold on/off

No.	Name	Default	Values
5[1)]	Counter start address	1008	0–1024
6[1)]	Timer/Counter hold area start address	1008	0–1024
7[1)]	Internal relay hold area start address (in word units)	248	0–256
8[1)]	Data register hold area start address	32710	0–32763
10	Link relay hold area start address for PLC Link 0 (in word units)	64	0–64
11	Link relay hold area start address for PLC Link 1 (in word units)	128	64–128
12	Link register hold area start address for PLC Link 0	128	0–128
13	Link register hold area start address for PLC Link 1	256	128–256
14[1)]	Step ladder hold/non-hold	Non-hold	Hold/Non-hold

[1)]
- These settings are effective if the optional backup battery is installed
- If no backup battery is used, do not change the default settings. Otherwise proper functioning of hold/non-hold values cannot be guaranteed.

Act on Error

No.	Name	Default	Values
4	Battery error indication	Disable	Disable: When a battery error occurs, a self-diagnostic error is not issued and the ERROR LED will not flash. Enable: When a battery error occurs, a self-diagnostic error is issued and the ERROR will LED flash.
4	DF-, P-function leading/falling edge detection	Holds result	Holds result/disregards result
20	Duplicate output	Enable	Fixed
23	I/O verification error	Stop	Stop/Continue
26	Operation error	Stop	Stop/Continue

Time-Out

No.	Name	Default	Values
30	Watchdog timer time-out	400.0ms	Fixed
31	Multi-frame communication time	6500.0ms	10.0–81900.0ms
32	Timeout value for the communication functions based on F145, F146, F152, F153	10000.0ms	10.0–81900.0ms
34	Constant scan time	0.0ms	0.0–350.0ms 0.0: Normal scan (non-constant)

PLC Link

No.	Name	Default	Values
46	PLC Link 0 and 1 allocation setting	Normal	Normal/Reverse
47	PLC link 0 - Highest station number in network	16	1–16
40	PLC link 0 - Link relays - Send/receive area - Number of words shared by all linked PLCs	0	0–64 words
42	PLC link 0 - Link relays - Send area - Start sending from this word address	0	0–63

No.	Name	Default	Values
43	PLC link 0 - Link relays - Send area - Number of words to send	0	0–64 words
41	PLC link 0 - Link registers - Send/receive area - Number of words shared by all linked PLCs	0	0–128 words
44	PLC link 0 - Link registers - Send area - Start sending from this word address	0	0–127
45	PLC link 0 - Link registers - Send area - Number of words to send	0	0–127 words
57	PLC link 1 - Highest station number in network	16	1–16
50	PLC link 1 - Link relays - Send/receive area - Number of words shared by all linked PLCs	0	0–64 words
52	PLC link 1 - Link relays - Send area - Start sending from this word address	64	64–127
53	PLC link 1 - Link relays - Send area - Number of words to send	0	0–64 words
51	PLC link 1 - Link registers - Send/receive area - Number of words shared by all linked PLCs	0	0–128 words
54	PLC link 1 - Link registers - Send area - Start sending from this word address	128	128–255
55	PLC link 1 - Link registers - Send area - Number of words to send	0	0–127 words

High-Speed Counter, Pulse Catch Input, Interrupt Input

No.	Name	Default	Values
400	High-speed counter: Channel 0	Unused	▪ Two-phase input (X0, X1) ▪ Two-phase input (X0, X1), Reset input (X2) ▪ Incremental input (X0) ▪ Incremental input (X0), Reset input (X2) ▪ Decremental input (X0) ▪ Decremental input (X0), Reset input (X2) ▪ Incremental input (X0), Decremental input (X1) ▪ Incremental input (X0), Decremental input (X1), Reset input (X2) ▪ Counter input (X0), Incremental/decremental control input (X1) ▪ Counter input (X0), Incremental/decremental control input (X1), Reset input (X2)
400	High-speed counter: Channel 1	Unused	▪ Incremental input (X1) ▪ Incremental input (X1), Reset input (X2) ▪ Decremental input (X1) ▪ Decremental input (X1), Reset input (X2)
401	High-speed counter: Channel 2	Unused	▪ Two-phase input (X3, X4) ▪ Two-phase input (X3, X4), Reset input (X5) ▪ Incremental input (X3) ▪ Incremental input (X3), Reset input (X5) ▪ Decremental input (X3) ▪ Decremental input (X3), Reset input ▪ Incremental input (X3), Decremental input (X4) ▪ Incremental input (X3), Decremental input (X4), Reset input ▪ Counter input (X0), Incremental/decremental control input (X5) ▪ Counter input (X3), Incremental/decremental control input (X4), Reset input X5)
401	High-speed counter: Channel 3	Unused	▪ Incremental input (X4) ▪ Incremental input (X4), Reset input (X5) ▪ Decremental input (X4) ▪ Decremental input (X4), Reset input (X5)

No.	Name	Default	Values
402	Pulse catch input: X0	Disable	Disable/Enable
402	Pulse catch input: X1	Disable	Disable/Enable
402	Pulse catch input: X2	Disable	Disable/Enable
402	Pulse catch input: X3	Disable	Disable/Enable
402	Pulse catch input: X4	Disable	Disable/Enable
402	Pulse catch input: X5	Disable	Disable/Enable
402	Pulse catch input: X6	Disable	Disable/Enable
402	Pulse catch input: X7	Disable	Disable/Enable
403	Interrupt input: X0→Interrupt 0	Unused	Rising edge/Falling edge
403	Interrupt input: X1→Interrupt 1	Unused	Rising edge/Falling edge
403	Interrupt input: X2→Interrupt 2	Unused	Rising edge/Falling edge
403	Interrupt input: X3→Interrupt 3	Unused	Rising edge/Falling edge
403	Interrupt input: X4→Interrupt 4	Unused	Rising edge/Falling edge
403	Interrupt input: X5→Interrupt 5	Unused	Rising edge/Falling edge
403	Interrupt input: X6→Interrupt 6	Unused	Rising edge/Falling edge
403	Interrupt input: X7→Interrupt 7	Unused	Rising edge/Falling edge

☞ **If the same input has been set as high-speed counter input, pulse catch input or interrupt input, the following order of precedence is effective: High-speed counter → Pulse catch → Interrupt.**

If reset input settings overlap for channel 0 and channel 1, the channel 1 setting takes precedence. If reset input settings overlap for channel 2 and channel 3, the channel 3 setting takes precedence.

The input modes two-phase, incremental/decremental, or incremental/decremental control require a second channel. If channel 0 or channel 2 have been set to one of these modes, the settings for channel 1 and channel 3, respectively, will be invalid.

The settings for pulse catch inputs and interrupt inputs can only be specified in the system registers.

TOOL Port

No.	Name	Default	Values
412	TOOL port - communication mode	MEWTOCOL-COM Slave	MEWTOCOL-COM Slave/Program controlled
410	TOOL port -station number	1	1–99
415	TOOL port - baud rate	115200 baud	115200/57600/38400/19200/9600/4800/2400 baud
413	TOOL port - sending data length	8 bits	7 bits/8 bits

No.	Name	Default	Values
413	TOOL port -sending parity check	With-Odd	None/With-Odd/With-Even
413	TOOL port - sending stop bit	1 bit	1 bit/2 bits
413	TOOL port - sending start code	No-STX	No-STX/STX
413	TOOL port - sending end code/reception done condition	CR	CR/CR+LF/ETX/None
420	TOOL port- -receive buffer starting address	0	0–32762
421	TOOL port - receive buffer capacity	0	0-2048
412	TOOL port - modem connection	Disable	Disable/Enable

COM Port

No.	Name	Default	Values
412	COM port 1 - communication mode	MEWTOCOL-COM Master/Slave	MEWTOCOL-COM Master/Slave/Program controlled/PLC Link/Modbus RTU Master/Slave
410	COM port 1 -station number	1	1–99
415	COM port 1 - baud rate1) 2)	9600 baud	115200/57600/38400/19200/9600/4800/2400 baud
413	COM port 1 - sending data length	8 bits	7 bits/8 bits
413	COM port 1 -sending parity check1)	With-Odd	None/With-Odd/With-Even
413	COM port 1 - sending stop bit	1 bit	1 bit/2 bits
413	COM port 1 - sending start code1)	No-STX	No-STX/STX
413	COM port 1 - sending end code/reception done condition1)	CR	CR/CR+LF/ETX/None
416	COM port 1- -receive buffer starting address	0	0–32762
417	COM port 1 - receive buffer capacity	0	0-2048
412	COM port 1 - modem connection	Disable	Disable/Enable
412	COM port 2 - communication mode	MEWTOCOL-COM Master/Slave	MEWTOCOL-COM Master/Slave/Program controlled/Modbus RTU Master/Slave
411	COM port 2 - station number	1	1–99
415	COM port 2 - baud rate	9600 baud	115200/57600/38400/19200/9600/4800/2400
414	COM port 2 - sending data length	8 bits	7 bits/8 bits
414	COM port 2 - sending parity check	With-Odd	None/With-Odd/With-Even
414	COM port 2 -sending stop bit	1 bit	1 bit/2 bits
414	COM port 2 - sending start code	No-STX	No-STX/STX

No.	Name	Default	Values
414	COM port 2 - sending end code/reception done condition	CR	CR/CR+LF/ETX/None
418	COM port 2 - receive buffer starting address	2048	0–32762
419	COM port 2 - receive buffer capacity	0	0–2048
412	COM port 2 - modem connection	Disable	Disable/Enable

[1] For PLC Link, the communication format and baud rate settings are fixed:
Data length: 8 bits
Parity: Odd
Stop bit: 1 bit
End code: CR
Start code: No STX
Other system register settings will be ignored.

[2] FPG-COM4: For RS485 connections (COM port 1), the baud rate must be set in the system registers and with the DIP switch.

40.8.4 System registers for FP0R

Memory size

No.	Name	Default	Values
0	Sequence program area size	12/16/32 kwords[1]	Fixed

[1] Depending on PLC type (12k, 16k, or 32k type)

Hold on/off [1]

No.	Name	Default	Values
5	Counter start address	1008	0–1024
6	Timer/Counter hold area start address	1008	Fixed/0–1024[3]
7	Internal relay hold area start address (in word units)	248	Fixed/0–256[3]
8	Data register hold area start address	12000/ 32450[2]	Fixed/0–32763[3]
10	Link relay hold area start address for PLC Link 0 (in word units)	64	Fixed/0–64[3]
11	Link relay hold area start address for PLC Link 1 (in word units)	128	Fixed/64–128[3]
12	Link register hold area start address for PLC Link 0	128	Fixed/0–128[3]
13	Link register hold area start address for PLC Link 1	256	Fixed/128–256[3]
14	Step ladder hold/non-hold	Non-hold	Fixed or Hold/Non-hold3)

[1] FP0R-T32: If the battery is empty and additional hold areas have been defined, the hold/non-hold operation becomes unstable. The data value will become indefinite. It is cleared to 0 the next time the power is turned on.

[2] Depending on PLC type (16k/32k type)

[3] Depending on PLC type (Fixed for C10, C14, C16, C32, variable for T32, F32)

Act on Error

No.	Name	Default	Values
4	DF-, P-function leading/falling edge detection	Holds result	Holds result/disregards result
20	Duplicate output	Enable	Fixed
23	I/O verification error	Stop	Stop/Continue
26	Operation error	Stop	Stop/Continue

Time-Out

No.	Name	Default	Values
30	Watchdog timer time-out	699.1ms	Fixed
31	Multi-frame communication time	6500.0ms	10.0–81900.0ms
32	Timeout value for the communication functions based on F145, F146, F152, F153	10000.0ms	10.0–81900.0ms
34	Constant scan time	0.0ms	0.0–600.0ms 0.0: Normal scan (non-constant)

PLC Link

No.	Name	Default	Values
46	PLC Link 0 and 1 allocation setting	Normal	Normal/Reverse
47	PLC link 0 - Highest station number in network	16	1–16
40	PLC link 0 - Link relays - Send/receive area - Number of words shared by all linked PLCs	0	0–64 words
42	PLC link 0 - Link relays - Send area - Start sending from this word address	0	0–63
43	PLC link 0 - Link relays - Send area - Number of words to send	0	0–64 words
41	PLC link 0 - Link registers - Send/receive area - Number of words shared by all linked PLCs	0	0–128 words
44	PLC link 0 - Link registers - Send area - Start sending from this word address	0	0–127
45	PLC link 0 - Link registers - Send area - Number of words to send	0	0–127 words
57	PLC link 1 - Highest station number in network	16	1–16
50	PLC link 1 - Link relays - Send/receive area - Number of words shared by all linked PLCs	0	0–64 words
52	PLC link 1 - Link relays - Send area - Start sending from this word address	64	64–127
53	PLC link 1 - Link relays - Send area - Number of words to send	0	0–64 words
51	PLC link 1 - Link registers - Send/receive area - Number of words shared by all linked PLCs	0	0–128 words
54	PLC link 1 - Link registers - Send area - Start sending from this word address	128	128–255
55	PLC link 1 - Link registers - Send area - Number of words to send	0	0–127 words

