ATC Function Library

User Manual

April, 2026 V1.0

※ Contents

※ Preface

This manual provides detailed introduction for the function blocks included in the ATC library. A thorough understanding of the relevant functions, operating methods, etc., is required for the correct use of the function modules.

Furthermore, please store this manual in an easily accessible location after reading.

Targe reader

This manual enables users of the HCFA ATC library function blocks to gain a comprehensive understanding of the function blocks and their operating methods within the library, as well as to master their application.

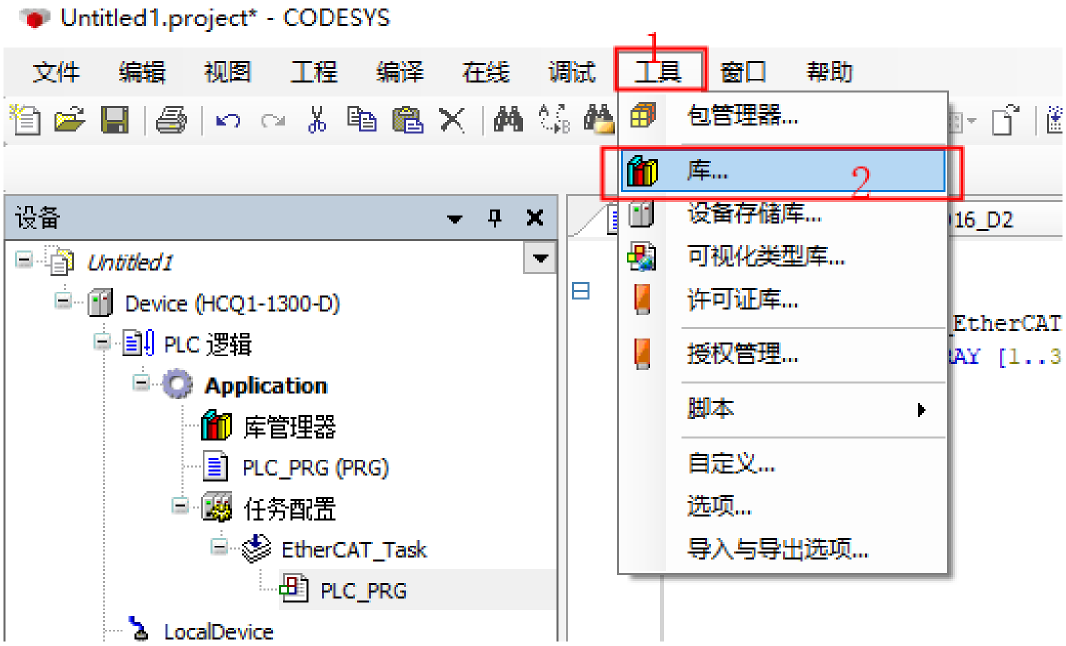
Update record

Version	Update time	Update Content
V0.9	20240320	Synchronization with the functional descriptions of the HCFA_ATCLib_1.15.14.compiled-library
V1.0	20240402	Synchronization with the functional descriptions of the HCFA_ATCLib_1.15.16.compiled-library

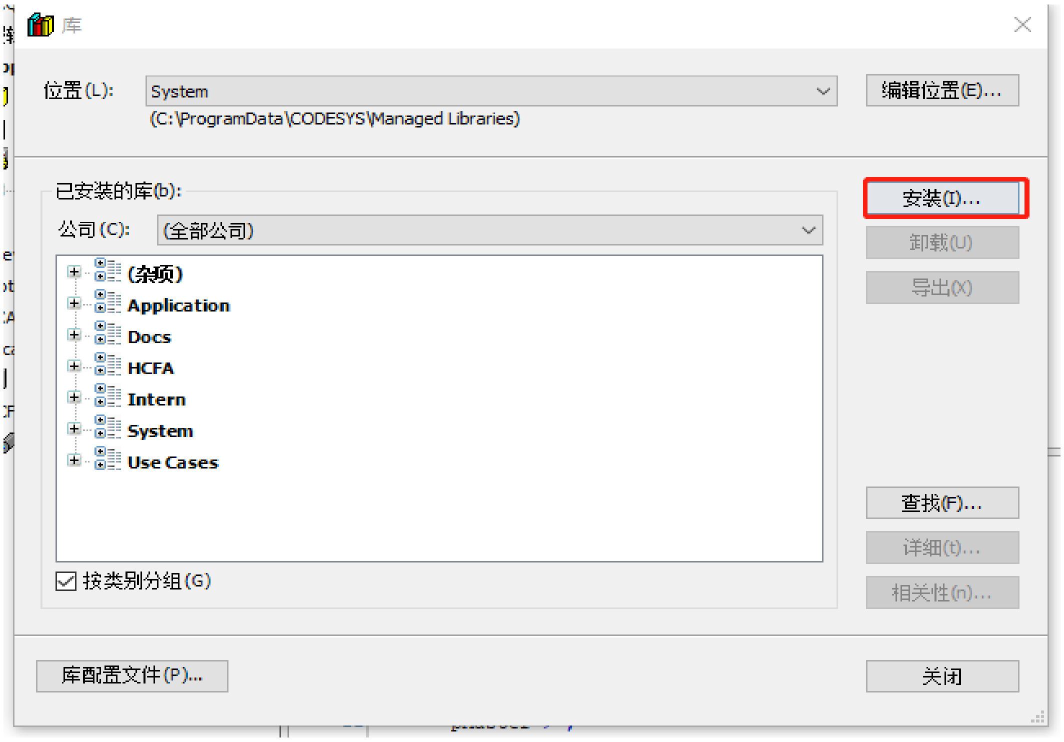
Chapter 1 Library File Addition

This manual provides a detailed explanation of the functions included in the HCFA_ATCLib_1.8.5 library. Before using the functions contained in the library file, the corresponding library file must be installed in CODESYS by following the steps below.

【1】In CODESYS, click [Tools]->[Library].

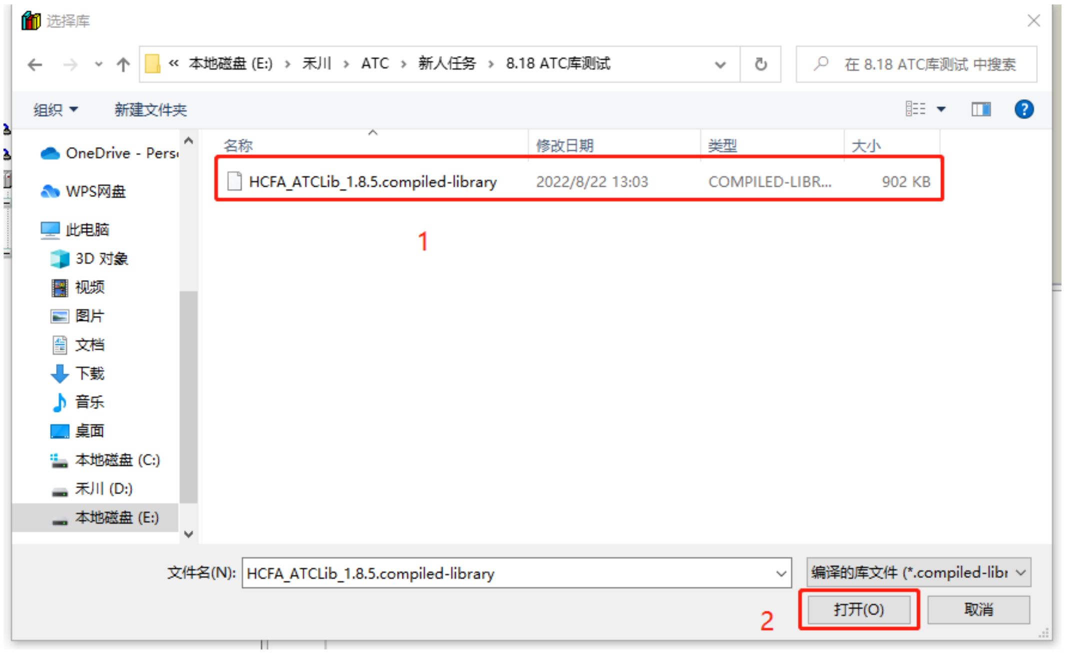


【2】In the pop-up "Library" window, click [Install].

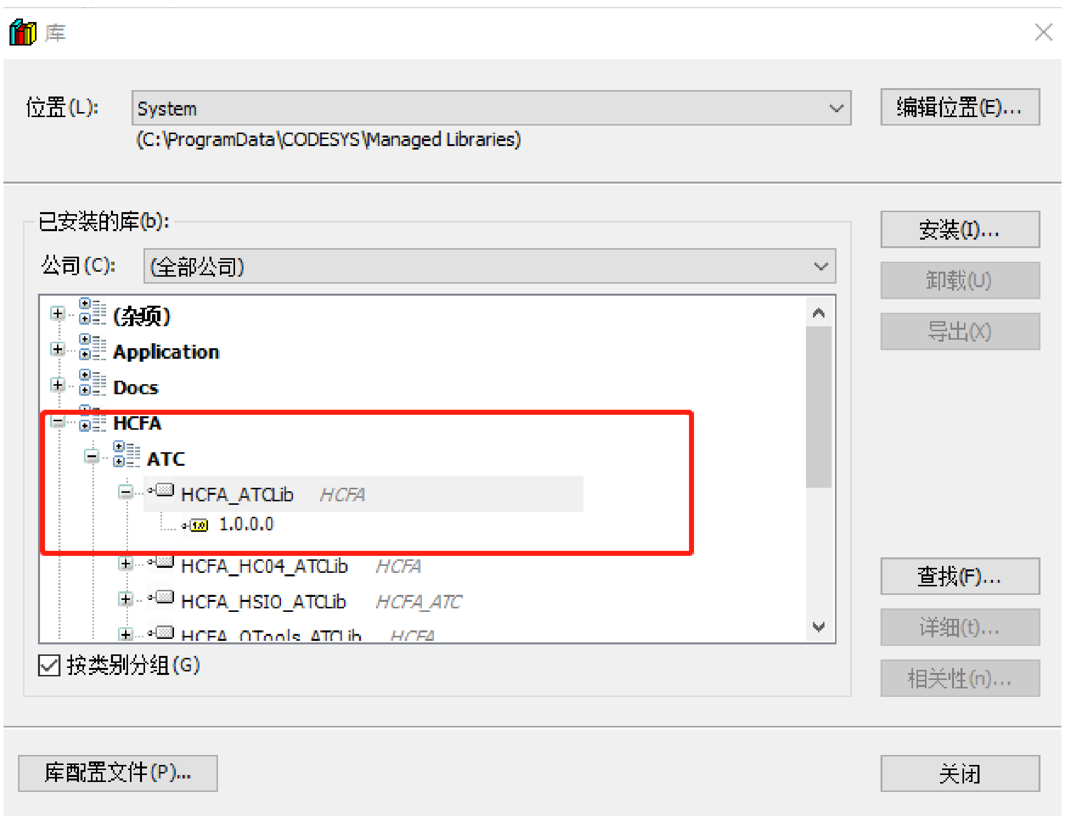


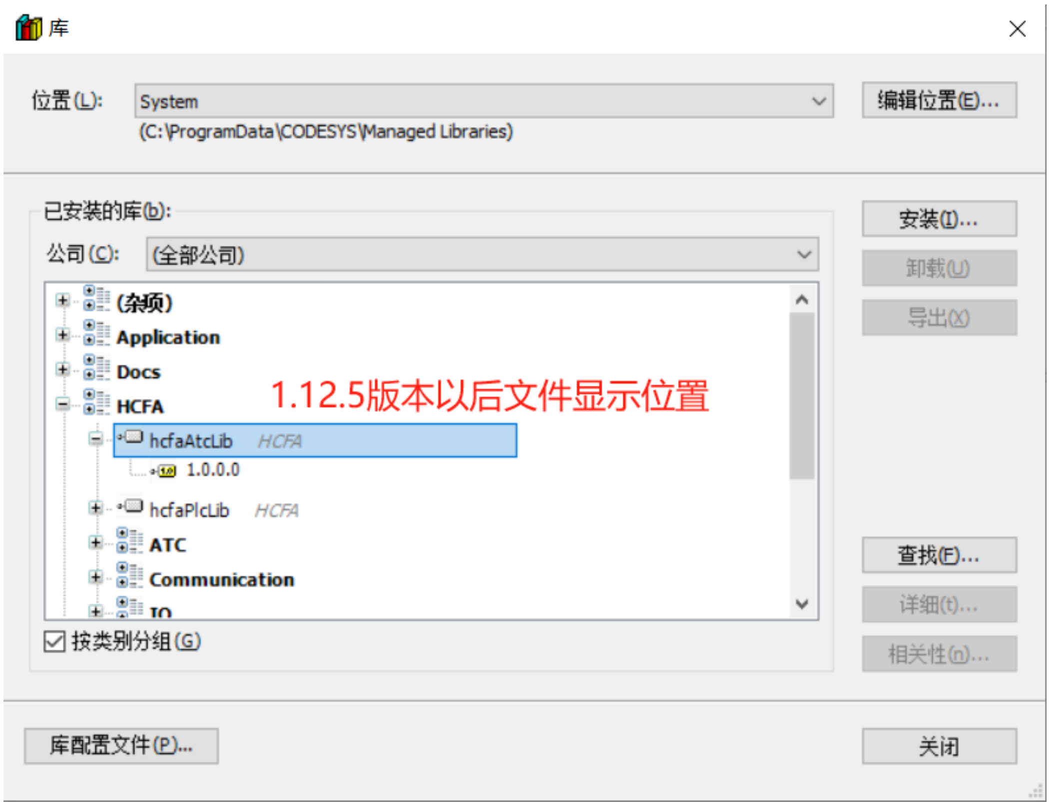
A • Library File Addition

【3】In the pop-up system dialog, select the corresponding version of the library file (here, HCFA_ATCLib_1.8.5) and click the [Open] to install the library file.

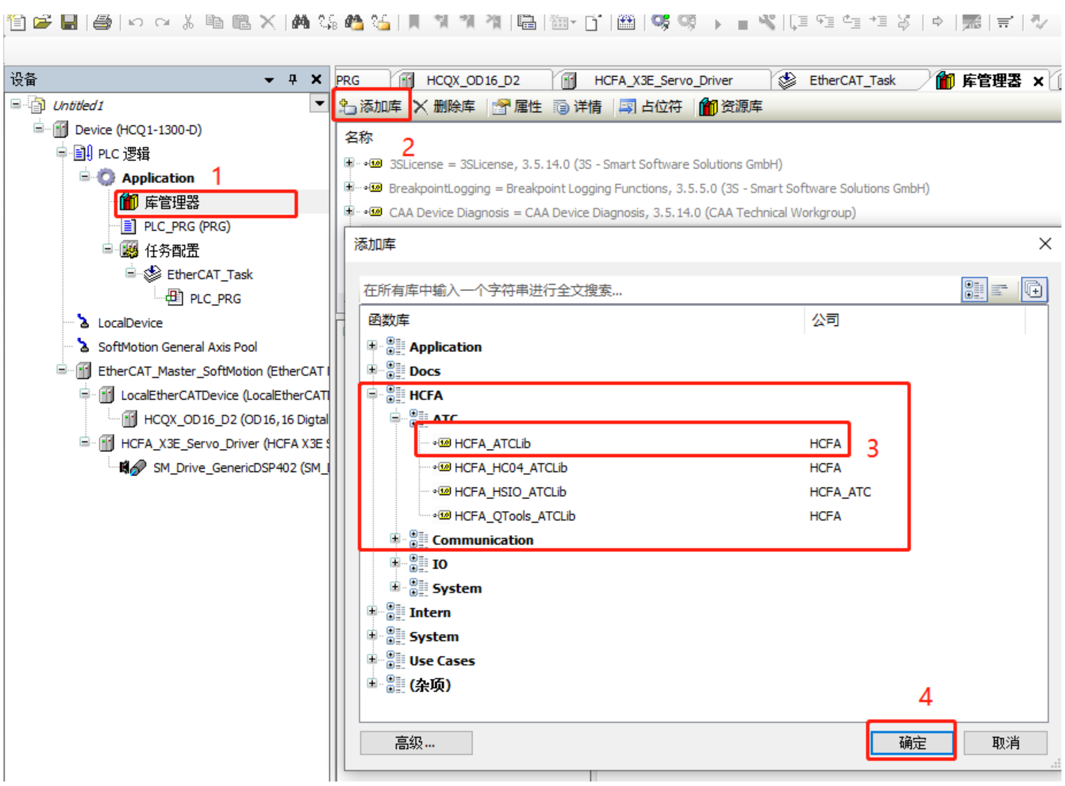


【4】After successful installation, the HCFA_ATCLib library with version number 1.0.0.0 can be seen in the "Library" window under [HCFA]->[ATC]. For library versions after 1.12.5, the file location is changed to hcfaAtcLib under [HCFA].

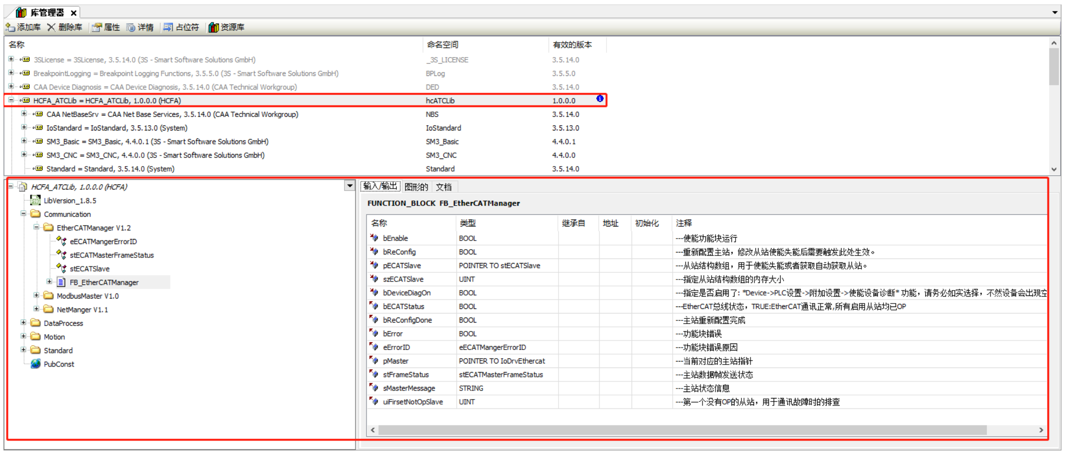




【5】In the CODESYS interface, double-click [Library Manager]->[Add Library] in the device tree. In the pop-up "Add Library" window, enable the [Show Advanced Libraries] option, then click [HCFA]->[ATC]->[HCFA_ATCLib] sequentially to select the newly installed library, and click "OK" to complete the addition. For library versions after 1.12.5, the addition location changes to directly under the [HCFA] path as described in step[4].



【6】After the addition is completed, it can be seen in the Library Manager that the library HCFA_ATCLib 1.0.0.0 (renamed to hcfaAtcLib 1.0.0.0 after 1.12.5) has been added. Clicking on it allows viewing of the detailed version and brief descriptions of each function block.



Chapter 2 Communication (Communication Management)

2.1 FB_EtherCATManager (FB)

This function block implements EtherCAT master network management. It controls the enable/disable status of slave stations and includes some basic diagnostic functions.

◆ Variables

<table>
<tr><th>Name</th><th colspan="3">FB_EtherCATManager (Communication management)</th></tr>
<tr><td>Supported modes</td><td>CSP</td><td>CSV</td><td>CST</td></tr>
<tr><td colspan="2">Graphical representation</td><td colspan="2">ST representation</td></tr>
<tr><td colspan="2">
</td><td colspan="2">FB_EtherCATManager(
bEnable:= ,
bReConfig:= ,
pECATSlave:= ,
szECATSlave:= ,
bDeviceDiagOn:= ,
bECATStatus=> ,
bAllSlaveOp=> ,
bReConfigDone=> ,
bError=> ,
eErrorID=> ,
pMaster=> ,
stFrameStatus=> ,
sMasterMessage=> ,
uiFirstNotOpSlave=>);</td></tr>
</table>

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bEnable	Enable	BOOL	TRUE, FALSE	FALSE	TRUE: Enables the function block; FALSE: Disables the function block.
bReConfig	Reconfigure master	BOOL	TRUE, FALSE	FALSE	TRUE: Reconfigures the master station.
pECATSlave	Slave structure array	POINT TO stECATSlave			Used to enable/disable slaves or automatically acquire slave pointers.
szECATSlave	Memory size	UINT		0	Specifies the memory size of the slave array structure.
bDeviceDiagOn	Diagnostic function	BOOL	TRUE, FALSE	FALSE	Specifies whether to enable the diagnostic function.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
bECATStatus	EtherCAT bus status	BOOL	TRUE, FALSE	TRUE: EtherCAT communication is normal; DC clock synchronization is successful.
bAllSlaveOp	All slaves OP status	BOOL	TRUE, FALSE	TRUE: All slave devices have entered the OP state.
bReConfigDone	Master reconfiguration done	BOOL	TRUE, FALSE	TRUE: Master station reconfiguration is complete.
bError	Error	BOOL	TRUE, FALSE	TRUE: An error occurred in the function block; execution has stopped.
ErrorID	Error ID	SMC_ERROR	0	Outputs the corresponding fault code when an error occurs in the function block.

pMaster	Master	POINTER TO IoDrivEthercat		Pointer to the current corresponding master station.
stFrameStatus	Transmission status	stECATMasterFrameStatus		Master station data frame transmission status.
sMasterMessage	Message	STRING		Master station status information.
uiFirsetNotOpSlave	Faulty slave ID	UINT	0	The first slave not in OP state, used for troubleshooting communication faults.

(3) Transition timing of output variables

Variable	Becomes TRUE when	Becomes FALSE when
bECATStatus	• ECAT bus communication is normal, and DC clock synchronization is complete.	• bError becomes TRUE. • uiFirstNotOpSlave outputs the faulty slave ID.
bAllSlaveOp	• All slave stations enter the OP state.	• The master station is reconfigured.
bReConfigDone	• Master station reconfiguration is complete.	• The master station is reconfigured.
bError	• A fault occurs during function block execution.	• The fault condition is cleared.

◆ Data types

(4) Slave structure

Name	Data type	Valid range	Initial value	Description
bDisableSlave	BOOL	TRUE, FALSE	FALSE	TRUE: Disables this slave.
pSlave	POINT TO stECATSlave			Automatically acquired slave pointer.
PAxis	POINT TO AXIS_REF_SM3			Valid when diagnostics are enabled.

◆ Key points

• After bEnable is set to TRUE, FB_EtherCATManager enters the execution state, and its other input pins are then processed by FB_EtherCATManager.

• After Enable is set to FALSE, FB_EtherCATManager will no longer execute any function block logic. Modifying its other input pins at this time will have no effect.

• The bReConfig pin controls master reconfiguration. After modifying the enable/disable settings in the slave structure(s), this pin must be triggered to make the changes effective.

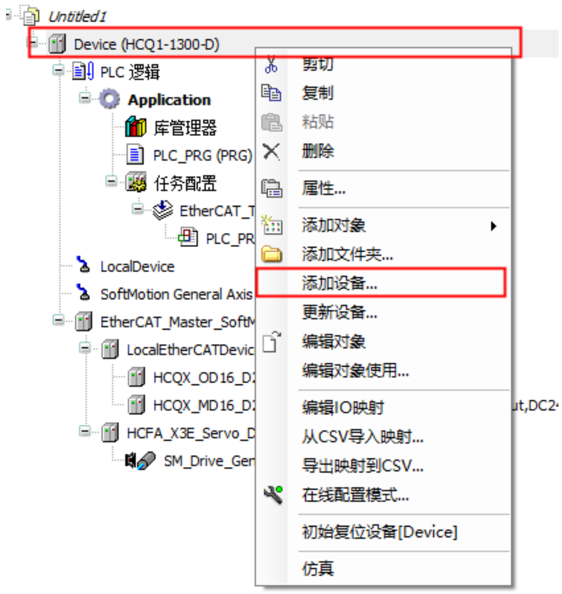
• pECATSlave (pointer to the start address of the slave structure array) points to a structure array of type stECATSlave. When the function block's bEnable pin is triggered, slave and axis pointers are automatically acquired.

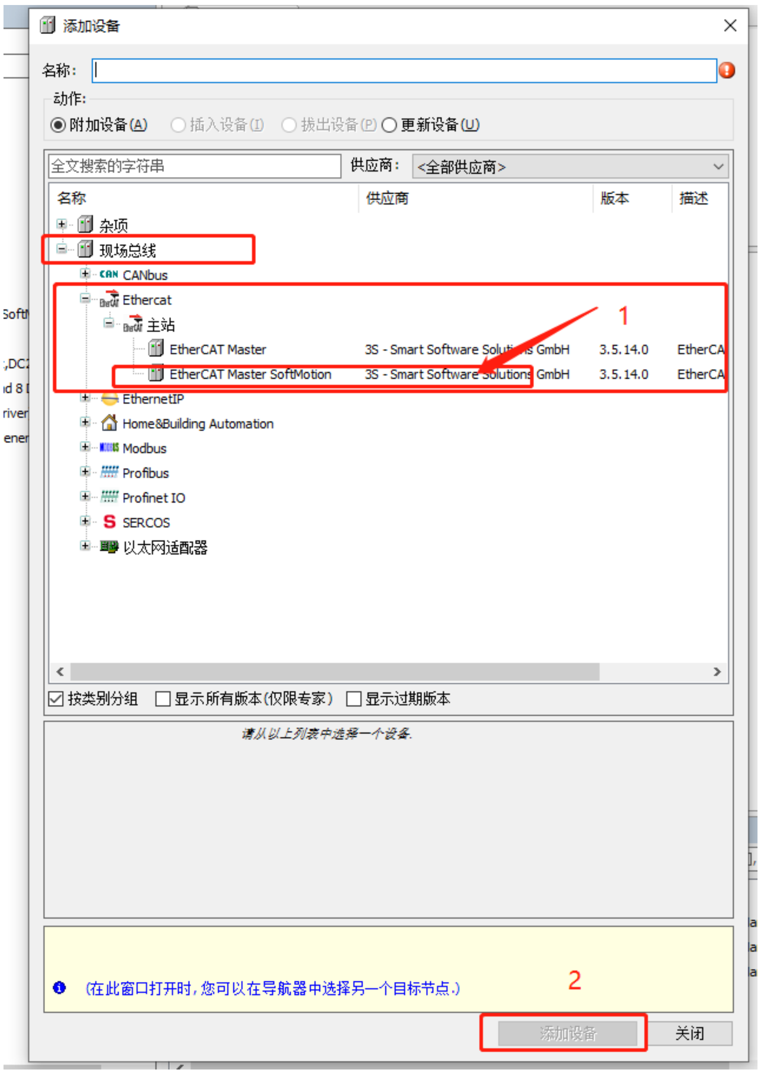
• bDeviceDiagOn (diagnostic function) specifies whether the Device -> PLC Settings -> Additional Settings -> Enable Device Diagnostics function is enabled. This must be set accurately. When diagnostics are enabled and this pin is set to TRUE, axis pointers under servo drives can be automatically acquired, and enable/disable operations on axes can be performed.

2.1.1 Usage example (diagnostic mode)

【1】Add an EtherCAT master station.

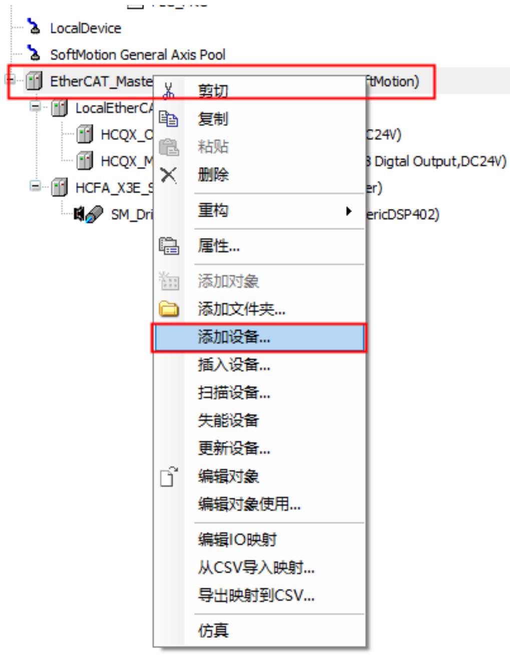
Right-click [Device] -> [Add Device]. In the pop-up "Add Device" window, navigate and click [Fieldbus] -> [EtherCAT] -> [Master] -> [EtherCAT Master SoftMotion] -> [Add Device] to complete adding the master station.

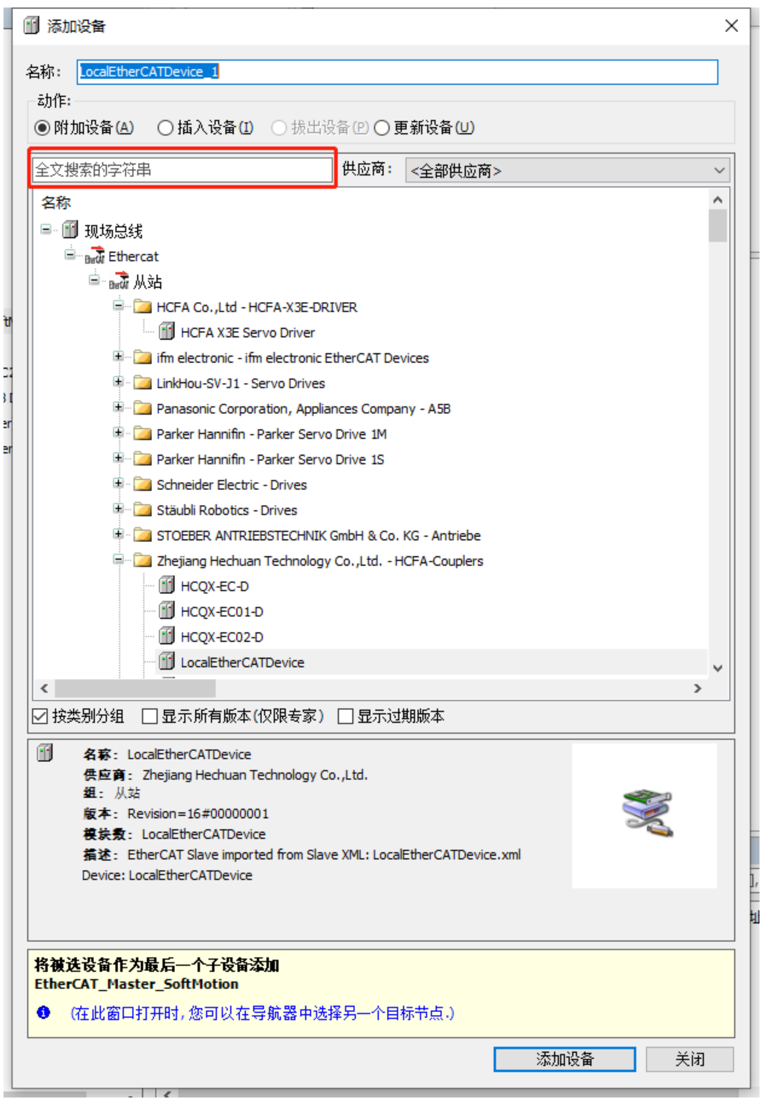




【2】Add EtherCAT slave stations.

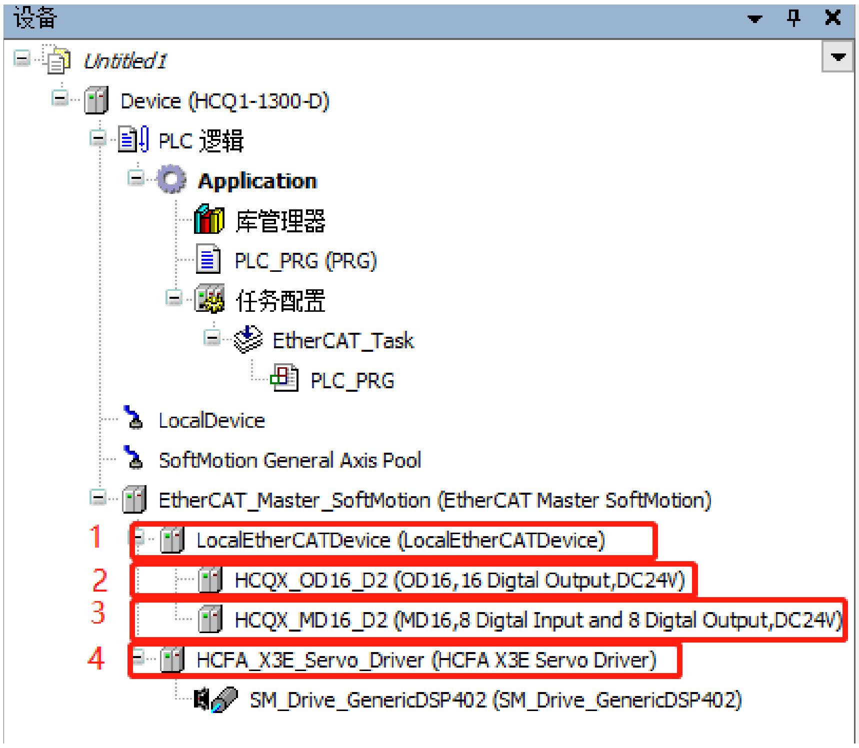
Right-click [EtherCAT_Master_SoftMotion] in the left device tree -> [Add Device]. In the pop-up "Add Device" window, add corresponding devices according to the actual connections. The search box can be used to find and add devices directly.





Alternatively, after logging into the PLC, right-click [EtherCAT_Master_SoftMotion] -> [Scan Devices]. After the scan completes, click [Copy All Devices to Project] to add all currently connected devices.

There are four slave devices here (Note: LocalEtherCATDevice is also a slave device).



【3】Declare the FB_EtherCATManager function block and the slave structure array.

Note: The array size can be determined based on the actual situation.

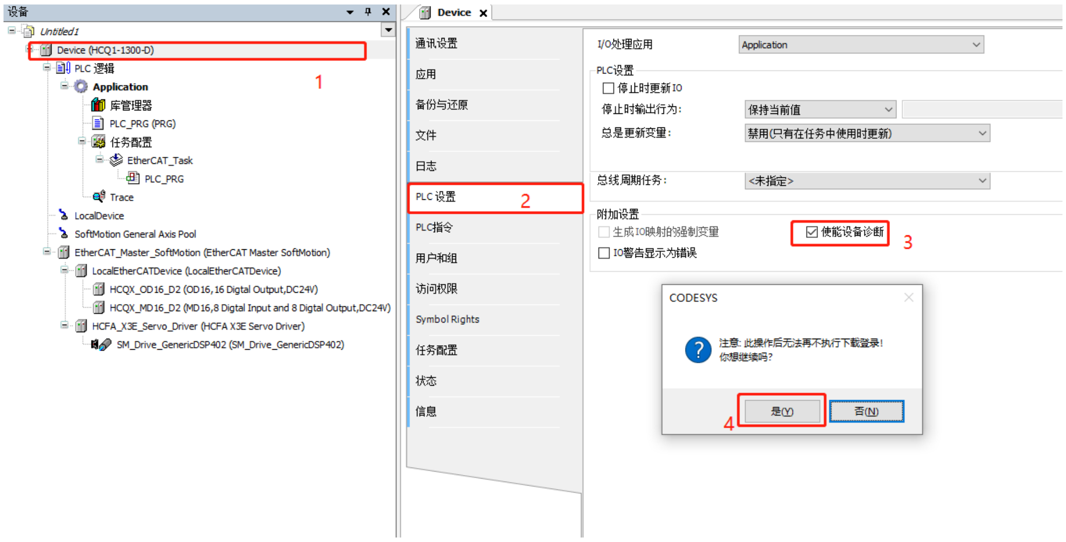
PLC_PRG | LocalEtherCATDevice

```
PROGRAM PLC_PRG
VAR
    EcatMan    :FB_EtherCATManager;
    ecat       :ARRAY [1..4] OF stECATSlave;
END_VAR
```

【4】Call the function block and assign the slave structure array to the corresponding pin.

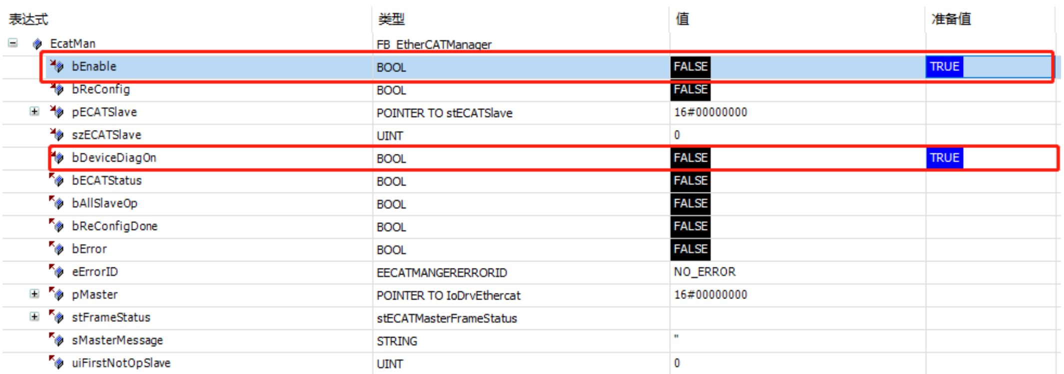
```
EcatMan(
    bEnable:= ,
    bReConfig:= ,
    pECATSlave:= ADR(ecat),
    szECATSlave:= SIZEOF(ecat),
    bDeviceDiagOn:= ,
    bECATStatus=> ,
    bAllSlaveOp=> ,
    bReConfigDone=> ,
    bError=> ,
    eErrorID=> ,
    pMaster=> ,
    stFrameStatus=> ,
    sMasterMessage=> ,
    uiFirstNotOpSlave=> );
```

【5】Enable the device's diagnostic mode. Double-click [Device] in the left device tree, select the [PLC Settings] tab, check the box for [Enable Device Diagnostics] under "Additional Settings", and click [OK] in the pop-up window.



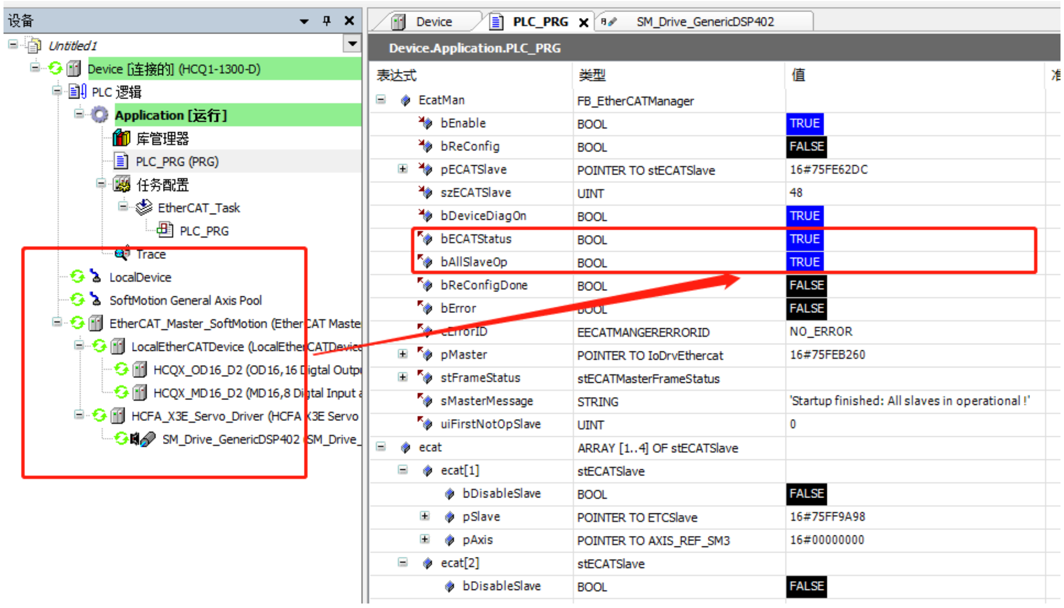
【6】Log into the PLC and start the program.

【7】In diagnostic mode, both bDeviceDiagOn and bEnable must be set to TRUE simultaneously to correctly enable diagnostic mode and acquire the axis pointers of the slaves. If the bDeviceDiagOn pin is set to TRUE after the bEnable pin, the function block actually operates in non-diagnostic mode and cannot acquire axis pointers.

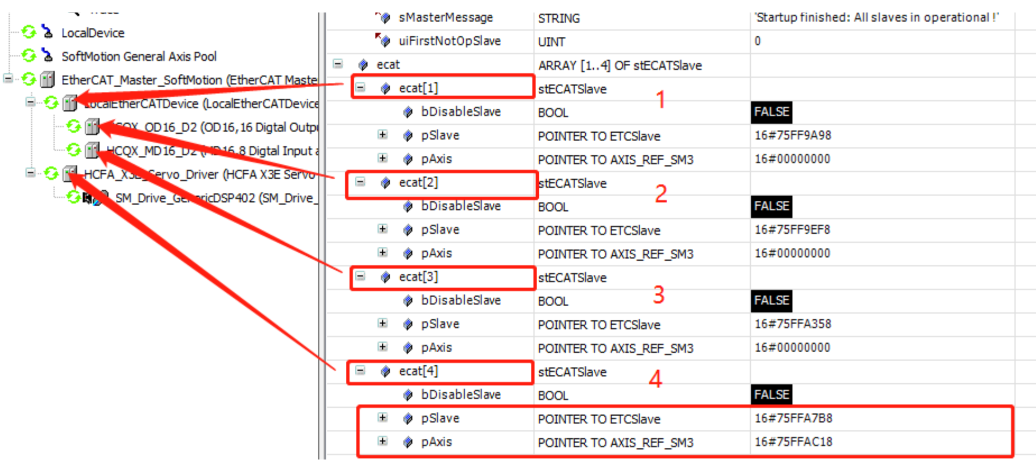


表达式	类型	值	准备值
EcatMan	FB_EtherCATManager		
bEnable	BOOL	FALSE	TRUE
bReConfig	BOOL	FALSE	
pECATSlave	POINTER TO stECATSlave	16#00000000	
szECATSlave	UINT	0	
bDeviceDiagOn	BOOL	FALSE	TRUE
bECATStatus	BOOL	FALSE	
bAllSlaveOp	BOOL	FALSE	
bReConfigDone	BOOL	FALSE	
bError	BOOL	FALSE	
eErrorID	EECATMANGERERRORID	NO_ERROR	
pMaster	POINTER TO IoDrvEthercat	16#00000000	
stFrameStatus	stECATMasterFrameStatus		
sMasterMessage	STRING	''	
uiFirstNotOpSlave	UINT	0	

When the function block is enabled, EtherCAT communication is established, and all slaves are in OP state (TRUE), both the bECATStatus and bAllSlaveOp pins output TRUE.

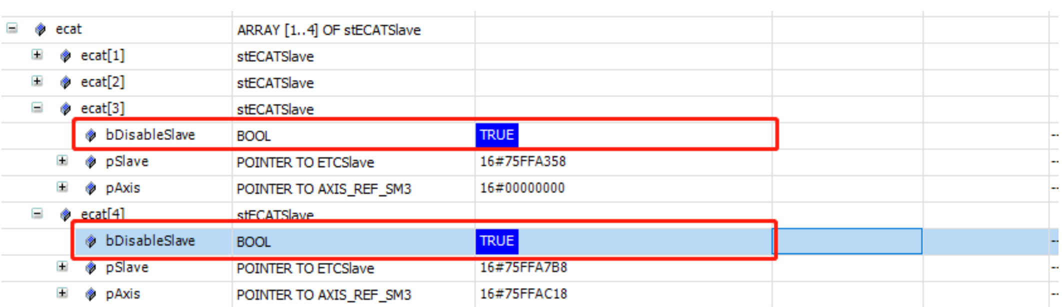


Simultaneously, the slave pointers and axis pointers can be obtained from the defined structure array. The array order corresponds to the slaves.

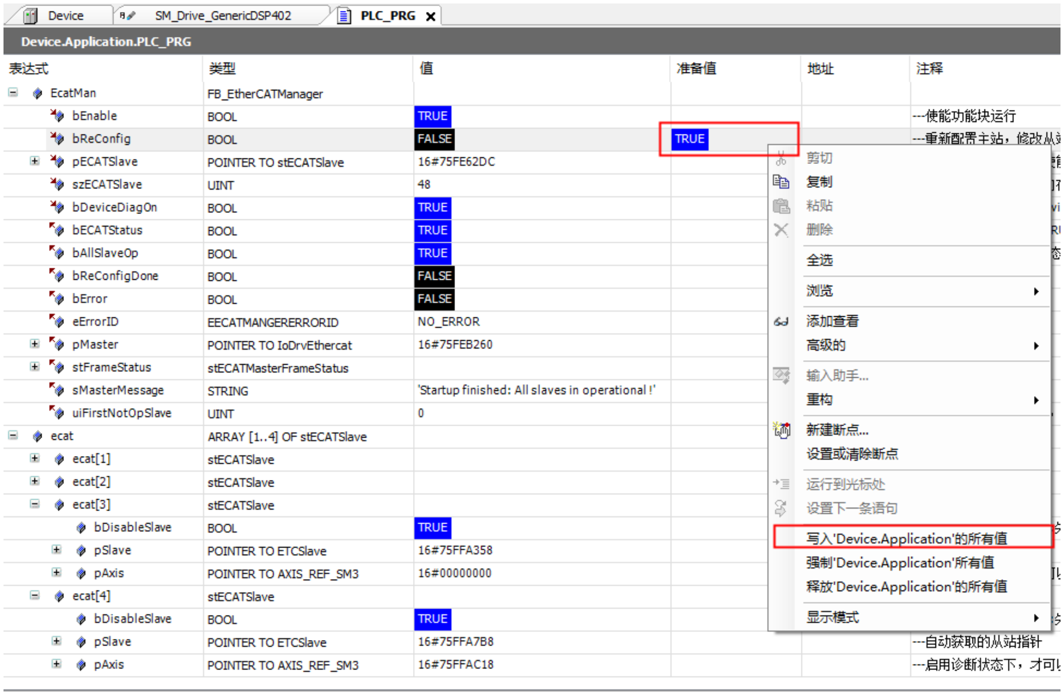


【8】Disable slave stations.

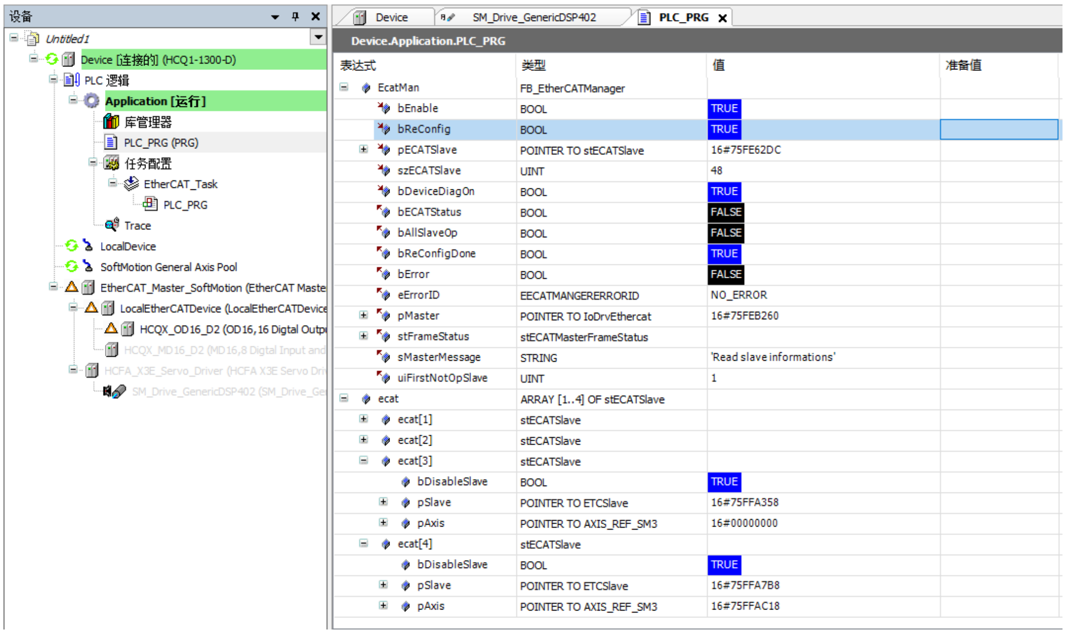
In the corresponding slave structures, enable/disable is controlled via the bDisableSlave pin (FALSE = Enable, TRUE = Disable). This example demonstrates disabling slave module HCQX_MD_D2 and servo slave HCFA_X3E_Servo_Driver, i.e., slaves 3-4 here.



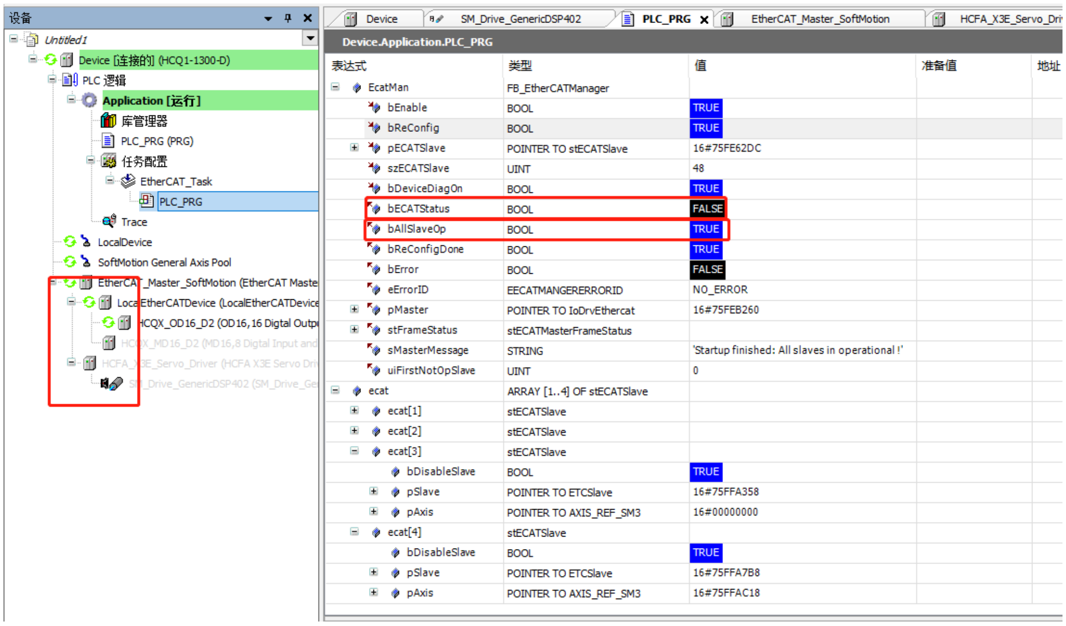
After modifying the pins of the slave structures, the FB_EtherCATManager's bReConfig pin must be triggered to make the changes effective.



After triggering the master reconfiguration pin, it can be observed in the left device tree that all slaves restart, and slaves 3-4 are disabled.



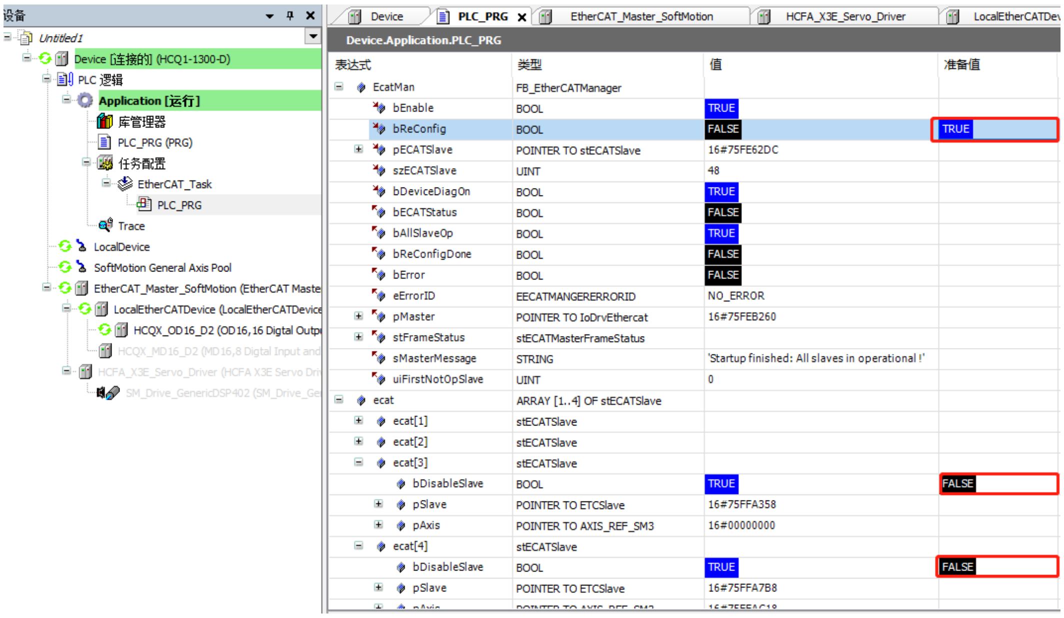
Simultaneously, the FB_EtherCATManager's bECATStatus and bAllSlaveOp pin outputs become FALSE. After the slave modules complete EtherCAT communication, bAllSlaveOp outputs TRUE. However, because the servo slave is disabled and the local device is a non-DC device, bECATStatus still outputs FALSE.



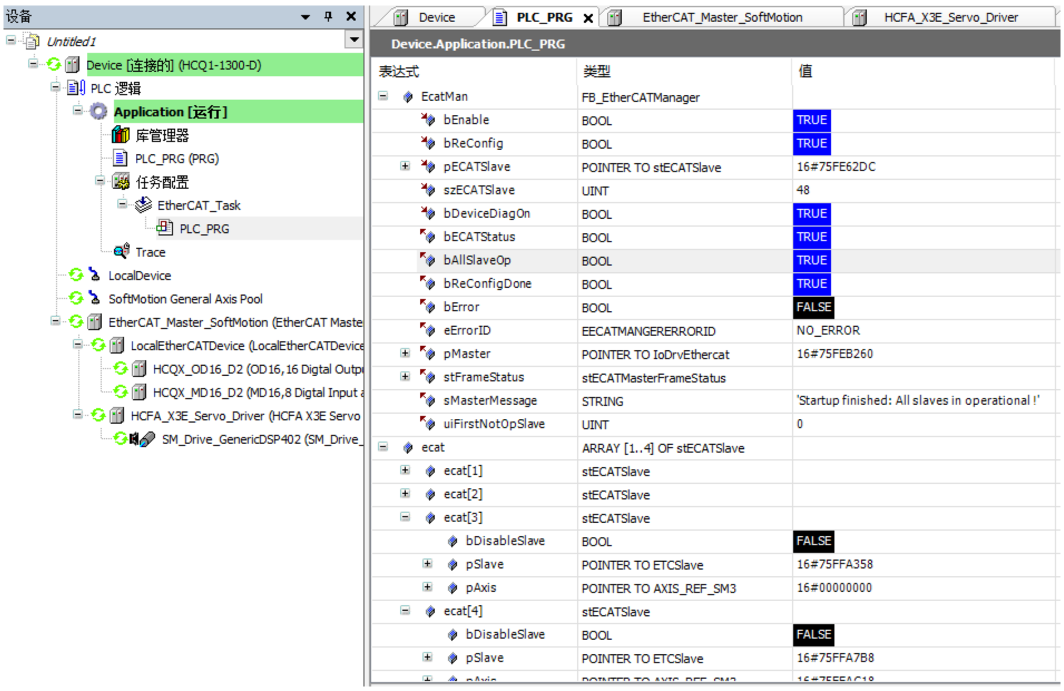
【9】Enable slave stations.

The operation is similar to disabling slaves. Set the bDisableSlave pin in the corresponding slave structures to FALSE, then trigger the FB_EtherCATManager's bReConfig pin again to make the changes effective. All slaves restart, and the corresponding slaves are re-enabled. When both the slave modules and servo slaves complete communication, the FB_EtherCATManager's bECATStatus and bAllSlaveOp pin outputs become TRUE. However, the axis error (SMC_DI_GENERAL_COMMUNICATION_ERROR), being a communication error, will not clear automatically. A separate call to the SMC3_ReinitDrive function block is required to reset the axis. Example:

```
FB_ReinitDrive(
    Axis:= SM_Drive_GenericDSP402,
    bExecute:= ,
    bVirtual:= ,
    bDone=> ,
    bBusy=> ,
    bError=> ,
    nErrorID=> );

IF EcatMan.bECATStatus AND SM_Drive_GenericDSP402.nAxisState = 1 THEN
    FB_ReinitDrive.bExecute:=TRUE;
ELSE
    FB_ReinitDrive.bExecute:=FALSE;
END_IF
```



It can be observed that all servo slaves restart, and upon communication completion, the SMC3_ReinitDrive function block is automatically triggered to initialize the axis.



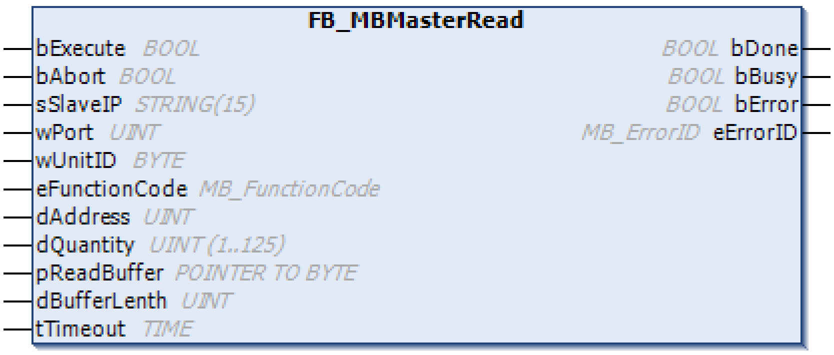
2.2 FB_MBMasterRead (FB)

This ModbusTCP master read function block enables data read operations by connecting to a slave station through a specified IP address and port, without requiring configuration in the device tree.

◆ Variables

Name	FB_MBMasterRead (Master MB read function block)		
Supported modes	CSP	CSV	CST

Graphical representation



ST representation

```
FB_MBMasterRead (
    bExecute:= ,
    bAbort:= ,
    sSlaveIP:= ,
    wPort:= ,
    wUnitID:= ,
    eFunctionCode:= ,
    dAddress:= ,
    dQuantity:= ,
    pReadBuffer:= ,
    dBufferLenth:= ,
    tTimeout:= ,
    bDone=> ,
    bBusy=> ,
    bError=> ,
    eErrorID=> );
```

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bExecute	Enable	BOOL	TRUE, FALSE	FALSE	TRUE: Enables the function block. FALSE: Disables the function block.
bAbort	Abort	BOOL	TRUE, FALSE	FALSE	TRUE: Aborts function block execution.
sSlaveIP	Slave IP address	STRING (15)			Slave IP address.
wPort	Port	UINT	0, positive integer	502	Slave port number.
wUnitID	Unit ID	BYTE	0, positive integer		Slave station number.
eFunctionCode	Function code	MB_FunctionCode			Modbus function code.
dAddress	Start address	UINT			Read start address.
dQuantity	Quantity	UINT (1..125)			Number of items to read, maximum 125 words.
pReadBuffer	Read buffer	POINTER TO BYTE			Read data buffer.
dBufferLenth	Buffer length	UINT			Read data buffer size, in words.
tTimeout	Timeout	TIME		T#50s0ms	Timeout duration.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
bDone	Done	BOOL	TRUE, FALSE	TRUE: Function block execution is complete.
bBusy	Busy	BOOL	TRUE, FALSE	TRUE: Function block is executing; TCP connection is normal.
bError	Error	BOOL	TRUE, FALSE	TRUE: An error occurred in the function block; execution has stopped.

eErrorID	Error ID	SMC_ERROR	0	Outputs the corresponding fault code when an error occurs in the function block.

(3) Transition timing of output variables

Variable	Becomes TRUE when	Becomes FALSE when
bDone	• Function block execution completes.	• Function block execution is not complete.
bBusy	• Function block is running and no error is present.	
bError	• An error occurs in the function block.	• No error is present in the function block.

◆ Key points

• After bExecute is set to TRUE, FB_MBMasterRead enters the execution state. At this time, its other input pins are processed by FB_MBMasterRead.

• After Enable is set to FALSE, FB_MBMasterRead will no longer execute any function block logic. Modifying its other input pins at this time will have no effect.

• The bAbort pin is used to abort function block execution. When bAbort is set to TRUE, all output pins become FALSE, the eErrorID information is cleared, and the data receive buffer is emptied. The bAbort pin must be set back to FALSE before the function block can be triggered again; otherwise, triggering is ineffective.

• The pReadBuffer pin is a pointer to the start address of a BYTE array. The array length should be set according to the memory type to be read. For example, reading a holding register requires two BYTES of space per register to store the data.

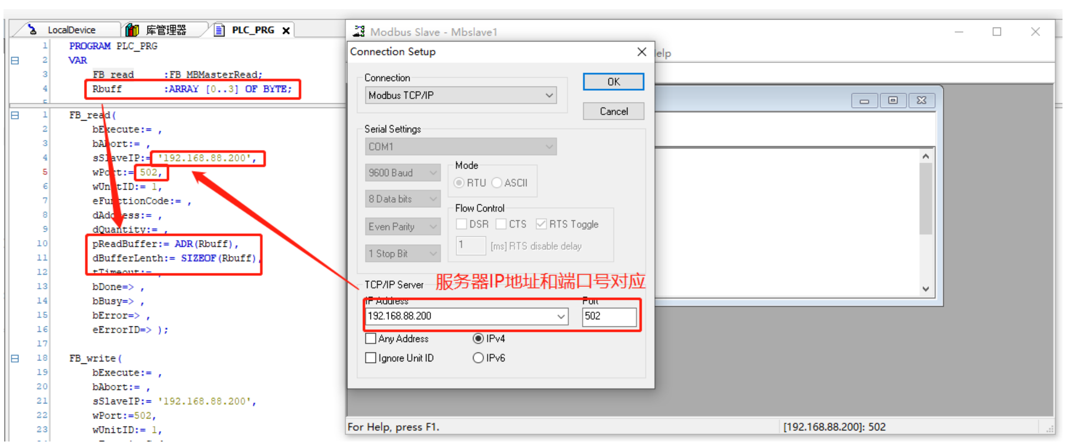
• Here, the data in the pReadBuffer read data buffer is stored in reverse byte order after reading. For example, if the array arr[0..1] BYTE reads register data 0002 (hex), the actual storage order is arr[0]: 02, arr[1]: 00.

2.2.1 Usage example (Q1 as master, client mode)

【1】Master Q1 setup

Declare and call the FB_MBMasterRead function block in the master Q1. Declare a BYTE-type array named Rbuff to be used as the data receive buffer. No other operations in the device tree are required.

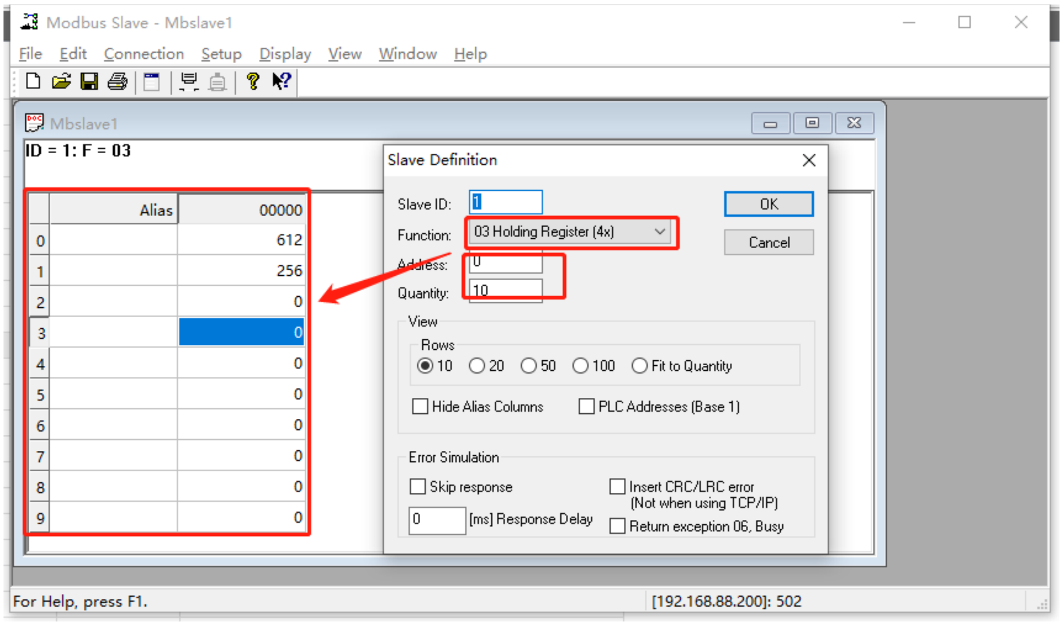
The specific settings of the function block pins are as follows.



Communication is established by connecting the master Q1's Port2 to a PC's ModbusTCP server tool. The host computer's network port IP address is 192.168.88.200, and the port number is set to 502.

【2】Host computer server setup

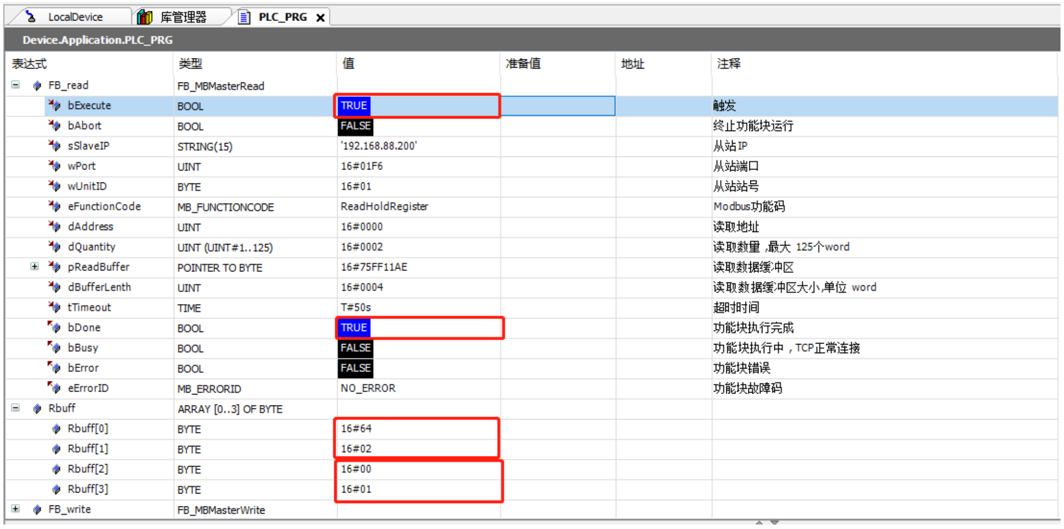
Set the server data type to Holding Register, with a start address of 0 and a length of 10, meaning there are 10 registers from address 0 to 9 for easy modification. Here, modify register address 0 to the value 612 (16#0264), and modify register address 1 to 256 (16#0010).



【3】Read data

Download and log the program into the master Q1 device. Set the function code, the target read start address for the registers, and the read length. This example demonstrates reading data from two registers at addresses 0-1.

Trigger the bExecute pin. After communication is complete, the bDone pin outputs TRUE. Simultaneously, the corresponding data can be seen read into the defined data buffer Rbuff[0..3]: 64 02, 00 01 (data is in reverse byte order), which matches the values 16#0264 and 16#0010 previously set in registers 0-1 on the Q1 slave.



LocalDevice | 库管理器 | PLC_PRG

Device.Application.PLC_PRG

表达式	类型	值	准备值	地址	注释
FB_read	FB_MBMasterRead				
bExecute	BOOL	TRUE			触发
bAbort	BOOL	FALSE			终止功能块运行
sSlaveIP	STRING(15)	'192.168.88.200'			从站 IP
wPort	UINT	16#01F6			从站端口
wUnitID	BYTE	16#01			从站站号
eFunctionCode	MB_FUNCTIONCODE	ReadHoldRegister			Modbus功能码
dAddress	UINT	16#0000			读取地址
dQuantity	UINT (UINT#1..125)	16#0002			读取数量 ,最大 125个word
pReadBuffer	POINTER TO BYTE	16#75FF11AE			读取数据缓冲区
dBufferLenth	UINT	16#0004			读取数据缓冲区大小,单位 word
tTimeout	TIME	T#50s			超时时间
bDone	BOOL	TRUE			功能块执行完成
bBusy	BOOL	FALSE			功能块执行中，TCP正常连接
bError	BOOL	FALSE			功能块错误
eErrorID	MB_ERRORID	NO_ERROR			功能块故障码
Rbuff	ARRAY [0..3] OF BYTE				
Rbuff[0]	BYTE	16#64			
Rbuff[1]	BYTE	16#02			
Rbuff[2]	BYTE	16#00			
Rbuff[3]	BYTE	16#01			
FB_write	FB_MBMasterWrite				

2.3 FB_MBMasterWrite (FB)

This ModbusTCP master write function block enables data write operations by connecting to a slave station through a specified IP address and port, without requiring configuration in the device tree.

◆ Variables

Name	FB_MBMasterWrite (Master MB write function block)		
Supported modes	CSP	CSV	CST
Graphical representation		ST representation	
		FB_MBMasterWrite (bExecute:= , bAbort:= , sSlaveIP:= , wPort:= , wUnitID:= , eFunctionCode:= , dAddress:= , dQuantity:= , dValue:= , pWriteBuffer:= , dBufferLenth:= , bDone=> , bError=> , bBusy=> , ErrorID=>);	

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bExecute	Enable	BOOL	TRUE, FALSE	FALSE	TRUE: Enables the function block. FALSE: Disables the function block.

bAbort	Abort	BOOL	TRUE, FALSE	FALSE	TRUE: Aborts function block execution.
sSlaveIP	Slave IP address	STRING (15)			Slave IP address.
wPort	Port	UINT	0, positive integer	502	Slave port number.
wUnitID	Unit ID	BYTE	0, positive integer		Slave station number.
eFunctionCode	Function code	MB_FunctionCode			Modbus function code.
dAddress	Start address	UINT			Write start address.
dQuantity	Quantity	UINT (1..125)			Number of items to write (for function codes 15, 16), maximum 125 words.
dValue	Value to write	WORD			Represents the value to be written to a single register.
pWriteBuffer	Write buffer	POINTER TO BYTE		SM3_Basic. NULL	Write data buffer (for function codes 15, 16).
dBufferLenth	Buffer length	UINT			Read data buffer size, in words.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
bDone	Done	BOOL	TRUE, FALSE	TRUE: Function block execution is complete.
bBusy	Busy	BOOL	TRUE, FALSE	TRUE: Function block is executing; TCP connection is normal.
bError	Error	BOOL	TRUE, FALSE	TRUE: An error occurred in the function block; execution has stopped.
eErrorID	Error ID	SMC_ERROR	0	Outputs the corresponding fault code when an error occurs in the function block.

(3) Transition timing of output variables

Variable	Becomes TRUE when	Becomes FALSE when
bDone	• Function block execution completes.	• Function block execution is not complete.
bBusy	• Function block is running and no error is present.	
bError	• An error occurs in the function block.	• No error is present in the function block.

◆ Key points

• After bExecute is set to TRUE, FB_MBMasterWrite enters the execution state. At this time, its other input pins are processed by FB_MBMasterWrite.

• After Enable is set to FALSE, FB_MBMasterWrite will no longer execute any function block logic. Modifying its other input pins at this time will have no effect.

• The bAbort pin is used to abort function block execution. When bAbort is set to TRUE, all output pins become FALSE, the eErrorID information is cleared, and the data receive buffer is emptied. The bAbort pin must be set back to FALSE before the function block can be triggered again; otherwise, triggering is ineffective.

• The pReadBuffer pin is a pointer to the start address of a BYTE array. The array length should be set according to the memory type to be read. For example, reading a holding register requires two BYTES of space per register to store the data.

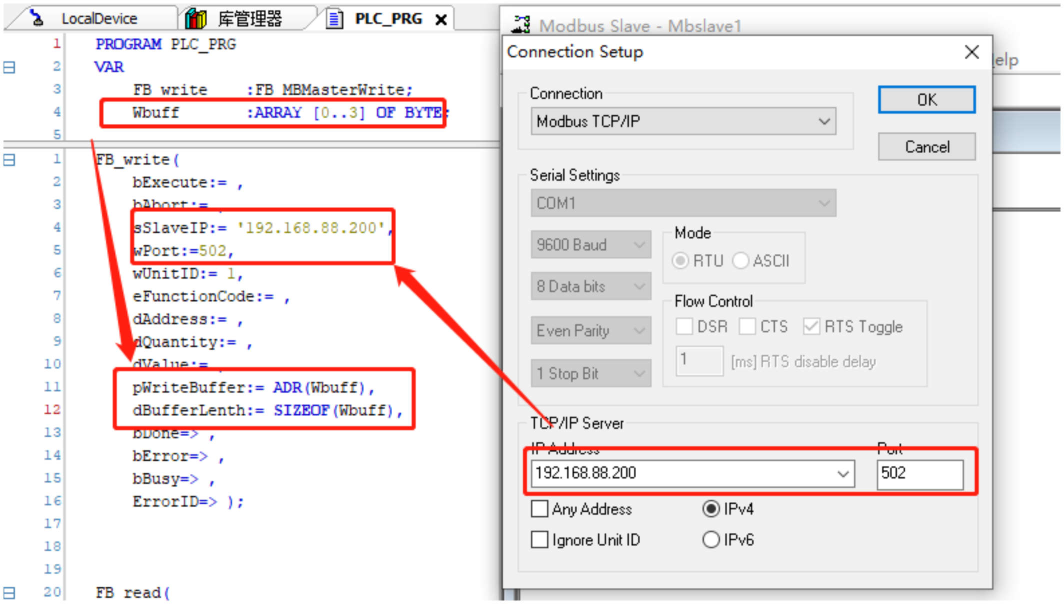
• Here, the data in the pReadBuffer read data buffer is stored in reverse byte order after reading. For example, if the array arr[0..1] BYTE reads register data 0002 (hex), the actual storage order is arr[0]: 02, arr[1]: 00.

2.3.1 Usage example (Q1 as master, client mode)

【1】Master Q1 setup

Declare and call the FB_MBMasterWrite function block in the master Q1. Declare a BYTE-type array named Wbuff to be used as the data transmit buffer. No other operations in the device tree are required.

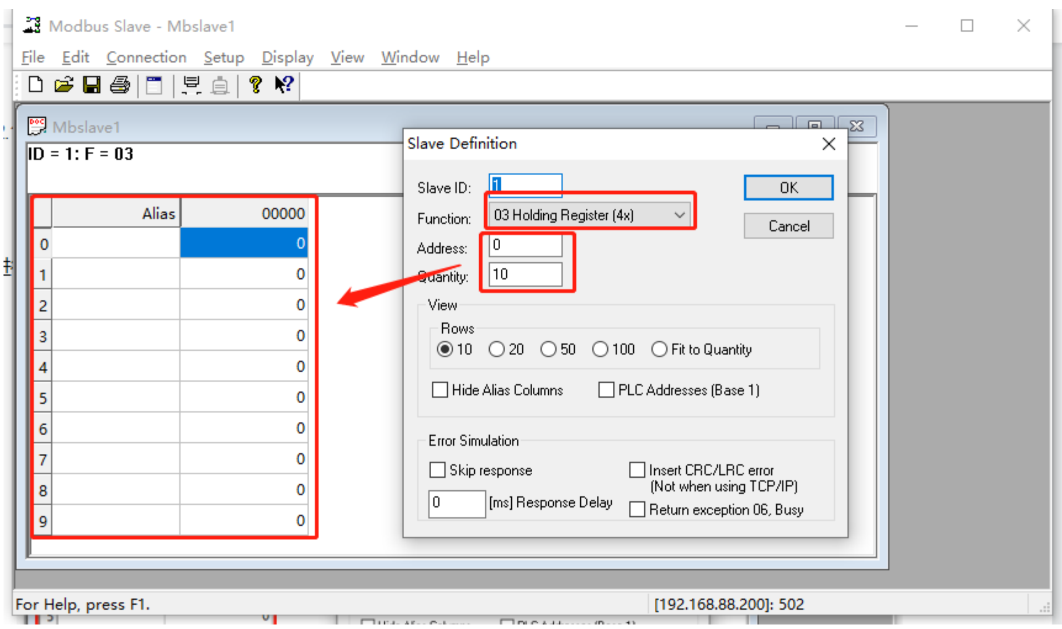
The specific settings of the function block pins are as follows.



Communication is established by connecting the master Q1's Port2 to a host computer's ModbusTCP server tool. The PC's network port IP address is 192.168.88.200, and the port number is set to 502.

【2】Host computer server setup

Set the server data type to Holding Register, with a start address of 0 and a length of 10, meaning there are 10 registers (addresses 0-9) available for modification.

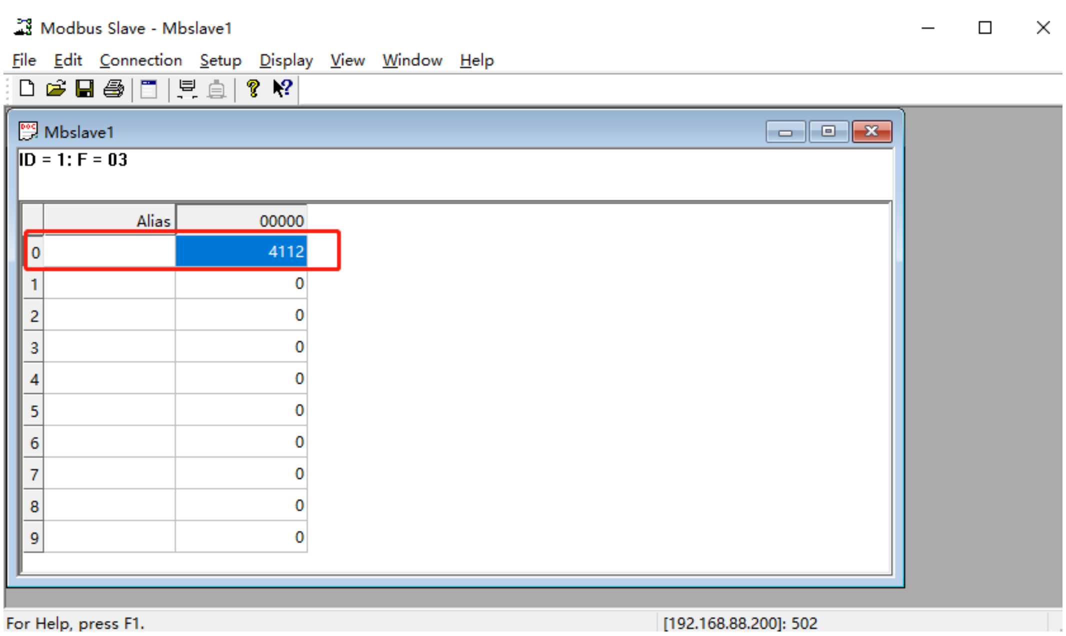


【3】Writing to a single register

Download and log the program into the master Q1 device. Set the function code and the target register address. The value to be written is set here to modify register address 0 to 16#1010, which is decimal 4112.

Trigger the bExecute pin. After communication is complete, the bDone pin outputs TRUE.

Simultaneously, in the host computer tool, the value of register address 0 is modified to 4112.



【4】Writing to multiple registers

Download and log the program into the master Q1 device. Set function code 16, the target start register address, the number of registers to operate on, and set the corresponding values to be written in the Wbuff data transmit buffer array.

This example demonstrates modifying two registers at addresses 1-2. The values are set to 16#0001 (decimal 1) and 16#1000 (decimal 4096), respectively. (Note: Data in the buffer is in reverse byte order.)

LocalDevice | 库管理器 | PLC_PRG

Device.Application.PLC_PRG

表达式	类型	值	准备值	地址	注释
FB_write	FB_MBMasterWrite				
bExecute	BOOL	FALSE			开启ModbusTCP客户端
bAbort	BOOL	FALSE			终止功能块运行
sSlaveIP	STRING(15)	'192.168.88.200'			需要连接的服务器的IP地址
wPort	UINT	16#01F6			服务器开启侦听的端口号 默认 502
wUnitID	BYTE	16#01			从站节点号
eFunctionCode	MB_FUNCTIONCODE	WriteSignleRegister	WriteMultipleRegister		Modbus 功能码 16功能码，写多个寄存器
dAddress	WORD	16#0000	16#0001		需要操作的寄存器或者线圈的起始地址
dQuantity	WORD (0..122)	16#0001	16#0002		写入寄存器个数，当功能码为16或者15时使用 单位 word，
dValue	WORD	16#1010			代表需要写入的单个寄存器的值
pWriteBuffer	POINTER TO BYTE	16#75FF11A8			当功能码为15或者16时的写入数据缓存区
dBufferLenth	UINT	16#0004			写入数据缓冲区大小 单位 word
bDone	BOOL	FALSE			TRUE 服务器连接成功
bError	BOOL	FALSE			TRUE：有错误产生
bBusy	BOOL	FALSE			功能块执行中
ErrorID	MB_ERRORID	NO_ERROR			报错代码
TCP_Client1	NBS.TCP_Client				
TCP_Read1	NBS.TCP_Read				
TCP_Write1	NBS.TCP_Write				
TCP_Write2	NBS.TCP_Write				
ipAddr1	NBS.IP_ADDR				
szCount	UDINT	16#00000000			

Wbuff	ARRAY [0..3] OF BYTE		
Wbuff[0]	BYTE	16#00	16#01
Wbuff[1]	BYTE	16#00	16#00
Wbuff[2]	BYTE	16#00	16#00
Wbuff[3]	BYTE	16#00	16#10

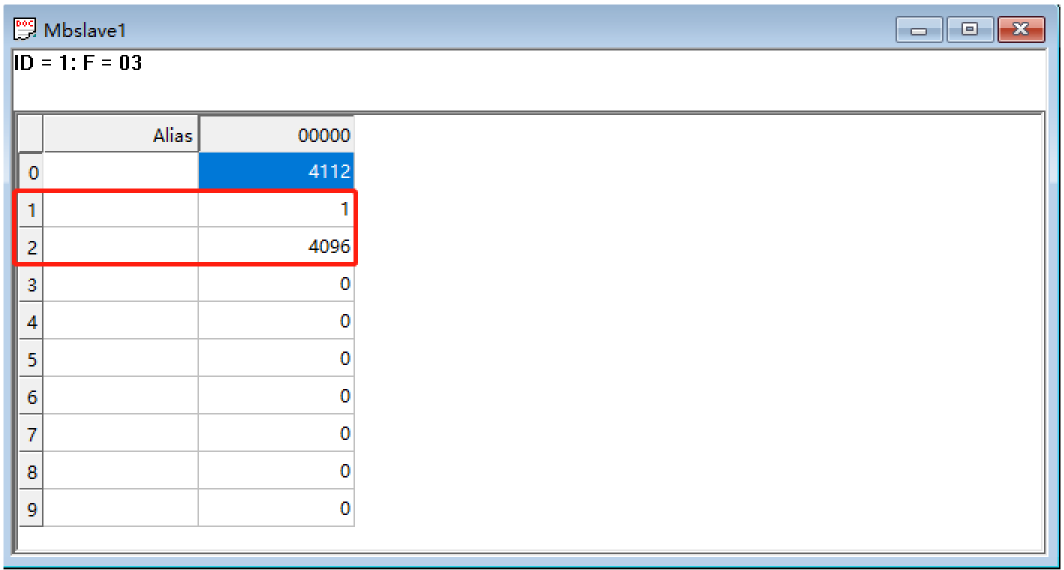
Trigger the bExecute pin. After communication is complete, the bDone pin outputs TRUE.

LocalDevice | 库管理器 | PLC_PRG

Device.Application.PLC_PRG

表达式	类型	值	准备值	地址	注释
FB_write	FB_MBMasterWrite				
bExecute	BOOL	TRUE			开启ModbusTCP客户端
bAbort	BOOL	FALSE			终止功能块运行
sSlaveIP	STRING(15)	'192.168.88.200'			需要连接的服务器的IP地址
wPort	UINT	16#01F6			服务器开启侦听的端口号 默认 502
wUnitID	BYTE	16#01			从站节点号
eFunctionCode	MB_FUNCTIONCODE	WriteMultipleRegister			Modbus 功能码
dAddress	WORD	16#0001			需要操作的寄存器或者线圈的起始地址
dQuantity	WORD (0..122)	16#0002			写入寄存器个数，当功能码为16或者15时使用 单位 word，
dValue	WORD	16#0000			代表需要写入的单个寄存器的值
pWriteBuffer	POINTER TO BYTE	16#75FF1814			当功能码为15或者16时的写入数据缓存区
dBufferLenth	UINT	16#0004			写入数据缓冲区大小 单位 word
bDone	BOOL	TRUE			TRUE 服务器连接成功
bError	BOOL	FALSE			TRUE：有错误产生
bBusy	BOOL	FALSE			功能块执行中
ErrorID	MB_ERRORID	NO_ERROR			报错代码
TCP_Client1	NBS.TCP_Client				
TCP_Read1	NBS.TCP_Read				
TCP_Write1	NBS.TCP_Write				
TCP_Write2	NBS.TCP_Write				
ipAddr1	NBS.IP_ADDR				

Simultaneously, in the host computer tool, the values of registers 1-2 are modified to 1 and 4096.



2.4 NetManager (Function group)

A function group for network adapter management. It supports reading all network adapter information and setting corresponding information, among other functions.

2.4.1 FC_GetAllAdapterInfo (FUN)

◆ **Variables**

<table>
<tr><th>Name</th><th colspan="3">FC_GetAllAdapterInfo (Get all device network adapter information)</th></tr>
<tr><td>Supported modes</td><td>CSP</td><td>CSV</td><td>CST</td></tr>
<tr><td colspan="2">Graphical representation</td><td colspan="2">ST representation</td></tr>
<tr><td colspan="2"></td><td colspan="2">FC_GetAllAdapterInfo(
AdapterInfo:= , AdapterCount=>);</td></tr>
</table>

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
AdapterInfo	Device information	POINTER TO stAdapterInfo			Pointer to the device information structure array.

(2) Output variable

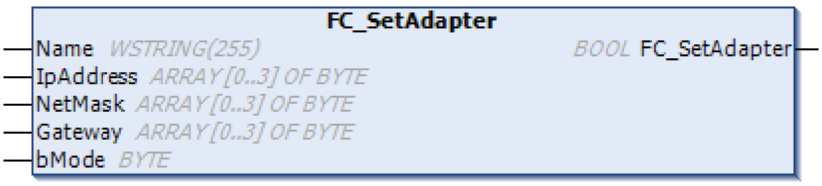
Output variable	Name	Data type	Valid range	Description
AdapterCount	Device count	UINT		Number of currently connected devices/adapters.

◆ **Key points**

・ The AdapterInfo pin is a pointer to a device structure array. When the function is called and the program is in the running state, the function automatically reads all network adapter information of the current device.

2.4.2 FC_SetAdapter (FUN)

◆ Variables

Name	FC_SetAdapter (Set network adapter information)		
Supported modes	CSP	CSV	CST
Graphical representation		ST representation	
		FC_SetAdapter(Name:= , IpAddress:= , NetMask:= , Gateway:= , bMode:=)	

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
Name	Name	WSTRING			Network adapter name.
IpAddress	IP address	ARRAY [0..3] OF BYTE			IP address.
NetMask	Subnet mask	ARRAY [0..3] OF BYTE			Subnet mask.
Gateway	Gateway	ARRAY [0..3] OF BYTE			Gateway.
bMode	Write mode	BYTE			Selection of which content to modify.

◆ Key points

- bMode is used to control whether to modify each piece of information for the current device. Bit0 represents writing IpAddress, Bit1 represents writing NetMask, and Bit2 represents writing Gateway.
- Example: Setting bMode to 5 (binary 0101) means modifying the IP address and the gateway.

2.4.3 Usage example

【1】Declare variables and call functions

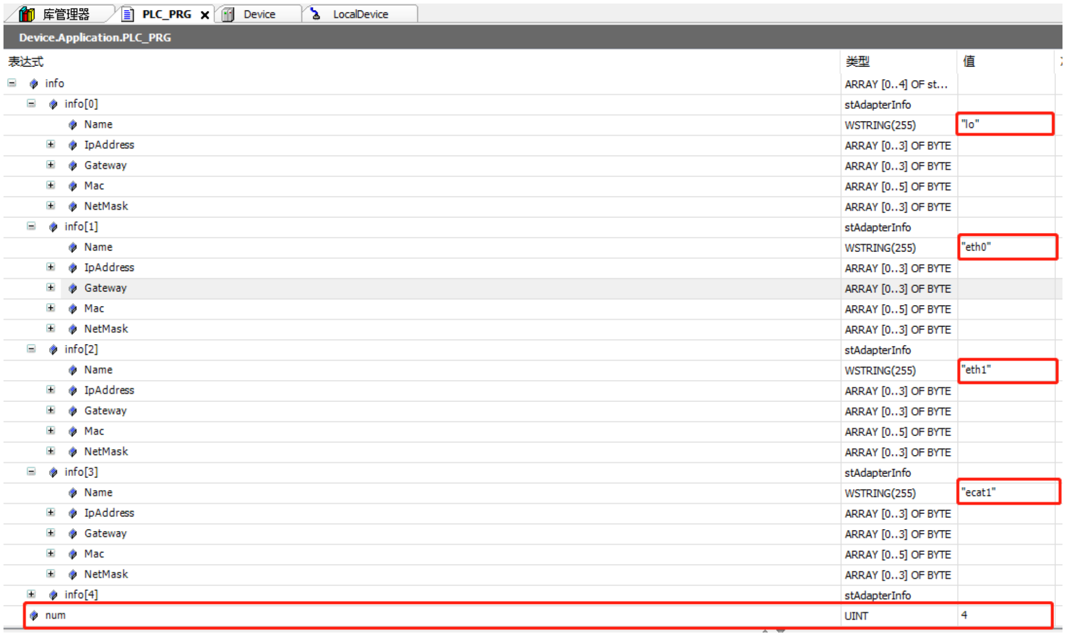
Map corresponding variables to each pin of the functions.

```
库管理器   PLC_PRG ×   Device   LocalDevice

PROGRAM PLC_PRG
VAR
    info    :ARRAY [0..4] OF  stAdapterInfo;
    num     :UINT;

    Name    :WSTRING;
    IP      :ARRAY [0..3] OF BYTE;
    Net     :ARRAY [0..3] OF BYTE;
    Getw    :ARRAY [0..3] OF BYTE;
    Mode    :BYTE;
END_VAR
```

```
FC_GetAllAdapterInfo(AdapterInfo:= info, AdapterCount=> num);

FC_SetAdapter(
    Name:= Name,
    IpAddress:= IP,
    NetMask:= Net,
    Gateway:= Getw,
    bMode:= Mode);
```

【2】Read all network adapter information of the device

After logging into the program, the FC_GetAllAdapterInfo function automatically reads all network adapter information of the current device and writes it into the info structure array. Simultaneously, the AdapterCount pin outputs the number of device adapters currently read.



库管理器 | PLC_PRG | Device | LocalDevice

Device.Application.PLC_PRG

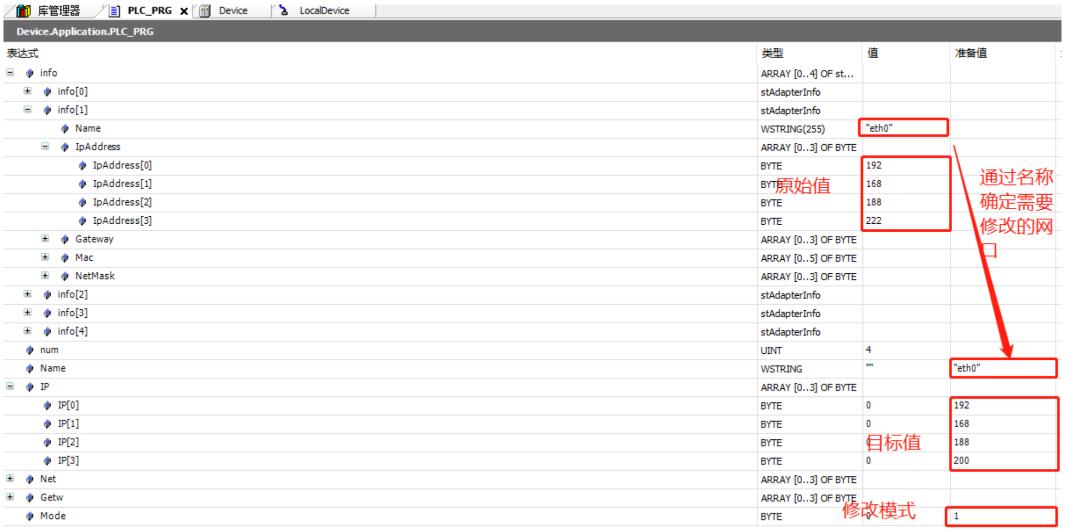
表达式	类型	值
info	ARRAY [0..4] OF st...	
info[0]	stAdapterInfo	
Name	WSTRING(255)	"lo"
IpAddress	ARRAY [0..3] OF BYTE	
Gateway	ARRAY [0..3] OF BYTE	
Mac	ARRAY [0..5] OF BYTE	
NetMask	ARRAY [0..3] OF BYTE	
info[1]	stAdapterInfo	
Name	WSTRING(255)	"eth0"
IpAddress	ARRAY [0..3] OF BYTE	
Gateway	ARRAY [0..3] OF BYTE	
Mac	ARRAY [0..5] OF BYTE	
NetMask	ARRAY [0..3] OF BYTE	
info[2]	stAdapterInfo	
Name	WSTRING(255)	"eth1"
IpAddress	ARRAY [0..3] OF BYTE	
Gateway	ARRAY [0..3] OF BYTE	
Mac	ARRAY [0..5] OF BYTE	
NetMask	ARRAY [0..3] OF BYTE	
info[3]	stAdapterInfo	
Name	WSTRING(255)	"ecat1"
IpAddress	ARRAY [0..3] OF BYTE	
Gateway	ARRAY [0..3] OF BYTE	
Mac	ARRAY [0..5] OF BYTE	
NetMask	ARRAY [0..3] OF BYTE	
info[4]	stAdapterInfo	
num	UINT	4

The read information for the Eth0 network adapter is displayed here.