High-Speed Counter, Pulse-Catch Input, Interrupt Input

No.	Name	Default	Values
400	High-speed counter: Channel 0	Unused	▪ Two-phase input (X0, X1) ▪ Two-phase input (X0, X1), Reset input (X2) ▪ Incremental input (X0) ▪ Incremental input (X0), Reset input (X2) ▪ Decremental input (X0) ▪ Decremental input (X0), Reset input (X2) ▪ Incremental input (X0), Decremental input (X1) ▪ Incremental input (X0), Decremental input (X1), Reset input (X2) ▪ Counter input (X0), Incremental/decremental control input (X1) ▪ Counter input (X0), Incremental/decremental control input (X1), Reset input (X2)
400	High-speed counter: Channel 1	Unused	▪ Incremental input (X1) ▪ Incremental input (X1), Reset input (X2) ▪ Decremental input (X1) ▪ Decremental input (X1), Reset input (X2)
400	High-speed counter: Channel 2	Unused	▪ Two-phase input (X3, X4) ▪ Two-phase input (X3, X4), Reset input (X5) ▪ Incremental input (X3) ▪ Incremental input (X3), Reset input (X5) ▪ Decremental input (X3) ▪ Decremental input (X3), Reset input (X5) ▪ Incremental input (X3), Decremental input (X4) ▪ Incremental input (X3), Decremental input (X4), Reset input (X5) ▪ Counter input (X3), Incremental/decremental control input (X4) ▪ Counter input (X3), Incremental/decremental control input (X4), Reset input X5)
400	High-speed counter: Channel 3	Unused	▪ Incremental input (X4) ▪ Incremental input (X4), Reset input (X5) ▪ Decremental input (X4) ▪ Decremental input (X4), Reset input (X5)
401	High-speed counter: Channel 4	Unused	▪ Two-phase input (X6, X7) ▪ Incremental input (X6) ▪ Decremental input (X6) ▪ Incremental input (X6), Decremental input (X7) ▪ Counter input (X6), Incremental/decremental control input (X7)
401	High-speed counter: Channel 5	Unused	▪ Incremental input (X7) ▪ Decremental input (X7)
402	Pulse output: Channel 0 (transistor types only)	Unused	▪ Pulse output (Y0, Y1) ▪ Pulse output (Y0, Y1), Home input (X4) ▪ Pulse output (Y0, Y1), Home input (X4), Position control trigger input (X0) ▪ PWM output (Y0)
402	Pulse output: Channel 1 (transistor types only)	Unused	▪ Pulse output (Y2, Y3) ▪ Pulse output (Y2, Y3), Home input (X5) ▪ Pulse output (Y2, Y3), Home input (X5), Position control trigger input (X1) ▪ PWM output (Y2)

No.	Name	Default	Values
402	Pulse output: Channel 2 (transistor types only)	Unused	▪ Pulse output (Y4, Y5) ▪ Pulse output (Y4, Y5), Home input (X6) ▪ Pulse output (Y4, Y5), Home input (X6), Position control trigger input (X2) ▪ PWM output (Y4)
402	Pulse output: Channel 3 (transistor types only)	Unused	▪ Pulse output (Y6, Y7) ▪ Pulse output (Y6, Y7), Home input (X7) ▪ Pulse output (Y6, Y7), Home input (X7), Position control trigger input (X3) ▪ PWM output (Y6)
403	Pulse catch input: X0	Disable	Disable/Enable
403	Pulse catch input: X1	Disable	Disable/Enable
403	Pulse catch input: X2	Disable	Disable/Enable
403	Pulse catch input: X3	Disable	Disable/Enable
403	Pulse catch input: X4	Disable	Disable/Enable
403	Pulse catch input: X5	Disable	Disable/Enable
403	Pulse catch input: X6	Disable	Disable/Enable
403	Pulse catch input: X7	Disable	Disable/Enable
404/ 405	Interrupt input: X0→Interrupt 0	Unused	Rising edge/Falling edge/Rising and falling edge
404/ 405	Interrupt input: X1→Interrupt 1	Unused	Rising edge/Falling edge/Rising and falling edge
404/ 405	Interrupt input: X2→Interrupt 2	Unused	Rising edge/Falling edge/Rising and falling edge
404/ 405	Interrupt input: X3→Interrupt 3	Unused	Rising edge/Falling edge/Rising and falling edge
404/ 405	Interrupt input: X4→Interrupt 4	Unused	Rising edge/Falling edge/Rising and falling edge
404/ 405	Interrupt input: X5→Interrupt 5	Unused	Rising edge/Falling edge/Rising and falling edge
404/ 405	Interrupt input: X6→Interrupt 6	Unused	Rising edge/Falling edge/Rising and falling edge
404/ 405	Interrupt input: X7→Interrupt 7	Unused	Rising edge/Falling edge/Rising and falling edge

☞ **If the same input has been set as high-speed counter input, pulse catch input or interrupt input, the following order of precedence is effective: High-speed counter → Pulse catch → Interrupt.**

If reset input settings overlap for channel 0 and channel 1, the channel 1 setting takes precedence. If reset input settings overlap for channel 2 and channel 3, the channel 3 setting takes precedence.

The input modes two-phase, incremental/decremental, or incremental/decremental control require a second channel. If channel 0, 2, or channel 4 has been set to one of these modes, the settings for channel 1, 3, and 5, respectively, will be invalid.

The settings for pulse catch inputs and interrupt inputs can only be specified in the system registers.

Transistor types (C16 and higher)

☞ **CPU outputs which have been specified as pulse output or PWM output cannot be used as normal outputs.**

Input numbers X4 to X7 can be used as home input of pulse output channels 0 to 3. When using the home return function, always set the home input. In this case, X4 to X7 cannot be used as high-speed counter inputs.

The output numbers for the deviation counter clear signal, which can be used with the home return function, are fixed for each channel.
For C16: Channel 0 = Y6, channel 1 = Y7
For C32/T32/F32: Channel 0 = Y8, channel 1 = Y9, channel 2 = YA, channel 3 = YB
If used for the deviation counter clear signal, these outputs are not available as pulse outputs.

Time Constants

No.	Name	Default	Values
430	Time constant of input X0	1.0ms	0.1ms 0.5ms 1.0ms 2.0ms 4.0ms 8.0ms 16.0ms 32.0ms 64.0ms
430	Time constant of input X1		
430	Time constant of input X2		
430	Time constant of input X3		
431	Time constant of input X4		
431	Time constant of input X5		
431	Time constant of input X6		
431	Time constant of input X7		
432[1]	Time constant of input X8		
432[1]	Time constant of input X9		
432[1]	Time constant of input XA		
432[1]	Time constant of input XB		
433[1]	Time constant of input XC		
433[1]	Time constant of input XD		
433[1]	Time constant of input XE		
433[1]	Time constant of input XF		

[1] 32k types only

TOOL Port

No.	Name	Default	Values
412	TOOL port - communication mode	MEWTOCOL-COM Slave	MEWTOCOL-COM Slave/Program controlled
410	TOOL port -station number	1	1–99
415	TOOL port - baud rate	115200 baud	115200/57600/38400/19200/9600/4800/2400 baud
413	TOOL port - sending data length	8 bits	7 bits/8 bits

No.	Name	Default	Values
413	TOOL port -sending parity check	With-Odd	None/With-Odd/With-Even
413	TOOL port - sending stop bit	1 bit	1 bit/2 bits
413	TOOL port - sending start code	No-STX	No-STX/STX
413	TOOL port - sending end code/reception done condition	CR	CR/CR+LF/ETX/None
420	TOOL port- -receive buffer starting address	0	0–12312 (16k type) 0–32762 (32k type)
421	TOOL port - receive buffer capacity	0	0-2048
412	TOOL port - modem connection	Disable	Disable/Enable

COM Port

No.	Name	Default	Values
412	COM port 1 - communication mode	MEWTOCOL-COM Master/Slave	MEWTOCOL-COM Master/Slave/Program controlled/PLC Link/Modbus RTU Master/Slave
410	COM port 1 -station number	1	1–99
415	COM port 1 - baud rate[1)]	9600 baud	115200/57600/38400/19200/9600/4800/2400 baud
413	COM port 1 - sending data length	8 bits	7 bits/8 bits
413	COM port 1 -sending parity check[1)]	With-Odd	None/With-Odd/With-Even
413	COM port 1 - sending stop bit	1 bit	1 bit/2 bits
413	COM port 1 - sending start code[1)]	No-STX	No-STX/STX
413	COM port 1 - sending end code/reception done condition[1)]	CR	CR/CR+LF/ETX/None
416	COM port 1- -receive buffer starting address	0	0–12312 (16k type) 0–32762 (32k type)
417	COM port 1 - receive buffer capacity	0	0-2048
412	COM port 1 - modem connection	Disable	Disable/Enable

[1)] For PLC Link, the communication format and baud rate settings are fixed:

Data length:	8 bits
Parity:	Odd
Stop bit:	1 bit
End code:	CR
Start code:	No STX

Other system register settings will be ignored.

40.8.5 System registers for FP0

Memory size

No.	Name	Default	Values
0	Sequence program area size	3/5/10 kwords[1]	Fixed

[1] Depending on PLC type (2.7k, 5k, or 10k type)

Hold on/off

No.	Name	Default	Values
5	Counter start address	100	0–144
6	Timer/Counter hold area start address	140/128/100[1]	Fixed/0–144[2]
7	Internal relay hold area start address (in word units)	61/55/48[1]	Fixed/0–63[2]
8	Data register hold area start address	1652/6112/12289[1]	Fixed/0–16382[2]
14	Step ladder hold/non-hold	Non-hold	Fixed/Hold/Non-hold[2]

- These settings are effective if the optional backup battery is installed
- If no backup battery is used, do not change the default settings. Otherwise proper functioning of hold/non-hold values cannot be guaranteed.

[1] Depending on PLC type (2,7k type/5k type/10k type)

[2] Depending on PLC type (Fixed value: 2,7k, 5k types, variable values: 10k type)

Act on Error

No.	Name	Default	Values
20	Duplicate output	Enable	Fixed
23	I/O verification error	Stop	Stop/Continue
24	Watchdog timer time-out by operation jam	Stop	Fixed
26	Operation error	Stop	Stop/Continue
27	Remote I/O slave link error	Stop	Stop/Continue

Time-Out

No.	Name	Default	Values
30	Watchdog timer time-out	210.0ms	Fixed
31	Multi-frame communication time	6500.0ms	10.0–81900.0ms
34	Constant scan time	0.0ms	0.0–160.0ms 0.0: Normal scan (non-constant)

High-Speed Counter, Pulse-Catch Input, Interrupt Input

No.	Name	Default	Values
400	High-speed counter: Channel 0	Unused	▪ Two-phase input (X0, X1) ▪ Two-phase input (X0, X1), Reset input (X2) ▪ Incremental input (X0) ▪ Incremental input (X0), Reset input (X2) ▪ Decremental input (X0) ▪ Decremental input (X0), Reset input (X2) ▪ Incremental input (X0), Decremental input (X1) ▪ Incremental input (X0), Decremental input (X1), Reset input (X2) ▪ Counter input (X0), Incremental/decremental control input (X1) ▪ Counter input (X0), Incremental/decremental control input (X1), Reset input (X2)
400	High-speed counter: Channel 1	Unused	▪ Incremental input (X1) ▪ Incremental input (X1), Reset input (X2) ▪ Decremental input (X1) ▪ Decremental input (X1), Reset input (X2)
401	High-speed counter: Channel 2	Unused	▪ Two-phase input (X3, X4) ▪ Two-phase input (X3, X4), Reset input (X5) ▪ Incremental input (X3) ▪ Incremental input (X3), Reset input (X5) ▪ Decremental input (X3) ▪ Decremental input (X3), Reset input (X5) ▪ Incremental input (X3), Decremental input (X4) ▪ Incremental input (X3), Decremental input (X4), Reset input (X5) ▪ Counter input (X3), Incremental/decremental control input (X4) ▪ Counter input (X3), Incremental/decremental control input (X4), Reset input (X5)
401	High-speed counter: Channel 3	Unused	▪ Incremental input (X4) ▪ Incremental input (X4), Reset input (X5) ▪ Decremental input (X4) ▪ Decremental input (X4), Reset input (X5)
402	Pulse catch input: X0	Disable	Disable/Enable
402	Pulse catch input: X1	Disable	Disable/Enable
402	Pulse catch input: X2	Disable	Disable/Enable
402	Pulse catch input: X3	Disable	Disable/Enable
402	Pulse catch input: X4	Disable	Disable/Enable
402	Pulse catch input: X5	Disable	Disable/Enable
403	Interrupt input: X0→Interrupt 0	Unused	Rising edge/Falling edge
403	Interrupt input: X1→Interrupt 1	Unused	Rising edge/Falling edge
403	Interrupt input: X2→Interrupt 2	Unused	Rising edge/Falling edge
403	Interrupt input: X3→Interrupt 3	Unused	Rising edge/Falling edge
403	Interrupt input: X4→Interrupt 4	Unused	Rising edge/Falling edge
403	Interrupt input: X5→Interrupt 5	Unused	Rising edge/Falling edge

☞ **If the same input has been set as high-speed counter input, pulse catch input or interrupt input, the following order of precedence is effective: High-speed counter → Pulse catch → Interrupt.**

If reset input settings overlap for channel 0 and channel 1, the channel 1 setting takes precedence. If reset input settings overlap for channel 2 and channel 3, the channel 3 setting takes precedence.

The input modes two-phase, incremental/decremental, or incremental/decremental control require a second channel. If channel 0 or channel 2 have been set to one of these modes, the settings for channel 1 and channel 3, respectively, will be invalid.

The settings for pulse catch inputs and interrupt inputs can only be specified in the system registers.

TOOL Port

No.	Name	Default	Values
410	TOOL port - station number	1	1–32
411	TOOL port - modem connection	Disable	Disable/Enable
411	TOOL port - sending data length	8 bits	7 bits/8 bits
414	TOOL port - baud rate	19200 baud	19200/9600 baud

COM Port

No.	Name	Default	Values
412	COM port 1 - communication mode	MEWTOCOL-COM Slave	MEWTOCOL-COM Slave/Program controlled
415	COM port 1 -station number	1	1–32
414	COM port 1 - baud rate	9600 baud	19200/9600/4800/2400/1200/600/300 baud
413	COM port 1 - sending data length	8 bits	7 bits/8 bits
413	COM port 1 -sending parity check	With-Odd	None/With-Odd/With-Even
413	COM port 1 - sending stop bit	1 bit	1 bit/2 bits
413	COM port 1 - sending start code	No-STX	No-STX/STX
413	COM port 1 - sending end code/reception done condition	CR	CR/CR+LF/ETX/None
417	COM port 1- -receive buffer starting address	0	0–1657/6141/16381[1)]
418	COM port 1 - receive buffer capacity	0	0–1658/6142/6144[1)]
416	COM port 1 - modem connection	Disable	Disable/Enable

[1)] Depending on PLC type (2.7k/5k/10k type)

40.8.6 System registers for FP-e

Memory size

No.	Name	Default	Values
0	Sequence program area size	3/5/10 kwords[1)]	Fixed

[1)] Depending on PLC type (2.7k, 5k, or 10k type)

Hold on/off

No.	Name	Default	Values
5	Counter start address	100	0–144
6	Timer/Counter hold area start address	140	0–144
7	Internal relay hold area start address (in word units)	61	0–63
8	Data register hold area start address	1652	0–1658
14	Step ladder hold/non-hold	Non-hold	Hold/Non-hold

- These settings are effective if the optional backup battery is installed
- If no backup battery is used, do not change the default settings. Otherwise proper functioning of hold/non-hold values cannot be guaranteed.

Act on Error

No.	Name	Default	Values
4	Battery error indication	Disable	Disable: When a battery error occurs, a self-diagnostic error is not issued and the ERROR LED will not flash. Enable: When a battery error occurs, a self-diagnostic error is issued and the ERROR will LED flash.
20	Duplicate output	Enable	Fixed
24	Watchdog timer time-out by operation jam	Stop	Fixed
26	Operation error	Stop	Stop/Continue

Time-Out

No.	Name	Default	Values
30	Watchdog timer time-out	210.0ms	Fixed
31	Multi-frame communication time	6500.0ms	10.0–81900.0ms
34	Constant scan time	0.0ms	0.0–160.0ms 0.0: Normal scan (non-constant)

High-Speed Counter, Pulse-Catch Input, Interrupt Input

No.	Name	Default	Values
400	High-speed counter: Channel 0	Unused	▪ Two-phase input (X0, X1) ▪ Two-phase input (X0, X1), Reset input (X2) ▪ Incremental input (X0) ▪ Incremental input (X0), Reset input (X2) ▪ Decremental input (X0) ▪ Decremental input (X0), Reset input (X2) ▪ Incremental input (X0), Decremental input (X1) ▪ Incremental input (X0), Decremental input (X1), Reset input (X2) ▪ Counter input (X0), Incremental/decremental control input (X1) ▪ Counter input (X0), Incremental/decremental control input (X1), Reset input (X2)
400	High-speed counter: Channel 1	Unused	▪ Incremental input (X1) ▪ Incremental input (X1), Reset input (X2) ▪ Decremental input (X1) ▪ Decremental input (X1), Reset input (X2)
401	High-speed counter: Channel 2	Unused	▪ Two-phase input (X3, X4) ▪ Two-phase input (X3, X4), Reset input (X5) ▪ Incremental input (X3) ▪ Incremental input (X3), Reset input (X5) ▪ Decremental input (X3) ▪ Decremental input (X3), Reset input (X5) ▪ Incremental input (X3), Decremental input (X4) ▪ Incremental input (X3), Decremental input (X4), Reset input (X5) ▪ Counter input (X3), Incremental/decremental control input (X4) ▪ Counter input (X3), Incremental/decremental control input (X4), Reset input X5)
401	High-speed counter: Channel 3	Unused	▪ Incremental input (X4) ▪ Incremental input (X4), Reset input (X5) ▪ Decremental input (X4) ▪ Decremental input (X4), Reset input (X5)
402	Pulse catch input: X0	Disable	Disable/Enable
402	Pulse catch input: X1	Disable	Disable/Enable
402	Pulse catch input: X2	Disable	Disable/Enable
402	Pulse catch input: X3	Disable	Disable/Enable
402	Pulse catch input: X4	Disable	Disable/Enable
402	Pulse catch input: X5	Disable	Disable/Enable
403	Interrupt input: X0→Interrupt 0	Unused	Rising edge/Falling edge
403	Interrupt input: X1→Interrupt 1	Unused	Rising edge/Falling edge
403	Interrupt input: X2→Interrupt 2	Unused	Rising edge/Falling edge
403	Interrupt input: X3→Interrupt 3	Unused	Rising edge/Falling edge
403	Interrupt input: X4→Interrupt 4	Unused	Rising edge/Falling edge
403	Interrupt input: X5→Interrupt 5	Unused	Rising edge/Falling edge

No.	Name	Default	Values
409	Number of temperature input values for averaging process	0	0–50

☞ **If the same input has been set as high-speed counter input, pulse catch input or interrupt input, the following order of precedence is effective: High-speed counter → Pulse catch → Interrupt.**

If reset input settings overlap for channel 0 and channel 1, the channel 1 setting takes precedence. If reset input settings overlap for channel 2 and channel 3, the channel 3 setting takes precedence.

The input modes two-phase, incremental/decremental, or incremental/decremental control require a second channel. If channel 0 or channel 2 have been set to one of these modes, the settings for channel 1 and channel 3, respectively, will be invalid.

The settings for pulse catch inputs and interrupt inputs can only be specified in the system registers.

TOOL Port

No.	Name	Default	Values
410	TOOL port - station number	1	1–32
411	TOOL port - modem connection	Disable	Disable/Enable
411	TOOL port - sending data length	8 bits	7 bits/8 bits
414	TOOL port - baud rate	19200 baud	19200/9600 baud

COM Port

No.	Name	Default	Values
412	COM port 1 - communication mode	MEWTOCOL-COM Slave	MEWTOCOL-COM Slave/Program controlled/Modbus RTU Slave
415	COM port 1 -station number	1	1–99
414	COM port 1 - baud rate	9600 baud	19200/9600/4800/2400/1200/600/300 baud
413	COM port 1 - sending data length	8 bits	7 bits/8 bits
413	COM port 1 -sending parity check	With-Odd	None/With-Odd/With-Even
413	COM port 1 - sending stop bit	1 bit	1 bit/2 bits
413	COM port 1 - sending start code	No-STX	No-STX/STX
413	COM port 1 - sending end code/reception done condition	CR	CR/CR+LF/ETX/None
417	COM port 1- -receive buffer starting address	0	0–1657
418	COM port 1 - receive buffer capacity	0	0-1658
416	COM port 1 - modem connection	Disable	Disable/Enable

40.8.7 System registers for FP2/FP2SH/FP10SH

Memory size (FP2 only)

No.	Name	Default	Values
0	Sequence program area size	12 kwords	2–16/2–32 1)
1	Machine language program area size	0 kwords	0–14/0–30 1)
2	Configuration area size	0 kwords	0–14/0–30 1)
3	File Register	4/20 kwords1)	0–14/0–30 1)

1) Depending on PLC type (16k/32k type)

Hold On/Off

FP2

No.	Name	Default	Values
51)	Counter start address	1000	0–1024
61)	Timer/Counter hold area start address	1000	0–1024
71)	Internal relay hold area start address (in word units)	200	0–253
81)	Data register hold area start address	4500	0–5998
9	File register hold area start address	3072/15360 2)	0–4093/0–20477 2)
10	Link relay hold area start address for PLC Link 0 (in word units)	64	0–64
11	Link relay hold area start address for PLC Link 1 (in word units)	128	64–128
12	Link register hold area start address for PLC Link 0	128	0–128
13	Link register hold area start address for PLC Link 1	256	128–256
141)	Step ladder hold/non-hold	Non-hold	Hold/Non-hold

1)
- These settings are effective if the optional backup battery is installed
- If no backup battery is used, do not change the default settings. Otherwise proper functioning of hold/non-hold values cannot be guaranteed.

2) Depending on PLC type (16k/32k type)

FP2SH/FP10SH

No.	Name	Default	Values
5	Counter start address	3000	0–3072
6	Timer/Counter hold area start address	3000	0–3072
7	Internal relay hold area start address (in word units)	500	0–887
8	Data register hold area start address	7680	30/60k: 0–10238 120k: 0–10236
9	File register hold area start address	24573	0–32765
10	Link relay hold area start address for PLC Link 0 (in word units)	64	0–64
11	Link relay hold area start address for PLC Link 1 (in word units)	128	64–128
16	Link relay hold area start address for MEWNET-H	640	128–640
12	Link register hold area start address for PLC Link 0	128	0–128
13	Link register hold area start address for PLC Link 1	256	128–256
17	Link register hold area start address for MEWNET-H	256	256–8448

No.	Name	Default	Values
18	Index register hold area start address	0	0–14
14	Step ladder hold/non-hold	Non-hold	Hold/Non-hold
15[1]	File register bank 1 hold area start address	0	0–32765
19[1]	File register bank 2 hold area start address	0	0–32765

[1] For FP2SH only

Act on Error

No.	Name	Default	Values
4	Battery error indication	Enable	Disable: When a battery error occurs, a self-diagnostic error is not issued and the ERROR LED will not flash. Enable: When a battery error occurs, a self-diagnostic error is issued and the ERROR will LED flash.
4	Internal relay (R)	Clear	Clear/Don't clear
4	Link relay (L)		
4	Timer/counter (T, C, SV, EV)		
4	Data register (DT)		
4	Link register (LD)		
4	File register (FL)		
4	Index register (I)		
4[1]	Error alarm relay (E)		
4	DF-, P-function leading/falling edge detection	Holds result	Holds result/disregards result
4[1]	Timer instruction operation	Synchronous	Synchronous/Asynchronous
4	Index modifier check	On	On/Off
20	Duplicate output	Enable	Fixed
21	Output unit fuse blow	Stop	Stop/Continue
22	Intelligent unit error		
23	I/O verification error		
24[1]	Watchdog timer time-out by operation jam		
26	Operation error		
27	Remote I/O slave link error		
28	I/O error in the remote I/O slave station		

[1] FP2SH/FP10SH only

Time-Out

No.	Name	Default	Values
29[1]	Operation time for the peripheral tasks	240.0µs	0.0–52428.0µs

No.	Name	Default	Values
30	Watchdog timer time-out	FP2: 627.5ms FP2SH/FP10SH: 100.0ms	FP2: Fixed FP2SH/FP10SH: 0.4–640.0ms
31	Multi-frame communication time	FP2/FP2SH: 10000ms FP10SH: 6500.0ms	FP2: 10.0–81900.0ms FP2SH/FP10SH: 10.0–81917.5ms
32	Timeout value for the communication functions based on F145, F146, F152, F153	10000.0ms	FP2: 10.0–81900.0ms FP2SH/FP10SH: 10.0–81917.5ms
33	Effective time setting for monitoring	FP2: 10000.0μs FP2SH/FP10SH: 163837.5μs	FP2: 2000–52428.0μs FP2SH/FP10SH: 2500.0–163837.5μs
34	Constant scan time	0.0ms	FP2: 0.0–620.0ms FP2SH/FP10SH: 0.0–640.0ms 0.0: Normal scan (non-constant)

1) FP2SH/FP10SH only

MEWNET-F

No.	Name	Default	Values
25	PLC start mode when MEWNET-F slave stations connection timeout occurs	Stop	Stop/Continue
35	Wait mode for checking connection	Wait	Wait/Don't wait
35	MEWNET-F slave stations connection timeout	0s	0–255s
36	Remote I/O update method	Synchronous	Synchronous/Asynchronous

PLC Link

No.	Name	Default	Values
46	PLC Link 0 and 1 allocation setting	Normal	Normal/Reverse
471)	PLC link 0 - Highest station number in network	16	1–16
40	PLC link 0 - Link relays - Send/receive area - Number of words shared by all linked PLCs	0	0–64 words
42	PLC link 0 - Link relays - Send area - Start sending from this word address	0	0–63
43	PLC link 0 - Link relays - Send area - Number of words to send	0	0–64 words
41	PLC link 0 - Link registers - Send/receive area - Number of words shared by all linked PLCs	0	0–128 words
44	PLC link 0 - Link registers - Send area - Start sending from this word address	0	0–127
45	PLC link 0 - Link registers - Send area - Number of words to send	0	0–127 words
571)	PLC link 1 - Highest station number in network	16	1–16
50	PLC link 1 - Link relays - Send/receive area - Number of words shared by all linked PLCs	0	0–64 words
52	PLC link 1 - Link relays - Send area - Start sending from this word address	64	64–127
53	PLC link 1 - Link relays - Send area - Number of words to send	0	0–64 words
51	PLC link 1 - Link registers - Send/receive area - Number of words shared by all linked PLCs	0	0–128 words
54	PLC link 1 - Link registers - Send area - Start sending from this word address	128	128–255

No.	Name	Default	Values
55	PLC link 1 - Link registers - Send area - Number of words to send	0	0–127 words

[1] Not for FP10SH

PROFIBUS/MEWNET-H

No.	Name	Default	Values
49	PROFIBUS, MEWNET-H link access method/scan	0	0–65535 (×256 bytes)