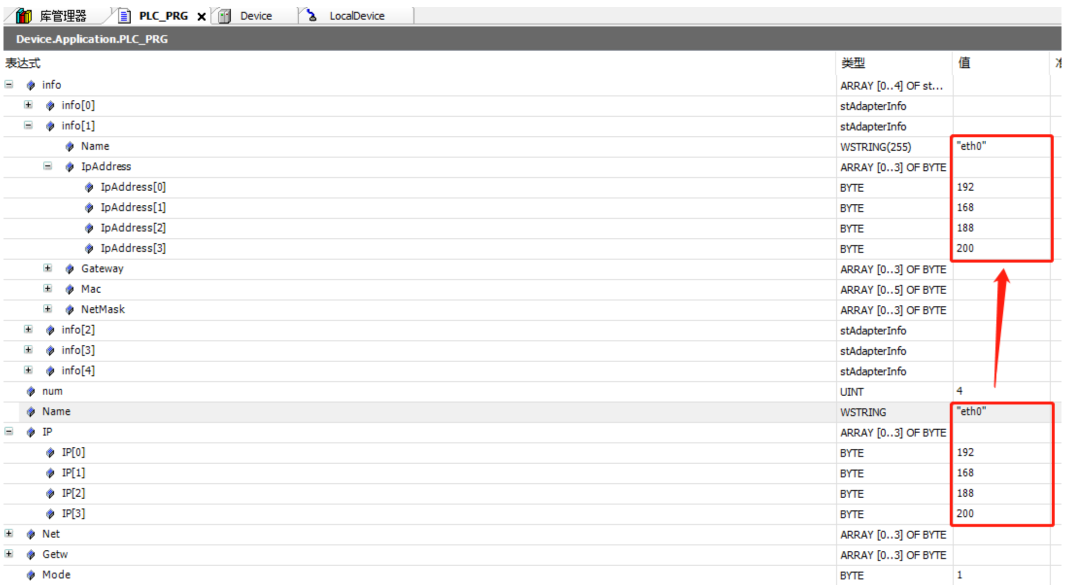
info[1]	stAdapterInfo	
Name	WSTRING(255)	"eth0"
IpAddress	ARRAY [0..3] OF BYTE	
IpAddress[0]	BYTE	192
IpAddress[1]	BYTE	168
IpAddress[2]	BYTE	188
IpAddress[3]	BYTE	222
Gateway	ARRAY [0..3] OF BYTE	
Gateway[0]	BYTE	0
Gateway[1]	BYTE	0
Gateway[2]	BYTE	0
Gateway[3]	BYTE	0
Mac	ARRAY [0..5] OF BYTE	
Mac[0]	BYTE	0
Mac[1]	BYTE	4
Mac[2]	BYTE	159
Mac[3]	BYTE	4
Mac[4]	BYTE	225
Mac[5]	BYTE	195
NetMask	ARRAY [0..3] OF BYTE	
NetMask[0]	BYTE	255
NetMask[1]	BYTE	255
NetMask[2]	BYTE	255
NetMask[3]	BYTE	0

【3】Modify specified information of a specified network adapter

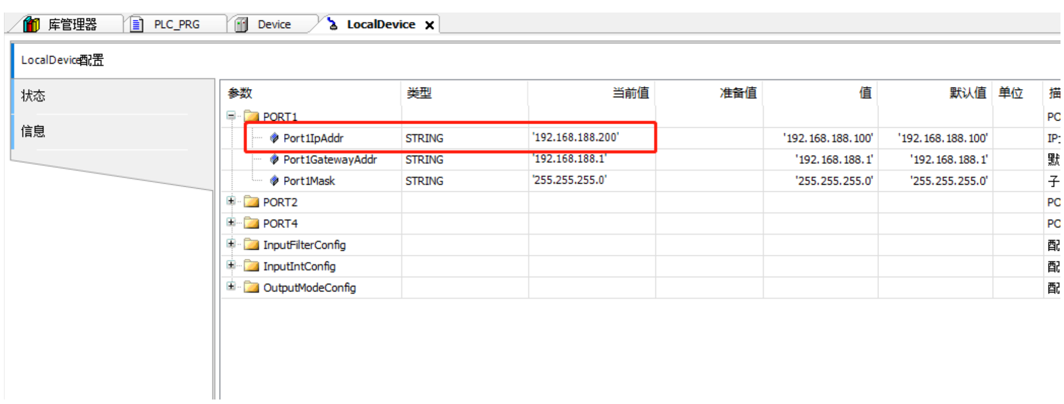
This example demonstrates modifying the IP address of the Eth0 adapter to 192.168.88.200.



After right-clicking to write the data, it can be seen that the IP address pin for the device adapter eth0 has been modified.



In the LocalDevice, it can be seen that the IP address of the device's Port1 Ethernet port has been changed to 192.168.188.200.

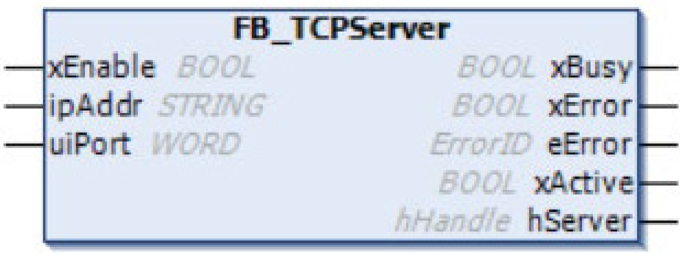


2.5 TCP (Function group)

TCP communication function blocks encapsulated based on socket.

2.5.1 FB_TCPServer (FB)

◆ **Variables**

Name	FB_TCPServer (Create server function block)
Graphical representation	ST representation
	FB_TCPServer (xEnable:= , ipAddr:= , uiPort:= , xBusy=> , xError=> , eError=> , xActive=> , hServer=>);

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
xEnable	Function block enable	BOOL	TRUE, FALSE	FALSE	
ipAddr	Server IP address	STRING		0	
uiPort	Server port number	WORD		0	

(2) Output variable

Output variable	Name	Data type	Valid range	Description
xBusy		BOOL	TRUE, FALSE	TRUE: Function block is running.
xError		BOOL	TRUE, FALSE	
eError		ErrorID		Error code.
xActive		BOOL	TRUE, FALSE	TRUE: Handle is output upon successful creation. FALSE: The current handle is inactive.
hServer	Handle	hHandle		The interface handle generated upon creation, provided to the connection function block.

2.5.2 FB_TCPConnection (FB)

◆ Variables

Name	FB_TCPConnection (Connection function block)
Graphical representation	ST representation
 	FB_TCPConnection(xEnable:= , hServer:= , xBusy=> , xError=> , eError=> , xActive=> , hHandle=>);

(1) Input variable

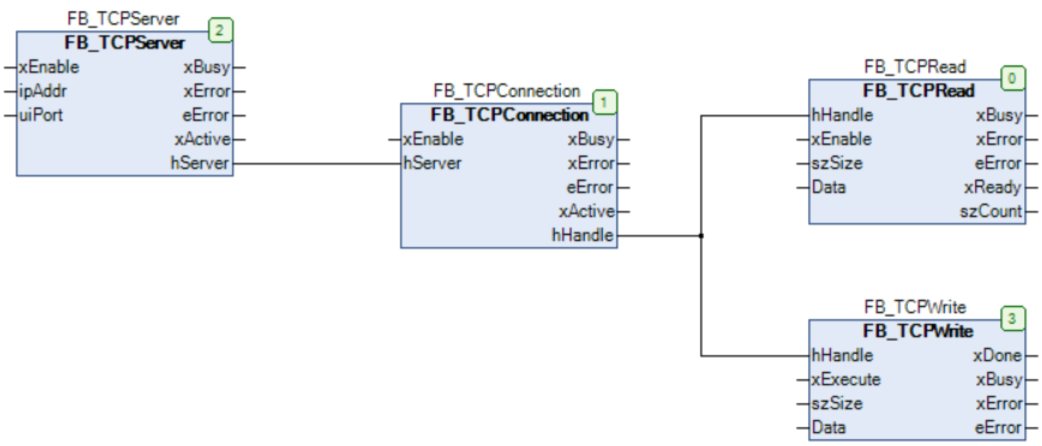
Input variable	Name	Data type	Valid range	Initial value	Description
xEnable	Function block enable	BOOL	TRUE, FALSE	FALSE	
hServer	Handle	hHandle			Accepts the handle generated by FB_ TCPServer.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
xBusy		BOOL	TRUE, FALSE	
xError		BOOL	TRUE, FALSE	
eError		ErrorID		Error code.
xActive		BOOL	TRUE, FALSE	TRUE: Handle is output upon successful creation. FALSE: The current handle is inactive.
hHandle		hHandle		Connection handle, provided to the read/write function blocks.

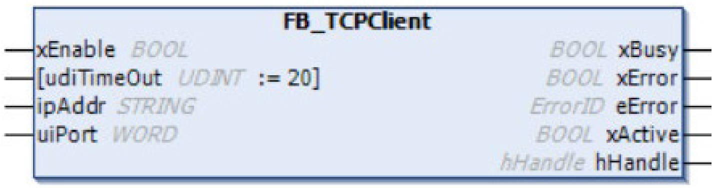
◆ Key points

- The handle connection is used in the following form.



2.5.3 FB_TCPClient (FB)

◆ Variables

Name	FB_TCPClient (Create client function block)
Graphical representation	ST representation
	FB_TCPClient (xEnable:= , udiTimeOut:=, ipAddr:=, uiPort:= , xBusy=> , xError=> , eError=> , xActive=> , hHandle=>);

(1) Input variable

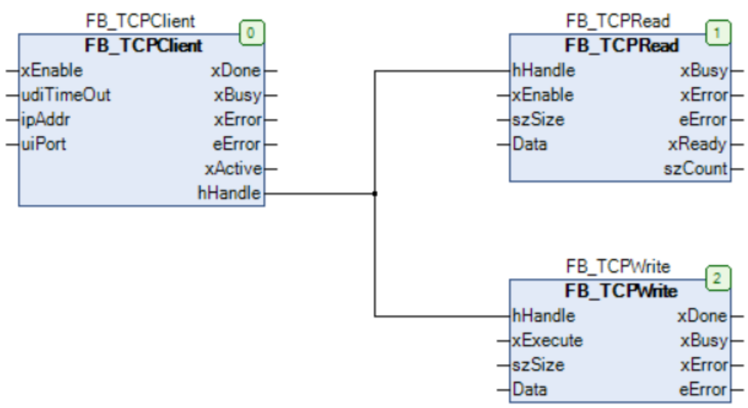
Input variable	Name	Data type	Valid range	Initial value	Description
xEnable	Function block enable.	BOOL	TRUE, FALSE	FALSE	
udiTimeOut	Handle	UDINT		20	Unit: ms. Connection timeout duration, affected by the task cycle in which the function block resides.
ipAddr	Server IP address.	STRING		0	
uiPort	Server port number.	WORD		0	

(2) Output variable

Output variable	Name	Data type	Valid range	Description
xBusy		BOOL	TRUE, FALSE	
xError		BOOL	TRUE, FALSE	
eError		ErrorID		Error code.
xActive		BOOL	TRUE, FALSE	TRUE: Handle is output upon successful creation. FALSE: The current handle is inactive.
hHandle		hHandle		Connection handle, provided to the read/write function blocks.

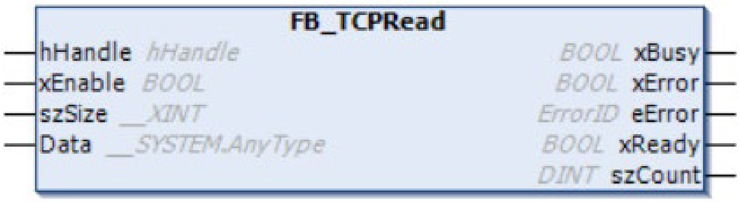
◆ Key points

- The handle connection is used in the following form.



2.5.4 FB_TCPRead (FB)

◆ Variables

Name	FB_TCPRead (Read function block)
Graphical representation	ST representation
	FB_TCPRead (hHandle:=, xEnable:= , szSize:=, Data:=, xBusy=> , xError=> , eError=> , xReady=> , szCount=>);

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
hHandle	Handle	hHandle			Accepts the handle created by the client or connection function block.
xEnable	Function block enable.	BOOL	TRUE, FALSE	FALSE	
szSize	Receive data length.	_XINT			Equivalent to DINT in 32-bit systems, LINT in 64-bit systems. Must not exceed the receive data buffer size.
Data	Receive data buffer.	ANY			

(2) Output variable

Output variable	Name	Data type	Valid range	Description
xBusy		BOOL	TRUE, FALSE	
xError		BOOL	TRUE, FALSE	
eError		ErrorID		Error code.
xReady		BOOL	TRUE, FALSE	Active when data is received, for one cycle only.
szCount		DINT		Length of the received data. Active when data is received, for one cycle only.

◆ Key points

- The Data pin can be directly assigned a sample data variable.

2.5.5 FB_TCPWrite (FB)

◆ Variables

Name	FB_TCPWrite (Write function block)
Graphical representation	ST representation
	FB_TCPWrite (hHandle:=, xExecute:= , szSize:= , Data:= , xDone=> , xBusy=> , xError=> , eError=>);

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
hHandle	Handle	hHandle			Accepts the handle created by the client or connection function block.
xExecute	Function block enable.	BOOL	TRUE, FALSE	FALSE	
szSize	Transmit data length.	_XINT			Equivalent to DINT in 32-bit systems, LINT in 64-bit systems. Must not exceed the transmit data buffer size.
Data	Transmit data buffer.	ANY			

(2) Output variable

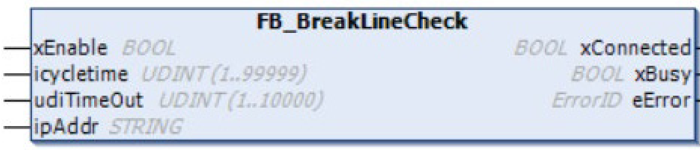
Output variable	Name	Data type	Valid range	Description
xDone		BOOL	TRUE, FALSE	When xExecute is not TRUE, it outputs a pulse for one cycle.
xBusy		BOOL	TRUE, FALSE	
xError		BOOL	TRUE, FALSE	
eError		ErrorID		Error code.

◆ Key points

- The Data pin can be directly assigned a sample data variable.

2.5.6 FB_BreakLineCheck (FB)

◆ Variables

Name	FB_BreakLineCheck (Connection break line detection function block)
Graphical representation	ST representation
	FB_BreakLineCheck (xEnable:= , icycletime:= , udiTimeOut:= , ipAddr:= , xConnected=> , xBusy=> , eError=>);

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
xEnable	Function block enable.	BOOL	TRUE, FALSE	FALSE	
icycletime	Detection trigger cycle interval count.	UDINT	1-99999	10	The cycle count of the current task period.
udiTimeOut	Connection timeout duration.	UDINT	1-99999	10	Timeout period.
ipAddr	Connected IP address.	STRING		'192.168.88.100'	

(2) Output variable

Output variable	Name	Data type	Valid range	Description
xConnected	Connection status.	BOOL	TRUE, FALSE	Outputs status at the interval set by icycletime.
xBusy		BOOL	TRUE, FALSE	
eError		ErrorID		Error code.

◆ Key points

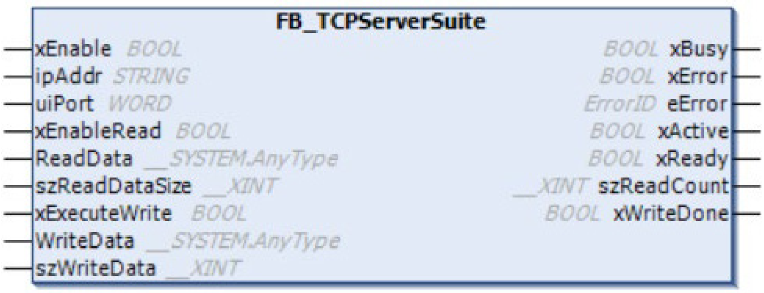
・This function block can only be used for connection break line detection, not for heartbeat detection.

・The icycletime pin value is the number of task cycle intervals. The task cycle time is the period of the task in which this function block resides.

・It is not recommended to place this function block in a low-priority task cycle, a short task cycle, or with a very short interval. Otherwise, the load rate may increase, affecting communication.

2.5.7 FB_TCPServerSuite (FB)

◆ Variables

Name	FB_TCPServerSuite (Server suite function block)
Graphical representation	ST representation
	FB_TCPServerSuite (xEnable:= , ipAddr:= , uiPort:= , xEnableRead:= , ReadData:= , szReadDataSize:= , xExecuteWrite:= , WriteData:= , szWriteData:= , xBusy=> , xError=> , eError=> , xActive=> , xReady=> , szReadCount=> , xWriteDone=>);

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
xEnable	Function block enable	BOOL	TRUE, FALSE	FALSE	
ipAddr	Server IP address	STRING		0	
uiPort	Server port number	WORD		0	
xEnableRead	Function block enable	BOOL	TRUE, FALSE	FALSE	
ReadData	Receive data buffer	ANY			
szReadDataSize	Receive data length	_XINT			Equivalent to DINT in 32-bit systems, LINT in 64-bit systems. Must not exceed the receive data buffer size.
xExecuteWrite	Function block enable	BOOL	TRUE, FALSE	FALSE	
WriteData	Transmit data buffer	ANY			
szWriteData	Transmit data length	_XINT			Equivalent to DINT in 32-bit systems, LINT in 64-bit systems. Must not exceed the transmit data buffer size.

(2) Output variable

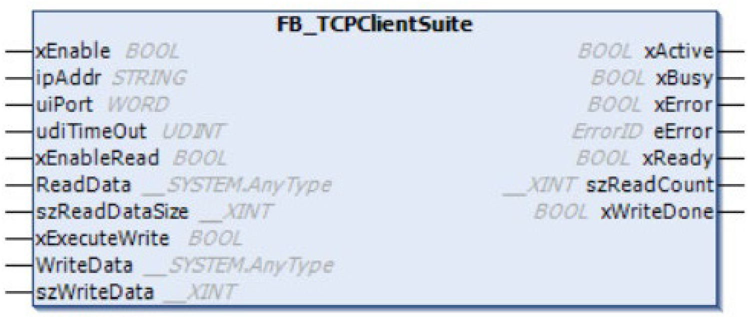
Output variable	Name	Data type	Valid range	Description
xBusy		BOOL	TRUE, FALSE	
xError		BOOL	TRUE, FALSE	
eError		ErrorID		Error code.
xActive		BOOL	TRUE, FALSE	Connection established pin.
xReady	Receive status	BOOL	TRUE, FALSE	Active when data is received, for one cycle only.
szReadCount	Received data length	DINT		Active when data is received, for one cycle only.
xWriteDone		BOOL	TRUE, FALSE	When xExecute is not TRUE, it outputs a pulse for one cycle.

◆ Key points

- This is a function block integrating FB_TCPServer, FB_TCPConnection, FB_TCPRead, and FB_TCPWrite.

2.5.8 FB_TCPClientSuite (FB)

◆ Variables

Name	FB_TCPClientSuite (Client suite function block)
Graphical representation	ST representation
 	FB_TCPClientSuite (xEnable:= , ipAddr:= , uiPort:= , udiTimeOut:= , xEnableRead:= , ReadData:= , szReadDataSize:= , xExecuteWrite:= , WriteData:= , szWriteData:= , xActive=> , xBusy=> , xError=> , eError=> , xReady=> , szReadCount=> , xWriteDone=>);

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
xEnable	Function block enable	BOOL	TRUE, FALSE	FALSE	
udiTimeOut	Handle	UDINT		20	Unit: ms. Connection timeout duration, affected by the task cycle in which the function block resides.
ipAddr	Server IP address	STRING		0	
uiPort	Server port number	WORD		0	
xEnableRead	Function block enable	BOOL	TRUE, FALSE	FALSE	
ReadData	Receive data buffer	ANY			
szReadDataSize	Receive data length	_XINT			Equivalent to DINT in 32-bit systems, LINT in 64-bit systems. Must not exceed the receive data buffer size.
xExecuteWrite	Function block enable	BOOL	TRUE, FALSE	FALSE	
WriteData	Transmit data buffer	ANY			
szWriteData	Transmit data length	_XINT			Equivalent to DINT in 32-bit systems, LINT in 64-bit systems. Must not exceed the transmit data buffer size.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
xActive		BOOL	TRUE, FALSE	Connection established pin.
xBusy		BOOL	TRUE, FALSE	
xError		BOOL	TRUE, FALSE	
eError		ErrorID		Error code.
xReady	Receive status	BOOL	TRUE, FALSE	Active when data is received, for one cycle only.
szReadCount	Received data length	DINT		Active when data is received, for one cycle only.
xWriteDone		BOOL	TRUE, FALSE	When xExecute is not TRUE, it outputs a pulse for one cycle.

◆ Key points

- This is a function block integrating FB_TCPClient, FB_TCPRead, and FB_TCPWrite.

Chapter 3 DataProcess (Data Processing)

3.1 StreamProcess (Function group)

A function group for stream processing. It is used to retrieve data of the required type from a continuous data stream (e.g., an array, a string) or to write data of a corresponding type into a data stream. Note: This function does not perform any check on the remaining length of the data stream. Ensure that the data length pointed to by the data stream pointer is sufficient for retrieving or writing the corresponding data type during use.

3.1.1 GET (Get data)

Name	Description
Get_DINT (FUN)	Convert BYTE to DINT
Get_INT (FUN)	Convert BYTE to INT
Get_LREAL(FUN)	Convert BYTE to LREAL
Get_REAL(FUN)	Convert BYTE to REAL
Get_UDINT (FUN)	Convert BYTE to UDINT
Get_UINT (FUN)	Convert BYTE to UINT
Get_USINT (FUN)	Convert BYTE to USINT

Function form:

Get_XXXX(pData:=);

pData is a pointer to the start address of a BYTE array.

3.1.2 SET (Write data)

Name	Description
Set_DINT (FUN)	Convert DINT to BYTE and write into a BYTE array
Set_INT (FUN)	Convert INT to BYTE and write into a BYTE array
Set_LREAL(FUN)	Convert LREAL to BYTE and write into a BYTE array
Set_REAL(FUN)	Convert REAL to BYTE and write into a BYTE array
Set_UDINT (FUN)	Convert UDINT to BYTE and write into a BYTE array
Set_UINT (FUN)	Convert UINT to BYTE and write into a BYTE array
Set_USINT (FUN)	Convert USINT to BYTE and write into a BYTE array

Function form:

Set_XXXX(in:= ,pData:=);

In is the value to be written.

pData is a pointer to the start address of a BYTE array.

◆ Key points

· Integer data conversion simply follows the radix conversion rules.

· The conversion between floating-point data and BYTE must adhere to floating-point normalization. Examples are provided below.

3.1.3 Usage example 1 (Integer data conversion)

【1】Get data

Declare some integer data variables to hold the function output values. Declare a 4-byte BYTE array to act as the data stream.

```
PROGRAM PLC_PRG
VAR
    testbyte    :ARRAY [0..3] OF BYTE;

    getDint     :DINT;
    getInt      :INT;
    getUdint    :UDINT;
    getUint     :UINT;
    getUsint    :USINT;
```

```
getDint :=Get_DINT(pData:= ADR(testbyte));
getInt  :=Get_INT (pData:=ADR(testbyte));
getUdint:=Get_UDINT(pData:=ADR(testbyte));
getUint :=Get_UINT(pData:=ADR(testbyte));
getUsint:=Get_USINT(pData:=ADR(testbyte));
```

After logging into the program, for ease of understanding, the BYTE array values are displayed and modified in binary format to 2#00010000 00000010 00001000 00011000 (in the array, the byte at a later address is the higher-order byte). It can be observed that different functions retrieve values from the array based on the space size they require.

表达式	类型	值
testbyte	ARRAY [0..3] OF BYTE	
testbyte[0]	BYTE	2#00011000
testbyte[1]	BYTE	2#00001000
testbyte[2]	BYTE	2#00000010
testbyte[3]	BYTE	2#00010000
getDint	DINT	2#00010000000000100000100000011000
getInt	INT	2#0000100000011000
getUdint	UDINT	2#00010000000000100000100000011000
getUint	UINT	2#0000100000011000
getUsint	USINT	2#00011000

【2】Write data

Declare some integer data variables to hold the values to be written by the function. Declare a 4-byte BYTE array to act as the data stream.

PLC_PRG | 库管理器 | Device

```
PROGRAM PLC_PRG
VAR
    testbyte     :ARRAY [0..3] OF BYTE;

    dintnum   :DINT;
    intnum    :INT;
    udintnum  :UDINT;
    uintnum   :UINT;
    usintnum  :USINT;

Set_DINT (in:= dintnum, pData:=ADR(testbyte));
Set_INT  (in:= intnum, pData:=ADR(testbyte));
Set_UDINT(in:= udintnum, pData:=ADR(testbyte));
Set_UINT (in:= uintnum, pData:=ADR(testbyte));
Set_USINT(in:= usintnum, pData:=ADR(testbyte));
```

Since writing data to the same array causes mutual interference, other functions are commented out here, demonstrating only writing DINT data. DINT is a 32-bit signed integer. The value of dintnum is modified to -158, which is 2#11111111 11111111 11111111 01100010 (two's complement).

PLC_PRG | 库管理器 | Device

Device.Application.PLC_PRG

表达式	类型	值	准备值
testbyte	ARRAY [0..3] OF BYTE		
testbyte[0]	BYTE	0	
testbyte[1]	BYTE	0	
testbyte[2]	BYTE	0	
testbyte[3]	BYTE	0	
dintnum	DINT	0	-158
intnum	INT	0	
udintnum	UDINT	0	
uintnum	UINT	0	
usintnum	USINT	0	
getDint	DINT	0	
getInt	INT	0	
getUdint	UDINT	0	
getUint	UINT	0	
getUsint	USINT	0	
testreal	ARRAY [0..3] OF BYTE		
testlreal	ARRAY [0..7] OF BYTE		

It can be seen that the values of the BYTE array testbyte have been modified.

PLC_PRG | 库管理器 | Device

Device.Application.PLC_PRG

表达式	类型	值
testbyte	ARRAY [0..3] OF BYTE	
testbyte[0]	BYTE	2#01100010
testbyte[1]	BYTE	2#11111111
testbyte[2]	BYTE	2#11111111
testbyte[3]	BYTE	2#11111111
dintnum	DINT	2#11111111111111111111111101100010

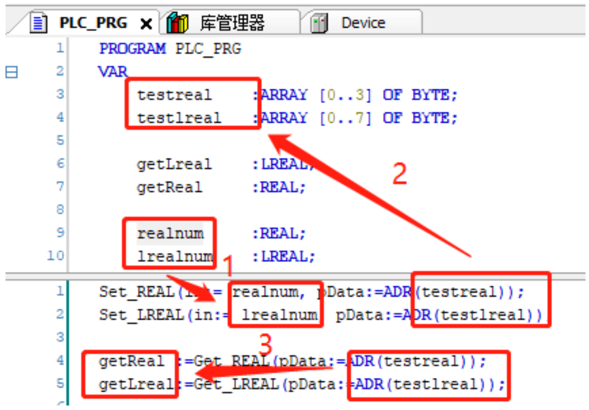
3.1.4 Usage example 2 (Floating-point data conversion)

For ease of understanding, the floating-point example first introduces writing data. This example will explain in detail the conversion method between floating-point numbers and binary.

Declare floating-point data as input for the write functions. Map the BYTE array address to the corresponding function output pin. (Ensure the BYTE array has sufficient space).

【1-2】First, use the write functions to write the values realnum and lrealnum into the array testxxxx.

【3】Then, use the read functions to read the values from array testxxxx into getreal and getlreal.



Example: Convert the real value 2.5 to binary representation (32-bit float composition: 1-bit sign, 8-bit exponent, 23-bit mantissa).

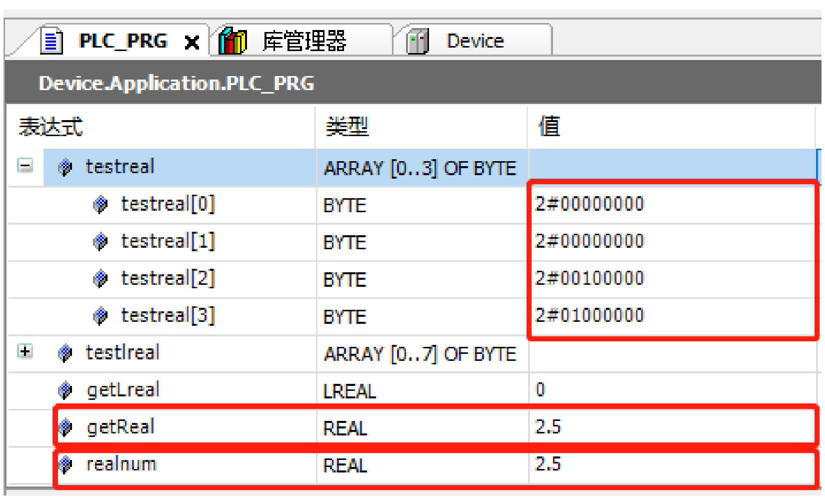
(1) Separate integer and fractional parts. Convert the decimal number 2.5 to binary: 2#10.1

(2) Shift 10.1 to the left to get 1.01*2^1.

Therefore, the following is obtained:

- Sign bit (S): 0 (positive number)
- Exponent (E): Should be 127 + (1) = 128, therefore, binary representation: 10000000.
- Mantissa (M): After padding to 23 bits: 01000000000000000000000.

The final 32-bit floating-point number in binary is: 2# 0 10000000 01000000000000000000000


PLC_PRG | 库管理器 | Device

Device.Application.PLC_PRG

表达式	类型	值
testreal	ARRAY [0..3] OF BYTE	
testreal[0]	BYTE	2#00000000
testreal[1]	BYTE	2#00000000
testreal[2]	BYTE	2#00100000
testreal[3]	BYTE	2#01000000
testlreal	ARRAY [0..7] OF BYTE	
getLreal	LREAL	0
getReal	REAL	2.5
realnum	REAL	2.5

It can be seen that the value obtained in the array is 2#01000000 00100000 00000000 00000000 (note: high/low byte order is reversed), which matches the calculation result. The value obtained by getReal is 2.5 (the inverse process is not exemplified; simply reverse the above process).

Example: Convert the lreal value 300 to binary representation (64-bit double precision composition: 1-bit sign, 11-bit exponent, 52-bit mantissa).

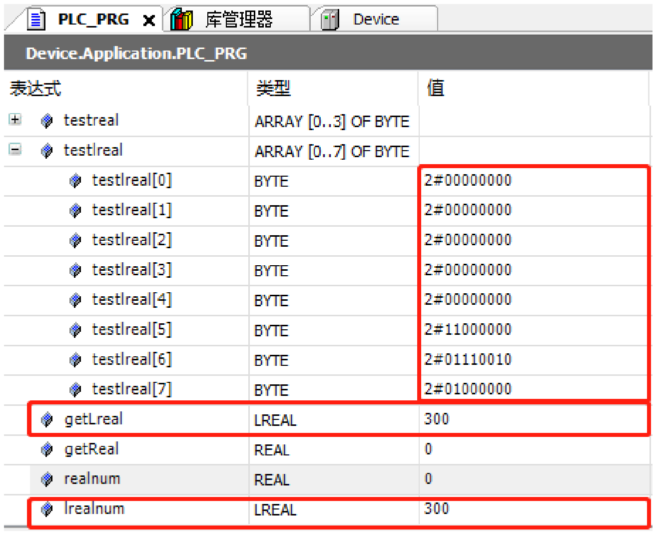
(3) Separate integer and fractional parts. Convert the decimal number 300 to binary: 2#00100101100

(4) Shift 00100101100 to the left to get 1.001011 * 2^8.

Therefore, the following is obtained:

- Sign bit (S): 0 (positive number)
- Exponent (E): Should be 1023 + (8) = 1031, therefore, binary representation: 2#10000000111.
- Mantissa (M): After padding to 52 bits: 00101100000000............

The final 64-bit floating-point number in binary is: 2# 0 10000000111 00101100000000.........

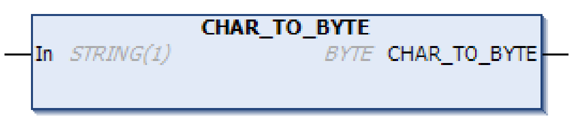


3.2 Type Convert (Function group)

A function group for type conversion, containing various data type conversions.

3.2.1 CHAR_TO_BYTE (FUN)

◆ Variables

Name	CHAR_TO_BYTE (Character to numeric value)		
Supported modes	CSP	CSV	CST
Graphical representation		ST representation	
		CHAR_TO_BYTE(In:=);	

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
IN	Input string	STRING (1)			Pointer to the device information structure.

(2) Output variable

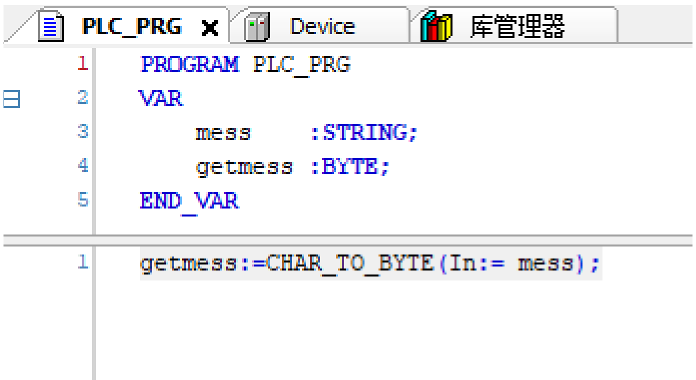
Output variable	Name	Data type	Valid range	Description
CHAR_TO_BYTE	Conversion result	BYTE		The BYTE value converted from the character.

◆ Key points

- Refer to the ASCII code table for the conversion relationship between strings and bytes.

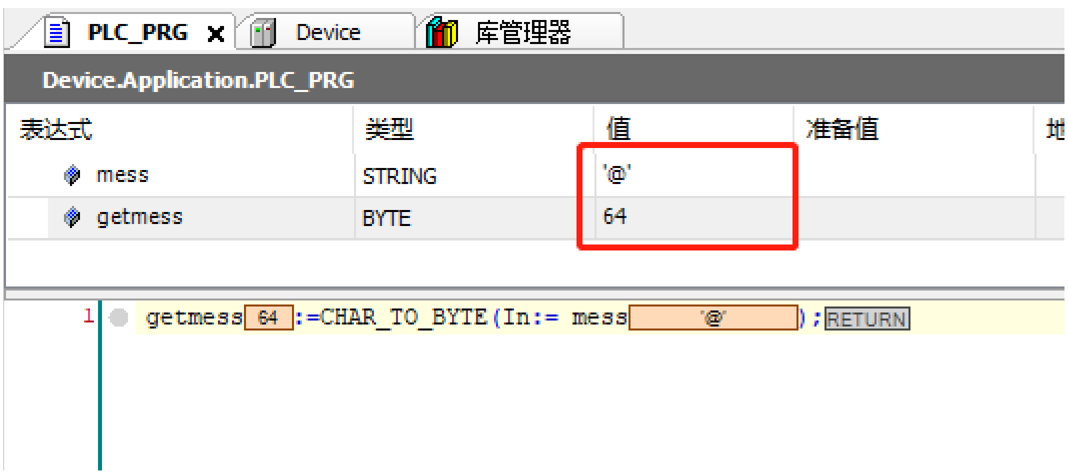
3.2.2 Usage example

【1】Declare the following two variables. Call the function to convert mess (STRING) to BYTE, and assign the output result to getmess (BYTE).



【2】Download and run the program.

Modify the value of mess to '@'. It can be seen that the value of getmess changes to 64, which corresponds to the conversion value in the ASCII code table.



3.3 Type Packing (Function group)

A function group for data packing/unpacking. It can combine or convert types like BYTE, BIT, etc., to/from other types.

3.3.1 Function group (Packing)

Name	Description
BOOL_PACK_BYTE (FUN)	Pack BOOLs into BYTE
BYTE_PACK_DINT (FUN)	Pack BYTEs into DINT
BYTE_PACK_INT (FUN)	Pack BYTEs into INT
BYTE_PACK_LREAL (FUN)	Pack BYTEs into LREAL
BYTE_PACK_REAL (FUN)	Pack BYTEs into REAL
BYTE_PACK_UDINT (FUN)	Pack BYTEs into UDINT
BYTE_PACK_UINT (FUN)	Pack BYTEs into UINT

3.3.2 Function group (UnPacking)

Name	Description
BYTE_UNPACK_BOOL (FUN)	Unpack BYTE into BOOLs
DINT_UNPACK_BYTE (FUN)	Unpack DINT into BYTEs
INT_UNPACK_BYTE (FUN)	Unpack INT into BYTEs
LREAL_UNPACK_BYTE (FUN)	Unpack LREAL into BYTEs
REAL_UNPACK_BYTE (FUN)	Unpack REAL into BYTEs
UDINT_UNPACK_BYTE (FUN)	Unpack UDINT into BYTEs
UINT_UNPACK_BYTE (FUN)	Unpack UINT into BYTEs

◆ **Key points**

・ This function group provides data packing/unpacking functions for different data types, similar to the StreamProcess function group.

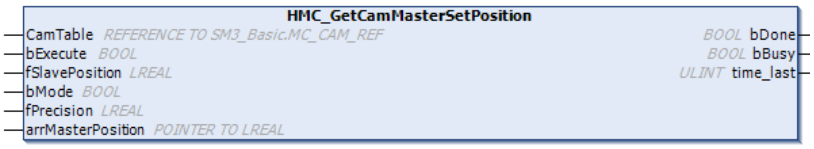
Chapter 4 Motion (Motion Control)

4.1 GetCamPosition

4.1.1 HMC_GetCamMasterSetPosition (FB)

This function block is used to calculate the master axis position based on an input slave axis position.

◆ Variables

<table>
<tr><th>Name</th><th colspan="3">HMC_GetCamMasterSetPosition</th></tr>
<tr><td>Supported modes</td><td>CSP</td><td>CSV</td><td>CST</td></tr>
<tr><td colspan="2">Graphical representation</td><td colspan="2">ST representation</td></tr>
<tr><td colspan="2">
</td><td colspan="2">HMC_GetCamMasterSetPosition(
CamTable:= ,
bExecute:= ,
fSlavePosition:= ,
bMode:= ,
fPrecision:= ,
arrMasterPosition:= ,
bDone=> ,
bBusy=> ,
time_last=>);</td></tr>
</table>

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
CamTable	Cam table	REFERENCE TO SM3_Basic.MC_CAM_REF			Input cam table. Only polynomial mode is supported.
bExecute	Enable	BOOL	TRUE, FALSE	FALSE	TRUE: Triggers the function.
fSlavePosition	Slave position	LREAL			Slave axis position.
bMode	Calculation mode	BOOL	TRUE, FALSE	FALSE	TRUE: Calculates one solution per cycle. FALSE: Outputs all solutions in one cycle.
fPrecision	Precision	LREAL		0.0001	Master position calculation precision. Unit: Application unit.
arrMasterPosition	Master position	POINTER TO LREAL			Array for storing the output master axis positions. Outputs multiple solutions when there are multiple matching ones.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
bDone	Done	BOOL	TRUE, FALSE	TRUE: Function block execution is complete.
bBusy	Busy	BOOL	TRUE, FALSE	TRUE: Function block is executing.
time_last	Calculation time	ULINT		Time taken for calculation. Unit: μs.

◆ Key points

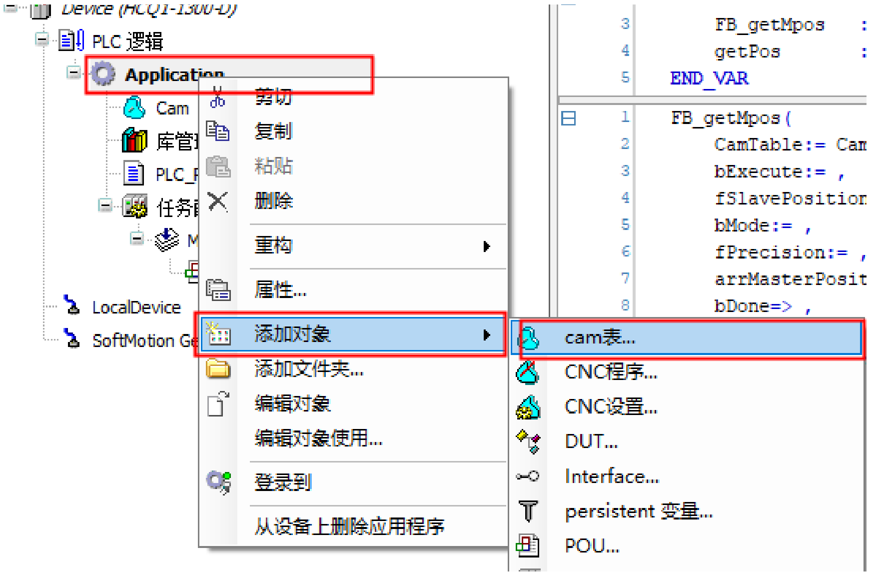
・The calculation mode can be set to TRUE to reduce fluctuations in the task cycle. However, calculating the points will require multiple cycles.

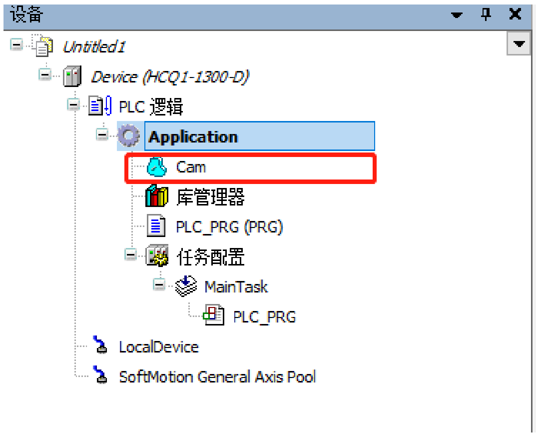
・arrMasterPosition is a pointer to the start address of an LREAL array, used to store multiple LREAL type data (master axis positions).

◆ Usage example

【1】Create a new cam table (polynomial).

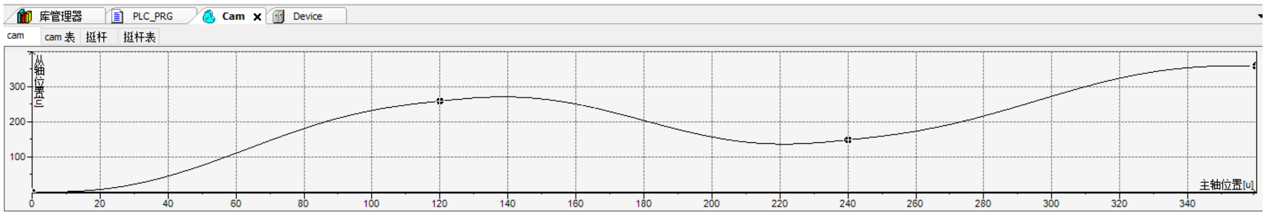
Right-click [Application] -> [Add Object] -> [Cam Table...] to add a cam table, named Cam here.





【2】Modify the master-slave position correspondence in the Cam table. The demonstration Cam is shown below.

(This table is for demonstrating this function block's calculation only.)



【3】Declare and call the function block. Declare an LREAL array named getPos to receive the master axis positions calculated by the function block.

```
PROGRAM PLC_PRG
VAR
    FB_getMpos    :HMC_GetCamMasterSetPosition;
    getPos        :ARRAY [0..5] OF LREAL;
END_VAR
```

```
FB_getMpos(
    CamTable:= Cam,
    bExecute:= ,
    fSlavePosition:= ,
    bMode:= ,
    fPrecision:= ,
    arrMasterPosition:= getPos,
    bDone=> ,
    bBusy=> ,
    time_last=> );
```

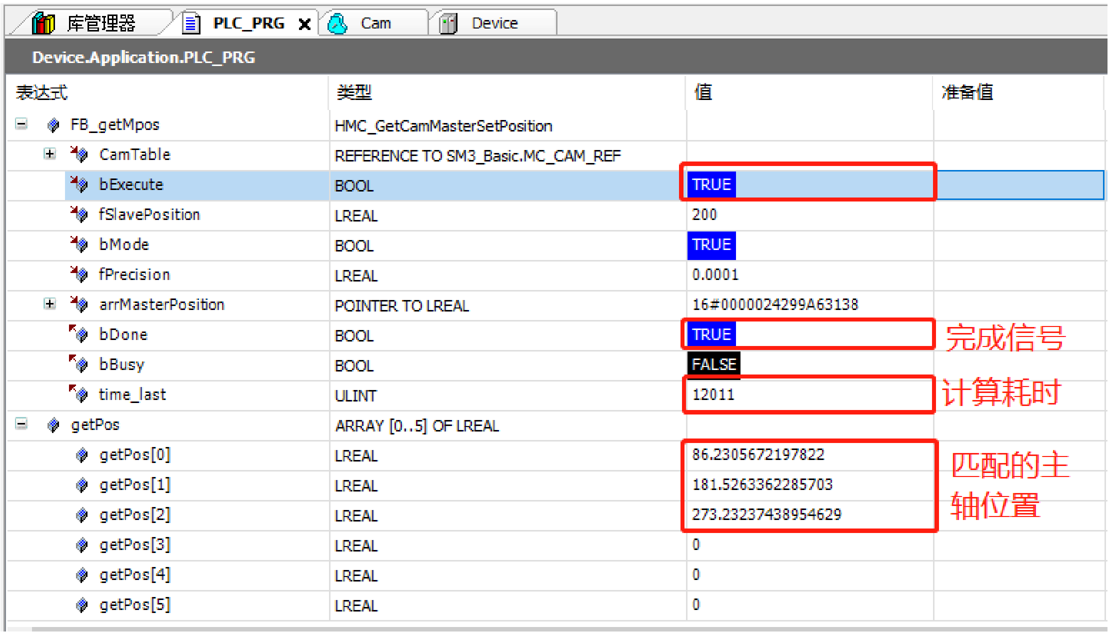
【4】Use the function block

This example demonstrates calculating the master axis position when the slave axis position is 200, with the calculation mode set to one solution per cycle.

Device.Application.PLC_PRG

表达式	类型	值	准备值
FB_getMpos	HMC_GetCamMasterSetPosition		
CamTable	REFERENCE TO SM3_Basic.MC_CAM_REF		
bExecute	BOOL	FALSE	
fSlavePosition	LREAL	0	200
bMode	BOOL	FALSE	TRUE
fPrecision	LREAL	0.0001	
arrMasterPosition	POINTER TO LREAL	16#0000024299A63138	
bDone	BOOL	FALSE	
bBusy	BOOL	FALSE	
time_last	ULINT	0	
getPos	ARRAY [0..5] OF LREAL		
getPos[0]	LREAL	0	
getPos[1]	LREAL	0	
getPos[2]	LREAL	0	
getPos[3]	LREAL	0	
getPos[4]	LREAL	0	
getPos[5]	LREAL	0	

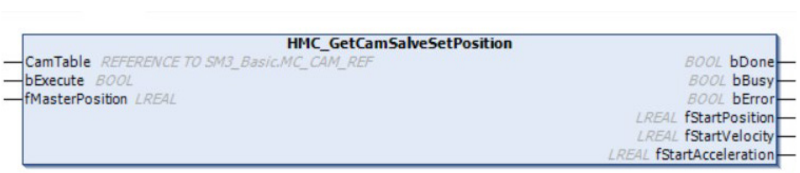
After triggering the function block, it calculates and outputs all matching master axis positions. Simultaneously, the bDone pin outputs TRUE, and the time_last pin outputs the time used for the calculation.



4.1.2 HMC_GetCamSalveSetPosition (FB)

This function block is used to calculate the slave axis position, velocity, and acceleration based on an input master axis position.

◆ Variables

<table>
<tr><th>Name</th><th colspan="3">HMC_GetCamMasterSetPosition</th></tr>
<tr><td>Supported modes</td><td>CSP</td><td>CSV</td><td>CST</td></tr>
<tr><td colspan="2">Graphical representation</td><td colspan="2">ST representation</td></tr>
<tr><td colspan="2"></td><td colspan="2">HMC_GetCamSalveSetPosition(
CamTable:= ,
bExecute:= ,
fMasterPosition:= ,
bDone=> ,
bBusy=> ,
bError=> ,
fStartPosition=> ,
fStartVelocity=> ,
fStartAcceleration=>);</td></tr>
</table>

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
CamTable	Cam table	REFERENCE TO SM3_Basic.MC_CAM_REF			Input cam table. Only polynomial mode is supported.
bExecute	Enable	BOOL	TRUE, FALSE	FALSE	TRUE: Triggers the function.
fMasterPosition	Master position	LREAL			Master axis position.

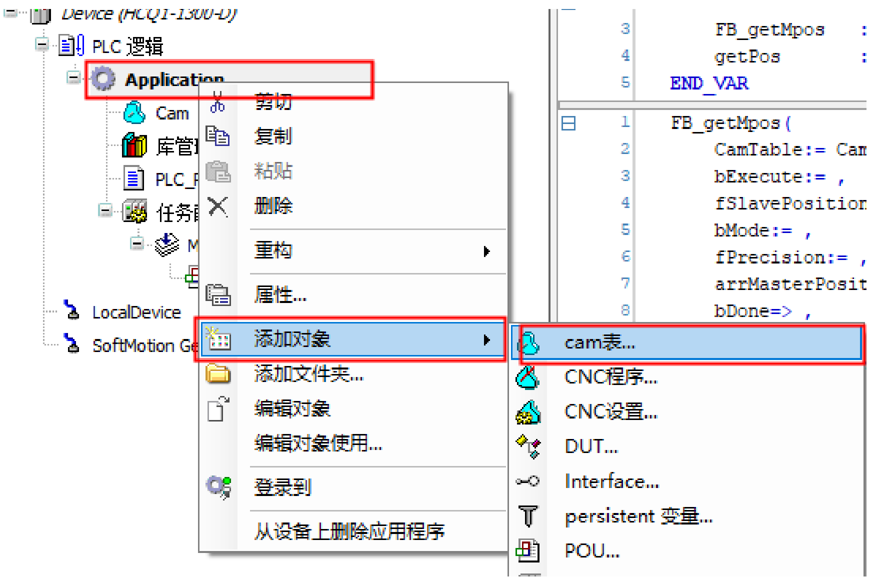
(2) Output variable

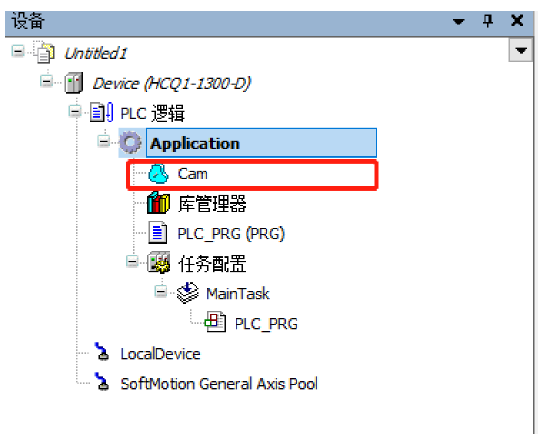
Output variable	Name	Data type	Valid range	Description
bDone	Done	BOOL	TRUE, FALSE	TRUE: Function block execution is complete.
bBusy	Busy	BOOL	TRUE, FALSE	TRUE: Function block is executing.
bError	Error	BOOL	TRUE, FALSE	TRUE: An error occurred in the function block.
fStartPosition	Slave position	LREAL		Slave position.
fStartVelocity	Slave velocity	LREAL		Slave velocity.
fStartAcceleration	Slave acceleration	LREAL		Slave acceleration.

◆ Usage example

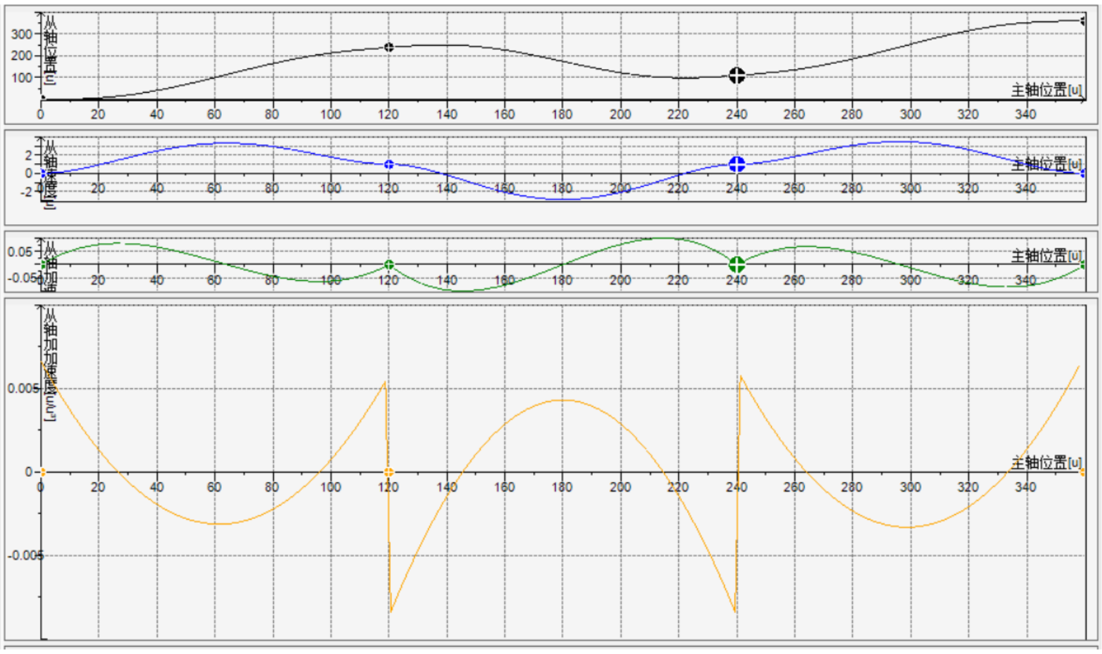
【1】Create a new cam table (polynomial)

Right-click [Application] -> [Add Object] -> [Cam Table...] to add a Cam table, named Cam here.



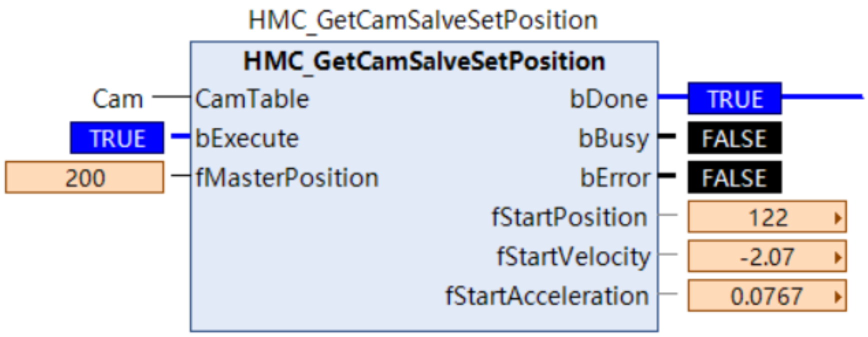


【2】Modify the master-slave position correspondence in the Cam table. The demonstration Cam is shown below. (This table is for demonstrating this function block's calculation only.)



【3】Declare and call the function block. Input a master axis position to calculate the corresponding slave axis position, velocity, and acceleration.

This example demonstrates calculating the slave axis parameters when the master axis position is 200.



4.2 GetVitualAxis

4.2.1 FB_CreatVitualAxis (FB)

This function block is used to create a virtual axis.

◆ Variables

<table>
<tr><th>Name</th><th colspan="3">FB_CreatVitualAxis</th></tr>
<tr><td>Supported modes</td><td>CSP</td><td>CSV</td><td>CST</td></tr>
<tr><td colspan="2">Graphical representation</td><td colspan="2">ST representation</td></tr>
<tr><td colspan="2"></td><td colspan="2">FB_CreatVitualAxis(
bCreateEx:= ,
bModulo:= ,
fModuloPeriod:= ,
bDone=> ,
pAxis=> ,
bError=>);</td></tr>
</table>

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bCreateEx	Trigger pin	BOOL			Rising edge trigger.
bModulo	Axis modality selection	BOOL	TRUE, FALSE	FALSE	TRUE: Modal axis. FALSE: Linear axis.
fModuloPeriod	Modulo period	REAL			Period for the modal axis.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
bDone	Done	BOOL	TRUE, FALSE	TRUE: Function block execution is complete.
pAxis	Output axis pointer	POINTER TO AXIS_REF_ VIRTUAL_SM3		Pointer to the generated virtual axis.
bError	Error	BOOL	TRUE, FALSE	TRUE: Creation failed, please check.

◆ Key points

・ The calculation mode can be set to TRUE to reduce fluctuations in the task cycle. However, calculating the points will require multiple cycles.

・ arrMasterPosition is a pointer to the start address of an LREAL array, used to store multiple LREAL type data (master axis positions).

◆ Usage example

After declaring the function block, call and configure it in the program.

```
(*生成虚轴,参数无误情况下，bCreateEx为TRUE后，bDone信号随即
给出，pAxis即为指向生成的虚轴的指针*)
FB_CreatVitualAxis(
    bCreateEx:=     TRUE,
    bModulo:=       bOn模态Off线性,
    fModuloPeriod:= f模态周期,
    bDone=> ,
    pAxis=> ,
    bError=> );
//---在所有轴控指令执行之前，调用m_Start()函数
FB_CreatVitualAxis.m_Start();
IF FB_CreatVitualAxis.bDone THEN    //---轴控功能块调用
    MC_Power(
        Axis:= FB_CreatVitualAxis.pAxis^,
        Enable:= TRUE,
        bRegulatorOn:= ,
        bDriveStart:= TRUE,
        Status=> ,
        bRegulatorRealState=> ,
        bDriveStartRealState=> ,
        Busy=> ,
        Error=> ,
        ErrorID=> );
    MC_Jog(
        Axis:= FB_CreatVitualAxis.pAxis^,
        JogForward:= bF,
        JogBackward:= bB,
        Velocity:= fVel,
        Acceleration:= fVel*10,
        Deceleration:= fVel*10,
        Jerk:= ,
        Busy=> ,
        CommandAborted=> ,
        Error=> ,
        ErrorId=> );
END_IF
//---在所有轴控指令之后调用m_End()函数;
FB_CreatVitualAxis.m_End();
```

4.3 HMC_GearIn

4.3.1 HMC_ActGearIn (FB)

Electronic gearing. This function block is used to couple and follow the master axis feedback position.

◆ **Variables**

<table>
<tr><th>Name</th><th colspan="3">HMC_ActGearIn</th></tr>
<tr><td>Supported modes</td><td>CSP</td><td>CSV</td><td>CST</td></tr>
<tr><td colspan="2">Graphical representation</td><td colspan="2">ST representation</td></tr>
<tr><td colspan="2">
</td><td colspan="2">HMC_ActGearIn(
Master:= ,
Slave:= ,
Execute:= ,
RatioNumerator:= ,
RatioDenominator:= ,
Acceleration:= ,
Deceleration:= ,
Jerk:= ,
InGear=> ,
Busy=> ,
CommandAborted=> ,
Error=> ,
ErrorID=>);</td></tr>
</table>

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bExecute	Enable	BOOL	TRUE, FALSE	FALSE	TRUE: Triggers the function
RatioNumerator	Ratio numerator	LREAL			Ratio numerator.
RatioDenominator	Ratio denominator	LREAL			Ratio denominator.
Acceleration	Acceleration	LREAL			
Deceleration	Deceleration	LREAL			
Jerk	Jerk	LREAL			

(2) Output variable

Output variable	Name	Data type	Valid range	Description
InGear	Gear in	BOOL	TRUE, FALSE	TRUE: Coupling is complete.
bBusy	Busy	BOOL	TRUE, FALSE	TRUE: Function block is executing.
CommandAborted	Command aborted	BOOL	TRUE, FALSE	TRUE: Function block execution is aborted.
Error	Error	BOOL	TRUE, FALSE	
ErrorID	Error ID	SMC_ERROR		

(3) Input/Output variable

Output variable	Name	Data type	Valid range	Description
Master	Master axis	AXIS_REF_SM3		Master axis
Slave	Slave axis	AXIS_REF_SM3		Slave axis

◆ **Key points**

- Couples and follows the master axis feedback position. The usage method is the same as MC_GearIn.

4.4 HMC_GearInMultiMaster

4.4.1 HMC_GearInMultiMaster (FB)

This function block is used to implement multi-master axis coupling.

◆ Variables

<table>
<tr><th>Name</th><th colspan="3">HMC_GetCamMasterSetPosition</th></tr>
<tr><td>Supported modes</td><td>CSP</td><td>CSV</td><td>CST</td></tr>
<tr><td colspan="2">Graphical representation</td><td colspan="2">ST representation</td></tr>
<tr><td colspan="2">
</td><td colspan="2">HMC_GearInMultiMaster(
Master1:= ,
Master2:= ,
Master3:= ,
Master4:= ,
Slave:= ,
SyncMode:= ,
Execute:= ,
GearRatio1:= ,
GearRatio2:= ,
GearRatio3:= ,
GearRatio4:= ,
Acceleration:= ,
Deceleration:= ,
Jerk:= ,
InGear=> ,
Busy=> ,
Active=> ,
CommandAborted=> ,
Error=> ,
ErrorID=>);</td></tr>
</table>

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
SyncMode	Synchronization mode	E_GearInMultiMasterSyncMode			Input cam table, only polynomial mode supported.
bExecute	Enable	BOOL	TRUE, FALSE	FALSE	TRUE: Triggers the function.
GearRatio1	Gear ratio 1	LREAL			
GearRatio2	Gear ratio 2	LREAL			
GearRatio3	Gear ratio 3	LREAL			
GearRatio4	Gear ratio 4	LREAL			
Acceleration	Acceleration	LREAL			
Deceleration	Deceleration	LREAL			
Jerk	Jerk	LREAL			

(2) Output variable

Output variable	Name	Data type	Valid range	Description
InGear	Gear synchronization	BOOL	TRUE, FALSE	TRUE: Function block gear synchronization is complete.

bBusy	Busy	BOOL	TRUE, FALSE	TRUE: Function block is executing.
Active	Active	BOOL	TRUE, FALSE	TRUE: Function block is active.
CommandAborted	Command aborted	BOOL	TRUE, FALSE	TRUE: Function block execution is aborted.
Error	Error	BOOL	TRUE, FALSE	TRUE: An error occurred in the function block.
ErrorID	Error ID	UDINT		

(3) Input/Output variable

Output variable	Name	Data type	Valid range	Description
Master1	Master axis 1	AXIS_REF_SM3		Master axis
Master2	Master axis 2	AXIS_REF_SM3		Master axis
Master3	Master axis 3	AXIS_REF_SM3		Master axis
Master4	Master axis 4	AXIS_REF_SM3		Master axis
Slave	Slave axis	AXIS_REF_SM3		Slave axis

◆ Key points

• E_GearInMultiMasterSyncMode.VELOSYNC, velocity synchronization. When coupling is complete, the slave axis velocity equals the master axis velocity multiplied by the ratio.

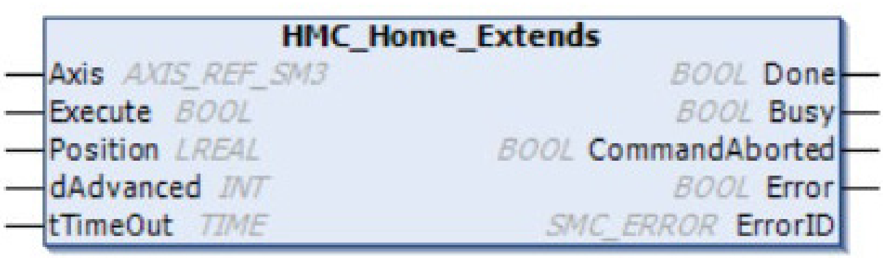
• E_GearInMultiMasterSyncMode.VELOSYNC, position and velocity synchronization. When coupling is complete, the slave axis velocity (position) equals the master axis velocity (position) multiplied by the ratio.

4.5 HMC_Home

4.5.1 HMC_Home_Extends (FB)

HCFA homing function block. This function block is used when homing issues occur with the standard MC_Home block.

◆ Variables

<table>
<tr><th>Name</th><th colspan="3">HMC_Home_Extends</th></tr>
<tr><td>Supported modes</td><td>CSP</td><td>CSV</td><td>CST</td></tr>
<tr><td colspan="2">Graphical representation</td><td colspan="2">ST representation</td></tr>
<tr><td colspan="2">
</td><td colspan="2">HMC_Home_Extends(
Axis:= ,
Execute:= ,
Position:= ,
Done=> ,
Busy=> ,
CommandAborted=> ,
Error=> ,
ErrorID=> ,
dAdvanced:= ,
tTimeOut:=);</td></tr>
</table>

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
Axis	Axis	XIS_REF_SM3			
Execute	Pin triggering	BOOL	TRUE, FALSE	FALSE	

Position	Position setting	LREAL		0	The absolute position where the signal is detected.
dAdvanced	Function mode selection	INT	0; 1; 2		0: Function is identical to MC_Home. 1: Attempt restart after timeout. 2: Reset first, then restart after timeout.
tTimeOut	Timeout	TIME		1S	Homing action timeout judgment.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
bDone	Done	BOOL	TRUE, FALSE	TRUE: Function block execution is complete.
Busy	Busy	BOOL	TRUE, FALSE	TRUE: Execution is in progress.
CommandAborted	Command aborted	BOOL	TRUE, FALSE	TRUE: Aborted by another command.
Error	Error	BOOL	TRUE, FALSE	TRUE: An error occurred.
ErrorID	Error ID	SMC_ERROR	0	Error ID

◆ Key points

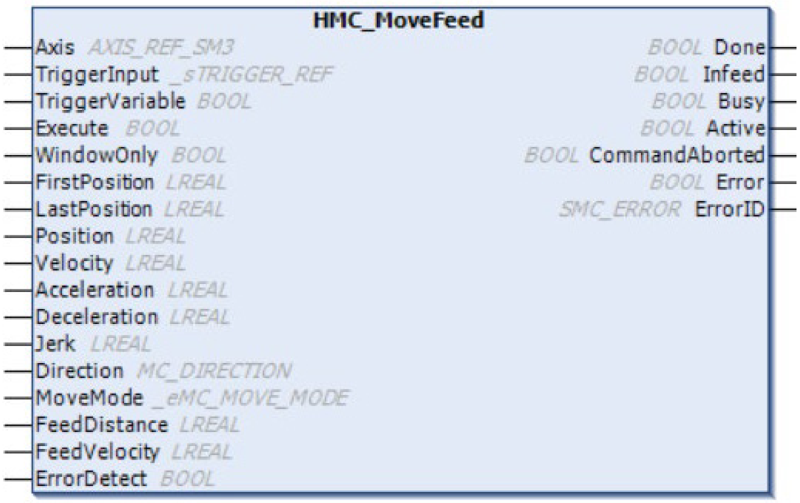
· Use in the same way as the MC_Home function block. Simply fill in the relevant parameters for "dAdvanced" and "tTimeOut".

4.6 OMRONMotion

4.6.1 HMC_MoveFeed (FB)

This function block performs positioning based on a specified travel distance from the point where an external interrupt input occurs.

◆ Variables

<table>
<tr><th>Name</th><th colspan="3">HMC_MoveFeed</th></tr>
<tr><td>Supported modes</td><td>CSP</td><td>CSV</td><td>CST</td></tr>
<tr><td colspan="2">Graphical representation</td><td colspan="2">ST representation</td></tr>
<tr><td colspan="2">
</td><td colspan="2">HMC_MoveFeed(
Axis:= ,
TriggerInput:= ,
TriggerVariable:= ,
Execute:= ,
WindowOnly:= ,
FirstPosition:= ,
LastPosition:= ,
Position:= ,
Velocity:= ,
Acceleration:= ,
Deceleration:= ,
Jerk:= ,
Direction:= ,
MoveMode:= ,
FeedDistance:= ,
FeedVelocity:= ,
ErrorDetect:= ,
Done=> ,
Infeed=> ,
Busy=> ,
Active=> ,
CommandAborted=> ,
Error=> ,
ErrorID=>);</td></tr>
</table>

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bExecute	Enable	BOOL	TRUE, FALSE	FALSE	TRUE: Triggers the function.
WindowOnly	Window only	BOOL			
FirstPosition	First position	LREAL			
LastPosition	Last position	LREAL			
Position	Target position	LREAL			
Velocity	Target velocity	LREAL			
Acceleration	Acceleration	LREAL			
Deceleration	Deceleration	LREAL			
Jerk	Jerk	LREAL			
Direction	Direction selection	MC_DIRECTION			

MoveMode	Move mode selection	_eMC_MOVE_MODE			Selects the move method. 0: Absolute positioning 1: Relative positioning 2: Velocity control.
FeedDistance	Feed distance	LREAL			Specifies the travel distance after the interrupt input. Set as positive for movement in the same direction as before the interrupt; set as negative for movement in the opposite direction.
FeedVelocity	Feed velocity	LREAL			Specifies the target velocity for movement after the interrupt input.
ErrorDetect	Error detection selection	BOOL			

(2) Output variable

Output variable	Name	Data type	Valid range	Description
Done	Done	BOOL	TRUE, FALSE	TRUE: Function block is complete.
Infeed	Infeed	BOOL	TRUE, FALSE	Receives a latch input, becomes TRUE during infeed.
bBusy	Busy	BOOL	TRUE, FALSE	TRUE: Function block is executing.
Active	Active	BOOL	TRUE, FALSE	TRUE: Function block is active.
CommandAborted	Command aborted	BOOL	TRUE, FALSE	TRUE: Function block execution is aborted.
Error	Error	BOOL	TRUE, FALSE	TRUE: An error occurred in the function block.
ErrorID	Error ID	UDINT		

(3) Input/Output variable

Output variable	Name	Data type	Valid range	Description
Axis	Axis	AXIS_REF_SM3		
TiggerInput	Trigger condition	_sTRIGGER_REF		
TiggerVariable	Trigger variable	BOOL		When specifying the controller mode under trigger conditions, specifies the input variable to trigger.

◆ Key points

- _eMC_TRIGGER_MODE (0: _mcDrive, 1: _mcController specifies the trigger mode. 0: Drive mode. 1: Controller mode.)
- Latch ID selection: _eMC_TRIGGER_LATCHID (0: _mcLatch1, 1: _mcLatch2. 0: Latch function 1. 1: Latch function 2.)
- PDO mapping: Latch Function (60B8h), Latch Status (60B9h), Latch Position 1 (60BAh), Latch Position 2 (60BCh).

• On the rising edge of Execute (Start), movement begins according to the setting of MoveMode (move mode selection), using one of the methods: absolute move, relative move, or velocity control.

• For absolute move, set the target position in Position (target position). For relative move, set the target distance in Position ((target position). For any move method, the move action is performed at the Velocity (target velocity).

• During movement, a relative positioning action occurs on the rising edge of the external input (interrupt input). From the feedback position, the axis moves the distance specified by FeedDistance (feed distance) at FeedVelocity (feed velocity).

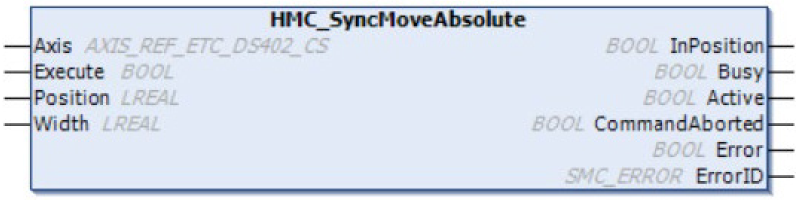
• When performing interrupt feed with an absolute or relative move command, if no interrupt signal is input before reaching the target position, the axis stops at the original target position. When stopping without an interrupt input, ErrorDetect (error detection selection) can specify whether to output an error. When error output is specified, CommandAborted (command aborted) becomes TRUE, and Busy (busy), Active (active) become FALSE.

• When using interrupt masking, set WindowOnly (window only) to TRUE and specify FirstPosition (first position) and LastPosition (last position). Interrupt feed positioning is executed by the first interrupt signal occurring while the feedback position is between FirstPosition (first position) and LastPosition (last position).

4.6.2 HMC_SyncMoveAbsolute (FB)

This function block outputs the specified target position for an axis on a per-cycle basis.

◆ Variables

<table>
<tr><th>Name</th><th colspan="3">HMC_SyncMoveAbsolute</th></tr>
<tr><td>Supported modes</td><td>CSP</td><td>CSV</td><td>CST</td></tr>
<tr><td colspan="2">Graphical representation</td><td colspan="2">ST representation</td></tr>
<tr><td colspan="2">
</td><td colspan="2">HMC_SyncMoveAbsolute(
Axis:= ,
Execute:= ,
Position:= ,
Width:= ,
InPosition=> ,
Busy=> ,
Active=> ,
CommandAborted=> ,
Error=> ,
ErrorID=>);</td></tr>
</table>

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bExecute	Enable	BOOL	TRUE, FALSE	FALSE	TRUE: Triggers the function.
Position		LREAL			
Width		LREAL			

(2) Output variable

Output variable	Name	Data type	Valid range	Description
InPosition	In position	BOOL	TRUE, FALSE	TRUE: Function block positioning is complete.
bBusy	Busy	BOOL	TRUE, FALSE	TRUE: Function block is executing.
Active	Active	BOOL	TRUE, FALSE	TRUE: Function block is active.
CommandAborted	Command aborted	BOOL	TRUE, FALSE	TRUE: Function block execution is aborted.
Error	Error	BOOL	TRUE, FALSE	TRUE: An error occurred in the function block.
ErrorID	Error ID	SMC_ERROR		

(3) Input/Output variable

Output variable	Name	Data type	Valid range	Description
Axis	Axis	AXIS_REF_SM3		

◆ Key points

• This command outputs the target position given by the user program to the servo drive, etc., in Cycle Synchronous Position (CSP) mode according to the task cycle. The target position is specified as an absolute position.

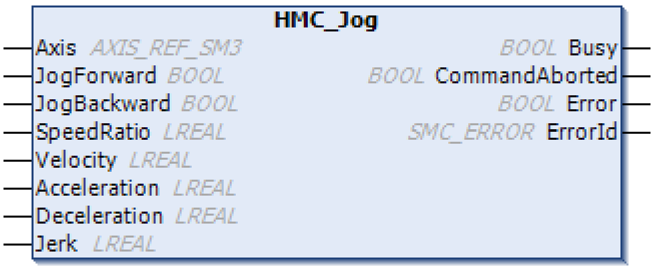
• When Position (target position) is not updated, InPosition (in position) becomes TRUE if the difference between the target position and the feedback position is within the axis parameter [In-position width].

4.7 OverrideVel

4.7.1 HMC_Jog (FB)

This function block is used for variable-speed jogging by changing the speed ratio.

◆ Variables

<table>
<tr><th>Name</th><th colspan="3">HMC_Jog (Variable-speed jog)</th></tr>
<tr><td>Supported modes</td><td>CSP</td><td>CSV</td><td>CST</td></tr>
<tr><td colspan="2">Graphical representation</td><td colspan="2">ST representation</td></tr>
<tr><td colspan="2">
</td><td colspan="2">HMC_Jog(
Axis:= ,
JogForward:= ,
JogBackward:= ,
SpeedRatio:= ,
Velocity:= ,
Acceleration:= ,
Deceleration:= ,
Jerk:= ,
Busy=> ,
CommandAborted=> ,
Error=> ,
ErrorId=>);</td></tr>
</table>

(1) Input/Output variable

Input/Output variable	Name	Data type	Description
Axis	Axis	AXIS_REF_SM3	Specifies the axis, an instance of AXIS_REF_SM3.

(2) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
JogForward	Jog forward	BOOL	TRUE, FALSE	FALSE	TRUE: Axis moves in the negative direction. FALSE: Axis stops moving in the negative direction.
JogBackward	Jog backward	BOOL	TRUE, FALSE	FALSE	TRUE: Axis moves in the negative direction. FALSE: Axis stops moving in the negative direction.
SpeedRatio	Speed ratio	LREAL	“0”~“1”	0	The proportion of the axis's actual velocity relative to the target velocity.
Velocity	Target velocity	LREAL	Positive number or "0".	0	Specifies the target velocity.
Acceleration	Acceleration	LREAL	Positive number or "0".	0	Acceleration
Deceleration	Deceleration	LREAL	Positive number or "0".	0	Deceleration
Jerk	Jerk	LREAL	Positive number or "0".	0	Specifies the jerk.

(3) Output variable

Output variable	Name	Data type	Valid range	Description
Busy	Busy	BOOL	TRUE, FALSE	TRUE: Function block is running.
CommandAborted	Command aborted	BOOL	TRUE, FALSE	TRUE: Function block execution is aborted.
Error	Error	BOOL	TRUE, FALSE	TRUE: An error occurred in the function block; execution has stopped.
ErrorID	Error ID	SMC_ERROR	0	Outputs the error code when an exception occurs.

◆ Key points

• Controls axis jog operation. Forward jog is controlled by JogForward. When set to TRUE, the axis performs forward jog at the set velocity, speed ratio, and acceleration. Reverse jog is controlled by JogBackward. When set to TRUE, the axis performs reverse jog at the set velocity, speed ratio, and acceleration.

• Changing the SpeedRatio during axis motion takes effect immediately. The SpeedRatio can be modified to change the axis's current motion speed as a proportion of the target velocity in real-time.

• During jog operation, if JogForward or JogBackward changes from TRUE to FALSE, the axis immediately decelerates to a stop according to the set deceleration.

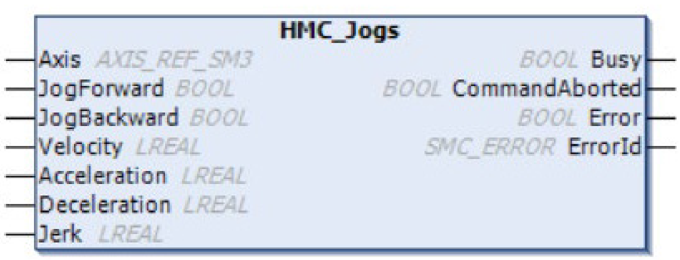
• The axis is in a Continuous Motion state during jogging.

• If both JogForward and JogBackward are set to TRUE simultaneously, the axis will stop moving or not start moving, and no error will be reported.

4.7.2 HMC_Jogs (FB)

A variant of the HMC_Jog function block. This block is used for variable-speed jogging by directly changing the velocity value.

◆ Variables

<table>
<tr><th>Name</th><th colspan="3">HMC_Jogs (Variable-speed jog)</th></tr>
<tr><td>Supported modes</td><td>CSP</td><td>CSV</td><td>CST</td></tr>
<tr><td colspan="2">Graphical representation</td><td colspan="2">ST representation</td></tr>
<tr><td colspan="2">
</td><td colspan="2">HMC_Jogs(
Axis:= ,
JogForward:= ,
JogBackward:= ,
Velocity:= ,
Acceleration:= ,
Deceleration:= ,
Jerk:= ,
Busy=> ,
CommandAborted=> ,
Error=> ,
ErrorId=>);</td></tr>
</table>

(1) Input/Output variable

Input/Output variable	Name	Data type	Description
Axis	Axis	AXIS_REF_SM3	Specifies the axis, an instance of AXIS_REF_SM3.

(2) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
JogForward	Jog forward	BOOL	TRUE, FALSE	FALSE	TRUE: Axis moves in the positive direction. FALSE: Axis stops moving in the positive direction.
JogBackward	Jog backward	BOOL	TRUE, FALSE	FALSE	TRUE: Axis moves in the negative direction. FALSE: Axis stops moving in the negative direction.
Velocity	Target velocity	LREAL	Positive number or "0".	0	Specifies the target velocity.
Acceleration	Acceleration	LREAL	Positive number or "0".	0	Acceleration
Deceleration	Deceleration	LREAL	Positive number or "0".	0	Deceleration
Jerk	Jerk	LREAL	Positive number or "0".	0	Specifies the jerk.

(3) Output variable

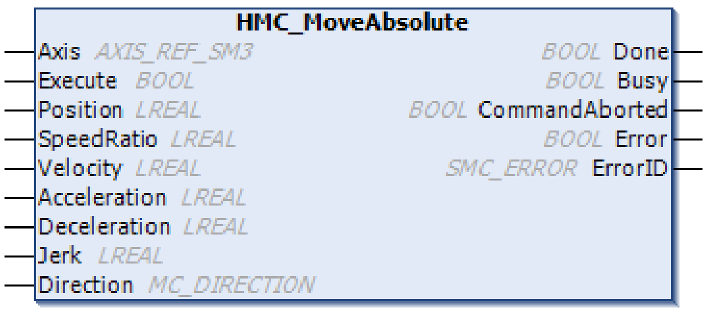
Output variable	Name	Data type	Valid range	Description
Busy	Busy	BOOL	TRUE, FALSE	TRUE: Function block is running.
CommandAborted	Command aborted	BOOL	TRUE, FALSE	TRUE: Function block execution is aborted.
Error	Error	BOOL	TRUE, FALSE	TRUE: An error occurred in the function block; execution has stopped.
ErrorID	Error ID	SMC_ERROR	0	Outputs the error code when an exception occurs.

◆ Key points

- Changes take effect immediately even during axis motion. The axis's current motion speed can be changed in real-time by modifying Velocity.
- For other considerations, refer to the description of the HMC_Jog function block.

4.7.3 HMC_MoveAbsolute (FB)

◆ Variables

Name	HMC_MoveAbsolute (Variable-speed absolute positioning)		
Supported modes	CSP	CSV	CST
Graphical representation		ST representation	
		HMC_MoveAbsolute(Axis:= , Execute:= , Position:= , SpeedRatio:= , Velocity:= , Acceleration:= , Deceleration:= , Jerk:= , Direction:= , Done=> , Busy=> , CommandAborted=> , Error=> , ErrorID=>);	

(1) Input/Output variable

Input/Output variable	Name	Data type	Description
Axis	Axis	AXIS_REF_SM3	Specifies the axis, an instance of AXIS_REF_SM3.

(2) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
Execute	Start	BOOL	TRUE, FALSE	FALSE	TRUE: Enables the function block.
Position	Target position	LREAL	Negative numbers, positive numbers, "0"	0	Specifies the target position in absolute coordinates, in [command units].
Velocity	Target velocity	LREAL	Positive number	0	Specifies the target velocity.
SpeedRatio	Speed ratio	LREAL	“0”~“1”	0	The proportion of the axis's actual velocity relative to the target velocity.
Velocity	Target velocity	LREAL	Positive number or "0".	0	Specifies the target velocity.
Acceleration	Acceleration	LREAL	Positive number or "0".	0	Acceleration
Deceleration	Deceleration	LREAL	Positive number or "0".	0	Deceleration
Jerk	Jerk	LREAL	Positive number or "0".	0	Specifies the jerk.
Direction	Direction selection	MC_Direction	Fastest, current, positive, shortest, negative	Shortest	Reference: MC_Direction

(3) Output variable

Output variable	Name	Data type	Valid range	Description
Done	Done	BOOL	TRUE, FALSE	TRUE: Function block is complete.
Busy	Busy	BOOL	TRUE, FALSE	TRUE: Function block is running.
CommandAborted	Command aborted	BOOL	TRUE, FALSE	TRUE: Function block execution is aborted.
Error	Error	BOOL	TRUE, FALSE	TRUE: An error occurred in the function block; execution has stopped.
ErrorID	Error ID	SMC_ERROR	0	Outputs the error code when an exception occurs.

◆ **Key points**

・ This is an absolute positioning function block. The Position data is the absolute position of the axis. The axis state is Discrete Motion during the execution of this function block.

・ Changing the SpeedRatio during axis motion takes effect immediately. The SpeedRatio can be modified to change the axis's current motion speed as a proportion of the target velocity in real-time.

・ The rising edge of Execute starts the function block. During function block execution, a new rising edge can be re-triggered. Each rising edge reloads the function block's input parameters and re-executes the function block.

・ If Acceleration or Deceleration is zero, starting the function block (Execute) will result in an error, and the axis state will be Standstill.

・ For Direction (direction selection), refer to MC_Direction.

4.8 RobotMove (Function group)

4.8.1 Interpolation models and model function blocks

4.8.1.1 FB_KimTransl_None2 (No-model 2-axis model)

The no-model 2-axis resembles a 2-axis gantry planar structure. Axis 1 controls the horizontal lateral motion (X-axis), and Axis 2 controls the horizontal longitudinal motion (Y-axis).

```
FB_KimTransl_None2(
            bWriteParameter:= ,          //写入模型以及轴配置参数
            outCartesianPos=> ,          //输出空间坐标
            bDone=> ,                    //模型参数配置完成信号
            bError=> ,                   //模型参数写入错误信号
            dOffsetA0:= ,                //A0轴的额外偏移量
            dOffsetA1:= ,                //A1轴的额外偏移量
            AxisX:= ,                    //水平X轴
            AxisY:= );                   //水平Y轴
```

4.8.1.2 FB_KimTransl_None3 (No-model 3-axis model)

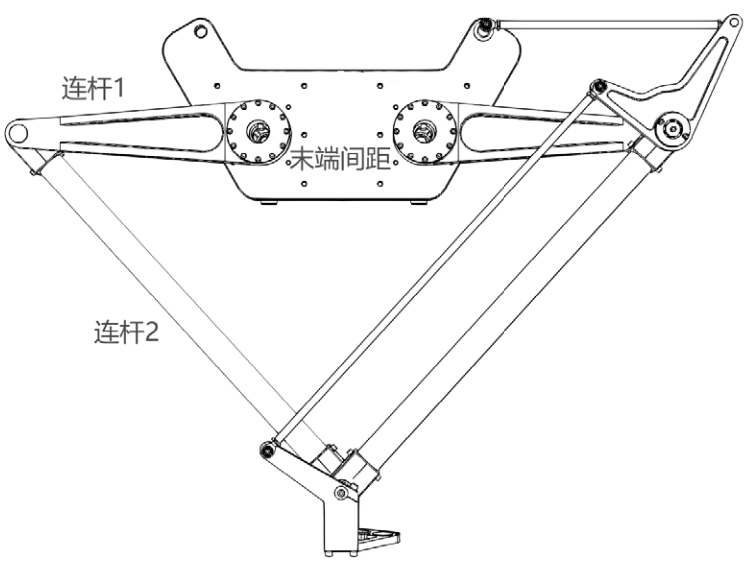
The no-model 3-axis resembles a 3-axis gantry spatial structure. Based on the 2-axis model, an additional Axis 3 is added to control vertical motion along the spatial Z-axis.

```
FB_KimTransl_None3(
        bWriteParameter:= ,          //写入模型以及轴配置参数
        outCartesianPos=> ,          //输出空间坐标
        bDone=> ,                    //模型参数配置完成信号
        bError=> ,                   //模型参数写入错误信号
        dOffsetA0:= ,                //A0轴的额外偏移量
        dOffsetA1:= ,                //A1轴的额外偏移量
        dOffsetA2:= ,                //A2轴的额外偏移量
        AxisX:= ,                    //空间X轴
        AxisY:= ,                    //空间Y轴
        AxisZ:= );                   //空间Z轴
```

4.8.1.3 FB_KimTransl_Delta2 (2-axis delta model)

The 2-axis Delta robot model is shown in the figure below. The left and right axes (modal) control the motion of the left and right link 1 (uniformly defined as Link 1), respectively. Three parameters must be set in the function block: the length of link 1 (dArmLength1), the length of link 2 (dArmLength2), and the distance between the end points when both links 1 are horizontal (dDistance). By default, when both links 1 are in the horizontal position, the spatial position of the robot's end point is (0,0).

```
FB_KimTransl_Delta2(
        bWriteParameter:= ,          //写入模型以及轴配置参数
        outCartesianPos=> ,          //输出空间坐标
        bDone=> ,                    //模型参数配置完成信号
        bError=> ,                   //模型参数写入错误信号
        dArmLength1:= ,              //连杆1长度
        dArmLength2:= ,              //连杆2长度
        dDistance:= ,                //两个连杆1的末端距离
        dOffsetA0:= ,                //A0轴的额外偏移量
        dOffsetA1:= ,                //A1轴的额外偏移量
        Axis0:= ,                    //左轴，（在坐标系中，位置在X轴的负半轴的为左轴）
        Axis1:= );                   //右轴，（在坐标系中，位置应在X轴的正半轴为右轴）
```



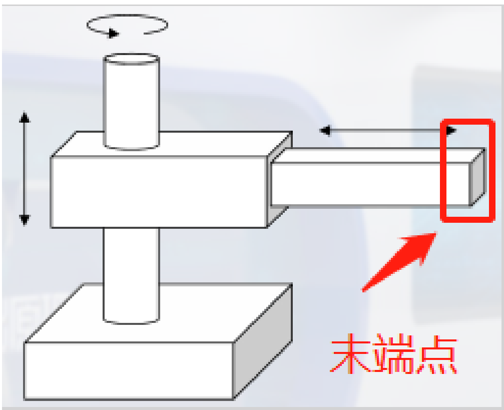
D • Motion (Motion Control)

4.8.1.4 FB_KimTransl_Polar2_Z (3-axis polar cylindrical coordinate model)

Axis 0 controls the rotational axis, Axis 1 controls the horizontal telescopic axis, and Axis 2 controls the vertical movement along the spatial Z-axis.

Note: The set of points reachable in space by the end point is a cylinder, hence it is called the cylindrical coordinate system model.

```
FB_KimTransl_Polar2_Z(
        bWriteParameter:= ,          //写入模型以及轴配置参数
        outCartesianPos=> ,          //输出空间坐标
        bDone=> ,                    //模型参数配置完成信号
        bError=> ,                   //模型参数写入错误信号
        dOffsetA0:= ,                //A0轴（旋转C轴）的额外偏移量
        dOffsetA1:= ,                //A1轴（线性R轴）的额外偏移量
        dMaxR:= ,                    //A1轴（线性R轴）的最大距离
        Axis0:= ,                    //A0轴（旋转C轴）
        Axis1:= ,                    //A1轴（线性R轴）
        Axis2:= );                   //A2轴（升降Z轴）
```



4.8.1.5 FB_KimTransl_Scara2_Z_Tool (4-axis Scara2 robot model)

The 4-axis Scara robot model is shown in the figure. Axis 0 (modal) controls the rotation of the main arm. Axis 1 (modal) controls the rotation of the forearm. Axis 2 (linear) controls the vertical movement of the auxiliary axis. Axis 3 (modal) controls the rotation of the tool axis, with counter-clockwise defined as the positive direction.