TOOL Port

FP2/FP2SH

No.	Name	Default	Values
410	TOOL port - station number	1	1–32
411	TOOL port - modem connection	Disable	Disable/Enable
411	TOOL port - sending data length	8 bits	7 bits/8 bits
414	TOOL port - baud rate	115200 baud	115200/57600/38400/19200/9600/4800/2400/1200 baud

FP10SH

No.	Name	Default	Values
414	TOOL port - baud rate	115200 baud	115200/57600/38400/19200/9600/4800/2400/1200 baud

COM Port

FP2/FP2SH

No.	Name	Default	Values
412	COM port 1 - communication mode	MEWTOCOL-COM Slave	MEWTOCOL-COM Slave/Program controlled
415	COM port 1 -station number	1	1–32
414	COM port 1 - baud rate	FP2: 19200 baud FP2SH: 9600 baud	115200/57600/38400/19200/9600/4800/2400/1200 baud
413	COM port 1 - sending data length	8 bits	7 bits/8 bits
413	COM port 1 -sending parity check	With-Odd	None/With-Odd/With-Even
413	COM port 1 - sending stop bit	1 bit	1 bit/2 bits
413	COM port 1 - sending start code	No-STX	No-STX/STX
413	COM port 1 - sending end code/reception done condition	CR	CR/CR+LF/ETX/None
417	COM port 1- -receive buffer starting address	0	FP2: 0–5997 FP2SH, 60k: 0–10237 FP2SH, 120k: 0–10235
418	COM port 1 - receive buffer capacity	0	FP2: 0–2048 FP2SH: 0–1024

No.	Name	Default	Values
416	COM port 1 - modem connection	Disable	Disable/Enable

FP10SH

No.	Name	Default	Values
412	COM port 1 - communication mode	MEWTOCOL-COM Slave	MEWTOCOL-COM Slave/Program controlled
417	COM port 1- -receive buffer starting address	0	30/60k: 0–10237 120k: 0–10235
418	COM port 1 - receive buffer capacity	0	0-1024

Multi-CPU Setting (FP10SH only)

No.	Name	Default	Values
420	I/O divided data 0	16#0	16#0–16#FFFF
421	I/O divided data 1	16#0	16#0–16#FFFF
422	I/O shared data 0	16#0	16#0–16#FFFF
423	I/O shared data 1	16#0	16#0–16#FFFF
424	WR CPU1 sending capacity	0	0–98 words
425	WR CPU1 send starting No.	0	0–97
426	WR CPU2 sending capacity	0	0–98 words
427	WR CPU2 send starting No.	0	0–97
428	WL CPU1 sending capacity	0	0–128 words
429	WL CPU1 send starting No.	0	0–127
430	WL CPU2 sending capacity	0	0–128 words
431	WL CPU2 send starting No.	0	0–127
432	DT CPU1 sending capacity	0	0–1024 words
433	DT CPU1 send starting No.	0	0–2047
434	DT CPU2 sending capacity	0	0–1024 words
435	DT CPU2 send starting No.	0	0–2047
436	LD CPU1 sending capacity	0	0–256 words
437	LD CPU1 send starting No.	0	0–255
438	LD CPU2 sending capacity	0	0–256 words
439	LD CPU2 send starting No.	0	0–255
440	FL CPU1 sending capacity	0	0–1024 words
441	FL CPU1 send starting No.	0	0–8818
442	FL CPU2 sending capacity	0	0–1024 words
443	FL CPU2 send starting No.	0	0–8818

I/O Access Control (FP2SH/FP10SH only)

No.	Name	Default	Values
444	I/O access control	Standard	Standard/Long/Selectable
445	Register value for slots 0 and 1	16#1010	16#0–16#FFFF
446	Register value for slots 2 and 3	16#1010	16#0–16#FFFF
447	Register value for slots 4 and 5	16#1010	16#0–16#FFFF
448	Register value for slots 6 and 7	16#1010	16#0–16#FFFF
449	Register value for expanded slots	16#0210	16#0–16#FFFF

40.9 Error codes

40.9.1 Table of syntax check error

In FPWIN Pro, syntax errors are detected by the compiler and are therefore not critical.

Error code	Name	Operation status	Description and steps to take
E1	Syntax error	Stops	A program with a syntax error has been written. Change to PROG. mode and correct the error.
E2 (* Note)	Duplicated output error	Stops	Two or more OT(Out) instructions and KP(Keep) instructions are programmed using the same relay. Change to PROG. mode and correct the program so that one relay is not used for two or more OT instructions and KP instructions. Or, set the duplicated output to “enable (K1)” in system register 20.
E3	Not paired error	Stops	For instructions which must be used in a pair such as jump (JP and LBL), one instruction is either missing or in an incorrect position. Change to PROG. mode and enter the two instructions which must be used in a pair in the correct positions.
E4	Parameter mismatch error	Stops	An instruction has been written which does not agree with system register settings. For example, the number setting in a program does not agree with the timer/counter range setting. Change to PROG. mode, check the system register settings, and change so that the settings and the instruction agree.
E5 (* Note)	Program area error	Stops	An instruction which must be written to a specific area (main program area or subprogram area) has been written to a different area (for example, a subroutine SUB to RET is placed before an ED instruction). Change to PROG. mode and enter the instruction into the correct area.
E6	Compile memory full error (Available PLC: FPΣ/FP-X/ FP2SH/FP10SH)	Stops	The program stored in the PLC is too large to compile in the program memory. Change to PROG. mode and reduce the total number of steps for the program.
E7	High-level instruction type error (Available PLC: FPΣ/FP-X/ FP2/FP2SH/FP3/ FP10SH)	Stops	In the program, high-level instructions, which execute in every scan and at the rising edge of the trigger, are programmed to be triggered by one contact [e.g., F0 (MV) and P0 (PMV) are programmed using the same trigger continuously]. Correct the program so that the high-level instructions executed in every scan and only at the rising edge are triggered separately.
E8	High-level instruction operand error	Stops	There is an incorrect operand in an instruction which requires a specific combination operands (for example, the operands must all be of a certain type). Enter the correct combination of operands.
E9	No program error (Available PLC: FP2SH/FP10SH)	Stops	Program may be damaged. Try to send the program again.
E10	Rewrite during RUN syntax error	Continues	When inputting with the programming tool software, a deletion, addition or change of order of an instruction (ED, LBL, SUB, RET, INT, IRET, SSTP, and STPE) that cannot perform a rewrite during RUN is being attempted. Nothing is written to the CPU.

◆NOTE

This error is also detected if you attempt to execute a rewrite containing a syntax error during RUN. In this case, nothing will be written to the CPU and operation will continue.

40.9.2 Table of self-Diagnostic errors

Not all errors apply to all PLCs.

E20 - E39

Error code	Name	Operation status	Description and steps to take
E20	CPU error	Stops	Probably a hardware abnormality. Please contact your dealer.
E21 E22 E23 E24 E25	RAM error	Stops	Probably an abnormality in the internal RAM. Please contact your dealer.
E26	User's ROM error	Stops	FP2, FP2SH, FP3, FP10SH: ROM is not installed. There may be a problem with the installed ROM. ▪ ROM contents are damaged ▪ Program size stored on the ROM is larger than the capacity of the ROM Check the contents of the ROM
			FP-X: If the master memory cassette is mounted, the master memor cassette may be damaged. Remove the master memory, and check whether the ERROR turns off. If the ERROR turned off, rewrite the master memory as its contents are damaged, and use it again. If the ERROR does not turn off, please contact your dealer.
			FP0, FP-e, FP∑, FP1 C14, C16: Probably an abnormality in the built-in ROM. Please contact your dealer.
			All FP-Ms and FP1 C24, C40, C56, and C72: Probably an abnormality in the memory unit or master memory unit. Program the memory unit or master memory unit again and try to operate. If the same error is detected, try to operate with another memory unit or master memory unit.
E27	Intelligent unit installation error	Stops	Intelligent units installed exceed the limitations (i.e. 4 or more link units). Turn off the power and re-configure intelligent units referring to the hardware manual.
E28	System register error	Stops	Probably an abnormality in the system register. Check the system register setting or initialize the system registers.
E29	Configuration parameter error	Stops	A parameter error was detected in the MEWNET-W2 configuration area. Set a correct parameter.
E30	Interrupt error 0	Stops	Probably a hardware abnormality. Please contact your dealer.

Error code	Name	Operation status	Description and steps to take
E31	Interrupt error 1	Stops	An interrupt occurred without an interrupt request. A hardware problem or error due to noise is possible. Turn off the power and check the noise conditions.
E32	Interrupt error 2	Stops	An interrupt occurred without an interrupt request. A hardware problem or error due to noise is possible. Turn off the power and check the noise conditions.
			There is no interrupt program for an interrupt which occurred. Check the number of the interrupt program and change it to agree with the interrupt request.
E33	Multi-CPU data unmatch error	CPU2 stops	This error occurs when a FP3/FP10SH is used as CPU2 for a multi-CPU system. Please contact your dealer.
E34	I/O status error	Stops	An abnormal unit is installed. Check the contents of special data register DT9036/DT90036 and locate the abnormal unit. Then turn off the power and replace the unit with a new one.
E35	MEWNET-F (remote I/O) slave illegal unit error	Stops	A unit, which cannot be installed on the slave station of the MEWNET-F link system, is installed on the slave station. Remove the illegal unit from the slave station.
E36	MEWNET-F limitation error	Stops	The number of slots or I/O points used for MEWNET-F exceeds the limitation. Re-configure the system so that the number of slots and I/O points is within the specified range.
E37	MEWNET-F I/O mapping error	Stops	I/O overlap or I/O setting that is over the range is detected in the allocated I/O and MEWNET-F I/O map. Re-configure the I/O map correctly.
E38	MEWNET-F slave I/O mapping error	Stops	I/O mapping for remote I/O terminal boards, remote I/O terminal units and I/O link unit is not correct. Re-configure the I/O map for slave stations according to the I/O points of the slave stations.
E39	IC memory card read error	Stops	When reading in the program from the IC memory card (due to automatic reading because of the dip switch 3 setting or program switching due to F14 (PGRD) instruction): ▪ IC memory card is not installed. ▪ There is no program file or it is damaged. ▪ Writing is disabled. ▪ There is an abnormality in the AUTOEXEC.SPG file. ▪ Program size stored on the card is larger than the capacity of the CPU. Install an IC memory card that has the program properly recorded and execute the read once again.

E40 and above

Error code	Name	Operation status	Description and steps to take
E40	I/O error	Selectable	With FP3/FP10SH, communication error in the MEWNET-TR system has occurred. For all other PLCs an abnormality in an I/O unit has been detected. Check the contents of special data registers DT9002 and DT9003/DT90002 and DT90003 and the erroneous MEWNET-TR master unit or abnormal I/O unit (also expansion unit or application cassette). Then check the unit. Selection of operation status using system register 21: ▪ to continue operation, set K1 (CONT) ▪ to stop operation, set K0 (STOP)
E41	Intelligent unit error	Selectable	An abnormality in an intelligent unit. Check the contents of special data registers DT9006 and DT9007/DT90006 and DT90007 and locate the abnormal intelligent unit. Then check the unit referring to its manual. Selection of operation status using system register 22: ▪ to continue operation, set K1 (CONT) ▪ to stop operation, set K0 (STOP)
E42	I/O unit verify error	Selectable	I/O unit wiring condition has changed compared to that at time of power-up. Check the contents of special data registers DT9010 and DT9011/DT90010 and DT90011 and locate the erroneous unit. Then check the unit and correct the wiring. Selection of operation status using system register 23: ▪ to continue operation, set K1 (CONT) ▪ to stop operation, set K0 (STOP)
E43	System watching dog timer error	Selectable	Scan time required for program execution exceeds the setting of the system watchdog timer. Check the program and modify it so that FP2SH/FP10SH can execute a scan within the specified time. Selection of operation status using system register 24: ▪ to continue operation, set K1 (CONT) ▪ to stop operation, set K0 (STOP)
E44	Slave station connecting time error for MEWNET-F system	Selectable	The time required for slave station connection exceeds the setting of the system register 35. Selection of operation status using system register 25: ▪ to continue operation, set K1 (CONT) ▪ to stop operation, set K0 (STOP)
E45	Operation error	Selectable	Operation became impossible when a high-level instruction was executed. Check the contents of special data registers DT9017 and DT9018/DT90017 and DT90018 to find the program address where the operation error occurred. Then correct the program. Refer to the explanation of operation error and the instruction. Selection of operation status using system register 26: ▪ to continue operation, set K1 (CONT) ▪ to stop operation, set K0 (STOP)

Error code	Name	Operation status	Description and steps to take
E46	Remote I/O communication error	Selectable	**MEWNET-F communication error:** A communication abnormally was caused by a transmission cable or during the power-down of a slave station. Check the contents of special data registers DT9131 to DT9137/DT90131 to DT90137 and locate the abnormal slave station and recover the slave condition. Selection of operation status using system register 27: ▪ to continue operation, set K1 (CONT) ▪ to stop operation, set K0 (STOP) **S-Link communication error (with FP0-SL1 unit only):** When one of the S-LINK errors (ERR1, 3 or 4) has been deteced,error code E46 (remote I/O (S-LINK) communication error) is stored. Selection of operation status using system register 27: ▪ to continue operation, set K1 (CONT) ▪ to stop operation, set K0 (STOP)
E47	MEWNET-F attribute error	Selectable	MEWNET-F communication error A communication abnormally was caused by a transmission cable or during the power-down of a slave station. Check the contents of special data registers DT9131 to DT9137/DT90131 to DT90137 and locate the abnormal slave station and recover the communication condition. Selection of operation status using system register27: ▪ to continue operation,set K1 ▪ to stop operation, set K0
E50	Backup battery error	Continues	The voltage of the backup battery lowered or the backup battery of CPU is not installed. Check the installation of the backup battery and then replace battery if necessary. By setting the system register 4 in K0 (NO), you can disregard this error. However, the BATT. LED turns on.
E51	MEWNET-F terminal station error	Continues	Terminal station settings were not properly performed.Check stations at both ends of the communication path, and set them in the terminal station using the dip switches.
E52	MEWNET-F I/O update synchronous error	Continues	Set the INITIALIZE/TEST selector to the INITIALIZE position while keeping the mode selector in the RUN position. If the same error occurs after this, please contact your dealer.
E53	Multi-CPU registration error	Continues	Abnormality was detected when the multi-CPU system was used. Please contact your dealer.
E54	IC memory card backup battery error	Continues	The voltage of the backup battery for the IC memory card is getting low. The BATT. LED does not turn on. Charge or replace the backup battery of IC memory card. (The contents of the IC memory card cannot be guaranteed.)
E55	IC memory card backup battery error	Continues	The voltage of the backup battery for IC memory card is getting low. The BATT. LED does not turn on. Charge or replace the backup battery of IC memory card. (The contents of the IC memory card cannot be guaranteed.)
E56	Incompatible IC memory card error	Continues	The IC memory card installed is not compatible with FP2SH/FP10SH. Replace the IC memory card compatible with FP2SH/FP10SH.
E57	No unit for the configuration	Continues	MEWNET-W2 The MEWNET-W2 link unit is not installed in the slot specified using the configuration data. Either install a unit in the specified slot or change the parameter.

Error code	Name	Operation status	Description and steps to take
E100 to E199	Self- diagnostic error set by F148 (ERR)/ P148 (PERR) instruction	Stops	The self-diagnostic error specified by the F148 (ERR)/P148 (PERR) instruction is occurred. Take steps to clear the error condition according to the specification you chose.
E200 to E299		Continues	

40.9.3 Table of communication check error

Error code	Name	Operation status	Description and steps to take
E63	PLC error mode (Available PLC: FP2/FP2SH/FP3/ FP10SH)	Stops	Transfer was attempted in the RUN mode. Switch the mode and execute once again.
E64	No ROM/RAM error (Available PLC: FP2/FP2SH/FP3/ FP10SH)	Stops	An abnormality occurred when loading RAM to ROM/IC memory card. There may be a problem with the ROM or IC memory card. - When loading, the specified contents exceeded the capacity (256 KB). - Write error occurs. - ROM or IC memory card is not installed. - ROM or IC memory card does not conform to specifications. Check the contents of the ROM or IC memory card.
E65	Protect error	Stops	Transfer was attempted during ROM operation or IC memory card operation. Switch the mode and execute once again.
E66	PLC write error address error (Available PLC: FP2/FP2SH/FP3/ FP10SH)	Continues	In the programming tool software, program editing is being attempted by online access, but the program is not in agreement. (The program disagreement lies in another block.)Check the program.
E68	Rewrite during RUN error (Available PLC: FP2/FP2SH/FP3/ FP10SH)	Continues	When inputting with the programming tool software, editing of an instruction (ED, SUB, RET, INT, IRET, SSTP, and STPE) that cannot perform a rewrite during RUN is being attempted. Nothing is written to the CPU.

40.10 Error codes

40.10.1 Error Codes E1 to E8

Error code	Name of error	Operation status of PLC	Description and steps to take
E1 **(see note)**	Syntax error	Stops	A program with a syntax error has been written. Change to PROG mode and correct the error.
E2 **(see note)**	Duplicated output error	Stops	Two or more operation results are output to the same relay. (This error also occurs if the same timer/counter number is being used.) Change to PROG mode and correct the error. This error is also detected during online editing. No changes will be downloaded and operation will continue.
E3	Not paired error	Stops	For instructions which must be used in a pair such as jump (JP and LBL), one instruction is either missing or in an incorrect position. Change to PROG mode and correct the error.
E4 **(see note)**	Parameter mismatch error	Stops	An instruction has been written which does not agree with system register settings. For example, the timer/counter number setting in a program does not agree with the timer/counter range setting. Change to PROG mode and correct the error.
E5 **(see note)**	Program area error	Stops	An instruction was written to the wrong program area (main program area or subprogram area) Change to PROG mode and correct the error. This error is also detected during online editing. No changes will be downloaded and operation will continue.
E6 **(see note)**	Compile memory full error	Stops	The program stored in the PLC is too large to compile in the program memory. Change to PROG mode and correct the error.
E7 **(see note)**	High-level instruction type error	Stops	In the program, high-level F and P instructions are triggered by the same operation result. (While the execution condition is TRUE, F instructions are executed in every scan. P instructions are executed only once, at the leading edge of the execution condition.) Correct the program so that the high-level instructions executed in every scan and at the leading edge are triggered separately.
E8	High-level instruction operand combination error	Stops	There is an incorrect operand in an instruction which requires a specific combination of operands (for example, the operands must all be of a certain type). Change to PROG mode and correct the error.

◆NOTE

In FPWIN Pro, these errors are detected by the compiler. Therefore, they are not critical.

40.10.2 Self-Diagnostic Error Codes

Error code	Name of error	Operation status of PLC	Description and steps to take
E26	User's ROM error	Stops	Probably a hardware problem. Please contact your dealer.

Error code	Name of error		Operation status of PLC	Description and steps to take
E27	Unit installation error		Stops	The number of installed units exceeds the limit. Turn off the power supply and check the restrictions on unit combinations.
E28	System register error		Stops	Probably an error in the system registers. Check the system register settings.
E30	Interrupt error 0		Stops	Probably a hardware problem. Please contact your dealer.
E31	Interrupt error 1		Stops	An interrupt occurred without an interrupt request. A hardware problem or error due to noise is possible. Turn off the power and check the noise conditions.
E32	Interrupt error 2		Stops	An interrupt occurred without an interrupt request. A hardware problem or error due to noise is possible. Turn off the power and check the noise conditions.
				There is no interrupt program for an interrupt which occurred. Check the number of the interrupt program and change it to agree with the interrupt request.
E34	I/O status error		Stops	A faulty unit is installed. Replace the unit with a new one.
E42	I/O unit verify error		Selectable	The connection condition of an I/O unit has changed compared to that at the time of power-up. Check the error using sys_wVerifyErrorUnit_0_15 and locate the faulty I/O unit. Set the operation status using system register 23 to continue operation.
E45	Operation error		Selectable	Operation became impossible when a high-level instruction was executed. The causes of calculation errors vary depending on the instruction. Set the operation status using system register 23 to continue operation.
E100–E299	Self-diagnostic error set by F148_ERR	E100–E199	Stops	The self-diagnostic error specified by the F148_ERR (see page 1000) instruction occurred. Take steps to clear the error condition according to the specification you chose.
		E200–E299	Continues	

40.10.3 MEWTOCOL-COM Error Codes

Error code	Name	Description
!21	NACK error	Link system error
!22	WACK error	
!23	Unit no. overlap	
!24	Transmission format error	
!25	Link unit hardware error	
!26	Unit no. setting error	
!27	No support error	
!28	No response error	
!29	Buffer closed error	
!30	Time-out error	
!32	Transmission impossible error	
!33	Communication stop	
!36	No destination error	
!38	Other communication error	
!40	BCC error	A transfer error occurred in the data received.
!41	Format error	A formatting error in the command received was detected.

Error code	Name	Description
!42	No support error	A non-supported command was received.
!43	Multiple frames procedure error	A different command was received when processing multiple frames.
!50	Link setting error	A non-existing route number was specified. Verify the route number by designating the transmission station.
!51	Transmission time-out error	Transmission to another device is not possible because the transmission buffer is full.
!52	Transmit disable error	Transmission processing to another device is not possible (link unit runaway, etc.).
!53	Busy error	Processing of command received is not possible because of multiple frame processing or because command being processed is congested.
!60	Parameter error	Content of specified parameter does not exist or cannot be used.
!61	Data error	There was a mistake in the contact, data area, data number designation, size designation, range, or format designation.
!62	Registration over error	Operation was done when number of registrations was exceeded or when there was no registration.
!63	PC mode error	PC command that cannot be processed was executed during RUN mode.
!64	External memory error	An abnormality occurred when loading RAM to ROM/IC memory card. There may be a problem with the ROM or IC memory card. When loading, the specified contents exceeded the capacity. Write error occurs. ▪ ROM or IC memory card is not installed. ▪ ROM or IC memory card does not conform to specifications
!65	Protect error	A program or system register write operation was executed when the protect mode (password setting or DIP switch, etc.) or ROM operation mode was being used.
!66	Address error	There was an error in the code format of the address data. Also, when exceeded or insufficient address data, there was a mistake in the range designation.
!67	No program error and no data error	Cannot be read because there is no program in the program area or the memory contains an error. Or, reading of non-registered data was attempted.
!68	Rewrite during RUN error	When inputting with programming tool software, editing of an instruction (ED, SUB, RET, INT, IRET, SSTP, and STPE) that cannot perform a rewrite during RUN is being attempted. Nothing is written to the CPU.
!70	SIM over error	Program area was exceeded during a program write process.
!71	Exclusive access control error	A command that cannot be processed was executed at the same time as a command being processed.

40.11 MEWTOCOL-COM Communication Commands

Command name	Code	Description
Read contact area	RC (RCS) (RCP) (RCC)	Reads the on and off status of contacts. - Specifies only one point. - Specifies multiple contacts. - Specifies a range in word units.
Write contact area	WC (WCS) (WCP) (WCC)	Turns contacts on and off. - Specifies only one point. - Specifies multiple contacts. - Specifies a range in word units.
Read data area	RD	Reads the contents of a data area.
Write data area	WD	Writes data to a data area.
Read timer/counter set value area	RS	Reads the value set for a timer/counter.
Write timer/counter set value area	WS	Writes a timer/counter setting value.
Read timer/counter elapsed value area	RK	Reads the timer/counter elapsed value.
Write timer/counter elapsed value area	WK	Writes the timer/counter elapsed value.
Register or Reset contacts monitored	MC	Registers the contact to be monitored.
Register or Reset data monitored	MD	Registers the data to be monitored.
Monitoring start	MG	Monitors a registered contact or data using MD and MC.
Preset contact area (fill command)	SC	Embeds the area of a specified range in a 16-point on and off pattern.
Preset data area (fill command)	SD	Writes the same contents to the data area of a specified range.
Read system register	RR	Reads the contents of a system register.
Write system register	WR	Specifies the contents of a system register.
Read the status of PLC	RT	Reads the specifications of the PLC and error codes if an error occurs.
Remote control	RM	Switches the operation mode of the PLC.
Abort	AB	Aborts communication.

40.12 Hexadecimal/Binary/BCD

Decimal	Hexadecimal	Binary data	BCD data (Binary Coded Decimal)
0	0000	0000 0000 0000 0000	0000 0000 0000 0000
1	0001	0000 0000 0000 0001	0000 0000 0000 0001
2	0002	0000 0000 0000 0010	0000 0000 0000 0010
3	0003	0000 0000 0000 0011	0000 0000 0000 0011
4	0004	0000 0000 0000 0100	0000 0000 0000 0100
5	0005	0000 0000 0000 0101	0000 0000 0000 0101
6	0006	0000 0000 0000 0110	0000 0000 0000 0110
7	0007	0000 0000 0000 0111	0000 0000 0000 0111
8	0008	0000 0000 0000 1000	0000 0000 0000 1000
9	0009	0000 0000 0000 1001	0000 0000 0000 1001
10	000A	0000 0000 0000 1010	0000 0000 0001 0000
11	000B	0000 0000 0000 1011	0000 0000 0001 0001
12	000C	0000 0000 0000 1100	0000 0000 0001 0010
13	000D	0000 0000 0000 1101	0000 0000 0001 0011
14	000E	0000 0000 0000 1110	0000 0000 0001 0100
15	000F	0000 0000 0000 1111	0000 0000 0001 0101
16	0010	0000 0000 0001 0000	0000 0000 0001 0110
17	0011	0000 0000 0001 0001	0000 0000 0001 0111
18	0012	0000 0000 0001 0010	0000 0000 0001 1000
19	0013	0000 0000 0001 0011	0000 0000 0001 1001
20	0014	0000 0000 0001 0100	0000 0000 0010 0000
21	0015	0000 0000 0001 0101	0000 0000 0010 0001
22	0016	0000 0000 0001 0110	0000 0000 0010 0010
23	0017	0000 0000 0001 0111	0000 0000 0010 0011
24	0018	0000 0000 0001 1000	0000 0000 0010 0100
25	0019	0000 0000 0001 1001	0000 0000 0010 0101
26	001A	0000 0000 0001 1010	0000 0000 0010 0110
27	001B	0000 0000 0001 1011	0000 0000 0010 0111
28	001C	0000 0000 0001 1100	0000 0000 0010 1000
29	001D	0000 0000 0001 1101	0000 0000 0010 1001
30	001E	0000 0000 0001 1110	0000 0000 0011 0000
31	001F	0000 0000 0001 1111	0000 0000 0011 0001
.	.	.	.
.	.	.	.
.	.	.	.
63	**003F**	0000 0000 0011 1111	0000 0000 0110 0011
.	.	.	.
.	.	.	.
.	.	.	.
255	**00FF**	0000 0000 1111 1111	0000 0010 0101 0101
.	.	.	.
.	.	.	.
.	.	.	.
9999	**270F**	0010 0111 0000 1111	1001 1001 1001 1001