Three parameters must be set in the function block: the length of the main arm (dArmLength1), the length of the forearm (dArmLength2), and the elbow configuration bElbowlow (TRUE for right-handed, FALSE for left-handed). In simulation, the default configuration is with both the main arm and forearm positioned along the positive half of the spatial x-axis, meaning the arm is fully extended pointing along the positive x-axis. The auxiliary axis is not moved vertically, and the tool axis rotation angle is 0. In this state, the end point is at the spatial coordinate (0,0,0) with the tool axis rotation angle at 0 degrees.

```
FB_KimTransl_Scara2_Z_Tool(
        bWriteParameter:= ,          //写入模型以及轴配置参数
        outCartesianPos=> ,          //输出空间坐标
        bDone=> ,                    //模型参数配置完成信号
        bError=> ,                   //模型参数写入错误信号
        dArmLength1:= ,              //连杆1长度（大臂）
        dArmLength2:= ,              //连杆2长度（小臂）
        dOffsetA0:= ,                //A0轴的额外偏移量
        dOffsetA1:= ,                //A1轴的额外偏移量
        bElbowLow:= ,                //肘部高低设定，TRUE为低右手系，FALSE为高左手系
        Axis0:= ,                    //大臂轴
        Axis1:= ,                    //小臂轴
        Axis2:= ,                    //辅助Z轴
        Axis3:= );                   //工具R轴
```

4.8.1.6 FB_KimTransl_Scara2_Z_Tool_ABS (4-axis Scara2 robot absolute model)

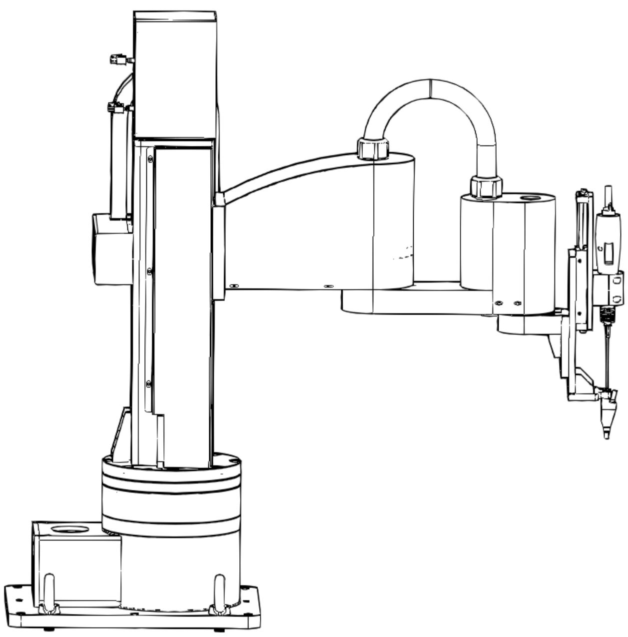
This is the absolute version of the FB_KimTransl_Scara2_Z_Tool model. The servo zero point corresponds to the spatial coordinate (L1+L2, 0, 0) and orientation (0,0,0).

```
FB_KimTransl_Scara2_Z_Tool_ABS(
        bWriteParameter:= ,          //写入模型以及轴配置参数
        outCartesianPos=> ,          //输出空间坐标
        bDone=> ,                    //模型参数配置完成信号
        bError=> ,                   //模型参数写入错误信号
        dArmLength1:= ,              //连杆1长度（大臂）
        dArmLength2:= ,              //连杆2长度（小臂）
        bElbowLow:= ,                //肘部高低设定，TRUE为低右手系，FALSE为高左手系
        Axis0:= ,                    //大臂轴
        Axis1:= ,                    //小臂轴
        Axis2:= ,                    //辅助Z轴
        Axis3:= );                   //工具R轴
```

4.8.1.7 FB_KimTransl_Scara3_Z (Three-joint Scara model)

This is a variant of the FB_KimTransl_Scara2_Z_Tool model, as shown in the figure below.

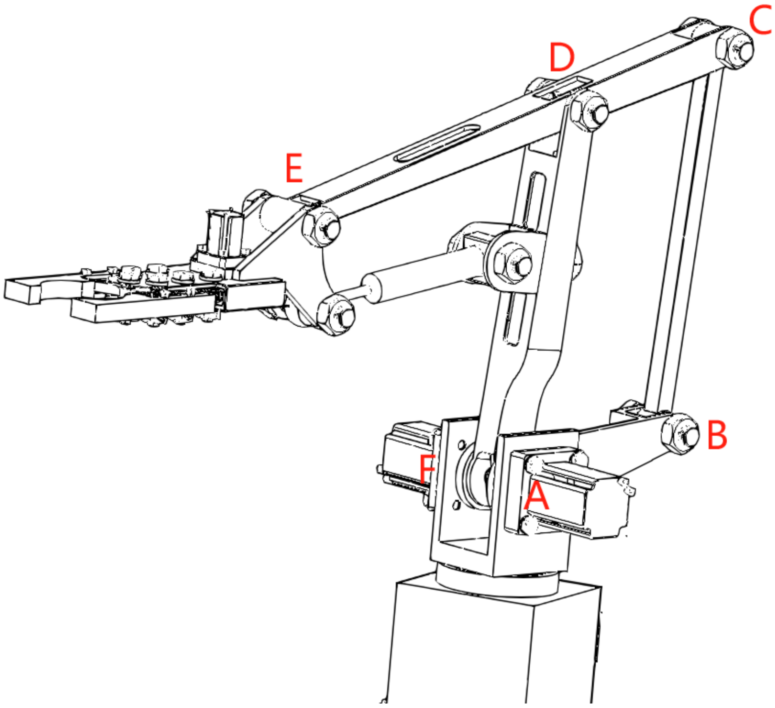
```
FB_KimTransl_Scara3_Z(
        bWriteParameter:= ,          //写入模型以及轴配置参数
        outCartesianPos=> ,          //输出空间坐标
        bDone=> ,                    //模型参数配置完成信号
        bError=> ,                   //模型参数写入错误信号
        dArmLength1:= ,              //连杆1长度（大臂）
        dArmLength2:= ,              //连杆2长度（小臂）
        dArmLength3:= ,              //连杆3长度
        bElbowLow:= ,                //肘部高低设定，TRUE为低右手系，FALSE为高左手系
        Axis0:= ,                    //大臂轴
        Axis1:= ,                    //小臂轴
        Axis2:= ,                    //辅助Z轴
        Axis3:= );                   //工具R轴
```



4.8.1.8 FB_KimTransl_SimilarScara2 (Scara-like model)

This model is a variant of the Scara model, as shown in the figure below.

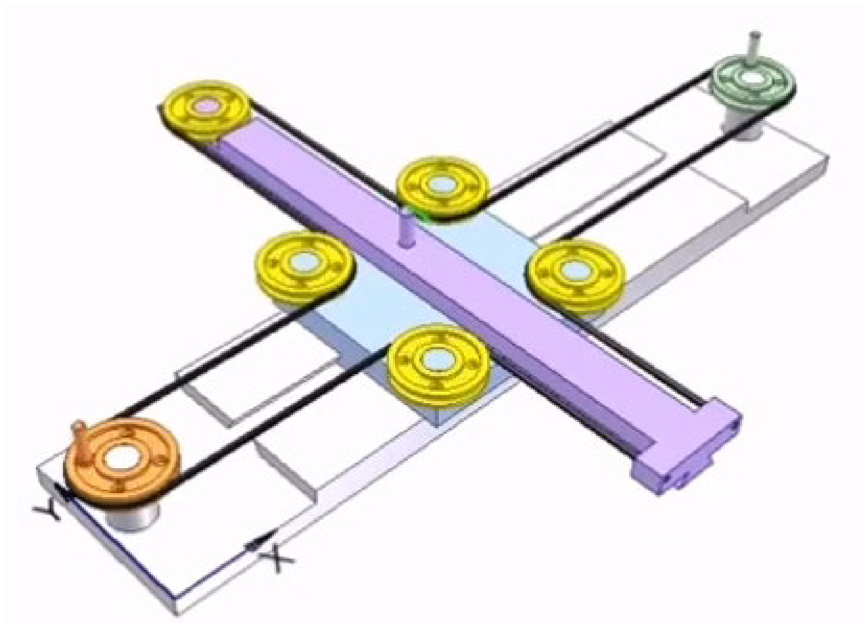
Note: Parts A and F are each controlled by two motors. FD is the length of the main arm axis; EC is the length of the forearm axis; CB is the length of the intermediate support side; DC is the length of the end side; AB is the length from the main arm axis to the intermediate support side on the end side.



4.8.1.9 FB_KimTransl_Trapezoid2 (2-axis T-shaped manipulator)

This structure is similar to what is also called a CoreXY or H-Bot structure.

```
FB_KimTransl_Trapezoid2(
          bWriteParameter:= ,          //写入模型以及轴配置参数
          outCartesianPos=> ,          //输出空间坐标
          bDone=> ,                    //模型参数配置完成信号
          bError=> ,                   //模型参数写入错误信号
          dOffsetX:= ,                 //空间坐标X起始偏移量
          dOffsetY:= ,                 //空间坐标Y起始偏移量
          AxisA0:= ,                   //A0轴
          AxisA1:= );                  //A1轴
```



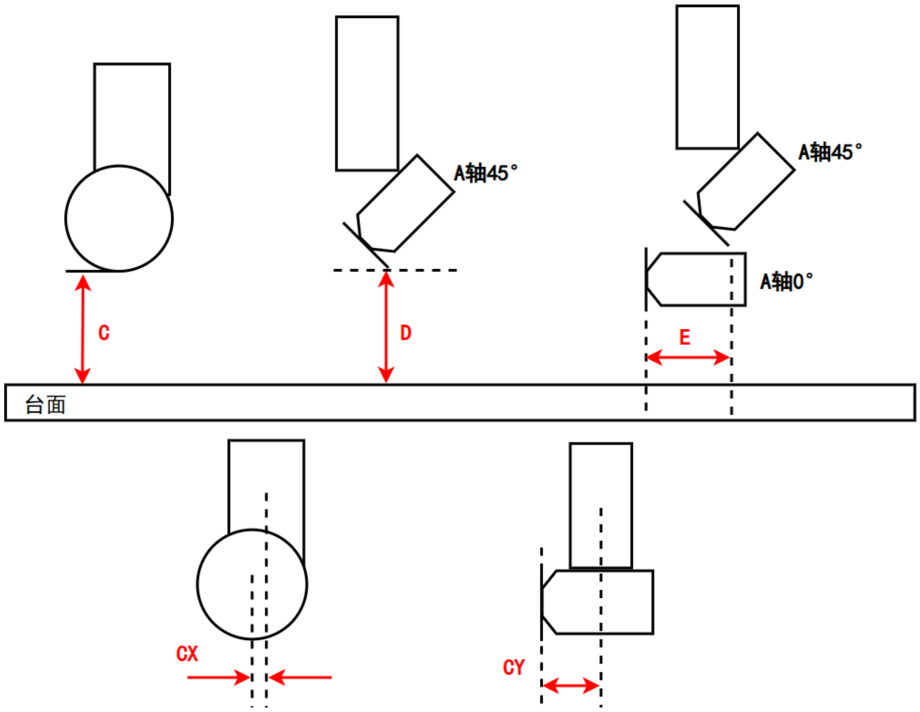
4.8.1.10 FB_KimTransl_GantryCutter2 (2D gantry with cutter)

```
FB_KimTransl_GantryCutter2(
        bWriteParameter:= ,            //写入模型以及轴配置参数
        outCartesianPos=> ,            //输出空间坐标
        bDone=> ,                      //模型参数配置完成信号
        bError=> ,                     //模型参数写入错误信号
        dOffsetX:= ,                   //A0轴的额外偏移量
        dOffsetY:= ,                   //A1轴的额外偏移量
        dOffsetR:= ,                   //A3轴的额外偏移量
        AxisX:= ,                      //X轴
        AxisY:= ,                      //Y轴
        AxisR:= );                     //R轴
```

4.8.1.11 FB_KimTransl_GantryCutter3 (3D gantry with cutter)

```
FB_KimTransl_GantryCutter3(
        bWriteParameter:= ,            //写入模型以及轴配置参数
        outCartesianPos=> ,            //输出空间坐标
        bDone=> ,                      //模型参数配置完成信号
        bError=> ,                     //模型参数写入错误信号
        dOffsetX:= ,                   //A0轴的额外偏移量
        dOffsetY:= ,                   //A1轴的额外偏移量
        dOffsetZ:= ,                   //A2轴的额外偏移量
        dOffsetR:= ,                   //A3轴的额外偏移量
        AxisX:= ,                      //X轴
        AxisY:= ,                      //Y轴
        AxisZ:= ,                      //Z轴
        AxisR:= );                     //R轴
```

4.8.1.12 FB_KimTransl_Axis4 (4-Axis bridge cutting machine)



Straight cut Z platform difference: Measure the distance as shown in Figure C, with the Z-axis at the zero position.

Angled cut Z platform difference: Measure the distance as shown in Figure D, with the Z-axis at the zero position and the A-axis at 45°.

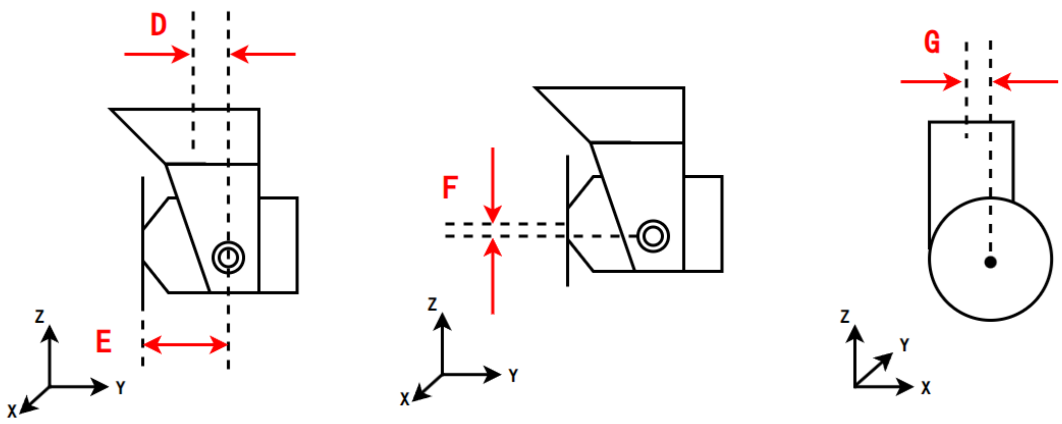
Angled cut Y difference: Measure the distance as shown in Figure E, the projection spacing on the Y-axis when the A-axis is at 45° and 0°.

C axis center deviation X: Measure the distance as shown in Figure CX.

C axis center deviation Y: Measure the distance as shown in Figure CY.

```
FB_KimTransl_Axis4(
        bWriteParameter:= ,         //写入模型以及轴配置参数
        outCartesianPos=> ,         //输出空间坐标
        bDone=> ,                   //模型参数配置完成信号
        bError=> ,                  //模型参数写入错误信号
        fZ1:= ,                     //直切Z平台差
        fZ2:= ,                     //斜切Z平台差
        frotateA_Y:= ,              //斜切Y差
        frotateC_X:= ,              //C轴心偏差X
        frotateC_Y:= ,              //C轴心偏差Y
        Apos:= ,                    //A轴角度
        dOffsetX:= ,                //X轴的额外偏移量
        dOffsetY:= ,                //Y轴的额外偏移量
        dOffsetZ:= ,                //Z轴的额外偏移量
        AxisX:= ,                   //X轴
        AxisY:= ,                   //Y轴
        AxisZ:= ,                   //Z轴
        AxisC:= );                  //C轴
```

4.8.1.13 FB_KimTransl_Axis5 (5-axis bridge cutting machine)



Offset between A-axis rotation center and C-axis rotation center: Measure the distance as shown in Figure D.

Offset between the inner side of the saw blade and the A-axis rotation center: Measure the distance as shown in Figure E.

Height difference between the saw blade center and the A-axis rotation center: Measure the distance as shown in Figure F.

Offset between the saw blade center and the C-axis rotation center: Measure the distance as shown in Figure G.

Saw blade thickness: Input the appropriate thickness value according to the specific blade used.

```
FB_KimTransl_Axis5(
          bWriteParameter:= ,          //写入模型以及轴配置参数
          outCartesianPos=> ,          //输出空间坐标
          bDone=> ,                    //模型参数配置完成信号
          bError=> ,                   //模型参数写入错误信号
          farm1:= ,                    //A轴旋转中心与C旋转中心偏移
          farm2:= ,                    //锯片内侧和A轴旋转中心偏移
          fsawW:= ,                    //锯片厚度
          fsawR:= ,                    //锯片半径R
          fsawZ:= ,                    //锯片中心和A轴旋转中心高度差
          fsawC:= ,                    //锯片中心和C旋转中心偏移
          dOffsetX:= ,                 //X轴的额外偏移量
          dOffsetY:= ,                 //Y轴的额外偏移量
          dOffsetZ:= ,                 //Z轴的额外偏移量
          AxisX:= ,                    //X轴
          AxisY:= ,                    //Y轴
          AxisZ:= ,                    //Z轴
          AxisA:= ,                    //A轴
          AxisC:= );                   //C轴
```

4.8.2 Motion control function blocks

4.8.2.1 HMC_RobotHandWheel (Handwheel spatial jog function block)

```
HMC_RobotHandWheel(
    bEnable:= ,               // 使能
    pRobotKimTransl:= ,       //机器人模型，必须使用，如果无需模型请使指定无模型即可
    HandWhellPosX:= ,         //手轮X位置
    HandWhellPosY:= ,         //手轮Y位置
    HandWhellPosZ:= ,         //手轮Z位置
    bBusy=> ,                 //插补运行中信号，TRUE为运行中，FALSE为插补运行完毕
    bError=> ,                //运行错误信号，插补计算错误和轴异常都会导致功能块报错
    outCartesianPos=> );      //根据当前轴位置以及运动学模型换算后的空间坐标位置
```

4.8.2.2 HMC_RobotJog (Interpolation jog function block)

```
HMC_RobotJog(
    fVelocity:= ,             //点动速度
    fAcceleration:= ,         //点动加速度
    fDeceleration:= ,         //点动减速度
    xJogForward:= ,           //X轴正方向点动
    xJogBackward:= ,          //X轴反方向点动
    yJogForward:= ,           //轴正方向点动
    yJogBackward:= ,          //轴反方向点动
    zJogForward:= ,           //Z轴正方向点动
    zJogBackward:= ,          //Z轴负方向点动
    pRobotKimTransl:= ,       //机器人模型，必须使用，如果无需模型请指定无模型。
    bBusy=> ,                 //插补运行中信号，TRUE为运行中，FALSE为插补运行完毕
    bError=> ,                //运行错误信号，插补计算错误和轴异常都会导致功能块报错
    outCartesianPos=> );      //根据当前轴位置以及运动学模型换算后得到的框架坐标(X,Y,Z)
```

4.8.2.3 HMC_RobotMove (Motion control function block)

The trajectory command parameter group supports up to 100 commands.

```
HMC_RobotMove(
    bExecute:= ,              //上升沿触发信号
    bPause:= ,                //插补暂停信号，为TRUE后停止插补，恢复FALSE后继续运行
    bAbort:= ,                //插补终止信号，为TRUE后终止插补运行并重置输出
    fOverride:= ,             //插补器当前运行的指令的速度比例，调整全局插补速度，必须填写参数
    bSmooth:= ,               //启用多段轨迹平滑功能，需要在功能块执行前选择
    fSmooth_R:= ,             //多段轨迹平滑半径
    fSmooth_Override:=,       //平滑功能速度缩放比例，值应当小于1
    pRobotKimTransl:=,        //机器人模型，必须使用，如果无需模型请指定无模型
    NumberOfCommand:=,        //轨迹指令使用个数
    stMoveCommand:= ,         //运动指令参数数组，最大支持100条指令
    bDone=> ,                 //插补完成信号
    bPaused=> ,               //暂停完成信号
    bBusy=> ,                 //插补运行中信号，TRUE为运行中，FALSE为插补运行完毕
    bError=> ,                //运行错误信号，插补计算错误和轴异常都会导致功能块报错
    eErrorID=> ,              //故障代码
    dCommandCount=> ,         //输出运动指令个数，当功能块将运动指令全部执行完毕后置为0
    dRunCount=> ,             //当前运行到了第几条指令
    outCartesianPos=> );      //根据当前轴位置以及运动学模型换算后得到的空间坐标位置
```

4.8.2.4 HMC_RobotMove_max1000 (Extended multi-axis motion control function block)

The trajectory command parameter group supports up to 1,000 commands.

```
HMC_RobotMove_max1000(
        bExecute:= ,                    //上升沿触发信号
        bPause:= ,                      //插补暂停信号，为TRUE后停止插补，恢复FALSE后继续运行
        bAbort:= ,                      //插补终止信号，为TRUE后终止插补运行并重置输出
        fOverride:= ,                   //插补器当前运行的指令的速度比例，调整全局插补速度，必须填写参数
        bSmooth:= ,                     //启用多段轨迹平滑功能，需要在功能块执行前选择
        fSmooth_R:= ,                   //多段轨迹平滑半径
        fSmooth_Override:=,             //平滑功能速度缩放比例，值应当小于1
        pRobotKimTransl:=,              //机器人模型，必须使用，如果无需模型请指定无模型
        NumberOfCommand:=,              //轨迹指令使用个数
        stMoveCommand:= ,               //运动指令参数数组，最大支持1000条指令
        bDone=> ,                       //插补完成信号
        bPaused=> ,                     //暂停完成信号
        bBusy=> ,                       //插补运行中信号，TRUE为运行中，FALSE为插补运行完毕
        bError=> ,                      //运行错误信号，插补计算错误和轴异常都会导致功能块报错
        eErrorID=> ,                    //故障代码
        dCommandCount=> ,               //输出运动指令个数，当功能块将运动指令全部执行完毕后置为0
        dRunCount=> ,                   //当前运行到了第几条指令
        outCartesianPos=> );            //根据当前轴位置以及运动学模型换算后得到的空间坐标位置
```

4.8.3 Motion command parameter stMoveParameter

4.8.3.1 Linear interpolation mode

Linear interpolation controls the machine's endpoint to move from the initial position to the target point along a straight-line path. Simply set the bMoveType mode to FALSE. The parameters used are as follows:

```
stCommand[1].fAcceleration :=;          //加速度
stCommand[1].fDeceleration :=;          //减速度
stCommand[1].fVelocity    :=;           //速度
stCommand[1].stTargetPos.X :=;          //末端点设定位置空间坐标X
stCommand[1].stTargetPos.Y :=;          //末端点设定位置空间坐标Y
stCommand[1].stTargetPos.Z :=;          //末端点设定位置空间坐标Z
stCommand[1].stTargetPos.A :=;          //末端点工具轴设定旋转角度A
```

4.8.3.2 Circular interpolation mode - radius mode

Determine the distance from the start point to the target point, and set the radius. This defines an isosceles triangle (legs equal to the radius). The vertex of this triangle is the arc center. The minor arc (arc with central angle ≤ 180°) is used as the motion path. The parameters used are as follows:

```
stCommand[1].bMoveType    :=TRUE;      //插补模式选择（TRUE为圆弧插补，FALSE为直线插补）
stCommand[1].fAcceleration :=;         //加速度
stCommand[1].fDeceleration :=;         //减速度
stCommand[1].fVelocity    :=;          //速度
stCommand[1].stArcParameter.eArcMode:=1;       //圆弧模式，圆弧插补时生效（1为半径模式）
stCommand[1].stArcParameter.bDirection:=;      //圆弧方向（TRUE为逆时针，FALSE为顺时针）
stCommand[1].stArcParameter.fRadius:= ;        //设置半径，半径模式启用,默认在XY平面插补
stCommand[1].stTargetPos.X :=;         //末端点设定位置空间坐标X
stCommand[1].stTargetPos.Y :=;         //末端点设定位置空间坐标Y
stCommand[1].stTargetPos.Z :=;         //末端点设定位置空间坐标Z
stCommand[1].stTargetPos.A :=;         //末端点工具轴设定旋转角度A
```

4.8.3.3 Circular interpolation mode - center mode

Determine the coordinates of the start point, target point, and arc center. The motion path is defined by the center coordinate. The spatial coordinate of the center should lie on the perpendicular bisector of the line between the start and target points. If not, the center coordinate will be automatically corrected, with a deviation not exceeding 10%. The arc motion direction bDirection can be selected freely for clockwise or counterclockwise rotation (minor arc: central angle ≤ 180°). The parameters used are as follows:

```
stCommand[1].bMoveType    :=TRUE;      //插补模式选择（TRUE为圆弧插补，FALSE为直线插补）
stCommand[1].fAcceleration :=;         //加速度
stCommand[1].fDeceleration :=;         //减速度
stCommand[1].fVelocity    :=;          //速度
stCommand[1].stArcParameter.eArcMode:=0;          //圆弧模式，圆弧插补时生效（0为圆心模式）
stCommand[1].stArcParameter.bDirection:=;         //圆弧方向（TRUE为劣弧，FALSE为优弧）
stCommand[1].stArcParameter.stMidPoint.X:=;       //圆心空间坐标X，圆心模式启用
stCommand[1].stArcParameter.stMidPoint.Y:=;       //圆心空间坐标Y，圆心模式启用
stCommand[1].stArcParameter.stMidPoint.Z:=;       //圆心空间坐标Z，圆心模式启用
stCommand[1].stTargetPos.X :=;                    //末端点设定位置空间坐标X
stCommand[1].stTargetPos.Y :=;                    //末端点设定位置空间坐标Y
stCommand[1].stTargetPos.Z :=;                    //末端点设定位置空间坐标Z
stCommand[1].stTargetPos.A :=;                    //末端点工具轴设定旋转角度A
```

4.8.3.4 Circular interpolation mode - cross point mode

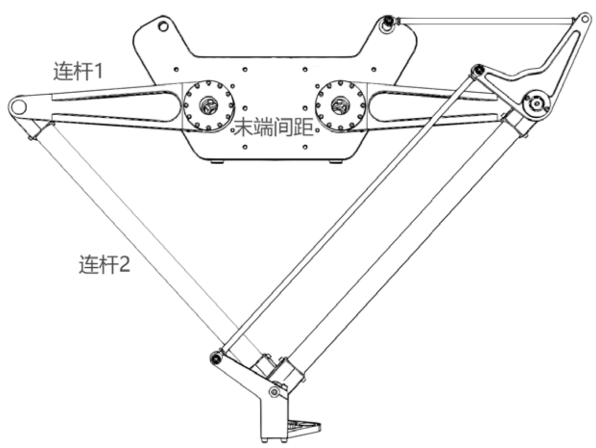
Determine the coordinates of the start point, target point, and cross point. Note that the three points must not be collinear, as this prevents defining a unique interpolation plane in 3D space. The circumcenter of the triangle formed by the three points (intersection of the perpendicular bisectors) serves as the arc center, defining the circumcircle. The arc motion direction bDirection can be selected for clockwise or counterclockwise rotation (minor arc: central angle ≤ 180°). The parameters used are as follows:

```
stCommand[1].bMoveType   :=TRUE;      //插补模式选择（TRUE为圆弧插补，FALSE为直线插补）
stCommand[1].fAcceleration :=;        //加速度
stCommand[1].fDeceleration :=;        //减速度
stCommand[1].fVelocity   :=;          //速度
stCommand[1].stArcParameter.eArcMode:=2;                 //圆弧模式，圆弧插补时生效（2为过渡点模式）
stCommand[1].stArcParameter.bDirection:=;                //圆弧方向（TRUE为劣弧，FALSE为优弧）
stCommand[1].stArcParameter.stAcorssPoint.X:=;           //过渡点空间坐标X，过渡点模式启用
stCommand[1].stArcParameter.stAcorssPoint.Y:=;           //过渡点空间坐标Y，过渡点模式启用
stCommand[1].stArcParameter.stAcorssPoint.Z:=;           //过渡点空间坐标Z，过渡点模式启用
stCommand[1].stTargetPos.X :=;        //末端点设定位置空间坐标X
stCommand[1].stTargetPos.Y :=;        //末端点设定位置空间坐标Y
stCommand[1].stTargetPos.Z :=;        //末端点设定位置空间坐标Z
stCommand[1].stTargetPos.A :=;        //末端点工具轴设定旋转角度A
```

4.8.4 Usage process examples

4.8.4.1 Example for 2-axis Delta model

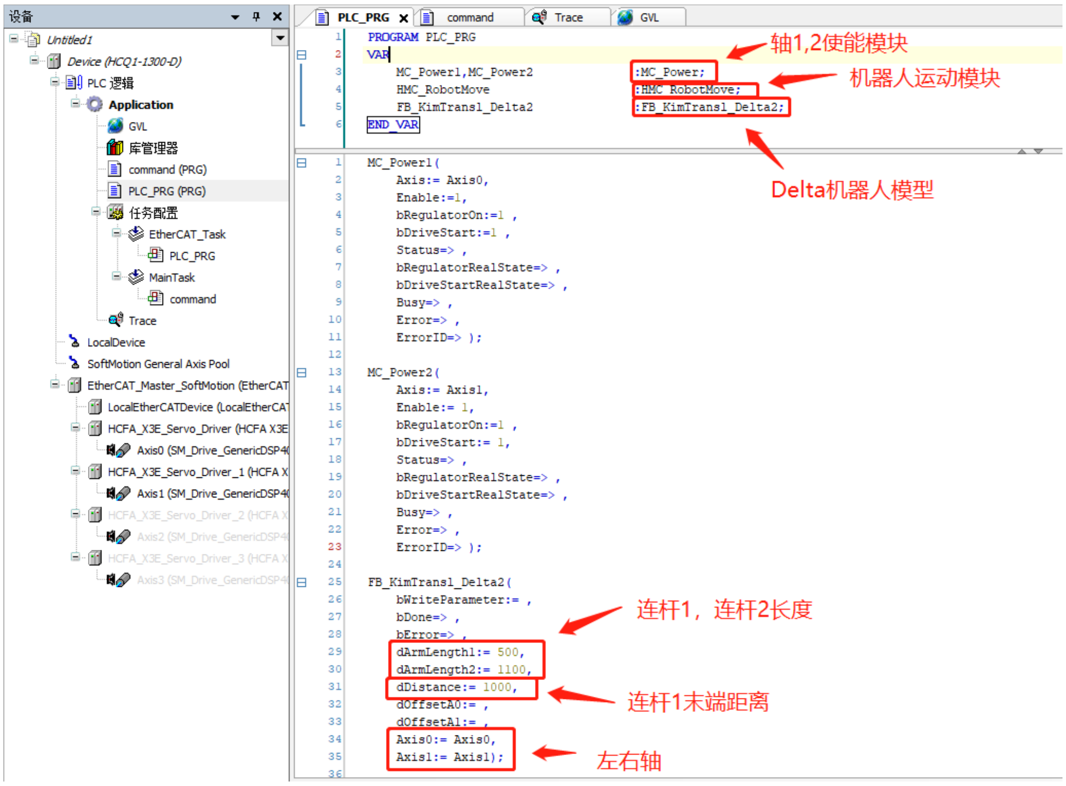
Example: Given a Delta robot (as shown below), Link 1 length is 500, Link 2 length is 1100. The starting ends of the two Link 1s are tangentially connected in a circle, so the end point distance is taken as twice the length of Link 1. The horizontal position is the origin. The positive direction for both axes is set inward, meaning the left axis clockwise is positive, and the right axis counterclockwise is positive. In this state, the robot endpoint position automatically converted by the function block is (0,0). To lift the end point up by 20, a linear interpolation motion command is used, setting the target coordinate to (0,20):



Declaration and calling of the Delta robot model function block and motion function block

Declare and call the HMC_RobotMove and FB_KimTransl_Delta2 function blocks in PLC_RPG. Also, enable Axis 0 and Axis 1. For convenience in this example, the enable value is directly written as 1, meaning auto-enable.

Parameters for FB_KimTransl_Delta: Link 1 length is 500, Link 2 length is 1100. dDistance is the distance between the ends of the two Link 1s. Since the starting ends are tangentially connected, dDistance here is Link 1 length * 2, i.e., 500 * 2=1000.




```
HMC_RobotMove(
    bExecute:= ,
    bPause:= ,
    bAbort:= ,
    fOverride:= 1,
    bSmooth:= ,
    fSmooth_R:= ,
    SmoothPathMode:= ,
    fSmooth_Override:= ,
    pRobotKimTransl:= ADR(FB_KimTransl_Delta2),
    NumberOfCommand:= NumberCommand,
    stMoveCommand:= stCommand,
    bDone=> ,
    bPaused=> ,
    bBusy=> ,
    bError=> ,
    eErrorID=> ,
    dCommandCount=> ,
    dRunCount=> ,
    outCartesianPos=> );
```

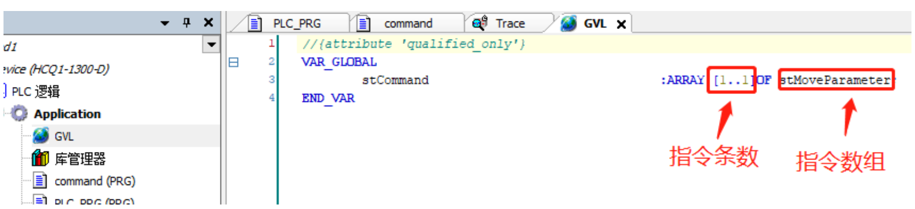
速度比例，必须填写

机器人模型地址

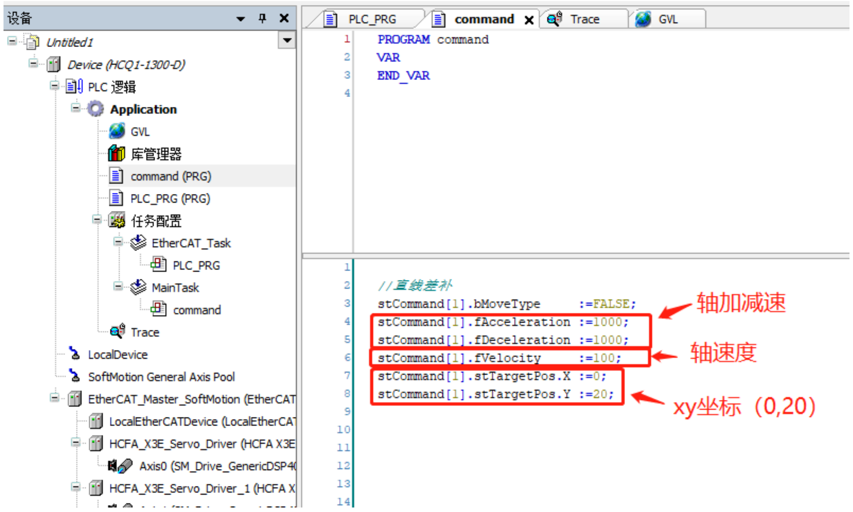
使用指令个数

运动指令数组名

Declare motion command array in GVL

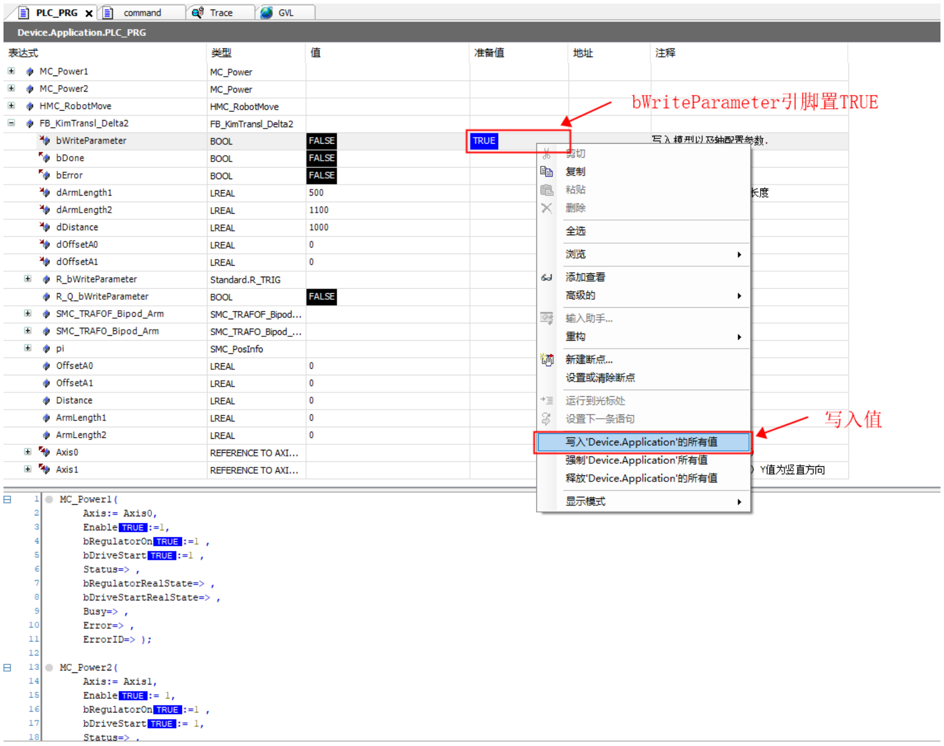


Fill in the parameters in POUcommand. To lift the end point up by 20, a linear interpolation motion command is used, setting the target coordinate to (0,20):



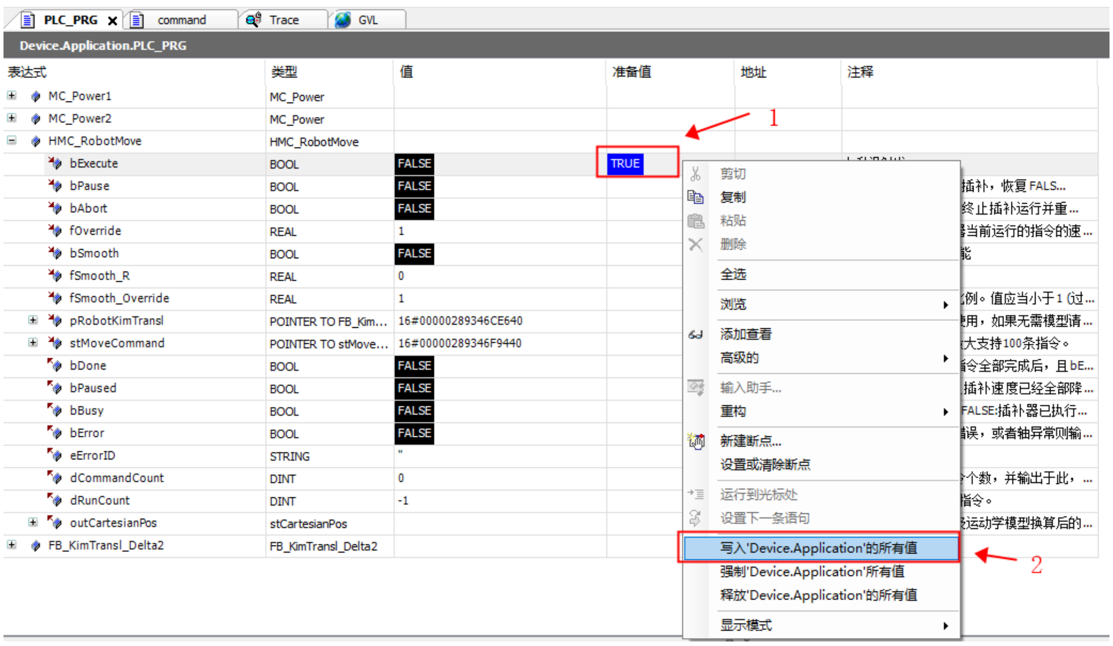
Implementing motion control

Trigger the bWriteParameter pin of the FB_KimTransl_Delta2 function block to write the model and axis configuration parameters.

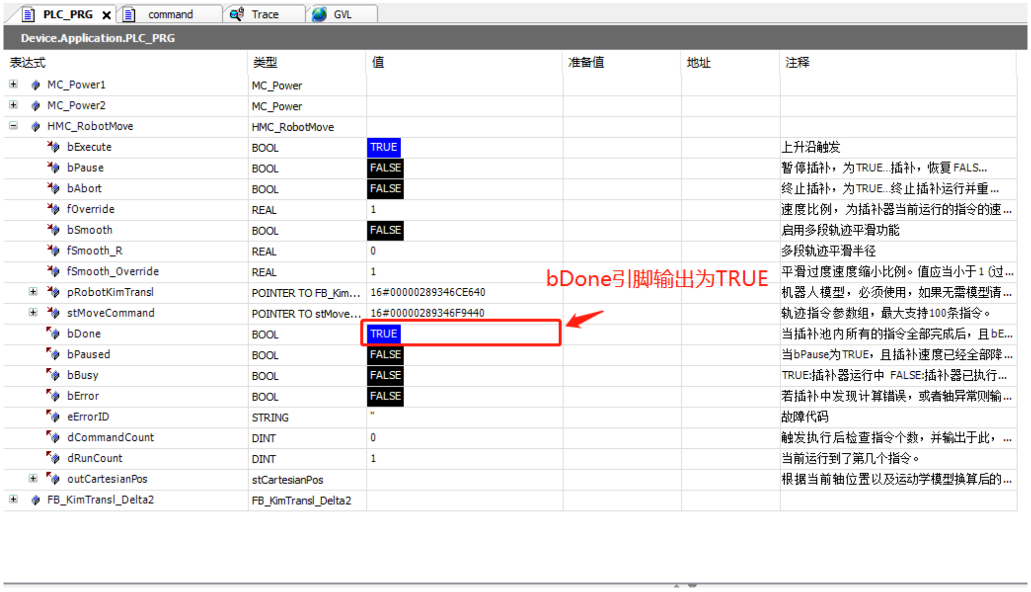


If parameters are correct and axes are error-free, the model and axis parameter configuration is completed. The bDone pin outputs TRUE, which indicates the completion of model parameter configuration.

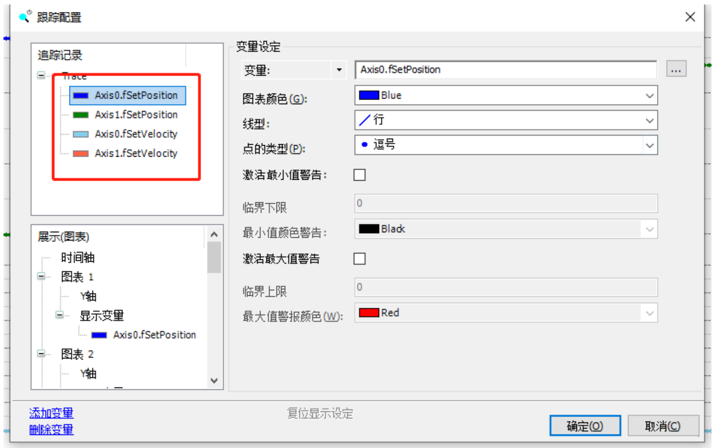
Trigger the bExecute pin of the HMC_RobotMove function block to start robot motion.



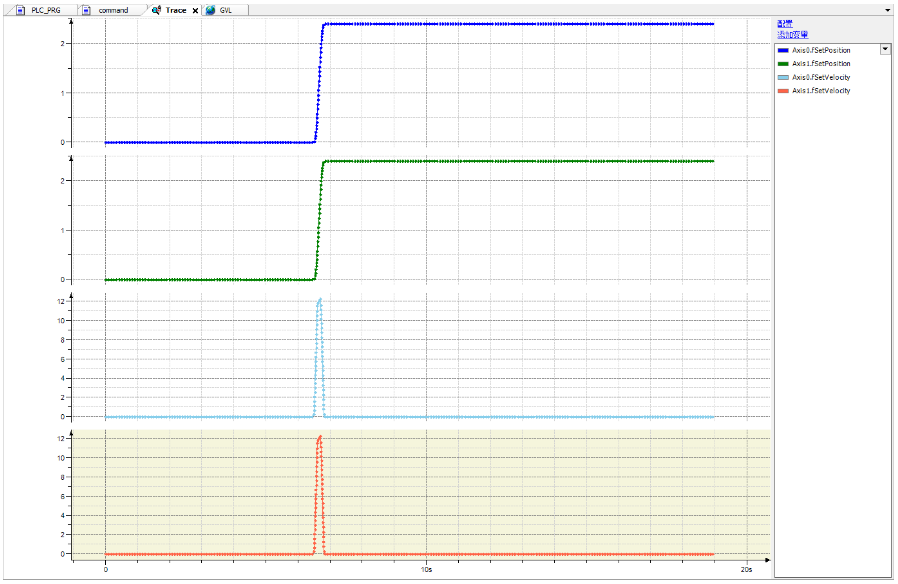
After the program finishes running all commands in the array (here, 1 command), the bDone pin outputs TRUE, indicating motion control is complete, and the robot end point has reached the specified position.



To observe the detailed axis motion process, add traces for the command position and command velocity of Axis 0 and Axis 1.



Repeat the operation and observe the command position and velocity in the trace. The trace shows both left and right axes rotate forward by 2.4 degrees, achieving the Delta robot end point lift. Conversion shows the lift distance is 20.



4.8.4.2 Example for 4-axis Scara model

Example: Given a 4-axis Scara robot as shown below. Assume the arm is in a right-hand configuration, with the main arm and forearm perpendicular. The auxiliary axis is not moved vertically. Thus, the end point spatial coordinate is xyz (500, 500, 0) and the tool axis rotation angle is 0. Main arm length is 500, forearm length is 500. The endpoint needs to be controlled to:

(1) Move from (500, 500, 0) to (800, 0, 0) in linear interpolation mode.

(2) Lower the auxiliary axis to -50, i.e., spatial coordinate (800, 0, -50).

(3) Retract the auxiliary axis, i.e., spatial coordinate (800, 0, 0).

(4) Move to (0, 800, 0) in radius mode, with a radius of 800.

(5) Lower the auxiliary axis to -50, i.e., spatial coordinate (0, 800, 0), while rotating the tool axis forward by 90 degrees.

(6) Retract the auxiliary axis, i.e., spatial coordinate (0, 800, 0), while rotating the tool axis backward by 90 degrees.

(7) Move counterclockwise to (800, 0, 0) in center mode, with (0, 0, 0) as the center.

Declaration and calling of the Scara robot model and motion function blocks

Declare and call the HMC_RobotMove and FB_KimTransl_Scara2_Z_Tool function blocks in PLC_RPG. Also, enable all 4 axes (Axis 0-Axis 3). For convenience, the enable value is directly written as 1, meaning auto-enable.

Parameters for FB_KimTransl_Scara2_Z_Tool: Main arm length is 500, forearm length is 500. Arm configuration is set to TRUE, meaning right-hand configuration.

```
FB_KimTransl_Scara2_Z_Tool(
    bWriteParameter:= ,
    bDone=> ,
    bError=> ,
    dArmLength1:= 500,
    dArmLength2:= 500,
    dOffsetA0:= ,
    dOffsetA1:= ,
    bElbowLow:= TRUE,
    Axis0:= Axis0,
    Axis1:= Axis1,
    Axis2:= Axis2,
    Axis3:= Axis3);
```

大小臂长度

手臂姿态选择

```
HMC_RobotMove(
    bExecute:= ,
    bPause:= ,
    bAbort:= ,
    fOverride:= 1, 速度比例，必须填写
    bSmooth:= ,
    fSmooth_R:= ,
    SmoothPathMode:= ,
    fSmooth_Override:= ,
    pRobotKimTransl:= ADR(FB_KimTransl_Scara2_Z_Tool), 机器人模型地址
    NumberOfCommand:= NumberCommand,   指令使用个数
    stMoveCommand:= stCommand,   运动指令数组名
    bDone=> ,
    bPaused=> ,
    bBusy=> ,
    bError=> ,
    eErrorID=> ,
    dCommandCount=> ,
    dRunCount=> ,
    outCartesianPos=> );
```

Declare motion command array in GVL

According to the example, declare a stMoveParameter array in GVL, using 7 motion commands.

```
GVL  Trace  command  PLC_PRG  Axis0  Axis1  Axis2  Axis3
//{attribute 'qualified_only'}
VAR_GLOBAL
        stCommand                    :ARRAY [1..7] OF stMoveParameter;
END_VAR
```

And fill in the parameters in POUcommand. As per the example, there are 7 motion commands in total:

```
//1.直綫差补移动到 (800,0,0)
stCommand[1].bMoveType      :=FALSE;
stCommand[1].fAcceleration :=300;
stCommand[1].fDeceleration :=300;
stCommand[1].fVelocity      :=100;
stCommand[1].stTargetPos.X :=800;
stCommand[1].stTargetPos.Y :=0;
stCommand[1].stTargetPos.Z :=0;
stCommand[1].stTargetPos.A :=0;

//2.將輔助軸下降50
stCommand[2].bMoveType      :=FALSE;
stCommand[2].fAcceleration :=300;
stCommand[2].fDeceleration :=300;
stCommand[2].fVelocity      :=100;
stCommand[2].stTargetPos.X :=800;
stCommand[2].stTargetPos.Y :=0;
stCommand[2].stTargetPos.Z :=-50;
stCommand[2].stTargetPos.A :=0;

//3.收回輔助軸
stCommand[3].bMoveType      :=FALSE;
stCommand[3].fAcceleration :=300;
stCommand[3].fDeceleration :=300;
stCommand[3].fVelocity      :=100;
stCommand[3].stTargetPos.X :=800;
stCommand[3].stTargetPos.Y :=0;
stCommand[3].stTargetPos.Z :=0;
stCommand[3].stTargetPos.A :=0;
```

```
//4.半径模式下，以800位半径，运动到（0,800,0）
stCommand[4].bMoveType      :=TRUE;
stCommand[4].fAcceleration :=300;
stCommand[4].fDeceleration :=300;
stCommand[4].fVelocity      :=100;
stCommand[4].stArcParameter.eArcMode:=1;
stCommand[4].stArcParameter.fRadius:=800;
stCommand[4].stTargetPos.X :=0;
stCommand[4].stTargetPos.Y :=800;
stCommand[4].stTargetPos.Z :=0;
stCommand[4].stTargetPos.A :=0;

//5.将辅助轴下降50,同时工具轴正向旋转90度
stCommand[5].bMoveType      :=FALSE;
stCommand[5].fAcceleration :=300;
stCommand[5].fDeceleration :=300;
stCommand[5].fVelocity      :=100;
stCommand[5].stTargetPos.X :=0;
stCommand[5].stTargetPos.Y :=800;
stCommand[5].stTargetPos.Z :=-50;
stCommand[5].stTargetPos.A :=90;

//6.收回辅助轴,同时工具轴逆向旋转90度
stCommand[6].bMoveType      :=FALSE;
stCommand[6].fAcceleration :=300;
stCommand[6].fDeceleration :=300;
stCommand[6].fVelocity      :=100;
stCommand[6].stTargetPos.X :=0;
stCommand[6].stTargetPos.Y :=800;
stCommand[6].stTargetPos.Z :=0;
stCommand[6].stTargetPos.A :=-90;


//7.圆心模式下，以（0,0,0）为圆心，顺时针运动到（800,0,0）
stCommand[7].bMoveType      :=TRUE;
stCommand[7].fAcceleration :=300;
stCommand[7].fDeceleration :=300;
stCommand[7].fVelocity      :=100;
stCommand[7].stArcParameter.eArcMode:=0;
stCommand[7].stArcParameter.bDirection:=FALSE;
stCommand[7].stArcParameter.stMidPoint.X:=0;
stCommand[7].stArcParameter.stMidPoint.Y:=0;
stCommand[7].stArcParameter.stMidPoint.Z:=0;
stCommand[7].stTargetPos.X :=800;
stCommand[7].stTargetPos.Y :=0;
stCommand[7].stTargetPos.Z :=0;
stCommand[7].stTargetPos.A :=0;
```

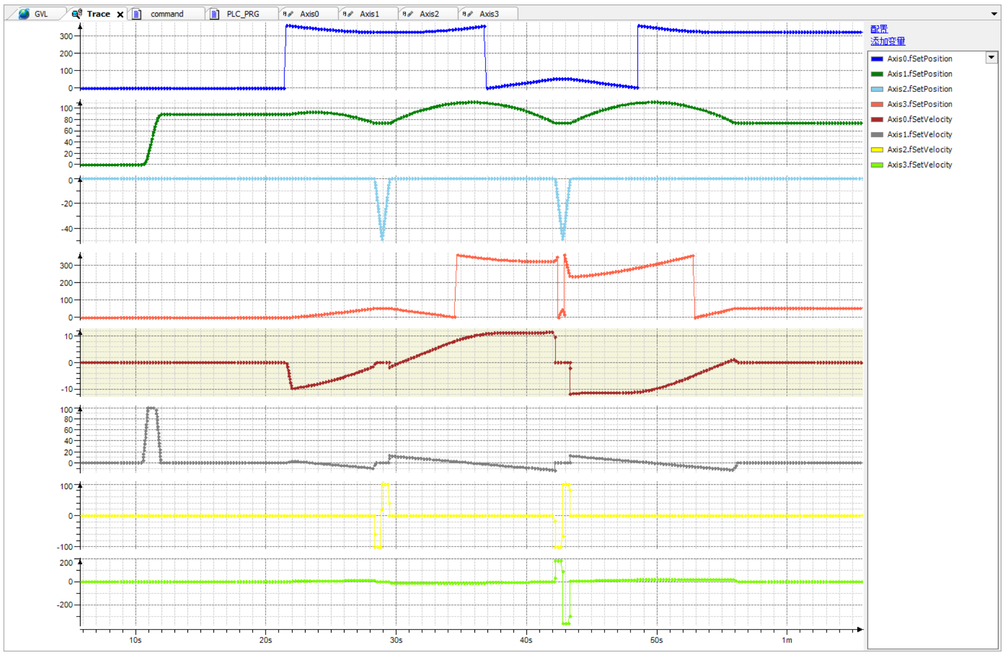
Implementing motion control

In the example simulation program, to adjust the robot's main arm and forearm to a perpendicular state, i.e., to position the end point at (500, 500, 0), the MC_MoveAdditive function block is used to rotate Axis1 forward by 90 degrees.

Note: In actual use, different model function blocks automatically read the current axis positions and convert them to the end point spatial coordinate (x, y, z) under their respective models. In simulation, because the function blocks internally default all axes to the initial position 0, some individual axis positioning operations are needed first to mimic the actual usage scenario. In simulation, first complete individual axis positioning before writing model parameters to the robot function block.

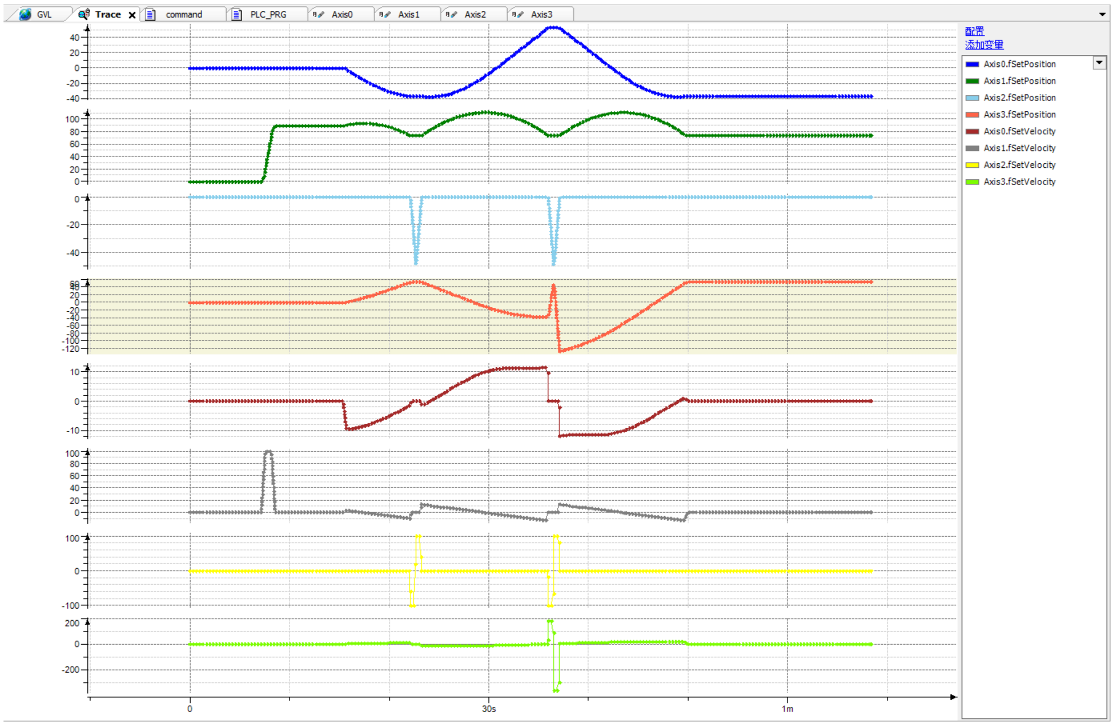
```
MC_MoveAdditive(
    Axis:= Axis1,
    Execute:= ,
    Distance:= 90,
    Velocity:= 100,
    Acceleration:= 300,
    Deceleration:= 300,
    Jerk:= ,
    Done=> ,
    Busy=> ,
    CommandAborted=> ,
    Error=> ,
    ErrorID=> );
```

Trigger the Execute pin of MC_MoveAdditive, the bWriteParameter pin of FB_KimTransl_Scara2_Z_Tool, and the bExecute pin of HMC_RobotMove sequentially. This starts the robot motion according to the 7 commands in the array. Trace the position and velocity of axes Axis0-Axis3. The specific performance is as shown in the figure:



Note: The "step" seen in the trace for Axis 0 and Axis 3 is not runaway motion. This "step" is caused by Axis 0 and Axis 3 having motion commands involving rotation from 0 degrees in the negative direction. (E.g., for a modal axis, rotating from 0 to -10 degrees, because modal axes in CODESYS have no negative values, it displays as 350 degrees.)

To make the visualization more intuitive, Axis0 and Axis3 are changed to linear axes here. Observing the trace plot again, the motion of all four axes appears relatively smooth:



Usage example for jog function block HMC_RobotJog

This example uses the 4-axis Scara robot model. The HMC_RobotJog function block controls the multi-axis motion of the model to achieve linear movement of the model's end point along the positive or negative direction of a single spatial axis (X, Y, or Z).

This example demonstrates controlling the end point from position (500, 500, 0) to move along the positive X-axis to its limit (i.e., arms fully extended). Theoretically, the final end point spatial position is ((√3)*500, 500, 0), meaning Axis0 rotates forward 30 degrees, and Axis1 rotates 0 degrees (arms fully extended).

Declare and call the HMC_RobotJog function block in PLC_RPG:

```
PROGRAM PLC_PRG
VAR
    MC_Power1,MC_Power2,MC_Power3,MC_Power4            :MC_Power;
    HMC_RobotMove                                      :HMC_RobotMove;
    MC_MoveAdditive                                    :MC_MoveAdditive;
    HMC_RobotJog                                       :HMC_RobotJog;
    FB_KimTransl_Scara2_Z_Tool                         :FB_KimTransl_Scara2_Z_Tool;

    HMC_RobotJog(
    fVelocity:= 100,
    fAcceleration:= 1000,
    fDeceleration:= 1000,
    xJogForward:= ,
    xJogBackward:= ,
    yJogForward:= ,
    yJogBackward:= ,
    zJogForward:= ,
    zJogBackward:= ,
    pRobotKimTransl:= ADR(FB_KimTransl_Scara2_Z_Tool),
    bBusy=> ,
    bError=> ,
    outCartesianPos=> );
```

速度设置

机器人模型

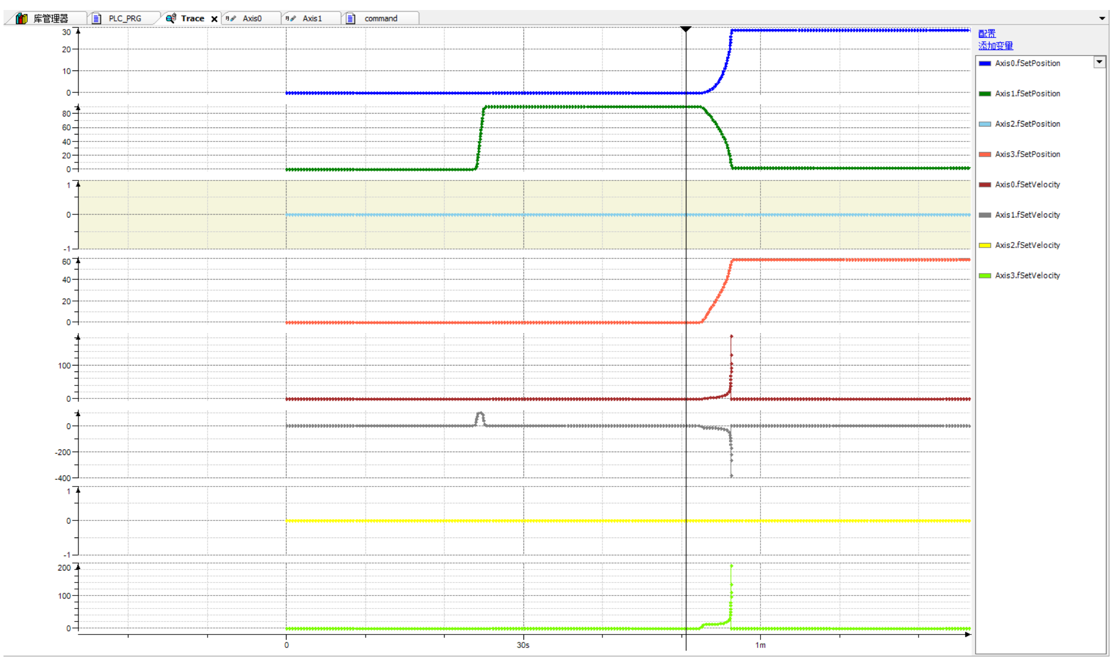
Similar to the command-based robot control above, first use MC_MoveAdditive to rotate Axis 1 forward 90 degrees, positioning the end point at (500, 500, 0). Then use the function block to write model parameters. Specific steps are as follows:

(1) Trigger the Execute pin of MC_MoveAdditive to rotate Axis 1 forward by 90 degrees.

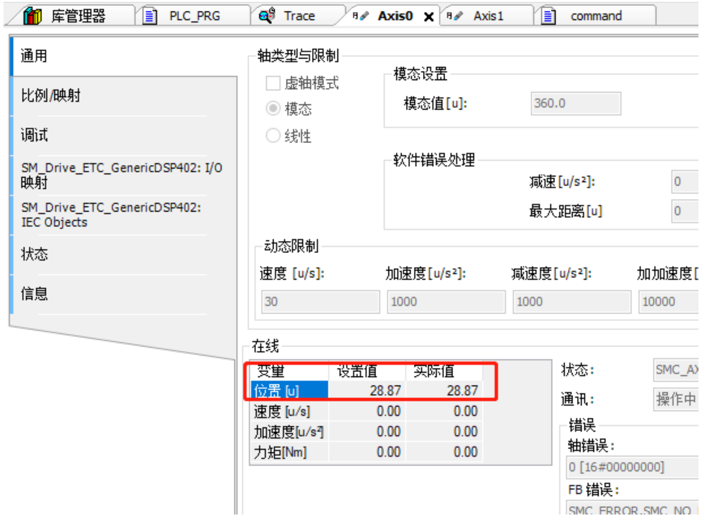
(2) Trigger the bWriteParameter pin of FB_KimTransl_Scara2_Z_Tool to write the model parameters.

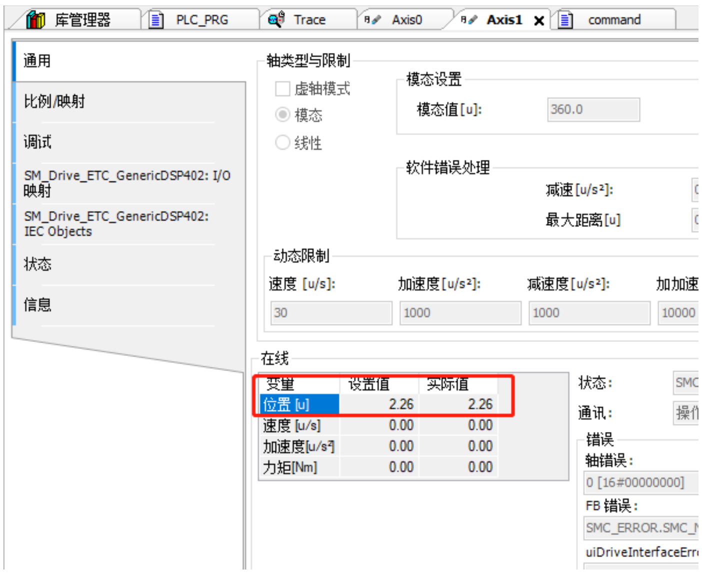
(3) Trigger the xJogForward pin of HMC_RobotJog to control the end point for linear motion along the positive X-axis.

The trace is as shown below:



It can be seen that Axis 0's final rotation angle is 28.87 degrees, and Axis 1's final rotation angle is 2.26 degrees. In practical use, reaching the theoretical state is not always possible, which is normal.


库管理器
PLC_PRG
Trace
Axis0
Axis1
command
通用
比例/映射
调试
SM_Drive_ETC_GenericDSP402: I/O映射
SM_Drive_ETC_GenericDSP402: IEC Objects
状态
信息
轴类型与限制
虚轴模式
模态
线性
模态设置
模态值[u]:
360.0
软件错误处理
减速[u/s²]:
最大距离[u]
0
0
动态限制
速度 [u/s]:
加速度[u/s²]:
减速度[u/s²]:
30
1000
1000
10000
在线
变量
设置值
实际值
位置 [u]
28.87
28.87
速度 [u/s]
0.00
0.00
加速度[u/s²]
0.00
0.00
力矩[Nm]
0.00
0.00
状态:
通讯:
操作中
错误
轴错误:
0 [16#00000000]
FB 错误:

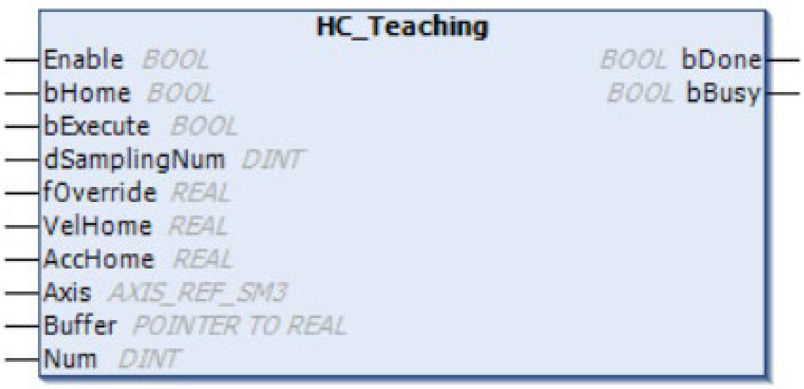

库管理器
PLC_PRG
Trace
Axis0
Axis1
command
通用
比例/映射
调试
SM_Drive_ETC_GenericDSP402: I/O映射
SM_Drive_ETC_GenericDSP402: IEC Objects
状态
信息
轴类型与限制
虚轴模式
模态
线性
模态设置
模态值[u]:
360.0
软件错误处理
减速[u/s²]:
最大距离[u]
动态限制
速度 [u/s]:
加速度[u/s²]:
减速度[u/s²]:
30
1000
1000
10000
在线
变量
设置值
实际值
位置 [u]
2.26
2.26
速度 [u/s]
0.00
0.00
加速度[u/s²]
0.00
0.00
力矩[Nm]
0.00
0.00
状态:
通讯:
错误
轴错误:
0 [16#00000000]
FB 错误:

4.9 Teaching

4.9.1 HC_teaching (FB)

After enabling the teaching function, it records axis positions and replays them, similar to robot lead-through teaching.

◆ **Variables**

<table>
<tr><th>Name</th><th colspan="3">HC_teaching (Teaching function block)</th></tr>
<tr><td>Supported modes</td><td>CSP</td><td>CSV</td><td>CST</td></tr>
<tr><td colspan="2">Graphical representation</td><td colspan="2">ST representation</td></tr>
<tr><td colspan="2">
</td><td colspan="2">HC_Teaching(
Enable:= ,
bHome:= ,
bExecute:= ,
dSamplingNum:= ,
fOverride:= ,
VelHome:= ,
AccHome:= ,
bDone=> ,
bBusy=> ,
Axis:= ,
Buffer:= ,
Num:=);</td></tr>
</table>

(1) Input/Output variable

Input/Output variable	Name	Data type	Valid range	Description
Axis	Axis	AXIS_REF_SM3		
Buffer	Data buffer	ARRAY [*] OF REAL		For recording teaching position information. Declare as a PERSISTENT variable.
Num	Number of array elements	DINT		For recording the count of array elements. Declare as a PERSISTENT variable.

(2) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
Enable	Start teaching	BOOL			When TRUE, starts recording axis positions.
bHome	Homing	BOOL			Returns to the recorded initial position.
bExecute	Enable	BOOL			Replays the teaching process.
dSamplingNum	Sampling cycle count	DINT		1	Sampling cycle count.
fOverride	Teaching speed override	Real		1	Teaching speed override.
VelHome	Homing velocity	Real			Homing velocity.
AccHome	Homing acceleration	Real			Homing acceleration.

(3) Output variable

Output variable	Name	Data type	Valid range	Description
bDone	Done	BOOL		
bBusy	Busy	BOOL		

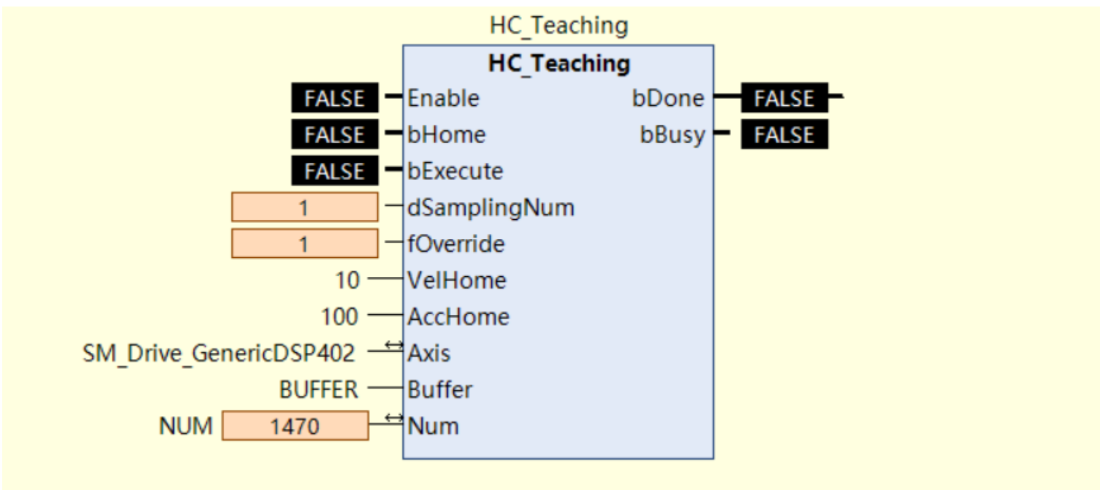
◆ Usage example

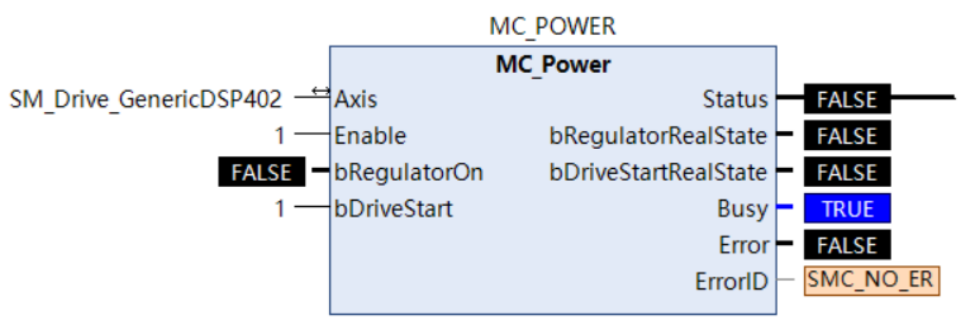
【1】Declare the following variables. (BUFFER, NUM can be declared as ordinary variables if power-off retention is not required.)

```
PROGRAM PLC_PRG
VAR
	HC_Teaching:HC_Teaching;
	MC_POWER:MC_Power;;
END_VAR
VAR RETAIN PERSISTENT
	BUFFER	:ARRAY [0..6000] OF REAL;
	NUM	:DINT;
END_VAR
```

【2】Download and run the program.

With servo power off, trigger Enable to start teaching. Teaching ends when the buffer is full or Enable is set to FALSE. Position data is kept in the buffer, and the data count is recorded.





Device.Application.PersistentVars

表达式	类型	值	准备值
BUFFER[41]	REAL	0.005493164	
BUFFER[42]	REAL	0.005493164	
BUFFER[43]	REAL	0.002746582	
BUFFER[44]	REAL	0.002746582	
BUFFER[45]	REAL	0.005493164	
BUFFER[46]	REAL	0.002746582	
BUFFER[47]	REAL	0.005493164	
BUFFER[48]	REAL	0.002746582	
BUFFER[49]	REAL	0.005493164	
BUFFER[50]	REAL	0.005493164	
BUFFER[51]	REAL	0.002746582	
BUFFER[52]	REAL	0.005493164	
BUFFER[53]	REAL	0.005493164	
BUFFER[54]	REAL	0.002746582	
BUFFER[55]	REAL	0.005493164	
BUFFER[56]	REAL	0.005493164	
BUFFER[57]	REAL	0.002746582	
BUFFER[58]	REAL	0.005493164	
BUFFER[59]	REAL	0.002746582	
BUFFER[60]	REAL	0.002746582	
BUFFER[61]	REAL	0.002746582	
BUFFER[62]	REAL	0.005493164	
BUFFER[63]	REAL	0.005493164	
BUFFER[64]	REAL	0.005493164	
BUFFER[65]	REAL	0.005493164	
BUFFER[66]	REAL	0.005493164	
BUFFER[67]	REAL	0.005493164	
BUFFER[68]	REAL	0.002746582	
BUFFER[69]	REAL	0.005493164	
BUFFER[70]	REAL	[illegible]	

【3】Execute homing

Enable the servo, trigger bHome, and the axis returns to the initial position.

【4】Replay teaching process

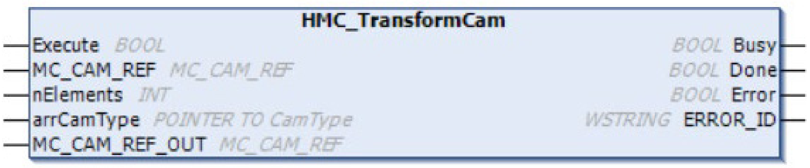
◆ Key points

・ If the axis is a modal axis, a dSamplingNum value that is too large can cause the motor to rotate in the wrong direction during replay. For example, with a sampling cycle of 100, the axis moves from 240 to 80. The forward distance is 200, but the reverse distance is 160. The servo may actually run in reverse to 80.

4.10 TransformCam

4.10.1 HMC_TransformCam (FB)

This function block is used to convert different curves into a Cam table.

Name	HMC_TransformCam		
Supported modes	CSP	CSV	CST
Graphical representation		ST representation	
		HMC_TransformCam(Execute:= , MC_CAM_REF:= , nElements:= , Busy=> , Done=> , Error=> , ERROR_ID=> , arrCamType:= , MC_CAM_REF_OUT:=);	

(1) Input/Output variable

Input/Output variable	Name	Data type	Valid range	Description
MC_CAM_REF_OUT	Converted cam table	MC_CAM_REF		

(2) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bExecute	Enable	BOOL			
MC_CAM_REF	Cam	MC_CAM_REF			
nElements	Number of elements	INT		3601	
arrCamType	Curve type array	ARRAY [*] OF Camtype			

(3) Output variable

Output variable	Name	Data type	Valid range	Description
Done	Done	BOOL		
Busy	Busy	BOOL		
Error	Error	BOOL		
ErrorID	Error ID	WSTRING		

◆ Key points

• CamType (ENUM): Polynomial5=0, 5th-order polynomial; Line=1, line; Sintype=2, trigonometric function; Polynomial3=3, 3rd-order polynomial; parabola=4, parabola; Exponential=5, exponential curve.

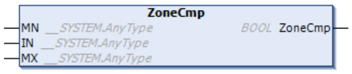
• MC_CAM_REF is the source Cam table. Using its XYVA parameters, it is converted into nElements curves of the types specified in the arrCamType array, and output to MC_CAM_REF_OUT.

Chapter 5 OmronUtils (Omron Instruction Functions)

5.1 Comparison instructions

5.1.1 ZoneCmp (Zone compare)

The instruction determines if the comparison data is between the specified lower and upper limits.

Instruction	Name	FB/FUN	Graphical representation	ST representation
ZoneCmp	Zone compare	FUN		Out:=ZoneCmp(MN, In, MX);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
MN	Lower limit	Input	Lower limit	Conforms to data type.	—	0
In	Comparison data		The value to compare			(*)
MX	Upper limit		Upper limit			0
Out	Comparison result	Output	Comparison result	Conforms to data type.	—	—

*Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
MN						○	○	○	○	○	○	○	○	○	○	○	○	○	○	
																(*)				
In						○	○	○	○	○	○	○	○	○	○	○	○	○	○	
																(*)				
MX						○	○	○	○	○	○	○	○	○	○	○	○	○	○	
																(*)				
Out	○																			

◆ Function

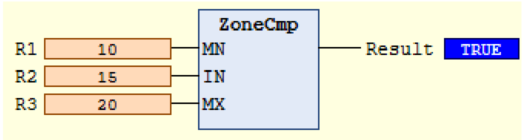
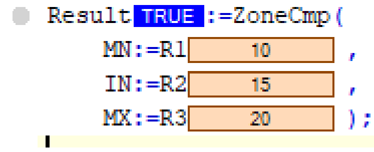
It determines if the comparison data "In" is between the upper limit "MX" and the lower limit "MN".

"Out" is TRUE when "MX" ≥ "In" ≥ "MN"; otherwise, it is FALSE.

Data type	Magnitude relationship
TIME	A value with a larger magnitude is considered greater.
DATE, TOD, DT	For date and time-of-day, the later one is considered greater.

Example with "MN"=INT#10, "In"=INT#20, "MX"=INT#30 is shown below. Variable Result is TRUE.

	FBD	ST
Variable declaration	VAR R1:REAL; R2:REAL; R3:REAL; Result:BOOL; END_VAR	

<table>
<tr><td>Program</td><td></td><td></td></tr>
<tr><td>Result</td><td colspan="2"></td></tr>
</table>

◆ Reference

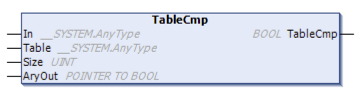
When comparing TIME, DT, or TOD types, match the precision of the values to be compared. The "TruncTime", "TruncDt", and "TruncTod" instructions are available to match value precision.

◆ Key points

- If the data types of "In", "MX", and "MN" differ, they are promoted to a data type encompassing the valid ranges of all involved types before comparison.
- When "In", "MX", or "MN" are real numbers, unexpected results may occur due to rounding errors, e.g., from division that does not yield an exact result.
- Signed integer types (SINT, INT, DINT, LINT) and unsigned integer types (USINT, UINT, UDINT, ULINT) cannot be compared.
- TIME, DATE, TOD, and DT types can only be compared with the same data type. Specifying different types causes an exception during linking.
- + ∞ is considered equal to + ∞ , and – ∞ is considered equal to – ∞ .
- If the value of "In" is Not-a-Number (NaN), "Out" is FALSE.
- When using this instruction in a ladder diagram, if an exception occurs in the circuit preceding this instruction, "Out" is FALSE.
- An exception occurs and "Out" is FALSE under the following conditions:
- When the value of "MN" is greater than the value of "MX".
- When either "MX" or "MN" is Not-a-Number (NaN).

5.1.2 TableCmp (Table compare)

The instruction compares comparison data against multiple defined intervals specified in a comparison table.

Instruction	Name	FB/FUN	Graphical representation	ST representation
TableCmp	Table compare	FUN		Out:=TableCmp(In, Table, Size,AryOut);

◆ Variables

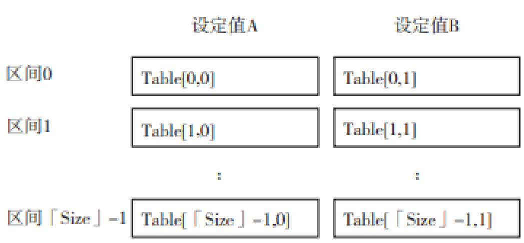
	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Comparison data	Input	The value to be compared.	Conforms to data type.	—	(*)
Table[ ] Two-dimensional array	Comparison table		A two-dimensional array with each defined interval as an element.			
Size	Comparison specification		Number of elements in Table[] to compare with "In".			1
AryOut[ ]	Individual compare Result array	Input/Output	Comparison result for each element of Table[]. TRUE: Match. FALSE: Mismatch.	Conforms to data type.	—	—
Out	Comparison result	Output	TRUE: All elements of Table[] match "In". FALSE: At least one element does not match.	Conforms to data type.	—	—

*Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

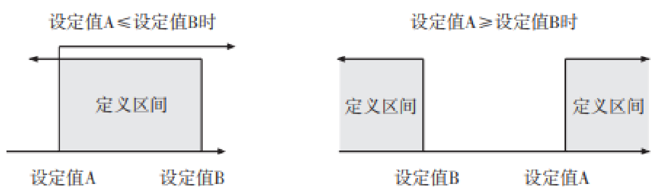
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○	○	○					
Two-dimensional array	A two-dimensional array with elements of the same data type as "In".																			
Size							○													
AryOut[ ] array	○																			
Out	○																			

◆ Function

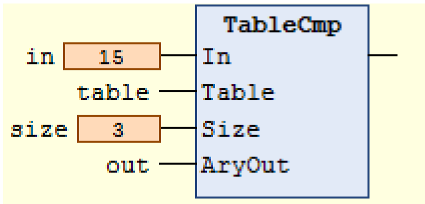
It compares the comparison data "In" with the "Size" groups of defined intervals specified by the comparison table Table[]. Table[] is a two-dimensional array. The first dimension is the interval number. The second dimension's element 0 represents the interval's set value A, and element 1 represents the interval's set value B.



设定值 A 和设定值 B 的定义区间指定方法如下图所示。设定值 A 和设定值 B 的值包含在定义区间内。



“In” 和表 [ ] 的比较结果保存在单独比较结果数组 AryOut[ ] 中。如果 “In” 在编入 i 号的定义区间内，则 AryOut[i] 的值为 TRUE ；如果超出，则 AryOut[i] 的值为 FALSE。如果 AryOut[ ] 的 “Size” 个元素均为 TRUE，则比较结果 “Out” 的值为 TRUE，否则为 FALSE。

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">VAR
    in:INT;
    size:UINT;
    out :ARRAY [1..10] OF BOOL;
    table: ARRAY [1..10,0..1] OF INT;
END_VAR</td></tr>
<tr><td>Program</td><td>
</td><td>TableCmp(
    In:=in ,
    Table:=Table[1,0] ,
    Size:=size ,
    AryOut:=out);</td></tr>
<tr><td>Result</td><td colspan="2"><table>
<tr><th>表达式</th><th>类型</th><th>值</th></tr>
<tr><td>in</td><td>INT</td><td>15</td></tr>
<tr><td>size</td><td>UINT</td><td>3</td></tr>
<tr><td>out</td><td>ARRAY [1..10] OF BOOL</td><td></td></tr>
<tr><td>out[1]</td><td>BOOL</td><td>TRUE</td></tr>
<tr><td>out[2]</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>out[3]</td><td>BOOL</td><td>TRUE</td></tr>
<tr><td>out[4]</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>table</td><td>ARRAY [1..10, 0..1] OF INT</td><td></td></tr>
<tr><td>table[1, 0]</td><td>INT</td><td>10</td></tr>
<tr><td>table[1, 1]</td><td>INT</td><td>20</td></tr>
<tr><td>table[2, 0]</td><td>INT</td><td>20</td></tr>
<tr><td>table[2, 1]</td><td>INT</td><td>10</td></tr>
<tr><td>table[3, 0]</td><td>INT</td><td>4</td></tr>
<tr><td>table[3, 1]</td><td>INT</td><td>20</td></tr>
</table></td></tr>
</table>

◆ Key points

- Set the data type of "In" and the elements of Table[] to be identical. Otherwise, an exception occurs during linking.
- Ensure Table[] is a two-dimensional array.
- If the size of the second dimension of Table[] is greater than or equal to 3, elements with index 2 and beyond in the second dimension are ignored.
- If the size of the AryOut[] array is greater than or equal to "Size", the comparison results are stored in AryOut[0] to AryOut["Size"-1]. Other array elements remain unchanged.
- Signed integer types (SINT, INT, DINT, LINT) and unsigned integer types (USINT, UINT, UDINT, ULINT) cannot be compared.
- When comparing real numbers, unexpected results may occur due to rounding errors, e.g., from division that does not yield an exact result.
- If the value of "Size" is 0, "Out" is FALSE, and AryOut[] remains unchanged.
- When using this instruction in a ladder diagram, if an exception occurs in the circuit preceding this instruction, "Out" is FALSE.
- An exception occurs and "Out" becomes FALSE under the following condition:
- When the array size of the second dimension of Table[] is 1.
- When the value of "Size" exceeds the size of the AryOut[] array.
- When the value of "Size" exceeds the size of the first dimension of the Table[ ] array.

5.1.3 AryCmpNE (Array batch compare - not equal)

The instruction compares corresponding elements of two arrays to determine if they are different.

Instruction	Name	FB/FUN	Graphical representation	ST representation
AryCmpNE	Array whole compare NE	FUN		AryCmpNE(In1, In2, Size, AryOut);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In[ ],In2[ ] array	Comparison data	Input	Array containing the values to be compared.	Conforms to data type.	—	(*)
Size	Number of elements to compare	Input	Number of elements to compare.	Conforms to data type.	—	1
AryOut[ ] array	Comparison result array	Input/Output	Comparison result array.	Conforms to data type.	—	—
Out	Return value	Output	Always TRUE.	TRUE only.	—	—

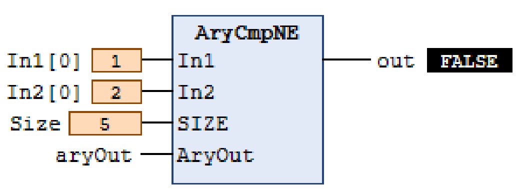
*Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

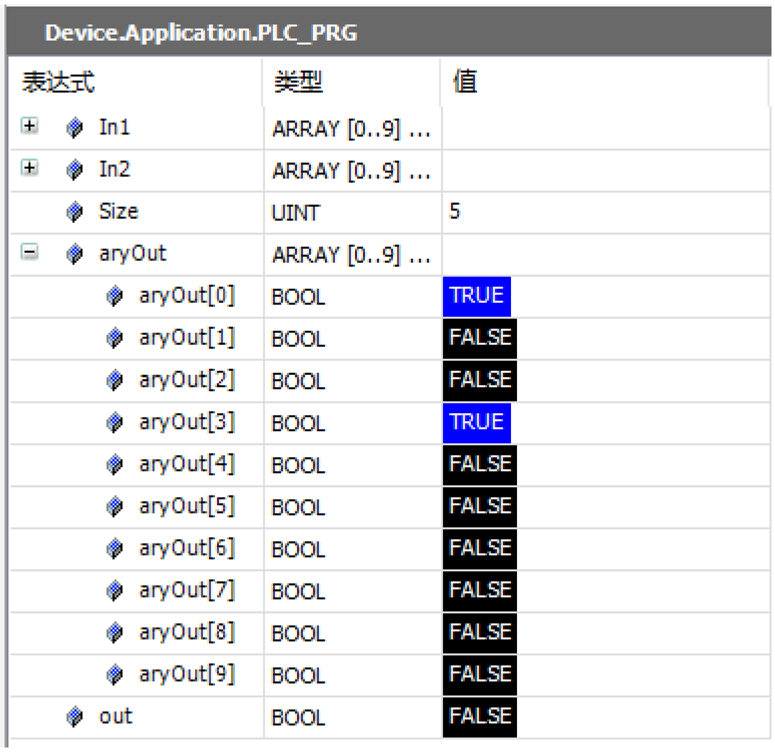
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○					
Two-dimensional array	A two-dimensional array with elements of the same data type as "In".																			
Size							○													
AryOut[ ] array	○																			
Out	○																			

◆ Function

It compares the two arrays In1[0] to In1["Size"-1] and In2[0] to In2["Size"-1] element by element based on their indices. The comparison results are stored in the corresponding elements AryOut[0] to AryOut["Size"-1] of the result array.

The arrays to be compared, In1 and In2, are initialized as shown below. Example with "Size" = 5.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In1: ARRAY[0..9] OF BYTE := [1, 2, 5, 6, 8, 5(0)]; In2: ARRAY[0..9] OF BYTE := [2, 2, 5, 7, 8, 5(0)]; Size :UINT:=5; aryOut :ARRAY [0..9] OF BOOL; out :BOOL; END_VAR	
Program		out FALSE :=AryCmpNE(In1:= In1[0] 1 , In2:= In2[0] 2 , SIZE:= Size 5 , AryOut:= aryOut); RETURN

Result	

Device.Application.PLC_PRG

表达式	类型	值
In1	ARRAY [0..9] ...	
In2	ARRAY [0..9] ...	
Size	UINT	5
aryOut	ARRAY [0..9] ...	
aryOut[0]	BOOL	TRUE
aryOut[1]	BOOL	FALSE
aryOut[2]	BOOL	FALSE
aryOut[3]	BOOL	TRUE
aryOut[4]	BOOL	FALSE
aryOut[5]	BOOL	FALSE
aryOut[6]	BOOL	FALSE
aryOut[7]	BOOL	FALSE
aryOut[8]	BOOL	FALSE
aryOut[9]	BOOL	FALSE
out	BOOL	FALSE

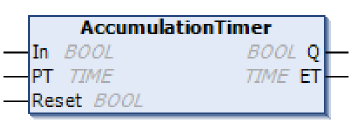
◆ **Key points**

- Set the data types of In1[] and In2[] to be identical.
- Ensure the size of the AryOut[] array is greater than or equal to "Size".
- When In1[] or In2[] contain real numbers, unexpected results may occur due to rounding errors, e.g., from division that does not yield an exact result.
- If the value of "Size" is 0, "Out" is TRUE, and AryOut[] remains unchanged.
- The return value "Out" is not used when employing this instruction in an ST program.
- An exception occurs and ENO becomes FALSE, and AryOut[] remains unchanged under the following condition:
- When the data types of In1[] and In2[] differ.
- When the size of any of the arrays In1[], In2[], or AryOut[] is less than "Size".

5.2 Timer instructions

5.2.1 AccumulationTimer (Accumulation timer)

A timer that accumulates the time during which the timer input is TRUE.

Instruction	Name	FB/FUN	Graphical representation	ST representation
Accumulation Timer	Accumulation timer	FUN		AccumulationTimer(In, PT, Reset, Q, ET);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Timer input	Input	TRUE: Timer operates. FALSE: Timer stops.	Conforms to data type.	—	FALSE
PT	Set time		Maximum time to count.	(*)	ms	0
Reset	Reset		TRUE: Timer resets. FALSE: Timer does not reset.	Conforms to data type.	—	FALSE
Q	Timer output	Output	TRUE: "ET" has reached "PT". FALSE: "ET" has not reached "PT".	Conforms to data type.	—	—
ET	Elapsed time.		Elapsed time.	(*)	ms	

*Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○																			
PT																○				
Reset	○																			
Q	○																			
ET																○				

◆ Function

A timer that accumulates the time during which the timer input "In" is TRUE. The set time unit is ns, with a timing resolution of 100ns.

When "Reset" is FALSE, the timer starts after "In" changes from FALSE to TRUE, and the elapsed time "ET" increases simultaneously with time.

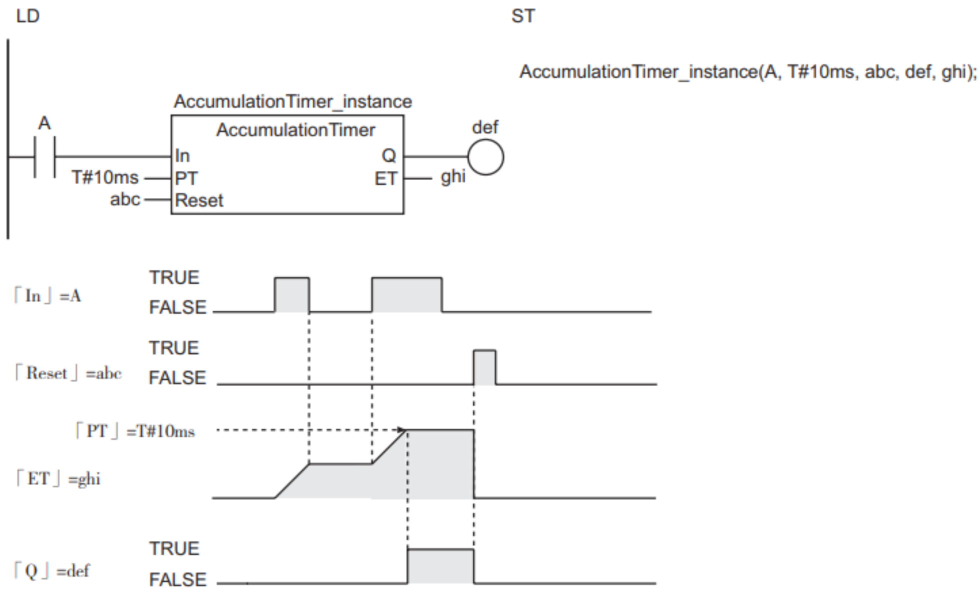
When "In" becomes FALSE, the timer stops. However, the "ET" value at that moment is retained.

When "In" becomes TRUE again, the timer restarts. "ET" increases from the retained value.

When "ET" reaches the set time "PT", the timer output "Q" becomes TRUE. At this point, "ET" stops increasing.

When "Reset" becomes TRUE, the timer resets. The value of "ET" becomes 0, and "Q" becomes FALSE.

“PT” =T#10ms 时的示例和时序图如下所示。变量 A 为 TRUE 的累计时间到达 10ms 时，变量 abc 的值变为 TRUE。



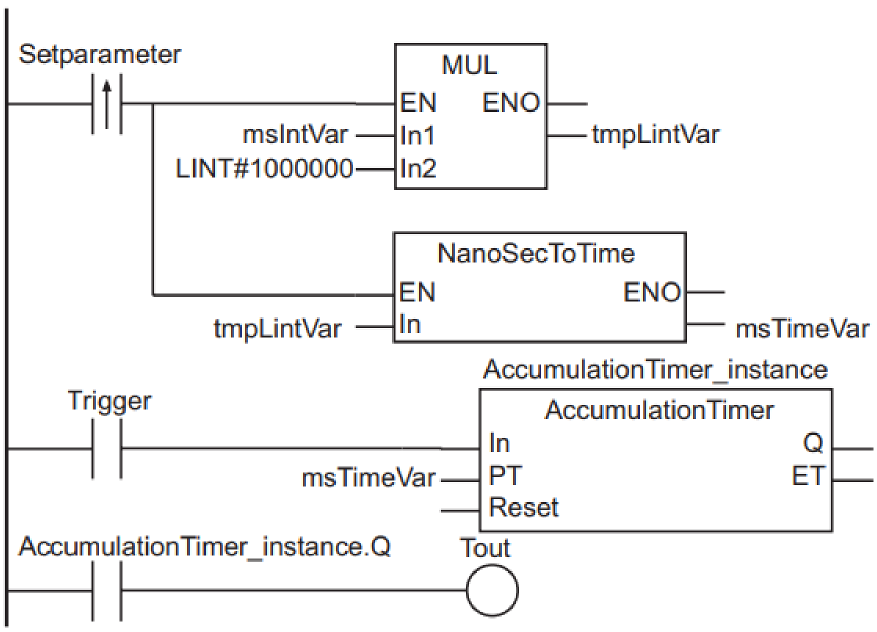
◆ Reference

To reset the timer output and elapsed time when "In" becomes FALSE, use the "TON instruction (P.2-124)".

When connecting to devices like touch panels that do not support the TIME data type, first convert the set time expressed as an integer to the TIME type before inputting it to this instruction.

To convert from an integer type to TIME, use the "NanoSecToTime instruction (P.2-617)". To convert from TIME to an integer type, use the "TimeToNanoSec instruction (P.2-615)". The time unit for these instructions is nanoseconds. A user program for setting time in milliseconds using an INT variable msIntVar is shown below.

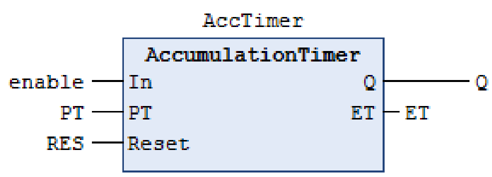
LD



ST

```
tmpLintVar:=msIntVar*LINT#1000000;
msTimeVar:=NanoSecToTime(tmpLintVar);
AccumulationTimer_instance(In:=Trigger, PT:=msTimeVar, Q=>Tout);
```

Usage example is shown below.

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2"><pre>VAR
    enable:BOOL;
    PT:TIME;
    RES:BOOL;
    Q:BOOL;
    ET:TIME;
    AccTimer: AccumulationTimer;
END_VAR</pre></td></tr>
<tr><td>Program</td><td>
</td><td><pre>AccTimer(
    In:=enable ,
    PT:=PT ,
    Reset:=RES ,
    Q=>Q ,
    ET=>ET);</pre></td></tr>
<tr><td>Result</td><td colspan="2"><table>
<tr><th>表达式</th><th>类型</th><th>值</th></tr>
<tr><td>enable</td><td>BOOL</td><td>TRUE</td></tr>
<tr><td>PT</td><td>TIME</td><td>T#1m</td></tr>
<tr><td>RES</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>Q</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>ET</td><td>TIME</td><td>T#6s959ms</td></tr>
<tr><td>AccTimer</td><td>AccumulationTimer</td><td></td></tr>
</table></td></tr>
</table>

◆ Key points

• "ET" and "Q" are updated when this instruction is executed. Therefore, strictly speaking, the condition for "Q" to become TRUE is not the moment the accumulated timer operation time equals "PT", but the moment this instruction is first executed after the accumulated time has reached "PT". Consequently, a delay of up to 1 task cycle may occur.

• "PT" and "ET" are set in units of ns, with a timing resolution of 100ns.

• When both "In" and "Reset" become TRUE, "Reset" takes priority. That is, the value of "ET" becomes 0, and "Q" becomes FALSE.

• If the value of "In" is already TRUE at the start of operation, timing begins from that moment.

• If T#0ms or a negative number is set for "PT", "Q" becomes TRUE after the value of "In" becomes TRUE.

• The value of "PT" can be changed before the value of "ET" reaches it. The behavior in this case is as follows.

Timer status	Value of "Q"	Changed "PT" value	Behavior
After timing completed	TRUE	—	"Q" value remains TRUE. "ET" value also remains unchanged (retains the previous "PT" value before the change).
During timing	FALSE	"PT" ≥ "ET"	When "In" becomes TRUE, timing continues. When "ET" reaches the changed "PT" value, "Q" becomes TRUE and "ET" stops increasing.
		"PT"<"ET"	When "In" becomes TRUE, "Q" immediately becomes TRUE. "ET" also immediately stops increasing.

• This instruction resides in the master control area. When a reset is executed via master control, the behavior is as follows.

• The timer stops. "ET" and "Q" retain their values at that moment.

• When the reset is released via master control, "ET" resumes increasing from the retained value.

・ When "Q" is connected to a subsequent Out instruction, FALSE is input to the Out instruction even if the value of "Q" is TRUE.

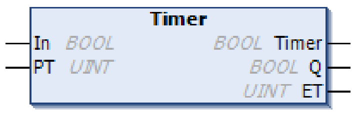
・ "Reset" is effective.

・ If this instruction is not executed due to the execution of a JMP system instruction (JMP instruction, etc.), the value of "ET" is not updated. However, timing continues during this period. Therefore, when this instruction is executed later, "ET" is updated to the correct value.

・ When using this instruction in a ladder diagram, if an exception occurs in the circuit preceding this instruction, the value of "Q" is FALSE.

5.2.2 Timer (100ms timer)

A timer that outputs TRUE after the set time has elapsed from its start. Both the set time unit and timing resolution are 100ms.

Instruction	Name	FB/FUN	Graphical representation	ST representation
Timer	100ms timer	FUN		Out:=Timer (In, PT,TimerDat, Q, ET);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Timer input	Input	TRUE: Timer operates. FALSE: Timer stops.	Conforms to data type.	—	FALSE
PT	Set time		The time from timer start until "Q" becomes TRUE.		ms	(*)
Out	Return value	Output	TRUE: Timer output is ON. FALSE: Timer output is OFF.	Conforms to data type.	—	—
Q	Timer output		Same meaning as "Out".			
ET	Elapsed		Remaining time		ms	

*Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○																			
PT							○													
Reset	○																			
Q	○																			
ET							○													

◆ Function

A timer that outputs TRUE after the set time has elapsed from its start. Both the set time unit and timing unit are 100ms.

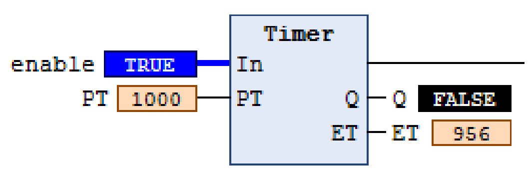
When the timer input "In" becomes FALSE, the timer resets. The remaining time "ET" is set to the set time "PT", and the timer output "Q" becomes FALSE.

When "In" changes from TRUE to FALSE, the timer starts. The value of "ET" decreases simultaneously with time.

When the value of "ET" reaches 0, the timer output "Q" becomes TRUE. At this point, decreasing the value of "ET" stops.

After the timer starts, if "In" becomes FALSE before "ET" reaches 0, the timer resets.

An example and timing diagram for "PT"= UINT#10 are shown below. 1000ms (1s) after the variable enable becomes TRUE, the value of the variable Q becomes TRUE.

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2"><pre>VAR
    enable:BOOL;
    PT:UINT;
    RES:BOOL;
    Q:BOOL;
    ET:UINT;
END_VAR</pre></td></tr>
<tr><td>Program</td><td>
</td><td><pre>Timer(
  In:=enable ,
  PT:=PT ,
  Q=>Q ,
  ET=>ET);</pre></td></tr>
<tr><td>Result</td><td colspan="2">

表达式	类型	值
enable	BOOL	TRUE
PT	UINT	1000
RES	BOOL	FALSE
Q	BOOL	FALSE
ET	UINT	857

</td></tr>
</table>

◆ Reference

For more precise timing, use the "TON instruction (P.2-124)", which times in 100ns units. The TON instruction times in 100ns units during instruction execution, enabling more accurate timing than the Timer instruction. However, the Timer instruction has a shorter execution time.

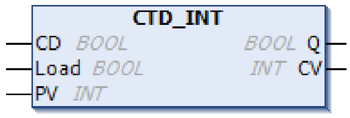
◆ Key points

- Timing occurs at the beginning of the POU describing this instruction. Therefore, regardless of where this instruction is executed within the same POU, the value of "ET" is the same.
- "Q" is updated when this instruction is executed. Strictly speaking, the condition for "Q" to become TRUE is not the moment the elapsed time from timer start equals "PT". The condition is the moment this instruction is first executed after the elapsed time from timer start has reached "PT". Consequently, a delay of up to 1 task cycle may occur.
- "TimerDat" is an input-output variable; no value transfer is needed. Ensure storage for the _sTimer structure and pass it to this instruction.
- Do not change the content of "TimerDat".
- If the value of "In" is already TRUE at the start of operation, timing begins from that moment.
- A change to the value of "PT" takes effect the next time the timer is reset. It is not reflected during the timing process.
- This instruction resides in the master control area. When a reset is executed via master control, the timer resets. The value of "PT" is set in "ET", and the value of "Q" becomes FALSE.
- If this instruction is not executed due to the execution of a JMP system instruction (JMP instruction, etc.), the value of "ET" is not updated. However, timing continues during this period. Therefore, when this instruction is executed later, "ET" is updated to the correct value.
- When using this instruction in a ladder diagram, if an exception occurs in the circuit preceding this instruction, the values of "Q" and "Out" become FALSE.

5.3 Counter instructions

5.3.1 CTD_** (Down counter group)

A counter that performs a decrement operation each time the counter input signal is received. The preset value and count value data type can be any of INT, DINT, LINT, UDINT, or ULINT.

Instruction	Name	FB/FUN	Graphical representation	ST representation
CTD_**	Down counter group	FUN		CTD_** (CD, Load, PV,Q, CV); The double asterisk key (**) refers to any of INT, DINT, LINT, UDINT, ULINT.

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
CD	Counter input	Input	Counter input	Conforms to data type.	—	FALSE
Load	Load signal		TRUE: Loads "PV" into "CV".			
PV	Preset value		Preset value for the counter.	Conforms to data type.		0
Q	Counter output	Output	TRUE: Counter output is ON. FALSE: Counter output is OFF.	Conforms to data type.	—	—
CV	Count value		Current value of the counter.	Conforms to data type.		

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CD	○																			
Load	○																			
PV								○	○			○	○							
Q	○																			
CV	Same data type as "PV".																			

◆ Function

A down counter. The preset value and count value data type can be any of INT, DINT, LINT, UDINT, or ULINT.

The instruction name varies based on the data type of "PV" and "CV". For example, if both are of type LINT, the instruction name is CTD_LINT.

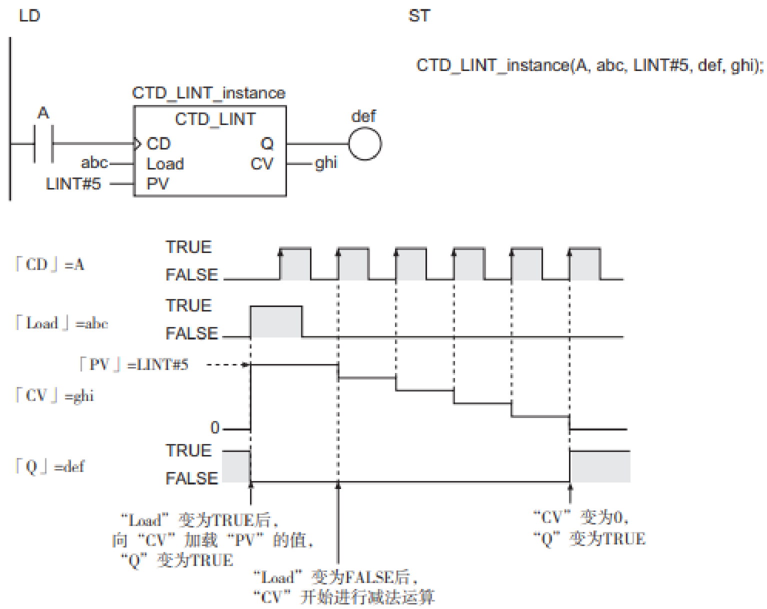
When the load signal "Load" is set to TRUE, the preset value "PV" is loaded into the count value "CV", and the counter output "Q" becomes FALSE.

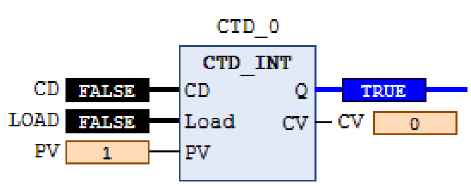
When the counter input signal "CD" is at a rising edge, "CV" is decremented. When the value of "CV" is less than 0, the value of "Q" becomes TRUE.

When the value of "CV" is less than 0, "CV" does not change even if "CD" is at a rising edge.

While "Load" is TRUE, "CD" is ignored. "CV" is not decremented.

An example and timing diagram for the CTD_LINT instruction with "PV"=LINT#5 are shown below.



<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2"><pre>VAR
    CD:BOOL;
    LOAD:BOOL;
    PV:INT;
    Q:BOOL;
    CV:INT;
    CTD_0:CTD_INT;
END_VAR</pre></td></tr>
<tr><td>Program</td><td>
</td><td><pre>CTD_0(
    CD:=CD ,
    LOAD:=LOAD ,
    PV:=PV ,
    Q=>Q ,
    CV=>CV);</pre></td></tr>
<tr><td>Result</td><td colspan="2"><table>
<tr><th>表达式</th><th>类型</th><th>值</th></tr>
<tr><td>CD</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>LOAD</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>PV</td><td>INT</td><td>1</td></tr>
<tr><td>Q</td><td>BOOL</td><td>TRUE</td></tr>
<tr><td>CV</td><td>INT</td><td>0</td></tr>
</table></td></tr>
</table>

◆ Reference

If a counter that performs an increment operation each time the counter input signal is received is needed, use "CTU_**".

If a counter that performs both increment and decrement operations is needed, use the "CTUD instruction (P.2-152)".

◆ Key points

- To restart the counter after the countdown finishes, first set the value of "Load" to TRUE and then to FALSE.
- Set the data types of "PV" and "CV" to be identical.
- If a negative number is set for "PV", when the value of "Load" becomes TRUE, the value of "PV" is loaded into "CV". Since the value of "CV" is less than 0, the value of "Q" immediately becomes TRUE. Subsequently, "CV" is not decremented even if "CD"

changes.

• If power is disconnected or the action mode is controlled by the program while the value of "CD" is FALSE, and then execution of this instruction is restarted, "CV" is decremented once if the value of "CD" becomes TRUE.

• When using this instruction in a ladder diagram, if an exception occurs in the circuit preceding this instruction, the value of "Q" becomes FALSE.

5.3.2 CTU_ ** (Up counter group)

A counter that performs an increment operation each time the counter input signal is received. The preset value and count value data type can be any of INT, DINT, LINT, UDINT, or ULINT.

Instruction	Name	FB/FUN	Graphical representation	ST representation
CTU_**	Up counter group	FUN		CTU_** (CU, Reset, PV, Q, CV); The double asterisk key (**) refers to any of INT, DINT, LINT, UDINT, ULINT.

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
CU	Counter input	Input	Counter input	Conforms to data type.	—	FALSE
Reset	Reset signal		TRUE: Resets "CV" to 0.			
PV	Preset value		Preset value for the counter.	Conforms to data type.		0
Q	Counter output	Output	TRUE: Counter output is ON. FALSE: Counter output is OFF.	Conforms to data type.	—	—
CV	Count value	Input	Current value of the counter.	Conforms to data type.		

*Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	○																			
Reset	○																			
PV								○	○			○	○							
Q	○																			
CV	Same data type as "PV".																			

◆ Function

An up counter. The preset value and count value data type can be any of DINT, LINT, UDINT, or ULINT. The instruction name varies based on the data type of "PV" and "CV". For example, if both are of type LINT, the instruction name is CTU_LINT.

When the reset signal "Reset" is set to TRUE, the count value "CV" becomes 0, and the counter output "Q" becomes FALSE.

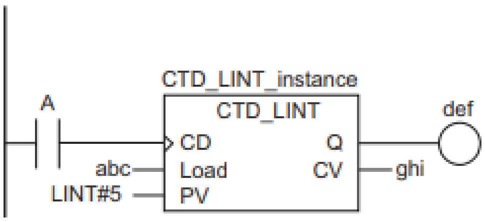
When the counter input signal "CU" is at a rising edge, "CV" is incremented. When the value of "CV" is greater than the value of the preset value "PV", the value of "Q" becomes TRUE.

When the value of "CV" is greater than the value of "PV", "CV" does not change even if a larger value is input to "CU".

While "Reset" is TRUE, "CU" is ignored. "CV" is not incremented.

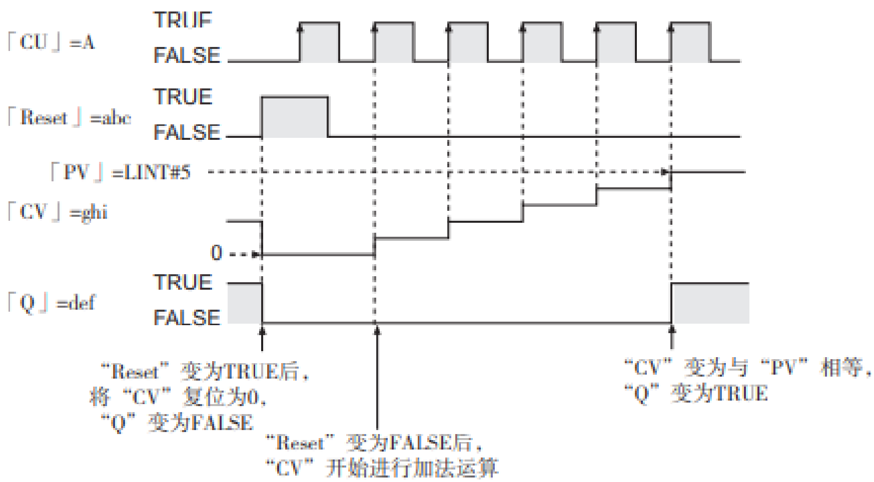
CTU_LINT instruction with "PV"=LINT#5 are shown below.

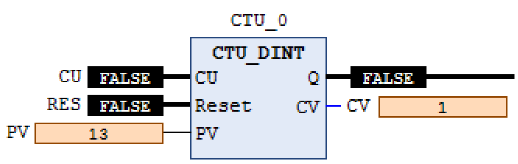
LD



ST

CTD_LINT_instance(A, abc, LINT#5, def, ghi);



	FBD	ST
Variable declaration	VAR CU:BOOL; RES:BOOL; PV:DINT; Q:BOOL; CV:DINT; CTU_0:CTU_DINT; END_VAR	
Program		CTU_0(CU:=CU , Reset:=RES , PV:=PV , Q=>Q , CV=>CV);
Result	表达式 / 类型 / 值 CU / BOOL / FALSE RES / BOOL / FALSE PV / DINT / 13 Q / BOOL / TRUE CV / DINT / 13	

◆ Reference

If a counter that performs a decrement operation each time the counter input signal is received is needed, use the "CTD_** instruction".

If a counter that performs both increment and decrement operations is needed, use the "CTUD_** instruction".

◆ Key points

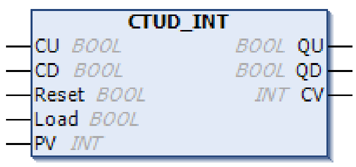
- To restart the counter after counting finishes, first set the value of "Reset" to TRUE and then to FALSE.
- If a negative number is set for "PV", when the value of "Reset" becomes TRUE, the value of "CV" becomes 0. Since the value of "CV" is greater than the value of "PV", the value of "Q" immediately becomes TRUE. Subsequently, "CV" is not incremented even if "CU" changes.
- Set the data types of "PV" and "CV" to be identical.
- If the value of "PV" is changed while the value of "Reset" is FALSE, the behavior is as follows.

Condition	Behavior
Greater than "CV" at that moment	Counting continues.
Less than "CV" at that moment	Counting ends. The value of "Q" becomes TRUE. The value of "CV" remains unchanged at that moment.

- If power is disconnected or the action mode is controlled by the program while the value of "CU" is FALSE, and then execution of this instruction is restarted, "CV" is incremented once if the value of "CU" becomes TRUE.
- When using this instruction in a ladder diagram, if an exception occurs in the circuit preceding this instruction, the value of "Q" becomes FALSE.

5.3.3 CTUD_ (Up/Down counter group)

A counter that performs increment and decrement operations based on the up counter and down counter inputs. The preset value and count value data type can be any of INT, DINT, LINT, UDINT, or ULINT.

Instruction	Name	FB/FUN	Graphical representation	ST representation
CTUD_**	Up/Down counter group	FUN		CTUD_** (CU, CD, Reset, Load, PV, QU, QD, CV); The double asterisk key (**) refers to any of INT, DINT, LINT, UDINT, ULINT.

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
CU	Up counter input	Input	Up counter input	Conforms to data type.	—	FALSE
CD	Down counter input		Down counter input			0
Reset	Reset signal		TRUE: Resets "CV" to 0.			
Load	Load signal		TRUE: Loads "PV" into "CV".			
PV	Preset value		Up counter's count completion value. Down counter's initial value.	Conforms to data type.		
QU	Up counter output	Output	TRUE: Up counter output is ON. FALSE: Up counter output is OFF.	Conforms to data type.	—	—
QD	Down counter output		TRUE: Down counter output is ON. FALSE: Down counter output is OFF.			
CV	Count value		Current value of the counter.	Conforms to data type.		

*Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	○																			
CD	○																			

Reset	○																			
Load	○																			
PV								○	○			○	○							
QU	○																			
QD	○																			
CV	Same data type as "PV".																			

◆ Function

A counter that performs increment and decrement operations based on the up counter and down counter input signals. It combines the functions of both an up counter and a down counter.

The preset value and count value data type can be any of DINT, LINT, UDINT, or ULINT. The instruction name varies based on the data type of "PV" and "CV". For example, if both are of type LINT, the instruction name is CTUD_LINT.

Up counter function

When the reset signal "Reset" is set to TRUE, the count value "CV" becomes 0, and the up counter output "QU" becomes FALSE.

When the up counter input signal "CU" is at a rising edge, "CV" is incremented. When the value of "CV" is greater than the value of the preset value "PV", the value of "QU" becomes TRUE.

When the value of "CV" is greater than the value of "PV", "CV" does not change even if a larger value is input to "CU".

Down counter function

When the load signal "Load" is set to TRUE, the preset value "PV" is loaded into the count value "CV", and the down counter output "QD" becomes FALSE.

When the down counter input signal "CD" is at a rising edge, "CV" is decremented. When the value of "CV" is less than 0, the value of "QD" becomes TRUE.

When the value of "CV" is less than 0, "CV" does not change even if a larger value is input to "CD".

Common functions for up and down counters

While "Load" or "Reset" is TRUE, "CU" and "CD" are ignored. "CV" is not incremented or decremented.

When "CU" and "CD" are simultaneously at a rising edge, "CV" does not change.

When both "Reset" and "Load" are TRUE, "Reset" takes priority, and the value of "CV" becomes 0.

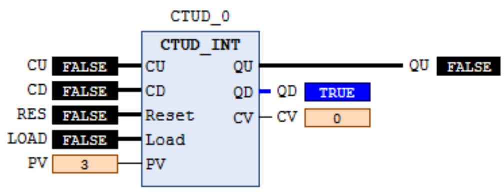
When "Reset" is set to TRUE, the value of "CV" becomes 0, therefore the value of "QD" becomes TRUE.

When "Load" is set to TRUE, the value of "CV" equals "PV", therefore the value of "QU" will become TRUE.

The relationship between "Reset", "Load", "CV", "QU", and "QD" is shown below. The value of "PV" is set greater than 0.

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2"><pre>VAR
    CD:BOOL;
    LOAD:BOOL;
    CU:BOOL;
    RES:BOOL;
    PV:INT;
    QU:BOOL;
    QD:BOOL;
    CV:INT;
    CTUD_0:CTUD_INT;
END_VAR</pre></td></tr>
</table>

Program




```
CTUD_0(
    CU:=CU ,
    CD:=CD ,
    Reset:=RES ,
    Load:=LOAD ,
    PV:=PV ,
    QU=>QU ,
    QD=>QD ,
    CV=>CV );
```

Result

表达式	类型	值
CD	BOOL	FALSE
LOAD	BOOL	FALSE
QD	BOOL	FALSE
CU	BOOL	FALSE
RES	BOOL	FALSE
QU	BOOL	TRUE
PV	INT	3
CV	INT	3

表达式	类型	值
CD	BOOL	FALSE
LOAD	BOOL	FALSE
QD	BOOL	TRUE
CU	BOOL	FALSE
RES	BOOL	FALSE
QU	BOOL	FALSE
PV	INT	3
CV	INT	0

◆ Reference

If a counter that performs a decrement operation each time the counter input signal is received is needed, use the "CTD_** instruction".

If a counter that performs both increment and decrement operations is needed, use the "CTUD_** instruction".

◆ Key points

・To restart the counter after counting finishes, first set the value of "Reset" to TRUE and then to FALSE.

・If a negative number is set for "PV", when the value of "Reset" becomes TRUE, the value of "CV" becomes 0. Since the value of "CV" is greater than the value of "PV", the value of "Q" immediately becomes TRUE. Subsequently, "CV" is not incremented even if "CU" changes.

・Set the data types of "PV" and "CV" to be identical.

・If the value of "PV" is changed while the value of "Reset" is FALSE, the behavior is as follows.

Condition	Behavior
Greater than "CV" at that moment	Counting continues.
Less than "CV" at that moment	Counting ends. The value of "Q" becomes TRUE. The value of "CV" remains unchanged at that moment.

・If power is disconnected or the action mode is controlled by the program while the value of "CU" is FALSE, and then execution of this instruction is restarted, "CV" is incremented once if the value of "CU" becomes TRUE.

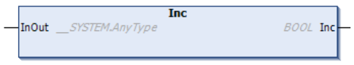
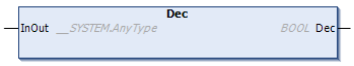
・ When using this instruction in a ladder diagram, if an exception occurs in the circuit preceding this instruction, the value of "Q" becomes FALSE.

5.4 Arithmetic instructions

5.4.1 Inc/Dec (Increment/Decrement)

Inc: Increments an integer value.

Dec: Decrements an integer value.

Instruction	Name	FB/FUN	Graphical representation	ST representation
Inc	Increment	FUN		Inc(InOut);
Dec	Decrement	FUN		Dec(InOut);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Inout	Object data	Input	Object data	Conforms to data type.	—	—
Out	Return value		Always TRUE.	TRUE only.	—	—

*Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut						○	○	○	○	○	○	○	○							
Out	○																			

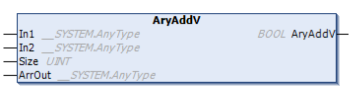
◆ Function

・ Inc: Increments the object data "InOut". When the maximum value of "InOut" is exceeded, it rolls over to the minimum value.

・ Dec: Decrements the object data "InOut". When the minimum value of "InOut" is exceeded, it rolls over to the maximum value.

5.4.2 AryAddV (Array element addition)

The instruction adds the same value to each element of an array.

Instruction	Name	FB/FUN	Graphical representation	ST representation
AryAddV	Array element addition	FUN		AryAddV(In1, In2, Size, AryOut);

◆ Variables

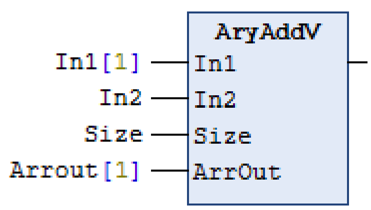
	Name	Input/Output	Description	Valid range	Unit	Initial value
In1[ ] array	Addend array	Input	Addend array	Conforms to data type.	—	(*)
In2	Addend		Addend			
Size	Number of elements		Number of elements in In1[] to be added.			0
Aryout[ ] array	Result array		Result array			—
Out	Return value	Output	Always TRUE.	TRUE only.	—	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[ ] array						○	○	○	○	○	○	○	○	○	○					
In2	Same data type as the source of In1[].																			
Size							○													
AryOut[ ] array	Same data type as the source of In1[].																			
Out	○																			

◆ Function

It adds the value of the addend "In2" to each of the first "Size" elements starting from In1[0] of the addend array In1[], and outputs the result to the result array AryOut[].

An example with "In2"=INT#11 and "Size"=UINT#3 is shown below.

	FBD	ST
Variable declaration	VAR In1: ARRAY [1..10] OF INT := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]; In2 :INT; Size :UINT; Arrout :ARRAY [1..10] OF INT; END_VAR	
Program		AryAddV(In1:=In1[1] 1 , In2:=In2 3 , Size:=Size 10 , ArrOut:=Arrout[1] 4);
Result	see table below	

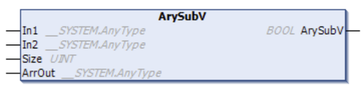
表达式	类型	值	准备值	地址	注释
In1	ARRAY [1..10] OF INT				
In2	INT	3			
Size	UINT	10			
Arrout	ARRAY [1..10] OF INT				
Arrout[1]	INT	4			
Arrout[2]	INT	5			
Arrout[3]	INT	6			
Arrout[4]	INT	7			
Arrout[5]	INT	8			
Arrout[6]	INT	9			
Arrout[7]	INT	10			
Arrout[8]	INT	11			
Arrout[9]	INT	12			
Arrout[10]	INT	13			

◆ Key points

- Set the data types of In1[], "In2", and AryOut[] to be identical.
- If the addition result is outside the valid range of AryOut[], the element in AryOut[] becomes an error value. In this case, no exception occurs, and the adjacent memory areas of the element are not corrupted.
- When the value of "Size" is 0, the values in AryOut[] remain unchanged.
- The return value "Out" is not used when employing this instruction in an ST program.
- An exception occurs and ENO becomes FALSE, and AryOut[] remains unchanged under the following conditions:
- When the data types of In1[], "In2", and AryOut[] differ.
- When the value of "Size" exceeds the array bounds of In1[] or AryOut[].

5.4.3 ArySubV (Array element subtraction)

The instruction subtracts the same value from each element of an array.

Instruction	Name	FB/FUN	Graphical representation	ST representation
ArySubV	Array element subtraction	FUN		ArySubV(In1, In2, Size, AryOut);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In1[ ] array	Minuend array	Input	Minuend array	Conforms to data type.	—	(*)
In2	Addend		Addend			
Size	Number of elements		Number of elements in In1[] to be added.			0
Aryout[array	Result array		Result array			—
Out	Return value	Output	Always TRUE.	TRUE only.	—	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[ ] array						○	○	○	○	○	○	○	○	○	○					
In2	Same data type as the source of In1[].																			
Size							○													
AryOut[ ] array	Same data type as the source of In1[].																			
Out	○																			

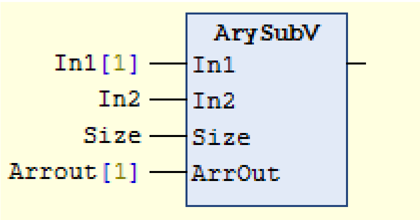
◆ Function

It subtracts the value of the subtrahend "In2" from each of the first "Size" elements starting from In1[0] of the minuend array In1[], and outputs the result to the result array AryOut[].

An example with "In2"=INT#11 and "Size"=UINT#3 is shown below.

	FBD	ST

Variable declaration	VAR
	    In1: ARRAY [1..10] OF INT := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
	    In2 :INT;
	    Size :UINT;
	    Arrout :ARRAY [1..10] OF INT;
	
	END_VAR	

Program		ArySubV(
		    In1:=In1[1] ,
		    In2:=In2 ,
		    Size:=Size ,
		    ArrOut:=Arrout[1]);

Result	see table below	

表达式	类型	值	准备值	地址	注释
In1	ARRAY [1..10] OF INT				
In2	INT	3			
Size	UINT	10			
Arrout	ARRAY [1..10] OF INT				
Arrout[1]	INT	-2			
Arrout[2]	INT	-1			
Arrout[3]	INT	0			
Arrout[4]	INT	1			
Arrout[5]	INT	2			
Arrout[6]	INT	3			
Arrout[7]	INT	4			
Arrout[8]	INT	5			
Arrout[9]	INT	6			
Arrout[10]	INT	7			

◆ Key points

- Set the data types of In1[], "In2", and AryOut[] to be identical.
- If the subtraction result is outside the valid range of AryOut[], the element in AryOut[] becomes an error value. In this case, no exception occurs, and the adjacent memory areas of the element are not corrupted.
- When the value of "Size" is 0, the values in AryOut[] remain unchanged.
- The return value "Out" is not used when employing this instruction in an ST program.
- An exception occurs and ENO becomes FALSE, and AryOut[] remains unchanged under the following conditions:
- When the data types of In1[], In2, and AryOut[] differ.
- When the value of "Size" exceeds the array bounds of In1[] or AryOut[].

5.4.4 AryMean (Array element mean value calculation)

The instruction calculates the mean value of array elements.

Instruction	Name	FB/FUN	Graphical representation	ST representation
AryMean	Array element mean value calculation	FUN		AryMean(In, Size, Out);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In1[ ] array	Source array	Input	Source array	Conforms to data type.	—	(*)
Size	Number of elements in source		Number of elements in In[].			0
Out	Calculation result		Calculation result.	Conforms to data type.	—	—

*Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[ ] array						○	○	○	○	○	○	○	○	○	○	○				
Size							○													
Out						○	○	○	○	○	○	○	○	○	○	○				

◆ Function

It calculates the mean value of the "Size" elements starting from In[0] in the source array In[].

An example with "Size"=UINT#5 is shown below.

	FBD	ST
Variable declaration	VAR In :ARRAY [1..10] OF INT := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] Size :UINT; out :INT; END_VAR	
Program	AryMean In[1] — In Size — Size out — Out	AryMean(In:=In[1] , Size:=Size , Out:=out) ;
Result	(see table below)	

表达式	类型	值	准备值	地址	注释
In	ARRAY [1..10] OF INT				
In[1]	INT	1			
In[2]	INT	2			
In[3]	INT	3			
In[4]	INT	4			
In[5]	INT	5			
In[6]	INT	6			
In[7]	INT	7			
In[8]	INT	8			
In[9]	INT	9			
In[10]	INT	10			
Size	UINT	10			
out	INT	5			

◆ Key points

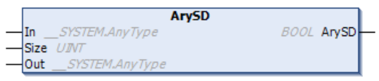
- When In[] and "Out" are integer types, the decimal part of the mean value is truncated.
- When the data types of In[] and "Out" differ, ensure the valid range of "Out" encompasses the valid range of In[].
- If the calculation result exceeds the valid range of "Out", the value of "Out" becomes an error value. No exception occurs

in this case.

- If an intermediate value during calculation exceeds the valid range of In[], the value of "Out" becomes an error value. No exception occurs in this case.
- When the value of "Size" is 0, the value of "Out" is 0.
- An exception occurs and ENO becomes FALSE, and "Out" remains unchanged under the following condition:
- When the value of "Size" exceeds the array bounds of In[].

5.4.5 ArySD (Array element standard deviation)

The instruction calculates the standard deviation of array elements.

Instruction	Name	FB/FUN	Graphical representation	ST representation
ArySD	Array element standard deviation	FUN		ArySD(In, Size, Out);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In1[ ] array	Source array	Input	Source array	Conforms to data type.	—	(*)
Size	Number of elements in source		Number of elements in In[].			0
Out	Calculation result		Calculation result.	Conforms to data type.	—	—

*Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[ ] array						○	○	○	○	○	○	○	○	○	○	○				
Size							○													
Out						○	○	○	○	○	○	○	○	○	○	○				

◆ Function

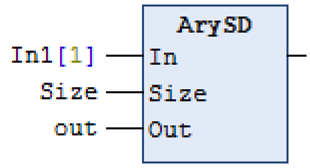
It calculates the standard deviation of the "Size" elements starting from In[0] in the source array In[].

$$\text{Standard deviation} = \sqrt{\frac{\sum_i (\mathrm{In}[i]-\mathrm{InM})^2}{\lceil \mathrm{Size} \rfloor - 1}}$$

i: Index of In[]. 0~ "Size" -1
InM: Mean value of In[0] to In["Size"-1].

An example with "Size"=UINT#5 is shown below.

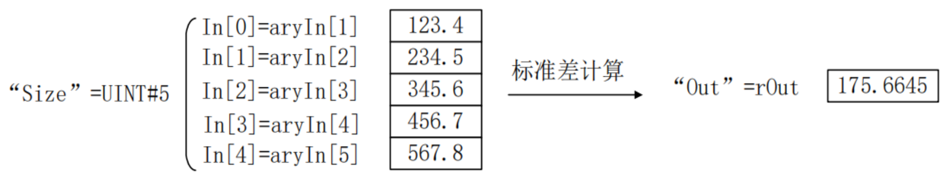
	FBD	ST
Variable declaration	VAR In1 :ARRAY [1..10] OF REAL ; out :REAL; Size :UINT; END_VAR	

Program		ArySD(In:=In[1] , Size:=Size , Out:=out);

Result

表达式	类型	值	准备值	地址	注释
In1	ARRAY [1..10] OF REAL				
In1[1]	REAL	123.4			
In1[2]	REAL	234.5			
In1[3]	REAL	345.6			
In1[4]	REAL	456.7			
In1[5]	REAL	567.8			
In1[6]	REAL	0			
In1[7]	REAL	0			
In1[8]	REAL	0			
In1[9]	REAL	0			
In1[10]	REAL	0			
out	REAL	175.66452			
Size	UINT	5			

Principle analysis

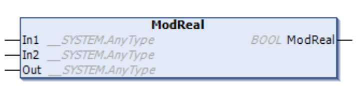


◆ Key points

- When In[] and "Out" are integer types, the decimal part of the mean value is truncated.
- When the data types of In[] and "Out" differ, ensure the valid range of "Out" encompasses the valid range of In[].
- If the calculation result exceeds the valid range of "Out", the value of "Out" becomes an error value. No exception occurs in this case.
- If an intermediate value during calculation exceeds the valid range of In[], the value of "Out" becomes an error value. No exception occurs in this case.
- When the value of "Size" is 0, the value of "Out" is 0.
- An exception occurs and ENO becomes FALSE, and "Out" remains unchanged under the following condition:
- When the value of "Size" exceeds the array bounds of In[].

5.4.6 ModReal (Real number remainder)

The instruction calculates the remainder after dividing real numbers.

Instruction	Name	FB/FUN	Graphical representation	ST representation
ModReal	Real number remainder	FUN		ModReal(In1, In2, Out);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In1[ ]	Dividend	Input	Dividend	Conforms to data type.	—	(*)
In2	Divisor		Divisor			
Out	Remainder		Remainder	Conforms to data type.	—	—

*Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[ ] array														○	○					
Size														○	○					
Out														○	○					

◆ Function

It calculates the remainder after dividing the dividend "In1" by the divisor "In2".

This instruction performs the operation using the following formula.

"Out" = "In1" - ("In1" / "In2") * "In2". The division inside the parentheses truncates the decimal part.

Therefore, example values for "In1", "In2", and "Out" are as follows.

Value of "In1"	Value of "In2"	Value of "Out"
9.9	3.14	0.48
9.9	-3.14	0.48
-9.9	3.14	-0.48
-9.9	-3.14	-0.48

An example with "In1"=REAL#20 and "In2"=REAL#3 is shown below. The value of variable abc is REAL#2.

	FBD	ST
Variable declaration	VAR In1 :REAL; In2 :REAL; out :REAL; END_VAR	
Program	ModReal In1 — In1 In2 — In2 out — Out	ModReal(In1:=In1 , In2:=In2 , Out:=out);
Result	表达式 / 类型 / 值 In1 / REAL / 20 In2 / REAL / 3 out / REAL / 2	

◆ Reference

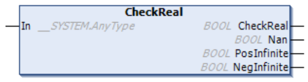
Use the "CheckReal instruction" to check if the value of "Out" is + ∞ , - ∞ , or Not-a-Number (NaN).

◆ Key points

- Use "In1", "In2", and "Out" by setting their attributes to the same type.

5.4.7 CheckReal (Real number check)

The instruction determines if a real number is infinite or Not-a-Number (NaN).

Instruction	Name	FB/FUN	Graphical representation	ST representation
CheckReal	Real number check	FUN		CheckReal(In, Nan, PosInfinite, NegInfinite);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Real number	Input	Real number	Conforms to data type.	—	(*)
Out	Return value	Output	Always TRUE.	TRUE only.	—	(*)
Nan	NaN determination result	Output	TRUE: NaN. FALSE: Not NaN.	Conforms to data type.	—	—
PosInfinite	Positive infinity determination result	Output	TRUE: Positive infinity. FALSE: Not positive infinity.	Conforms to data type.	—	—
NegInfinite	Negative infinity determination result	Output	TRUE: Negative infinity. FALSE: Not negative infinity.	Conforms to data type.	—	—

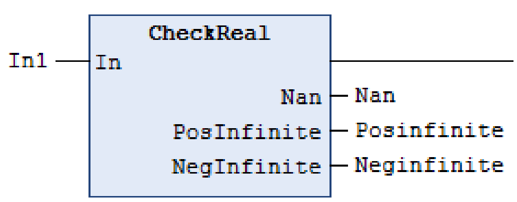
*Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out	○																			
Nan	○																			
PosInfinite	○																			
NegInfinite	○																			

◆ Function

It determines if the real number "In" is NaN, positive infinity, or negative infinity, and outputs the results to "Nan", "PosInfinite", and "NegInfinite" respectively.

Usage example with "In"=10#2.44 is shown below.

	FBD	ST
Variable declaration	VAR In1 :REAL; Nan :BOOL; Posinfinite:BOOL; Neginfinite:BOOL; END_VAR	
Program		CheckReal(In:=In1 , Nan=>Nan , PosInfinite=>Posinfinite , NegInfinite=>Neginfinite);

Result	

表达式	类型	值
In1	REAL	2.44
Nan	BOOL	FALSE
Posinfinite	BOOL	FALSE
Neginfinite	BOOL	FALSE

◆ Reference

Use this instruction to detect if the result of an arithmetic operation using real numbers is NaN, positive infinity, or negative infinity.

◆ Key points

- The return value "Out" is not used when employing this instruction in an ST program.

5.5 Bit string operation instructions

5.5.1 AryAnd/AryOr/AryXor/AryXorN

The instruction performs bitwise logical operations on each corresponding bit of each element between two arrays for BOOLean and bit string data types.

AryAnd: Logical AND.

AryOr: Logical OR.

AryXor: Exclusive OR.

AryXorN: Exclusive NOR.

Instruction	Name	FB/FUN	Graphical representation	ST representation
AryAnd	Array logical AND	FUN		AryAnd(In1, In2, Size, AryOut);
AryOr	Array logical OR	FUN		AryOr(In1, In2, Size, AryOut);
AryXor	Array exclusive OR	FUN		AryXor(In1, In2, Size, AryOut);
AryXorN	Array exclusive NOR	FUN		AryXorN(In1, In2, Size, AryOut);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In1[ ],In2[ ] array	Source array	Input	Source array	Conforms to data type.	—	(*)
Size	Number of elements	Input	Number of source elements.	Conforms to data type.	—	0
AryOut[ ] array	Result array	Input	Result array	Conforms to data type.	—	—
Out	Return value	Output	Always TRUE.	TRUE only.	—	—

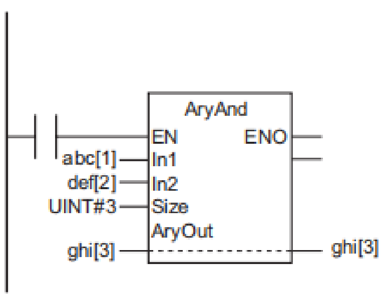
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[ ] array	○	○	○	○	○															
Out	Same data type as In1[].																			
Size							○													
AryOut[ ]	Same data type as In1[].																			
Out	○																			

◆ Function

It performs a logical operation on each corresponding bit of the first "Size" elements of the source arrays In1[] and In2[]. The result is stored in the corresponding element of AryOut[]. Therefore, the data types of In1[], In2[], and AryOut[] must be identical.

An example for the AryAnd instruction with "Size"=UINT#3 is shown below.

LD

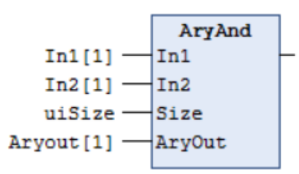


ST

AryAnd(abc[1], def[2], UINT#3, ghi[3]);

「Size」=UINT#3

In1[0]=abc[1]	TRUE	AND	In2[0]=def[2]	TRUE	→	AryOut[0]=ghi[3]	TRUE
In1[1]=abc[2]	FALSE	AND	In2[1]=def[3]	TRUE	→	AryOut[1]=ghi[4]	FALSE
In1[2]=abc[3]	FALSE	AND	In2[2]=def[4]	FALSE	→	AryOut[2]=ghi[5]	FALSE

	FBD	ST
Variable declaration	VAR In1 :ARRAY[1..5] OF BYTE; In2 :ARRAY[1..5] OF BYTE; Aryout :ARRAY[1..5] OF BYTE; uiSize :UINT:=5; END_VAR	
Program		AryAnd(In1:=In1[1] , In2:=In2[1] , Size:=uiSize , AryOut:=Aryout[1]);

Result	In1	ARRAY [1..5] OF BYTE	
	In1[1]	BYTE	2#11111111
	In1[2]	BYTE	2#11111111
	In1[3]	BYTE	2#11111111
	In1[4]	BYTE	2#11111111
	In1[5]	BYTE	2#11111111
	In2	ARRAY [1..5] OF BYTE	
	In2[1]	BYTE	2#11110000
	In2[2]	BYTE	2#00001111
	In2[3]	BYTE	2#10101010
	In2[4]	BYTE	2#00000000
	In2[5]	BYTE	2#11111111
	Aryout	ARRAY [1..5] OF BYTE	
	Aryout[1]	BYTE	2#11110000
	Aryout[2]	BYTE	2#00001111
	Aryout[3]	BYTE	2#10101010
	Aryout[4]	BYTE	2#00000000
	Aryout[5]	BYTE	2#11111111

◆ Key points

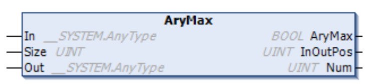
- Set the data types of In1[], In2[], and AryOut[] to be identical.
- Set the number of elements in the AryOut[] array to be greater than or equal to "Size".
- When the value of "Size" is 0, the values in AryOut[] remain unchanged.
- The return value "Out" is not used when employing this instruction in an ST program.
- An exception occurs and ENO becomes FALSE, and AryOut[] remains unchanged under the following conditions:
- When the data types of In1[], In2[], and AryOut[] differ.
- When the value of "Size" exceeds the number of elements in any of In1[], In2[], or AryOut[].

5.6 Selection instructions

5.6.1 AryMax/AryMin (Array variable maximum /minimum retrieval)

AryMax: Retrieves the maximum value of 1-dimensional array elements.

AryMin: Retrieves the minimum value of 1-dimensional array elements.

Instruction	Name	FB/FUN	Graphical representation	ST representation
AryMax	Array variable maximum retrieval	FUN		Out:=AryMax(In, Size, InOutPos, Num);
AryMin	Array variable minimum retrieval	FUN		Out:=AryMin(In, Size, InOutPos, Num);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In[ ] array	Source array	Input	Source array	Conforms to data type.	—	(*)
Size	Number of source elements		Number of elements in In[] to be examined.			0
InOutPos	Element index	Output	Element index of the retrieved value.		—	—
Out	Retrieval result	Input	Retrieval result.		—	—
Num	Number of retrieved values	Output	Number of retrieved values.			

*Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ] array						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Size							○													
InOutPos							○													
Out						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Out							○													

◆ Function

It examines the first "Size" array elements starting from In[0] of the source array In[]. The retrieved value is assigned to "Out", its element index to "InOutPos", and the number of retrieved values to "Num". If "Num" is greater than 1, "InOutPos" holds the index of the lowest element among the retrieved values.

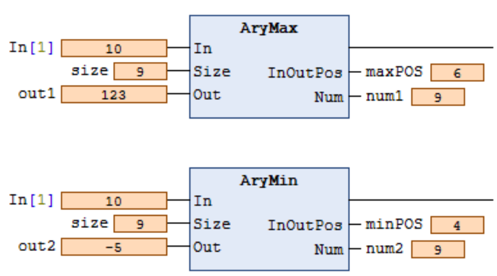
The magnitude relationship for data types other than integer and real numbers is determined as shown in the table below.

Instruction	Name
TIME	A value with a larger magnitude is considered greater.
DATE, TOD, Dt	For dates and time-of-day, the later one is considered greater.
STRING	Determined by comparing ASCII codes of characters.

AryMax: Retrieves the maximum value.

AryMin: Retrieves the minimum value.

An example for the AryMax instruction with "Size"=UINT#6 is shown below. The input parameter sent to In[] is abc[2], so elements from abc[2] onward become the retrieval objects.

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2"><pre>VAR
    In   :ARRAY [1..10] OF REAL;
    size   :UINT;
    out1   :REAL;
    out2   :REAL;
    maxPOS   :UINT;
    minPOS   :UINT;
    num1   :UINT;
    num2   :UINT;
END_VAR</pre></td></tr>
<tr><td>Program</td><td>
</td><td><pre>AryMax(
    In:=In[1] ,
    Size:=size ,
    Out:=out1 ,
    InOutPos=>maxPOS ,
    Num=>num1);

AryMin(
    In:=In[1] ,
    Size:=size ,
    Out:=out2 ,
    InOutPos=>minPOS ,
    Num=>num2);</pre></td></tr>
</table>

Result

表达式	类型	值	准备值	地址	注释
In	ARRAY [1..10] OF REAL				
In[1]	REAL	10			
In[2]	REAL	23			
In[3]	REAL	12			
In[4]	REAL	3			
In[5]	REAL	-5			
In[6]	REAL	3			
In[7]	REAL	123			
In[8]	REAL	-2			
In[9]	REAL	1			
In[10]	REAL	3			
size	UINT	9			
out1	REAL	123			
out2	REAL	-5			
maxPOS	UINT	6			
minPOS	UINT	4			
num1	UINT	9			
num2	UINT	9			

◆ Reference

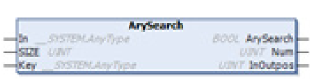
When comparing TIME, DT, or TOD data types, ensure the precision of the values to be compared is aligned. The "TruncTime", "TruncDt", and "TruncTod" instructions are available to match the value precision.

◆ Key points

- When the data types of In[] and "Out" differ, ensure the valid range of "Out" encompasses the valid range of In[].
- When In[] contains real numbers, the expected result may not be obtained due to rounding errors.
- Ensure In[] is a one-dimensional array.
- When the value of "Size" is 0, the values of "Out" and "Num" become 0. The value of "InOutPos" remains unchanged.
- When In[] is of STRING type and the value of "Size" is 0, "Out" contains only the null character.
- An exception occurs and ENO becomes FALSE, and "Out" remains unchanged under the following conditions:
- When the value of "Size" exceeds its valid range.
- When the Size exceeds the array bounds of In[].
- When In[] is not a one-dimensional array.
- When In[] is of STRING type and is not NULL-terminated.
- When In[] is of STRING type and contains more characters than the size of "Out".

5.6.2 ArySearch (Array search)

The instruction searches for a specified value within a 1-dimensional array.

Instruction	Name	FB/FUN	Graphical representation	ST representation
ArySearch	Array search	FUN		Out:=ArySearch(In, Size, Key, InOutPos, Num);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In[ ] array	Source array	Input	Source array	Conforms to data type.	—	(*)
Size	Number of source elements		Number of elements in In[] to be searched.			0
Key	Search key		Value to search for.		—	
InOutPos	Element index	Output	Element index of the found value.		—	—
Out	Search result		Search result.			—
Num	Number of found values		Number of found values.			

*Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ] array	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Arrays of enumerated types can also be specified.																			
Size							○													
Key	Same data type as the source of In[].																			
InOutPos							○													
Out	○																			
Num							○													

◆ Function

It searches the first "Size" elements starting from In[0] of the 1-dimensional source array In[] for elements equal to the search key "Key". The search result "Out", element index "InOutPos", and number of found values "Num" are as shown in the table below.

Presence of element(s) equal to "Key"	"Out"	"InOutPos"	"Num"
Present	TRUE	The index of the lowest element among those equal to "Key".	Number of elements equal to "Key".
Absent	FALSE	Unchanged	0

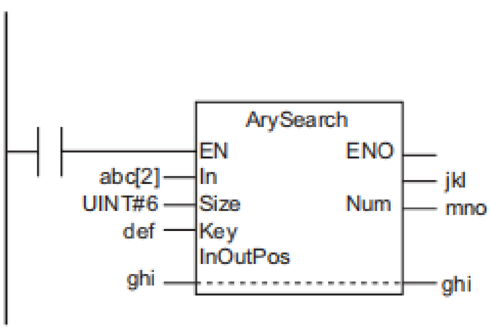
The magnitude relationship for data types other than integer and real numbers is determined as shown in the table below.

Instruction	Name
TIME	A value with a larger magnitude is considered greater.
DATE, TOD, Dt	For dates and time-of-day, the later one is considered greater.

An example with "Size"=UINT#6 is shown below.

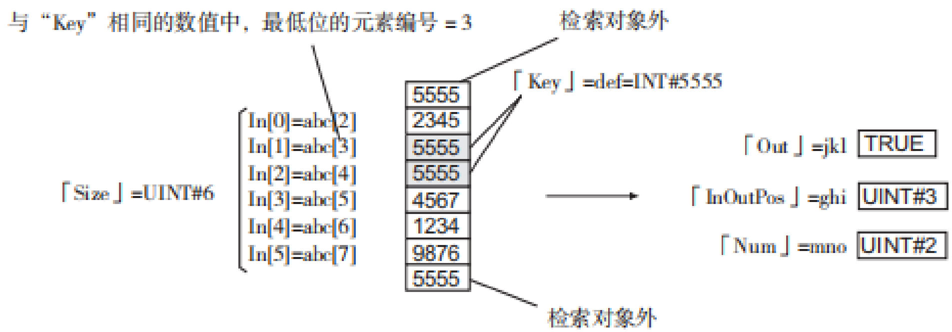
The input parameter sent to In[] is abc[2], so elements from abc[2] onward become the search objects.

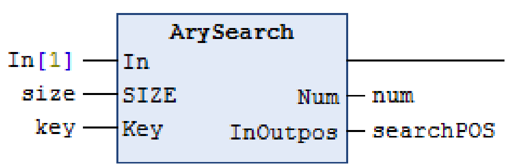
LD



ST

jkl:=ArySearch(abc[2], UINT#6, def, ghi, mno);



<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">VAR
In :ARRAY [1..10] OF REAL;
size :UINT;
key :REAL;
searchPOS :UINT;
num :UINT;
END_VAR</td></tr>
<tr><td>Program</td><td></td><td>ArySearch(
In:=In[1] ,
SIZE:= size,
Key:=key ,
Num=>num ,
InOutpos=>searchPOS);</td></tr>
<tr><td>Result</td><td colspan="2"><table>
<tr><th>表达式</th><th>类型</th><th>值</th><th>准备值</th><th>地址</th><th>注释</th></tr>
<tr><td>In</td><td>ARRAY [1..10] OF REAL</td><td></td><td></td><td></td><td></td></tr>
<tr><td>In[1]</td><td>REAL</td><td>10</td><td></td><td></td><td></td></tr>
<tr><td>In[2]</td><td>REAL</td><td>23</td><td></td><td></td><td></td></tr>
<tr><td>In[3]</td><td>REAL</td><td>12</td><td></td><td></td><td></td></tr>
<tr><td>In[4]</td><td>REAL</td><td>3</td><td></td><td></td><td></td></tr>
<tr><td>In[5]</td><td>REAL</td><td>-5</td><td></td><td></td><td></td></tr>
<tr><td>In[6]</td><td>REAL</td><td>3</td><td></td><td></td><td></td></tr>
<tr><td>In[7]</td><td>REAL</td><td>123</td><td></td><td></td><td></td></tr>
<tr><td>In[8]</td><td>REAL</td><td>-2</td><td></td><td></td><td></td></tr>
<tr><td>In[9]</td><td>REAL</td><td>1</td><td></td><td></td><td></td></tr>
<tr><td>In[10]</td><td>REAL</td><td>3</td><td></td><td></td><td></td></tr>
<tr><td>size</td><td>UINT</td><td>10</td><td></td><td></td><td></td></tr>
<tr><td>key</td><td>REAL</td><td>3</td><td></td><td></td><td></td></tr>
<tr><td>searchPOS</td><td>UINT</td><td>3</td><td></td><td></td><td></td></tr>
<tr><td>num</td><td>UINT</td><td>3</td><td></td><td></td><td></td></tr>
</table></td></tr>
</table>

◆ Reference

When comparing TIME, DT, or TOD types, match the precision of the values to be compared. The "TruncTime", "TruncDt", and "TruncTod" instructions are available to match value precision.

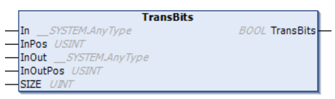
◆ Key points

- Ensure "Key" is of the same data type as the elements of In[].
- When the value of "Size" is 0, the values of "Out" and "Num" become 0. The value of "InOutPos" remains unchanged.
- Always use a variable as the input parameter for "Key". An exception occurs during linking if a constant is passed.
- When "Key" is an enumerated type, enumeration elements cannot be passed directly. An exception occurs during linking if passed directly.
- An exception occurs and ENO becomes FALSE, and "Out", "Num", and "InOutPos" remain unchanged under the following conditions:
  - When "Size" exceeds the array bounds of In[].
  - When In[] or "Key" is of STRING type and is not null-terminated.
  - When In[] is not a one-dimensional array.

5.7 Data transfer instructions

5.7.1 TransBits (Multi-bit transfer)

The instruction transfers multiple bits within a bit string.

Instruction	Name	FB/FUN	Graphical representation	ST representation
TransBits	Multi-bit transfer	FUN		TransBits(In, InPos, InOut, InOutPos, Size);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Source	Input	Source	Conforms to data type.	—	(*1)
InPos	Source bit position		Transfer start bit position in "In".	(*2)		0
InOutPos	Destination bit position		Transfer start bit position in "Out".	(*3)	—	
Size	Number of bits		Number of bits to transfer.	(*4)		1
InOut	Destination		Destination	Conforms to data type.	—	—
Out	Return value	Output	Always TRUE.	TRUE only.	—	—

*1 Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

*2 0 to (number of bits in "In") - 1

*3 0 to (number of bits in "InOut") - 1

*4 0 to (number of bits in "In")

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
InPos						○														

InOutPos						○														
Size						○														
InOut		○	○	○	○															
Out	○																			

◆ Function

It transfers "Size" bits from the bit position "InPos" in the source "In" to the bit position "InOutPos" in the destination "InOut".

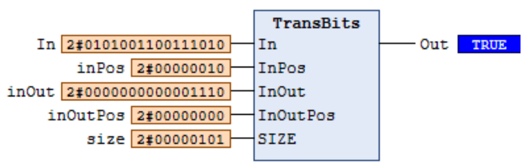
An example with "In"=2#0101001100111010, "InPos"=USINT#2, "InOutPos"=USINT#0, "Size"=USINT#5 is shown below.

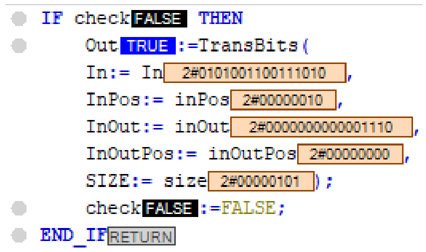
	FBD	ST

Variable declaration

```
PROGRAM PLC_PRG
VAR
    check       :BOOL;
    In          :WORD:=2#0101001100111010;
    inPos       :USINT:=2;
    inOutPos    :USINT;
    size        :USINT:=5;
    inOut       :WORD;
    Out         :BOOL;
END_VAR
```

Program





Result

表达式	类型	值
check	BOOL	FALSE
In	WORD	2#0101001100111010
inPos	USINT	2#00000010
inOutPos	USINT	2#00000000
size	USINT	2#00000101
inOut	WORD	2#0000000000001110
Out	BOOL	TRUE

◆ Reference

Overlap between the source and destination data areas is allowed.

◆ Key points

・ Do not specify source or destination bit positions beyond the highest bit of "In" and "InOut". This causes an exception, and no action is taken.

・ When the value of "Size" is 0, no transfer occurs.

・ Bits in "InOut" not involved in the transfer remain unchanged.

・ The return value "Out" is not used when employing this instruction in an ST program.

・ An exception occurs and ENO becomes FALSE, and "InOut" remains unchanged under the following conditions:

・ When the value of "InPos" exceeds its valid range.

・ When the value of "InOutPos" exceeds its valid range.

- When the value of "Size" exceeds its valid range.
- When the specification of "InPos" and "Size" exceeds the number of bits in "In".
- When the specification of "InOutPos" and "Size" exceeds the number of bits in "InOut".

5.7.2 SetBlock (Block set)

The instruction transfers the value of a variable or constant to multiple array elements.

Instruction	Name	FB/FUN	Graphical representation	ST representation
SetBlocks	Block set	FUN		SetBlock(In, AryOut, Size);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Source	Input	Source	Conforms to data type.	—	(*)
Size	Number of elements		Number of array elements to transfer.			1
AryOut [ ] array	Destination array		Destination array	Conforms to data type.	—	—
Out	Return value	Output	Always TRUE.	TRUE only.	—	—

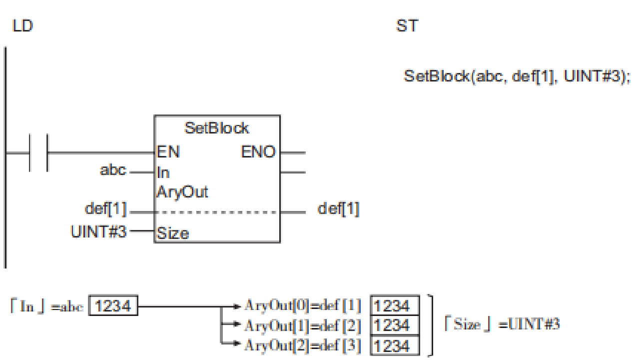
* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

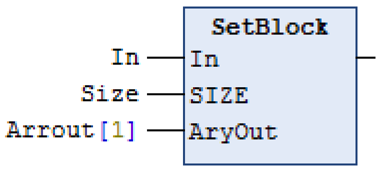
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Enumerated types, entire structures, or a single structure element can also be specified.																			
Size						○														
Aryout[ ] array	An array whose elements are of the same data type as "In".																			
Out	○																			

◆ Function

It transfers the value of the source "In" to the first "Size" positions starting from AryOut[0] of the destination array AryOut[].

An example with "Size"=UINT#3 is shown below.



<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">VAR
    In:REAL;
    Size:USINT;
    Arrout:ARRAY [1..10] OF REAL;
END_VAR</td></tr>
<tr><td>Program</td><td>
</td><td>SetBlock(
    In:= In,
    SIZE:= Size,
    AryOut:=Arrout[1]);</td></tr>
<tr><td>Result</td><td colspan="2">
<table>
<tr><th>表达式</th><th>类型</th><th>值</th><th>准备值</th><th>地址</th><th>注释</th></tr>
<tr><td>In</td><td>REAL</td><td>4</td><td></td><td></td><td></td></tr>
<tr><td>Size</td><td>USINT</td><td>6</td><td></td><td></td><td></td></tr>
<tr><td>Arrout</td><td>ARRAY [1..10] OF REAL</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Arrout[1]</td><td>REAL</td><td>4</td><td></td><td></td><td></td></tr>
<tr><td>Arrout[2]</td><td>REAL</td><td>4</td><td></td><td></td><td></td></tr>
<tr><td>Arrout[3]</td><td>REAL</td><td>4</td><td></td><td></td><td></td></tr>
<tr><td>Arrout[4]</td><td>REAL</td><td>4</td><td></td><td></td><td></td></tr>
<tr><td>Arrout[5]</td><td>REAL</td><td>4</td><td></td><td></td><td></td></tr>
<tr><td>Arrout[6]</td><td>REAL</td><td>4</td><td></td><td></td><td></td></tr>
<tr><td>Arrout[7]</td><td>REAL</td><td>0</td><td></td><td></td><td></td></tr>
<tr><td>Arrout[8]</td><td>REAL</td><td>0</td><td></td><td></td><td></td></tr>
<tr><td>Arrout[9]</td><td>REAL</td><td>0</td><td></td><td></td><td></td></tr>
<tr><td>Arrout[10]</td><td>REAL</td><td>0</td><td></td><td></td><td></td></tr>
</table>
</td></tr>
</table>

◆ Key points

- Set the data types of "In" and AryOut[] to be identical. If they differ, an exception will occur during linking.
- When "In" and AryOut[] are of STRING type, ensure their area sizes are consistent.
- When the value of "Size" is 0, "Out" is TRUE, and AryOut[] remains unchanged.
- The return value "Out" is not used when employing this instruction in an ST program.
- An exception occurs and ENO becomes FALSE, and AryOut[] remains unchanged under the following conditions:
- When the value of "Size" exceeds the array bounds of AryOut[].

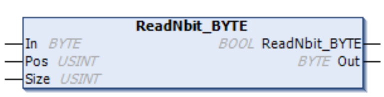
5.7.3 ReadNbit_**** (Read N-bits within bit string)

ReadNbit_BYTE: Reads N bits within BYTE data.

ReadNbit_WORD: Reads N bits within WORD data.

ReadNbit_DWORD: Reads N bits within DWORD data.

ReadNbit_LWORD: Reads N bits within LWORD data.

<table>
<tr><th>Instruction</th><th>Name</th><th>FB/FUN</th><th>Graphical representation</th><th>ST representation</th></tr>
<tr><td>ReadNbit_***</td><td>Bit string read</td><td>FUN</td><td>
</td><td>ReadNbit_****(In:= , Pos:= , Size:= , Out=>)
The four asterisks (****) are a placeholder for one of the data types: BYTE, WORD, DWORD, or LWORD.</td></tr>
</table>

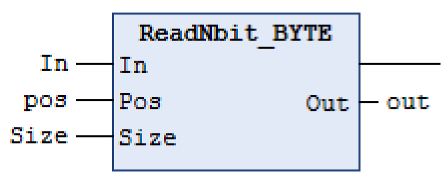
◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Source	Input	Source	Conforms to data type.	—	—
Pos	Start bit position		Start bit position			0
Size	Number of bits N		Number of bits N	Conforms to data type.	—	1
Out	Read result	Output	Read bit string result.	Conforms to data type.	—	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Pos						○														
Size						○														
Out		○	○	○	○															

◆ Function

It reads and copies "Size" bits of data starting from position "Pos" in the source "In" to "Out".

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">VAR
In:BYTE;
pos:USINT;
Size:USINT;
out:BYTE;
END_VAR</td></tr>
<tr><td>Program</td><td></td><td>ReadNbit_BYTE(
In:=In ,
Pos:=pos ,
Size:=Size ,
Out=>out);</td></tr>
<tr><td>Result</td><td colspan="2"><table><tr><th>表达式</th><th>类型</th><th>值</th></tr><tr><td>In</td><td>BYTE</td><td>2#10101111</td></tr><tr><td>pos</td><td>USINT</td><td>2#00000000</td></tr><tr><td>Size</td><td>USINT</td><td>2#00000100</td></tr><tr><td>out</td><td>BYTE</td><td>2#00001111</td></tr></table></td></tr>
</table>

◆ Key points

- Set the data types of "In" and "Out" to be identical. If they differ, an exception occurs during linking.
- When the value of "Size" is 0, "Out" is TRUE, and "Out" remains unchanged.

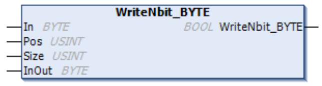
5.7.4 WriteNbit_**** (Write N-bits within bit string)

WriteNbit_BYTE: Writes N bits within BYTE data.

WriteNbit_WORD: Writes N bits within WORD data.

WriteNbit_DWORD: Writes N bits within DWORD data.

WriteNbit_LWORD: Writes N bits within LWORD data.

Instruction	Name	FB/FUN	Graphical representation	ST representation
WriteNbit_***	Bit string write	FUN		WriteNbit_****(In:= , Pos:= , Size:= , Out=>) The four asterisks (****) are a placeholder for one of the data types: BYTE, WORD, DWORD, or LWORD.

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Data source	Input	Data source to be copied.	Conforms to data type.	—	—
Pos	Write start position		Start position for writing.			0
Size	Number of bits		Number of bits to copy.	Conforms to data type.	—	1
InOut	Data to write	Output	Data to be written.	Conforms to data type.	—	—

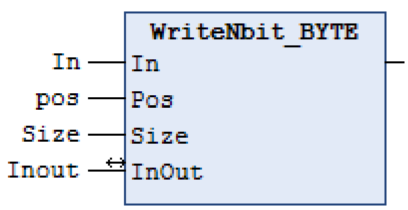
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Pos						○														
Size						○														
Out		○	○	○	○															

◆ Function

It copies "Size" bits of data starting from bit 0 of the data source "In" to bits "Pos" through "Pos+Size" of "InOut".

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">

```
VAR
    In:BYTE;
    pos:USINT;
    Size:USINT;
    Inout:BYTE;
END_VAR
```

</td></tr>
<tr><td>Program</td><td> </td><td>

```
WriteNbit_BYTE(
    In:=In ,
    Pos:=pos ,
    Size:=Size ,
    InOut:=Inout );
```

</td></tr>
<tr><td>Result</td><td colspan="2">

表达式	类型	值	准
In	BYTE	2#10101111	
pos	USINT	2#00000000	
Size	USINT	2#00000110	
Inout	BYTE	2#00101111	

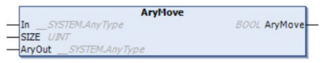
</td></tr>
</table>

◆ Key points

- Set the data types of "In" and "InOut" to be identical. If they differ, an exception occurs during linking.
- When the value of "Size" is 0, "Out" is TRUE, and "Out" remains unchanged.
- Data size is not zero. Position does not exceed data length. The sum of position and number of bits to write does not exceed data length.

5.7.5 AryMove (Array move)

The instruction transfers multiple array elements. The source and destination data types can differ.

Instruction	Name	FB/FUN	Graphical representation	ST representation
AryMove	Array transfer	FUN		AryMove(In, AryOut, Size);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In[ ] array	Source array	Input	Source array	Conforms to data type.	—	(*)
Size	Number of elements		Number of elements to transfer.			1
AryOut[ ] array	Destination array		Destination array	Conforms to data type.	—	—
Out	Return value	Output	Always TRUE.	TRUE only.	—	—

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

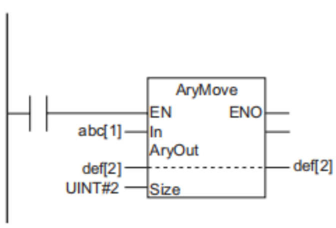
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Enumerated types, entire structures, or a single structure element can also be specified.																			
Size							○													
AryOut	An array whose elements are of the same data type as "In".																			
Out	○																			

◆ Function

It transfers the first "Size" elements starting from In[0] of the source array In[] to the elements starting from AryOut[0] of the destination array AryOut[]. The data types of In[] and AryOut[] may differ.

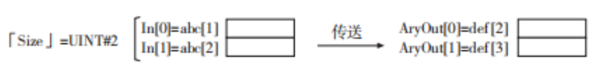
An example with "Size"=UINT#2 is shown below.

LD

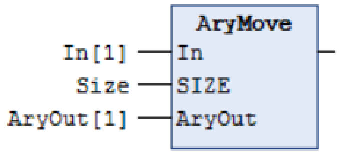
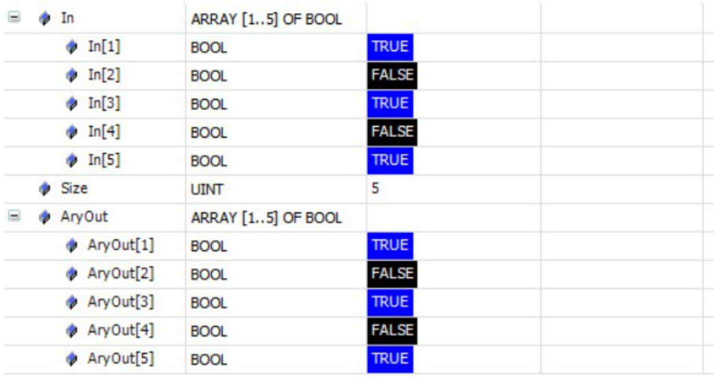


ST

AryMove(abc[1], def[2], UINT#2);



	FBD	ST
Variable declaration	VAR In :ARRAY[1..5] OF BOOL ; Size :UINT; AryOut :ARRAY [1..5] OF BOOL; END_VAR	

Program		`AryMove(In:=In[1] , SIZE:=Size , AryOut:=AryOut[1] );`
Result		

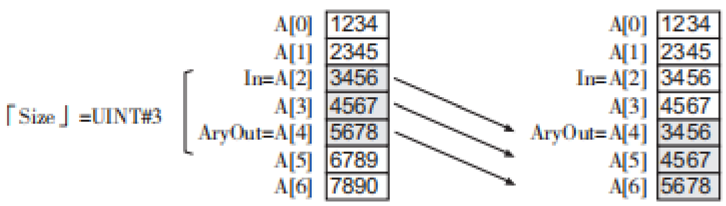
In	ARRAY [1..5] OF BOOL	
In[1]	BOOL	TRUE
In[2]	BOOL	FALSE
In[3]	BOOL	TRUE
In[4]	BOOL	FALSE
In[5]	BOOL	TRUE
Size	UINT	5
AryOut	ARRAY [1..5] OF BOOL	
AryOut[1]	BOOL	TRUE
AryOut[2]	BOOL	FALSE
AryOut[3]	BOOL	TRUE
AryOut[4]	BOOL	FALSE
AryOut[5]	BOOL	TRUE

◆ Reference

・When the data types of In[] and AryOut[] are identical, the MemCopy instruction can be used for high-speed processing.

・The same array can be specified for both In[] and AryOut[]. In this case, overlap between source and destination elements is allowed.

An example with In[0]=A[2], AryOut[0]=A[4], "Size"=UINT#3 is shown below.



◆ Key points

・When the data types of In[] and AryOut[] differ, ensure the valid range of AryOut[] encompasses the valid range of In[], regardless of which of the following data type groups they belong to: BYTE, WORD, DWORD, LWORD, USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, LREAL.

・When In[] is an array of structures, set the data types of In[] and AryOut[] to be identical.

・When the value of "Size" is 0, "Out" is TRUE, and AryOut[] remains unchanged.

・The return value "Out" is not used when employing this instruction in an ST program.

・An exception occurs and ENO becomes FALSE, and AryOut[] remains unchanged under the following conditions:

・When the value of "Size" exceeds the size of In[] or AryOut[].

・When In[] or AryOut[] are arrays of STRING type, and any element to be transferred is not null-terminated.

・When In[] or AryOut[] are arrays of STRING type, and the string length of a transfer element exceeds the size of the AryOut[] element.

5.7.6 Clear (Initialize)

The instruction initializes a variable.

Instruction	Name	FB/FUN	Graphical representation	ST representation
Clear	Initialize	FUN		Out:=Clear(InOut);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
InOut	Object to initialize	Input/Output	Object to initialize	Conforms to data type.	—	—
Out	Return value	Output	Always TRUE.	TRUE only.		

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Enumerated types, entire structures, or a single structure element can also be specified.																			
Out	○																			

◆ Function

It initializes the value of the object "InOut".

If an initial value attribute is set for the variable, it is initialized to that value. If not set, it is initialized to the default initial value for its data type.

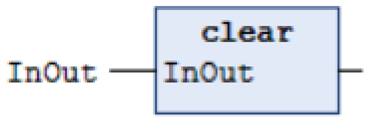
The default initial value for each data type is shown in the table below.

Data type	Default value
BOOL	FALSE
BYTE, WORD, DWORD, LWORD	16#0
USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, LREAL	0
TIME	T#0ms
DATE	D#1970-1-1
TOD	TOD#0:0:0
DT	DT#1970-1-1-0:0:0
STRING	-

When "InOut" is an entire array, a single array element, an entire structure, or a single structure element, it is processed as shown in the table below.

“InOut”	Processing
Entire array	Initializes all elements of the array.
A single array element	Initializes only that element.
Entire structure	Initializes all members of the structure.
A single structure element	Initializes only that structure element.

An example is shown below. Initializes the value of variable abc. For example, if abc=INT#10, abc becomes INT#0.

	FBD	ST
Variable declaration	VAR InOut :INT:=10; END_VAR	
Program		Clear(InOut:=InOut);
Result	InOut INT 0	

◆ Reference

・When "InOut" is an array used as a stack, also clear the variable that manages the stack count when executing this instruction.

・When initializing cam data variables with this instruction, the values are set to zero, not to the values saved via the MC_SaveCamTable instruction.

◆ Key points

・The return value "Out" is not used when employing this instruction in an ST program.

・When initializing variables of an enumerated type, set the initial value attribute. If not set, the variable value becomes zero.

5.8 Shift instructions

5.8.1 AryShiftReg (Shift register)

The instruction shifts the entire bit string composed of array elements left by 1 bit and inserts an input value into the least significant bit.

Instruction	Name	FB/FUN	Graphical representation	ST representation
AryShiftReg	Shift register	FB		AryShiftReg_instance(Shift, Reset, In, InOut, Size, P_CY=>);

◆ Variables

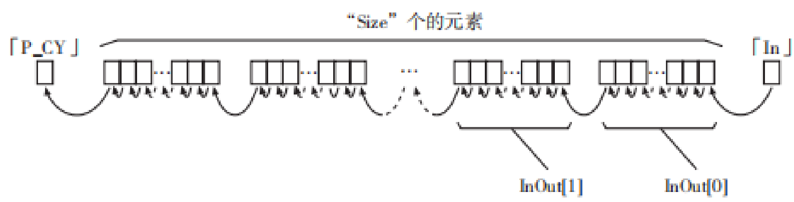
	Name	Input/Output	Description	Valid range	Unit	Initial value
Shift	Shift	Input	FALSE → TRUE triggers the shift.	Conforms to data type.	—	FALSE
Reset	Reset		TRUE: Performs a reset.			
In	Input value		Value inserted into the LSB of InOut[].			
Size	Number of array elements		Number of elements used for the shift register within InOut[].		—	1
InOut[ ] array	Bit string array		Bit string array.	Conforms to data type.		—
P_CY	Carry flag	Output	Value held in the carry flag status.	Conforms to data type.	—	FALSE

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Shift	○																			
Reset	○																			
In	○																			
Size							○													
InOut[ ] array	○	○	○	○	○															
P_CY	○																			

◆ Function

When "Shift" changes from FALSE to TRUE, the first "Size" array elements starting from InOut[0] of the bit string array InOut[] are shifted left (towards higher significance) by 1 bit.

The input value "In" is inserted into the least significant bit. The bit shifted out from the most significant bit of the array is output to the carry (CY) flag "P_CY".

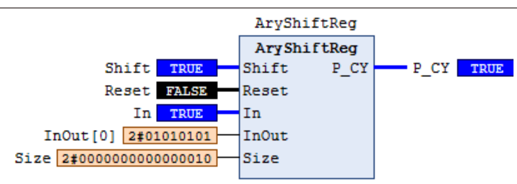


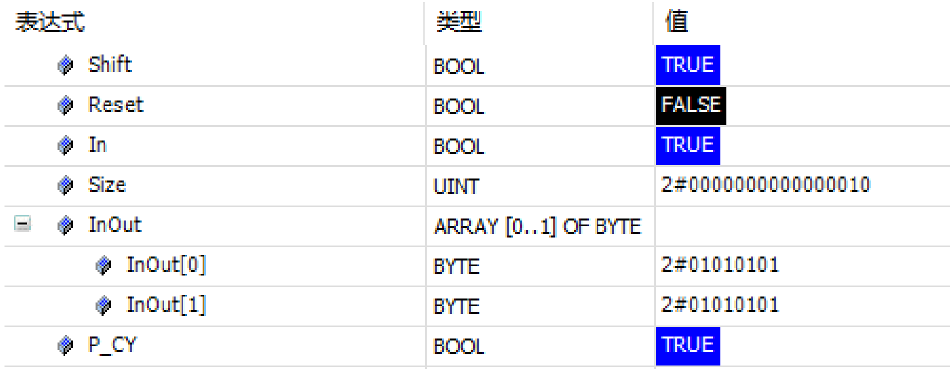
When "Reset" is TRUE, FALSE is set to all bits of the first "Size" elements starting from InOut[0] and to the carry (CY) flag.

An example is shown below for a BYTE-type array InOut[], with InOut[0] = 2#10101010, InOut[1] = 2#10101010, "Size"=UINT#2, and "In" = TRUE, when a single Shift is triggered.

InOut[0] = 2#10101010, InOut[1] = 2#10101010 , "P_CY" = FALSE.

→ InOut[0] = 2#01010101, InOut[1] = 2#01010101,"P_CY"=TRUE.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR Shift :BOOL; Reset :BOOL; In :BOOL; Size :UINT; InOut :ARRAY [0..1] OF BYTE; P_CY :BOOL; AryShiftReg :HCFA_OmronUitls.AryShiftReg; END_VAR	
Program		AryShiftReg(Shift TRUE := Shift TRUE , Reset FALSE := Reset FALSE , In TRUE := In TRUE , InOut:= InOut[0] 2#01010101 , Size 2#0000000000000010 := Size 2#0000000000000010 P_CY TRUE => P_CY TRUE);

Result	

表达式	类型	值
Shift	BOOL	TRUE
Reset	BOOL	FALSE
In	BOOL	TRUE
Size	UINT	2#0000000000000010
InOut	ARRAY [0..1] OF BYTE	
InOut[0]	BYTE	2#01010101
InOut[1]	BYTE	2#01010101
P_CY	BOOL	TRUE

Related system-defined variables

Variable name	Name	Data type	Description
P_CY	Carry (CY) flag	BOOL	Value held in the carry flag status.

◆ Key points

- When "Reset" is TRUE, the shift operation is not executed even if "Shift" changes from FALSE to TRUE.
- When the value of "Size" is 0, InOut[] remains unchanged.
- An exception occurs and ENO becomes FALSE, and InOut[] remains unchanged under the following condition:
- When the value of "Size" exceeds the array bounds of InOut[].

5.8.2 AryShiftRegLR (Left/Right shift register)

It shifts the entire bit string composed of array elements left or right by 1 bit and inserts an input value.

Instruction	Name	FB/FUN	Graphical representation	ST representation
AryShiftRegLR	Left/Right shift register	FB		AryShiftReg_instance(ShiftL, ShiftR, Reset, In, InOut, Size, P_CY=>);

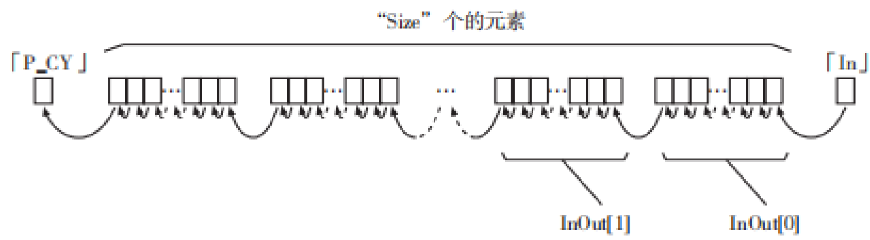
◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
ShiftL	Left shift	Input	FALSE → TRUE triggers a left shift.	Conforms to data type.	—	FALSE
ShiftR	Right shift		FALSE → TRUE triggers a right shift.			
Reset	Reset		TRUE: Performs a reset.			
In	Input value		Value inserted into the LSB of InOut[].			
Size	Number of array elements		Number of elements used for the shift register within InOut[].		—	1
InOut[ ] array	Bit string array		Bit string array.	Conforms to data type.		—
P_CY	Carry flag	Output	Value held in the carry flag status.	Conforms to data type.	—	FALSE

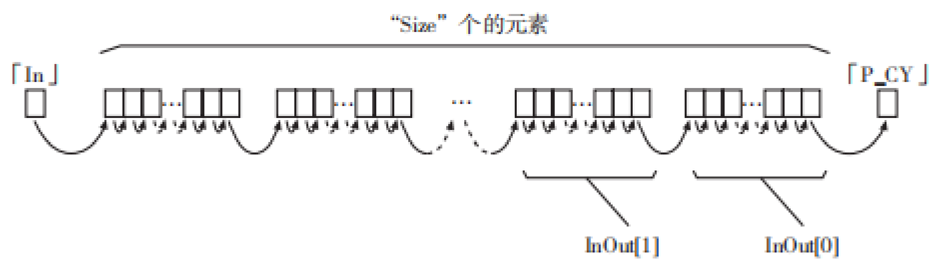
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ShiftL	○																			
ShiftR	○																			
Reset	○																			
In	○																			
Size							○													
InOut[ ] array	○	○	○	○	○															
P_CY	○																			

◆ Function

When "ShiftL" changes from FALSE to TRUE, the first "Size" array elements starting from InOut[0] of the bit string array InOut[] are shifted left by 1 bit. The input value "In" is inserted into the least significant bit. The bit shifted out from the most significant bit of the array is output to the carry (CY) flag "P_CY".



When "ShiftR" changes from FALSE to TRUE, the elements are shifted right by 1 bit, and "In" is inserted into the most significant bit. The bit shifted out from the least significant bit of the array is output to the carry (CY) flag "P_CY".



When "Reset" is TRUE, FALSE is set to all bits of the first "Size" elements starting from InOut[0] and to "P_CY".

An example is shown below for a BYTE-type array InOut[], with InOut[0] = 2#10101010, InOut[1] = 2#10101010, "Size"=UINT#2, and "In" = TRUE, when "ShiftL" changes from FALSE to TRUE.

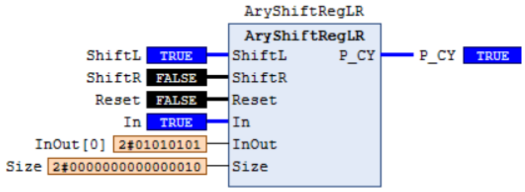
InOut[0] = 2#10101010, InOut[1] = 2#10101010 , "P_CY" = FALSE.

→ InOut[0] = 2#01010101, InOut[1] = 2#01010101,"P_CY"=TRUE.

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">

```
PROGRAM PLC_PRG
VAR
    ShiftL    :BOOL;
    ShiftR    :BOOL;
    Reset     :BOOL;
    In        :BOOL;
    Size      :UINT;
    InOut     :ARRAY [0..1] OF BYTE;
    P_CY      :BOOL;
    AryShiftRegLR    :HCFA_OmronUitls.AryShiftRegLR;
END_VAR
```

</td></tr>
<tr><td>Program</td><td>



</td><td>

```
AryShiftRegLR(
    ShiftL TRUE := ShiftL TRUE,
    ShiftR FALSE := ShiftR FALSE,
    Reset FALSE := Reset FALSE,
    In TRUE := In TRUE,
    InOut:= InOut[0] 2#01010101,
    Size 2#0000000000000010 := Size 2#0000000000000010,
    P_CY TRUE => P_CY TRUE);
```

</td></tr>
<tr><td>Result</td><td colspan="2">

表达式	类型	值
ShiftL	BOOL	TRUE
ShiftR	BOOL	FALSE
Reset	BOOL	FALSE
In	BOOL	TRUE
Size	UINT	2#0000000000000010
InOut	ARRAY [0..1] OF BYTE	
InOut[0]	BYTE	2#01010101
InOut[1]	BYTE	2#01010101
P_CY	BOOL	TRUE

</td></tr>
</table>

Related system-defined variables

Variable name	Name	Data type	Description
P_CY	Carry (CY) flag	BOOL	Value held in the carry flag status.

◆ Key points

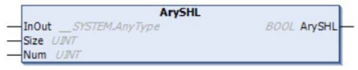
- When "Reset" is TRUE, the shift operation is not executed even if "Shift" changes from FALSE to TRUE.
- When both "ShiftL" and "ShiftR" change from FALSE to TRUE simultaneously, no shift operation is performed.
- ENO becomes TRUE when the shift operation proceeds normally upon "Shift" changing from FALSE to TRUE, or when the reset operation proceeds normally upon "Reset" becoming TRUE.
- When the value of "Size" is 0, InOut[] remains unchanged.
- An exception occurs and ENO becomes FALSE, and InOut[] remains unchanged under the following condition:.
- When the value of "Size" exceeds the array bounds of InOut[].

5.8.3 ArySHL/ArySHR (Array shift left/right N elements)

The instruction shifts multiple array elements.

ArySHL: Shifts left (towards higher indices).

ArySHR: Shifts right (towards lower indices).

Instruction	Name	FB/FUN	Graphical representation	ST representation
ArySHL	Array shift left N elements	FUN		ArySHL(InOut, Size, Num);
ArySHR	Array shift right N elements	FUN		ArySHR(InOut, Size, Num);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Size	Number of elements to shift	Input	Number of elements to be shifted.	Conforms to data type.	—	1
Num	Number of shift positions		Number of positions to shift.			
InOut[ ] array	Array to shift		Array to be shifted.			—
Out	Return value	Output	Always TRUE.	TRUE only.		—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							○													
Num							○													
InOut[ ] array	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Arrays of structures can also be used.																			
Out	○																			

◆ Function

It shifts the first "Size" elements (at the higher indices) of the target array InOut[] by "Num" positions. Values shifted out of the array are discarded. The vacated positions are filled with the initial value of the InOut[] data type. If an initial value attribute is set for InOut[], that value is used. Otherwise, the default initial value for the data type is used. If InOut[] is an array of structures, all members of each structure element are initialized.

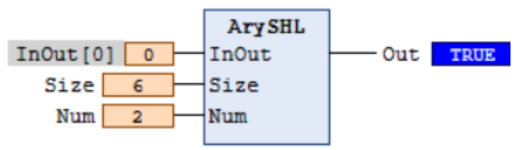
The default initial value for each data type is shown in the table below.

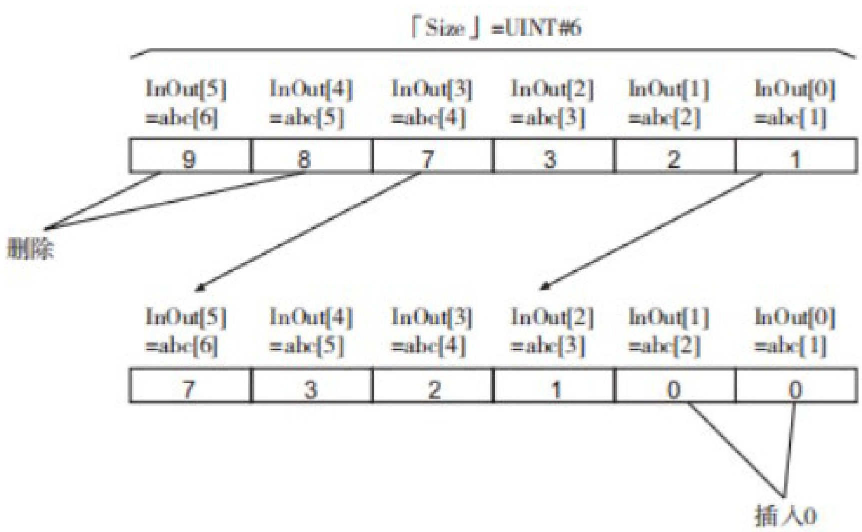
Data type	Default value
BOOL	FALSE
BYTE, WORD, DWORD, LWORD	16#0
USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, LREAL	0
TIME	T#0ms
DATE	D#1970-1-1
TOD	TOD#0:0:0
DT	DT#1970-1-1-0:0:0
STRING	“

ArySHL: Shifts left (towards the higher indices of the array)

ArySHR: Shifts right (towards the lower indices of the array).

An example for the ArySHL instruction with "Size"=UINT#6 and "Num"=UINT#2 is shown below.