40.13 ASCII Codes

								b7									
								b6		0	0	0	0	1	1	1	1
								b5		0	0	1	1	0	0	1	1
								b4		0	1	0	1	0	1	0	1
										Most significant digit							
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	ASCII HEX code		0	1	2	3	4	5	6	7
				0	0	0	0	Least significant digit	0	NUL	DEL	SP AC E	0	@	P		p
				0	0	0	1		1	SOH	DC1	!	1	A	Q	a	q
				0	0	1	0		2	STX	DC2	″	2	B	R	b	r
				0	0	1	1		3	ETX	DC3	#	3	C	S	c	s
				0	1	0	0		4	EOT	DC4	$	4	D	T	d	t
				0	1	0	1		5	ENQ	NAK	%	5	E	U	e	u
				0	1	1	0		6	ACK	SYN	&	6	F	V	f	v
				0	1	1	1		7	BEL	ETB	′	7	G	W	g	w
				1	0	0	0		8	BS	CAN	(	8	H	X	h	x
				1	0	0	1		9	HT	EM	)	9	I	Y	i	y
				1	0	1	0		A	LF	SUB	∗	:	J	Z	j	z
				1	0	1	1		B	VT	ESC	+	;	K	[	k	{
				1	1	0	0		C	FF	FS	,	<	L	\	l	?
				1	1	0	1		D	CR	GS	–	=	M	]	m	}
				1	1	1	0		E	SO	RS	.	>	N	∧	n	~
				1	1	1	1		F	SI	US	/	?	O	_	o	DEL

40.14 Availability of all instructions on all PLC types

Instruction ● available ❍ partially available	FP0 2,7k / 5k	FP0 10k T32	FP0R C,F	FP0R 32k T32	FPΣ 12k, 16k	FPΣ 32k	FP-X CT, CP, C38A	FP-X L, C40RT0A	FP-X0 2,5k	FP-X0 8k	FP-e 2,7k	FP3, FP-C	FP2 16k	FP2 32k	FP2SH 32k	FP2SH 60k, 120k	FP10SH	page
ABS	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	67
ACOS	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	73
ActivateStepsOfStoppedSfc	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
ADD_DT_TIME	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	282
ADD_TIME	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	330
ADD_TOD_TIME	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	283
Adr_Of_Var	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
Adr_Of_VarOffs	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
AdrDT_Of_Offs	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
AdrFL_Of_Offs	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
AdrLast_Of_Var	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
AllSfcsStopped	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
ALT			●	●	●	●	●	●	●	●			●	●	●	●	●	888
AreaOffs_ToVar	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
ASIN	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	71
ATAN	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	76
ATAN2_YX	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	77
BOOL_TO_DINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	168
BOOL_TO_DWORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	132
BOOL_TO_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	145
BOOL_TO_STRING	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	210
BOOL_TO_UDINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	180
BOOL_TO_UINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	157
BOOL_TO_WORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	120
BOOL16_TO_WORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	121
BOOL32_TO_DWORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	133
BOOLS_TO_DWORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	134
BOOLS_TO_WORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	122
BRK												●	●	●	●	●	●	1016
ClearReceiveBuffer	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	754
CONCAT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	270
CONCAT_DATE_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	284
CONCAT_DATE_TOD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	285
CONCAT_DT_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	286
CONCAT_TIME_INT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	331
CONCAT_TOD_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	288
ControlSfc	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
ControlSfcAndData	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
COS	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	72
CRC16	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	84

Instruction ● available ❍ partially available	FP0 2,7k / 5k	FP0 10k T32	FP0R C,F	FP0R 32k T32	FPΣ 12k, 16k	FPΣ 32k	FP-X CT, CP, C38A	FP-X L, C40RT0A	FP-X0 2,5k	FP-X0 8k	FP-e 2,7k	FP3, FP-C	FP2 16k	FP2 32k	FP2SH 32k	FP2SH 60k, 120k	FP10SH	page
CSTP_CLEAR_STEP	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
CT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	705
CT_FB	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	702
CTD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	318
CTU	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	316
CTUD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	320
DATE_TO_STRING	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	216
DATE_TO_UDINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	190
DAY_OF_WEEK1	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	289
DELETE	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	272
DF	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	884
DFI			●	●			●	●	●	●	●		●	●	●	●	●	886
DFN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	885
DINT_TO_BCD_DWORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	245
DINT_TO_BOOL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	117
DINT_TO_DWORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	138
DINT_TO_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	149
DINT_TO_REAL	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	195
DINT_TO_STRING	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	221
DINT_TO_STRING_LEADING_ZEROS	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	225
DINT_TO_TIME	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	203
DINT_TO_UDINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	186
DINT_TO_UINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	162
DINT_TO_WORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	126
DIV_TIME_DINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	333
DIV_TIME_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	332
DIV_TIME_REAL	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	334
DT_TO_DATE	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	206
DT_TO_STRING	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	217
DT_TO_TOD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	208
DT_TO_UDINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	191
DWORD_BCD_TO_DINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	170
DWORD_BCD_TO_UDINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	183
DWORD_TO_BOOL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	115
DWORD_TO_BOOL32	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	239
DWORD_TO_BOOLS	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	242
DWORD_TO_DINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	171
DWORD_TO_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	148
DWORD_TO_REAL	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	193
DWORD_TO_STRING	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	214
DWORD_TO_TIME	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	201
DWORD_TO_UDINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	182
DWORD_TO_UINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	160
DWORD_TO_WORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	124

Instruction ● available ❍ partially available	FP0 2,7k / 5k	FP0 10k T32	FP0R C,F	FP0R 32k T32	FPΣ 12k, 16k	FPΣ 32k	FP-X CT, CP, C38A	FP-X L, C40RT0A	FP-X0 2,5k	FP-X0 8k	FP-e 2,7k	FP3, FP-C	FP2 16k	FP2 32k	FP2SH 32k	FP2SH 60k, 120k	FP10SH	page
Elem_OfArray1D	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
Elem_OfArray2D	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
Elem_OfArray3D	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
ETLANADDR_TO_STRING	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	235
ETLANADDR_TO_STRING_NO_LEADING_ZEROS	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	236
EXP	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	81
ExpansionUnitNumberToIOWordOffset_FP0	●	●	●	●	●	●												1129
ExpansionUnitNumberToIOWordOffset_FPX_FP0							●	●	●	●								1130
EXPT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	82
F_TRIG	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	313
F0_MV	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	806
F1_DMV	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	808
F10_BKMV	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	820
F10_BKMV_NUMBER	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	822
F10_BKMV_NUMBER_OFFSET	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	824
F10_BKMV_OFFSET	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	823
F100_SHR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	558
F101_SHL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	560
F102_DSHR			●	●	●	●	●	●	●	●			●	●	●	●	●	562
F103_DSHL			●	●	●	●	●	●	●	●			●	●	●	●	●	564
F105_BSR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	566
F106_BSL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	568
F108_BITR			●	●	●	●	●	●	●	●			●	●	●	●	●	570
F109_BITL			●	●	●	●	●	●	●	●			●	●	●	●	●	572
F11_COPY	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	825
F110_WSHR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	574
F111_WSHL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	576
F112_WBSR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	578
F113_WBSL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	580
F115_FIFT			●	●	●	●	●	●	●	●		●	●	●	●	●	●	484
F116_FIFR			●	●	●	●	●	●	●	●		●	●	●	●	●	●	488
F117_FIFW			●	●	●	●	●	●	●	●		●	●	●	●	●	●	492
F118_UDC	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	708
F119_LRSR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	582
F12_EPRD	●	●	●	●	●	●	●	●	●	●	●		●	●				827
F12_ICRD															●	●	●	829
F120_ROR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	586
F121_ROL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	588
F122_RCR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	590
F123_RCL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	592
F125_DROR			●	●	●	●	●	●	●	●			●	●	●	●	●	594

Instruction ● available ❍ partially available	FP0 2,7k / 5k	FP0 10k T32	FP0R C,F	FP0R 32k T32	FPΣ 12k, 16k	FPΣ 32k	FP-X CT, CP, C38A	FP-X L, C40RT0A	FP-X0 2,5k	FP-X0 8k	FP-e 2,7k	FP3, FP-C	FP2 16k	FP2 32k	FP2SH 32k	FP2SH 60k, 120k	FP10SH	page
F126_DROL			●	●	●	●	●	●	●	●			●	●	●	●	●	596
F127_DRCR			●	●	●	●	●	●	●	●			●	●	●	●	●	598
F128_DRCL			●	●	●	●	●	●	●	●			●	●	●	●	●	600
F13_ICWT															●	●	●	831
F130_BTS	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	538
F131_BTR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	539
F132_BTI	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	540
F133_BTT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	541
F135_BCU	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	543
F136_DBCU	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	544
F137_STMR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	934
F138_TIMEBCD_TO_SECBCD		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	866
F139_SECBCD_TO_TIMEBCD		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	867
F14_PGRD															●	●	●	833
F140_STC	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	998
F141_CLC	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	999
F142_WDT															●	●	●	1000
F143_IORF	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	843
F145_SEND			●	●		●	●			●		●	●	●	●	●	●	787
F145_WRITE_DATA			●	●		●	●			●								767
F145_WRITE_DATA_TYPE_OFFS			●	●		●	●			●								770
F145F146_MODBUS_COMMAND			●	●		●	●			●								778
F145F146_MODBUS_MASTER			●	●		●	●			●								780
F146_READ_DATA			●	●		●	●			●								773
F146_READ_DATA_TYPE_OFFS			●	●		●	●			●								775
F146_RECV			●	●		●	●			●		●	●	●	●	●	●	789
F147_PR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	845
F148_ERR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1001
F149_MSG	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1003
F15_XCH	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	836
F150_READ		●	●	●	●	●	●					●	●	●	●	●	●	848
F151_WRT		●	●	●	●	●	●					●	●	●	●	●	●	851
F152_RMRD												●	●	●	●	●	●	792
F153_RMWT												●	●	●	●	●	●	795
F155_SMPL			●	●		●	●			●		●		●	●	●	●	1004
F156_STRG			●	●		●	●			●		●		●	●	●	●	1005
F157_ADD_DTBCD_TIMEBCD		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	868
F158_SUB_DTBCD_TIMEBCD		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	870
F159_MTRN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	742
F159_MWRT_PARA													●	●	●	●		719
F16_DXCH	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	837
F160_DSQR			●	●	●	●	●	●	●	●		●	●	●	●	●	●	415
F161_MRCV	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	756
F161_MRD_PARA													●	●	●	●		728

Instruction ● available ❍ partially available	FP0 2,7k / 5k	FP0 10k T32	FP0R C,F	FP0R 32k T32	FPΣ 12k, 16k	FPΣ 32k	FP-X CT, CP, C38A	FP-X L, C40RT0A	FP-X0 2,5k	FP-X0 8k	FP-e 2,7k	FP3, FP-C	FP2 16k	FP2 32k	FP2SH 32k	FP2SH 60k, 120k	FP10SH	page
F161_MRD_STATUS													●	●	●	●		730
F165_HighSpeedCounter_Cam			●	●														895
F166_HighSpeedCounter_Set	●	●	●	●	●	●	●	●	●	●	●							901
F166_PulseOutput_Set			●	●														1027
F167_HighSpeedCounter_Reset	●	●	●	●	●	●	●	●	●	●	●							905
F167_PulseOutput_Reset			●	●														1030
F168_PulseOutput_Home	●	●									●							1036
F168_PulseOutput_Trapezoidal	●	●									●							1033
F169_PulseOutput_Jog	●	●									●							1040
F17_SWAP	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	839
F170_PulseOutput_PWM	●	●									●							1043
F171_PulseOutput_Home					●	●	●											1052
F171_PulseOutput_Jog_Positioning			●	●														1056
F171_PulseOutput_Trapezoidal Except FP-X 16k L14|FP-X 32k L30,L60			●	●	●	●	●	❍	●	●								1046
F172_PulseOutput_Jog Except FP-X 16k L14|FP-X 32k L30,L60			●	●	●	●	●	❍	●	●								1061
F173_PulseOutput_PWM Except FP-X 16k L14|FP-X 32k L30,L60			●	●	●	●	●	❍	●	●								1067
F174_PulseOutput_DataTable Except FP-X 16k L14|FP-X 32k L30,L60			●	●	●	●	●	❍	●	●								1070
F175_PulseOutput_Linear			●	●	●	●	●			●								1073
F176_PulseOutput_Center					●	●												1078
F176_PulseOutput_Pass					●	●												1082
F177_PulseOutput_Home Except FP-X 16k L14|FP-X 32k L30,L60			●	●				❍	●	●								1086
F178_HighSpeedCounter_Measure			●	●						●								909
F18_BXCH			●	●	●	●	●	●	●	●			●	●	●	●	●	841
F180_SCR_DUT											●							971
F181_DSP											●							975
F182_FILTER			●	●		●	●	●	●	●	●							551
F183_DSTM	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	935
F19_SJP												●	●	●	●	●	●	1011
F190_MV3			●	●	●	●	●	●	●	●			●	●	●	●	●	854
F191_DMV3			●	●	●	●	●	●	●	●			●	●	●	●	●	856
F2_MVN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	810
F20_ADD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	342