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">PROGRAM PLC_PRG
VAR
Num :UINT;
Size :UINT;
InOut :ARRAY [0..5] OF BYTE;
Out :BOOL;
check :BOOL;
END_VAR</td></tr>
<tr><td>Program</td><td>
</td><td>IF check FALSE THEN
out TRUE :=ArySHL(
InOut:= InOut[0] 0 ,
Size:= Size 6 ,
Num:= Num 2);
check FALSE :=FALSE;
END_IF RETURN</td></tr>
<tr><td>Result</td><td colspan="2"><table>
<tr><th>表达式</th><th>类型</th><th>值</th></tr>
<tr><td>Num</td><td>UINT</td><td>2</td></tr>
<tr><td>Size</td><td>UINT</td><td>6</td></tr>
<tr><td>InOut</td><td>ARRAY [0..5] OF BYTE</td><td></td></tr>
<tr><td>InOut[0]</td><td>BYTE</td><td>0</td></tr>
<tr><td>InOut[1]</td><td>BYTE</td><td>0</td></tr>
<tr><td>InOut[2]</td><td>BYTE</td><td>1</td></tr>
<tr><td>InOut[3]</td><td>BYTE</td><td>2</td></tr>
<tr><td>InOut[4]</td><td>BYTE</td><td>3</td></tr>
<tr><td>InOut[5]</td><td>BYTE</td><td>7</td></tr>
<tr><td>Out</td><td>BOOL</td><td>TRUE</td></tr>
</table></td></tr>
</table>



◆ Reference

When InOut[] is of BOOL type, it is equivalent to shifting a "Size"-bit bit string by "Num" bits.

◆ **Key points**

- When the value of "Num" is 0, no shift operation is performed.
- When the value of "Num" is greater than "Size", all values from InOut[0] to InOut["Size"-1] are initialized.
- The return value "Out" is not used when employing this instruction in an ST program.
- An exception occurs and ENO becomes FALSE, and InOut[] remains unchanged under the following condition:
- When the value of "Size" exceeds the array bounds of InOut[].

5.9 Data conversion instructions

5.9.1 Swap (Byte swap)

The instruction swaps the high-order and low-order bytes of a 16-bit value.

Instruction	Name	FB/FUN	Graphical representation	ST representation
Swap	Byte swap	FB		Out:=Swap(In);

◆ **Variables**

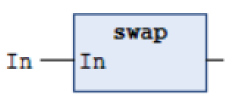
	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Conversion source	Input	Conversion source	Conforms to data type.	—	0
Out	Conversion result	Output	Conversion result		—	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Shift			○																	
Reset			○																	

◆ **Function**

It swaps the high-order and low-order bytes of the conversion source "In" and assigns the result to "Out".

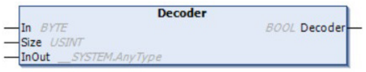
An example with "In"=WORD#16#1234 is shown below.

	FBD	ST
Variable declaration	`VAR In :WORD:=16#1234; Out :WORD; END_VAR`	
Program		Out:= SWAP(In:= In);
Result	In WORD 16#1234 / Out WORD 16#3412	

5.9.2 Decoder (Bit decoder)

The instruction sets a single bit specified by an array element (up to 256 bits) to TRUE, and all other bits to FALSE.

The instruction swaps the high-order and low-order bytes of a 16-bit value.

Instruction	Name	FB/FUN	Graphical representation	ST representation
Decoder	Bit decoder	FUN		Decoder(In, Size, InOut);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Bit position	Input	Bit position to convert.	Conforms to data type.	—	0
Size	Number of bits		Number of bits to convert.	0~8	Bit	1
InOut[ ] array	Target array		Target array.	Conforms to data type.	—	—
Out	Return value	Output	Always TRUE.	TRUE only.	—	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○																		
Size						○														
InOut[ ] array	○	○	○	○	○															
Out	○																			

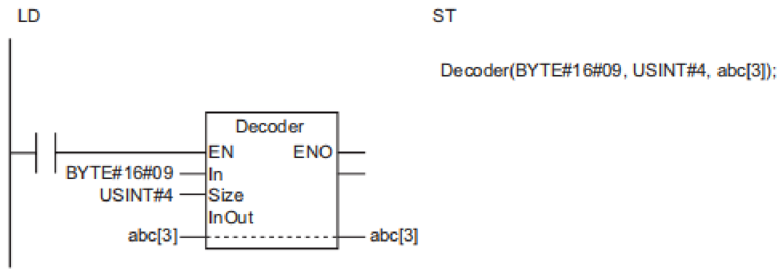
◆ Function

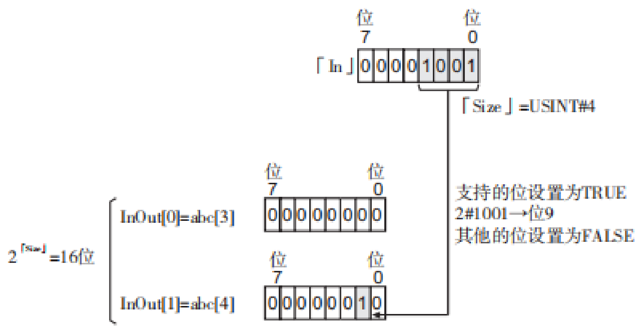
It sets a specific bit within the first 2^"Size" bits starting from InOut[0] of the target array InOut[] to TRUE. All other bits are set to FALSE. The bit position of the TRUE bit is specified by the lower "Size" bits of the conversion bit position "In".

When passing the input/output parameter to InOut[], always include an element index, e.g., array[3].

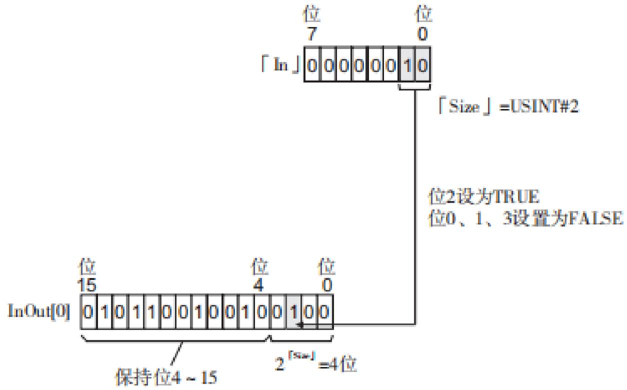
An example with "In"=BYTE#16#09, "Size"=USINT#4, and InOut[] as a BYTE-type array. The value of the lower "Size" bits of "In" is 16#9, which is decimal 9. Therefore, the 9th bit from the LSB of InOut[] is TRUE; all other bits are FALSE.

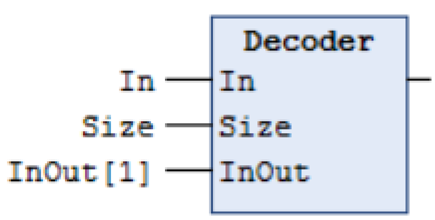
InOut[] is a BYTE array. The 9th bit from the LSB is bit 1 of InOut[1]. Thus, bit 1 of InOut[1] is TRUE; all bits of InOut[0] and bits other than bit 1 of InOut[1] are FALSE.





If the number of bits in an InOut[] element is greater than the number represented by "Size", the remaining bits retain their values. Example: "In"=BYTE#16#02, "Size"=USINT#2, InOut[] is a WORD-type array. "Size"=USINT#2, so the lower 4 bits of InOut[0] are set. Bits 4-15 of InOut[0] retain their values.



	FBD	ST
Variable declaration	VAR In :BYTE; Size :USINT; InOut :ARRAY[1..5] OF BYTE; END_VAR	
Program		Decoder(In:=In , Size:=Size , InOut:=InOut[1]);

Result	

In	BYTE	2#00001001
Size	USINT	2#00000110
InOut	ARRAY [1..5] OF BYTE	
InOut[1]	BYTE	2#00000000
InOut[2]	BYTE	2#00000010
InOut[3]	BYTE	2#00000000
InOut[4]	BYTE	2#00000000
InOut[5]	BYTE	2#00000000

◆ Reference

To calculate the bit position of a TRUE bit within array elements of up to 256 bits, use the "Encoder instruction (P.2-394)".

◆ Key points

- When the value of "Size" is 0, all bits of InOut[] are FALSE.
- The return value "Out" is not used when employing this instruction in an ST program.
- An exception occurs, ENO becomes FALSE, and InOut[] remains unchanged under the following conditions:
- When "Size" is outside its valid range.
- When 2^"Size" exceeds the number of bits in the InOut[] array elements.
- When InOut[] is not an array of BOOL or bit string data types.
- When an array without an index is passed to InOut[].

5.9.3 Encoder (Bit encoder)

The instruction calculates the bit position of a TRUE bit within array elements of up to 256 bits.

Instruction	Name	FB/FUN	Graphical representation	ST representation
Encoder	Bit encoder	FUN		Out:=Encoder(In, Size);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In[ ] array	Source array	Input	Source array	Conforms to data type.	—	(*)
Size	Number of bits to check		Number of bits to check.	0~8	Bit	1
Out	Check result	Output	Check result.	Conforms to data type.	—	—

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

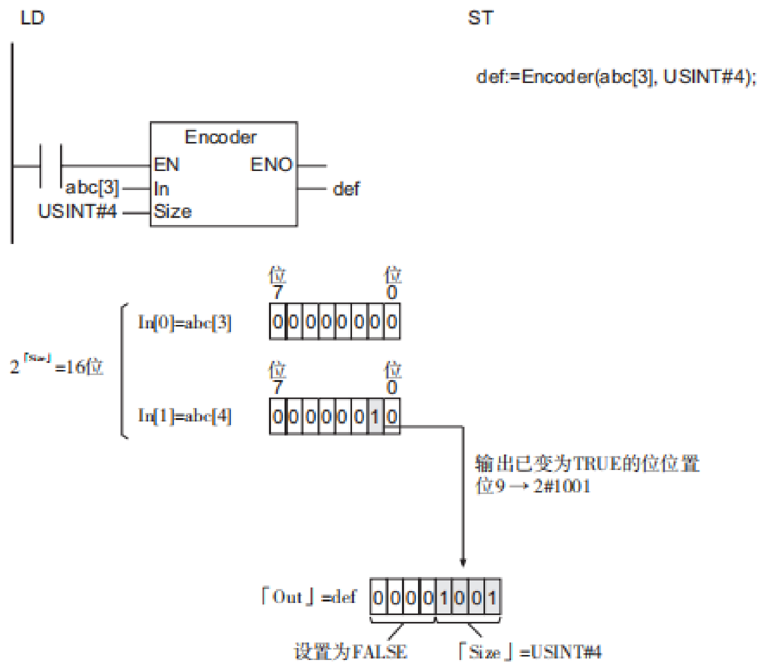
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ] array	○	○	○	○	○															
Size						○														
Out		○																		

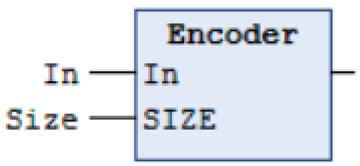
◆ Function

It calculates the bit position of a TRUE bit within a specified range of the source array In[]. The bit position is calculated within the range of 2^"Size" bits starting from In[0]. The bit position of the TRUE bit within this range, expressed in binary, is stored in the lower "Size" bits of the conversion result "Out". The remaining bits of "Out" are set to FALSE.

If multiple bits are TRUE within the specified range, the bit position of the highest-order TRUE bit is calculated. When passing the input parameter to In[], always include an element index, e.g., array[3]. Example for "Size"=USINT#4, In[] as a BYTE-type array.

"Size"=USINT#4, so the range for calculation is the 2^4=16 bits starting from In[0]. In the figure below, the 9th bit within this range is TRUE. Since "Size"=USINT#4, the calculated value 9 (2#1001) is stored in the lower 4 bits of "Out". The upper 4 bits of "Out" are set to FALSE.



	FBD	ST
Variable declaration	VAR In :ARRAY [1..5] OF BYTE; Size :USINT; Out :BYTE; END_VAR	
Program		Out:=Encoder(In:=In[1] , SIZE:=Size);
Result	In — ARRAY [1..5] OF BYTE In[1] — BYTE — 2#00000000 In[2] — BYTE — 2#00000000 In[3] — BYTE — 2#00000101 In[4] — BYTE — 2#00000001 In[5] — BYTE — 2#00000000 Size — USINT — 2#00000101 Out — BYTE — 2#00011000	

◆ Reference

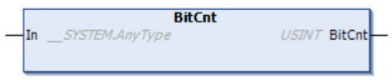
To set a single bit to TRUE and all others to FALSE within array elements of up to 256 bits, use the "Decoder instruction".

◆ **Key points**

- When the value of "Size" is 0, all bits of "Out" are FALSE.
- An exception occurs, ENO becomes FALSE, and "Out" remains unchanged under the following conditions:
- When "Size" is outside its valid range.
- When 2^"Size" exceeds the number of bits in the In[] array elements.
- When all bits specified by "Size" within In[] are FALSE.
- When In[] is not an array of BOOL or bit string data types.
- When an array without an index is passed to In[].

5.9.4 BitCnt (Bit counter)

The instruction counts the total number of bits set to TRUE within a bit string.

Instruction	Name	FB/FUN	Graphical representation	ST representation
BitCnt	Bit count	FUN		Out:=BitCnt(In);

◆ **Variables**

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Count target	Input	Target for counting TRUE bits.	Conforms to data type.	—	(*)
Out	Count result	Output	Number of TRUE bits.	0 to number of bits in "In".	—	—

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

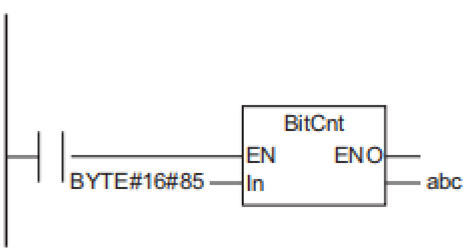
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Out						○														

◆ **Function**

It counts the total number of bits set to TRUE in the count target "In".

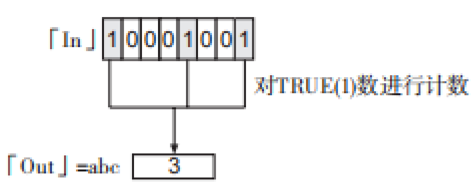
An example with "In" as BYTE type, value BYTE#16#85, is shown below.

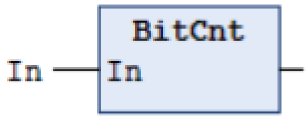
LD



ST

abc:=BitCnt(BYTE#16#85);



	FBD	ST
Variable declaration	VAR In :BYTE; Out :USINT; END_VAR	
Program		Out:=BitCnt(In:=In);
Result	In BYTE 255 Out USINT 8	

5.9.5 LineToColm (Line-to-column bit conversion)

The instruction decomposes a bit string and outputs to bit positions specified by array elements.

Instruction	Name	FB/FUN	Graphical representation	ST representation
LineToColm	Line to bit string conversion	FUN		LineToColm(In, InOut, Size, Pos);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Source	Input	Source.	Conforms to data type.	—	(*)
Size	Number of result elements		Number of elements in the result.	0 to number of bits in "In".		1
Pos	Target bit position		Target bit position for conversion.	0 to number of bits in InOut[] – 1.		0
InOut[ ] array	Result array		Result.	Conforms to data type.	—	—
Out	Return value	Output	Always TRUE.	TRUE only.	—	—

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Size						○														
Pos						○														
InOut[ ] array		○	○	○	○															
Out	○																			

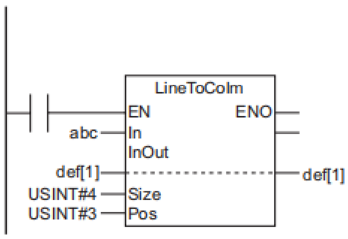
◆ Function

It decomposes a bit string and outputs to bit positions specified by array elements.

First, it extracts "Size" bits from the LSB of the source "In" and decomposes them into individual bits. Then, it stores each bit into the "Pos"-th bit of successive elements starting from InOut[0] in the result array InOut[]. This writes to "Size" array elements. Bits that are not written retain their previous values.

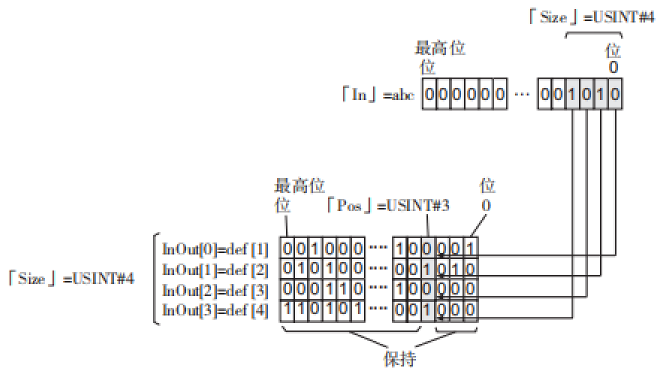
An example with "Pos"=USINT#3 and "Size"=USINT#4 is shown below.

LD



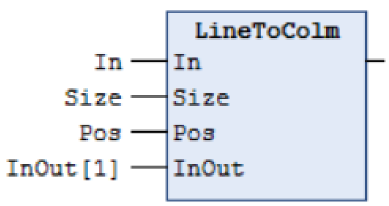
ST

```
LineToColm(abc, def[1], USINT#4, USINT#3);
```



	FBD	ST

Variable declaration	`VAR`
	`In :BYTE;`
	`Size :USINT;`
	`Pos :USINT;`
	`InOut :ARRAY[1..5] OF BYTE;`
	`END_VAR`	

Program		`LineToColm(In:=In ,`
		`Size:=Size ,`
		`Pos:=Pos ,`
		`InOut:=InOut[1] );`

Result		

Name	Type	Value
In	BYTE	2#00011
Size	USINT	2#00000
Pos	USINT	2#00000
InOut	ARRAY [1..5] OF BYTE	
InOut[1]	BYTE	2#00100
InOut[2]	BYTE	2#00100
InOut[3]	BYTE	2#00100
InOut[4]	BYTE	2#00100
InOut[5]	BYTE	2#00100

◆ Key points

- When the value of "Size" is 0, all bits of "Out" are FALSE.
- An exception occurs, ENO becomes FALSE, and "Out" remains unchanged under the following conditions:
- When "Size" is outside its valid range.
- When the value of "Pos" is outside its valid range.
- When the value of "Size" exceeds the array bounds of InOut[].
- When InOut[] is not an array of bit string data types.

- When an array without an index is passed to InOut[].

5.9.6 Gray (Gray code conversion)

The instruction converts Gray code to an angle.

Instruction	Name	FB/FUN	Graphical representation	ST representation
Gray	Gray code conversion	FUN		LineToColm(In, InOut, Size, Pos);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Conversion source	Input	Gray code to convert.	Conforms to data type.	—	0
Resolution	Resolution		Resolution.	Enumeration type _eGRY_RESOLUTION		_R256
ERC	Encoder remainder compensation		Encoder remainder compensation value.	0 to the value of "Resolution".		0
ZPC	Zero point compensation		Zero point compensation value.			
Out	Conversion result	Output	Conversion result.	(*)	°	—

* 0 ~ 3.59999999999999e+2.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In			○																	
Resolution	Enumeration _eGRY_RESOLUTION																			
ERC							○													
ZPC							○													
Out															○					

◆ Function

It converts the Gray code output value "In" from a rotary encoder to an angular value. The unit of the result "Out" is degrees.

"Resolution" is of the enumeration type _eGRY_RESOLUTION. The meanings of the enumeration elements are as follows.

Enumeration element	Meaning
_R256	256
_R1B	1-bit (2)
_R2B	2-bit (4)
_R3B	3-bit (8)
_R4B	4-bit (16)
_R5B	5-bit (32)
_R6B	6-bit (64)
_R7B	7-bit (128)
_R8B	8-bit (256)
_R9B	9-bit (512)
_R10B	10-bit (1024)

_R11B	10-bit (2048)
_R12B	12-bit (4096)
_R13B	13-bit (28192)
_R14B	14-bit (16384)
_R15B	15-bit (32768)
_R360	360
_R720	720
_R1024	1024

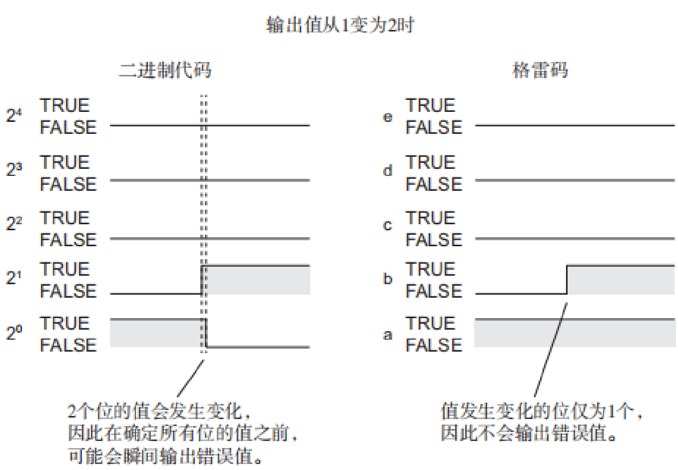
◆ Gray code

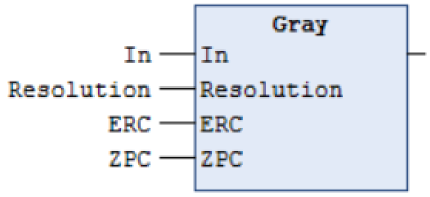
Gray code is a binary numeral system also known as reflected binary code. Its key feature is that codes for successive numbers differ by only one bit (e.g., 0 and 1, 1 and 2). Gray code is used in applications like absolute encoder outputs.

A 4-bit binary code and its Gray code equivalent are listed below.

Decimal	Binary code				Gray code			
	2^3	2^2	2^1	2^0	d	c	b	a
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

Using Gray code ensures that when the encoder output increases or decreases by 1, only one bit changes, preventing momentary erroneous outputs. The difference in encoder output changes between Gray code and binary code is shown below.

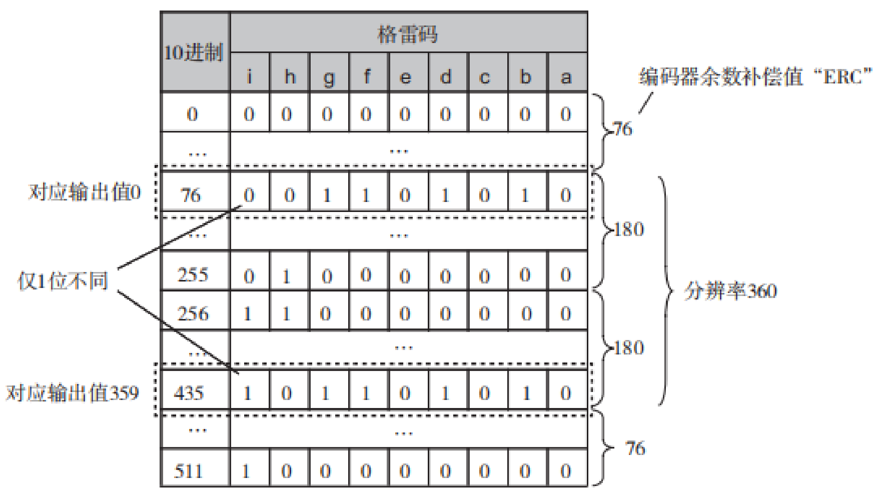


	FBD	ST
Variable declaration	VAR In :WORD; Resolution :_eGRY_RESOLUTION; ERC :UINT; ZPC :UINT; Out :LREAL; END_VAR	
Program		Out:=Gray(In:=In , Resolution:=Resolution ERC:=ERC, ZPC:=ZPC);
Result	In WORD 16#01A9 Resolution _EGRY_RESOLUTION _R10B ERC UINT 16#0000 ZPC UINT 16#0151 Out LREAL 348.75	

Encoder Remainder Compensation (ERC)

"ERC" is a compensation value that defines the Gray code range, used when the encoder resolution is not a power of two. It ensures that the Gray codes for the maximum and minimum encoder outputs differ by only one bit.

Example: An absolute encoder with a resolution of 360 uses 9-bit Gray code. The 9-bit range is 0 to 511. The Gray code is used for a range of ±180 around the center of 0-511, i.e., from 76 to 435. Thus, the Gray code for output 0 is 001101010 (decimal 76), and for output 359 is 101101010 (decimal 435), differing by only one bit. Here, the "ERC" value is 76.



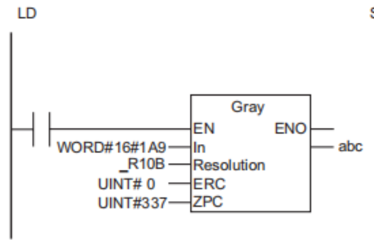
Zero Point Compensation (ZPC)

"ZPC" is set when the zero point (origin) angle of the rotary encoder is shifted. For example, to shift the origin of a rotary encoder with a resolution of 256 by 90°, set "ZPC" to 256×(90/360)=64.

Description example

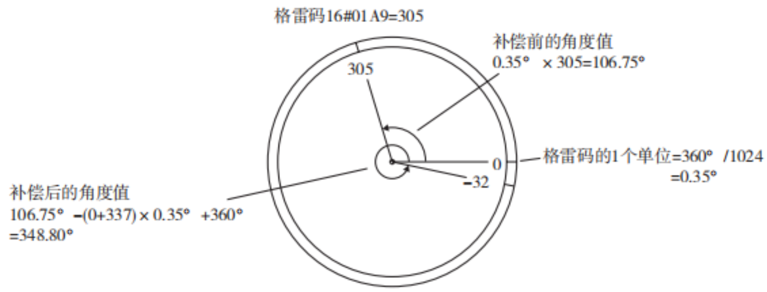
Example with "In"=WORD#16#1A9, "Resolution"=_R10B, "ERC"=UINT#0, "ZPC"=UINT#337. First, resolution is 10 bits, so one unit of Gray code equals 360°/1024=0.35°. Gray code 16#01A9 corresponds to decimal 305. Thus, the uncompensated angle is 0.35°×305=106.75°. "ERC"=0, "ZPC"=337. The compensated angle is 106.75°-(0+337)×0.35°=-11.20°. Since "Out" must be >0°, the result is -11.20°+360°=348.80°. "Out" becomes LREAL#348.8.

LD



ST

```
abc:=Gray(WORD#16#1A9, _R10B, UINT#0, UINT#337);
```

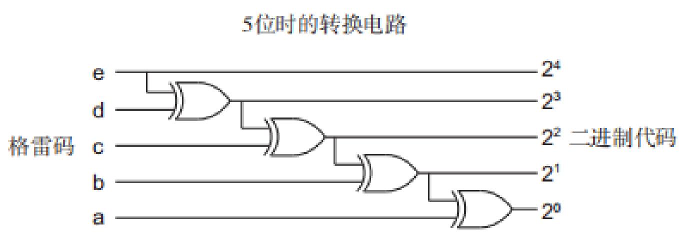


Reference

Refer to the manual of the rotary encoder being used to determine the appropriate values for "Resolution" and "ERC".

Conversion from gray code to binary code

Conversion from Gray code to binary code can be performed as follows. The logic symbol in the diagram represents XOR.



◆ Key points

- An exception occurs, "ENO" becomes FALSE, and "Out" remains unchanged under the following conditions:
- When the value of "Resolution" is outside its valid range.
- When the value of "ERC" is greater than the resolution specified by "Resolution".
- When the value of "ZPC" is greater than the resolution specified by "Resolution".
- When the value of "In" converted to a bit string is less than the value of "ERC".
- When the value of the bit string after compensation by "ERC" exceeds the resolution specified by "Resolution".

5.9.7 PWLLineChk (Polyline data check)

The instruction determines if the polyline data used by the polyline approximation conversion (no polyline data check) instruction is sorted in ascending order of X-coordinates.

Instruction	Name	FB/FUN	Graphical representation	ST representation
PWLLineChk	Polyline data check	FUN		Out:=PWLLineChk(Line,Num);

◆ Variables

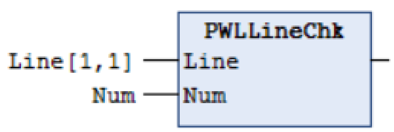
	Name	Input/Output	Description	Valid range	Unit	Initial value
Line[ ] array	Polyline data array	Input	Polyline data array	Conforms to data type.	—	(*)
Num	Number of polyline data points		Number of polyline data points			1
Out	Judgment result	Output	Judgment result	Conforms to data type.	—	—

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○	○	○					
Size							○													
Out	○																			

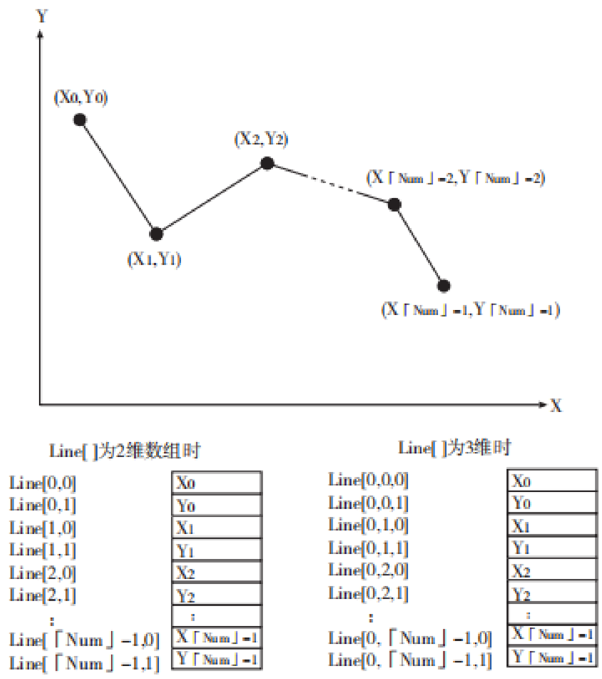
◆ Function

It determines if the elements of the polyline data array Line[] used by the PWLApproxNoLineChk instruction are sorted in ascending order of their X-coordinates. If they are in ascending order, the judgment result "Out" is TRUE; otherwise, "Out" is FALSE.

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">VAR
Line :ARRAY[1..5,1..2] OF REAL;
Num :UINT:=5;
Out :BOOL;
END_VAR</td></tr>
<tr><td>Program</td><td>
</td><td>Out:=PWLLineChk(Line:=Line[1,1] ,
Num:=Num);</td></tr>
<tr><td>Result</td><td colspan="2">Line | ARRAY [1..2, 1..5] ...
Line[1, 1] | REAL | 1
Line[1, 2] | REAL | 1.1
Line[1, 3] | REAL | 2
Line[1, 4] | REAL | -1.1
Line[1, 5] | REAL | 3
Line[2, 1] | REAL | 2
Line[2, 2] | REAL | 4
Line[2, 3] | REAL | 0
Line[2, 4] | REAL | 5
Line[2, 5] | REAL | -3
Num | UINT | 5
Out | BOOL | TRUE</td></tr>
</table>

Elements of polyline data Line[] and number of data points "Num"

Line[] is two-dimensional. Set the size of the first dimension to 2. As shown in the figure, store the coordinate values of each polyline point (X0, Y0), (X1, Y1)... as elements of Line[]. The number of polyline data points "Num" is half the number of elements in Line[] used for the polyline approximation calculation.



Description example

An example checking if a polyline data array abc[] with 4 data points is sorted in ascending X-order is shown below. "Num"=UINT#4, the element values of abc[] are as follows:

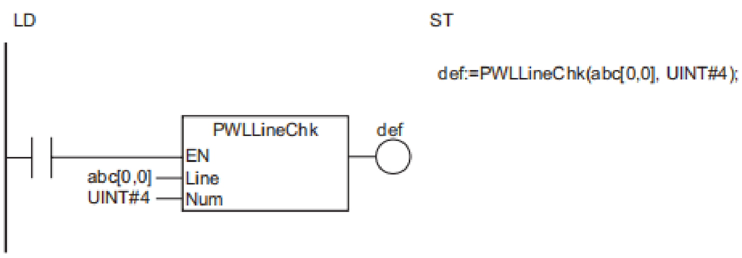
abc[0.0]=X0 =LREAL#1.0, abc[0,1]=Y0 =LREAL#5.0,

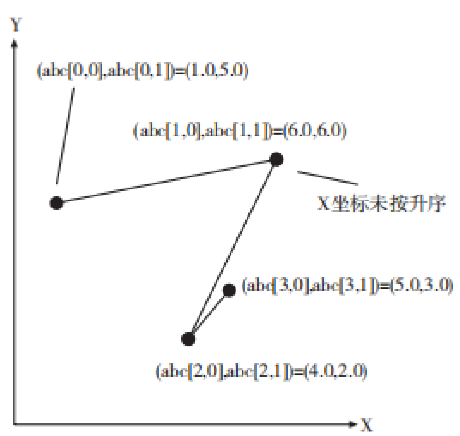
abc[1.0]=X1 =LREAL#6.0, abc[1,1]=Y1 =LREAL#6.0,

abc[2.0]=X2 =LREAL#4.0, abc[2,1]=Y2 =LREAL#2.0,

abc[3.0]=X3 =LREAL#5.0, abc[3,1]=Y3 =LREAL#3.0

The X-coordinates are not in ascending order, therefore "Out" is FALSE.





◆ Key points

- Line[] is two-dimensional. Set the size of the first dimension to 2.
- An exception occurs, and "Out" becomes FALSE under the following conditions:
- When the value of "Num" exceeds the array bounds of Line[].
- When Line[] is of REAL type and contains elements that are NaN, + ∞ , or − ∞ .
- When Line[] is not a supported data type.

5.9.8 MovingAverage (Moving average)

The instruction calculates the moving average.

Instruction	Name	FB/FUN	Graphical representation	ST representation
Moving Average	Moving average	FUN		MovingAverage(In, Buf, Out, BufSize, CurIndex, Q);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Input value	Input	Value used for average calculation.	Conforms to data type.	—	(*)
BufSize	Maximum number of elements	Input	Maximum number of elements used for average calculation.	Conforms to data type.	—	1
CurIndex	Input save position	Input/Output	Position in Buf[] where "In" is saved.	Conforms to data type.	—	—
Buf[ ] array	Input save array	Input	Array storing the "In" values.	Conforms to data type.	—	—
Q	Calculation complete flag	Input/Output	TRUE: Number of values saved to Buf[] is greater than "BufSize". FALSE: Number of values saved to Buf[] is less than "BufSize".	Conforms to data type.	—	—
Out	Calculation result	Input	Calculation result.	Conforms to data type.	—	—

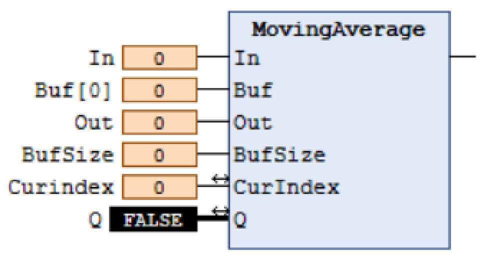
* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○	○	○					
BufSize							○													
CurIndex							○													
Buf[ ] array	An array with elements of the same data type as "In".																			
Q	○																			
Out						○	○	○	○	○	○	○	○	○	○					

◆ Function

Each time this instruction is executed, the input value "In" is saved into the input save array Buf[]. The average of the saved values is then stored in the calculation result "Out". The maximum number of elements used for the average calculation is specified by "BufSize".

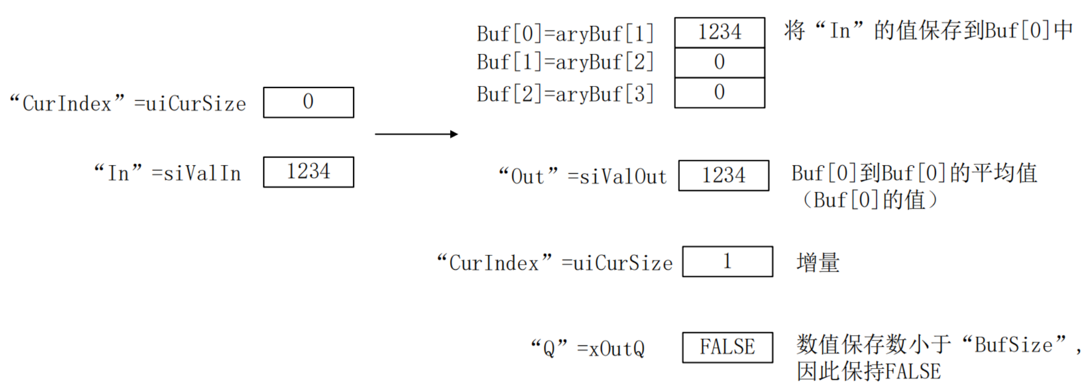
	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In :UINT; BufSize :UINT; Curindex:UINT; Buf :ARRAY [0..3] OF UINT; Q :BOOL; Out :UINT; check :BOOL; END VAR	

Program		IF check FALSE THEN
		MovingAverage(
		In:= In 0,
		Buf:= Buf[0] 0,
		Out:= Out 0,
		BufSize:= BufSize 0,
		CurIndex:= Curindex 0,
		Q:= Q FALSE);
		check FALSE :=FALSE;
		END_IF RETURN

Initial value input

Set the input save position "CurIndex" to 0, then execute this instruction. After clearing Buf[0] to Buf["BufSize"-1] of the input save array Buf[], the initial input value "In" is saved to Buf[0]. The calculation complete flag "Q" is FALSE, indicating the number of values saved to Buf[] has not yet reached "BufSize". When "Q" is FALSE, the average is calculated using values from Buf[0] to Buf["CurIndex"+1]. The result is saved to "Out". Then, the value of "CurIndex" is incremented.

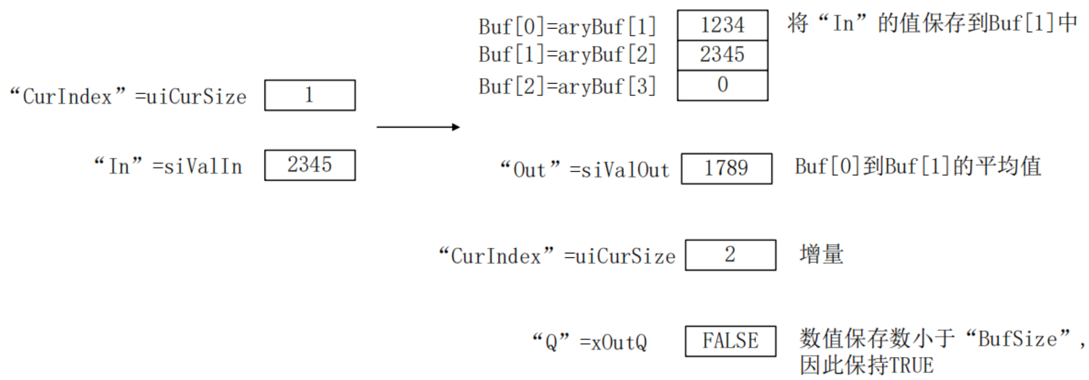
第1次执行指令



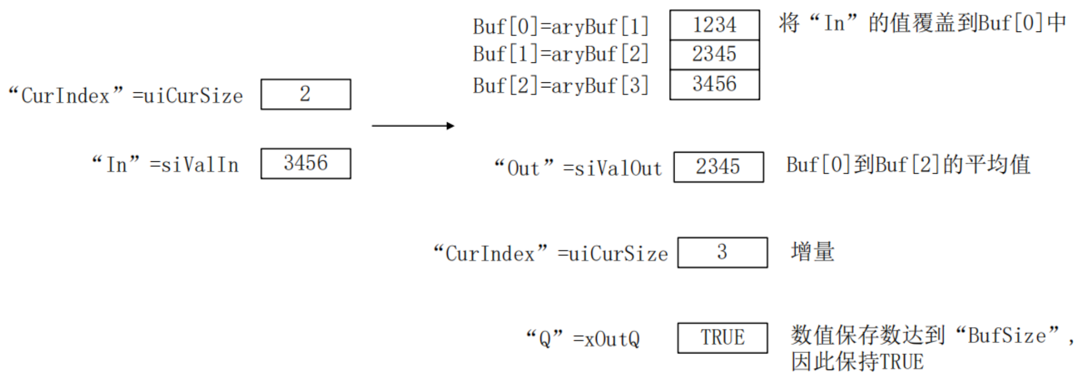
Input until the BufSize-th execution

On each execution, the value of "In" is saved to Buf["CurIndex"]. The average is calculated using the values from Buf[0] to Buf["CurIndex"] (a total of "CurIndex"+1 values), and the result is stored in "Out". Once the number of executions reaches "BufSize", the value of "Q" becomes TRUE.

第2次执行指令



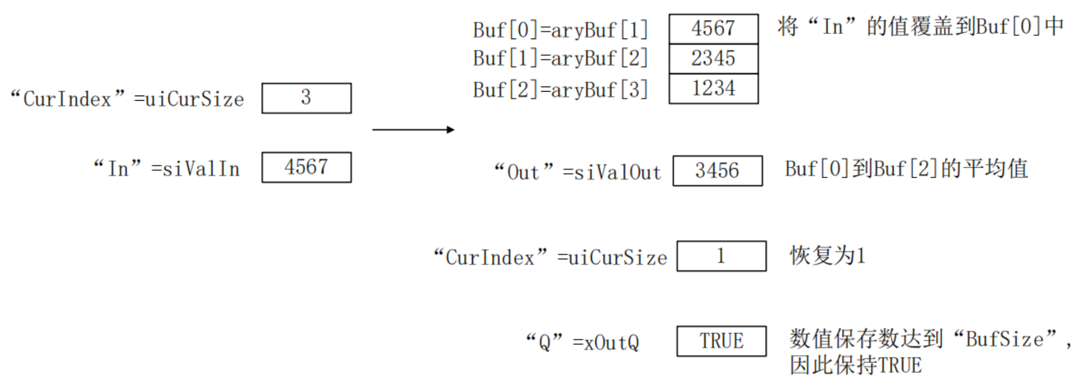
第3次执行指令

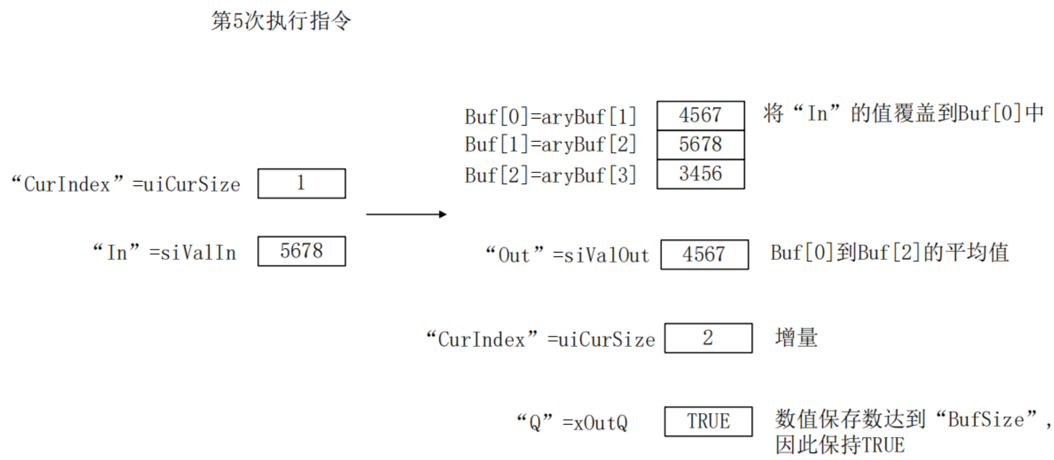


Input after the BufSize-th execution

On each execution, the value of "In" overwrites a value cyclically within Buf[0] to Buf["BufSize"-1]. The average is calculated using the current values in Buf[0] to Buf["BufSize"-1], and the result is stored in "Out". After "CurIndex" reaches "BufSize", it resets to 1 and then increments. "Q" remains TRUE.

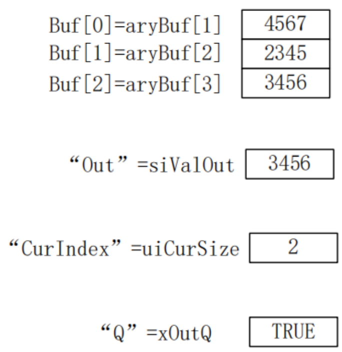
第4次执行指令

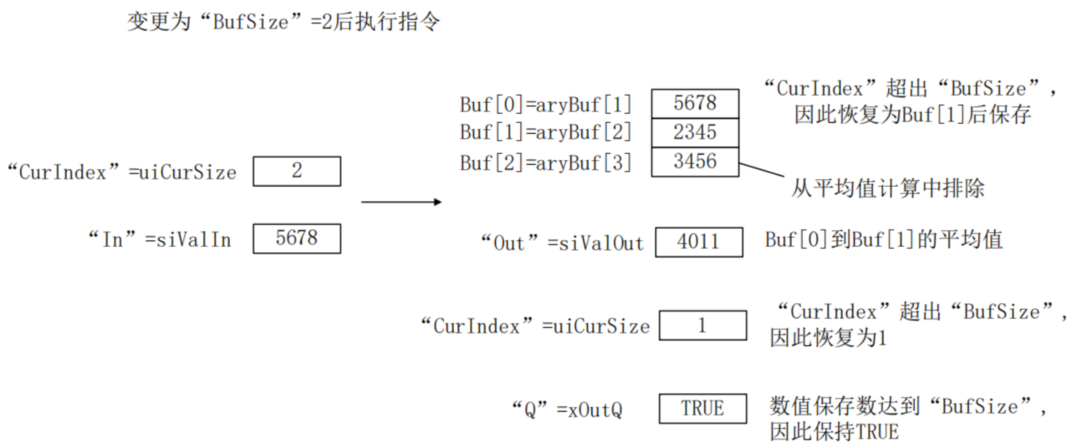




Initialization of saved values

If "CurSize" is set to 0 and then this instruction is executed, the values from Buf[0] to Buf["BufSize"-1] are set to 0, and the current value of "In" is then saved to Buf[0]. "CurIndex" becomes 1, and "Q" becomes FALSE. Change of "BufSize" Value: If the value of "BufSize" is changed and then this instruction is executed, operation proceeds based on the new "BufSize" value and the current "CurIndex" value.





◆ Key points

- The data types of "In", "Out", and the elements of Buf[] must be identical. Otherwise, an exception occurs during compilation.
- Even if the calculation result exceeds the valid range of "Out", no exception occurs; an illegal value is stored in "Out".
- When the value of "BufSize" is 0, the values of "Out" and "CurIndex" are 0, and "Q" becomes TRUE.

- After changing the value of "BufSize", "CurIndex" retains its current value.
- When the value of "BufSize" exceeds the size of Buf[], the function return value is FALSE, and "Out" remains unchanged.

5.9.9 DispartReal (Separate mantissa and exponent of a real number)

The instruction separates a real number into a signed mantissa (fraction) part and an exponent part.

Instruction	Name	FB/FUN	Graphical representation	ST representation
DispartReal	Separate mantissa and exponent of real number	FUN		Out:=DispartReal(In,Fraction, Exponent);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Real number	Input	Real number to be separated.	Conforms to data type.	—	(*1)
Out	Return value	Output	Always TRUE.	TRUE only.	—	—
Fraction	Signed mantissa (fraction)	Output	Signed mantissa (fraction).	(*2)		
Exponent	Exponent part	Output	Exponent part.	(*3)		

*1 Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

*2 Valid range varies depending on the combination of data types for "In" and "Fraction". See the Function section for details.

*3 When "In" is REAL type, valid range is -44 to 32. When "In" is LREAL type, valid range is -322 to 294.

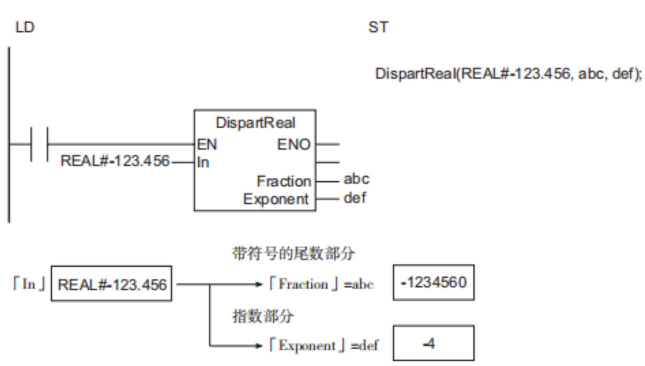
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In															○					
Out	○																			
Fraction													○							
Exponent											○									

◆ Function

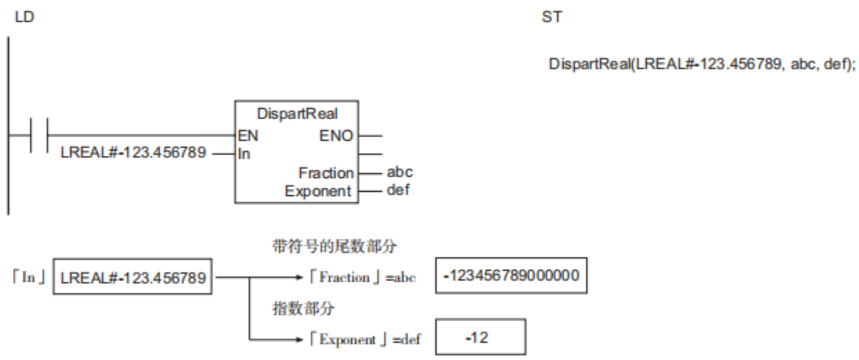
It separates the real number "In" into the signed mantissa (fraction) part "Fraction" and the exponent part "Exponent".

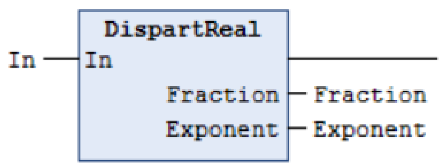
When "In" is of REAL type, "Fraction" is a 7-digit integer. When "In" is of LREAL type, "Fraction" is a 15-digit integer.

An example with "In" as REAL type, value REAL#-123.456, is shown below.



An example with "In" as REAL type, value REAL#-123.456789, is shown below.



	FBD	ST
Variable declaration	VAR In :LREAL; Fraction :LINT; Exponent :INT; END_VAR	
Program		DispartReal(In:=In , Fraction=>Fraction , Exponent=>Exponent);
Result	In LREAL -123.456789 Fraction LINT -123456789000000 Exponent INT -12	

◆ Reference

To combine a signed mantissa (fraction) and an exponent to obtain a real number, use the "UniteReal instruction (P.2-427)".

◆ Key points

・ When converting to an integer, errors may occur depending on the value of "In".

・ When the number of significant digits in "In" exceeds that of "Fraction", rounding is performed to keep it within the valid range of "Fraction". Rounding is handled as shown below.

Decimal part value	Action	Example
Less than 0.5	Truncate	1.49 to 1 -1.48 to -1
Greater than or equal to 0.5	Increment	1.51 to 2 -1.51 to -2

・ An exception occurs, ENO becomes FALSE, and "Fraction" and "Exponent" remain unchanged under the following conditions:

・ When the value of "In" is Not-a-Number (NaN) or infinite.

5.9.10 UnitReal (Combine mantissa and exponent into real number)

The instruction combines a signed mantissa (fraction) part and an exponent part into a real number.

Instruction	Name	FB/FUN	Graphical representation	ST representation
UniteReal	Combine mantissa and exponent into real number	FUN		Out:=DispartReal(In,Fraction, Exponent);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Fraction	Signed mantissa (fraction)	Input	Signed mantissa (fraction)	Conforms to data type.	—	(*)
Exponent	Exponent part		Exponent part			0
Out	Real number	Output	Real number	Conforms to data type.	—	—

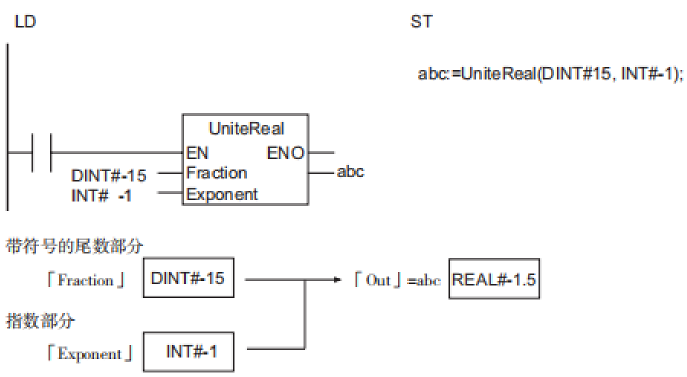
* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Fraction													○							
Exponent											○									
Out															○					

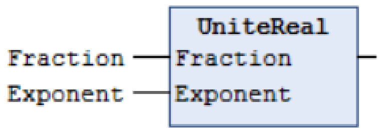
◆ Function

It combines the signed mantissa (fraction) part "Fraction" and the exponent part "Exponent" to obtain the real number "Out".

An example with "Fraction"=DINT#-15 and "Exponent"=INT#-1 is shown below.



	FBD	ST
Variable declaration	VAR Fraction :LINT; Exponent :INT; Out :LREAL; END_VAR	

Program		Out:=UniteReal(Fraction:=Fraction , Exponent:=Exponent);
Result	Fraction LINT -15 Exponent INT -1 Out LREAL -1.5	

◆ Reference

To separate a real number into a signed mantissa (fraction) part and an exponent part, use the "DispartReal instruction".

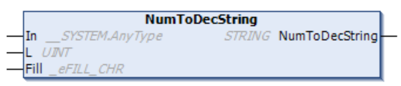
◆ Key points

- When converting an integer to a real number, errors may occur depending on the values of "Fraction" and "Exponent".
- If the combination result exceeds the valid range of "Out", "Out" becomes infinity with the same sign as "Fraction" when "Exponent" is positive. If "Exponent" is negative, "Out" becomes 0.

5.9.11 NumToDecString/NumToHexString (Fixed-length decimal/hexadecimal string conversion)

NumToDecString: Converts an integer to a fixed-length decimal string format.

NumToHexString: Converts an integer to a fixed-length hexadecimal string format.

Instruction	Name	FB/FUN	Graphical representation	ST representation
NumToDecString	Fixed-length decimal string conversion	FUN		Out:=NumToDecString(In, L, Fill);
NumToHexString	Fixed-length hexadecimal string conversion	FUN		Out:=NumToHexString(In, L, Fill);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Integer	Input	Integer	Conforms to data type.	—	(*)
L	Character count		Character count for "Out".	0~1985		1
Fill	Fill character		Fill character.	_BLANK _ZERO	—	_BLANK
Out	String	Output	String.	Conforms to data type.	—	—

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○							
L							○													
Fill	Enumeration elements of _eFILL_CHR: Refer to functional description.																			
Out																				○

◆ Function

NumToDecString: Converts the integer "In" to a string in UTF-8 half-width alphanumeric decimal format. A leading '-' (minus sign) is added if "In" is negative.

NumToHexString: Converts the integer "In" to a string in UTF-8 half-width alphanumeric hexadecimal format. If "In" is negative, it is represented in two's complement form (bit inversion + 1).

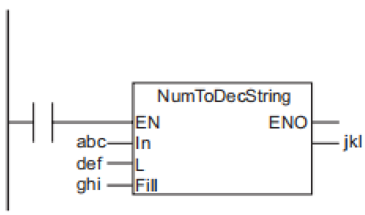
Both instructions set the character length of the output string "Out" to "L". If the result has fewer characters than "L", the character specified by "Fill" is added to the higher-order positions. If the conversion result has more characters than "L", only the "L" lowest-order characters from the result are assigned to "Out". "Out" is terminated with a null character. The character count "L" does not include the null character.

"Fill" is of the enumeration type _eFILL_CHR. The meaning of the enumeration element is as follows.

Enumeration element	Meaning
_BLANK	' ' (half-width space character)
_ZERO	'0'

Several examples for the NumToDecString instruction are shown below.

LD



ST

jkl:=NumToDecString(abc, def, ghi);

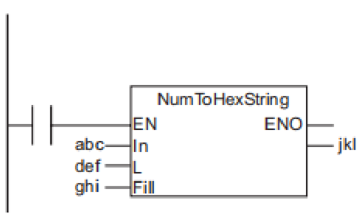
「In」=abc=INT#128、「L」=def=UINT#8、「Fill」=ghi=_BLANK
「Out」=jkl `     128`

「In」=abc=INT#-128、「L」=def=UINT#8、「Fill」=ghi=_BLANK
「Out」=jkl `    -128`

「In」=abc=INT#-128、「L」=def=UINT#8、「Fill」=ghi=_ZERO
「Out」=jkl `-0000128`

Several examples for the NumToHexString instruction are shown below.

LD



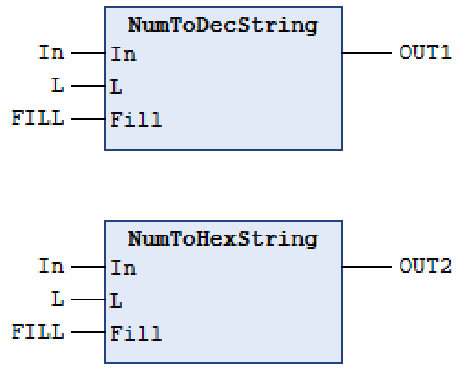
ST

jkl:=NumToHexString(abc, def, ghi);

「In」=abc=INT#128、「L」=def=UINT#8、「Fill」=ghi=_BLANK
「Out」=jkl `      80`

「In」=abc=INT#128、「L」=def=UINT#8、「Fill」=ghi=_ZERO
「Out」=jkl `00000080`

「In」=abc=INT#-128、「L」=def=UINT#8、「Fill」=ghi=_BLANK
「Out」=jkl `FFFFFF80`

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">VAR
In:INT;
L:UINT;
FILL:_eFILL_CHR;
OUT:STRING;
END_VAR</td></tr>
<tr><td>Program</td><td>
</td><td>OUT1:=NumToDecString(
In:=In ,
L:=L ,
Fill:=FILL);

OUT2:=NumToHexString(
In:=In ,
L:=L ,
Fill:=FILL);</td></tr>
<tr><td>Result</td><td colspan="2">
<table>
<tr><th>表达式</th><th>类型</th><th>值</th></tr>
<tr><td>In</td><td>INT</td><td>12345</td></tr>
<tr><td>L</td><td>UINT</td><td>6</td></tr>
<tr><td>FILL</td><td>_EFILL_CHR</td><td>_ZERO</td></tr>
<tr><td>OUT1</td><td>STRING</td><td>'012345'</td></tr>
<tr><td>OUT2</td><td>STRING</td><td>'003039'</td></tr>
</table>
</td></tr>
</table>

◆ Key points

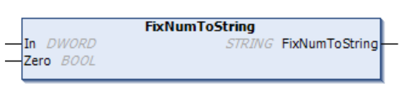
- When the value of "L" is 0, the value of "Out" remains unchanged.
- When the conversion result has more characters than "L", the "L" lowest-order characters from the result are stored in "Out". An example is shown below.

Instruction	Value of "In"	Value of "L"	Value of "Out"
NumToDecString	128	2	20
NumToHexString			80

- An exception occurs, ENO becomes FALSE, and "Out" remains unchanged under the following conditions:
- When the value of "L" is outside its valid range.
- When the value of "Fill" is outside its valid range.
- When the conversion result exceeds the range of "Out".

5.9.12 FixNumToString (Fixed-point number to string conversion)

The instruction converts a signed fixed-point number to a decimal string format.

Instruction	Name	FB/FUN	Graphical representation	ST representation
FixNumToString	Fixed-point number to string conversion	FUN		Out:=FixNumToString(In, Zero);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Fixed-point number	Input	Signed fixed-point number.	Conforms to data type.	—	0
Zero	Zero display		Display when decimal places are less than 3. TRUE: Pads with "0". FALSE: Does not pad with "0".			TRUE
Out	Decimal string	Output	Decimal string.	Conforms to data type.	—	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In				○																
Zero	○																			
Out																				○

◆ Function

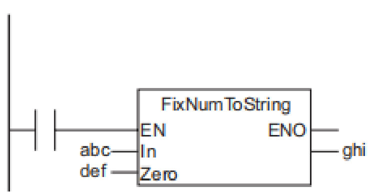
It converts the signed fixed-point number "In" to a decimal string. The conversion steps are as follows.

1. Convert "In" from hexadecimal format to decimal format.

2. Divide that value by 1,000.

When the number of decimal places in "In" is less than 3, the zero display "Zero" specifies whether to pad with '0' before the 3rd decimal place in "Out". If "Zero" is TRUE, it is padded with '0'. "Out" is terminated with a null character. Several examples are shown below.

LD



ST

ghi:=FixNumToString(abc, def);

"In" =abc	"Out" =ghi	
	"Zero" =def=TRUE	"Zero" =def=FALSE
16#0001462C (10#83500)	'83.500'	'83.5'
16#00051AA4 (10#334500)	'334.500'	'334.5'
16#0003BEFC (10#245500)	'245.500'	'245.5'

	FBD	ST
Variable declaration	VAR In:INT; L:UINT; FILL:_eFILL_CHR; OUT:STRING; END_VAR	

Program	In — In FixNumToString — OUT1 Zero — Zero	OUT1:=FixNumToString(In:=In , Zero:=Zero);
Result	表达式 / 类型 / 值 In / INT / 12345 L / UINT / 6 FILL / _EFILL_CHR / _ZERO OUT1 / STRING / '012345' OUT2 / STRING / '003039'	

◆ Key points

- Digits beyond the 4th decimal place of "In" are truncated.
- Underscores (16#5F) within the "In" string are ignored.
- An exception occurs, ENO becomes FALSE, and "Out" remains unchanged under the following conditions:
- When "In" is not null-terminated.
- When the content of "In" contains characters that cannot be converted to a numeric value.
- When the content of "In" contains a decimal point but no fractional part.
- When the conversion result exceeds the valid range of "Out".

5.9.13 StringToFixNum (String to fixed-point number conversion)

The instruction converts a decimal string to a signed fixed-point number.

Instruction	Name	FB/FUN	Graphical representation	ST representation
StringToFixNum	String to fixed-point number conversion	FUN	StringToFixNum In STRING — DWORD StringToFixNum	Out:=StringToFixNum(In);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Decimal string	Input	Decimal string.	Conforms to data type.	—	''
Out	Fixed-point number	Output	Signed fixed-point number.	Conforms to data type.	—	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out				○																

◆ Function

It converts the decimal string "In" to a fixed-point number. The conversion steps are as follows.

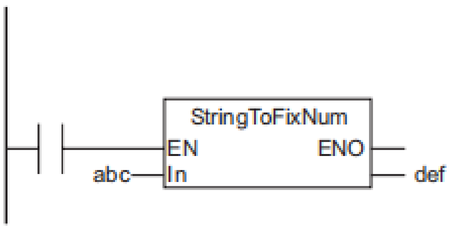
1. Multiply the numeric value represented by "In" by 1,000.

2. Truncate the fractional part of that value.

3. Convert that value to a 32-bit hexadecimal format (DWORD type).

Example values are shown below.

LD



ST

def:=StringToFixNum(abc);

"In"=abc	"Out"=def
'83.5'	16#0001462C (10#83500)
'334.5'	16#00051AA4 (10#334500)
'245.5'	16#0003BEFC (10#245500)

	FBD	ST
Variable declaration	VAR In:STRING; out:DWORD; END_VAR	
Program	In — StringToFixNum (In) — out	out:=StringToFixNum(In:=In);
Result	表达式 / 类型 / 值 In / STRING / '83.5' Out / DWORD / 83500	

◆ Key points

- Digits beyond the 4th decimal place of "In" are truncated.
- Underscores (16#5F) within the "In" string are ignored.
- An exception occurs, ENO becomes FALSE, and "Out" remains unchanged under the following conditions:
- When "In" is not null-terminated.
- When the content of "In" contains characters that cannot be converted to a numeric value.
- When the content of "In" contains a decimal point but no fractional part.
- When the conversion result exceeds the valid range of "Out".

5.9.14 DtToString (Date and time to string conversion)

The instruction converts date and time to a string format.

Instruction	Name	FB/FUN	Graphical representation	ST representation
DtToString	Date and time to string conversion	FUN	DtToString: In (DATE_AND_TIME) — (STRING) DtToString	Out:=DtToString(In);

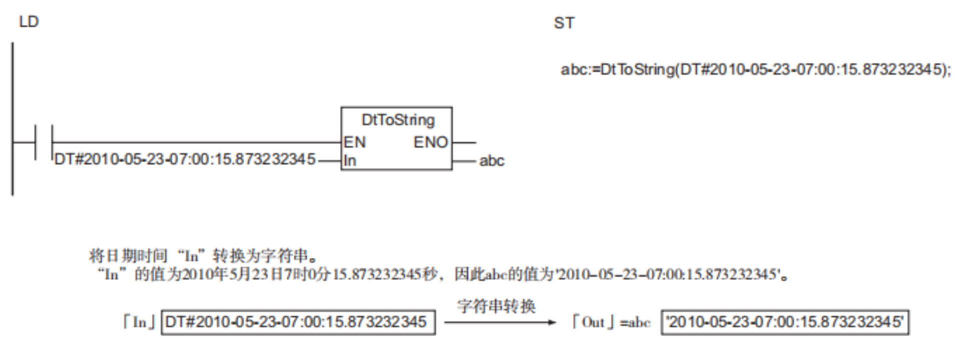
◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Date and time	Input	Date and time	Conforms to data type.	Year, month, day, hour, minute, second	DT#1970-1-1- 0:0:0
Out	String	Output	String	30 bytes (29 half-width alphanumeric characters + terminating null character).	—	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			○	
Out																				○

◆ Function

It converts the date and time "In" to a string. The string "Out" is terminated with a null character. An example with "In" representing 2010-05-23 07:00:15.873232345 is shown below. Variable abc has the value '2010-05-23-07:00:15.873232345'.



	FBD	ST
Variable declaration	VAR In:DATE_AND_TIME; out:STRING; END_VAR	
Program	In — DtToString In — out	out:=DtToString(In:=In);
Result	表达式 / 类型 / 值 In / DATE_AND_TIME / DT#1970-1-1-0:0:0 out / STRING / '1970-01-01-00:00:00'	

◆ Key points

- An exception occurs, ENO becomes FALSE, and "Out" remains unchanged under the following conditions:
- When the conversion result exceeds the valid range of "Out".

5.9.15 DateToString (Date to string conversion)

The instruction converts a date to a string format.

Instruction	Name	FB/FUN	Graphical representation	ST representation
DateToString	Date to string conversion	FUN		Out:=DateToString(In);

◆ Variables

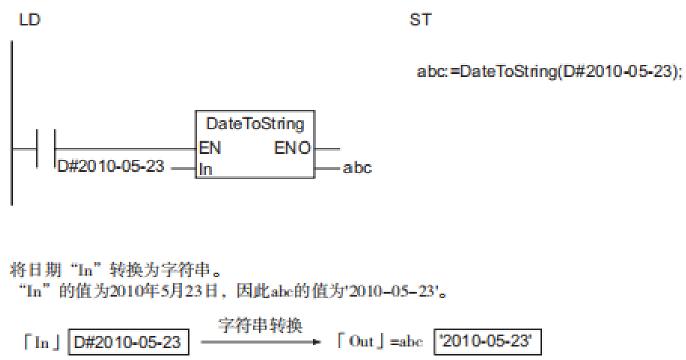
	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Date	Input	Date	Conforms to data type.	Year, month, day	D#1970-1-1
Out	String	Output	String	11 bytes (10 half-width alphanumeric characters + terminating null character).	—	—

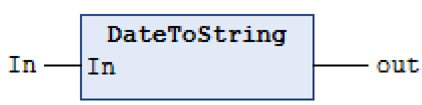
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			○	
Out																				○

◆ Function

It converts the date "In" to a string. The string "Out" is terminated with a null character.

An example with "In" representing 2010-05-23 is shown below. Variable abc has the value '2010-05-23'.



	FBD	ST
Variable declaration	VAR In:DATE; out:STRING; END_VAR	
Program		out:=DateToString(In:=In);

Result	表达式	类型	值
	In	DATE	D#1970-1-1
	out	STRING	'1970-01-01'

◆ Key points

- An exception occurs, ENO becomes FALSE, and "Out" remains unchanged under the following conditions:
- When the conversion result exceeds the valid range of "Out".

5.9.16 StringToAry (String to array conversion)

The instruction converts a string to a BYTE-type array.

Instruction	Name	FB/FUN	Graphical representation	ST representation
StringToAry	String to array conversion	FUN		StringToAry(In:= , AryOut:=);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	String	Input	String	Conforms to data type.	—	‘ ’
AryOut[ ] array	BYTE-type array	Input/Output	BYTE-type array	Conforms to data type.	—	—
Out	Number of bytes converted	Output	Number of bytes converted	0~1985	BYTE	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
AryOut[ ] array		○																		
Out							○													

◆ Function

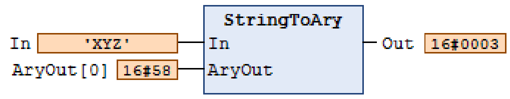
It treats the character codes of the string "In" as numeric values and saves them character by character into the BYTE-type array AryOut[]. "Out" holds the number of bytes converted.

An example with "In"='XYZ' is shown below.

	FBD	ST
Variable declaration		

```
PROGRAM PLC_PRG
VAR
    In        :STRING(1985)  :='XYZ';
    AryOut    :ARRAY [0..4] OF BYTE;
    Out       :UINT;
    check     :BOOL;
END_VAR
```

Program




```
IF check FALSE THEN
    Out 16#0003 :=StringToAry(
        In:= In 'XYZ' ,
        AryOut:= AryOut[0] 16#58 );
    check FALSE :=FALSE;
END_IF RETURN
```

Result

表达式	类型	值
In	STRING(1985)	'XYZ'
AryOut	ARRAY [0..4] OF BYTE	
AryOut[0]	BYTE	16#58
AryOut[1]	BYTE	16#59
AryOut[2]	BYTE	16#5A
AryOut[3]	BYTE	16#00
AryOut[4]	BYTE	16#00
Out	UINT	16#0003

◆ Key points

- The null character terminating "In" is not stored in AryOut[].
- When "In" is a string consisting only of a null character, the value of "Out" becomes 0, and AryOut[] remains unchanged.
- When the byte count of "In" exceeds the number of elements in AryOut[], the excess portion cannot be stored.

5.9.17 AryToString (Array to string conversion)

The instruction converts a BYTE-type array to a string.

Instruction	Name	FB/FUN	Graphical representation	ST representation
AryToString	Array to string conversion	FUN		StringToAry(In:= , AryOut:=);

◆ Variables

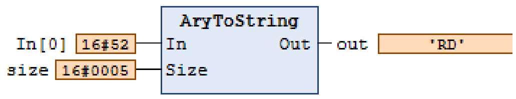
	Name	Input/Output	Description	Valid range	Unit	Initial value
In[ ] array	BYTE-type array	Input	BYTE-type array, maximum element count 1985.	Conforms to data type.	—	(*)
Size	Number of elements to convert	Input/Output	Number of elements in In[] to be converted.	0~1985		1
Out	String	Output	String	Conforms to data type.	—	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ] array		○																		
Size							○													
Out																				○

◆ Function

It treats the values of the elements starting from In[0] of the BYTE-type array as character codes and saves them in the string "Out". "Out" is terminated with a null character. "Size" specifies the number of elements in In[] to convert. If a null character is found within In[0] to In["Size"-1], only the content up to that point is stored in "Out".

An example with "Size"=UINT#5 is shown below. Only data before the null character is converted to "Out", i.e., In[0] = 16#52, In[1] = 16#44. This converts to the string 'RD'.

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2"><pre>PROGRAM PLC_PRG
VAR
    In: ARRAY[0..4] OF byte := [16#52, 16#44, 16#0, 16#52, 16#44];
    size   :UINT:=5;
    out   :STRING;
    check   :BOOL;
END_VAR</pre></td></tr>
<tr><td>Program</td><td>
</td><td><pre>IF check FALSE THEN
    AryToString(
        In:= In[0] 16#52 ,
        Size:= size 16#0005 ,
        Out=> out 'RD');
    check FALSE :=FALSE;
END_IF RETURN</pre></td></tr>
<tr><td>Result</td><td colspan="2"><table>
<tr><th>表达式</th><th>类型</th><th>值</th></tr>
<tr><td>In</td><td>ARRAY [0..4] OF BYTE</td><td></td></tr>
<tr><td>In[0]</td><td>BYTE</td><td>16#52</td></tr>
<tr><td>In[1]</td><td>BYTE</td><td>16#44</td></tr>
<tr><td>In[2]</td><td>BYTE</td><td>16#00</td></tr>
<tr><td>In[3]</td><td>BYTE</td><td>16#52</td></tr>
<tr><td>In[4]</td><td>BYTE</td><td>16#44</td></tr>
<tr><td>size</td><td>UINT</td><td>16#0005</td></tr>
<tr><td>out</td><td>STRING</td><td>'RD'</td></tr>
</table></td></tr>
</table>

◆ Key points

- When the value of "Size" is 0, "Out" is a string containing only a null character.
- An exception occurs, ENO becomes FALSE, and "Out" remains unchanged under the following conditions:
- When the value of "Size" exceeds the array bounds of In[].
- When the conversion result exceeds the valid range of "Out".

5.9.18 RoundUp (Real number round up)

The instruction rounds up the first decimal place.

<table>
<tr><th>Instruction</th><th>Name</th><th>FB/FUN</th><th>Graphical representation</th><th>ST representation</th></tr>
<tr><td>RoundUp</td><td>Real number round up</td><td>FUN</td><td>
</td><td>RoundUp(In, Out);</td></tr>
</table>

◆ Variables

<table>
<tr><th></th><th>Name</th><th>Input/Output</th><th>Description</th><th>Valid range</th><th>Unit</th><th>Initial value</th></tr>
<tr><td>In</td><td>Conversion target</td><td rowspan="2">Input</td><td>Conversion target</td><td>Conforms to data type.</td><td>—</td><td>(*)</td></tr>
<tr><td>Out</td><td>Conversion result</td><td>Conversion result</td><td>Conforms to data type.</td><td>—</td><td>—</td></tr>
</table>

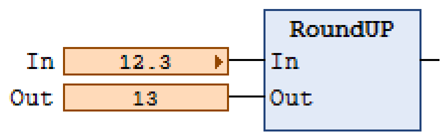
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out												○	○							

◆ Function

It rounds up the first decimal place of the real number "In" to make it an integer.

An example with "In" of type REAL = 12.34 is shown below.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In :REAL:=12.34; Out :DINT; check :BOOL; END_VAR	

Program		IF check FALSE THEN
		RoundUP(
		In:=In 12.3 ,
		Out:=Out 13);
		check FALSE :=FALSE;
		END_IF RETURN

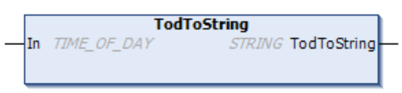
Result	表达式 / 类型 / 值 In / REAL / 12.34 Out / DINT / 13	

◆ Key points

- If the conversion result exceeds the valid range of "Out", the value of "Out" becomes an error value.

5.9.19 TodToString (Time of day to string conversion)

The instruction converts a time of day to a string format.

Instruction	Name	FB/FUN	Graphical representation	ST representation
TodToString	Time of day to string conversion	FUN		Out:=TodToString(In);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Time of day	Input	Time of day	Conforms to data type.	Hour, minute, second	TOD#0:0:0
Out	String	Output	String	13 bytes (12 half-width alphanumeric characters + terminating null character).	—	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																		○		
Out																				○

◆ Function

It converts the time of day "In" to a string. "Out" is terminated with a null character.

An example with the input variable "In" = 23:23:23 is shown below.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In :TOD; Out :STRING; END_VAR	
Program	In TOD#23:23:23 — TodToString In — Out '23:23:23'	out '23:23:23' :=TodToString(In:= In TOD#23:23:23); RETURN
Result	表达式 / 类型 / 值 In / TIME_OF_DAY / TOD#23:23:23 Out / STRING / '23:23:23'	

◆ Key points

- An exception occurs, ENO becomes FALSE, and "Out" remains unchanged under the following conditions:
- When the conversion result exceeds the valid range of "Out".

5.9.20 StringToDt (String to date and time conversion)

The instruction converts a string to a date and time format.

Instruction	Name	FB/FUN	Graphical representation	ST representation
StringToDt	String to date and time conversion	FUN	StringToDt In STRING — BOOL StringToDt DATE_AND_TIME Out	StringToDt(In:= , Out=>);

◆ Variables

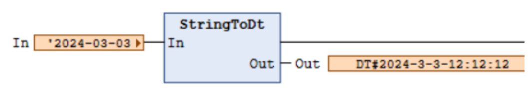
	Name	Input/Output	Description	Valid range	Unit	Initial value
In	String	Input	String	Conforms to data type.		
Out	Date	Output	Date and time	Conforms to data type.	Year, month, day, hour, minute, second	DT#1970-1-1-0:0:0

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out																			○	

◆ Function

It converts the string "In" to a date and time.

An example with the input variable "In" ='2024-03-03-12:12:12' is shown below.

	FBD	ST
Variable declaration	VAR In :STRING; Out :DT; END_VAR	
Program		StringToDt(In:= In '2024-03-03', Out=>Out DT#2024-3-3-12:12:12);
Result	表达式 / 类型 / 值 In / STRING / '2024-03-03-12:12:12' Out / DATE_AND_TIME / DT#2024-3-3-12:12:12	

5.9.21 AryToWstring (Array to wide string conversion)

The instruction converts a BYTE-type array to a wide string.

Instruction	Name	FB/FUN	Graphical representation	ST representation
AryToWstring	Array to wide string conversion	FUN		AryToWstring(In:= , Size:= , Out=>);

◆ Variables

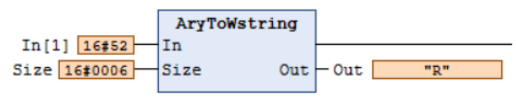
	Name	Input/Output	Description	Valid range	Unit	Initial value
In[ ] array	BYTE-type array	Input	BYTE-type array, maximum element count 1985.	Conforms to data type.	—	(*)
Size	Number of elements to convert		Number of elements in In[] to be converted.	0~1985		1
Out	Wide string	Output	Wide string.	Conforms to data type.	—	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ] array		○																		
Size							○													
Out																				○

◆ Function

It treats the values of two consecutive elements starting from In[0] of the BYTE-type array as a single character code, and saves it in the wide string "Out". "Out" is terminated with null characters. "Size" specifies the number of elements in In[] to convert. If two consecutive null characters are found within In[0] to In["Size"-1], only the content up to that point is stored in "Out".

An example with "Size"=UINT#6 is shown below. Only data before the null characters is converted to "Out", i.e., In[0] = 16#52, In[1] = 16#00. This converts to the wide string 'R'.

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">VAR
In :ARRAY[1..6] OF BYTE :=[16#52,16#00,16#0,16#0,16#44,16#00];
Size :UINT :=6;
Out :WSTRING;
END_VAR</td></tr>
<tr><td>Program</td><td></td><td>AryToWstring(In:= In[1] 16#52,
Size:= Size 16#0006,
Out=> Out "R");</td></tr>
<tr><td>Result</td><td colspan="2">
<table>
<tr><td>In</td><td>ARRAY [1..6] OF BYTE</td><td></td></tr>
<tr><td>In[1]</td><td>BYTE</td><td>16#52</td></tr>
<tr><td>In[2]</td><td>BYTE</td><td>16#00</td></tr>
<tr><td>In[3]</td><td>BYTE</td><td>16#00</td></tr>
<tr><td>In[4]</td><td>BYTE</td><td>16#00</td></tr>
<tr><td>In[5]</td><td>BYTE</td><td>16#53</td></tr>
<tr><td>In[6]</td><td>BYTE</td><td>16#00</td></tr>
<tr><td>Size</td><td>UINT</td><td>16#0006</td></tr>
<tr><td>Out</td><td>WSTRING</td><td>"R"</td></tr>
</table>
</td></tr>
</table>

◆ Key points

- When the value of "Size" is 0, "Out" is a wide string containing only null characters.
- Conversion of one Wstring character requires two Bytes.
- An exception occurs, ENO becomes FALSE, and "Out" remains unchanged under the following conditions:
- When the value of "Size" exceeds the array bounds of In[].
- When the conversion result exceeds the valid range of "Out".

5.9.22 WstringToAry (Wide string to array conversion)

The instruction converts a wide string to a BYTE-type array.

Instruction	Name	FB/FUN	Graphical representation	ST representation
WstringToAry	Wide string to array conversion	FUN		WstringToAry(In:= , AryOut:=);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Wide string	Input	Wide string	Conforms to data type.	—	''
AryOut[ ] array	Byte-type array	Input/Output	Byte-type array	Conforms to data type.	—	—
Out	Number of bytes converted	Output	Number of bytes converted	0~1985	BYTE	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
AryOut[ ] array		○																		
Out							○													

◆ Function

It treats the character codes of the wide string "In" as numeric values and saves them character by character into the BYTE-type array AryOut[]. "Out" holds the number of bytes converted.

An example with "In"= "ABC" is shown below.

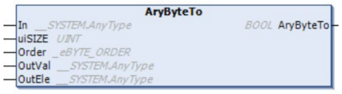
	FBD	ST
Variable declaration	VAR AryOut :ARRAY[1..6] OF BYTE ; Out :UINT; In :WSTRING :="ABC"; END_VAR	
Program	In "ABC" — In WstringToAry Out — Out 16#0003 AryOut[1] 16#41 — AryOut	Out 16#0003 :=WstringToAry(In:= In "ABC" AryOut:= AryOut[1] 16#41);
Result	AryOut ARRAY [1..6] OF BYTE AryOut[1] BYTE 16#41 AryOut[2] BYTE 16#00 AryOut[3] BYTE 16#42 AryOut[4] BYTE 16#00 AryOut[5] BYTE 16#43 AryOut[6] BYTE 16#00 Out UINT 16#0003 In WSTRING "ABC"	

◆ Key points

- The null characters terminating "In" are not stored in AryOut[].
- When "In" is a wide string consisting only of null characters, the value of "Out" becomes 0, and AryOut[] remains unchanged.
- When the byte count of "In" exceeds the number of elements in AryOut[], the excess portion cannot be stored.
- Conversion of one Wstring character requires two Bytes.

5.9.23 AryByteTo (Convert from byte array)

The instruction combines elements of a BYTE-type array and saves the result to a variable.

Instruction	Name	FB/FUN	Graphical representation	ST representation
AryByteTo	Convert from byte array	FUN		AryByteTo(In:= , uiSIZE:= , Order:= , OutVal:= , OutEle:=);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In[] array	Source array		Source array.	Conforms to data type.		(*)
Size	Number of elements		Number of elements in In[] to convert.	Conforms to data type.		
Order	Conversion order		Order of conversion.	_LOW_HIGH, _HIGH_LOW		_LOW_HIGH
OutVal	Result array		Result array.	Conforms to data type.		
OutEle	Result array element		Starting element in the result array.	Conforms to data type.		

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] array		○																		
Size							○													
Order	Enumeration type _eBYTE_ORDER. See Function description for enumeration values.																			
OutVal	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
OutEle	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

◆ Function

It combines the first "Size" elements of the source array In[], calculated based on the data type size of the element "OutEle" in the array "OutVal", and saves the result to "OutVal".

The order for combining In[] elements is specified in the conversion order "Order". "Order" is of the enumeration type _eBYTE_ORDER. The meanings of the enumeration values are as follows.

Enumeration value	Meaning
_LOW_HIGH	Low-order byte first, then high-order byte.
_HIGH_LOW	High-order byte first, then low-order byte.

When the data type of "OutEle" is 2 bytes or more, the processing steps are as follows:

1. Combine In[0] and In[1] according to the "Order" value to create 1 word (2 bytes) of data. If "Order" is _LOW_HIGH, the high-order byte is In[1] and the low-order byte is In[0]. If "Order" is _HIGH_LOW, the high-order byte is In[0] and the low-order byte is In[1].

2. Similarly combine values from In[2] and In[3] onward to create multiple 1-word data segments.

3. Continue until "Size" elements are used, saving the data sequentially into "OutVal".

Example with "OutVal" as DWORD-type array, Size =UINT#4, "In" as the 1st element of a BYTE array, Order =_LOW_HIGH, "OutEle" as the 1st element of the "OutVal" array:

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_LOW_HIGH	
Size	UINT	16#0004	
AryIn	ARRAY [1..6] OF BYTE		
AryIn[1]	BYTE	16#78	
AryIn[2]	BYTE	16#56	
AryIn[3]	BYTE	16#34	
AryIn[4]	BYTE	16#12	
AryIn[5]	BYTE	16#00	
AryIn[6]	BYTE	16#00	
AryOut	ARRAY [1..3] OF DWORD		
AryOut[1]	DWORD	16#00000000	
AryOut[2]	DWORD	16#12345678	
AryOut[3]	DWORD	16#00000000	

```
AryByteTo(In:= AryIn[1] 16#78 , uiSIZE:= Size 16#0004 , Order:= odertest _LOW_HIGH ,
          OutVal:= AryOut, Outele:= AryOut[2] 16#12345678 );
```

Example with "OutVal" as DWORD-type array, Size =UINT#4, "In" as the 3rd element of a BYTE array, Order =_HIGH_LOW, "OutEle" as the 2nd element of the "OutVal" array:

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_HIGH_LOW	
Size	UINT	16#0004	
AryIn	ARRAY [1..6] OF BYTE		
AryIn[1]	BYTE	16#78	
AryIn[2]	BYTE	16#56	
AryIn[3]	BYTE	16#34	
AryIn[4]	BYTE	16#12	
AryIn[5]	BYTE	16#91	
AryIn[6]	BYTE	16#92	
AryOut	ARRAY [1..3] OF DWORD		
AryOut[1]	DWORD	16#00000000	
AryOut[2]	DWORD	16#91923412	
AryOut[3]	DWORD	16#00000000	

```
AryByteTo(In:= AryIn[3] 16#34 , uiSIZE:= Size 16#0004 , Order:= odertest _HIGH_LOW ,
          OutVal:= AryOut, Outele:= AryOut[2] 16#91923412 );
```

Example when the data type of "OutEle" is 1 byte:

Example with "OutVal" as SINT-type array, Size =UINT#3, "In" as the 1st element of a BYTE array, Order =_LOW_HIGH, "OutEle" as the 1st element of the "OutVal" array:

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_LOW_HIGH	
Size	UINT	16#0003	
AryIn	ARRAY [1..6] OF BYTE		
AryIn[1]	BYTE	16#11	
AryIn[2]	BYTE	16#22	
AryIn[3]	BYTE	16#33	
AryIn[4]	BYTE	16#00	
AryIn[5]	BYTE	16#00	
AryIn[6]	BYTE	16#00	
AryOut	ARRAY [1..4] OF SINT		
AryOut[1]	SINT	16#11	
AryOut[2]	SINT	16#22	
AryOut[3]	SINT	16#33	
AryOut[4]	SINT	16#00	

```
AryByteTo(In:= AryIn[1] 16#11 , uiSIZE:= Size 16#0003 , Order:= odertest _LOW_HIGH ,
          OutVal:= AryOut, Outele:= AryOut[1] 16#11 );
```

Other conditions same as above, Order =_HIGH_LOW, example:

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_HIGH_LOW	
Size	UINT	16#0003	
AryIn	ARRAY [1..6] OF BYTE		
AryIn[1]	BYTE	16#11	
AryIn[2]	BYTE	16#22	
AryIn[3]	BYTE	16#33	
AryIn[4]	BYTE	16#00	
AryIn[5]	BYTE	16#00	
AryIn[6]	BYTE	16#00	
AryOut	ARRAY [1..4] OF SINT		
AryOut[1]	SINT	16#22	
AryOut[2]	SINT	16#11	
AryOut[3]	SINT	16#00	
AryOut[4]	SINT	16#33	

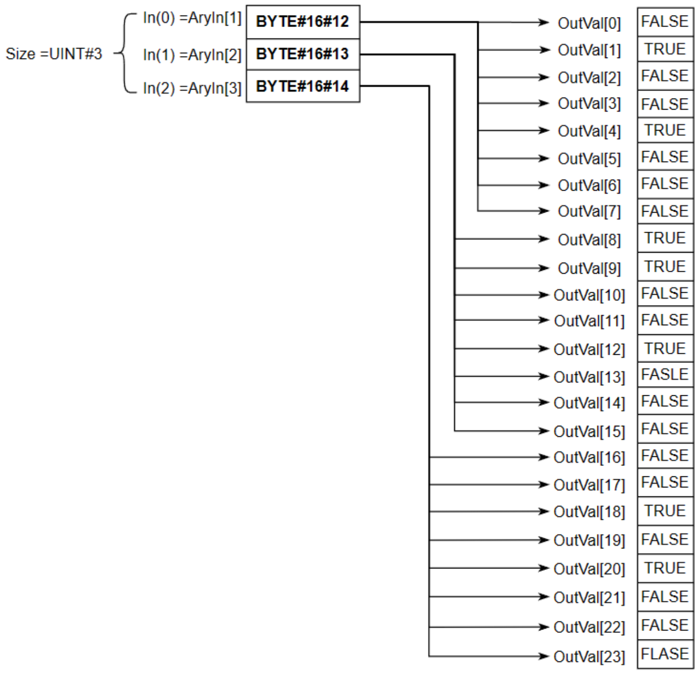
```
AryByteTo(In:= AryIn[1] 16#11 , uiSIZE:= Size 16#0003 , Order:= odertest _HIGH_LOW ,
          OutVal:= AryOut, Outele:= AryOut[1] 16#22 );
```

Example when the data type of "OutEle" is BOOL:

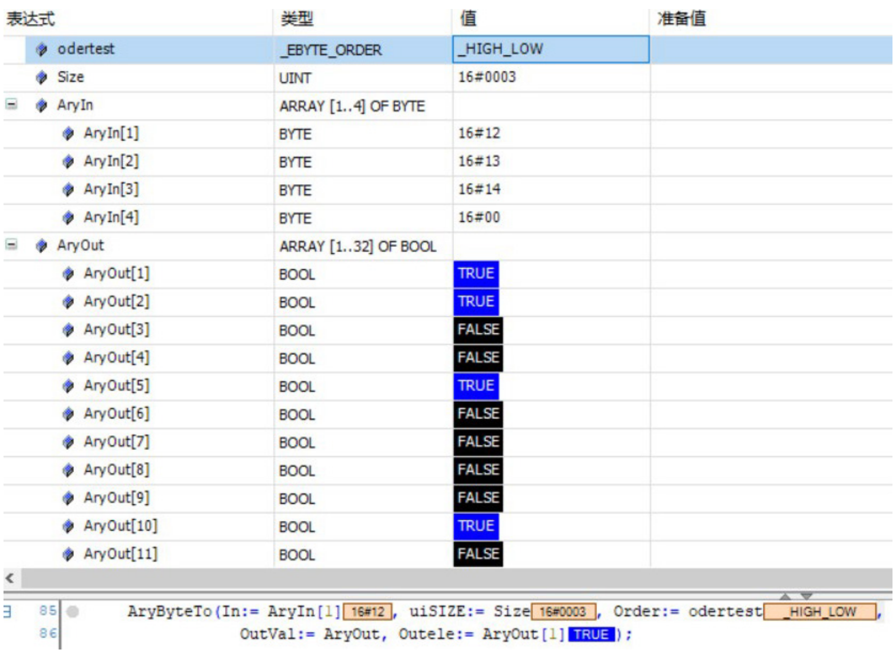
Example with "OutVal" as BOOL-type array, Size =UINT#3, "In" as the 1st element of a BYTE array, Order =_LOW_HIGH, "OutEle" as the 1st element of the "OutVal" array:

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_LOW_HIGH	
Size	UINT	16#0003	
AryIn	ARRAY [1..4] OF BYTE		
AryIn[1]	BYTE	16#12	
AryIn[2]	BYTE	16#13	
AryIn[3]	BYTE	16#14	
AryIn[4]	BYTE	16#00	
AryOut	ARRAY [1..32] OF BOOL		
AryOut[1]	BOOL	FALSE	
AryOut[2]	BOOL	TRUE	
AryOut[3]	BOOL	FALSE	
AryOut[4]	BOOL	FALSE	
AryOut[5]	BOOL	TRUE	
AryOut[6]	BOOL	FALSE	
AryOut[7]	BOOL	FALSE	
AryOut[8]	BOOL	FALSE	
AryOut[9]	BOOL	TRUE	
AryOut[10]	BOOL	TRUE	
AryOut[11]	BOOL	FALSE	

```
AryByteTo(In:= AryIn[1] 16#12 , uiSIZE:= Size 16#0003 , Order:= odertest _LOW_HIGH ,
          OutVal:= AryOut, Outele:= AryOut[1] FALSE );
```

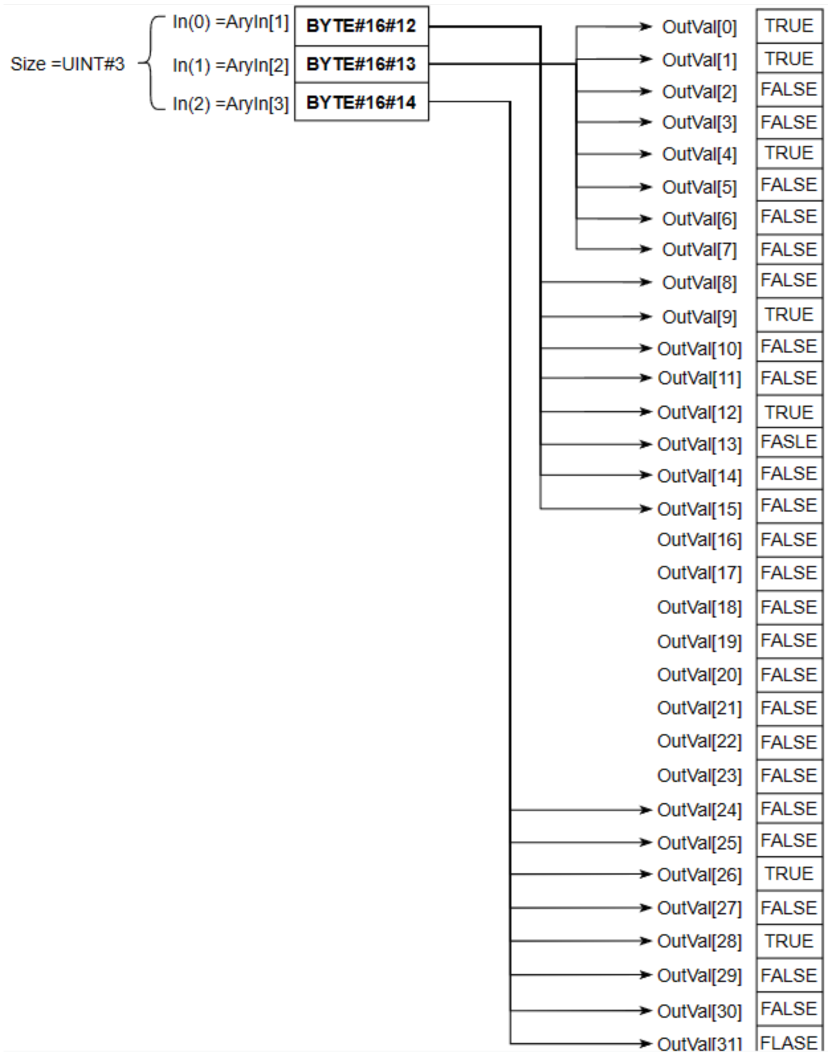


Other conditions same as above, Order =_HIGH_LOW, example:



表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_HIGH_LOW	
Size	UINT	16#0003	
AryIn	ARRAY [1..4] OF BYTE		
AryIn[1]	BYTE	16#12	
AryIn[2]	BYTE	16#13	
AryIn[3]	BYTE	16#14	
AryIn[4]	BYTE	16#00	
AryOut	ARRAY [1..32] OF BOOL		
AryOut[1]	BOOL	TRUE	
AryOut[2]	BOOL	TRUE	
AryOut[3]	BOOL	FALSE	
AryOut[4]	BOOL	FALSE	
AryOut[5]	BOOL	TRUE	
AryOut[6]	BOOL	FALSE	
AryOut[7]	BOOL	FALSE	
AryOut[8]	BOOL	FALSE	
AryOut[9]	BOOL	FALSE	
AryOut[10]	BOOL	TRUE	
AryOut[11]	BOOL	FALSE	

```
AryByteTo(In:= AryIn[1] 16#12 , uiSIZE:= Size 16#0003 , Order:= odertest _HIGH_LOW ,
    OutVal:= AryOut, Outele:= AryOut[1] TRUE );
```

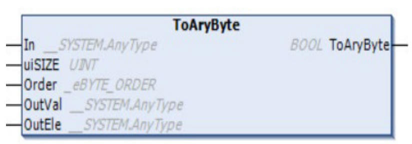


◆ Key points

- The input array "In" must be in Byte array element format.
- For "OutVal", specify the array. For "OutEle", specify the starting array element for the operation.
- If the output array has insufficient memory for the conversion, the function returns status FALSE and performs no action.

5.9.24 ToAryByte (Convert to byte array)

The instruction splits a variable into 1-byte units and saves them to a BYTE-type array.

Instruction	Name	FB/FUN	Graphical representation	ST representation
ToAryByte	Convert to byte array	FUN		ToAryByte(In:= , uiSIZE:= , Order:= , OutVal:= , OutEle:=);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In[] array	Source array		Source array.	Conforms to data type.		(*)
Size	Number of elements		Number of elements in In[] to convert.	Conforms to data type.		
Order	Conversion order		Order of conversion.	_LOW_HIGH, _HIGH_LOW		_LOW_HIGH
OutVal	Result array		BYTE array.	Conforms to data type.		
OutEle	Result array element		Starting element in the result array.	Conforms to data type.		

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] array	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Size							○													
Order	Enumeration type _eBYTE_ORDER. See Function description for enumeration values.																			
OutVal		○																		
OutEle		○																		

◆ Function

It splits the first "Size" elements of the source array In[] into 1-byte units, and saves the result to the BYTE array "OutVal" starting from element "OutEle".

The order for converting In[] values into 1-byte units is specified in the conversion order "Order". "Order" is of the enumeration type _eBYTE_ORDER. The meanings of the enumeration values are as follows.

Enumeration value	Meaning
_LOW_HIGH	Low-order byte first, then high-order byte.
_HIGH_LOW	High-order byte first, then low-order byte.

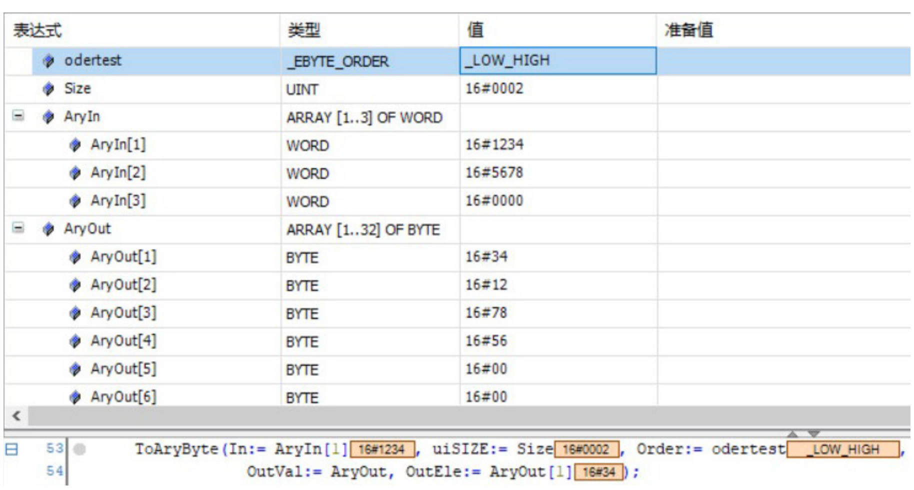
When the data type of the elements in "In" is 2 bytes or more, the processing steps are as follows:

1. Split the value of "In" into 2-byte units. Then, within each 2-byte unit, split further into high-order and low-order bytes.

2. If "Order" is _LOW_HIGH, save to "OutVal" starting from element "OutEle" in the order: low-order byte first, then high-order byte. If "Order" is _HIGH_LOW, save in the order: high-order byte first, then low-order byte.

3. Continue until data from "Size" elements is used, saving the data sequentially into "OutVal".

Example with "In" as the 1st element of a WORD-type array, Size =UINT#2, Order =_LOW_HIGH, "OutEle" as the 1st element of the "OutVal" array:



Example with "In" as the 2nd element of a WORD-type array, Size =UINT#2, Order = _HIGH_LOW, "OutEle" as the 2nd element of the "OutVal" array:

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_HIGH_LOW	
Size	UINT	16#0002	
AryIn	ARRAY [1..3] OF WORD		
AryIn[1]	WORD	16#1010	
AryIn[2]	WORD	16#2345	
AryIn[3]	WORD	16#6789	
AryOut	ARRAY [1..32] OF BYTE		
AryOut[1]	BYTE	16#00	
AryOut[2]	BYTE	16#23	
AryOut[3]	BYTE	16#45	
AryOut[4]	BYTE	16#67	
AryOut[5]	BYTE	16#89	
AryOut[6]	BYTE	16#00	

```
ToAryByte(In:= AryIn[2] 16#2345, uiSIZE:= Size 16#0002, Order:= odertest _HIGH_LOW,
          OutVal:= AryOut, OutEle:= AryOut[2] 16#23);
```

Example when the data type of "OutEle" is 1 byte:

Example with "In" as the 1st element of a SINT-type array, Size =UINT#3, Order =_LOW_HIGH, "OutEle" as the 1st element of the "OutVal" array:

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_LOW_HIGH	
Size	UINT	16#0003	
AryIn	ARRAY [1..4] OF SINT		
AryIn[1]	SINT	16#11	
AryIn[2]	SINT	16#22	
AryIn[3]	SINT	16#33	
AryIn[4]	SINT	16#00	
AryOut	ARRAY [1..32] OF BYTE		
AryOut[1]	BYTE	16#11	
AryOut[2]	BYTE	16#22	
AryOut[3]	BYTE	16#33	
AryOut[4]	BYTE	16#00	
AryOut[5]	BYTE	16#00	

```
ToAryByte(In:= AryIn[1] 16#11, uiSIZE:= Size 16#0003, Order:= odertest _LOW_HIGH,
          OutVal:= AryOut, OutEle:= AryOut[1] 16#11);
```

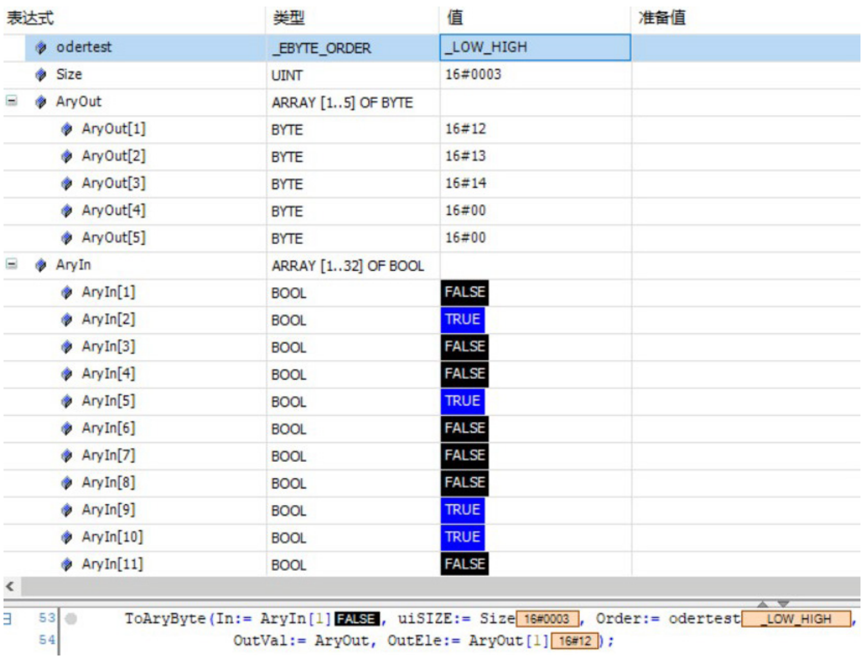
Other conditions same as above, Order =_HIGH_LOW, example:

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_HIGH_LOW	
Size	UINT	16#0003	
AryIn	ARRAY [1..4] OF SINT		
AryIn[1]	SINT	16#11	
AryIn[2]	SINT	16#22	
AryIn[3]	SINT	16#33	
AryIn[4]	SINT	16#00	
AryOut	ARRAY [1..32] OF BYTE		
AryOut[1]	BYTE	16#22	
AryOut[2]	BYTE	16#11	
AryOut[3]	BYTE	16#00	
AryOut[4]	BYTE	16#33	
AryOut[5]	BYTE	16#00	

```
ToAryByte(In:= AryIn[1] 16#11, uiSIZE:= Size 16#0003, Order:= odertest _HIGH_LOW,
          OutVal:= AryOut, OutEle:= AryOut[1] 16#22);
```

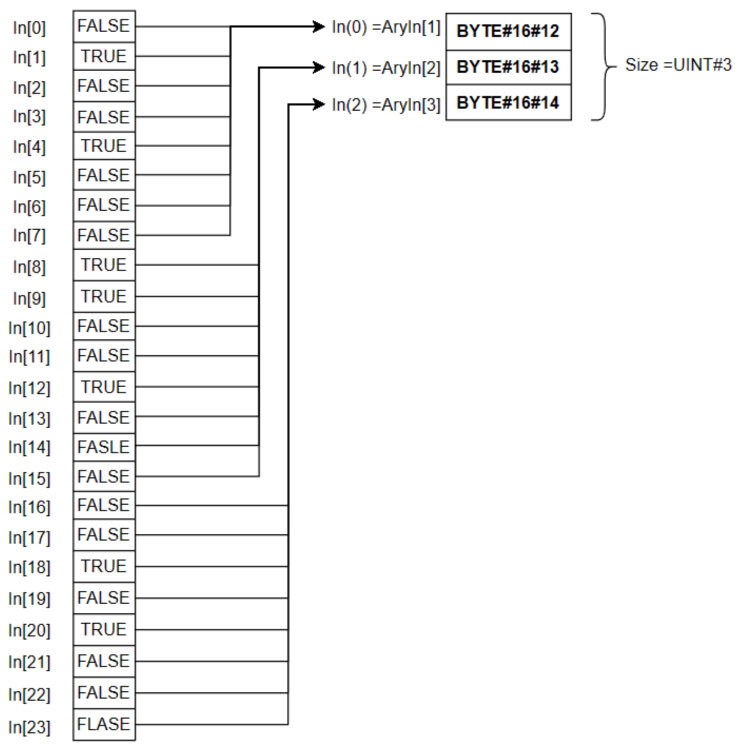
Example when the data type of "OutEle" is BOOL:

Example with "OutVal" as BOOL-type array, Size =UINT#3, "In" as the 1st element of a BYTE array, Order =_LOW_HIGH, "OutEle" as the 1st element of the "OutVal" array:

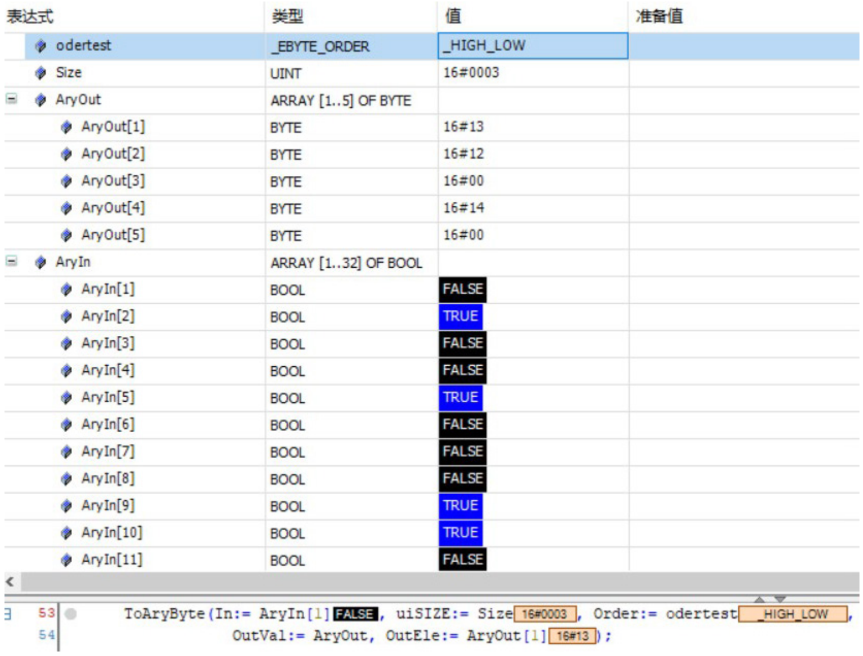


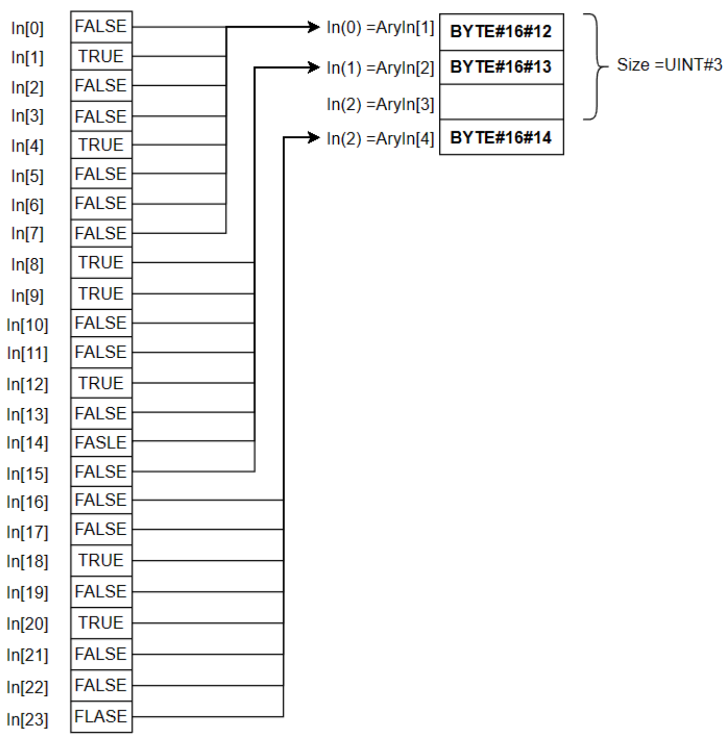
表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_LOW_HIGH	
Size	UINT	16#0003	
AryOut	ARRAY [1..5] OF BYTE		
AryOut[1]	BYTE	16#12	
AryOut[2]	BYTE	16#13	
AryOut[3]	BYTE	16#14	
AryOut[4]	BYTE	16#00	
AryOut[5]	BYTE	16#00	
AryIn	ARRAY [1..32] OF BOOL		
AryIn[1]	BOOL	FALSE	
AryIn[2]	BOOL	TRUE	
AryIn[3]	BOOL	FALSE	
AryIn[4]	BOOL	FALSE	
AryIn[5]	BOOL	TRUE	
AryIn[6]	BOOL	FALSE	
AryIn[7]	BOOL	FALSE	
AryIn[8]	BOOL	FALSE	
AryIn[9]	BOOL	TRUE	
AryIn[10]	BOOL	TRUE	
AryIn[11]	BOOL	FALSE	

```
ToAryByte(In:= AryIn[1] FALSE, uiSIZE:= Size 16#0003, Order:= odertest _LOW_HIGH,
          OutVal:= AryOut, OutEle:= AryOut[1] 16#12);
```



Other conditions same as above, Order =_HIGH_LOW, example:





◆ Key points

- For "In", input the starting array element for the operation.
- For "OutVal", specify a BYTE array. For "OutEle", specify the starting array element for the operation.
- If the output array has insufficient memory for the conversion, the function returns status FALSE and performs no action.

5.10 FSC instructions

5.10.1 StringSum (SUM value calculation)

The instruction calculates the sum value of a string.

Instruction	Name	FB/FUN	Graphical representation	ST representation
StringSum	Sum value calculation	FUN		Out:=StringSum(In, Size);

◆ Variables

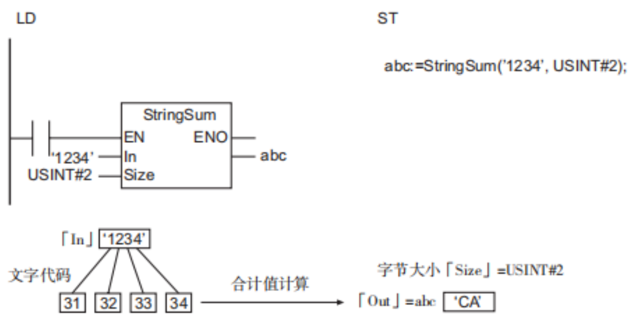
	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Source string	Input	Source string.	Conforms to data type.	—	‘’
Size	Byte size		Byte size for the sum value.	1,2	BYTE	1
Out	Sum value	Output	Sum value.	Byte count as indicated by "Size".	BYTE	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Size						○														
Out																				○

◆ Function

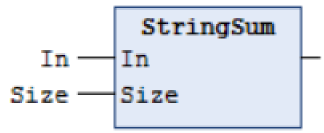
It calculates the sum value of the source string "In" (the sum of the character codes for each character). The sum value "Out" has the number of bytes specified by the byte size "Size". "Out" is expressed as a hexadecimal string, terminated with a NULL character.

An example with "In"='1234' and "Size"=USINT#2 is shown below.



In the example above, if "Size"=USINT#1, the value of "Out" is 'A'.

	FBD	ST
Variable declaration	VAR In :STRING; Size :USINT; Out :STRING; END_VAR	

Program		Out:=StringSum(In:=In , Size:=Size);
Result	In STRING '1234' Size USINT 2 Out STRING 'CA'	

◆ Key points

・When the sum of the character codes of "In" exceeds the number of bits representable by "Size", the higher-order bits are discarded.

・ An exception occurs and ENO becomes FALSE, "Out" remains unchanged under the following conditions:

・When the value of "Size" exceeds its valid range.

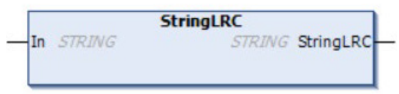
・When "In" is not NULL-terminated.

・When the byte count of "In" is 0 (contains only a NULL character).

・When the size of the processed result string exceeds the size of "Out".

5.10.2 StringLRC (LRC value calculation <string>)

The instruction calculates the LRC value (Longitudinal Redundancy Check) of a string.

Instruction	Name	FB/FUN	Graphical representation	ST representation
StringLRC	LRC value calculation (String)	FUN		Out:=StringLRC(In);

◆ Variables

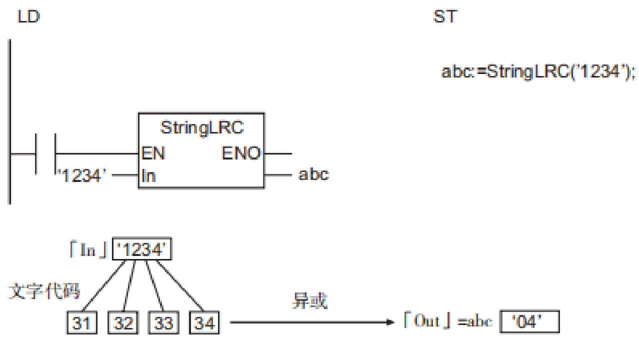
	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Source string	Input	Source string	Conforms to data type.	—	''
Out	LRC value	Output	LRC value	Maximum 3 bytes (2 half-width alphanumeric characters + terminating NULL character).	—	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out																				○

◆ Function

It calculates the LRC value (Longitudinal Redundancy Check) of the source string "In". The LRC value is the XOR of the character codes of each character in "In". The LRC value "Out" is expressed as a hexadecimal string, terminated with a NULL character.

An example with "In"='1234' is shown below.



	FBD	ST
Variable declaration	VAR In :STRING; Out :STRING; END_VAR	
Program	StringLRC In — In	Out:=StringLRC(In:=In);
Result	In STRING '1234' Out STRING '04'	

◆ Key points

- An exception occurs and ENO becomes FALSE, "Out" remains unchanged under the following conditions:
- When "In" is not NULL-terminated.
- When the byte count of "In" is 0 (contains only a NULL character).
- When the byte count of "Out" exceeds its valid range.

5.10.3 CRC16 (CRC16 general function block <string>)

The instruction calculates the CRC16 value (Cyclic Redundancy Check) of a string.

Instruction	Name	FB/FUN	Graphical representation	ST representation
CRC16	CRC value calculation (String)	FUN	CRC16 In STRING — STRING CRC16 CrcMode _eCRC16_MODE	Out:=CRC16(In, CrcMode);

◆ Variables

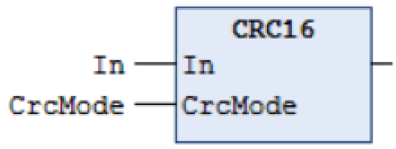
	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Source string	Input	Source string	Conforms to data type.	—	''
CrcMode	CRC model	Input	_eCRC16_MODE	—	—	_CRC16_ CCITT
Out	CRC value	Output	CRC value	Maximum 5 bytes (4 half-width alphanumeric characters + terminating NULL character).	—	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
CrcMode	Enumeration _eCRC16_MODE																			
Out																				○

◆ Function

It calculates the CRC value (Cyclic Redundancy Check) of the source string "In". The CRC value is the checksum obtained by checking each character of "In" through a CRC model. The CRC value "Out" is expressed as a hexadecimal string, terminated with a NULL character.

An example with "In"= '12' and "CrcMode" = _CRC16_MODBUS is shown below.

	FBD	ST
Variable declaration	VAR In :STRING; CrcMode :_eCRC16_MODE; Out :STRING; END_VAR	
Program		Out:=CRC16(In:=In , CrcMode:=CrcMode);
Result	In STRING '12' CrcMode _ECRC16_MODE _CRC16_MODBUS Out STRING 'F595'	

Enumeration element	Meaning
_CRC16_CCITT	CCITT check
_CRC16_CCITT_FALSE	CCITT_FALSE check
_CRC16_XMODEM	XMODEM check
_CRC16_X25	X25 check
_CRC16_MODBUS	MODBUS check
_CRC16_IBM	IBM check
_CRC16_MAXIM	MAXIM check
_CRC16_USB	USB check

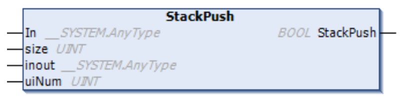
◆ Key points

- An exception occurs and ENO becomes FALSE, "Out" remains unchanged under the following conditions:
- When "In" is not NULL-terminated.
- When the byte count of "In" is 0 (contains only a NULL character).

5.11 Stack/Table instructions

5.11.1 StackPush (Save stack data)

The instruction saves a value to a stack.

Instruction	Name	FB/FUN	Graphical representation	ST representation
StackPush	Save stack data	FUN		StackPush(In, InOut, Size, Num);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Input value	Input	Value, structure, or a single structure element to be input to the stack.	Conforms to data type.	—	''
Size	Stack element count		Number of array elements for the stack.			1
InOut[ ] array	Stack array		Array constituting the stack.	Conforms to data type.	—	—
uiNum	Saved element count	Input/Output	Number of elements saved in the stack.			
Out	Return value	Output	Always TRUE.	TRUE only.	—	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Enumerated types, entire structures, or a single structure element can also be specified.																			
Size							○													
InOut[ ] array	An array with elements of the same data type as "In".																			
uiNum							○													
Out	○																			

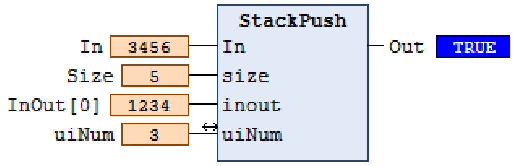
◆ Function

It is assumed that "Num" elements are currently saved in the stack array InOut[]. The input value "In" overwrites the next element InOut["Num"]. Then, "Num" is incremented. The stack element count "Size" specifies the number of elements in InOut[] to be used as the stack.

An example with "Size"=UINT#5 and "Num"=UINT#2 is shown below.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In :INT; Size :UINT :=5; InOut :ARRAY[0..6] OF INT := [1234, 2345, 5(0)]; uiNum :UINT :=2; Out :BOOL; check :BOOL; END_VAR	

Program




```
IF checkFALSE THEN
    OutTRUE:=StackPush(
        In:= In 3456,
        size:= Size 5,
        inout:= InOut[0] 1234,
        uiNum:=uiNum 3 );
    checkFALSE:=FALSE;
END_IFRETURN
```

Result

表达式	类型	值
In	INT	3456
Size	UINT	5
InOut	ARRAY [0..6] O...	
InOut[0]	INT	1234
InOut[1]	INT	2345
InOut[2]	INT	3456
InOut[3]	INT	0
InOut[4]	INT	0
InOut[5]	INT	0
InOut[6]	INT	0
uiNum	UINT	3
Out	BOOL	TRUE

◆ Reference

To extract the lowest or highest value from the stack, use the "StackFIFO/StackLIFO instruction".

◆ Key points

- Set the data types of "In" and the elements of InOut[] to be identical.
- When transferring an element from an array to InOut[], the element's subsequent data is the processing target.
- When the value of "Size" is 0, the values of InOut[] and "Num" remain unchanged.
- Always use a variable as the input parameter for "In". An exception occurs during linking if a constant is passed.
- When "In" is an enumerated type, enumeration elements cannot be passed directly. An exception occurs during linking if passed directly.
- The return value "Out" is not used when employing this instruction in an ST program.
- An exception occurs and ENO becomes FALSE, and InOut[] remains unchanged under the following conditions:
- When the data types of "In" and the elements of InOut[] differ.
- When the value of "Size" is not 0 and "Num" ≥ "Size".
- When the value of "Size" exceeds the array bounds of InOut[].
- When "In" is of STRING type and is not NULL-terminated.
- When "In" and InOut[] are of STRING type, and the byte count of "In" exceeds the size of InOut[].

5.11.2 StackFIFO/StackLIFO (First-In-First-Out / Last-In-First-Out)

StackFIFO: Extracts the value at the lowest position of the stack (first-in).

StackLIFO: Extracts the value at the highest position of the stack (last-in).

Instruction	Name	FB/FUN	Graphical representation	ST representation
StackFIFO	First-In-First-Out	FUN		StackFIFO(InOut, OutVal, Size, Num);
StackLIFO	Last-In-First-Out	FUN		StackLIFO(InOut, OutVal, Size, Num);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Size	Stack element count	Input	Number of stack elements.	Conforms to data type.	—	1
InOut[ ] array	Stack array		Array constituting the stack.			—
OutVal	Output value		Value or entire structure output from the stack.	Conforms to data type.	—	—
Num	Saved element count	Input/Output	Number of elements saved in the stack.			
Out	Return value	Output	Always TRUE.	TRUE only.	—	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							○													
InOut[ ] array	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Arrays of enumerated types or arrays of structures can also be specified.																			
OutVal	Same data type as the elements of InOut[].																			
Num							○													
Out	○																			

◆ Function

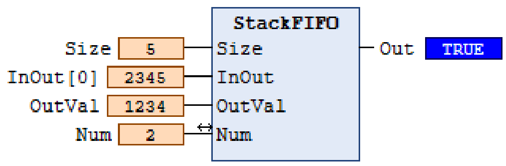
It is assumed that "Num" elements are currently saved in the stack array InOut[]. A value is extracted from it and assigned to the output value "OutVal". The stack element count "Size" specifies the number of elements in InOut[] to be used as the stack.

StackFIFO: Extracts the value at the lowest position of the stack. The value of InOut[0] is assigned to "OutVal". Then, the "Num"-1 elements from InOut[1] onward are each shifted one position towards the lower indices of the stack array. Finally, "Num" is decremented.

An example with "Size"=UINT#5 and "Num"=UINT#3 is shown below.

	FBD	ST
Variable declaration		

```
PROGRAM PLC_PRG
VAR
    Size     :UINT :=5;
    InOut    :ARRAY[0..6] OF INT := [1234, 2345,3456,4(0)];
    OutVal   :INT;
    Num      :UINT :=3;
    Out      :BOOL;
    check    :BOOL;
END_VAR
```

Program		IF check FALSE THEN
		Out TRUE :=StackFIFO(
		Size:= Size 5 ,
		InOut:= InOut[0] 2345 ,
		OutVal:= OutVal 1234 ,
		Num:= Num 2);
		check FALSE :=FALSE;
		END_IF RETURN

Result

表达式	类型	值
Size	UINT	5
InOut	ARRAY [0..6] O...	
InOut[0]	INT	2345
InOut[1]	INT	3456
InOut[2]	INT	0
InOut[3]	INT	0
InOut[4]	INT	0
InOut[5]	INT	0
InOut[6]	INT	0
OutVal	INT	1234
Num	UINT	2
Out	BOOL	TRUE

StackLIFO: Extracts the value at the highest position of the stack. The value of InOut["Num"-1] is assigned to "OutVal". "Num" is decremented.

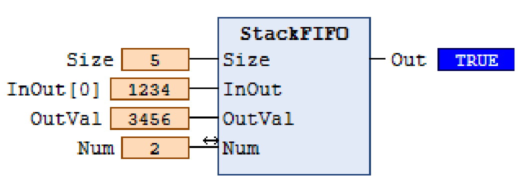
An example with "Size"=UINT#5 and "Num"=UINT#3 is shown below.

	FBD	ST

Variable declaration

```
PROGRAM PLC_PRG
VAR
    Size    :UINT :=5;
    InOut   :ARRAY[0..6] OF INT := [1234, 2345,3456,4(0)];
    OutVal  :INT;
    Num     :UINT :=3;
    Out     :BOOL;
    check   :BOOL;
END_VAR
```

Program




```
IF check FALSE THEN
    Out TRUE :=StackLIFO(
        Size:= Size 5 ,
        InOut:= InOut[0] 1234 ,
        OutVal:= OutVal 3456 ,
        Num:= Num 2 );
    check FALSE :=FALSE;
END_IF RETURN
```

Result	(see table below)

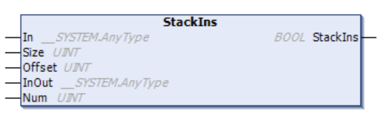
表达式	类型	值
Size	UINT	5
InOut	ARRAY [0..6] O...	
InOut[0]	INT	1234
InOut[1]	INT	2345
InOut[2]	INT	0
InOut[3]	INT	0
InOut[4]	INT	0
InOut[5]	INT	0
InOut[6]	INT	0
OutVal	INT	3456
Num	UINT	2
Out	BOOL	TRUE

◆ Key points

- Set the data types of "In" and the elements of InOut[] to be identical.
- When transferring an element from an array to InOut[], the element's subsequent data is the processing target.
- When the value of "Size" is 0, the values of InOut[] and "Num" remain unchanged.
- The return value "Out" is not used when employing this instruction in an ST program.
- An exception occurs and ENO becomes FALSE, and InOut[] remains unchanged under the following conditions:
- When the data types of the elements of InOut[] and "OutVal" differ.
- When the values of "Num" and "Size" are not 0, and "Num" > "Size".
- When the value of "Size" exceeds the array bounds of InOut[].
- When InOut[] is a STRING-type array and all its elements are not NULL-terminated.
- When InOut[] is a STRING-type array, and the byte count of an element exceeds the size of "OutVal".

5.11.3 StackIns (Insert stack data)

The instruction inserts a value at an arbitrary position within a stack.

Instruction	Name	FB/FUN	Graphical representation	ST representation
StackIns	Insert stack data	FUN		StackIns(In, InOut, Size, Num, Offset);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Insert value	Input	Value, entire structure, or a single structure element to insert into the stack.	Conforms to data type.	—	(*)
Size	Stack element count		Number of stack elements.			1
OffSet	Offset position		Offset position in the stack for inserting "In".			0
InOut[ ] array	Stack array		Array constituting the stack.	Conforms to data type.	—	—
Num	Saved element count	Input/Output	Number of elements saved in the stack.	Conforms to data type.	—	—
Out	Return value	Output	Always TRUE.	TRUE only.	—	—

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Enumerated types, entire structures, or a single structure element can also be specified.																			
Size							○													
OffSet							○													
InOut[ ] array	An array with elements of the same data type as "In".																			
Num							○													
Out	○																			

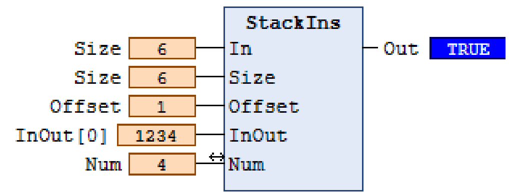
◆ Function

It is assumed that "Num" elements are currently saved in the stack array InOut[]. The insert value "In" is inserted at the position InOut["Offset"] determined by the offset "Offset". The elements at the same or higher indices, i.e., InOut["Offset"] to InOut["Num"-1], are each shifted one position towards the higher indices of the stack array. Then, "Num" is incremented. The stack element count "Size" specifies the number of elements in InOut[] to be used as the stack.

An example with "Size"=UINT#6, "Num"=UINT#3, and "Offset"=UINT#1 is shown below.

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2"><pre>PROGRAM PLC_PRG
VAR
    In   :INT;
    Size   :UINT :=6;
    Offset :UINT :=1;
    InOut   :ARRAY[0..6] OF INT := [1234, 3456,4567,4(0)];
    Num   :UINT :=3;
    Out   :BOOL;
    check   :BOOL;
END_VAR</pre></td></tr>
</table>

Program




```
IF check FALSE THEN
    Out TRUE :=StackIns(
        In:= In 2345 ,
        Size:= Size 6 ,
        Offset:= Offset 1 ,
        InOut:= InOut[0] 1234 ,
        Num:= Num 4 );
        check FALSE :=FALSE;
END_IF RETURN
```

Result

表达式	类型	值
In	INT	2345
Size	UINT	6
Offset	UINT	1
InOut	ARRAY [0..6] O...	
InOut[0]	INT	1234
InOut[1]	INT	2345
InOut[2]	INT	3456
InOut[3]	INT	4567
InOut[4]	INT	0
InOut[5]	INT	0
InOut[6]	INT	0
Num	UINT	4
Out	BOOL	TRUE

◆ Key points

- Set the data types of "In" and InOut[] to be identical.
- When transferring an element from an array to InOut[], the element's subsequent data is the processing target.
- When the value of "Size" is 0, the values of InOut[] and "Num" remain unchanged.
- Always use a variable as the input parameter for "In". An exception occurs during linking if a constant is passed.
- When "In" is an enumerated type, enumeration elements cannot be passed directly. An exception occurs during linking if passed directly.
- The return value "Out" is not used when employing this instruction in an ST program.
- An exception occurs and ENO becomes FALSE, and InOut[] remains unchanged under the following conditions:
- When the data types of "In" and the elements of InOut[] differ.
- When the value of "Size" is not 0, and the condition "Size" > "Num" ≥ "Offset" is not satisfied.
- When the value of "Size" exceeds the array bounds of InOut[].
- When "In" is of STRING type and is not NULL-terminated.
- When "In" and InOut[] are of STRING type, and the byte count of "In" exceeds the size of InOut[].

5.11.4 StackDel (Delete stack data)

The instruction deletes a value at an arbitrary position within a stack.

Instruction	Name	FB/FUN	Graphical representation	ST representation
StackDel	Delete stack data	FUN		StackDel(InOut, Size, Num, Offset);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Size	Stack element count	Input	Number of stack elements.	Conforms to data type.	—	1
OffSet	Offset position		Offset position in the stack for deletion.			0
InOut[ ] array	Stack array		Array constituting the stack.	Conforms to data type.	—	—
Num	Saved element count	Input/Output	Number of elements saved in the stack.	Conforms to data type.	—	—
Out	Return value	Output	Always TRUE.	TRUE only.	—	—

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							○													
OffSet							○													
InOut[ ] array	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Arrays of enumerated types or arrays of structures can also be specified.																			
Num							○													
Out	○																			

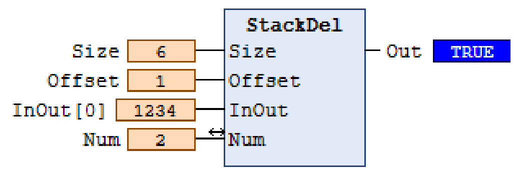
◆ Function

It is assumed that "Num" elements are currently saved in the stack array InOut[]. The value at the position InOut["Offset"], determined by the offset "Offset", is deleted. The elements at higher indices, i.e., InOut["Offset"+1] to InOut["Num"-1], are each shifted one position towards the lower indices of the stack array. Then, "Num" is decremented. The stack element count "Size" specifies the number of elements in InOut[] to be used as the stack.

An example with "Size"=UINT#6, "Num"=UINT#3, and "Offset"=UINT#1 is shown below (see example for initial InOut array values).

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR Size :UINT :=6; Offset :UINT :=1; InOut :ARRAY[0..6] OF INT := [1234, 2345,3456,4(0)]; Num :UINT :=3; Out :BOOL; check :BOOL; END_VAR	

Program




```
IF checkFALSE THEN
    OutTRUE:=StackDel(
        Size:= Size 6,
        Offset:= Offset 1,
        InOut:= InOut[0] 1234,
        Num:= Num 2);
        checkFALSE:=FALSE;
END_IFRETURN
```

Result

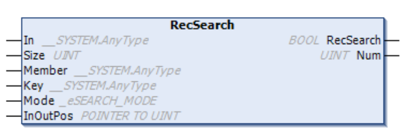
表达式	类型	值
Size	UINT	6
Offset	UINT	1
InOut	ARRAY [0..6] O...	
InOut[0]	INT	1234
InOut[1]	INT	3456
InOut[2]	INT	0
InOut[3]	INT	0
InOut[4]	INT	0
InOut[5]	INT	0
InOut[6]	INT	0
Num	UINT	2
Out	BOOL	TRUE

◆ Key points

- When transferring an element from an array to InOut[], the element's subsequent data is the processing target.
- When the value of "Size" or "Num" is 0, the values of InOut[] and "Num" remain unchanged.
- The return value "Out" is not used when employing this instruction in an ST program.
- An exception occurs and ENO becomes FALSE, and InOut[] remains unchanged under the following conditions:
- When the values of "Num" and "Size" are not 0, and the condition "Size" ≥ "Num" > "Offset" is not satisfied.

5.11.5 RecSearch (Record search)

The instruction searches for elements matching a specified search key within an array of structures, using a designated method.

Instruction	Name	FB/FUN	Graphical representation	ST representation
RecSearch	Record search	FUN		Out:=RecSearch(In, Size, Member, Key, Mode, InOutPos, Num);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In[ ] array	Search array	Input	Array of structures to be searched.	—	—	(*)
Size	Number of elements		Number of array elements to search.	Conforms to data type.		1
Member	Member to search		Member of the structure in In[] to be searched.			(*)
Key	Search key		Search value.			
Mode	Search method		Search method.	_LINEAR, _BIN_ASC, _BIN_DESC		_LINEAR
InOutPos[ ] array	Matching element index		Index of the matching element.	Conforms to data type.	—	—
Out	Search result	Input/Output	TRUE: Matching element found. FALSE: No matching element found.	Conforms to data type.	—	—
Num	Number of matching elements	Output	Number of matching elements.		—	—

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ] array	An array of structures is specified.																			
Size							○													
Member						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Same data type as the "Member to Search" in In[].																			
Key	Same data type as "Member".																			
Mode	Enumeration _eSEARCH_MODE. See the Function section for enumeration elements.																			
InOutPos[ ] array							○													
Out	○																			
Num							○													

◆ Function

Within the first "Size" elements (In[0] to In["Size"-1]) of the array of structures In[], searches for elements where the value of the specified structure member "Member" matches the search key "Key". Pass the member of any element in In[] as the argument for "Member". If a matching element exists, the search result "Out" becomes TRUE. The index of the matching element and the number of matches are assigned to InOutPos[0] and "Num" respectively. If there are two or more matches, the index of the lowest matching element in In[] is assigned to InOutPos[0]. If no matching element exists, "Out" becomes FALSE, and InOutPos[0] and "Num" become 0. When passing the input parameter to In[], always include an element index, e.g., array[3].

The search method "Mode" is of the enumeration type _eSEARCH_MODE. The meanings of the enumeration elements are as follows.

Enumeration element	Meaning
_LINEAR	Linear search
_BIN_ASC	Ascending binary search
_BIN_DESC	Descending binary search

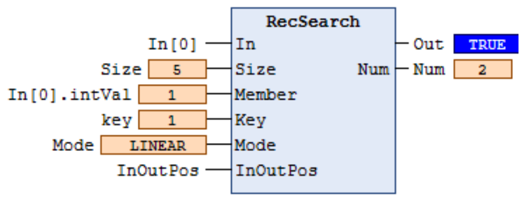
In linear search, the search is performed sequentially from the first element of In[].

An example with "Size"=UINT#5, "Key"=INT#1, "Mode"=_LINEAR is shown below. The example shows that the Int member values of In[0] and In[3] match "Key".

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">

```
PROGRAM PLC_PRG
VAR
    In    :ARRAY[0..4] OF str:= [(intVal := 1, realVal := 2.3),
                                  (intVal := 5, realVal := 536),
                                  (intVal := 2, realVal := 5.6),
                                  (intVal := 1, realVal := 6.3),
                                  (intVal := 8, realVal := 55)];
    Size      :UINT :=5;
    key       :INT:=1;
    Mode      :HCFA_OmronUitls._eSEARCH_MODE;
    InOutPos:ARRAY [0..4] OF UINT;
    Num       :UINT;
    Out       :BOOL;
    check     :BOOL;
END_VAR
```

</td></tr>
<tr><td>Program</td><td>



</td><td>

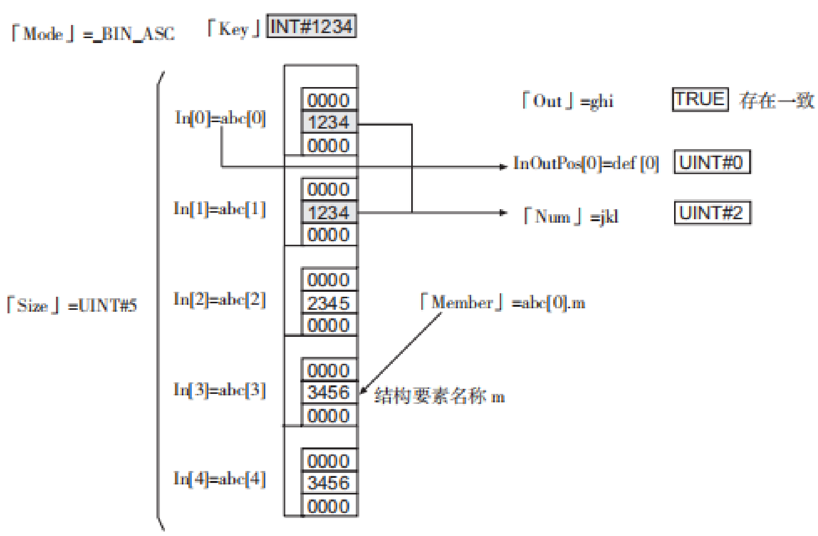
```
IF check FALSE THEN
    Out TRUE :=RecSearch(
        In:= In[0],
        Size:= Size 5 ,
        Member:= In[0].intVal 1 ,
        Key:= key 1 ,
        Mode:= Mode _LINEAR ,
        InOutPos:= InOutPos,
        Num=> Num 2 );
        check FALSE :=FALSE;
END_IF
```

</td></tr>
</table>

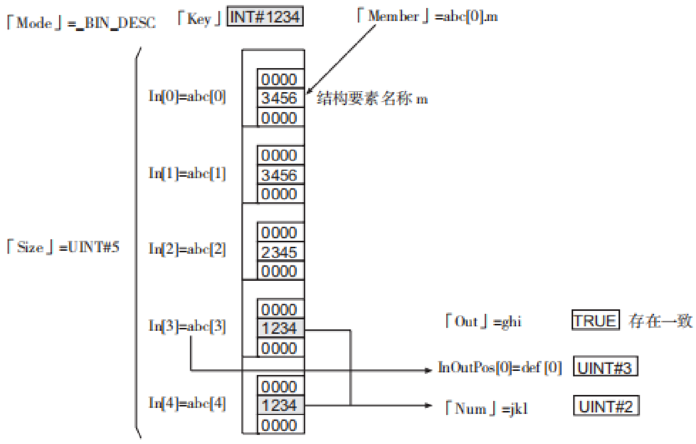
Result

表达式	类型	值
In	ARRAY [0..4] O...	
In[0]	str	
intVal	INT	1
realVal	REAL	2.3
In[1]	str	
intVal	INT	5
realVal	REAL	536
In[2]	str	
intVal	INT	2
realVal	REAL	5.6
In[3]	str	
intVal	INT	1
realVal	REAL	6.3
In[4]	str	
intVal	INT	8
realVal	REAL	55
Size	UINT	5
key	INT	1
Mode	_ESEARCH_MODE	_LINEAR
InOutPos	ARRAY [0..4] O...	
InOutPos[0]	UINT	0
InOutPos[1]	UINT	3
InOutPos[2]	UINT	0
InOutPos[3]	UINT	0
InOutPos[4]	UINT	0
Num	UINT	2
Out	BOOL	TRUE

For ascending binary search, the array elements passed as the input parameter for In[] must be sorted in ascending order before executing this instruction. The instruction then performs a binary search. The element order and result for the same example after applying ascending binary search are shown below.



For descending binary search, the array elements passed as the input parameter for In[] must be sorted in descending order before executing this instruction. The instruction then performs a binary search. The element order and result for the same example after applying descending binary search are shown below.

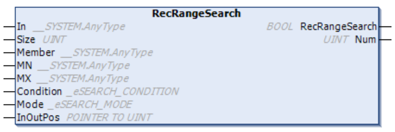


◆ Key points

- When transferring an element from an array to In[], the element's subsequent data is the processing target.
- When "Member" is a real number, results may be unexpected due to rounding errors inherent to the value.
- When "Key" is a real number, do not specify it as Not-a-Number (NaN).
- When the value of "Size" is 0, "Out" is FALSE, "Num" is 0, and InOutPos[] remains unchanged.
- When "Mode" is _BIN_ASC (or _BIN_DESC), correct results cannot be obtained if the elements of In[] are not sorted in ascending (or descending) order. Sort the elements in ascending (or descending) order before executing this instruction.
- An exception occurs and ENO becomes FALSE, and "Out", InOutPos[], and "Num" remain unchanged under the following conditions:
  - When the value of "Size" exceeds the array bounds of In[].
  - When "Member" is not a member of the structures in In[].
  - When the data type is not supported for "Member".
  - When the data types of "Key" and "Member" differ.
  - When In[] is not an array of structures.
  - When "Member" or "Key" is of STRING type and is not NULL-terminated.

5.11.6 RecRangeSearch (Range-specified record search)

The instruction searches for elements matching a specified range condition within an array of structures, using a designated method.

Instruction	Name	FB/FUN	Graphical representation	ST representation
RecRange Search	Range-specified record search	FUN		Out:=RecRangeSearch(In, Size, Member, MN, MX, Condition, Mode, InOutPos, Num);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In[ ] array	Search array	Input	Array of structures to be searched.	—	—	(*)
Size	Number of elements		Number of array elements to search.	Conforms to data type.		1
Member	Member to search		Member of the structure in In[] to be searched.			(*)
MN	Lower bound		Upper bound for search condition.			
MX	Upper bound		Lower bound for search condition.			
Condition	Search condition		Search condition.	_EQ_BOTH, _EQ_MIN, _EQ_MAX, _NE_BOTH		_EQ_BOTH
Mode	Search method		Search method.	_LINEAR, _BIN_ASC, _BIN_DESC		_LINEAR
InOutPos[ ] array	Matching element index	Input/Output	Index of the matching element.	Conforms to data type.	—	—
Out	Search result	Output	TRUE: Matching element found. FALSE: No matching element found.	Conforms to data type.	—	—
Num	Number of matching elements		Number of matching elements.		—	—

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ] array	An array of structures is specified.																			
Size							○													
Member						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Same data type as the "Member to Search" in In[].																			
MN	Same data type as "Member".																			
MX	Same data type as "Member".																			
Condition	Enumeration _eSEARCH_CONDITION. See the Function section for enumeration elements.																			

Mode	Enumeration _eSEARCH_MODE. See the Function section for enumeration elements.																			
InOutPos[ ] array							○													
Out	○																			
Num							○													

◆ Function

Within the first "Size" elements (In[0] to In["Size"-1]) of the array of structures In[], searches for elements where the value of the specified structure member "Member" matches the search condition. The search condition and method are specified by "Condition" and "Mode", detailed below. Pass the member of any element in In[] as the argument for "Member". If an element matches the condition, the search result "Out" becomes TRUE. The index of the matching element and the number of matches are assigned to InOutPos[0] and "Num" respectively. If there are two or more matches, the index of the lowest matching element in In[] is assigned to InOutPos[0]. If no element matches, "Out" becomes FALSE, and InOutPos[0] and "Num" become 0. When passing the input parameter to In[], always include an element index, e.g., array[3].

The search condition "Condition" is of the enumeration type _eSEARCH_CONDITION. The meanings of the enumeration elements are as follows.

Enumeration element	Meaning
_EQ_BOTH	"MN"≤"Member"≤"MX"
_EQ_MIN	"MN"≤"Member" < "MX"
_EQ_MAX	"MN" < "Member"≤"MX"
_NE_BOTH	"MN" < "Member" < "MX"

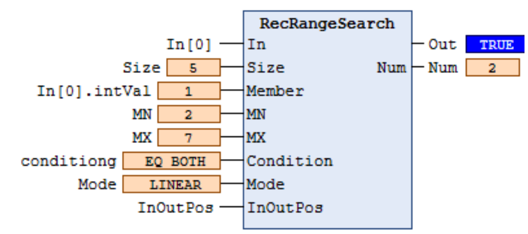
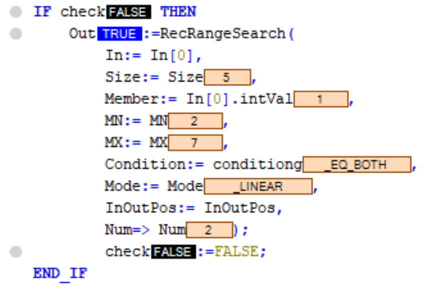
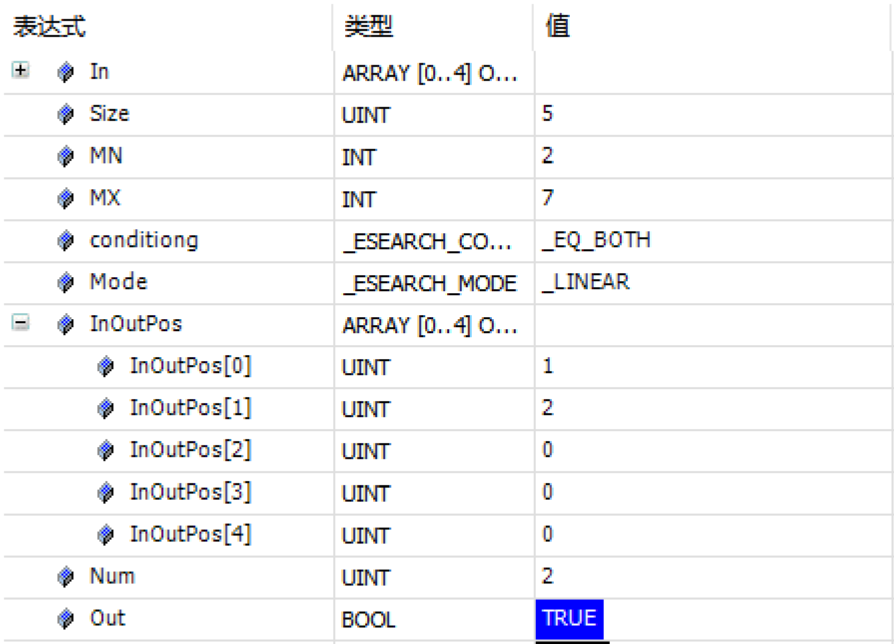
The search method "Mode" is of the enumeration type _eSEARCH_MODE. The meanings of the enumeration elements are as follows.

Enumeration element	Meaning
_LINEAR	Linear search
_BIN_ASC	Ascending binary search
_BIN_DESC	Descending binary search

In linear search, the search is performed sequentially from the first element of In[].

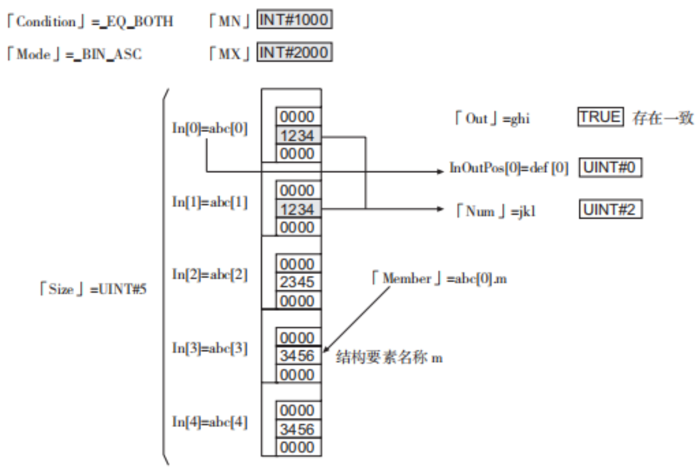
An example with "Size"=UINT#5, "MN"=INT#2, "MX"=INT#7, "Condition"=_EQ_BOTH, and "Mode"=_LINEAR is shown below.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In :ARRAY[0..4] OF str:= [(intVal := 1, realVal := 2.3), (intVal := 5, realVal := 536), (intVal := 2, realVal := 5.6), (intVal := 1, realVal := 6.3), (intVal := 8, realVal := 55)]; Size :UINT :=5; MN :INT :=2; MX :INT :=7; conditiong :HCFA_OmronUitls._eSEARCH_CONDITION; Mode :HCFA_OmronUitls._eSEARCH_MODE; InOutPos :ARRAY [0..4] OF UINT; Num :UINT; Out :BOOL; check :BOOL; END_VAR	

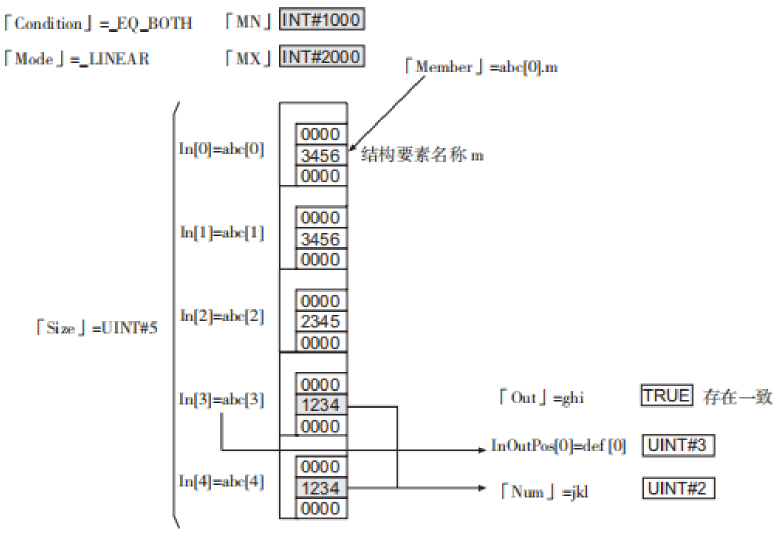
<table>
<tr><td>Program</td><td>
</td><td>
</td></tr>
<tr><td>Result</td><td colspan="2">
<table>
<tr><th>表达式</th><th>类型</th><th>值</th></tr>
<tr><td>In</td><td>ARRAY [0..4] O...</td><td></td></tr>
<tr><td>Size</td><td>UINT</td><td>5</td></tr>
<tr><td>MN</td><td>INT</td><td>2</td></tr>
<tr><td>MX</td><td>INT</td><td>7</td></tr>
<tr><td>conditiong</td><td>_ESEARCH_CO...</td><td>_EQ_BOTH</td></tr>
<tr><td>Mode</td><td>_ESEARCH_MODE</td><td>_LINEAR</td></tr>
<tr><td>InOutPos</td><td>ARRAY [0..4] O...</td><td></td></tr>
<tr><td>InOutPos[0]</td><td>UINT</td><td>1</td></tr>
<tr><td>InOutPos[1]</td><td>UINT</td><td>2</td></tr>
<tr><td>InOutPos[2]</td><td>UINT</td><td>0</td></tr>
<tr><td>InOutPos[3]</td><td>UINT</td><td>0</td></tr>
<tr><td>InOutPos[4]</td><td>UINT</td><td>0</td></tr>
<tr><td>Num</td><td>UINT</td><td>2</td></tr>
<tr><td>Out</td><td>BOOL</td><td>TRUE</td></tr>
</table></td></tr>
</table>

For ascending binary search, the array elements passed as the input parameter for In[] must be sorted in ascending order before executing this instruction. The instruction then performs a binary search.

The element order and result for the same example after applying ascending binary search are shown below.



For descending binary search, the array elements passed as the input parameter for In[] must be sorted in descending order before executing this instruction. The instruction then performs a binary search. The element order and result for the same example after applying descending binary search are shown below.



◆ Key points

- Set the data types of "Member", "MN", and "MX" to be identical to the data type of the searched structure member in In[].
- When transferring an element from an array to In[], the element's subsequent data is the processing target.
- When "Member" is a real number, results may be unexpected due to rounding errors inherent to the value.
- When "MN" or "MX" are real numbers, do not specify them as Not-a-Number (NaN).
- When the value of "Size" is 0, "Out" is FALSE, "Num" is 0, and InOutPos[] remains unchanged.
- When "Mode" is _BIN_ASC (or _BIN_DESC), correct results cannot be obtained if the elements of In[] are not sorted in ascending (or descending) order. Sort the elements in ascending (or descending) order before executing this instruction.
- An exception occurs and ENO becomes FALSE, and "Out", InOutPos[], and "Num" remain unchanged under the following conditions:
- When the data types of the searched structure member "Member" in In[], "MN", and "MX" differ.
- When "MN" > "MX".
- When the value of "Mode" is outside its valid range.
- When "Member" is not a member of the structures in In[].
- When the data type is not supported for "Member".
- When In[] is not an array of structures.
- When "Member", "MN", or "MX" is of STRING type and is not NULL-terminated.

5.11.7 RecSort (Record sort)

The instruction sorts the elements of an array of structures.

Instruction	Name	FB/FUN	Graphical representation	ST representation
RecSort	Record sort	FUN		RecSort_instance(Execute, InOut, Size, Member, Order, Done, Busy, Error);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Size	Number of elements	Input	Number of elements to sort.	Conforms to data type.	—	1
Member	Member to sort		Member of the structure in InOut[] to be sorted.			(*)
Order	Sort order		Sort order.	_ASC, _DESC		_ASC
InOut[ ] array	Array to sort		Array of structures to be sorted.	—	—	—

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							○													
Member						○	○	○	○	○	○	○	○	○	○	○	○	○	○	
	Same data type as the "Member to sort" in InOut[].																			
Order	Enumeration _eSORT_ORDER. See the Function section for enumeration elements.																			
InOut[ ] array	An array of structures is specified.																			

◆ Function

When "Execute" is TRUE, sorts the first "Size" elements (InOut[0] to InOut["Size"-1]) of the array of structures InOut[] according to the value of the specified structure member "Member". The sort order is specified by "Order", detailed below. Pass the member of any element in InOut[] as the argument for "Member". When passing the input/output parameter to InOut[], always include an element index, e.g., array[3].

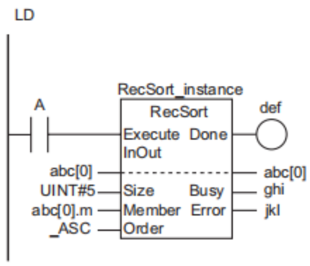
The sort order "Order" is of the enumeration type _eSORT_ORDER. The meanings of the enumeration elements are as follows.

Enumeration element	Meaning
_ASC	_Ascending
_DESC	_Descending

The magnitude relationship for data types other than integer and real numbers is determined as shown in the table below.

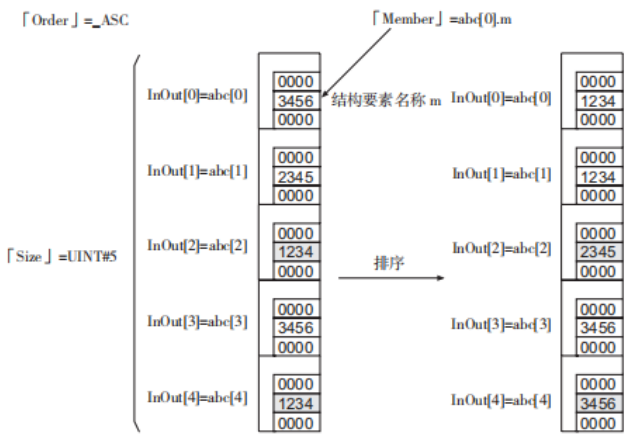
Data type	Meaning
TIME	A value with a larger magnitude is considered greater.
DATE, TOD, DT	For dates and time-of-day, the later one is considered greater.

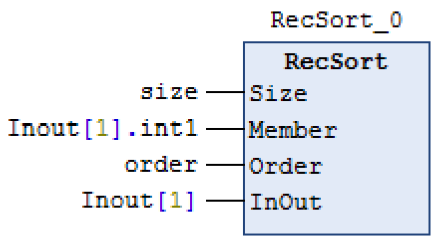
An example with "Size"=UINT#5 and "Order"=_ASC is shown below.



ST

RecSort_instance(A, abc[0], UINT#5, abc[0].m, _ASC, def, ghi, jkl);



	FBD	ST
Variable declaration	VAR size:UINT; order:_eSORT_ORDER; Inout:ARRAY [1..10] OF DUT; RecSort_0:RecSort; END_VAR	
Program		RecSort_0(Size:=size , Member:=Inout[1].intl , Order:=order , InOut:=Inout[1]);

Result

表达式	类型	值	准备值	地址	注
size	UINT	10			
order	_ESORT_ORDER	_ASC			
Inout	ARRAY [1..10] OF DUT				
Inout[1]	DUT				
real1	REAL	0			
int1	INT	0			
Inout[2]	DUT				
Inout[3]	DUT				
Inout[4]	DUT				
real1	REAL	0			
int1	INT	123			
Inout[5]	DUT				
real1	REAL	234			
int1	INT	222			
Inout[6]	DUT				
Inout[7]	DUT				
Inout[8]	DUT				
Inout[9]	DUT				
Inout[10]	DUT				
real1	REAL	0			
int1	INT	2342			

◆ Key points

・ Once this instruction starts execution, it continues processing to completion even if "Execute" becomes FALSE or the execution time exceeds the task period.

・ Confirm normal completion by checking if the value of "Done" becomes TRUE.

・ For the timing diagram of "Execute", "Done", "Busy", and "Error", refer to "Chapter Description (P.2-2)".

・ When "Member" is a real number, results may be unexpected due to rounding errors inherent to the value.

・ When transferring an element from an array to InOut[], the element's subsequent data is the processing target.

・ When the value of "Size" is 0, "Done" is TRUE, and InOut[] remains unchanged.

・ An exception occurs and "Error" becomes TRUE under the following conditions:

・ When the value of "Order" exceeds its valid range.

・ When the value of "Size" exceeds the array bounds of InOut[].

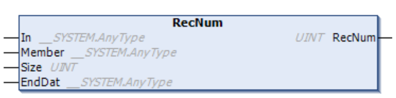
・ When "Member" is not a member of the structures in InOut[].

・ When the data type is not supported for "Member".

・ When InOut[] is not an array of structures.

5.11.8 RecNum (Get record count)

The instruction calculates the number of records (elements) in an array of structures up to a specified end-of-data marker.

Instruction	Name	FB/FUN	Graphical representation	ST representation
RecNum	Get record count	FUN		Out:=RecNum(In, Member, Size, EndDat);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In[ ] array	Source array	Input	Array of structures to be processed.	—	—	(*)
Member	Member		Member of the structure in In[] to be processed.	Conforms to data type.		
Size	Element count		Element count.			
EndDat	End-of-data marker		End-of-data marker.			
Out	Record count	Output	Record count.	Conforms to data type.	—	—

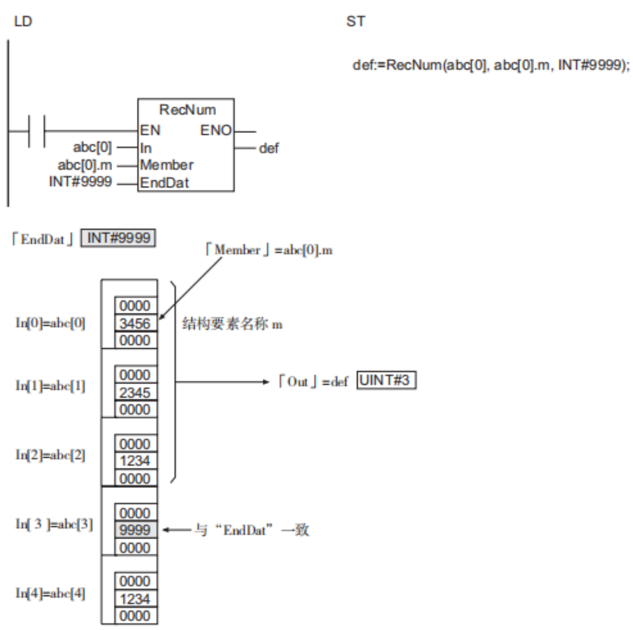
* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ] array	An array of structures is specified.																			
Member	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Enumerated types can also be specified.																			
	Same data type as the processed member in In[].																			
Size							○													
EndDat	Same data type as "Member".																			
Out							○													

◆ Function

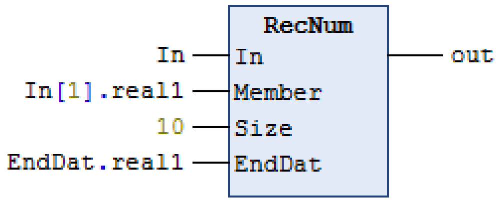
Starting from the beginning of the array of structures In[], searches for an element where the value of the specified member "Member" matches the end-of-data marker "EndDat". The number of elements before the matching element (the record count) is then assigned to "Out". Pass the member of any element in In[] as the argument for "Member". When passing the input parameter to In[], always include an element index, e.g., array[3].

An example with "EndDat"=INT#9999 is shown below.



<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">

```
VAR
    In:ARRAY [1..10] OF DUT;
    size:UINT;
    EndDat:DUT;
    out:UINT;
END_VAR
```

</td></tr>
<tr><td>Program</td><td>


</td><td>

```
out:=RecNum(
    In:=In[1] ,
    Member:=In[1].real1 ,
    Size:=size ,
    EndDat:=EndDat.real1 );
```

</td></tr>
<tr><td>Result</td><td colspan="2">

表达式	类型	值	准备值	地址	注
In[4]	DUT				
In[5]	DUT				
In[6]	DUT				
In[7]	DUT				
real1	REAL	12			
int1	INT	33			
In[8]	DUT				
real1	REAL	23			
int1	INT	56			
In[9]	DUT				
real1	REAL	0			
int1	INT	0			
In[10]	DUT				
real1	REAL	45			
int1	INT	33			
size	UINT	10			
EndDat	DUT				
real1	REAL	23			
int1	INT	0			
out	UINT	7			

</td></tr>
</table>

◆ Key points

- If no member in In[] matches "EndDat", the total number of elements in In[] is assigned to "Out".
- When "Member" is a real number, results may be unexpected due to rounding errors inherent to the value.
- When "EndDat" is a real number, do not specify it as Not-a-Number (NaN).
- When transferring an element from an array to In[], the element's subsequent data is the processing target.
- An exception occurs and ENO becomes FALSE, and "Out" remains unchanged under the following conditions:
- When "Member" is not a member of the structures in In[].
- When "Member" or "EndDat" is of STRING type and is not NULL-terminated.
- When the data type is not supported for "Member".
- When the data types of "Member" and "EndDat" differ.
- When In[] is not an array of structures.

5.11.9 RecMax/RecMin (Record maximum/minimum retrieval)

RecMax: Retrieves the maximum value of a specified member within an array of structures.

RecMin: Retrieves the minimum value of a specified member within an array of structures.

Instruction	Name	FB/FUN	Graphical representation	ST representation
RecMax	Record maximum retrieval	FUN		RecMax(In, Size,Member, Out, InOutPos, Num);
RecMin	Record minimum retrieval	FUN		RecMin(In, Size, Member, Out, InOutPos, Num);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In[ ] array	Source array	Input	Array of structures to be processed.	—	—	(*)
Size	Number of elements	Input	Number of elements to search.	Conforms to data type.	—	1
Member	Member	Input	Member of the structure in In[] to be processed.	Conforms to data type.	—	(*)
Out	Record count	Input	Retrieval result.	Conforms to data type.	—	—
InOutPos[ ] array	Result element index	Output	Index of the result element.	Conforms to data type.	—	—
Num	Number of results	Output	Number of retrieval results.	Conforms to data type.	—	0

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ] array	An array of structures is specified.																			
Size							○													
Member	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Same data type as the processed member in In[].																			
InOutPos[ ] array							○													
Out						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Num							○													

◆ Function

Within the first "Size" elements (In[0] to In["Size"-1]) of the array of structures In[], examines the values of the specified structure member "Member". Pass the member of any element in In[] as the argument for "Member". The index of the result element and the number of results are assigned to InOutPos[0] and "Num" respectively. If there are two or more results, the index of the lowest result element in In[] is assigned to InOutPos[0]. When passing the input parameter to In[], always include an element index, e.g., array[3].

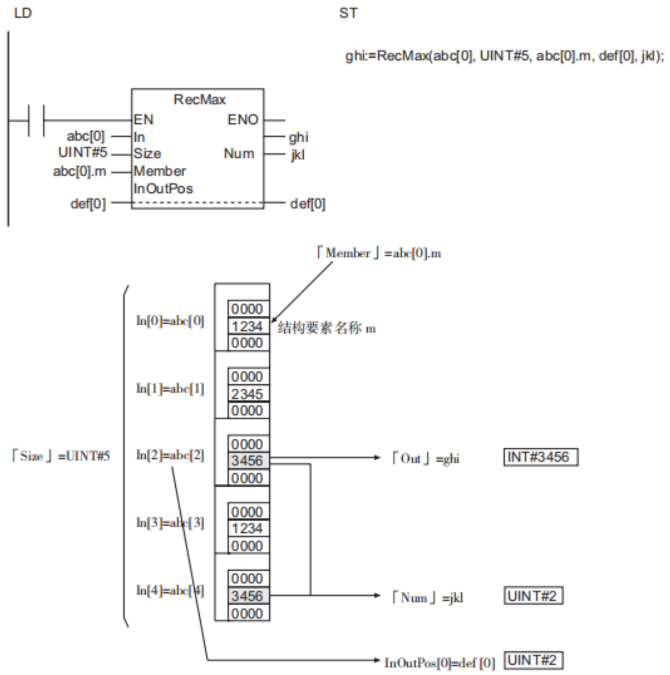
The magnitude relationship for data types other than integer and real numbers is determined as shown in the table below.

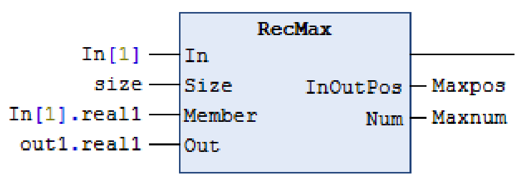
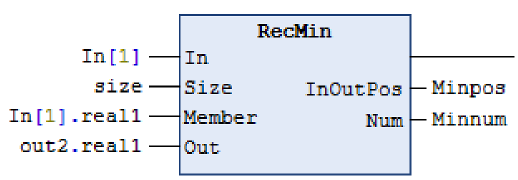
Data type	Meaning
TIME	A value with a larger magnitude is considered greater.
DATE, TOD, DT	For dates and time-of-day, the later one is considered greater.
STRING	Determined by comparing the ASCII codes of the first characters.

RecMax: Retrieves the maximum value. The maximum value of the specified member is assigned to the result "Out".

RecMin: Retrieves the minimum value. The minimum value of the specified member is assigned to the result "Out".

An example for the RecMax instruction with "Size"=UINT#5 is shown below.



	FBD	ST
Variable declaration	VAR size :UINT; In :ARRAY [1..10] OF DUT; out1 :UINT; out2 :UINT; Maxpos :UINT; Minpos :UINT; Maxnum :UINT; Minnum :UINT; END_VAR	
Program	 	RecMax(In:=In[1] , Size:=size , Member:=In[1].real1 , Out:=out1.real1 , InOutPos=>Maxpos , Num=>Maxnum); RecMin(In:=In[1] , Size:=size , Member:=In[1].real1 , Out:=out2.real1 , InOutPos=>Minpos , Num=>Minnum);

Result

表达式	类型	值	准备值	地址	注
In[2]	DUT				
In[3]	DUT				
In[4]	DUT				
In[5]	DUT				
In[6]	DUT				
In[7]	DUT				
In[8]	DUT				
real1	REAL	1			
int1	INT	0			
In[9]	DUT				
real1	REAL	3456			
int1	INT	0			
In[10]	DUT				
out1	DUT				
real1	REAL	3456			
int1	INT	0			
out2	DUT				
real1	REAL	1			
int1	INT	0			
Maxpos	UINT	8			
Minpos	UINT	7			
Maxnum	UINT	10			
Minnum	UINT	10			

◆ Key points

• When the data types of "Member" and "Out" differ, choose from the following data types to ensure the valid range of "Out" encompasses the valid range of "Member". (INT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, LREAL)

• When "Member" is a real number, results may be unexpected due to rounding errors inherent to the value.

• When transferring an element from an array to In[], the element's subsequent data is the processing target.

• When "In" is an enumerated type, always use a variable as the input parameter for "In". An exception occurs during linking if a constant is passed.

• When the value of "Size" is 0, the values of "Out" and "Num" become 0. The value of InOutPos[] remains unchanged.

• An exception occurs and ENO becomes FALSE, "Out", InOutPos[], and "Num" remain unchanged under the following conditions:

• When the value of "Size" exceeds the array bounds of In[].

• When "Member" is not a member of the structures in In[].

• When the size of the InOutPos[] array is less than the dimension of In[].

• When an array without an index is passed to In[].

• When the data type is not supported for "Member".

• When "Member" is of STRING type and is not NULL-terminated.

5.12 String instructions

5.12.1 ClearString (String clear)

The instruction clears a string.

Instruction	Name	FB/FUN	Graphical representation	ST representation
ClearString	String clear	FUN		ClearString(InOut);

◆ Variables

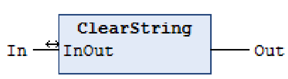
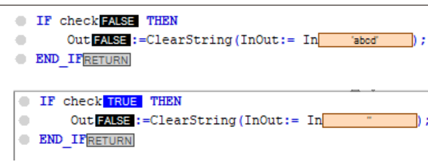
	Name	Input/Output	Description	Valid range	Unit	Initial value
InOut	String to clear	Input/Output	String to be cleared.	Conforms to data type.	—	—
Out	Return value	Output	Always TRUE.	TRUE only.		

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out	○																			

◆ Function

It clears the string "InOut". The entire area of "InOut" is filled with NULL characters. An example is shown below.

A STRING-type variable In with default value 'abcd' is cleared after the function call.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR check :BOOL; In :STRING :='abcd'; Out :BOOL; END_VAR	
Program		
Result	表达式 / 类型 / 值 check / BOOL / FALSE In / STRING / 'abcd' Out / BOOL / FALSE	表达式 / 类型 / 值 check / BOOL / TRUE In / STRING / '' Out / BOOL / FALSE

◆ Key points

- The return value "Out" is not used when employing this instruction in an ST program.

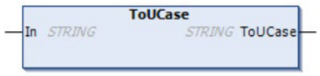
E

• OmronUtils (Omron Instruction Functions)

5.12.2 ToUCase/ToLCase (String case conversion)

ToUCase: Converts all half-width letters in a string to uppercase.

ToLCase: Converts all half-width letters in a string to lowercase.

Instruction	Name	FB/FUN	Graphical representation	ST representation
ToUCase	String to uppercase	FUN		Out:=ToUCase(In);
ToLCase	String to lowercase	FUN		Out:=ToLCase(In);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Source string	Input	Source string to be converted.	Conforms to data type.	—	''
Out	Converted string	Output	Converted string.	Conforms to data type.		—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out																				○

◆ Function

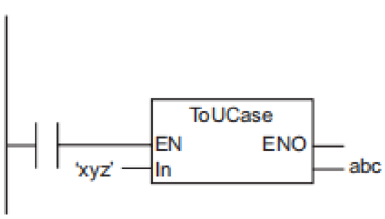
ToUCase: Converts all half-width letters in the source string "In" to uppercase.

ToLCase: Converts all half-width letters in the source string "In" to lowercase.

For all instructions, the output string is terminated with a NULL character. Characters that are not half-width letters are unaffected.

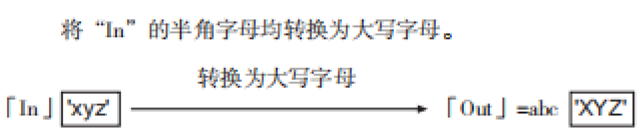
An example for the ToUCase instruction with "In"='xyz' is shown below. The variable abc has the value 'XYZ'.

LD

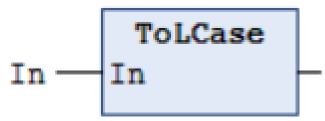


ST

abc:=ToUCase('xyz');



	FBD	ST
Variable declaration	VAR In :STRING; Out :STRING; END_VAR	

Program		Out:=ToLCase(In:=In);
Result	In STRING 'AABb' Out STRING 'aabb'	

◆ Key points

- Full-width letters are not converted.
- An exception occurs and ENO becomes FALSE, "Out" remains unchanged under the following conditions:
- When "In" is not NULL-terminated.
- When the character code of "In" is abnormal.
- When the conversion result exceeds the size of "Out".

5.12.3 TrimL/TrimR (String left/right trim)

TrimL: Removes spaces from the beginning of a string.

TrimR: Removes spaces from the end of a string.

Instruction	Name	FB/FUN	Graphical representation	ST representation
TrinL	String left trim	FUN		Out:=TrinL(In);
TrinR	String right trim	FUN		Out:=TrinR(In);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Source string	Input	Source string to be trimmed.	Conforms to data type.	—	''
Out	Trimmed string	Output	Trimmed string.	Conforms to data type.		—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out																				○

◆ Function

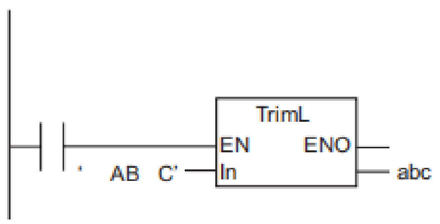
TrimL: Removes spaces from the beginning of the source string "In" up to the first non-space character. If the beginning has no spaces, no action is taken.

TrimR: Removes spaces from the end of the source string "In", starting from the last character. If the end has no spaces, no action is taken.

For all instructions, the output string is terminated with a NULL character. Both ASCII space (16#20) and Chinese full-width space (16#E38080) are treated as spaces.

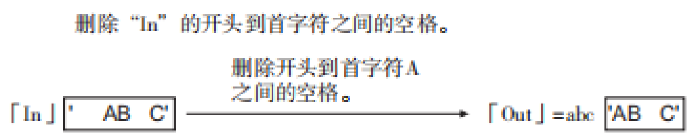
An example for the TrimL instruction with "In"=' AB C' is shown below. The variable abc has the value 'AB C'.

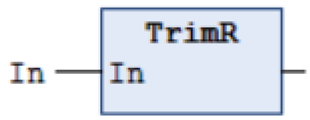
LD



ST

```
abc:=TrimL('   AB  C');
```



	FBD	ST
Variable declaration	VAR In :STRING; Out :STRING; END_VAR	
Program		Out:=TrimR(In:=In);
Result	In STRING 'AABB ' Out STRING 'AABB'	

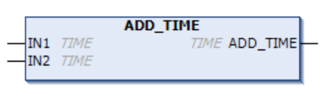
◆ Key points

- An exception occurs and ENO becomes FALSE, and "Out" remains unchanged under the following conditions:
- When "In" is not NULL-terminated.
- When the character code of "In" is abnormal.
- When the conversion result exceeds the size of "Out".

5.13 Time/Time of day instructions

5.13.1 ADD_TIME (Add time)

The instruction adds two time durations.

Instruction	Name	FB/FUN	Graphical representation	ST representation
ADD_TIME	Time addition	FUN		Out:=ADD_TIME(In1, In2);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In1	Augend time	Input	Augend time.	Conforms to data type.	ms	T#0S
In2	Addend time		Addend time.			
Out	Sum time	Output	Sum time.	Conforms to data type.	ms	—

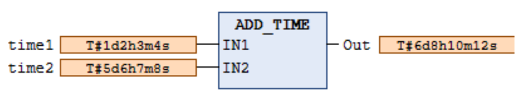
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																○				
In2																○				
Out																○				

◆ Function

It adds the time durations "In1" and "In2". The result "Out" is also a time duration.

An example with "In1"=T#1d2h3m4s and "In2"=T#5d6h7m8s is shown below.

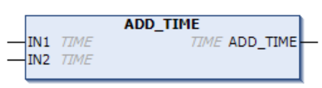
	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR time1 :TIME; time2 :TIME; out :TIME; check :BOOL; END_VAR	

Program		IF check FALSE THEN
		Out T#6d8h10m12s :=ADD_TIME(
		IN1:=time1 T#1d2h3m4s ,
		IN2:=time2 T#5d6h7m8s);
		check FALSE :=FALSE;
		END_IF

Result	表达式 / 类型 / 值 time1 / TIME / T#1d2h3m4s time2 / TIME / T#5d6h7m8s out / TIME / T#6d8h10m12s	

5.13.2 ADD_TOD_TIME (Add time of day and time)

The instruction adds a time duration to a time of day.

Instruction	Name	FB/FUN	Graphical representation	ST representation
ADD_TOD_TIME	Time of day and time addition	FUN		Out:=ADD_TOD_TIME(In1, In2);

◆ Variables

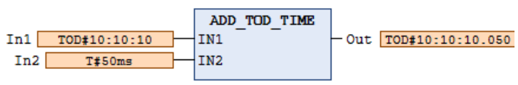
	Name	Input/Output	Description	Valid range	Unit	Initial value
In1	Time of day	Input	Time of day.	Conforms to data type.	Hour, minute, second	TOD#0:0:0
In2	Time duration		Time duration.		ms	T#0ms

Out	Result time of day	Output	Resulting time of day.	Conforms to data type.	Hour, minute, second	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																		○		
In2																○				
Out																		○		

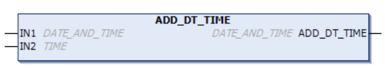
◆ Function

It adds the time duration "In2" to the time of day "In1". The result "Out" is a time of day.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In1 :TOD; In2 :TIME; out :TOD; END_VAR	
Program		out TOD#10:10:10.050 :=ADD_TOD_TIME(IN1:= In1 TOD#10:10:10 , IN2:=In2 T#50ms);
Result	表达式 / 类型 / 值 In1 / TIME_OF_DAY / TOD#10:10:10 In2 / TIME / T#50ms out / TIME_OF_DAY / TOD#10:10:10.050	

5.13.3 ADD_DT_TIME (Add date-time and time)

The instruction adds a time duration to a date and time.

Instruction	Name	FB/FUN	Graphical representation	ST representation
ADD_DT_TIME	Date-time and time addition	FUN		Out:=ADD_DT_TIME(In1, In2);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In1	Date and time	Input	Date and time.	Conforms to data type.	Year, month, day, hour, minute, second	TDT#1970-1-1-0:0:0
In2	Time duration		Time duration.		ns	T#0s

Out	Result date and time	Output	Resulting date and time.	Conforms to data type.	Year, month, day, hour, minute, second	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																			○	
In2																○				
Out																			○	

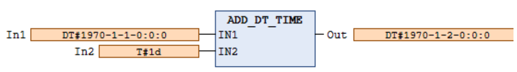
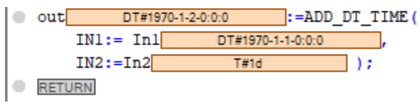
◆ Function

It adds the time duration "In2" to the date and time "In1". The result "Out" is a date and time. Leap years are accounted for.

An example with "In1"=DT#1970-1-1-0:0:0 and "In2"=T#1d is shown below.

	FBD	ST

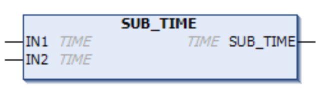
Variable declaration	`PROGRAM PLC_PRG`
	`VAR`
	`In1 :DT;`
	`In2 :TIME;`
	`out :DT;`
	`END_VAR`	

Program		
Result		

表达式	类型	值
In1	DATE_AND_TIME	DT#1970-1-1-0:0:0
In2	TIME	T#1d
out	DATE_AND_TIME	DT#1970-1-2-0:0:0

5.13.4 SUB_TIME (Subtract time)

The instruction subtracts a time duration from another time duration.

Instruction	Name	FB/FUN	Graphical representation	ST representation
SUB_TIME	Time subtraction	FUN		Out:=SUB_TIME(In1, In2);

◆ Variables

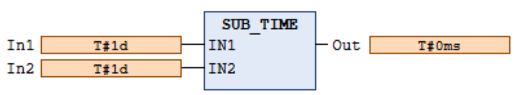
	Name	Input/Output	Description	Valid range	Unit	Initial value
In1	Minuend time	Input	Minuend time.	Conforms to data type.	ms	T#0ms
In2	Subtrahend time		Subtrahend time.			
Out	Difference time	Output	Difference time.	Conforms to data type.	ms	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																○				
In2																○				
Out																○				

◆ Function

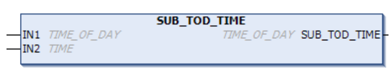
It subtracts the time duration "In2" from the time duration "In1". The result "Out" is also a time duration.

An example with "In1"="In2"=T#1d is shown below.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In1 :TIME; In2 :TIME; out :TIME; END_VAR	
Program		out T#0ms :=SUB_TIME(IN1:=In1 T#1d , IN2:=In2 T#1d); RETURN
Result	表达式 | 类型 | 值 out | TIME | T#0ms In2 | TIME | T#1d In1 | TIME | T#1d	

5.13.5 SUB_TOD_TIME (Subtract time from time of day)

The instruction subtracts a time duration from a time of day.

Instruction	Name	FB/FUN	Graphical representation	ST representation
SUB_TOD_TIME	Subtract time from time of day	FUN		Out:=SUB_TOD_TIME(In1, In2);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In1	Minuend time of day	Input	Minuend time of day.	Conforms to data type.	Hour, minute, second	TOD# 0:0:0
In2	Subtrahend time		Subtrahend time.		ms	T#0s
Out	Difference time of day	Output	Resulting time of day.	Conforms to data type.	Hour, minute, second	—

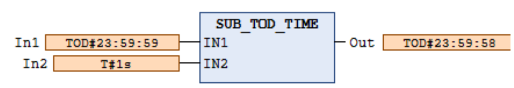
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																		○		
In2																○				

Out																	○			

◆ Function

It subtracts the time duration "In2" from the time of day "In1". The result "Out" is a time of day.

An example with "In1"=TOD#23:59:59 and "In2"=T#1s is shown below.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In1 :TOD; In2 :TIME; out :TOD; END_VAR	
Program		out TOD#23:59:58 :=SUB_TOD_TIME(IN1:=In1 TOD#23:59:59 , IN2:=In2 T#1s); RETURN
Result	表达式 / 类型 / 值 out / TIME_OF_DAY / TOD#23:59:58 In2 / TIME / T#1s In1 / TIME_OF_DAY / TOD#23:59:59	

5.13.6 SUB_TOD_TOD (Time of day subtraction)

The instruction subtracts a time of day from another time of day.

Instruction	Name	FB/FUN	Graphical representation	ST representation
SUB_TOD_TOD	Time of day subtraction	FUN		Out:=SUB_TOD_TOD(In1, In2);

◆ Variables

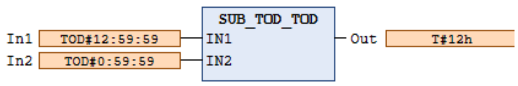
	Name	Input/Output	Description	Valid range	Unit	Initial value
In1	Minuend time of day	Input	Minuend time of day.	Conforms to data type.	Hour, minute, second	TOD#0:0:0
In2	Subtrahend time of day		Subtrahend time of day.			
Out	Difference time	Output	Difference time.	Conforms to data type.	ms	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																		○		
In2																		○		
Out																○				

◆ Function

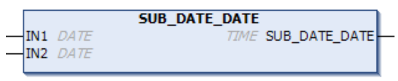
It subtracts the time of day "In2" from the time of day "In1". The subtraction result "Out" is a time duration.

An example with "In1" = TOD#12:59:59 and "In2" = TOD#0:59:59 is shown below.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In1 :TOD; In2 :TOD; out :TIME; END_VAR	
Program		out T#12h :=SUB_TOD_TOD(IN1:=In1 TOD#12:59:59 , IN2:=In2 TOD#0:59:59); RETURN
Result	表达式 / 类型 / 值 out / TIME / T#12h In2 / TIME_OF_DAY / TOD#0:59:59 In1 / TIME_OF_DAY / TOD#12:59:59	

5.13.7 SUB_DATE_DATE (Date subtraction)

The instruction subtracts a date from another date.