Instruction ● available ❍ partially available	FP0 2,7k / 5k	FP0 10k T32	FP0R C,F	FP0R 32k T32	FPΣ 12k, 16k	FPΣ 32k	FP-X CT, CP, C38A	FP-X L, C40RT0A	FP-X0 2,5k	FP-X0 8k	FP-e 2,7k	FP3, FP-C	FP2 16k	FP2 32k	FP2SH 32k	FP2SH 60k, 120k	FP10SH	page
F21_DADD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	344
F215_DAND			●	●	●	●	●	●	●	●			●	●	●	●	●	528
F216_DOR			●	●	●	●	●	●	●	●			●	●	●	●	●	530
F217_DXOR			●	●	●	●	●	●	●	●			●	●	●	●	●	532
F218_DXNR			●	●	●	●	●	●	●	●			●	●	●	●	●	534
F219_DUNI			●	●	●	●	●	●	●	●			●	●	●	●	●	536
F22_ADD2	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	346
F23_DADD2	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	348
F230_DTBCD_TO_SEC			●	●		●	●	●	●	●			●	●	●	●		872
F231_SEC_TO_DTBCD			●	●		●	●	●	●	●			●	●	●	●		873
F235_GRY			●	●	●	●	●	●	●	●			●	●	●	●	●	665
F236_DGRY			●	●	●	●	●	●	●	●			●	●	●	●	●	666
F237_GBIN			●	●	●	●	●	●	●	●			●	●	●	●	●	667
F238_DGBIN			●	●	●	●	●	●	●	●			●	●	●	●	●	668
F240_COLM			●	●	●	●	●	●	●	●			●	●	●	●	●	669
F241_LINE			●	●	●	●	●	●	●	●			●	●	●	●	●	671
F25_SUB	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	366
F250_BTOA			●	●		●	●	●	●	●								673
F251_ATOB			●	●		●	●	●	●	●								677
F252_ACHK			●	●		●	●	●	●	●								682
F26_DSUB	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	368
F27_SUB2	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	370
F270_MAX			●	●	●	●	●	●	●	●	●		●	●	●	●	●	451
F271_DMAX			●	●	●	●	●	●	●	●	●		●	●	●	●	●	453
F272_MIN			●	●	●	●	●	●	●	●	●		●	●	●	●	●	457
F273_DMIN			●	●	●	●	●	●	●	●	●		●	●	●	●	●	459
F275_MEAN			●	●	●	●	●	●	●	●	●		●	●	●	●	●	463
F276_DMEAN			●	●	●	●	●	●	●	●	●		●	●	●	●	●	465
F277_SORT			●	●	●	●	●	●	●	●	●		●	●	●	●	●	501
F278_DSORT			●	●	●	●	●	●	●	●	●		●	●	●	●	●	503
F28_DSUB2	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	372
F282_SCAL			●	●	●	●	●	●	●	●	●		●	●	●	●	●	469
F283_DSCAL			●	●	●	●	●	●	●	●	●		●	●	●	●	●	472
F284_RAMP			●	●		●	●	●	●	●								475
F285_LIMT			●	●	●	●	●	●	●	●			●	●	●	●	●	878
F286_DLIMT			●	●	●	●	●	●	●	●			●	●	●	●	●	880
F287_BAND			●	●	●	●	●	●	●	●	●		●	●	●	●	●	433
F288_DBAND			●	●	●	●	●	●	●	●	●		●	●	●	●	●	435
F289_ZONE			●	●	●	●	●	●	●	●	●		●	●	●	●	●	440
F290_DZONE			●	●	●	●	●	●	●	●	●		●	●	●	●	●	442
F3_DMVN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	812
F30_MUL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	390
F300_BSIN													●	●	●	●	●	417
F301_BCOS													●	●	●	●	●	419

Instruction ● available ❍ partially available	FP0 2,7k / 5k	FP0 10k T32	FP0R C,F	FP0R 32k T32	FPΣ 12k, 16k	FPΣ 32k	FP-X CT, CP, C38A	FP-X L, C40RT0A	FP-X0 2,5k	FP-X0 8k	FP-e 2,7k	FP3, FP-C	FP2 16k	FP2 32k	FP2SH 32k	FP2SH 60k, 120k	FP10SH	page
F302_BTAN													●	●	●	●	●	421
F303_BASIN													●	●	●	●	●	423
F304_BACOS													●	●	●	●	●	425
F305_BATAN													●	●	●	●	●	427
F309_FMV	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	858
F31_DMUL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	392
F310_FADD	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F311_FSUB	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F312_FMUL	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F313_FDIV	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	410
F314_SIN	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F315_COS	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F316_TAN	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F317_ASIN	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F318_ACOS	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F319_ATAN	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F32_DIV	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	402
F320_LN	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F321_EXP	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F322_LOG	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F323_PWR	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F324_FSQR	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F325_FLT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	684
F326_DFLT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	685
F327_INT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	687
F328_DINT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	689
F329_FIX	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F33_DDIV	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	404
F330_DFIX	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F331_ROFF	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F332_DROFF	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F333_FINT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	691
F334_FRINT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	693
F335_FSIGN	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	695
F336_FABS	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	1246
F337_RAD	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	697
F338_DEG	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	699
F34_MULW			●	●	●	●	●	●	●	●			●	●	●	●	●	394
F345_FCMP			●	●	●	●	●	●	●	●			●	●	●	●	●	1246
F346_FWIN			●	●	●	●	●	●	●	●			●	●	●	●	●	614
F347_FLIMT			●	●	●	●	●	●	●	●			●	●	●	●	●	1246
F348_FBAND			●	●	●	●	●	●	●	●			●	●	●	●	●	437
F349_FZONE			●	●	●	●	●	●	●	●			●	●	●	●	●	444

Instruction ● available ❍ partially available	FP0 2,7k / 5k	FP0 10k T32	FP0R C,F	FP0R 32k T32	FPΣ 12k, 16k	FPΣ 32k	FP-X CT, CP, C38A	FP-X L, C40RT0A	FP-X0 2,5k	FP-X0 8k	FP-e 2,7k	FP3, FP-C	FP2 16k	FP2 32k	FP2SH 32k	FP2SH 60k, 120k	FP10SH	page
F35_INC	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	358
F350_FMAX													●	●	●	●	●	455
F351_FMIN													●	●	●	●	●	461
F352_FMEAN													●	●	●	●	●	467
F353_FSORT													●	●	●	●	●	505
F354_FSCAL			●	●		●	●	●	●	●			●	●	●	●	●	478
F355_PID_DUT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	942
F356_PID_PWM Except FP-X 16k L14|FP-X 32k L30,L60			●	●		●	●	❍	●	●								945
F36_DINC	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	360
F37_DEC	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	382
F373_DTR			●	●	●	●	●	●	●	●			●	●	●	●	●	616
F374_DDTR			●	●	●	●	●	●	●	●			●	●	●	●	●	618
F38_DDEC	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	384
F39_DMULD			●	●	●	●	●	●	●	●			●	●	●	●	●	396
F4_GETS													●	●	●	●		814
F40_BADD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	350
F41_DBADD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	352
F410_SET_INDEXREG_BANK															●	●	●	
F411_CHANGE_INDEXREG_BANK															●	●	●	
F412_RESTORE_INDEXREG_BANK															●	●	●	
F414_SET_FILEREG_BANK																●		
F415_CHANGE_FILEREG_BANK																●		
F416_RESTORE_FILEREG_BANK																●		
F42_BADD2	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	354
F43_DBADD2	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	356
F45_BSUB	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	374
F46_DBSUB	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	376
F47_BSUB2	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	378
F48_DBSUB2	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	380
F5_BTM	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	512
F50_BMUL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	398
F51_DBMUL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	400
F52_BDIV	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	406
F53_DBDIV	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	408
F55_BINC	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	362
F56_DBINC	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	364
F57_BDEC	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	386
F58_DBDEC	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	388
F6_DGT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	514
F60_CMP	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	604
F61_DCMP	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	606
F62_WIN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	608

Instruction ● available ❍ partially available	FP0 2,7k / 5k	FP0 10k T32	FP0R C,F	FP0R 32k T32	FPΣ 12k, 16k	FPΣ 32k	FP-X CT, CP, C38A	FP-X L, C40RT0A	FP-X0 2,5k	FP-X0 8k	FP-e 2,7k	FP3, FP-C	FP2 16k	FP2 32k	FP2SH 32k	FP2SH 60k, 120k	FP10SH	page
F63_DWIN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	610
F64_BCMP	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	612
F65_WAN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	518
F66_WOR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	520
F67_XOR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	522
F68_XNR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	524
F69_WUNI			●	●	●	●	●	●	●	●			●	●	●	●	●	526
F7_MV2			●	●	●	●	●	●	●	●			●	●	●	●	●	816
F70_BCC	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	412
F71_HEX2A	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	622
F72_A2HEX	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	625
F73_BCD2A	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	628
F74_A2BCD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	631
F75_BIN2A	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	635
F76_A2BIN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	638
F77_DBIN2A	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	641
F78_DA2BIN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	644
F8_DMV2			●	●	●	●	●	●	●	●			●	●	●	●	●	818
F80_BCD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	647
F81_BIN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	649
F82_DBCD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	651
F83_DBIN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	653
F84_INV	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	545
F85_NEG	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	447
F86_DNEG	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	449
F87_ABS	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	429
F88_DABS	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	431
F89_EXT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	655
F90_DECO	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	657
F91_SEGT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	659
F92_ENCO	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	660
F93_UNIT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	547
F94_DIST	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	549
F95_ASC	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	662
F96_SRC	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	479
F97_DSRC			●	●	●	●	●	●	●	●			●	●	●	●	●	481
F98_CMPR			●	●	●	●	●	●	●	●		●	●	●	●	●	●	496
F99_CMPW			●	●	●	●	●	●	●	●		●	●	●	●	●	●	499
FIND	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	274
FNS_InitConfigDataTable					●	●							●	●	●	●		799
FNS_InitConfigNameTable					●	●							●	●	●	●		802
GET_RTC_DT		●		●	●	●	●			●	●	●	●	●	●	●	●	290
GET_RTC_DTBCD		●		●	●	●	●			●	●	●	●	●	●	●	●	874

Instruction ● available ❍ partially available	FP0 2,7k / 5k	FP0 10k T32	FP0R C,F	FP0R 32k T32	FPΣ 12k, 16k	FPΣ 32k	FP-X CT, CP, C38A	FP-X L, C40RT0A	FP-X0 2,5k	FP-X0 8k	FP-e 2,7k	FP3, FP-C	FP2 16k	FP2 32k	FP2SH 32k	FP2SH 60k, 120k	FP10SH	page
GET_RTC_INT		●		●	●	●	●			●	●	●	●	●	●	●	●	1246
GetPointer	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
GT_ActivateScreen	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1134
GT_ChangeBacklightBrightness	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1136
Hsc_TargetValueMatch_Reset	●	●	●	●	●	●	●	●	●	●	●							1167
Hsc_TargetValueMatch_Set	●	●	●	●	●	●	●	●	●	●	●							1169
HscControl_CountingDisable	●	●	●	●	●	●	●	●	●	●	●							1140
HscControl_CountingEnable	●	●	●	●	●	●	●	●	●	●	●							1142
HscControl_ElapsedValueContinue	●	●	●	●	●	●	●	●	●	●	●							1144
HscControl_ElapsedValueReset	●	●	●	●	●	●	●	●	●	●	●							1146
HscControl_HscInstructionClear	●	●	●	●	●	●	●	●	●	●	●							1148
HscControl_ResetInputDisable	●	●	●	●	●	●	●	●	●	●	●							1150
HscControl_ResetInputEnable	●	●	●	●	●	●	●	●	●	●	●							1151
HscControl_SetDefaults	●	●	●	●	●	●	●	●	●	●	●							1152
HscControl_WriteElapsedValue	●	●	●	●	●	●	●	●	●	●	●							1153
HscInfo_GetControlCode	●	●	●	●	●	●	●	●	●	●	●							1156
HscInfo_GetCurrentSpeed	●	●	●	●	●	●	●	●	●	●	●							1157
HscInfo_IsActive	●	●	●	●	●	●	●	●	●	●	●							1158
HscInfo_IsChannelEnabled	●	●	●	●	●	●	●	●	●	●	●							1159
HscInfo_IsCountingDisabled	●	●	●	●	●	●	●	●	●	●	●							1160
HscInfo_IsElapsedValueReset	●	●	●	●	●	●	●	●	●	●	●							1161
HscInfo_IsResetInputDisabled	●	●	●	●	●	●	●	●	●	●	●							1162
HscInfo_ReadElapsedValue	●	●	●	●	●	●	●	●	●	●	●							1164
HscInfo_ReadTargetValue	●	●	●	●	●	●	●	●	●	●	●							1165
ICTL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1017
INSERT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	276
INT_TO_BCD_WORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	244
INT_TO_BOOL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	116
INT_TO_DINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	172
INT_TO_DWORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	137
INT_TO_REAL	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	194
INT_TO_STRING	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	218
INT_TO_STRING_LEADING_ZEROS	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	220
INT_TO_TIME	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	202
INT_TO_UDINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	184
INT_TO_UINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	161
INT_TO_WORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	125
IPADDR_TO_STRING	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	233
IPADDR_TO_STRING_NO_LEADING_ZEROS	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	234
Is_AreaDT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
Is_AreaFL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
IS_VALID_DATE_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	291
IS_VALID_DT_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	292