Instruction	Name	FB/FUN	Graphical representation	ST representation
SUB_DATE_DATE	Date subtraction	FUN		Out:=SUB_DATE_DATE(In1,In2);

◆ Variables

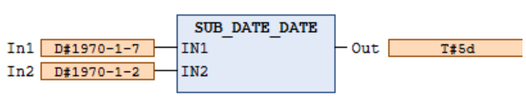
	Name	Input/Output	Description	Valid range	Unit	Initial value
In1	Minuend date	Input	Minuend date.	Conforms to data type.	Year, month, day	D#1970-1-1
In2	Subtrahend date		Subtrahend date.			
Out	Difference time	Output	Difference time.	Conforms to data type.	ms	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																	○			
In2																	○			
Out																○				

◆ Function

It subtracts the date "In2" from the date "In1". The subtraction result "Out" is a time duration.

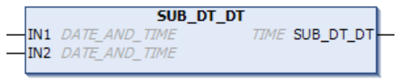
An example with "In1"=D#1970-1-7 and "In2"=D#1970-1-2 is shown below.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In1 :DATE; In2 :DATE; out :TIME; END_VAR	
Program		out T#5d :=SUB_DATE_DATE(IN1:=In1 D#1970-1-7 , IN2:=In2 D#1970-1-2); RETURN

Result	表达式	类型	值
	In1	DATE	D#1970-1-7
	In2	DATE	D#1970-1-2
	out	TIME	T#5d

5.13.8 SUB_DT_DT (Date and time subtraction)

The instruction subtracts a date and time from another date and time.

Instruction	Name	FB/FUN	Graphical representation	ST representation
SUB_DT_DT	Date and time subtraction	FUN		Out:=SUB_DT_DT(In1, In2);

◆ Variables

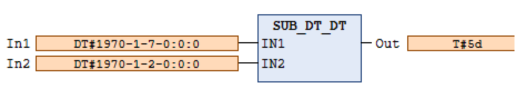
	Name	Input/Output	Description	Valid range	Unit	Initial value
In1	Minuend date and time.	Input	Minuend date and time.	Conforms to data type.	Year, month, day, hour, minute, second	DT#1970-1-1-0:0:0
In2	Subtrahend date and time.		Subtrahend date and time.			
Out	Difference time.	Output	Difference time.	Conforms to data type.	ns	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																			○	
In2																			○	
Out																○				

◆ Function

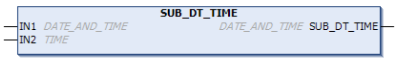
It subtracts the date and time "In2" from the date and time "In1". The subtraction result "Out" is a time duration.

An example with "In1"=DT#1970-1-7-0:0:0 and "In2"=DT#1970-1-2-0:0:0 is shown below.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In1 :DT; In2 :DT; out :TIME; END_VAR	
Program		out T#5d :=SUB_DT_DT(IN1:=In1 DT#1970-1-7-0:0:0 , IN2:=In2 DT#1970-1-2-0:0:0); RETURN
Result	表达式 / 类型 / 值 In1 / DATE_AND_TIME / DT#1970-1-7-0:0:0 In2 / DATE_AND_TIME / DT#1970-1-2-0:0:0 out / TIME / T#5d	

5.13.9 SUB_DT_TIME (Subtract time from date and time)

The instruction subtracts a time duration from a date and time.

Instruction	Name	FB/FUN	Graphical representation	ST representation
SUB_DT_TIME	Subtract time from date and time	FUN		Out:=SUB_DT_TIME(In1, In2);

◆ Variables

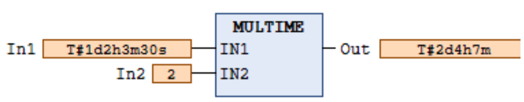
	Name	Input/Output	Description	Valid range	Unit	Initial value
In1	Minuend date and time	Input	Minuend date and time.	Conforms to data type.	Year, month, day, hour, minute, second	DT#1970-1-1-0:0:0
In2	Subtrahend time		Subtrahend time.		ms	T#0s
Out	Difference date and time	Output	Resulting date and time.	Conforms to data type.	Year, month, day, hour, minute, second	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																			○	
In2																○				
Out																			○	

◆ Function

It subtracts the time duration "In2" from the date and time "In1". The subtraction result "Out" is a date and time. Leap years are accounted for.

An example with "In1"=DT#1970-1-7-0:0:0 and "In2"=T#1d is shown below.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In1 :DT; In2 :TIME; out :DT; END_VAR	
Program		out T#2d4h7m :=MULTIME(IN1:=In1 T#1d2h3m30s , IN2:=In2 2); RETURN
Result	表达式 | 类型 | 值 In1 | TIME | T#1d2h3m30s In2 | USINT | 2 out | TIME | T#2d4h7m	

5.13.10 MULTIME (Time multiplication)

The instruction multiplies a time duration by a specified multiplier.

Instruction	Name	FB/FUN	Graphical representation	ST representation
MULTIME	Time multiplication	FUN	MULTIME IN1 TIME IN2 __SYSTEM.AnyType TIME MULTIME	Out:=MULTIME(In1, In2);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In1	Multiplicand time	Input	Multiplicand time.	Conforms to data type.	ms	T#0s
In2	Multiplier		Multiplier		—	(*)
Out	Product time	Output	Product time.	Conforms to data type.	ms	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																○				
In2						○	○	○	○	○	○	○	○	○	○					
Out																○				

◆ Function

It multiplies the time duration "In1" by the multiplier "In2". The multiplication result "Out" is a time duration. Leap years are accounted for.

An example with "In1"=T#1d2h3m30s and "In2"=INT#2 is shown below.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In1 :TIME; In2 :USINT; out :TIME; END_VAR	
Program	MULTIME In1 T#1d2h3m30s — IN1 In2 2 — IN2 Out T#2d4h7m	out T#2d4h7m :=MULTIME(IN1:=In1 T#1d2h3m30s, IN2:=In2 2); RETURN
Result	表达式 / 类型 / 值 In1 TIME T#1d2h3m30s In2 USINT 2 out TIME T#2d4h7m	

5.13.11 DIVTIME (Time division)

The instruction divides a time duration by a specified divisor.

Instruction	Name	FB/FUN	Graphical representation	ST representation
MULTIME	Time division	FUN	DIVTIME IN1 TIME IN2 __SYSTEM.AnyType TIME DIVTIME	Out:=MULTIME(In1, In2);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In1	Dividend time	Input	Dividend time.	Conforms to data type.	ms	T#0s
In2	Divisor		Divisor.		—	(*)
Out	Quotient time	Output	Quotient time.	Conforms to data type.	ms	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																○				
In2						○	○	○	○	○	○	○	○	○	○					
Out																○				

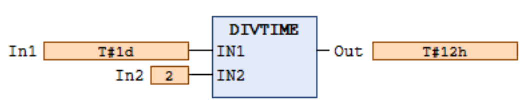
◆ Function

It divides the time duration "In1" by the divisor "In2". The division result "Out" is a time duration.

An example with "In1"=T#1d and "In2"=INT#2 is shown below.

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">

```
PROGRAM PLC_PRG
VAR
    In1         :TIME;
    In2         :USINT;
    out         :TIME;
END_VAR
```

</td></tr>
<tr><td>Program</td><td>
</td><td>

```
out T#12h :=DIVTIME(
    IN1:=In1 T#1d ,
    IN2:=In2 2 );
RETURN
```

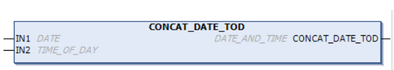
</td></tr>
<tr><td>Result</td><td colspan="2">

表达式	类型	值
In1	TIME	T#1d
In2	USINT	2
out	TIME	T#12h

</td></tr>
</table>

5.13.12 CONCAT_DATE_TOD (Combine date and time of day)

The instruction combines a date and a time of day.

Instruction	Name	FB/FUN	Graphical representation	ST representation
CONCAT_DATE_TOD	Combine date and time of day	FUN		Out:=CONCAT_DATE_TOD(In1, In2);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In1	Date	Input	Date.	Conforms to data type.	Year, month, day	D#1970-1-1
In2	Time of day		Time of day.		Hour, minute, second	TOD#0:0:0

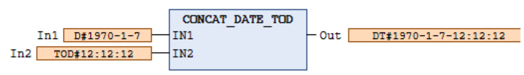
Out	Combined date and time	Output	Combined date and time.	Conforms to data type.	Year, month, day, hour, minute, second	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																	○			
In2																		○		
Out																			○	

◆ Function

It combines the date "In1" and the time of day "In2". The combination result "Out" is a date and time.

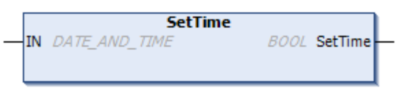
An example with "In1"=D#1970-1-7 and "In2" = TOD#12:12:12 is shown below.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In1 :DATE; In2 :TOD; out :DT; END_VAR	
Program		out DT#1970-1-7-12:12:12 :=CONCAT_DATE_TOD(IN1:=In1 D#1970-1-7 , IN2:=In2 TOD#12:12:12); RETURN
Result		

表达式	类型	值
In1	DATE	D#1970-1-7
In2	TIME_OF_DAY	TOD#12:12:12
out	DATE_AND_TIME	DT#1970-1-7-12:12:12

5.13.13 SetTime (Clock adjustment)

The instruction sets the UTC time.

Instruction	Name	FB/FUN	Graphical representation	ST representation
SetTime	Clock adjustment	FUN		SetTime(In);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	UTC time data	Input	Current date and time to set the system clock to.	Conforms to data type.	Year, month, day, hour, minute, second	DT#1970-1-1-0:0:0
Out	Return value	Output	Clock is TRUE.	TRUE only.	—	—

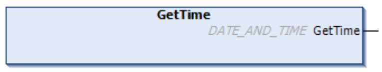
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			○	
Out	○																			

◆ Key points

- Set the system UTC time to the value of date and time "In".
- It is recommended to use this instruction in the EtherCat_Task. It may cause blocking.

5.13.14 GetTime (Get time)

The instruction reads the current UTC time.

Instruction	Name	FB/FUN	Graphical representation	ST representation
GetTime	Get time	FUN		Out:=GetTime();

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Out	Current time	Output	Current time.	Conforms to data type.	Year, month, day, hour, minute, second	—

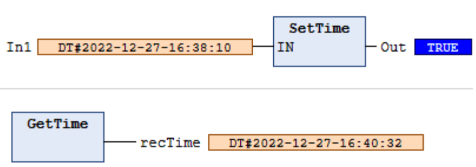
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out																			○	

◆ Function

It reads the current time. The current time is the standard time for the set time zone, not GMT (Greenwich Mean Time).

Example: Set the system time to "In" = DT#2022-12-27-16:38:10 using the SetTime instruction, then read the system time into "recTime" using the GetTime instruction.

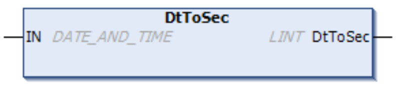
	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In1 :DATE; In2 :TOD; out :DT; END_VAR	

Program		IF set FALSE THEN
		out TRUE :=SetTime(IN:= In1 DT#2022-12-27-16:38:10);
		set FALSE :=FALSE;
		END_IF
		IF Out TRUE THEN
		recTime DT#2022-12-27-16:40:50 :=GetTime();
		END_IF
		RETURN

Result	表达式 / 类型 / 值 In1 / DATE_AND_TIME / DT#2022-12-27-16:38:10 Out / BOOL / TRUE recTime / DATE_AND_TIME / DT#2022-12-27-16:39:48 set / BOOL / FALSE

5.13.15 DtToSec (Date and time to seconds conversion)

The instruction converts a date and time to the number of seconds elapsed since 1970-01-01 00:00:00.

Instruction	Name	FB/FUN	Graphical representation	ST representation
DtToSec	Date and time to seconds conversion	FUN		Out:=DtToSec(In);

◆ Variables

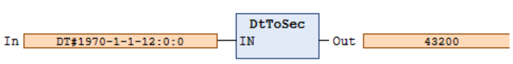
	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Date and time	Input	Date and time.	Conforms to data type.	Year, month, day, hour, minute, second	DT#1970-1-1-0:0:0
Out	Seconds	Output	Seconds elapsed since 1970-01-01 00:00:00.	0~18446744073	Seconds	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			○	
Out													○							

◆ Function

It converts the date and time "In" to the number of seconds elapsed since 1970-01-01 00:00:00. The result unit is seconds. Values less than 1 second are truncated.

An example with "In" = DT#1970-1-1-12:0:0 is shown below.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In :DT; Out :LINT; END_VAR	
Program		Out 43200 :=DtToSec(IN:= In DT#1970-1-1-12:0:0); RETURN
Result	表达式 / 类型 / 值 In / DATE_AND_TIME / DT#1970-1-1-12:0:0 Out / LINT / 43200	

5.13.16 DateToSec (Date to seconds conversion)

The instruction converts a date to the number of seconds elapsed since 1970-01-01 00:00:00.

Instruction	Name	FB/FUN	Graphical representation	ST representation
DateToSec	Date to seconds conversion	FUN		Out:=DateToSec(In);

◆ Variables

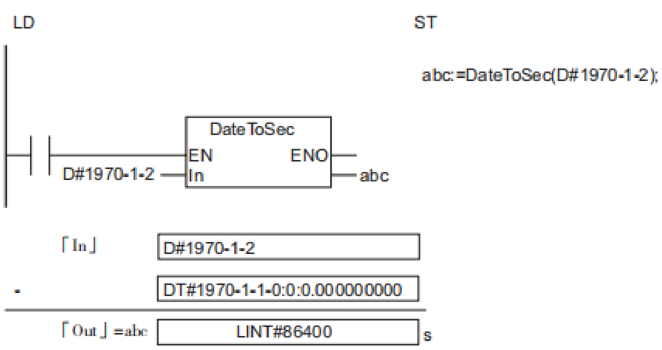
	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Date	Input	Date.	Conforms to data type.	Year, month, day	D#1970-1-1
Out	Seconds	Output	Seconds elapsed since 1970-01-01 00:00:00.	0~18446744073	Seconds	—

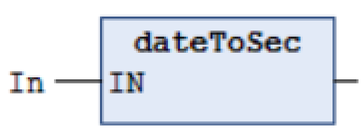
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	○			
Out													○							

◆ Function

It converts the date "In" at 00:00:00 to the number of seconds elapsed since 1970-01-01 00:00:00. The result unit is seconds.

An example with "In"=D#1970-1-2 is shown below.



	FBD	ST
Variable declaration	VAR In :DATE; Out :LINT; END_VAR	
Program		Out:=DateToSec(IN:=In);
Result	In DATE D#1970-1-2 Out LINT 86400	

5.13.17 TodToSec (Time of day to seconds conversion)

The instruction converts a time of day to the number of seconds elapsed since 00:00:00.

Instruction	Name	FB/FUN	Graphical representation	ST representation
TodToSec	Time of day to seconds conversion	FUN		Out:=TodToSec(In);

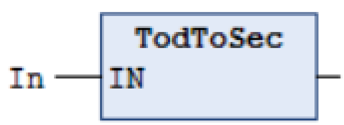
◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Time of day	Input	Time of day.	Conforms to data type.	Hour, Minute, Second	TOD#0:0:0
Out	Seconds	Output	Seconds elapsed since 00:00:00.	0~86399	Seconds	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																		○		
Out													○							

◆ Function

It converts the time of day "In" to the number of seconds elapsed since 00:00:00. The result unit is seconds. Values less than 1 second are truncated.

	FBD	ST
Variable declaration	VAR In :TOD; Out :LINT; END_VAR	
Program		Out:=TodToSec(IN:=In);
Result	In TIME_OF_DAY TOD#23:59:59 Out LINT 86399	

5.13.18 SecToDt (Seconds to date and time conversion)

The instruction converts the number of seconds elapsed since 1970-01-01 00:00:00 to a date and time.

Instruction	Name	FB/FUN	Graphical representation	ST representation
SecToDt	Seconds to date and time conversion	FUN		Out:=SecToDt(In);

◆ Variables

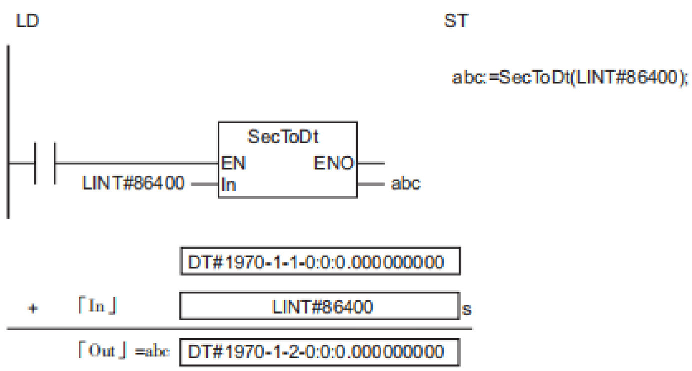
	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Seconds	Input	Seconds elapsed since 1970-01-01 00:00:00.	0~18446744073	Seconds	0
Out	Date and time	Output	Date and time.	Conforms to data type.	Year, month, day, hour, minute, second	—

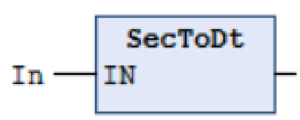
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													○							
Out																			○	

◆ Function

It converts the number of seconds "In" elapsed since 1970-01-01 00:00:00 to a date and time.

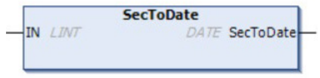
An example with "In"=LINT#86400 is shown below.



	FBD	ST
Variable declaration	VAR In :LINT; Out :DT; END_VAR	
Program		Out:=SecToDt(IN:=In);
Result	In LINT 86400 Out DATE_AND_TIME DT#1970-1-2-0:0:0	

5.13.19 SecToDate (Seconds to date conversion)

The instruction converts the number of seconds elapsed since 1970-01-01 00:00:00 to a date.

Instruction	Name	FB/FUN	Graphical representation	ST representation
SecToDate	Seconds to date conversion	FUN		Out:=SecToDate(In);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Seconds	Input	Seconds elapsed since 1970-01-01 00:00:00.	0~18446744073	Seconds	0
Out	Date	Output	Date.	Conforms to data type.	Year, month, day	—

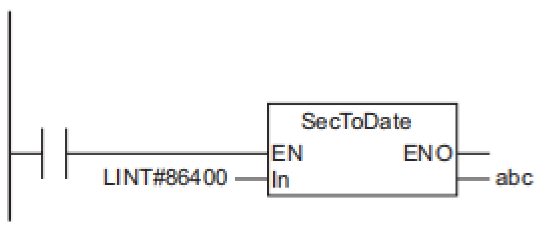
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													○							
Out																	○			

◆ Function

It converts the number of seconds "In" elapsed since 1970-01-01 00:00:00 to a date. Values less than 1 day are truncated.

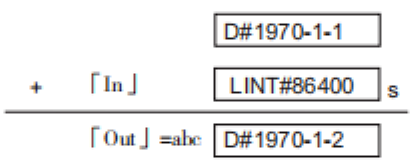
An example with "In"=LINT#86400 is shown below.

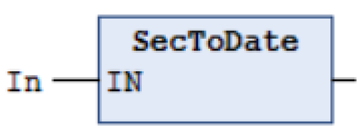
LD



ST

abc:=SecToDate(LINT#86400);



	FBD	ST
Variable declaration	VAR In :LINT; Out :DATE; END_VAR	
Program		Out:=SecToDate(IN:=In);

Result	In	LINT	86400
	Out	DATE	D#1970-1-2

5.13.20 SecToTod (Seconds to time of day conversion)

The instruction converts the number of seconds elapsed since 00:00:00 to a time of day.

Instruction	Name	FB/FUN	Graphical representation	ST representation
SecToTod	Seconds to time of day conversion	FUN	SecToTod: IN LINT → TIME_OF_DAY SecToTod	Out:=SecToTod(In);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Seconds	Input	Seconds elapsed since 00:00:00.	Conforms to data type. (*)	Seconds	0
Out	Time of day	Output	Time of day.	Conforms to data type.	Hour, minute, second	—

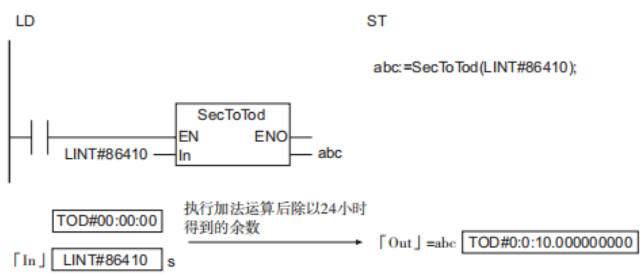
* Negative numbers are excluded.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													○							
Out																		○		

◆ Function

It converts the number of seconds "In" elapsed since 00:00:00 to a time of day. If the value of "In" is 24 hours or more, the remainder of "In" divided by 24 hours is converted to the time of day.

An example with "In"=LINT#86410 is shown below.



	FBD	ST
Variable declaration	VAR In :LINT; Out :TOD; END_VAR	
Program	In — IN SecToTod	Out:=SecToTod(IN:=In);

Result	In	LINT	86410
	Out	TIME_OF_DAY	TOD#0:0:10

◆ Key points

- An exception occurs and ENO becomes FALSE, "Out" remains unchanged under the following condition:
- When the value of "In" exceeds its valid range.

5.13.21 TimeToNanoSec (Time to nanoseconds conversion)

The instruction converts a time duration to nanoseconds.

Instruction	Name	FB/FUN	Graphical representation	ST representation
TimeToNanoSec	Time to nanoseconds conversion	FUN		Out:=TimeToNanoSec(In);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Time	Input	Time.	Conforms to data type.		T#0s
Out	Nanoseconds	Output	Nanoseconds.	(*)	ns	—

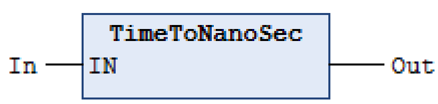
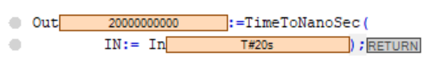
* -2,147,483,648 ~ 2,147,483,647

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																○				
Out													○							

◆ Function

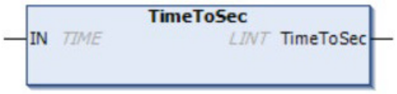
It converts the time duration "In" to nanoseconds.

An example with "In"=T#20s is shown below.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In :TIME; Out :LINT; END_VAR	
Program		
Result	表达式 / 类型 / 值 In / TIME / T#20s Out / LINT / 20000000000	

5.13.22 TimeToSec (Time to seconds conversion)

The instruction converts a time duration to seconds.

Instruction	Name	FB/FUN	Graphical representation	ST representation
TimeToSec	Time to seconds conversion	FUN		Out:=TimeToSec(In);

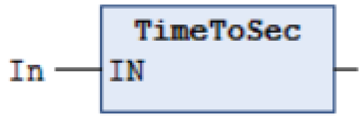
◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Time	Input	Time.	Conforms to data type.		T#0s
Out	Seconds	Output	Seconds.	9223372036~9223372036	Seconds	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																○				
Out													○							

◆ Function

It converts the time duration "In" to seconds. Values less than 1 second are truncated.

	FBD	ST
Variable declaration	VAR In :TIME; Out :LINT; END_VAR	
Program		Out:=TimeToSec(IN:=In);
Result	In TIME T#1d1h1m1s Out LINT 90061	

◆ Key points

- The unit of "In" is nanoseconds, and the unit of "Out" is seconds.

5.13.23 NanoSecToTime (Nanoseconds to time conversion)

The instruction converts nanoseconds to a time duration.

Instruction	Name	FB/FUN	Graphical representation	ST representation
NanoSecToTime	Nanoseconds to time conversion	FUN		Out:=NanoSecToTime(In);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Seconds	Input	Seconds.	(*)	ns	0
Out	Time	Output	Time.	Conforms to data type.	ns	—

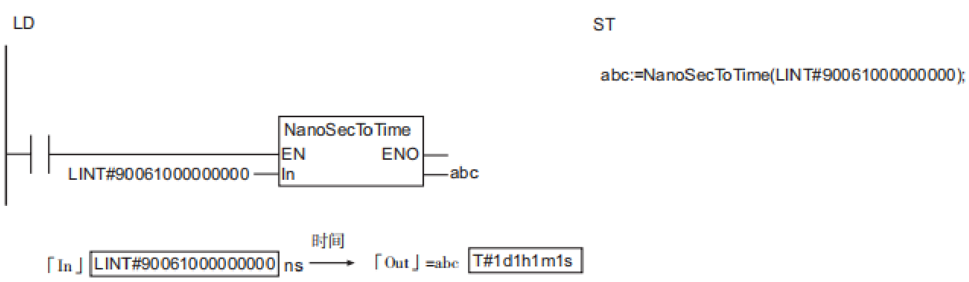
* -9223372036854775808 ~ 9223372036854775807

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													○							
Out																○				

◆ Function

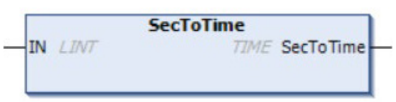
It converts the number of nanoseconds "In" to a time duration.

When "In"=LINT#90061000000000, the example is shown below.



5.13.24 SecToTime (Seconds to time conversion)

The instruction converts a number of seconds to a time duration.

Instruction	Name	FB/FUN	Graphical representation	ST representation
SecToTime	Seconds to time conversion	FUN		Out:=SecToTime(In);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Seconds	Input	Seconds.	0~4294967	s	0
Out	Time	Output	Time.	Conforms to data type.	s	—

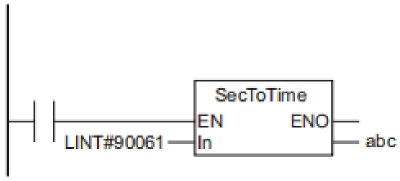
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													○							
Out																○				

◆ Function

It converts the number of seconds "In" to a time duration.

An example with "In"=LINT#90061 is shown below.

LD | ST



abc:=SecToTime(LINT#90061);

「In」 LINT#90061 s —时间→ 「Out」=abc T#1d1h1m1s

	FBD	ST
Variable declaration	VAR In :LINT; Out :TIME; END_VAR	
Program	In — SecToTime IN —	Out:=SecToTime(IN:=In);
Result	In LINT 90061 Out TIME T#1d1h1m1s	

◆ **Key points**

- The unit of "In" is seconds.
- An exception occurs and ENO becomes FALSE, "Out" remains unchanged under the following condition:
- When the value of "In" exceeds its valid range.

5.13.25 ChkLeapYear (Leap year check)

The instruction determines if a given year is a leap year.

Instruction	Name	FB/FUN	Graphical representation	ST representation
ChkLeapYear	Leap year check	FUN	ChkLeapYear IN UINT — BOOL ChkLeapYear	Out:=ChkLeapYear(In);

◆ **Variables**

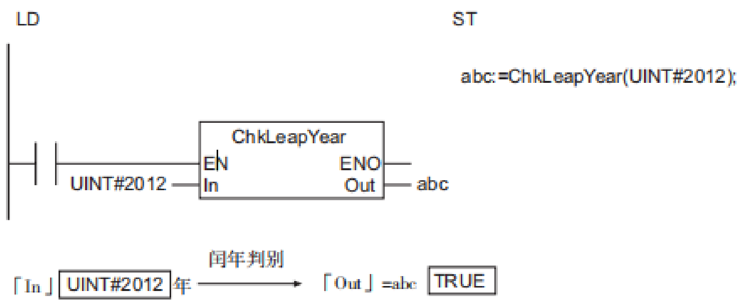
	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Year	Input	Year (Gregorian calendar).	1970~2554	Year	1970
Out	Judgment result	Output	TRUE: Is a leap year. FALSE: Is not a leap year.	Conforms to data type.	—	—

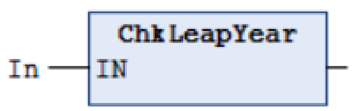
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In							○													
Out	○																			

◆ Function

It determines if the year "In" (Gregorian calendar) is a leap year. If it is a leap year, the judgment result "Out" is TRUE. Otherwise, it is FALSE.

An example with "In"=UINT#2012 is shown below.



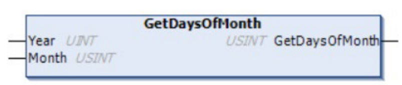
	FBD	ST
Variable declaration	VAR In :UINT; Out :BOOL; END_VAR	
Program		Out:=ChkLeapYear(IN:=In);
Result	In UINT 2012 Out BOOL TRUE	

◆ Key points

- When the value of "In" exceeds its valid range, no exception occurs, and the value of "Out" becomes an error value.

5.13.26 GetDaysOfMonth (Get days of month)

The instruction obtains the number of days in a specified month.

Instruction	Name	FB/FUN	Graphical representation	ST representation
GetDaysOfMonth	Get days of month	FUN		Out:=GetDaysOfMonth(Year, Month);

◆ Variables

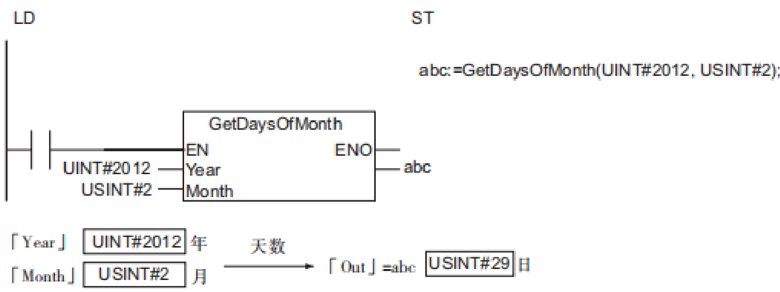
	Name	Input/Output	Description	Valid range	Unit	Initial value
Year	Year	Input	Year (Gregorian calendar).	1970~2554	Year	1970
Month	Month		Month	1~12	Month	1
Out	Number of days	Output	Number of days.	28~31	Day	—

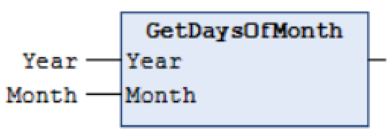
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Year							○													
Month						○														
Out						○														

◆ Function

It obtains the number of days for the month "Month" in the year "Year".

An example with "Year"=UINT#2012 and "Month"=USINT#2 is shown below.



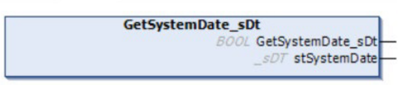
	FBD	ST
Variable declaration	VAR Year :UINT; Month :USINT; Days :USINT; END_VAR	
Program		Days:=GetDaysOfMonth(Year:=Year , Month:=Month);
Result	Year UINT 2012 Month USINT 2 Days USINT 29	

◆ Key points

- When the value of "Year" exceeds its valid range, no exception occurs, and the value of "Out" becomes an error value.
- An exception occurs and ENO becomes FALSE, "Out" remains unchanged under the following condition:
- When the value of "Month" exceeds its valid range.

5.13.27 GetSystemDate_sDt (Get system time in _sDT format)

The instruction obtains the system local time.

Instruction	Name	FB/FUN	Graphical representation	ST representation
GetSystemDate_ sDt	Get system local time	FUN		GetSystemDate_sDt(stSystemDate=>);

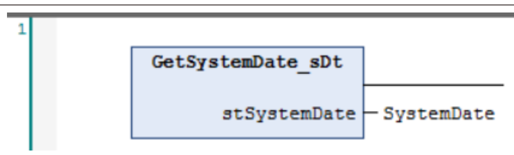
◆ **Variables**

	Name	Input/Output	Description	Valid range	Unit	Initial value
stSystemDate	Date and time	Output	Date and time, in _sDT format.	Conforms to data type.	Year, month, day, hour, minute, second	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
stSystemDate																			○	

◆ **Function**

It reads the current moment. The current moment is the standard time for the configured time zone.

	FBD	ST
Variable declaration	VAR SystemDate :hcfaAtcLib._sDT; END_VAR	
Program		GetSystemDate_sDt(stSystemDate=> SystemDate);
Result	SystemDate — hcfaAtcLib._sDT Year — UINT — 2023 Month — USINT — 11 Day — USINT — 12 Hour — USINT — 15 Minute — USINT — 28 Sec — USINT — 38 MilliS — UINT — 340	

5.13.28 DaysToMonth (Days to month conversion)

The instruction calculates the month based on the number of days elapsed since January 1st.

Instruction	Name	FB/FUN	Graphical representation	ST representation
DaysTo Month	Days to month conversion	FUN		Out:=DaysToMonth(Year,Days);

◆ **Variables**

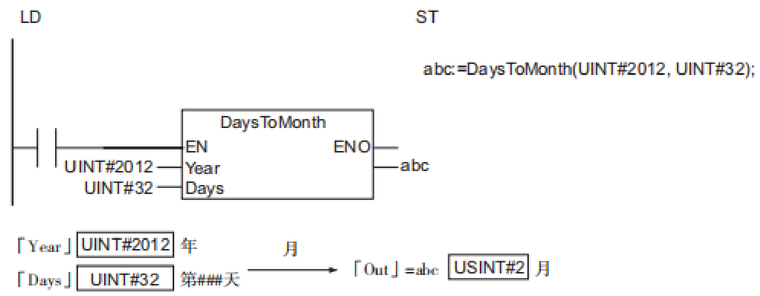
	Name	Input/Output	Description	Valid range	Unit	Initial value
Year	Year	Input	Year (Gregorian calendar).	1970~2554	Year	1970
Days	Days	Input	Number of days since January 1st.	1~365 (1~366 if "Year" is a leap year).	Day	1
Out	Month	Output	Month.	1~12	Month	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Year							○													
Days							○													
Out						○														

◆ Function

It calculates the month based on the number of days "Days" elapsed since January 1st of the year "Year".

An example with "Year"=UINT#2012 and "Days"=UINT#32 is shown below.



	FBD	ST
Variable declaration	VAR Year :UINT; Days :UINT; Out :USINT; END_VAR	
Program	DaysToMonth Year — Year Days — Days	Out:=DaysToMonth(Year:=Year , Days:=Days);
Result	Year UINT 2012 Days UINT 32 Out USINT 2	

◆ Key points

- When the value of "Year" exceeds its valid range, no exception occurs, and the value of "Out" becomes an error value.
- An exception occurs and ENO becomes FALSE, "Out" remains unchanged under the following condition:
- When the value of "Days" exceeds its valid range.

5.13.29 GetDayOfWeek (Get day of week)

The instruction obtains the day of the week for a specified date.

Instruction	Name	FB/FUN	Graphical representation	ST representation
GetDayOfWeek	Get day of week	FUN		Out:=GetDayOfWeek(In);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Date	Input	Date.	Conforms to data type.	Year, month, day	(*)
Out	Day of week	Output	Day of week.	_MON,_TUE, _WED,_THU, _FEI,_SAT, _SUN	Day of week	—

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	○		○	
Out	Enumeration _eDAYOFWEEK. See the Function section for enumeration elements.																			

◆ Function

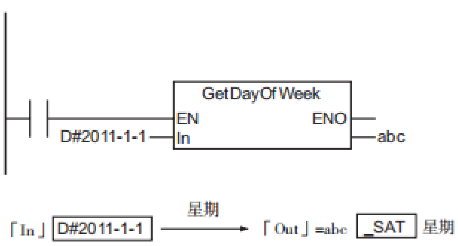
It obtains the day of the week for the date "In".

"Out" is of the enumeration type _eDAYOFWEEK. The meanings of the enumeration elements are as follows.

Enumeration element	Meaning
_MON	Monday
_TUE	Tuesday
_WED	Wednesday
_THU	Thursday
_FRI	Friday
_SAT	Saturday
_SUN	Sunday

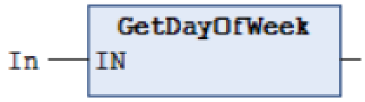
An example with "In"=D#2011-1-1 is shown below.

LD



ST

```
abc:=GetDayOfWeek(D#2011-1-1);
```

	FBD	ST
Variable declaration	VAR In :DATE; Out :_eDAYOFWEEK; END_VAR	
Program		Out:=GetDayOfWeek(IN:=In);
Result	In DATE_AND_TIME DT#2011-1-1-0:0:0 Out _EDAYOFWEEK _SAT	

5.13.30 GetWeekOfYear (Get week of year)

The instruction calculates the week number of the year for a specified date.

Instruction	Name	FB/FUN	Graphical representation	ST representation
GetWeekOfYear	Get week of year	FUN		Out:=GetWeekOfYear(In);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Date	Input	Date.	Conforms to data type.	Year, month, day	(*)
Out	Week number	Output	Week number of the year.	1~54	Day of week	—

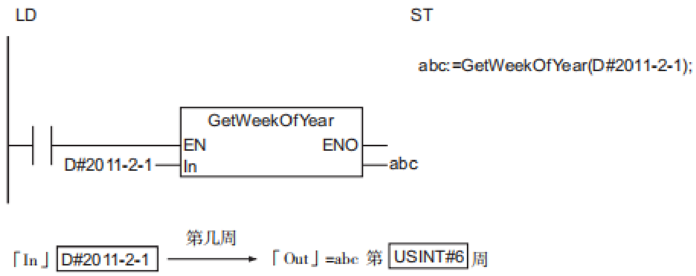
* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	○		○	
Out						○														

◆ Function

It calculates the week number of the year for the date "In". A week is defined as Monday to Sunday. The week number increments when Sunday changes to Monday. January 1st is always in week 1. For example, if January 1st is a Thursday, then January 1st (Thursday) to January 4th (Sunday) is week 1; January 5th (Monday) to January 11th (Sunday) is week 2.

An example with "In"=D#2011-2-1 is shown below.



	FBD	ST
Variable declaration	VAR In :DATE; Out :USINT; END_VAR	
Program	In — GetWeekOfYear IN —	Out:=GetWeekOfYear(IN:=In);
Result	In DATE D#2011-2-1 Out USINT 4	

5.13.31 DtToDateStruct (Date and time decomposition)

The instruction decomposes a date and time into "year", "month", "day", "hour", "minute", and "second".

Instruction	Name	FB/FUN	Graphical representation	ST representation
DtToDateStruct	Date and time decomposition	FUN	DtToDateStruct IN DATE_AND_TIME _sDT DtToDateStruct	Out:=DtToDateStruct(In, DateStruct);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Date and time	Input	Date and time.	Conforms to data type.	Year, month, day, hour, minute, second	DT#1970-1-1-0:0:0
Out	Return value	Output	Always TRUE.	TRUE only.	—	—
DateStruct	Decomposed date and time	Output	Date and time decomposed into "year", "month", "day", "hour", "minute", "second".	—	—	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			○	
Out	○																			
DateStruct	Structure _sDT. See the Function section for details.																			

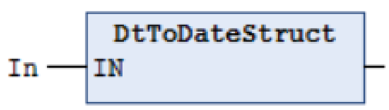
◆ Function

It decomposes the date and time "In" into "year", "month", "day", "hour", "minute", and "second".

The decomposed date and time "DateStruct" is of the structure type _sDT. The specifications is as follows.

Variable	Name	Description	Data type	Valid range	Unit	Initial value
DateStruct	Decomposed date and time	Date and time decomposed into "year", "month", "day", "hour", "minute", "second".	_sDt	—	—	—
Year	Year	Year	UINT	1970~2554	Year	
Month	Month	Month	USINT	1~12	Month	
Day	Day	Day	USINT	1~31	Day	—
Hour	Hour	Hour	USINT	0~23	Hour	
Min	Minute	Minute	USINT	0~59	Minute	
Sec	Second	Second	USINT	0~59	Second	

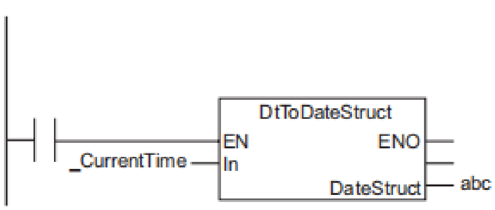
An example with "In"=DT#1970-1-1-12:12:12 is shown below.

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">VAR
In :DATE_AND_TIME;
Out :_sDT;
END_VAR</td></tr>
<tr><td>Program</td><td>
</td><td>Out:=DtToDateStruct(IN:=In);</td></tr>
<tr><td>Result</td><td colspan="2"><table>
<tr><td>In</td><td>DATE_AND_TIME</td><td>DT#1970-1-1-12:12:12</td></tr>
<tr><td>Out</td><td>_sDT</td><td></td></tr>
<tr><td>Year</td><td>UINT</td><td>1970</td></tr>
<tr><td>Month</td><td>USINT</td><td>1</td></tr>
<tr><td>Day</td><td>USINT</td><td>1</td></tr>
<tr><td>Hour</td><td>USINT</td><td>12</td></tr>
<tr><td>Minute</td><td>USINT</td><td>12</td></tr>
<tr><td>Sec</td><td>USINT</td><td>12</td></tr>
</table></td></tr>
</table>

◆ Reference

To combine the decomposed "year", "month", "day", "hour", "minute", and "second" into a date and time, use the "Date-StructToDt instruction". An example for obtaining the current time is shown below.

●LD



●ST

DtToDateStruct(_CurrentTime, abc);

◆ Key points

- The return value "Out" is not used when employing this instruction in an ST program.

5.13.32 DateStructToDt (Date and time composition)

The instruction combines a date and time decomposed into "year", "month", "day", "hour", "minute", and "second".

Instruction	Name	FB/FUN	Graphical representation	ST representation
DateStructToDt	Date and time composition	FUN		Out:=DateStructToDt(In);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Decomposed date and time	Input	Date and time decomposed into "year", "month", "day", "hour", "minute", "second".	—	—	—
Out	Decomposed date and time	Output	Decomposed date and time	Conforms to data type.	Year, month, day, hour, minute, second	—

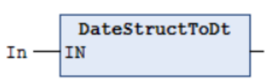
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	Structure _sDT. See the Function section for details.																			
Out																			○	

◆ Function

It combines the date and time "In", decomposed into "year", "month", "day", "hour", "minute", and "second".

"In" is of the structure type _sDT. The specification is as follows.

Variable	Name	Description	Data type	Valid range	Unit	Initial value
DateStruct	Decomposed date and time	Date and time decomposed into "year", "month", "day", "hour", "minute", "second".	_sDt	—	—	—
Year	Year	Year	UINT	1970~2554	Year	—
Month	Month	Month	USINT	1~12	Month	
Day	Day	Day	USINT	1~31	Day	
Hour	Hour	Hour	USINT	0~23	Hour	
Min	Minute	Minute	USINT	0~59	Minute	
Sec	Second	Second	USINT	0~59	Second	

	FBD	ST
Variable declaration	VAR In :_sDT; Out :DATE_AND_TIME; END_VAR	
Program		Out:=DateStructToDt(IN:=In);

Result	(table below)

In	_sDT	
Year	UINT	1970
Month	USINT	1
Day	USINT	1
Hour	USINT	12
Minute	USINT	12
Sec	USINT	12
Out	DATE_AND_TIME	DT#1970-1-1-12:12:12

◆ Key points

- An exception occurs and ENO becomes FALSE, "Out" remains unchanged under the following conditions:
- When the value of any structure element of "In" exceeds its valid range.
- When the processing result exceeds the valid range of "Out".

5.13.33 TruncTime (Time truncation)

The instruction truncates values smaller than a specified unit in an LTIME-type variable.

Instruction	Name	FB/FUN	Graphical representation	ST representation
TruncTime	Time truncation	FUN		Out:=TruncTime(In, Accuracy);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Source time	Input	Source time for truncation.	Conforms to data type.	ns	T#0s
Accuracy	Truncation unit	Input	The smallest time unit that is not truncated.	(1, 2, 4, 8)	—	0
Out	Truncated time	Output	Truncated time.	Conforms to data type.	ns	—

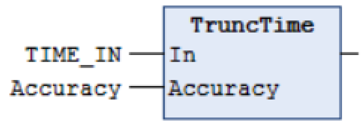
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																○				
Accuracy		○																		
Out																○				

◆ Function

It truncates values in the source time "In" that are smaller than the truncation unit "Accuracy". The truncated value is stored in the truncated time "Out".

"Accuracy" is of BYTE type. The meanings of its values are as follows.

Accuracy	Meaning
1	Second
2	Millisecond
4	Microsecond
8	Nanosecond

	FBD	ST
Variable declaration	VAR TIME_IN :LTIME; TIME_OUT :LTIME; Accuracy :BYTE; END_VAR	
Program		TIME_OUT:=TruncTime(In:=TIME_IN, Accuracy:=Accuracy);
Result	TIME_IN LTIME LTIME#12d20h38m31s111ms111us111ns TIME_OUT LTIME LTIME#12d20h38m31s Accuracy BYTE 1	

5.13.34 TruncDt (Date and time truncation)

The instruction truncates values smaller than a specified unit in a DT-type variable.

Instruction	Name	FB/FUN	Graphical representation	ST representation
TruncDt	Date and time truncation	FUN		Out:=TruncDt(In, Accuracy);

◆ Variables

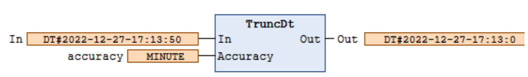
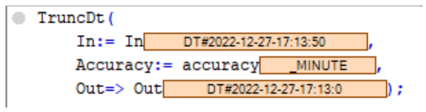
	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Source date and time	Input	Source date and time for truncation.	Conforms to data type.	Year, month, day, hour, minute, second	DT#1970-1-1-0:0:0
Accuracy	Truncation unit		The smallest time unit that is not truncated.	_MILLISEC, _SEC, _MINUTE, _HOUR	—	_MILLISEC
Out	Truncated date and time	Output	Truncated date and time.	Conforms to data type.	Year, month, day, hour, minute, second	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			○	
Accuracy	Enumeration _eSUBSEC. See the Function section for enumeration elements.																			
Out																			○	

◆ Function

It truncates values in the source time "In" that are smaller than the truncation unit "Accuracy". The truncated value is stored in the truncated time "Out".

An example with "In" = DT#2022-12-27-17:13:50 and "accuracy" = _MINUTE is shown below.

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2"><pre>PROGRAM PLC_PRG
VAR
    In   :DT;
    accuracy   :HCFA_OmronUitls._eSUBSEC;
    Out   :DT;
END_VAR</pre></td></tr>
<tr><td>Program</td><td>
</td><td>
</td></tr>
<tr><td>Result</td><td colspan="2"><table>
<tr><th>表达式</th><th>类型</th><th>值</th></tr>
<tr><td>In</td><td>DATE_AND_TIME</td><td>DT#2022-12-27-17:13:50</td></tr>
<tr><td>accuracy</td><td>_ESUBSEC</td><td>_MINUTE</td></tr>
<tr><td>Out</td><td>DATE_AND_TIME</td><td>DT#2022-12-27-17:13:0</td></tr>
</table></td></tr>
</table>

"Accuracy" is of the enumeration type _eSUBSEC. The meanings of the enumeration elements are as follows.

Enumeration element	Meaning
_MILLISEC	Millisecond
_SEC	Second
_MINUTE	Minute
_HOUR	Hour

5.13.35 TruncTod (Time of day truncation)

The instruction truncates values smaller than a specified unit in a TOD-type variable.

Instruction	Name	FB/FUN	Graphical representation	ST representation
TruncTod	Time of day truncation	FUN		Out:=TruncTod(In, Accuracy);

◆ Variables

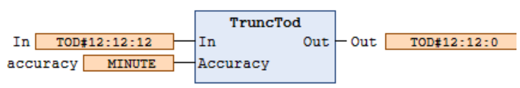
	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Source time of day	Input	Source time of day for truncation.	Conforms to data type.	Hour, minute, second	TOD#0:0:0
Accuracy	Truncation unit	Input	The smallest time unit that is not truncated.	_MILLISEC, _SEC, _MINUTE, _HOUR	—	_MILLISEC
Out	Truncated time of day	Output	Truncated time of day.	Conforms to data type.	Hour, minute, second	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																		○		
Accuracy	Enumeration _eSUBSEC. See the Function section for enumeration elements.																			
Out																		○		

◆ Function

It truncates values in the source time of day "In" that are smaller than the truncation unit "Accuracy". The truncated value is stored in the truncated time of day "Out".

An example with "In" = TOD#12:12:12 and "accuracy" = _MINUTE is shown below.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR In :TOD; accuracy :HCFA_OmronUitls._eSUBSEC; Out :TOD; END_VAR	
Program		TruncTod(In:= In TOD#12:12:12, Accuracy:= accuracy _MINUTE, Out=> Out TOD#12:12:0);
Result	表达式 | 类型 | 值 In | TIME_OF_DAY | TOD#12:12:12 accuracy | _ESUBSEC | _MINUTE Out | TIME_OF_DAY | TOD#12:12:0	

"Accuracy" is of the enumeration type _eSUBSEC. The meanings of the enumeration elements are as follows.

Enumeration element	Meaning
_MILLISEC	Millisecond
_SEC	Second
_MINUTE	Minute
_HOUR	Hour

5.13.36 TimeToMilliSec (Time to milliseconds conversion)

The instruction converts a time duration to milliseconds.

Instruction	Name	FB/FUN	Graphical representation	ST representation
TimeToMilliSec	Time to milliseconds conversion	FUN		Out:=TimeToMilliSec(in:=);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Time	Input	Time.	Conforms to data type.		T#0s
Out	Milliseconds	Output	Milliseconds.	9223372036~9223372036	Milliseconds	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																○				
Out													○							

◆ Function

	FBD	ST
Variable declaration	VAR Intime :TIME; Out :LINT; END_VAR	
Program	Intime — TimeToMilliSec in — Out	Out :=TimeToMilliSec(in:= Intime);
Result	Intime TIME T#5m36s Out LINT 336000	

5.13.37 MilliSecToTime (Milliseconds to time conversion)

The instruction converts milliseconds to a time duration.

Instruction	Name	FB/FUN	Graphical representation	ST representation
MilliSecToTime	Milliseconds to time conversion	FUN	MilliSecToTime in LINT — TIME MilliSecToTime	Out:= MilliSecToTime (in:=);

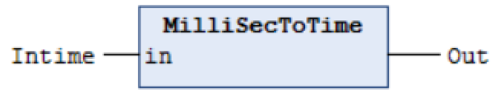
◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Milliseconds	Input	Milliseconds.	9223372036~9223372036	Milliseconds	
Out	Time	Output	Time.	Conforms to data type.		T#0s

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													○							
Out																○				

◆ Function

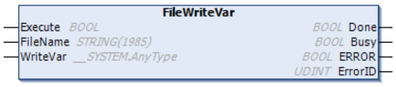
	FBD	ST
Variable declaration	VAR Intime :LINT; Out :TIME; END_VAR	

Program	 	`Out :=MilliSecToTime(in:= Intime);`
Result	Intime LINT 34567 Out TIME T#34s567ms	

5.14 SD Memory card instructions

5.14.1 FileWriteVar (Variable to file write)

The instruction writes the value of a single variable in binary format to a specified file on the SD memory card.

Instruction	Name	FB/FUN	Graphical representation	ST representation
FileWriteVar	Variable to file write	FB	 	FileWriteVar_instance(Execute, FileName, WriteVar, Done, Busy, Error, ErrorID);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Execute	Function block trigger	Input	Function block trigger.	Conforms to data type.	—	FALSE
FileName	Specified file name		File name to write to.	Maximum 66 bytes (65 half-width alphanumeric characters + terminating NULL character).	—	‘’
WriteVar	Specified variable		Variable to be written.			(*)
Done	Done	Output	Done.	Conforms to data type.	—	FALSE
Busy	Busy		Busy.		—	
Error	Error		Error.		—	
ErrorID	Error ID		Error code.		—	0

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

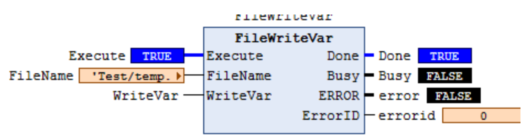
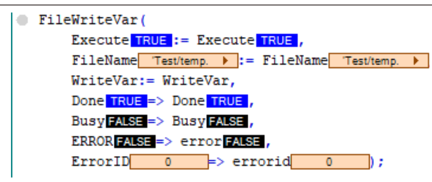
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileName																				○
WriteVar	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Also applicable to enumerated types, entire arrays, a single array element, entire structures, or a single structure element.																			
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

◆ Function

It writes the value of a single variable "WriteVar" in binary format to the file specified by "FileName" on the SD memory card. Enumerated types, entire arrays, a single array element, entire structures, or a single structure element can also be specified for "WriteVar". If a file with the same name as "FileName" does not exist on the SD card, a new file is created. If "FileName" includes a directory path and that directory does not exist on the SD card, the directory is created.

Note: A directory is created only if the deepest specified directory in the path does not exist.

An example is shown below. Writes the entire array variable WriteVar[] to a file named 'Test/temp.bin'. WriteVar[] is set as a BYTE-type array variable with 5 elements.

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">PROGRAM PLC_PRG
VAR
FileWriteVar :FileWriteVar;
Execute :BOOL;
FileName :STRING :='Test/temp.bin';
WriteVar :ARRAY [0..4] OF BYTE;
Done :BOOL;
Busy :BOOL;
error :BOOL;
errorid :UDINT;
END_VAR</td></tr>
<tr><td>Program</td><td>
</td><td>
</td></tr>
<tr><td>Result</td><td colspan="2"><table>
<tr><th>表达式</th><th>类型</th><th>值</th></tr>
<tr><td>FileWriteVar</td><td>FileWriteVar</td><td></td></tr>
<tr><td>WriteVar</td><td>ARRAY [0..4] O...</td><td></td></tr>
<tr><td>WriteVar[0]</td><td>BYTE</td><td>1</td></tr>
<tr><td>WriteVar[1]</td><td>BYTE</td><td>5</td></tr>
<tr><td>WriteVar[2]</td><td>BYTE</td><td>3</td></tr>
<tr><td>WriteVar[3]</td><td>BYTE</td><td>7</td></tr>
<tr><td>WriteVar[4]</td><td>BYTE</td><td>22</td></tr>
<tr><td>errorid</td><td>UDINT</td><td>0</td></tr>
<tr><td>Busy</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>error</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>FileName</td><td>STRING</td><td>'Test/temp.bin'</td></tr>
<tr><td>Execute</td><td>BOOL</td><td>TRUE</td></tr>
<tr><td>Done</td><td>BOOL</td><td>TRUE</td></tr>
</table></td></tr>
</table>

◆ Key points

- The instruction continues execution to completion even if "Execute" becomes FALSE or the execution time exceeds the task period. Confirm normal completion by checking if the value of "Done" becomes TRUE.

- When "WriteVar" is an entire structure, padding areas may be inserted between members depending on the specific structure.

- An exception occurs and "Error" is TRUE under the following conditions:

- When the SD memory card is not in a usable state.

- When the SD memory card is write-protected.
- When there is insufficient free space on the SD memory card.
- When the value of "FileName" is an invalid file name.
- When the number of files or directories that can be created is exceeded.
- When a file with the same name as "FileName" already exists and is being accessed.
- When a file with the same name as "FileName" already exists and the value of "OverWrite" is FALSE.
- When a file with the same name as "FileName" already exists and that file is read-only.
- When an exception occurs preventing access while the SD memory card is being accessed.

5.14.2 FileReadVar (File to variable read)

The instruction reads the value from a specified file on the SD memory card in binary format and writes it to a variable.

Instruction	Name	FB/FUN	Graphical representation	ST representation
FileReadVar	File to variable read	FB		FileReadVar_instance(Execute, FileName, ReadVar, Done, Busy, Error, ErrorID);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Execute	Function block trigger	Input	Function block trigger.	Conforms to data type.	—	FALSE
FileName	Specified file name		File name to read from.	Maximum 66 bytes (65 half-width alphanumeric characters + terminating NULL character).	—	‘’
ReadVar	Specified variable		Variable to read into.			(*)
Done	Done	Output	Done.	Conforms to data type.	—	FALSE
Busy	Busy		Busy.		—	
Error	Error		Error.		—	
ErrorID	Error ID		Error code.		—	0

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileName																				○
ReadVar	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Also applicable to enumerated types, entire arrays, a single array element, entire structures, or a single structure element.																			
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

◆ Function

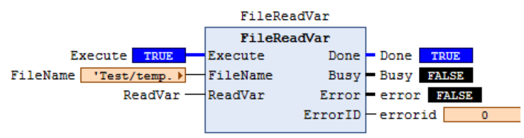
It reads the internal value of the file specified by "FileName" on the SD memory card in binary format. The read value is assigned to the read target variable "ReadVar". Enumerated types, entire arrays, a single array element, entire structures, or a single structure element can also be specified for "ReadVar".

An example is shown below. Reads the contents of a file named 'Test/temp.bin'. Writes the data into the array variable ReadVar[]. ReadVar[] is set as a BYTE-type array variable with 5 elements.

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">

```
PROGRAM PLC_PRG
VAR
    FileReadVar        :FileReadVar;
    Execute            :BOOL;
    FileName           :STRING :='Test/temp.bin';
    ReadVar            :ARRAY [0..4] OF BYTE;
    Done               :BOOL;
    Busy               :BOOL;
    error              :BOOL;
    errorid            :UDINT;
END_VAR
```

</td></tr>
<tr><td>Program</td><td>



</td><td>

```
FileReadVar(
    Execute TRUE := Execute TRUE,
    FileName 'Test/temp. := FileName 'Test/temp.
    ReadVar:= ReadVar,
    Done TRUE => Done TRUE,
    Busy FALSE => Busy FALSE,
    Error FALSE => error FALSE,
    ErrorID 0 => errorid 0);

RETURN
```

</td></tr>
<tr><td>Result</td><td colspan="2">

表达式	类型	值
⊞ FileReadVar	FileReadVar	
Execute	BOOL	TRUE
FileName	STRING	'Test/temp.bin'
⊟ ReadVar	ARRAY [0..4] O...	
ReadVar[0]	BYTE	1
ReadVar[1]	BYTE	5
ReadVar[2]	BYTE	3
ReadVar[3]	BYTE	7
ReadVar[4]	BYTE	22
Done	BOOL	TRUE
Busy	BOOL	FALSE
error	BOOL	FALSE
errorid	UDINT	0

</td></tr>
</table>

◆ Key points

・The instruction continues execution to completion even if "Execute" becomes FALSE or the execution time exceeds the task period. Confirm normal completion by checking if the value of "Done" becomes TRUE.

・If the specified file size is larger than the size of "ReadVar", no exception occurs; only data corresponding to the size of "ReadVar" is read.

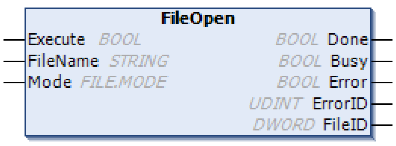
・If the specified file size is smaller than the size of "ReadVar", no exception occurs; only data corresponding to the size of the specified file is read. The remaining area of "ReadVar" retains its value from before the instruction execution.

・When "ReadVar" is an entire structure, padding areas may be inserted between members depending on the specific structure.

- An exception occurs and "Error" is TRUE under the following conditions:
- When the SD memory card is not in a usable state.
- When the value of "FileName" is an invalid file name.
- When the file specified in "FileName" does not exist.
- When the file specified in "FileName" is being accessed.
- When an exception occurs preventing access while the SD memory card is being accessed.

5.14.3 FileOpen (File open)

This instruction opens a specified file on the SD memory card.

Instruction	Name	FB/FUN	Graphical representation	ST representation
FileOpen	File open	FB		FileOpen_instance(Execute, FileName, Mode, Done, Busy, Error, ErrorID, FileID);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Execute	Function block trigger	Input	Function block trigger.	Conforms to data type.	—	FALSE
FileName	Specified file name		File name to open.	Maximum 66 bytes (65 half-width alphanumeric characters + terminating NULL character).	—	‘’
Mode	Open mode		Enumeration HCFA_OmronUitls.file.MODE.	MWRITE MREAD MRDWR MAPPD		(*)
Done	Done	Output	Done.	Conforms to data type.	—	FALSE
Busy	Busy		Busy.			
Error	Error		Error.			
ErrorID	Error ID		Error code.			0
FileID	File ID		File handle.			0

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileName																				○
Mode	Enumeration HCFA_OmronUitls.FILE.MODE. See the Function section for enumeration elements.																			

Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												
FileID				○																

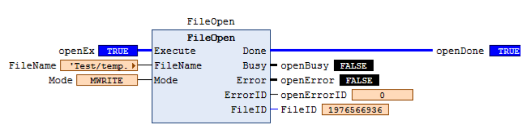
◆ Function

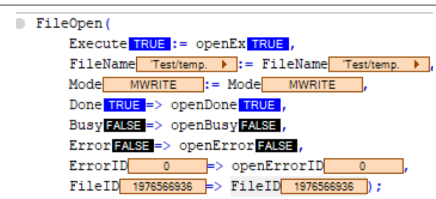
It opens the file specified by "FileName" on the SD memory card in the mode specified by "Mode". After opening the file, it outputs the file ID "FileID". Use "FileID" when specifying the file for instructions like FileRead and FileWrite.

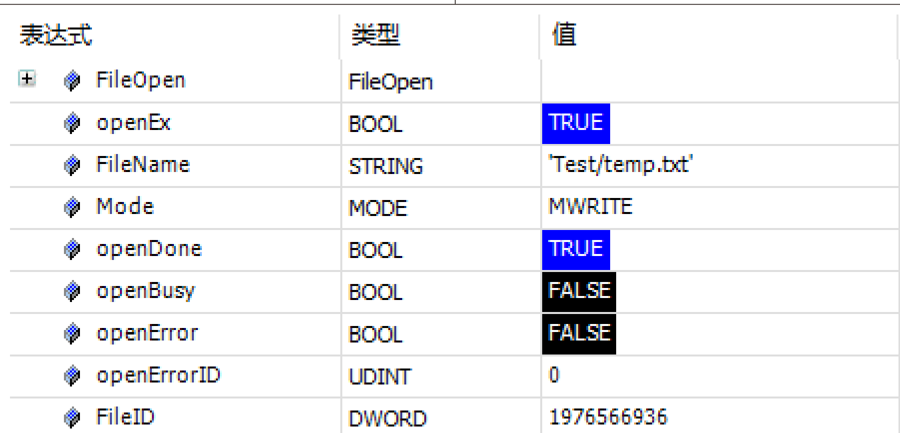
An example is shown below. Opens a file named 'Test/temp.txt' and outputs the file handle to "FileID".

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">

```
VAR
    FileOpen        :FileOpen;
    openEx          :BOOL;
    FileName        :STRING :='Test/temp.txt';
    Mode            :HCFA_OmronUitls.file.MODE;
    openDone        :BOOL;
    openBusy        :BOOL;
    openError       :BOOL;
    openErrorID     :UDINT;
    FileID          :DWORD;
```

</td></tr>
<tr><td>Program</td><td>


</td><td>


</td></tr>
<tr><td>Result</td><td colspan="2">

<table>
<tr><th>表达式</th><th>类型</th><th>值</th></tr>
<tr><td>FileOpen</td><td>FileOpen</td><td></td></tr>
<tr><td>openEx</td><td>BOOL</td><td>TRUE</td></tr>
<tr><td>FileName</td><td>STRING</td><td>'Test/temp.txt'</td></tr>
<tr><td>Mode</td><td>MODE</td><td>MWRITE</td></tr>
<tr><td>openDone</td><td>BOOL</td><td>TRUE</td></tr>
<tr><td>openBusy</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>openError</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>openErrorID</td><td>UDINT</td><td>0</td></tr>
<tr><td>FileID</td><td>DWORD</td><td>1976566936</td></tr>
</table>
</td></tr>
</table>

"Mode" is of the enumeration type _eFOPEN_MODE. The meanings of the enumeration elements are as follows.

Enumeration element	Meaning
MWRITE	Write access. The file will be overwritten or created. Read permission.
MREAD	The file will be opened for reading only. Read/write access permission.
MRDWR	The file will be overwritten or created.
MAPPD	The file will be opened in WRITE mode, but data written will be appended to the end of the file.

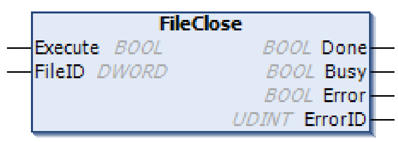
◆ Key points

・ The instruction continues execution to completion even if "Execute" becomes FALSE or the execution time exceeds the task period. Confirm normal completion by checking if the value of "Done" becomes TRUE.

- This instruction must be executed before the FileSeek, FileRead, FileWrite, FileGets, and FilePuts instructions.
- Always execute the FileClose instruction to close a file opened with this instruction after use.
- The value is stored in "FileID" when this instruction completes, i.e., when the value of "Done" changes from FALSE to TRUE.
- If the SD memory card is removed while a file is open, the file remains open. However, reading/writing to the file will not be possible after re-inserting the SD card. To read/write the file, reopen it.
- An exception occurs and "Error" is TRUE under the following conditions:
- When the SD memory card is not in a usable state.
- When the SD memory card is write-protected.
- When "Mode" is AM_READ, AM_APPEND, or AM_READ_PLUS, and the file specified in "FileName" does not exist.
- When the value of "FileName" is an invalid file name.
- When the number of files or directories that can be created is exceeded.
- When the file specified in "FileName" is being accessed.
- When the file specified in "FileName" is read-only.
- When an exception occurs preventing access while the SD memory card is being accessed.

5.14.4 FileClose (File close)

This instruction closes a specified file on the SD memory card.

Instruction	Name	FB/FUN	Graphical representation	ST representation
FileCose	File open	FB		FileClose_instance(Execute, FileID, Done, Busy, Error, ErrorID);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Execute	Function block trigger	Input	Function block trigger.	Conforms to data type.	—	FALSE
FileID	File ID		File handle.			0
Done	Done	Output	Done.			FALSE
Busy	Busy		Busy.			
Error	Error		Error.			
ErrorID	Error ID		Error code.			0

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileID				○																
Done	○																			
Busy	○																			

Error	○																			
ErrorID								○												

◆ Function

It closes the file specified by "FileID" on the SD memory card.

An example is shown below. Closes the file whose file ID is set to the value of the variable FileID.

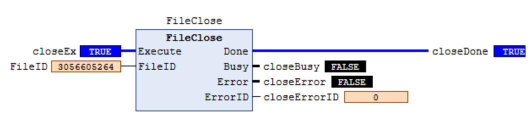
	FBD	ST

Variable declaration

```
PROGRAM PLC_PRG
VAR
    FileOpen        :HCFA_OmronUitls.FileOpen;
    openEx          :BOOL;
    FileName        :STRING:='Test/Temp.txt';
    Mode            :HCFA_OmronUitls.FILE.MODE;
    openDone        :BOOL;
    openBusy        :BOOL;
    openError       :BOOL;
    openErrorID     :UDINT;
    FileID          :DWORD;

    FileClose       :HCFA_OmronUitls.FileClose;
    closeEx         :BOOL;
    closeDone       :BOOL;
    closeBusy       :BOOL;
    closeError      :BOOL;
    closeErrorID    :UDINT;
END_VAR
```

Program

FBD:



ST:

```
FileClose(
    Execute TRUE := closeEx TRUE,
    FileID 3056605264 := FileID 3056605264,
    Done TRUE => closeDone TRUE,
    Busy FALSE => closeBusy FALSE,
    Error FALSE => closeError FALSE,
    ErrorID 0 => closeErrorID 0); RETURN
```

Result

Expression	Type	Value
FileID	DWORD	3056605264
⊞ FileClose	HCFA_Omron...	
closeEx	BOOL	TRUE
closeDone	BOOL	TRUE
closeBusy	BOOL	FALSE
closeError	BOOL	FALSE
closeErrorID	UDINT	0

◆ Key points

・ Do not execute this instruction multiple times consecutively for the same file, as it may cause abnormal PLC operation or even a system crash.

・ The instruction continues execution to completion even if "Execute" becomes FALSE or the execution time exceeds the task period. Confirm normal completion by checking if the value of "Done" becomes TRUE.

・ "FileID" must be obtained beforehand by executing the FileOpen instruction.

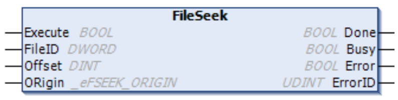
・ Always execute this instruction to close a file opened with the FileOpen instruction after use; otherwise, unexpected results may occur.

・ If the SD memory card is removed while a file is open, the file remains open. However, reading/writing to the file will not be possible after re-inserting the SD card. To read/write the file, reopen it.

- An exception occurs and "Error" is TRUE under the following conditions:
- When the SD memory card is not in a usable state.
- When the file specified in "FileID" is being accessed.
- When the file specified in "FileID" does not exist.
- When an exception occurs preventing access while the SD memory card is being accessed.

5.14.5 FileSeek (File seek)

This instruction sets the file position indicator for a specified file on the SD memory card.

Instruction	Name	FB/FUN	Graphical representation	ST representation
FileSeek	File seek	FB		FileSeek(Execute,FileID, Offset, Origin, Done, Busy, Error, ErrorID);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Execute	Function block trigger	Input	Function block trigger.	Conforms to data type.	—	FALSE
FileID	File ID		File ID for which to set the file position indicator.		—	0
Offset	Offset	Output	Offset position from "Origin".		BYTE	
Origin	Origin		Reference position for the file position indicator.	_SEEK_SET, _SEEK_CUR, _SEEK_END	—	_SEEK_SET
Done	Done		Done.	Conforms to data type.	—	FALSE
Busy	Busy		Busy.			
Error	Error		Error.			
ErrorID	Error ID		Error code.			0

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileID				○																
Offset												○								
ORigin	Enumeration _eFSEEK_ORIGIN. See the Function section for enumeration elements.																			
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

◆ Function

It sets the file position indicator for the file specified by file ID "FileID" on the SD memory card. The file position indicator is the position within the file from which reading or writing starts when instructions like FileRead and FileWrite are executed. For example, to read from the beginning of a file, set the file position indicator to the beginning of the file using the FileSeek instruction, then execute the FileRead instruction. The file position indicator is set to the position calculated by adding the offset "Offset" to the reference position "Origin".

"Origin" is of the enumeration type _eFSEEK_ORIGIN. The meanings of the enumeration elements are as follows.

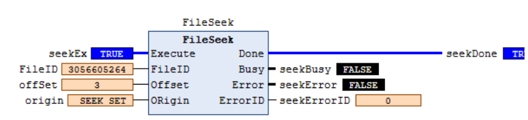
Enumeration element	Meaning
_SEEK_SET	Beginning of the file.
_SEEK_CUR	Current position of the file position indicator.
_SEEK_END	End of the file.

An example is shown below. Sets the file position indicator to a position 3 bytes from the beginning of the file.

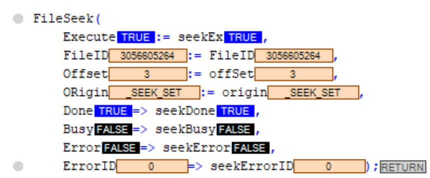
<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">

```
PROGRAM PLC_PRG
VAR
    FileID          :DWORD;
    FileSeek        :HCFA_OmronUitls.FileSeek;
    seekEx          :BOOL;
    offSet          :DINT;
    origin          :HCFA_OmronUitls._eFSEEK_ORIGIN;
    seekDone        :BOOL;
    seekBusy        :BOOL;
    seekError       :BOOL;
    seekErrorID     :UDINT;
```

</td></tr>
<tr><td>Program</td><td>



</td><td>



</td></tr>
<tr><td>Result</td><td colspan="2">

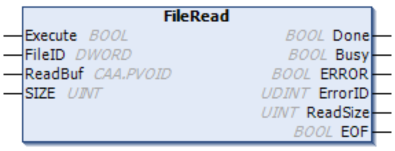
表达式	类型	值	准
FileID	DWORD	3056605264	
FileSeek	HCFA_Omron...		
seekEx	BOOL	TRUE	
offSet	DINT	3	
origin	_EFSEEK_ORI...	_SEEK_SET	
seekDone	BOOL	TRUE	
seekBusy	BOOL	FALSE	
seekError	BOOL	FALSE	
seekErrorID	UDINT	0	

</td></tr>
</table>

E • OmronUtils (Omron Instruction Functions)

5.14.6 FileRead (File read)

This instruction reads data from a specified file on the SD memory card.

Instruction	Name	FB/FUN	Graphical representation	ST representation
FileRead	File read	FB		FileRead(Execute, FileID, ReadBuf, Size, Done, Busy, Error, ErrorID, ReadSize, EOF);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Execute	Function block trigger	Input	Function block trigger.	Conforms to data type.	—	FALSE
FileID	File ID		File ID for which to set the file position indicator.	Conforms to data type.	—	0
Size	Number of elements to read		Number of elements to read.			1
ReadBuf	Read buffer		Destination for read data.	Conforms to data type.	—	—
ReadSize	Actual number of elements read	Output	Actual number of elements read.	Conforms to data type.	—	—
EOF	End of file		Determines if the end of file is reached. TRUE: Reached. FALSE: Not reached.			
Done	Done		Done.	Conforms to data type.	—	FALSE
Busy	Busy		Busy.			
Error	Error		Error.			
ErrorID	Error ID		Error code.			

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileID				○																
Size							○													
ReadBuf	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Arrays of enumerated types or arrays of structures can also be specified.																			
ReadSize							○													
EOF	○																			
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

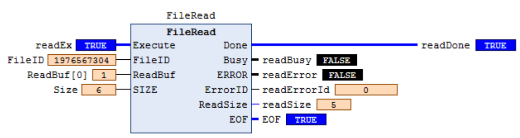
◆ Function

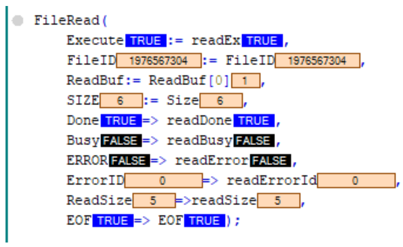
It reads data starting from the current file position indicator of the file specified by file ID "FileID" on the SD memory card and saves it to the read buffer ReadBuf[]. The file position indicator can be set to any position beforehand using the FileSeek instruction. The amount of data to be read is (size of ReadBuf[] data type) × "Size", i.e., "Size" elements of ReadBuf[]. ReadBuf[] can be an array of enumerated types or an array of structures. The actual number of elements read is stored in "ReadSize". Normally, the value of "Size" matches the value of "ReadSize". If the amount of data from the file position indicator to the end of file is less than "Size", no exception occurs; data up to the end of file is saved to ReadBuf[]. In this case, the value of "ReadSize" is less than the value of "Size". Furthermore, if reading reaches the end of file, the end-of-file flag "EOF" becomes TRUE. Otherwise, "EOF" is FALSE.

An example is shown below. Reads a file named 'Test/temp.txt' (the file is first opened in read access mode via the FileOpen instruction to obtain a file handle). ReadBuf[] is set as a BYTE-type array variable with 6 elements, attempting to read 6 BYTEs of data. It reads the byte array previously written to the 'Test/temp.txt' file using the FileWrite instruction. It also reads to the end of the file, and EOF outputs TRUE.

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">

```
FileRead        :FileRead;
readEx          :BOOL;
ReadBuf         :ARRAY [0..5] OF BYTE;
Size            :UINT:=6;
readDone        :BOOL;
readBusy        :BOOL;
readError       :BOOL;
readErrorId     :UDINT;
readSize        :UINT;
EOF             :BOOL;
```

</td></tr>
<tr><td>Program</td><td>


</td><td>


</td></tr>
<tr><td>Result</td><td colspan="2">
<table>
<tr><td>FileRead</td><td>FileRead</td><td></td></tr>
<tr><td>readEx</td><td>BOOL</td><td>TRUE</td></tr>
<tr><td>ReadBuf</td><td>ARRAY [0..5] O...</td><td></td></tr>
<tr><td>ReadBuf[0]</td><td>BYTE</td><td>1</td></tr>
<tr><td>ReadBuf[1]</td><td>BYTE</td><td>5</td></tr>
<tr><td>ReadBuf[2]</td><td>BYTE</td><td>8</td></tr>
<tr><td>ReadBuf[3]</td><td>BYTE</td><td>35</td></tr>
<tr><td>ReadBuf[4]</td><td>BYTE</td><td>44</td></tr>
<tr><td>ReadBuf[5]</td><td>BYTE</td><td>0</td></tr>
<tr><td>Size</td><td>UINT</td><td>6</td></tr>
<tr><td>readDone</td><td>BOOL</td><td>TRUE</td></tr>
<tr><td>readBusy</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>readError</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>readErrorId</td><td>UDINT</td><td>0</td></tr>
</table>
</td></tr>
</table>

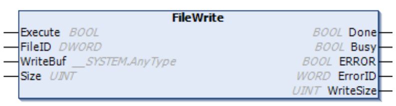
◆ Key points

- The instruction continues execution to completion even if "Execute" becomes FALSE or the execution time exceeds the task period. Confirm normal completion by checking if the value of "Done" becomes TRUE.

- If the value of "Size" exceeds the array bounds of WriteBuf[], a program exception occurs and "Error" becomes TRUE.
- "FileID" must be obtained beforehand by executing the FileOpen instruction.
- When WriteBuf[] is an array of structures, padding areas may be inserted between members depending on the specific structure.
- An exception occurs and "Error" is TRUE under the following conditions:
- When the SD memory card is not in a usable state.
- When the file specified in "FileID" is being accessed.
- When the file specified in "FileID" does not exist.
- When an exception occurs preventing access while the SD memory card is being accessed.
- When the file specified in "FileID" is not opened in a writable mode.

5.14.7 FileWrite (File write)

This instruction writes data to a specified file on the SD memory card.

Instruction	Name	FB/FUN	Graphical representation	ST representation
FileWrite	File write	FB		FileWrite(Execute, FileID, WriteBuf, Size, Done, Busy, Error, ErrorID, WriteSize);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Execute	Function block trigger	Input	Function block trigger.	Conforms to data type.	—	FALSE
FileID	File ID	Input	ID of the file to write to.	Conforms to data type.	—	0
WriteBuf[ ] array	Write buffer		Data to be written.			(*)
Size	Number of elements to write		Number of elements to write.			1
Done	Done	Output	Done.	Conforms to data type.	—	FALSE
Busy	Busy		Busy.			
Error	Error		Error.			
ErrorID	Error ID		Error code.			
WriteSize	Actual number of elements written		Actual number of elements written.			0

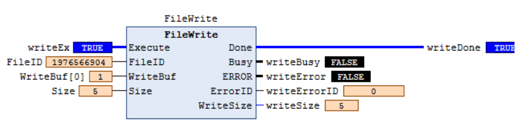
* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileID				○																
WriteBuf[ ] array	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Arrays of enumerated types or arrays of structures can also be specified.																			
Size							○													
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												
WriteSize							○													

◆ Function

It writes data to the file specified by file ID "FileID" on the SD memory card, starting at the current file position indicator. The file position indicator can be set to any position beforehand using the FileSeek instruction. The data to be written is the content of the write buffer WriteBuf[]. The amount of data to be written is (size of WriteBuf[] data type) × "Size", i.e., "Size" elements of WriteBuf[]. WriteBuf[] can be an array of enumerated types or an array of structures. The actual amount of data written is output to "WriteSize".

An example is shown below. Writes the entire array variable WriteBuf[] to a file named 'Test/temp.txt' (the file is first opened in write access mode via the FileOpen instruction to obtain a file handle). WriteBuf[] is set as a BYTE-type array variable with 5 elements.

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">PROGRAM PLC_PRG
VAR
FileID :DWORD;
FileWrite :HCFA_OmronUitls.FileWrite;
writeEx :BOOL;
writeBuff :ARRAY [0..4] OF BYTE;
writeDone :BOOL;
writeBusy :BOOL;
writeError :BOOL;
writeErrorID :BOOL;
writeSize :UINT;</td></tr>
<tr><td>Program</td><td>
</td><td>FileWrite(
Execute TRUE := writeEx TRUE,
FileID 1976566904 := FileID 1976566904,
WriteBuf:= WriteBuf[0] 1,
Size 5 := Size 5,
Done TRUE => writeDone TRUE,
Busy FALSE => writeBusy FALSE,
ERROR FALSE => writeError FALSE,
ErrorID 0 =>writeErrorID 0,
WriteSize 5 => writeSize 5);</td></tr>
</table>

Result

FileWrite	FileWrite	
writeEx	BOOL	TRUE
WriteBuf	ARRAY [0..4] O...	
WriteBuf[0]	BYTE	1
WriteBuf[1]	BYTE	5
WriteBuf[2]	BYTE	8
WriteBuf[3]	BYTE	35
WriteBuf[4]	BYTE	44
writeDone	BOOL	TRUE
writeBusy	BOOL	FALSE
writeError	BOOL	FALSE
writeErrorID	UDINT	0
writeSize	UINT	5

◆ Key points

・The instruction continues execution to completion even if "Execute" becomes FALSE or the execution time exceeds the task period. Confirm normal completion by checking if the value of "Done" becomes TRUE.

・If the value of "Size" exceeds the array bounds of WriteBuf[], a program exception occurs and "Error" becomes TRUE.

・"FileID" must be obtained beforehand by executing the FileOpen instruction.

・When WriteBuf[] is an array of structures, padding areas may be inserted between members depending on the specific structure.

・An exception occurs and "Error" is TRUE under the following conditions:

・When the SD memory card is not in a usable state.

・When the file specified in "FileID" is being accessed.

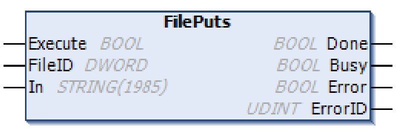
・When the file specified in "FileID" does not exist.

・When an exception occurs preventing access while the SD memory card is being accessed.

・When the file specified in "FileID" is not opened in a writable mode.

5.14.8 FilePuts (File write string)

This instruction writes a string to a specified file on the SD memory card.

Instruction	Name	FB/FUN	Graphical representation	ST representation
FilePuts	File write string	FB		FilePuts(Execute, FileID, In, Done, Busy, Error, ErrorID);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Execute	Function block trigger	Input	Function block trigger.	Conforms to data type.	—	FALSE
FileID	File ID		ID of the file to write to.			0
In	String to write		String to be written.			‘’

Done	Done	Output	Done.	Conforms to data type.	—	FALSE
Busy	Busy		Busy.			
Error	Error		Error.			
ErrorID	Error ID		Error code.			

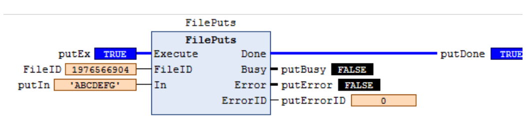
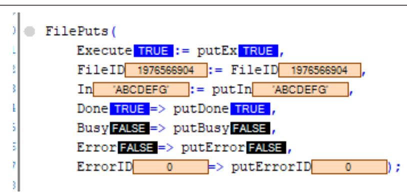
* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileID				○																
In																				○
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

◆ Function

It writes the string "In" to the position of the file position indicator in the file on the SD memory card specified by file ID "FileID". The file position indicator should be set to any position beforehand using the FileSeek instruction.

An example is shown below. Writes "putIn" to a file named 'Test/String.txt' (the file is first opened in write access mode via the FileOpen instruction to obtain a file handle). An example with "putIn" = 'ABCDEFG' is shown below.

	FBD	ST
Variable declaration	FilePuts putEx putIn putDone putBusy putError putErrorID	:FilePuts; :BOOL; :STRING; :BOOL; :BOOL; :BOOL; :UDINT;
Program		
Result		

E
• OmronUtils (Omron Instruction Functions)

◆ Key points

・The instruction continues execution to completion even if "Execute" becomes FALSE or the execution time exceeds the task period. Confirm normal completion by checking if the value of "Done" becomes TRUE.

・If the value of "Size" exceeds the array bounds of WriteBuf[], a program exception occurs and "Error" becomes TRUE.

・"FileID" must be obtained beforehand by executing the FileOpen instruction.

・When WriteBuf[] is an array of structures, padding areas may be inserted between members depending on the specific structure.

・An exception occurs and "Error" is TRUE under the following conditions:

・When the SD memory card is not in a usable state.

・When the file specified in "FileID" is being accessed.

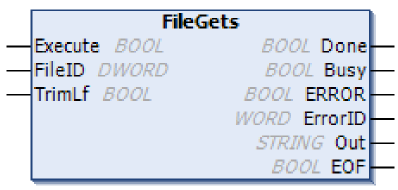
・When the file specified in "FileID" does not exist.

・When an exception occurs preventing access while the SD memory card is being accessed.

・When the file specified in "FileID" is not opened in a writable mode.

5.14.9 FileGets (File read string)

This instruction reads one line of text from a specified file on the SD memory card.

Instruction	Name	FB/FUN	Graphical representation	ST representation
FileGets	File read string	FB		FileGets_instance(Execute, FileID, TrimLF, Done, Busy, Error, ErrorID, Out, EOF);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Execute	Function block trigger	Input	Function block trigger.	Conforms to data type.	—	FALSE
FileID	File ID		ID of the file to read from.			0
TrimLF	Trim line break flag		Flag to trim line breaks from the read string. TRUE: Trims. FALSE: Does not trim.			FALSE
Out	Read string		Read string.	Conforms to data type.	—	—
EOF	End of file	Output	Determines if the end of file is reached. TRUE: Reached. FALSE: Not reached.			
Done	Done		Done.			FALSE
Busy	Busy		Busy.			
Error	Error		Error.			
ErrorID	Error ID		Error code.			—

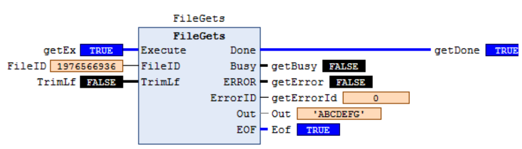
* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileID				○																
TrimLF	○																			
Out																				○
EOF	○																			
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

◆ Function

It reads one line of text starting from the current file position indicator of the file specified by file ID "FileID" on the SD memory card. The file position indicator can be set to any position beforehand using the FileSeek instruction. Line breaks are used to identify the separation between lines. The read string is written to the read string "Out". Line breaks are automatically identified from three types: CR, LF, and CR+LF. When the trim line break flag "TrimLF" is TRUE, the line break is removed from the string before writing to "Out". Furthermore, if reading reaches the end of file, the end-of-file flag "EOF" becomes TRUE. Otherwise, "EOF" is FALSE.

An example is shown below. Reads a file named 'Test/String.txt' and outputs the data to "Out" (the file is first opened in read access mode via the FileOpen instruction to obtain a file handle).

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2"><pre>FileGets   :FileGets;
getEx   :BOOL;
TrimLf   :BOOL;
getDone   :BOOL;
getBusy   :BOOL;
getError   :BOOL;
getErrorId   :UDINT;
Out   :STRING;
Eof   :BOOL;</pre></td></tr>
<tr><td>Program</td><td></td><td><pre>FileGets(
    Execute TRUE := getEx TRUE,
    FileID 1976566936 := FileID 1976566936,
    TrimLf FALSE := TrimLf FALSE,
    Done TRUE => getDone TRUE,
    Busy FALSE => getBusy FALSE,
    ERROR FALSE => getError FALSE,
    ErrorID 0 => getErrorId 0,
    Out 'ABCDEFG' => Out 'ABCDEFG',
    EOF TRUE => Eof TRUE);</pre></td></tr>
</table>

Result	

FileGets	FileGets	
getEx	BOOL	TRUE
TrimLf	BOOL	FALSE
getDone	BOOL	TRUE
getBusy	BOOL	FALSE
getError	BOOL	FALSE
getErrorId	UDINT	0
Out	STRING	'ABCDEFG'
Eof	BOOL	TRUE

◆ Key points

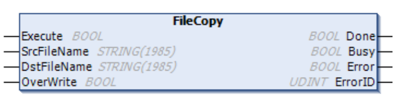
・The instruction continues execution to completion even if "Execute" becomes FALSE or the execution time exceeds the task period. Confirm normal completion by checking if the value of "Done" becomes TRUE.

・If the length of a line exceeds 1985 bytes, the first 1985 bytes of the string (plus the terminating NULL character) are stored in "Out".

- "FileID" must be obtained beforehand by executing the FileOpen instruction.
- An exception occurs and "Error" is TRUE under the following conditions:
- When the SD memory card is not in a usable state.
- When the file specified in "FileID" is being accessed.
- When the file specified in "FileID" does not exist.
- When an exception occurs preventing access while the SD memory card is being accessed.
- When the file specified in "FileID" is not opened in a writable mode.

5.14.10 FileCopy (File copy)

This instruction copies a specified file on the SD memory card.

Instruction	Name	FB/FUN	Graphical representation	ST representation
FileCopy	File copy	FB		FileCopy(Execute, SrcFileName, DstFileName, OverWrite, Done, Busy, Error, ErrorID);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Execute	Function block trigger	Input	Function block trigger.	Conforms to data type.	—	FALSE
SrcFile Name	Source file name		Source file name.	Maximum 66 bytes (65 half-width alphanumeric characters + terminating NULL character).	—	‘’
In	Destination file name		Destination file name.			
OverWrite	Overwrite flag		TRUE: Overwrite allowed. FALSE: Overwrite prohibited.	Conforms to data type.		FALSE

Done	Done	Output	Done.			FALSE
Busy	Busy		Busy.			
Error	Error		Error.			
ErrorID	Error ID		Error code.			—

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

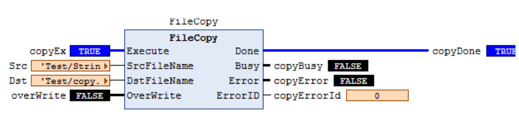
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
SrcFileName																				○
In																				○
OverWrite	○																			
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

◆ Function

It copies the file specified by the source file name "SrcFileName" on the SD memory card to the destination file name "DstFileName". If a file with the same name as "DstFileName" already exists on the SD card, the following processing is performed according to the value of the overwrite flag "OverWrite".

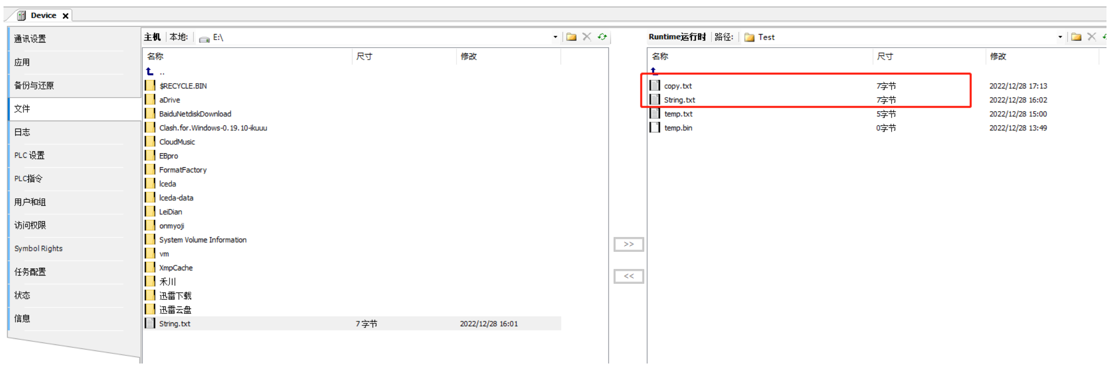
Value of "OverWrite"	Processing
TRUE (Overwrite allowed)	Overwrites the existing file.
FALSE (Overwrite prohibited)	An exception occurs if overwriting is not performed.

An example is shown below. Copies a file named 'Test/String.txt' to a new file named 'Test/copy.txt'.

	FBD	ST
Variable declaration	PROGRAM PLC_PRG VAR FileCopy :FileCopy; copyEx :BOOL; Src :STRING:='Test/String.txt'; Dst :STRING:='Test/copy.txt'; overWrite :BOOL; copyDone :BOOL; copyBusy :BOOL; copyError :BOOL; copyErrorId :UDINT; END_VAR	
Program		FileCopy(Execute TRUE := copyEx TRUE, SrcFileName 'Test/Strin := Src 'Test/Strin, DstFileName 'Test/copy. := Dst 'Test/copy., OverWrite FALSE := overWrite FALSE, Done TRUE => copyDone TRUE, Busy FALSE => copyBusy FALSE, Error FALSE => copyError FALSE, ErrorID 0 => copyErrorId 0);

Result

表达式	类型	值
FileCopy	FileCopy	
copyEx	BOOL	TRUE
Src	STRING	'Test/String.txt'
Dst	STRING	'Test/copy.txt'
overWrite	BOOL	FALSE
copyDone	BOOL	TRUE
copyBusy	BOOL	FALSE
copyError	BOOL	FALSE
copyErrorId	UDINT	0



◆ Key points

• The instruction continues execution to completion even if "Execute" becomes FALSE or the execution time exceeds the task period. Confirm normal completion by checking if the value of "Done" becomes TRUE.

• If the SD memory card is removed while a file is open, the file remains open. However, reading/writing to the file will not be possible after re-inserting the SD card. To read/write the file, reopen it.

• An exception occurs and "Error" is TRUE under the following conditions:

• When the SD memory card is not in a usable state.

• When the SD memory card is write-protected.

• When there is insufficient free space on the SD memory card.

• When the file specified in "SrcFileName" does not exist.

• When the value of "SrcFileName" is an invalid file name.

• When the value of "DstFileName" is an invalid file name.

• When the number of files or directories that can be created is exceeded.

• When a file with the same name as "DstFileName" already exists and is being accessed.

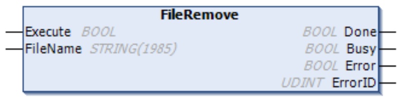
• When a file with the same name as "DstFileName" already exists and the value of "OverWrite" is FALSE.