Instruction ● available ❍ partially available	FP0 2,7k / 5k	FP0 10k T32	FP0R C,F	FP0R 32k T32	FPΣ 12k, 16k	FPΣ 32k	FP-X CT, CP, C38A	FP-X L, C40RT0A	FP-X0 2,5k	FP-X0 8k	FP-e 2,7k	FP3, FP-C	FP2 16k	FP2 32k	FP2SH 32k	FP2SH 60k, 120k	FP10SH	page
IS_VALID_TOD_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	294
IsAnyHscChannelEnabled	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
IsAnyPulseChannelEnabled	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
IsClockCalendarSupported	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
IsCommunicationError	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	765
IsDataUnitTypeSupported	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
IsF145F146Error			●	●		●	●			●		●	●	●	●	●	●	783
IsF145F146NotActive			●	●		●	●			●		●	●	●	●	●	●	782
IsFileRegisterAreaSupported	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
IsHscChannelEnabled	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
IsInstructionSupported	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
IsPlcLink	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	725
IsProgramControlled	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	726
IsPulseChannelEnabled	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
IsReceptionDone	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	761
IsReceptionDoneByTimeOut	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	762
IsSystemVariableSupported	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
IsTransmissionDone	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	764
JP	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1010
KEEP	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	508
LBL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1014
LEFT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	264
LEN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	262
LIMIT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	86
LN	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	79
LOG	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	80
LOOP	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1013
LSR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	556
MAX	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	254
MC	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1008
MCE	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1009
MID	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	268
MIN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	255
MOD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	68
MOVE	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	60
MUL_TIME_DINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	336
MUL_TIME_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	335
MUL_TIME_REAL	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	337
MUX	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	256
NOT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
NSTL_NEXT_STEP	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
NSTP_NEXT_STEP_PULSE	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
OutputCompilerError	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331

Instruction ● available ❍ partially available	FP0 2,7k / 5k	FP0 10k T32	FP0R C,F	FP0R 32k T32	FPΣ 12k, 16k	FPΣ 32k	FP-X CT, CP, C38A	FP-X L, C40RT0A	FP-X0 2,5k	FP-X0 8k	FP-e 2,7k	FP3, FP-C	FP2 16k	FP2 32k	FP2SH 32k	FP2SH 60k, 120k	FP10SH	page
OutputCompilerWarning	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
P13_EPWT	●	●	●	●	●	●	●	●	●	●	●		●	●				834
P13_ICWT															●	●	●	1331
PID_FB	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	952
PID_FB_DUT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	954
Pulse_TargetValueMatch_Reset			●	●														1238
Pulse_TargetValueMatch_Set			●	●														1241
PulseControl_CountingDisable	●	●	●	●	●	●	●	●	●	●	●							1331
PulseControl_CountingEnable	●	●	●	●	●	●	●	●	●	●	●							1201
PulseControl_DeceleratedStop			●	●														1203
PulseControl_ElapsedValueContinue	●	●	●	●	●	●	●	●	●	●	●							1205
PulseControl_ElapsedValueReset	●	●	●	●	●	●	●	●	●	●	●							1207
PulseControl_JogPositionControl			●	●														1209
PulseControl_NearHome	●	●	●	●	●	●	●	●	●	●	●							1210
PulseControl_PulseOutputContinue	●	●	●	●	●	●	●	●	●	●	●							1212
PulseControl_PulseOutputStop	●	●	●	●	●	●	●	●	●	●	●							1214
PulseControl_SetDefaults	●	●	●	●	●	●	●	●	●	●	●							1216
PulseControl_TargetValueMatchClear			●	●														1331
PulseControl_WriteElapsedValue	●	●	●	●	●	●	●	●	●	●	●							1217
PulseInfo_GetControlCode	●	●	●	●	●	●	●	●	●	●	●							1221
PulseInfo_GetCurrentSpeed	●	●	●	●	●	●	●	●	●	●	●							1222
PulseInfo_IsActive	●	●	●	●	●	●	●	●	●	●	●							1224
PulseInfo_IsChannelEnabled	●	●	●	●	●	●	●	●	●	●	●							1225
PulseInfo_IsCountingDisabled	●	●	●	●	●	●	●	●	●	●	●							1226
PulseInfo_IsElapsedValueReset	●	●	●	●	●	●	●	●	●	●	●							1227
PulseInfo_IsHomeInputTrue	●	●	●	●	●	●	●	●	●	●	●							1228
PulseInfo_IsPulseOutputStopped	●	●	●	●	●	●	●	●	●	●	●							1229
PulseInfo_IsTargetValueMatchActive			●	●														1230
PulseInfo_ReadAccelerationForbiddenAreaStartingPosition			●	●														1231
PulseInfo_ReadCorrectedFinalSpeed			●	●					●	●								1232
PulseInfo_ReadCorrectedInitialSpeed			●	●					●	●								1233
PulseInfo_ReadElapsedValue	●	●	●	●	●	●	●	●	●	●	●							1234
PulseInfo_ReadTargetValue	●	●	●	●	●	●	●	●	●	●	●							1235
PulseInfo_ReadTargetValueMatchValue			●	●														1236
PulseOutput_Center_FB					●	●												1174
PulseOutput_Home_FB	●	●	●	●	●	●	●	●	●	●	●							1177
PulseOutput_Jog_FB	●	●	●	●	●	●	●	●	●	●	●							1180
PulseOutput_Jog_Positioning0_FB			●	●														1182
PulseOutput_Jog_Positioning1_FB			●	●														1185
PulseOutput_Jog_TargetValue_FB Except FP-X 16k L14|FP-X 32k L30,L60			●	●	●	●	●	❍	●	●								1187

Instruction ● available ❍ partially available	FP0 2,7k / 5k	FP0 10k T32	FP0R C,F	FP0R 32k T32	FPΣ 12k, 16k	FPΣ 32k	FP-X CT, CP, C38A	FP-X L, C40RT0A	FP-X0 2,5k	FP-X0 8k	FP-e 2,7k	FP3, FP-C	FP2 16k	FP2 32k	FP2SH 32k	FP2SH 60k, 120k	FP10SH	page
PulseOutput_Linear_FB			●	●	●	●	●			●								1189
PulseOutput_Pass_FB					●	●												1192
PulseOutput_Trapezoidal_FB	●	●	●	●	●	●	●	●	●	●	●							1195
R_TRIG	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	312
ReadDataFromFileRegisterBank																●		861
REAL_TO_DINT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	175
REAL_TO_DWORD	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	141
REAL_TO_INT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	152
REAL_TO_STRING	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	229
REAL_TO_TIME	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	204
REAL_TO_UDINT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	187
REAL_TO_UINT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	164
ReceiveCharacters	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	753
ReceiveData	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	751
REPLACE	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	278
RIGHT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	266
ROL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	100
ROR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	98
RS	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	308
RST	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	509
SCALE_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	956
SCALE_INT_UINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	958
SCALE_REAL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	960
SCALE_UINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	962
SCALE_UINT_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	964
SCLR_CLEAR_MULTIPLE_STEPS			●	●	●	●	●	●	●	●			●	●	●	●	●	1331
SEL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	258
SendCharacters	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	738
SendCharactersAndClearString	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	740
SET	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	509
SET_RTC_DT		●		●	●	●	●			●	●	●	●	●	●	●	●	295
SET_RTC_DTBCD		●		●	●	●	●			●	●	●	●	●	●	●	●	875
SET_RTC_INT		●		●	●	●	●			●	●	●	●	●	●	●	●	1246
SetCommunicationMode	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	718
SfcOutputsReset	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
SfcRunning	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
SfcStopped	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
SfcTransitionsInhibited	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
SHL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	96
SHR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	94
SIN	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	70
Size_Of_Var	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
SmoothSignal_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	966

Instruction ● available ❍ partially available	FP0 2,7k / 5k	FP0 10k T32	FP0R C,F	FP0R 32k T32	FPΣ 12k, 16k	FPΣ 32k	FP-X CT, CP, C38A	FP-X L, C40RT0A	FP-X0 2,5k	FP-X0 8k	FP-e 2,7k	FP3, FP-C	FP2 16k	FP2 32k	FP2SH 32k	FP2SH 60k, 120k	FP10SH	page
SmoothSignal_REAL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	967
SmoothSignal_UINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	968
SPLIT_DATE_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	296
SPLIT_DT_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	297
SPLIT_TIME_INT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	338
SPLIT_TOD_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	299
SQRT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	69
SR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	306
SSTP_STEP_START	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1331
StartStopAllSfcs	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
StartStopAllSfcsAndInitData	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
StartStopSfc	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
StartStopSfcAndInitData	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
STPE_STEP_LADDER_END	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
STRING_TO_DINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	178
STRING_TO_DINT_STEPSAVER	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	179
STRING_TO_DWORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	143
STRING_TO_DWORD_STEPSAVER	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	144
STRING_TO_ETLANADDR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	250
STRING_TO_ETLANADDR_STEPSAVER	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	251
STRING_TO_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	155
STRING_TO_INT_STEPSAVER	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	156
STRING_TO_IPADDR	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	248
STRING_TO_IPADDR_STEPSAVER	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	249
STRING_TO_REAL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	199
STRING_TO_UDINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	189
STRING_TO_UINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	166
STRING_TO_UINT_STEPSAVER	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	167
STRING_TO_WORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	130
STRING_TO_WORD_STEPSAVER	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	131
SUB_DATE_DATE	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	300
SUB_DT_DT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	301
SUB_DT_TIME	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	302
SUB_TIME	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	339
SUB_TOD_TIME	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	303
SUB_TOD_TOD	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	304
SYS1			●	●	●	●	●	●	●	●								991
SYS2			●	●	●	●	●			●								994
TAN	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	75
TIME_TO_DINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	177
TIME_TO_DWORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	142
TIME_TO_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	154
TIME_TO_REAL	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	198
TIME_TO_STRING			●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	231

Instruction ● available ❍ partially available	FP0 2,7k / 5k	FP0 10k T32	FP0R C,F	FP0R 32k T32	FPΣ 12k, 16k	FPΣ 32k	FP-X CT, CP, C38A	FP-X L, C40RT0A	FP-X0 2,5k	FP-X0 8k	FP-e 2,7k	FP3, FP-C	FP2 16k	FP2 32k	FP2SH 32k	FP2SH 60k, 120k	FP10SH	page
TIME_TO_WORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	129
TM_100ms	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	930
TM_100ms_FB	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	920
TM_10ms	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	928
TM_10ms_FB	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	917
TM_1ms	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	926
TM_1ms_FB	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	914
TM_1s	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	932
TM_1s_FB	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	923
TOD_TO_STRING	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	237
TOD_TO_UDINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	192
TOF	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	324
TON	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	326
TP	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	328
TRUNC_TO_DINT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	176
TRUNC_TO_INT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	153
TRUNC_TO_UDINT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	188
TRUNC_TO_UINT	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	165
UDINT_TO_BCD_DWORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	247
UDINT_TO_BOOL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	119
UDINT_TO_DATE	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	207
UDINT_TO_DINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	174
UDINT_TO_DT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	205
UDINT_TO_DWORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	140
UDINT_TO_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	151
UDINT_TO_REAL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	197
UDINT_TO_STRING	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	223
UDINT_TO_STRING_LEADING_ZEROS	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	226
UDINT_TO_TOD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	209
UDINT_TO_UINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	163
UDINT_TO_WORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	128
UINT_TO_BCD_WORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	246
UINT_TO_BOOL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	118
UINT_TO_DINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	173
UINT_TO_DWORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	139
UINT_TO_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	150
UINT_TO_REAL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	196
UINT_TO_STRING	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	227
UINT_TO_STRING_LEADING_ZEROS	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	228
UINT_TO_UDINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	185
UINT_TO_WORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	127
Unit_AnalogInOut_FP0_A21	●	●	●	●	●	●	●	●	●	●								1092
Unit_AnalogInOut_FPG_A44					●	●												1119

Instruction ● available ❍ partially available	FP0 2,7k / 5k	FP0 10k T32	FP0R C,F	FP0R 32k T32	FPΣ 12k, 16k	FPΣ 32k	FP-X CT, CP, C38A	FP-X L, C40RT0A	FP-X0 2,5k	FP-X0 8k	FP-e 2,7k	FP3, FP-C	FP2 16k	FP2 32k	FP2SH 32k	FP2SH 60k, 120k	FP10SH	page
Unit_AnalogInput_FP0_A80	●	●	●	●	●	●	●	●	●	●								1097
Unit_AnalogInput_FP0_RTD_INT	●	●	●	●	●	●	●	●	●	●								1102
Unit_AnalogInput_FP0_RTD_REAL	●	●	●	●	●	●	●	●	●	●								1108
Unit_AnalogInput_FP0_TC4_TC8	●	●	●	●	●	●	●	●	●	●								1114
Unit_AnalogOutput_FP0_A04I	●	●	●	●	●	●	●	●	●	●								1123
Unit_AnalogOutput_FP0_A04V	●	●	●	●	●	●	●	●	●	●								1126
Var_ToAreaOffs	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	1246
WITHIN_LIMITS	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	112
WORD_BCD_TO_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	147
WORD_BCD_TO_UINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	159
WORD_TO_BOOL	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	114
WORD_TO_BOOL16	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	238
WORD_TO_BOOLS	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	240
WORD_TO_DINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	169
WORD_TO_DWORD	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	136
WORD_TO_INT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	146
WORD_TO_STRING	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	212
WORD_TO_TIME	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	200
WORD_TO_UDINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	181
WORD_TO_UINT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	158
WriteDataToFileRegisterBank																●		863

Record of Changes

Manual No.	Date	Description of Changes
ACGM0313V2EN	July 2012	Complete update in accordance with software version 6.4. For details on the new information, see the section new in this version 6.4 in the online help.
ACGM0313V1EN	FEB. 2011	First edition

Index