• When a file with the same name as "DstFileName" already exists and that file is read-only.

• When an exception occurs preventing access while the SD memory card is being accessed.

5.14.11 FileRemove (File delete)

This instruction deletes a specified file on the SD memory card.

Instruction	Name	FB/FUN	Graphical representation	ST representation
FileRemove	File delete	FB		FileRemove(Execute, FileName, Done, Busy, Error, ErrorID);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Execute	Function block trigger	Input	Function block trigger.	Conforms to data type.	—	FALSE
FileName	Specified file name	Input	File name to be deleted.	Maximum 66 bytes (65 half-width alphanumeric characters + terminating NULL character).	—	‘’
Done	Done	Output	Done.	Conforms to data type.	—	FALSE
Busy	Busy	Output	Busy.	Conforms to data type.	—	FALSE
Error	Error	Output	Error.	Conforms to data type.	—	FALSE
ErrorID	Error ID	Output	Error code.	Conforms to data type.	—	FALSE

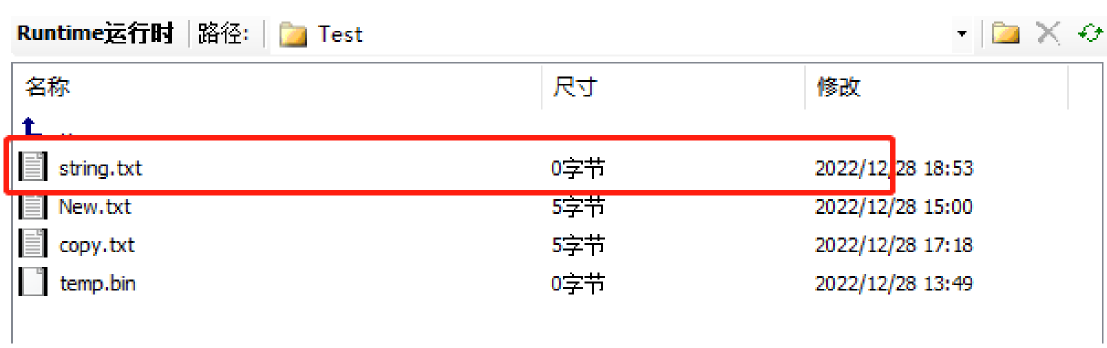
* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

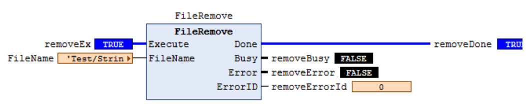
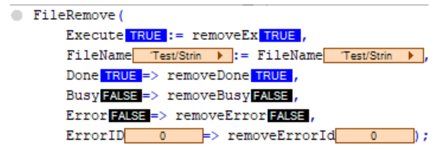
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileName																				○
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

◆ Function

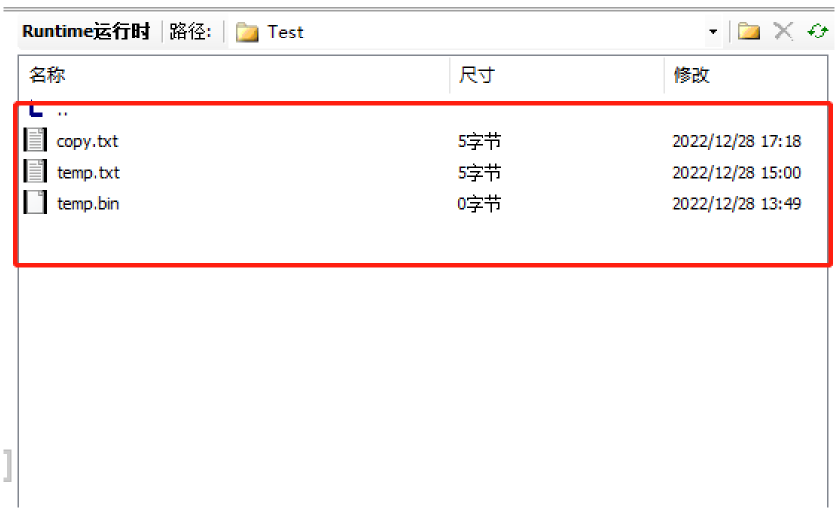
It deletes the file specified by the file name "FileName" on the SD memory card.

An example is shown below. A file named 'Test/String.txt' exists in the PLC. The FileRemove instruction is called to delete this file.



<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2"><pre>
PROGRAM PLC_PRG
VAR
    FileRemove   :FileRemove;
    removeEx   :BOOL;
    FileName   :STRING:='Test/String.txt';
    removeDone   :BOOL;
    removeBusy   :BOOL;
    removeError   :BOOL;
    removeErrorId   :UDINT;
END_VAR
</pre></td></tr>
<tr><td>Program</td><td>
</td><td>
</td></tr>
<tr><td>Result</td><td colspan="2"><table>
<tr><th>表达式</th><th>类型</th><th>值</th></tr>
<tr><td>FileRemove</td><td>FileRemove</td><td></td></tr>
<tr><td>removeEx</td><td>BOOL</td><td>TRUE</td></tr>
<tr><td>FileName</td><td>STRING</td><td>'Test/String.txt'</td></tr>
<tr><td>removeDone</td><td>BOOL</td><td>TRUE</td></tr>
<tr><td>removeBusy</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>removeError</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>removeErrorId</td><td>UDINT</td><td>0</td></tr>
</table></td></tr>
</table>

The file named 'Test/String.txt' has been removed.



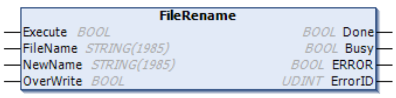
◆ Key points

- The instruction continues execution to completion even if "Execute" becomes FALSE or the execution time exceeds the task period. Confirm normal completion by checking if the value of "Done" becomes TRUE.
- If the SD memory card is removed while a file is open, the file remains open. However, reading/writing to the file will not be possible after re-inserting the SD card. To read/write the file, reopen it.
- An exception occurs and "Error" is TRUE under the following conditions:
- When the SD memory card is not in a usable state.
- When the file specified in "FileID" is being accessed.

- When the file specified in "FileID" does not exist.
- When an exception occurs preventing access while the SD memory card is being accessed.
- When the file specified in "FileID" is not opened in a writable mode.

5.14.12 FileRename (File rename)

This instruction changes the name of a specified file or directory on the SD memory card.

Instruction	Name	FB/FUN	Graphical representation	ST representation
FileRename	File rename	FB		FileRename(Execute, FileName, NewName, OverWrite, Done, Busy, Error, ErrorID);

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Execute	Function block trigger	Input	Function block trigger.	Conforms to data type.	—	FALSE
FileName	Original file name	Input	Original file name.	Maximum 66 bytes (65 half-width alphanumeric characters + terminating NULL character).	—	''
NewName	New file name	Input	New file name.	Maximum 66 bytes (65 half-width alphanumeric characters + terminating NULL character).	—	''
OverWrite	Overwrite flag	Input	TRUE: Overwrite allowed. FALSE: Overwrite prohibited.	Conforms to data type.	—	FLASE
Done	Done	Output	Done.	Conforms to data type.	—	FALSE
Busy	Busy	Output	Busy.	Conforms to data type.	—	FALSE
Error	Error	Output	Error.	Conforms to data type.	—	FALSE
ErrorID	Error ID	Output	Error code.	Conforms to data type.	—	FALSE

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

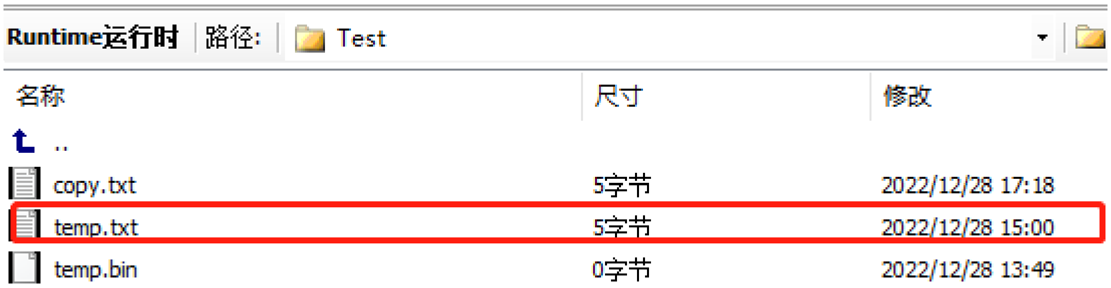
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileName																				○
NewName																				○
OverWrite	○																			
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

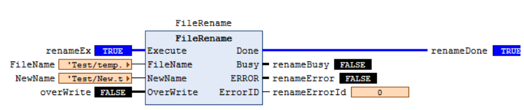
◆ Function

It changes the name of the file or directory specified by the original file name "FileName" on the SD memory card to the new file name "NewName". If a file or directory with the same name as "NewName" already exists on the SD card, the following processing is performed according to the value of the overwrite flag "OverWrite".

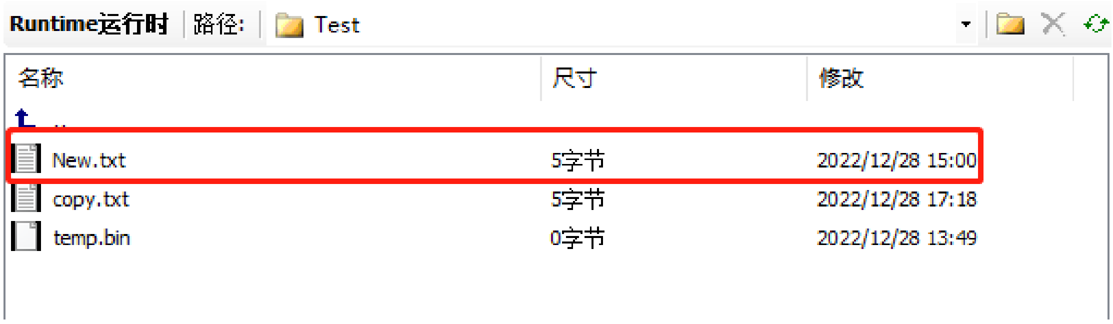
Value of "OverWrite"	Processing
TRUE (Overwrite allowed)	Overwrites the existing file.
FALSE (Overwrite prohibited)	An exception occurs if overwriting is not performed.

An example is shown below. A file named 'Test/temp.txt' exists in a folder. The FileRename instruction is called to change the file name to 'Test/New.txt'.



<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">PROGRAM PLC_PRG
VAR
    FileRename :FileRename;
    renameEx :BOOL;
    FileName :STRING:='Test/temp.txt';
    NewName :STRING:='Test/New.txt';
    overWrite :BOOL;
    renameDone :BOOL;
    renameBusy :BOOL;
    renameError :BOOL;
    renameErrorId :UDINT;
END_VAR</td></tr>
<tr><td>Program</td><td></td><td>FileRename(
    Execute TRUE := renameEx TRUE,
    FileName 'Test/temp. := FileName 'Test/temp.,
    NewName 'Test/New.t := NewName 'Test/New.t,
    OverWrite FALSE := overWrite FALSE,
    Done TRUE => renameDone TRUE,
    Busy FALSE => renameBusy FALSE,
    ERROR FALSE =>renameError FALSE,
    ErrorID 0 => renameErrorId 0);
RETURN</td></tr>
<tr><td>Result</td><td colspan="2"><table>
<tr><th>表达式</th><th>类型</th><th>值</th></tr>
<tr><td>FileRename</td><td>FileRename</td><td></td></tr>
<tr><td>renameEx</td><td>BOOL</td><td>TRUE</td></tr>
<tr><td>FileName</td><td>STRING</td><td>'Test/temp.txt'</td></tr>
<tr><td>NewName</td><td>STRING</td><td>'Test/New.txt'</td></tr>
<tr><td>overWrite</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>renameDone</td><td>BOOL</td><td>TRUE</td></tr>
<tr><td>renameBusy</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>renameError</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>renameErrorId</td><td>UDINT</td><td>0</td></tr>
</table></td></tr>
</table>

In the PLC, the file has been renamed to 'New.txt'.



◆ **Key points**

・The instruction continues execution to completion even if "Execute" becomes FALSE or the execution time exceeds the task period. Confirm normal completion by checking if the value of "Done" becomes TRUE.

・If the directories of "FileName" and "NewName" are different, the file is moved to the directory specified by "NewName".

・If the SD memory card is removed while a file is open, the file remains open. However, reading/writing to the file will not be possible after re-inserting the SD card. To read/write the file, reopen it.

・An exception occurs and "Error" is TRUE under the following conditions:

・When the SD memory card is not in a usable state.

・When the SD memory card is write-protected.

・When the file specified in "FileName" does not exist.

・When the value of "FileName" or "NewName" is an invalid file name.

・When the file specified in "FileName" is being accessed.

・When the number of files or directories that can be created is exceeded.

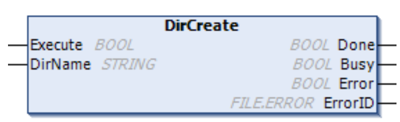
・When a file with the same name as "NewName" already exists and the value of "OverWrite" is FALSE.

・When a file with the same name as "NewName" already exists, the value of "OverWrite" is TRUE, and that file is read-only.

・When an exception occurs preventing access while the SD memory card is being accessed.

5.14.13 DirCreate (Directory create)

The instruction creates a directory with a specified name on the SD memory card.

Instruction	Name	FB/FUN	Graphical representation	ST representation
DirCreate	Directory create	FB		DirCreate(Execute, DirName, Done, Busy, Error, ErrorID);

◆ **Variables**

	Name	Input/Output	Description	Valid range	Unit	Initial value
Execute	Function block trigger	Input	Function block trigger.	Conforms to data type.	—	FALSE
DirName	Directory name	Input	Name of the directory to be created.	Maximum 66 bytes (65 half-width alphanumeric characters + terminating NULL character).	—	''

Done	Done	Output	Done.	Conforms to data type.	—	FALSE
Busy	Busy		Busy.			
Error	Error		Error.			
ErrorID	Error ID		Error code.			—

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
DirName																				○
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

◆ Function

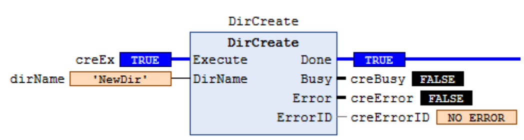
It creates a directory on the SD memory card with the name specified by the directory name "DirName".

An example is shown below. The DirCreate instruction is called to create a new directory named 'NewDir' in the root directory of the PLC.

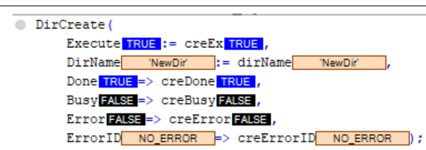
<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>Variable declaration</td><td colspan="2">

```
PROGRAM PLC_PRG
VAR
    DirCreate       :DirCreate;
    creEx           :BOOL;
    dirName         :STRING :='NewDir';
    creDone         :BOOL;
    creBusy         :BOOL;
    creError        :BOOL;
    creErrorID      :HCFA_OmronUitls.FILE.ERROR;
```

</td></tr>
<tr><td>Program</td><td>



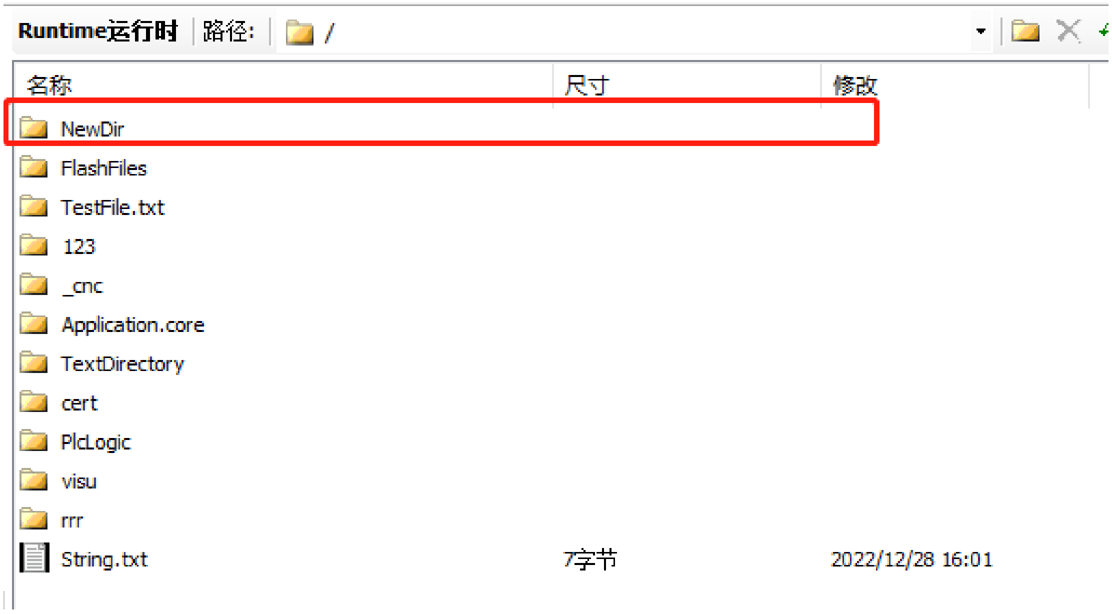
</td><td>



</td></tr>
<tr><td>Result</td><td colspan="2">

表达式	类型	值
DirCreate	DirCreate	
creEx	BOOL	TRUE
dirName	STRING	'NewDir'
creDone	BOOL	TRUE
creBusy	BOOL	FALSE
creError	BOOL	FALSE
creErrorID	ERROR	NO_ERROR

</td></tr>
</table>



◆ Key points

• The instruction continues execution to completion even if "Execute" becomes FALSE or the execution time exceeds the task period. Confirm normal completion by checking if the value of "Done" becomes TRUE.

• If the SD memory card is removed while a file is open, the file remains open. However, reading/writing to the file will not be possible after re-inserting the SD card. To read/write the file, reopen it.

• An exception occurs and "Error" is TRUE under the following conditions:

• When the SD memory card is not in a usable state.

• When the SD memory card is write-protected.

• When there is insufficient free space on the SD memory card.

• When the number of directories that can be created is exceeded.

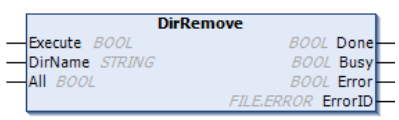
• When the directory specified in "DirName" already exists.

• When the value of "DirName" is an invalid directory name.

• When an exception occurs preventing access while the SD memory card is being accessed.

5.14.14 DirRemove (Directory remove)

This instruction deletes a specified directory on the SD memory card.

Instruction	Name	FB/FUN	Graphical representation	ST representation
DirRemove	Directory remove	FB		DirRemove(Execute, DirName, All, Done, Busy, Error, ErrorID)

◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
Execute	Function block trigger	Input	Function block trigger.	Conforms to data type.	—	FALSE
DirName	Directory name		Name of the directory to be deleted.	Maximum 66 bytes (65 half-width alphanumeric characters + terminating NULL character).	—	‘’
ALL	All flag		Specifies handling when files/ subdirectories exist in the directory. TRUE: Deletes along with files/ subdirectories. FALSE: Does not delete.	Conforms to data type.		FASLE
Done	Done	Output	Done.	Conforms to data type.	—	FALSE |
Busy	Busy		Busy.			
Error	Error		Error.			
ErrorID	Error ID		Error code.			

* Initial value is not applicable if an input parameter is omitted. An exception occurs during linking.

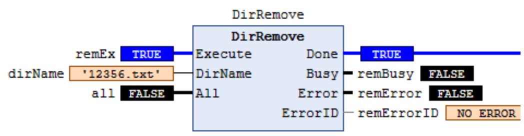
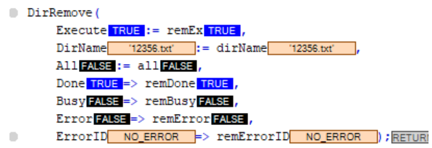
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
DirName																				○
All	○																			
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

◆ Function

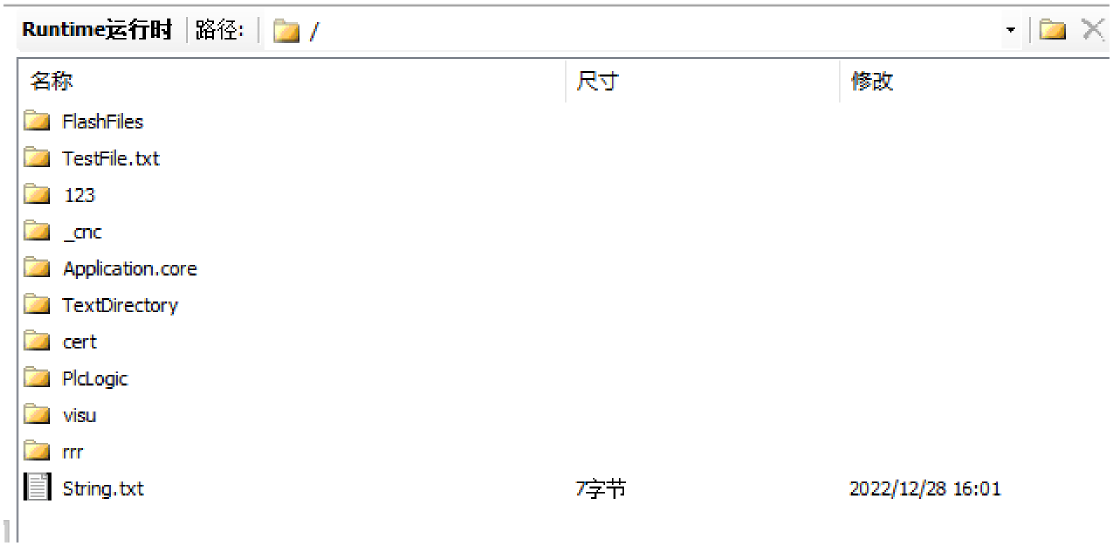
It deletes the directory specified by the directory name "DirName" on the SD memory card. If files or subdirectories exist within the specified directory, the following processing is performed according to the value of the "All" flag.

Value of "All"	Processing
TRUE	Deletes the specified directory along with all files and subdirectories.
FALSE	Does not delete the specified directory, causing an exception.

An example is shown below. A folder named '12356.txt' exists in the root directory of the PLC. The DirRemove instruction is called to delete this folder.

	FBD	ST
Variable declaration	DirRemove :DirRemove; remEx :BOOL; all :BOOL; remDone :BOOL; remBusy :BOOL; remError :BOOL; remErrorID :HCFA_OmronUitls.FILE.ERROR;	
Program		
Result	DirRemove DirRemove remEx BOOL TRUE all BOOL FALSE remDone BOOL TRUE remBusy BOOL FALSE remError BOOL FALSE remErrorID ERROR NO_ERROR	

In the PLC, the folder '12356.txt' has been removed.



◆ Key points

• The instruction continues execution to completion even if "Execute" becomes FALSE or the execution time exceeds the task period. Confirm normal completion by checking if the value of "Done" becomes TRUE.

• If the SD memory card is removed while a file is open, the file remains open. However, reading/writing to the file will not be possible after re-inserting the SD card. To read/write the file, reopen it.

• An exception occurs and "Error" is TRUE under the following conditions:

• When the SD memory card is not in a usable state.

• When the SD memory card is write-protected.

• When the value of "All" is TRUE and the directory specified in "DirName" is being accessed by another instruction.

• When the value of "All" is FALSE and files or directories exist within the directory specified in "DirName".

• When the directory specified in "DirName" is read-only.

• When the file specified in "DirName" does not exist.

• When an exception occurs preventing access while the SD memory card is being accessed.

5.15 Hexadecimal character conversion instructions

5.15.1 HexStringToNum_ (Hexadecimal string to integer)

FC_ByteToStrHex: Converts a hexadecimal numeric value represented by a Byte to a single hexadecimal character.

FC_StrHexToByte: Converts a single hexadecimal character to a decimal number, represented by a Byte.

HexStringToNum_DINT: Hexadecimal string to DINT integer.

HexStringToNum_INT: Hexadecimal string to INT integer.

HexStringToNum_LINT: Hexadecimal string to LINT integer.

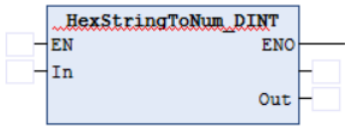
HexStringToNum_SINT: Hexadecimal string to SINT integer.

HexStringToNum_UDINT: Hexadecimal string to UDINT integer.

HexStringToNum_UINT: Hexadecimal string to UINT integer.

HexStringToNum_ULINT: Hexadecimal string to ULINT integer.

HexStringToNum_USINT: Hexadecimal string to USINT integer.

Instruction	Name	FB/FUN	Graphical representation	ST representation
HexStringToNum_***	Hexadecimal string to integer	FUN		HexStringToNum_DINT(In:= , Out=>);

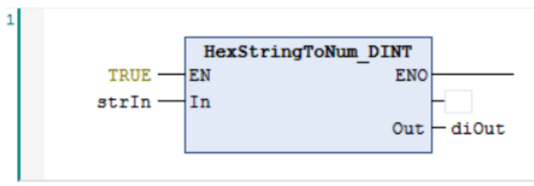
◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Hexadecimal string	Input		Conforms to data type.	—	—
Out	Integer	Output			—	—

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out						○	○	○	○	○	○	○	○							

It converts the hexadecimal format string "In" to an integer. The instruction name varies depending on the specific data type. The HexStringToNum_DINT instruction is used as an example below:

LD:



ST:

```
strIn        :STRING;
diOut        :DINT;

HexStringToNum_DINT(In:=strIn , Out=>diOut );
```

strIn:=

					1	1	1

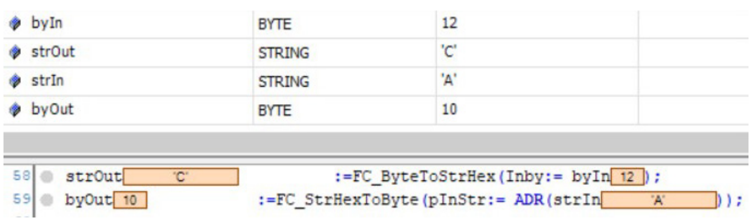
diOut:=DINT#273

strIn:=

				-	1	1	1

diOut:=DINT#-273

Examples for the FC_ByteToStrHex and FC_StrHexToByte functions are as follows:



◆ Key points

・If the conversion result exceeds the valid range of "Out", no exception is displayed; the displayed value will be an invalid number.

・If a non-hexadecimal character is entered, no exception is displayed; the displayed value will be an invalid number.

5.16 Sequential I/O instructions

5.16.1 TestABit (Bit test)

This instruction outputs the value of a specified bit in a bit field.

Instruction	Name	FB/FUN	Graphical representation	ST representation
TestABit	Bit test	FUN		Out :=TestABit(In:= ,usiPos:=);

◆ Variables

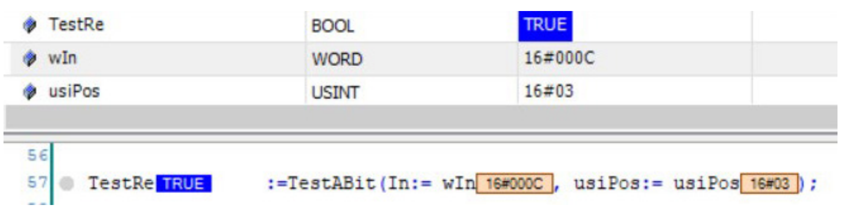
	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Bit field	Input	Bit field.	Conforms to data type.		(*)
usiPos	Bit position	Input	Position of the specified bit.	Number of bits, starting from bit 0.		0
Out	Return value		Return value of TestABit.	Conforms to data type.		

	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
usiPos						○														
Out	○																			

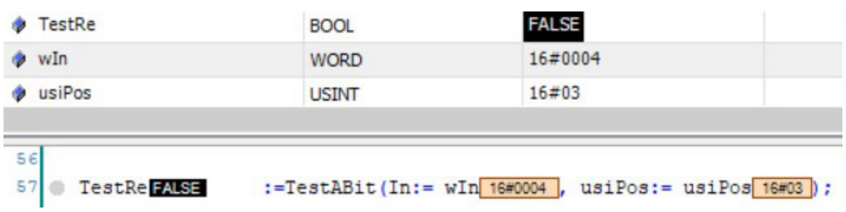
◆ Function

It returns TRUE/FALSE for the value at the "Pos" position of the bit field "In" via TestABit.

Example: Input is a WORD-type data with value 16#C (2#1100), "usiPos" reads bit 3, and the return value is TRUE.



Example: Input is a WORD-type data with value 16#4 (2#0100), "usiPos" reads bit 3, and the return value is FALSE.



5.16.2 SetABit/ResetABit (Set a bit / Reset a bit)

SetABit: Sets the specified bit in a bit field to TRUE.

ResetABit: Sets the specified bit in a bit field to FALSE.

Instruction	Name	FB/FUN	Graphical representation	ST representation
SetABit	Set a bit	FUN		Out := SetABit(In:= , usiPos:=);
ResetABit	Reset a bit	FUN		Out := ResetABit(In:= , usiPos:=);

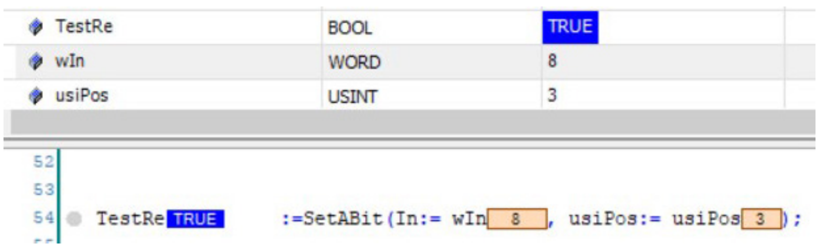
◆ Variables

	Name	Input/Output	Description	Valid range	Unit	Initial value
In	Bit field	Input	Bit field.	Conforms to data type.		(*)
usiPos	Bit position		Position of the specified bit.	Number of bits, starting from bit 0.		0
Out	Return value		Return value of SetABit/ResetABit.	Conforms to data type.		

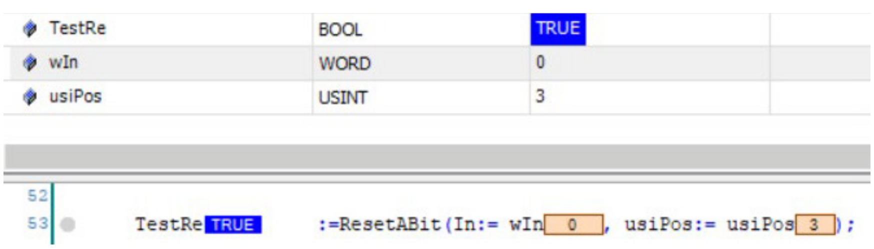
	BOOL	Bit string				Integer								Real		Time, duration, date, string				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
usiPos						○														
Out	○																			

◆ Function

Using the SetABit function, it sets bit 3 of wIn to TRUE. The result is shown in the figure below:



Using the ResetABit function, it sets bit 3 of wIn to FALSE. The result is shown in the figure below:



◆ Key points

- The function return value is TRUE upon successful completion.

Chapter 6 Standard Library

6.1 CheckDevice (Function group)

It is used to check if the controller is a product of HCFA. The following functions are similar; choose between the function block or function form based on preference.

6.1.1 Function block FB_CheckPAC (FB)

Name	FB_CheckPAC (PAC detection)		
Supported modes	CSP	CSV	CST
Graphical representation		ST representation	
		FB_CheckPAC(eResult=>);	

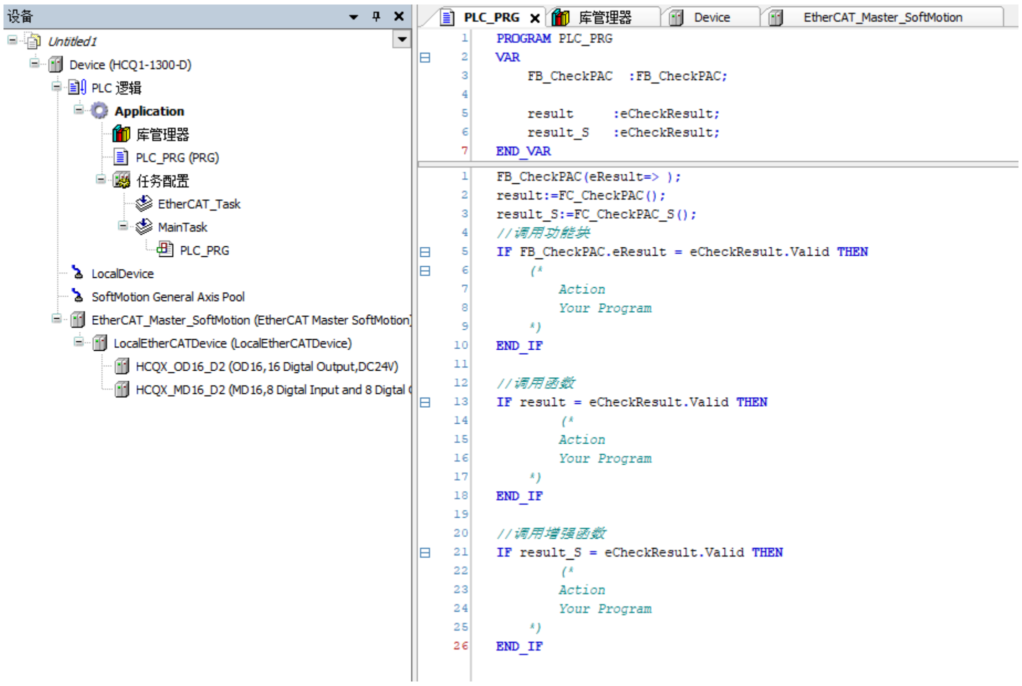
6.1.2 Function form

Name	Description
FC_CheckPAC (FUN)	Standard function for checking HCFA controller.
FC_CheckPAC_S (FUN)	Enhanced function for checking the HCFA controller. A fault mechanism is activated if the controller is not from HCFA.

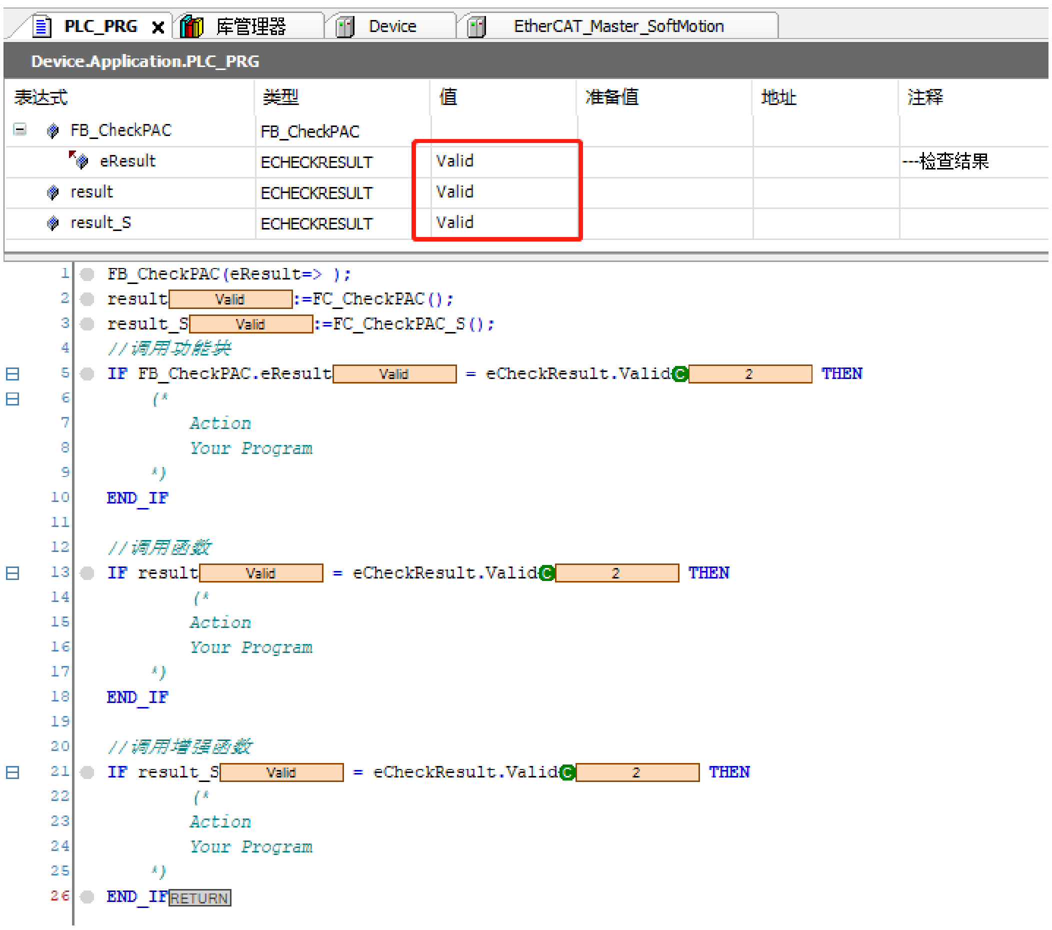
6.1.3 eCheckResult (ENUM)

Status	Type	Value	Description
NotCalled	INT	0	Not called.
Checking	INT	1	Checking.
Valid	INT	2	Controller valid.
Invalid	INT	3	Controller invalid.

6.1.4 Usage example



The three methods achieve the same result; choose based on preference.



6.2 LockMachine (Function group)

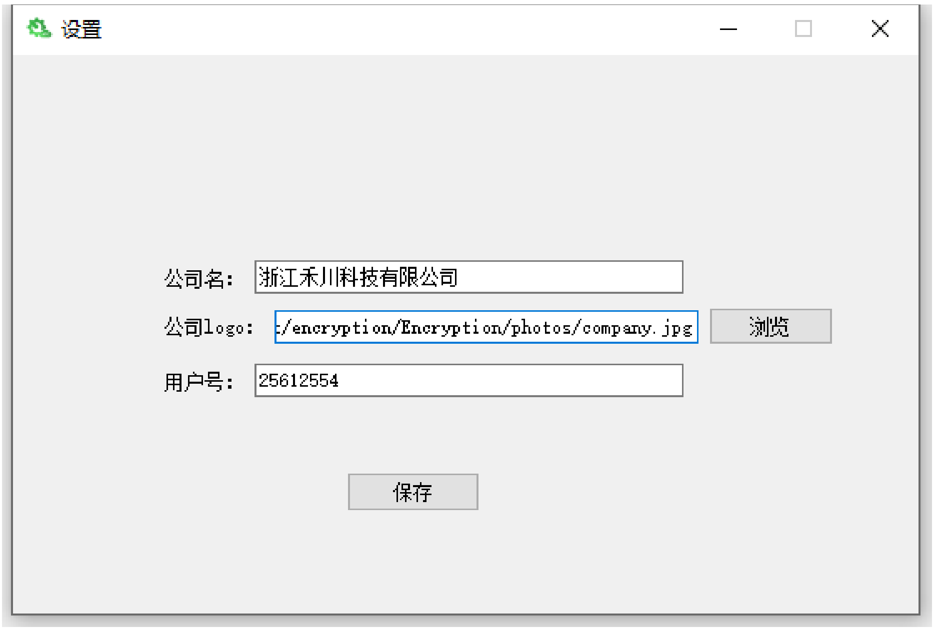
It requires the HCFA encryption software to work with CODESYS to unlock the PLC machine.

6.2.1 Main function description

6.2.1.1 Interface introduction

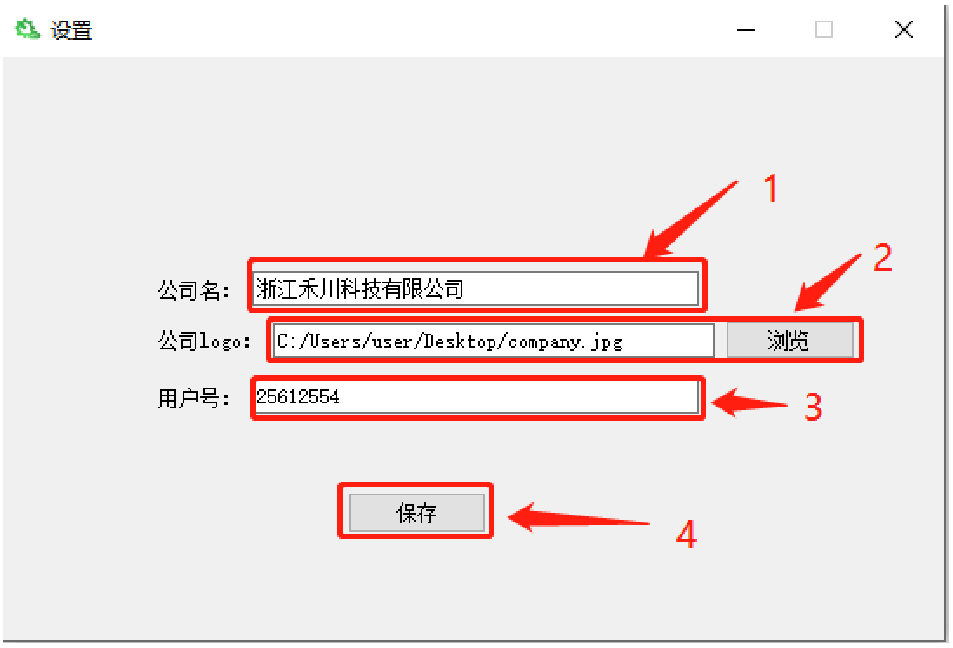
Main interface

Settings interface

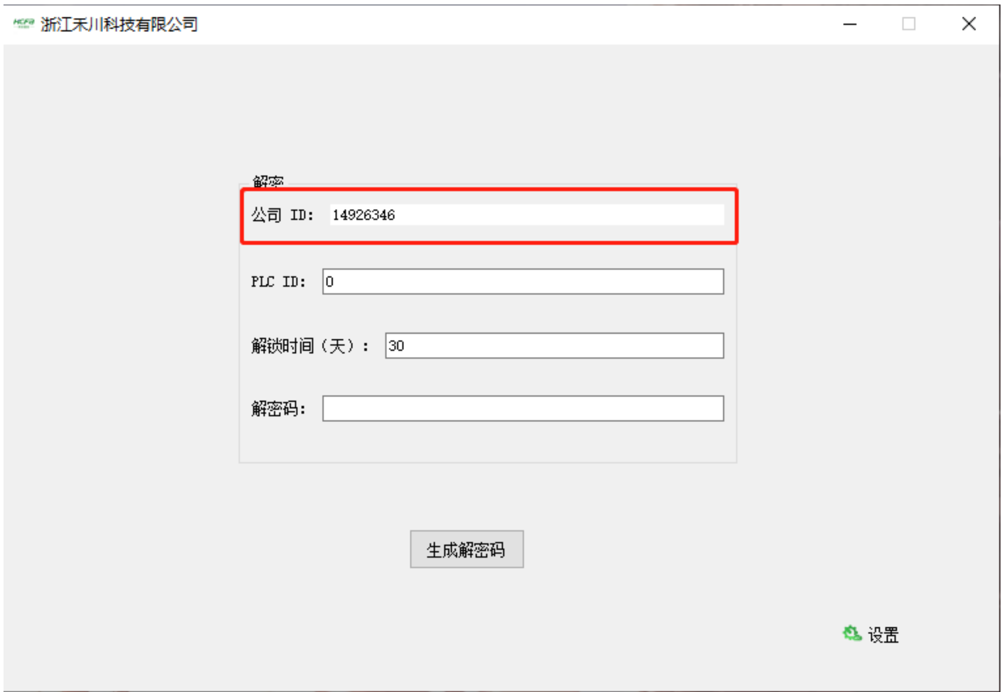


6.2.1.2 Software initialization settings

For first-time use, it is recommended to modify the software name and icon, and input the user number information. Using HCFA as an example, click [Settings] in the opened settings interface. Input the company name, company logo image, and user number. Click [Save] to complete the modifications. The main interface will then randomly generate a company ID.



The main interface after modification is shown in the figure below:



6.2.1.3 Obtaining the unlock code and unlocking the PLC

Enter the randomly generated PLC ID (obtained after CODESYS connects to the PLC) into the "PLC ID" text box on the main interface, and set the unlock duration (in days).

For example, the PLC ID generated by the function block in CODESYS is 55604, and a 30-day unlock is required.

Click the [Generate Unlock Code] button. An unlock code will be generated in the "Unlock Code" field, e.g., 233492974.

Save the unlock code. Simply input the unlock code into the KEY input of the FB_LockMachine function block in CODESYS, and trigger a rising edge on the bCheckKey input. The system will automatically verify the code. If correct, the PLC will be unlocked for 30 days. After 30 days, the PLC automatically locks and requires unlocking again. See the figure below.

6.2.2 CODESYS usage instructions

6.2.2.1 Function block introduction

FB_LockMachine

A function block used to determine if the PLC is locked and to unlock the PLC. A brief introduction follows.

ST example:

FB_LockMachine(

bCheckKey:=Execute , → Trigger signal

Key:= KEY, → KEY is used to input the unlock code generated by the software.

bKeyCorrect=> , → Unlock code correct signal.

bKeyWrong=> , → KEY is used to input the unlock code generated by the software.

MachineValid=> , → Machine valid status signal

bUnLocked=> , → Machine permanently unlocked signal.

PLCID=>PLCID , → Randomly generated ID after connecting to PLC; input to the unlock software to generate the unlock code.

dRemainingDay=>lastDay); → Remaining usable days of the machine.

FB_getRTCDate

FB_getRTCDate(

Enable:= , → Trigger signal

dwDate=> , → Date

wYear=> , → Year

wMonth:= , → Month

wDay:= , → Day

wHour:= , → Hour

wMinute:= , → Minute

wSecond:= , → Second

bError=>); → Error signal

FB_setRTCDate

A function block used to set the PLC system time. A brief introduction follows.

ST example:

FB_setRTCDate(

Execute:=, → Trigger signal

wYear:= , → Year

wMonth:= , → Month

wDay:= , → Day

```
wHour:= ,                 → Hour
wMinute:= ,               → Minute
wSecond:= ,               → Second
Done=> , );               → Time setting completion signal.
```

6.2.2.2 Adding the CODESYS project

Create a new CODESYS project file. Declare and call the three function blocks FB_LockMachine, FB_getRTCDate, and FB_setRTCDate in a POU. For debugging convenience, some variables declared in the example program are provided for reference below:

POU Device

```
PROGRAM POU
VAR
    fb   :FB_LockMachine;
    fb_setdate   :FB_setRTCDate;
    fb_get   :FB_getRTCDate;

    UserID:UDINT:=96366123;//用户ID
    KEY:UDINT;//解密码

    timeRTC:DATE;//系统时间
    lastDay:DWORD;//剩余天数
    locked:BOOL;//锁机信号
    PLCID:UDINT;//PLCID,每次解锁完随机更换一次

    daytime:DATE:=D#2022-5-26;
    first:BOOL;
    Execute:BOOL;
```

Declare function blocks

The example project declaration and call are shown in the figure below.

POU Device

```
PROGRAM POU
VAR
    fb   :FB_LockMachine;
    fb_setdate   :FB_setRTCDate;
    fb_get   :FB_getRTCDate;
```

Call function blocks

```
fb(
    bCheckKey:=Execute ,
    Key:= KEY,
    bKeyCorrect=> ,
    bKeyWrong=> ,
    MachineValid=> ,
    bUnLocked=> ,
    PLCID=>PLCID ,
    dRemainingDay=>lastDay);

fb_setdate(
    wYear:= ,
    wMonth:= ,
    wDay:= ,
    wHour:= ,
    wMinute:= ,
    wSecond:= ,
    Done=> , );
```

```
fb_get(
    Enable:= ,
    dwDate=> ,
    wYear=> ,
    wMonth=> ,
    wDay=> ,
    wHour=> ,
    wMinute=> ,
    wSecond=> ,
    bError=> );
```

For first-time use, a function to set the expiration date and company ID must be added.

First-time use requires writing two data items, daytime and UserID, into the function block.

In the example program, define two variables, UserID(UDINT) and daytime(DATE), in a POU. Use these variables to pass the required values to the function block's fb. vendorID and fb.dueTimeinputs. Refer to the figure below for the specific program code.

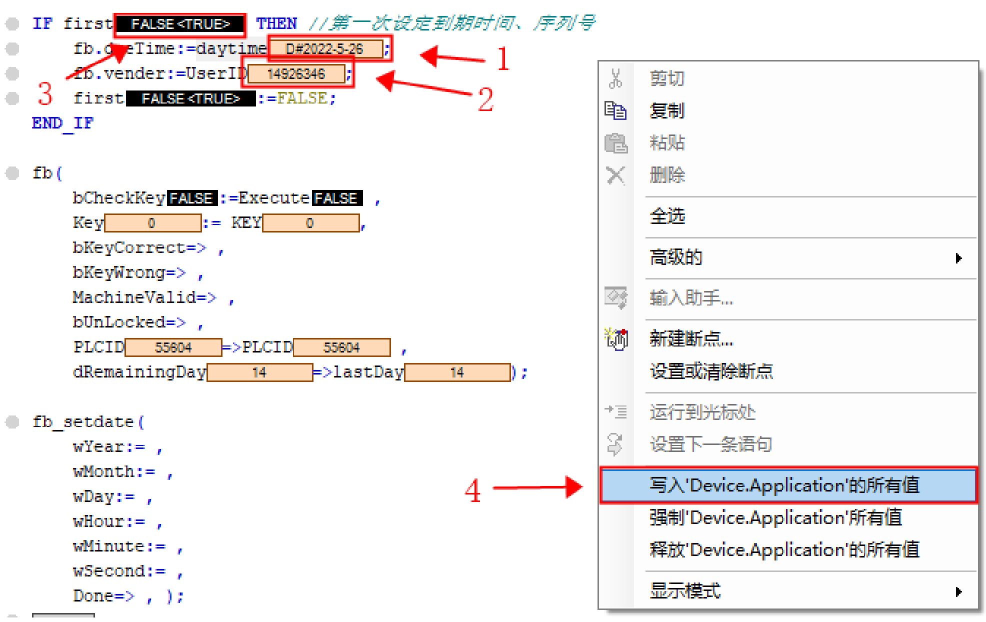
```
POU
PROGRAM POU
VAR
    fb  :FB_LockMachine;
    fb_setdate  :FB_setRTCDate;
    fb_get  :FB_getRTCDate;

    UserID:UDINT:=14926346;//用户ID
    daytime:DATE:=D#2022-5-26;

IF first THEN //第一次设定到期时间、序列号
    fb.dueTime:=daytime;
    fb.venderID:=UserID;
    first:=FALSE;
END_IF
```

Click Login, download the program to the PLC, and run the program.

6.2.2.3 Detailed usage instructions

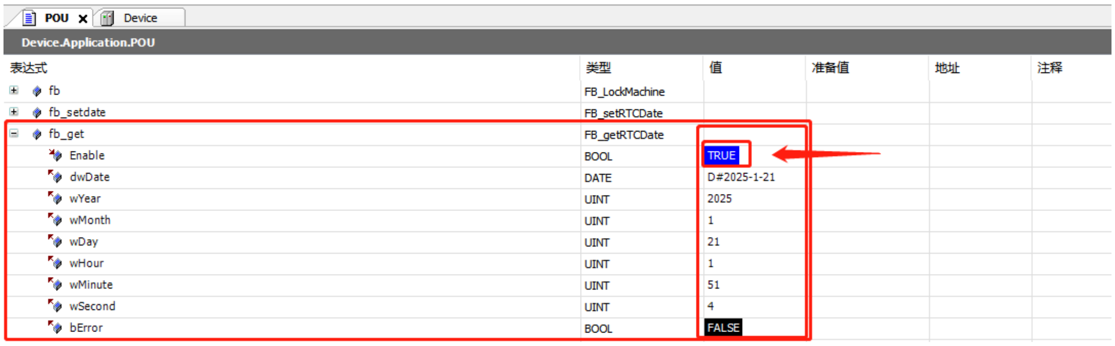
First-time use

First-time use requires an IF statement to write the "Company ID" (randomly generated on the unlock software's main interface) and set the "First Expiration Time". For example: Expiration time: D#2022-5-26, Company ID: 14926346. In the example program, triggering first after writing these two pieces of information completes the write operation.



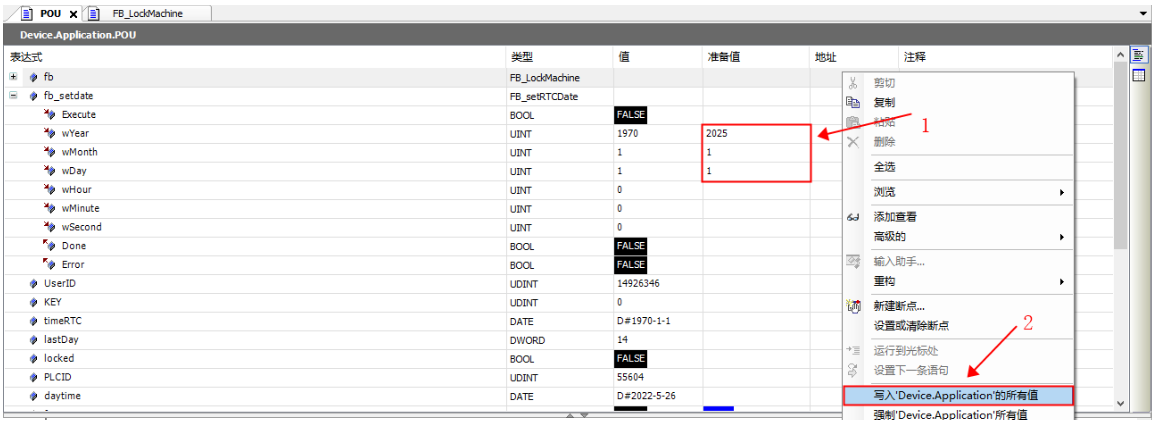
Reading PLC System time using FB_getRTCDate

In the FB_getRTCDate function block, trigger the Enable input to read the PLC's system time. In the example program, the read system time is D#2025-1-21.

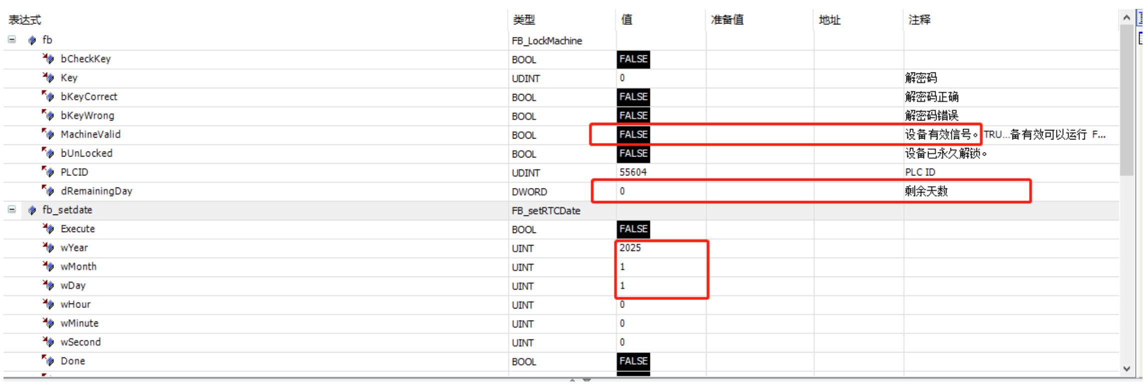


Modifying PLC system time using FB_setRTCDate

This example modifies the current system time to 2025-1-1.

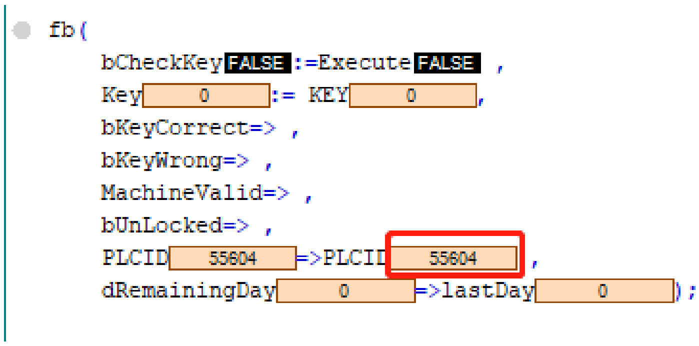


After the modification, it can be seen that the FB_LockMachine function block's MachineValid signal is FALSE, and the remaining days are 0. The PLC is locked and unusable.



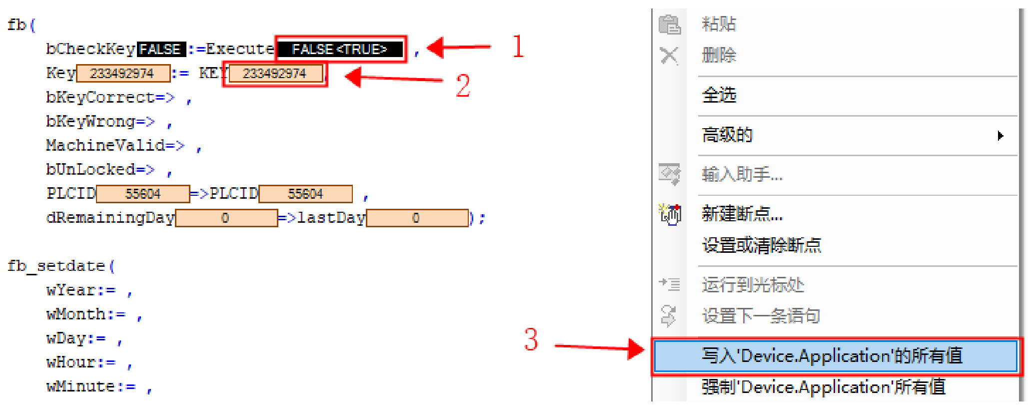
Unlocking the PLC using FB_LockMachine

After connecting to the PLC and running the program, the function block automatically generates a PLC ID, as shown below:



Enter the PLC ID code into the unlock software to generate the unlock code:

As previously mentioned in section 8.2.1.3, a 30-day unlock code 233492974 was generated for the PLC machine with ID 55604. Input this code into the function block's KEY input, and trigger a rising edge on the bCheckKey input to complete the activation.



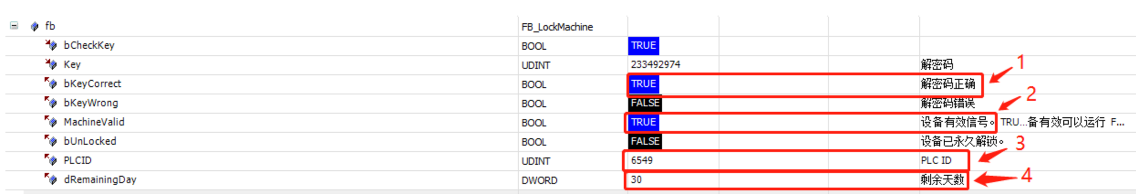
Four changes in the function block can be observed:

[1] The unlock code correct signal (bKeyCorrect) becomes TRUE. (If the code is wrong, bKeyCorrect remains FALSE, and the unlock code wrong signal (bKeyWrong) becomes TRUE).

[2] The machine valid signal (MachineValid) becomes TRUE. The device is unlocked.

[3] The PLC ID updates. (A new random PLC ID is generated after each unlock).

[4] The remaining days update. (30 days were activated here, so the remaining time updates to 30 days).



This completes one PLC unlock operation. When the PLC expires, simply repeat the process.

6.2.3 Common issues

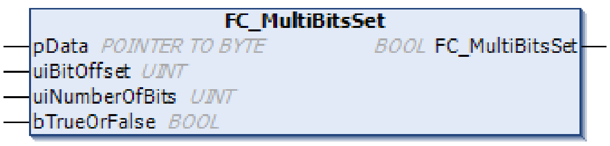
6.2.3.1 Accidental system time setting causing machine lock

Exercise caution when using the FB_setRTCDate function block to set the system time. Note that the input date must not be set to an earlier time. For example, if the read system time is 2022-5-29, setting it to 2022-5-28 will lock the PLC, requiring reactivation.

6.3 FC_MultiBitsSet (FUN)

It sets the state of multiple consecutive bits to TRUE or FALSE.

◆ Variable

Name	FC_MultiBitsSet		
Supported modes	CSP	CSV	CST
Graphical representation		ST representation	
		FC_MultiBitsSet(pData:= , uiBitOffset:= , uiNumberOfBits:= , bTrueOrFalse:=);	

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
pData	Start address	POINTER TO BYTE			Start address.
uiBitOffset	Offset	UINT		0	Bit offset, specifies from which bit to start operation.
uiNumberOfBits	Operand	UINT		1	Number of bits to operate on.
bTrueOrFalse	Target value	BOOL	TRUE, FALSE	FALSE	Whether to write TRUE or FALSE to each bit.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
FC_MultiBitsSet	Done	BOOL	TRUE, FALSE	TRUE: Function block execution completed.

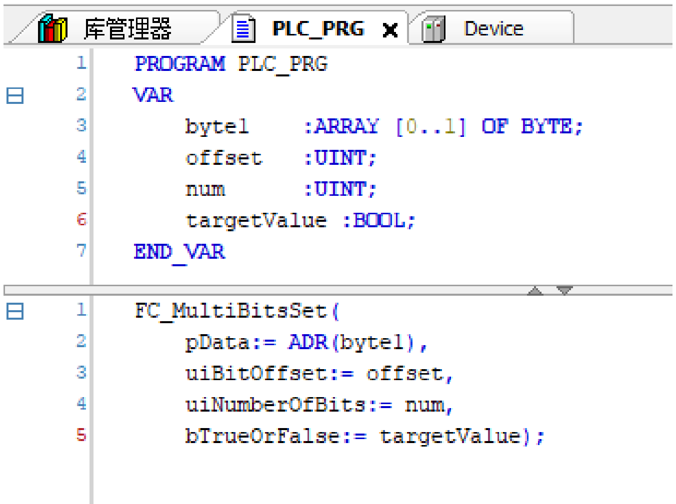
◆ Key points

・ pData is a pointer type. When addressing an actual physical address, do not use a format like %QX10.3; use %QB10 instead. This is because %QX10.3 ultimately points to the same address as %QX10.0, which can be misleading.

・ The function block starts modifying in real-time upon execution. The value set on the bTrueOrFalse pin is the value written to the specified number of bits starting at the set address.

6.3.1 Usage example 1 (Modifying virtual addresses)

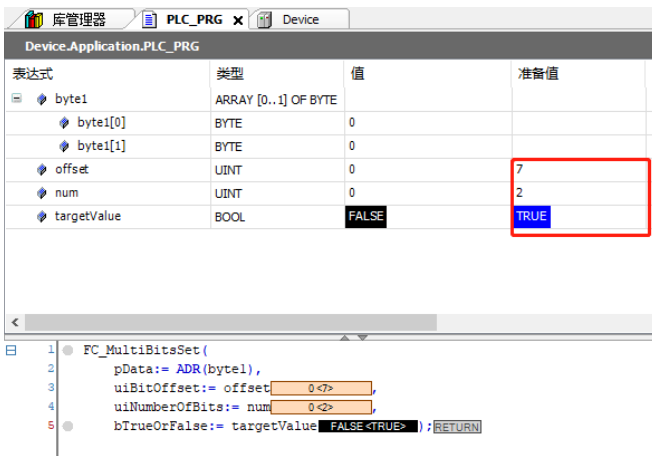
[1] Define a BYTE array and corresponding UINT, BOOL variables, then call the function.




```
PROGRAM PLC_PRG
VAR
    byte1      :ARRAY [0..1] OF BYTE;
    offset     :UINT;
    num        :UINT;
    targetValue :BOOL;
END_VAR
```

```
FC_MultiBitsSet(
    pData:= ADR(byte1),
    uiBitOffset:= offset,
    uiNumberOfBits:= num,
    bTrueOrFalse:= targetValue);
```

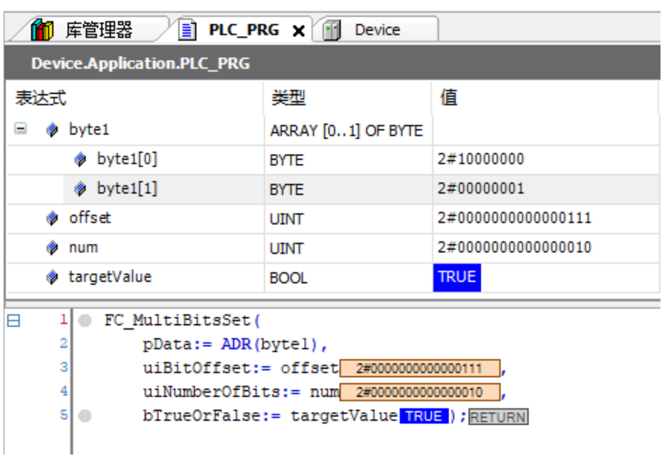
[2] Example: Write TRUE to 2 bits starting from Bit7 of the start address.



表达式	类型	值	准备值
byte1	ARRAY [0..1] OF BYTE		
byte1[0]	BYTE	0	
byte1[1]	BYTE	0	
offset	UINT	0	7
num	UINT	0	2
targetValue	BOOL	FALSE	TRUE

```
FC_MultiBitsSet(
    pData:= ADR(byte1),
    uiBitOffset:= offset[0 <7>],
    uiNumberOfBits:= num[0 <2>],
    bTrueOrFalse:= targetValue[FALSE <TRUE>]);RETURN
```

The result is shown below. For easier viewing, the data display mode is switched to binary. Bit7 and Bit8 have been set to TRUE (the left side is the high-order bit in a BYTE).



表达式	类型	值
byte1	ARRAY [0..1] OF BYTE	
byte1[0]	BYTE	2#10000000
byte1[1]	BYTE	2#00000001
offset	UINT	2#0000000000000111
num	UINT	2#0000000000000010
targetValue	BOOL	TRUE

```
FC_MultiBitsSet(
    pData:= ADR(byte1),
    uiBitOffset:= offset[2#0000000000000111],
    uiNumberOfBits:= num[2#0000000000000010],
    bTrueOrFalse:= targetValue[TRUE]);RETURN
```

6.3.2 Usage example 2 (Modifying physical addresses)

[1] Define a BYTE array (QB0 is the starting address for the local IO outputs of the HCFA Q1 controller) and corresponding UINT, BOOL variables, then call the function.

库管理器 | PLC_PRG | Device | GeneralIO

```
PROGRAM PLC_PRG
VAR
    realbyte   AT%QB0  :BYTE;
    offset     :UINT;
    num        :UINT;
    targetValue :BOOL;
END_VAR
```

```
FC_MultiBitsSet(
    pData:= ADR(realbyte),
    uiBitOffset:= offset,
    uiNumberOfBits:= num,
    bTrueOrFalse:= targetValue);
```

[2] Example: Write TRUE to 2 bits starting from Bit7 of the start address.

库管理器 | PLC_PRG | Device

Device.Application.PLC_PRG

表达式	类型	值	准备值
byte1	ARRAY [0..1] OF BYTE		
byte1[0]	BYTE	0	
byte1[1]	BYTE	0	
offset	UINT	0	7
num	UINT	0	2
targetValue	BOOL	FALSE	TRUE

```
FC_MultiBitsSet(
    pData:= ADR(byte1),
    uiBitOffset:= offset[0 <7>],
    uiNumberOfBits:= num[0 <2>],
    bTrueOrFalse:= targetValue[FALSE <TRUE>]);[RETURN]
```

The result is shown below. For easier viewing, the data display mode is switched to binary. Bit7 and Bit8 have been set to TRUE (the left side is the high-order bit in a BYTE).

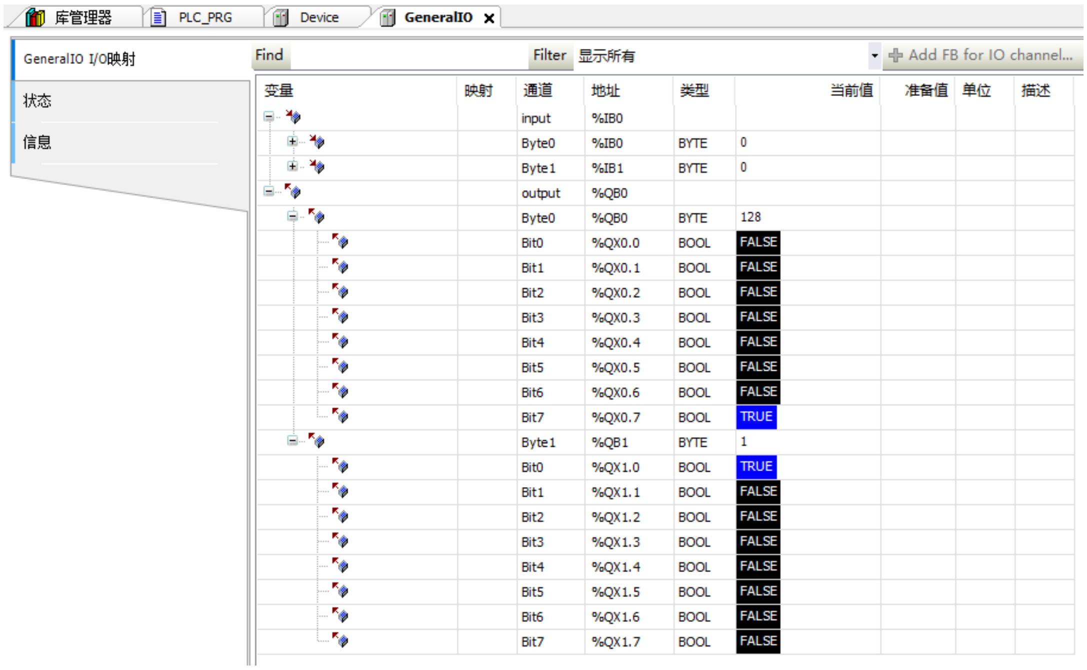
库管理器 | PLC_PRG | Device

Device.Application.PLC_PRG

表达式	类型	值
byte1	ARRAY [0..1] OF BYTE	
byte1[0]	BYTE	2#10000000
byte1[1]	BYTE	2#00000001
offset	UINT	2#0000000000000111
num	UINT	2#0000000000000010
targetValue	BOOL	TRUE

```
FC_MultiBitsSet(
    pData:= ADR(byte1),
    uiBitOffset:= offset[2#0000000000000111],
    uiNumberOfBits:= num[2#0000000000000010],
    bTrueOrFalse:= targetValue[TRUE]);[RETURN]
```

In General IO, the Q1 output terminals QX0.7 and QX1.0 are set to TRUE, meaning Bit7 and Bit8 are set to TRUE.



变量	映射	通道	地址	类型	当前值	准备值	单位	描述
		input	%IB0					
		Byte0	%IB0	BYTE	0			
		Byte1	%IB1	BYTE	0			
		output	%QB0					
		Byte0	%QB0	BYTE	128			
		Bit0	%QX0.0	BOOL	FALSE			
		Bit1	%QX0.1	BOOL	FALSE			
		Bit2	%QX0.2	BOOL	FALSE			
		Bit3	%QX0.3	BOOL	FALSE			
		Bit4	%QX0.4	BOOL	FALSE			
		Bit5	%QX0.5	BOOL	FALSE			
		Bit6	%QX0.6	BOOL	FALSE			
		Bit7	%QX0.7	BOOL	TRUE			
		Byte1	%QB1	BYTE	1			
		Bit0	%QX1.0	BOOL	TRUE			
		Bit1	%QX1.1	BOOL	FALSE			
		Bit2	%QX1.2	BOOL	FALSE			
		Bit3	%QX1.3	BOOL	FALSE			
		Bit4	%QX1.4	BOOL	FALSE			
		Bit5	%QX1.5	BOOL	FALSE			
		Bit6	%QX1.6	BOOL	FALSE			
		Bit7	%QX1.7	BOOL	FALSE			

6.4 RAND (Function group)

Random number generation functions, includes three specific functions. See each function's description for details.

Name	Description
RAND (FUN)	Generates a random number.
RAND_LIMIT (FUN)	Generates a random number with upper and lower limits.
RAND_neg (FUN)	Generates a signed random number.

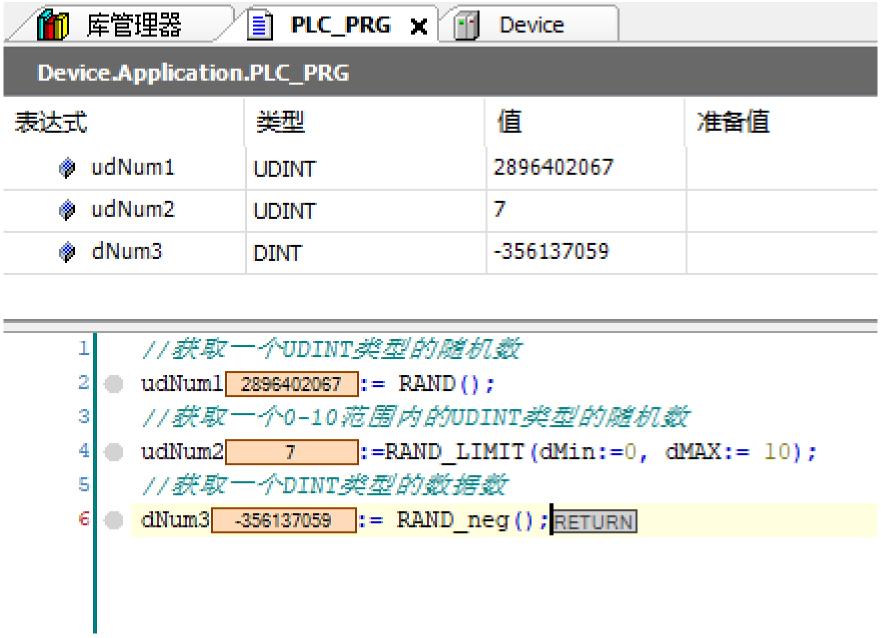
6.4.1 Usage example

库管理器 PLC_PRG

```
PROGRAM PLC_PRG
VAR
    udNum1  :UDINT;
    udNum2  :UDINT;
    dNum3   :DINT;
END_VAR
```

```
//获取一个UDINT类型的随机数
udNum1:= RAND();
//获取一个0-10范围内的UDINT类型的随机数
udNum2:=RAND_LIMIT(dMin:=0, dMAX:= 10);
//获取一个DINT类型的数据数
dNum3:= RAND_neg();
```

The functions continuously output random numbers of the corresponding type.

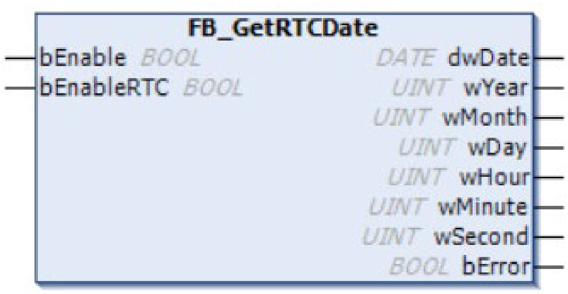


6.5 RTCTime (Function group)

PLC RTC clock read and modify function group.

6.5.1 Read function block FB_GetRTCDate (FB)

◆ Variable

Name	FB_GetRTCDate (FB) (Get RTC time)		
Supported modes	CSP	CSV	CST
Graphical representation		ST representation	
		FB_GetRTCDate(bEnable:= , bEnableRTC:= , dwDate=> , wYear=> , wMonth=> , wDay=> , wHour=> , wMinute=> , wSecond=> , bError=>);	

(1) Input variable

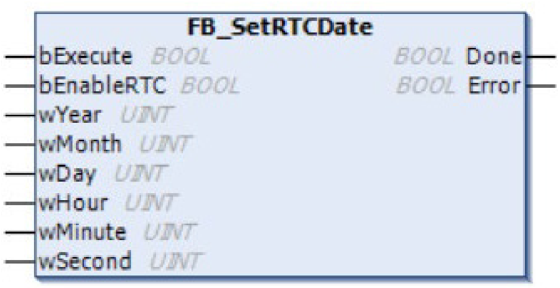
Input variable	Name	Data type	Valid range	Initial value	Description
bEnable	Enable	BOOL	TRUE, FALSE	FALSE	TRUE: Enables the function block. FALSE: Disables the function block.
bEnableRTC	Time zone select pin	BOOL	TRUE, FALSE	TRUE	TRUE: Reads local time. FALSE: Reads UTC time.

(2) Output variable

Output variable	Name	Data type	Valid range	Initial value	Description
dwDate	Date	DATE			Date
wYear	Year	UINT		1970	Year
wMonth	Month	UINT		1	Month
wDay	Day	UINT		1	Day
wHour	Hour	UINT			Hour
wMinute	Minute	UINT			Minute
wSecond	Second	UINT			Second
bError	Error	BOOL	TRUE, FALSE		TRUE: An exception occurred in the function block; execution stopped.

6.5.2 Set Function block FB_SetRTCDate (FB)

Name	FB_SetRTCDate (FB) (Set RTC time)		
Supported modes	CSP	CSV	CST
Graphical representation		ST representation	




```
FB_SetRTCDate(
        bExecute:= ,
        bEnableRTC:= ,
        wYear:= ,
        wMonth:= ,
        wDay:= ,
        wHour:= ,
        wMinute:= ,
        wSecond:= ,
        Done=> ,
        Error=> );
```

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
Execute	Enable	BOOL	TRUE, FALSE	FALSE	TRUE: Enables the function block. FALSE: Disables the function block.
bEnableRTC	Time zone select pin	BOOL	TRUE, FALSE	TRUE	TRUE: Sets local time. FALSE: Sets UTC time.
wYear	Year	UINT		1970	Year
wMonth	Month	UINT		1	Month
wDay	Day	UINT		1	Day
wHour	Hour	UINT			Hour
wMinute	Minute	UINT			Minute
wSecond	Second	UINT			Second

(2) Output variable

Output variable	Name	Data type	Valid range	Description
bDone	Done	BOOL	TRUE, FALSE	TRUE: Function block execution completed.
bError	Error	BOOL	TRUE, FALSE	TRUE: An exception occurred in the function block; execution stopped.

6.5.3 Usage example

[1] Declare and call the function blocks FB_GetRTCDate and FB_SetRTCDate.

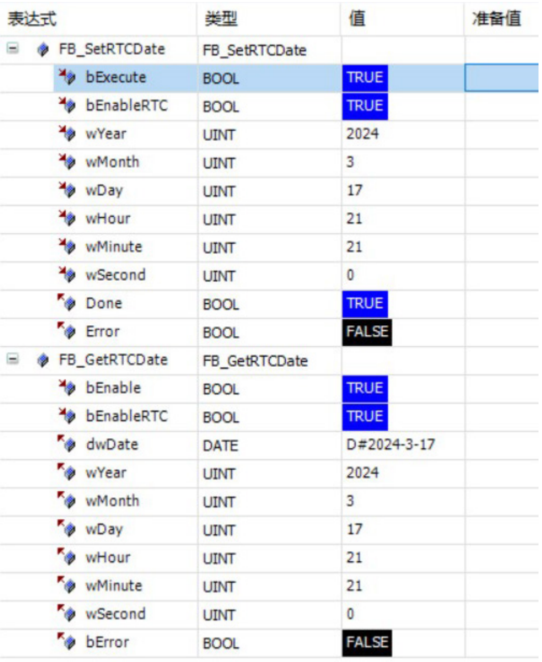
```
VAR
    FB_GetRTCDate       :FB_GetRTCDate;
    FB_SetRTCDate    :FB_SetRTCDate;
END_VAR
```

Connect to the PLC and the program. Enable the FB_SetRTCDate function block. After a successful write, read the current UTC time in the PLC via FB_GetRTCDate.

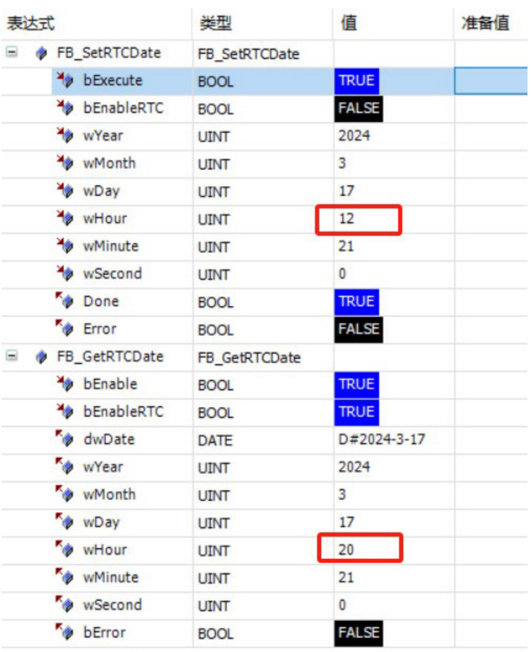
表达式	类型	值	准备值
FB_GetRTCDate	FB_GetRTCDate		
bEnable	BOOL	TRUE	
bEnableRTC	BOOL	TRUE	
dwDate	DATE	D#2024-3-17	
wYear	UINT	2024	
wMonth	UINT	3	
wDay	UINT	17	
wHour	UINT	21	
wMinute	UINT	16	
wSecond	UINT	21	
bError	BOOL	FALSE	

表达式	类型	值	准备值
FB_GetRTCDate	FB_GetRTCDate		
bEnable	BOOL	TRUE	
bEnableRTC	BOOL	FALSE	
dwDate	DATE	D#2024-3-17	
wYear	UINT	2024	
wMonth	UINT	3	
wDay	UINT	17	
wHour	UINT	13	
wMinute	UINT	17	
wSecond	UINT	15	
bError	BOOL	FALSE	

Enable the FB_SetRTCDate function block. After a successful write, read the current RTC time in the PLC via FB_GetRTC-Date.



表达式	类型	值	准备值
FB_SetRTCDate	FB_SetRTCDate		
bExecute	BOOL	TRUE	
bEnableRTC	BOOL	TRUE	
wYear	UINT	2024	
wMonth	UINT	3	
wDay	UINT	17	
wHour	UINT	21	
wMinute	UINT	21	
wSecond	UINT	0	
Done	BOOL	TRUE	
Error	BOOL	FALSE	
FB_GetRTCDate	FB_GetRTCDate		
bEnable	BOOL	TRUE	
bEnableRTC	BOOL	TRUE	
dwDate	DATE	D#2024-3-17	
wYear	UINT	2024	
wMonth	UINT	3	
wDay	UINT	17	
wHour	UINT	21	
wMinute	UINT	21	
wSecond	UINT	0	
bError	BOOL	FALSE	



表达式	类型	值	准备值
FB_SetRTCDate	FB_SetRTCDate		
bExecute	BOOL	TRUE	
bEnableRTC	BOOL	FALSE	
wYear	UINT	2024	
wMonth	UINT	3	
wDay	UINT	17	
wHour	UINT	12	
wMinute	UINT	21	
wSecond	UINT	0	
Done	BOOL	TRUE	
Error	BOOL	FALSE	
FB_GetRTCDate	FB_GetRTCDate		
bEnable	BOOL	TRUE	
bEnableRTC	BOOL	TRUE	
dwDate	DATE	D#2024-3-17	
wYear	UINT	2024	
wMonth	UINT	3	
wDay	UINT	17	
wHour	UINT	20	
wMinute	UINT	21	
wSecond	UINT	0	
bError	BOOL	FALSE	

Key points

- It is not recommended to place the FB_SetRTCDate and FB_GetRTCDate function blocks in motion control tasks like EtherCatTask. Setting the time can cause blocking, leading to errors.

6.6 Filter instructions (Function group)

6.6.1 ArithmeticAverageFilter (Arithmetic mean filter)

Name	FB_ArithmeticAverageFilter (Arithmetic mean filter)		
Supported modes	CSP	CSV	CST
Graphical representation		ST representation	
		FB_ArithmeticAverageFilter(bEnable:= , fSample:= , uiSampleNum:= , uiSampleCycle:= , bBusy=> , bValid=> , bError=> , eError=> , fValue=>);	

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bEnable	Function block enable	BOOL	TRUE, FALSE	FALSE	TRUE: Function block executes; FALSE: Does not execute.
fSample	Input sample value	REAL		0	Input sample value.
uiSampleNum	Input sample value	UINT	1-1000		Input sample value.
uiSampleCycle	Input sample cycle	UINT	1-1000	0	Input sample cycle.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
bBusy	Instruction executing	BOOL	TRUE, FALSE	TRUE: Function block is running.
bValid	Output valid	BOOL	TRUE, FALSE	TRUE: Instruction execution is valid.
bError	Error	BOOL	TRUE, FALSE	TRUE: An exception occurred.
eErrorID	Error ID	WSTRING		
fValue	Filtered output valid value	REAL		Filtered output valid value.

◆ Key points

• When the bEnable level is enabled, given the number of scan cycles uiSampleCycle, the sampling count uiSampleNum, and the input sampling variable fValue. It continuously samples uiSampleNum times and performs an arithmetic mean calculation.

• A larger uiSampleNum value results in higher signal smoothness but lower sensitivity.

• A smaller uiSampleNum value results in lower signal smoothness but higher sensitivity.

• Selection of uiSampleNum value: For general flow, uiSampleNum=12; for pressure: uiSampleNum=4.

• The input parameter uiSampleCycle in the filter function block is the number of sampling cycles. uiSampleCycle=1 means sampling occurs every cycle; uiSampleCycle=3 means sampling occurs once every 3 cycles.

• The sampled data is stored in an array. The user must handle the data themselves, extracting the data from the array one by one.

6.6.2 DebounceFilter (Debounce filter)

<table>
<tr><th>Name</th><th colspan="3">FB_DebounceFilter (Debounce filter)</th></tr>
<tr><td>Supported modes</td><td>CSP</td><td>CSV</td><td>CST</td></tr>
<tr><td colspan="2">Graphical representation</td><td colspan="2">ST representation</td></tr>
<tr><td colspan="2">
</td><td colspan="2">FB_DebounceFilter(
bEnable:= ,
fSample:= ,
uiUpLimit:= ,
fReference:= ,
uiSampleCycle:= ,
bBusy=> ,
bValid=> ,
bError=> ,
ErrorID=> ,
fVaule=>);</td></tr>
</table>

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bEnable	Function block enable	BOOL	TRUE, FALSE	FALSE	TRUE: Function block executes; FALSE: Does not execute.
fSample	Input sample value	REAL			Input value to be filtered.
uiUpLimit	Set filter counter upper limit	REAL	1-65535		Set the filter counter upper limit.
fReference	Input reference value	REAL			Input reference value.
uiSampleCycle	Input reference value	UINT	1-1000		Input sample cycle.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
bBusy	Instruction executing	BOOL	TRUE, FALSE	TRUE: Function block is executing.
bValid	Output valid	BOOL	TRUE, FALSE	TRUE: Instruction execution is valid.
bError	Error	BOOL	TRUE, FALSE	TRUE: An exception occurred.
eErrorID	Error ID	WSTRING		
fValue	Filtered output valid value	LREAL		Filtered output valid value.

◆ Key points

・ When the bEnable level is enabled, given the number of scan cycles uiSampleCycle, the sampling reference value fReference, and a filter upper limit uiUpLimit, input the sampling variable fSample. Compare each sample value with the current valid value:

・ If sample value = current valid value, then the current count value is cleared.

・ If sample value <> current valid value, then the count value increments by 1, and it checks if the count overflows (count > upper limit uiUpLimit).

・ If the count overflows, the current sample value replaces the current filter value output, and the count is cleared.

・ It provides good filtering for slowly changing measured parameters, but is less effective for rapidly changing parameters.

6.6.3 FirstOrderLagFilter (First-order Llg filter)

Name	FB_FirstOrderLagFilter (First-order lag filter)		
Supported modes	CSP	CSV	CST

Graphical representation	ST representation
	FB_FirstOrderLagFilter(bEnable:= , fSample:= , fCoefficient:= , fReference:= , uiSampleCycle:= , bBusy=> , bValid=> , bError=> , ErrorID=> , fVaule=>);

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bEnable	Function block enable	BOOL	TRUE, FALSE	FALSE	TRUE: Function block executes; FALSE: Does not execute.
fSample	Input sample value	REAL		0	Input value to be filtered.
fCoefficient	Input first-order low-pass filter coefficient a=0 ~1	REAL	0-1		Input first-order low-pass filter coefficient.
fReference	Input valid value	REAL		0	Input valid value.
uiSampleCycle	Input sample cycle	UINT	1-1000	0	Input sample cycle.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
bBusy	Instruction executing	BOOL	TRUE, FALSE	TRUE: Function block is executing.
bValid	Output valid	BOOL	TRUE, FALSE	TRUE: Instruction execution is valid.
bError	Error	BOOL	TRUE, FALSE	TRUE: An exception occurred.
eErrorID	Error ID	WSTRING		
fValue	Filtered output valid value	LREAL		Filtered output valid value.

◆ Key points

• When the bEnable level is enabled, given the number of scan cycles uiSampleCycle, the sampling reference value fReference, the first-order low-pass filter coefficient fCoefficient (range 0~1), and the sampling variable fSample.

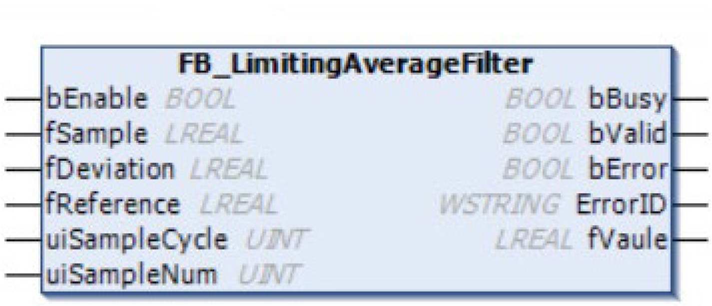
• Calculation formula: This filter result = fCoefficient * This sample value + (1 - fCoefficient) * Last filter result

• Suitable for applications with high-frequency fluctuation and periodic interference. However, it suffers from phase lag and low sensitivity, and cannot eliminate interference signals with a frequency higher than half the sampling frequency.

6.6.4 LimitingAverageFilter (Limiting average filter)

Name	FB_LimitingAverageFilter (Limiting average filter)		
Supported modes	CSP	CSV	CST

Graphical representation



ST representation

```
FB_LimitingAverageFilter(
    bEnable:= ,
    fSample:= ,
    fDeviation:= ,
    fReference:= ,
    uiSampleCycle:= ,
    uiSampleNum:= ,
    bBusy=> ,
    bValid=> ,
    bError=> ,
    ErrorID=> ,
    fVaule=> );
```

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bEnable	Function block enable	BOOL	TRUE, FALSE	FALSE	TRUE: Function block executes; FALSE: Does not execute.
fSample	Input sample value	REAL		0	Input value to be filtered.
fDeviation	Input maximum allowed deviation between two samples	REAL		0	Input maximum allowed deviation between two samples.
fReference	Input valid value	REAL		0	Input valid value.
uiSampleCycle	Input sample cycle	UINT	1-1000	0	Input sample cycle.
uiSampleNum	Input sample count	UINT	1-1000	0	Input sample count.

(2) Output variable

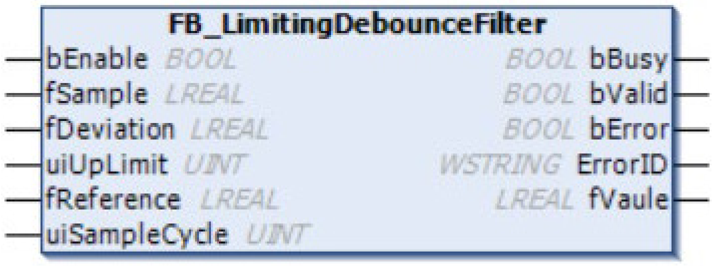
Output variable	Name	Data type	Valid range	Description
bBusy	Instruction executing	BOOL	TRUE, FALSE	TRUE: Function block is executing.
bValid	Output valid	BOOL	TRUE, FALSE	TRUE: Instruction execution is valid.
bError	Error	BOOL	TRUE, FALSE	TRUE: An exception occurred.
eErrorID	Error ID	WSTRING		
fValue	Filtered output valid value	LREAL		Filtered output valid value.

◆ Key points

• After the bEnable level is enabled, given the number of scan cycles uiSampleCycle, the sampling count uiSampleNum, the sampling reference valid value fReference, and the maximum allowed deviation between two samples fDeviation, input the sampling variable fSample.

• Each newly sampled data first undergoes limiting processing, then is fed into a queue for recursive average filtering. This combines the limiting filter method and the recursive average filter method, eliminating sampling deviations caused by sporadic interference.

6.6.5 LimitingDebounceFilter (Limiting debounce filter)

<table>
<tr><th>Name</th><th colspan="3">FB_LimitingDebounceFilter (Limiting debounce filter)</th></tr>
<tr><td>Supported modes</td><td>CSP</td><td>CSV</td><td>CST</td></tr>
<tr><td colspan="2">Graphical representation</td><td colspan="2">ST representation</td></tr>
<tr><td colspan="2">
</td><td colspan="2">FB_LimitingDebounceFilter(
bEnable:= ,
fSample:= ,
fDeviation:= ,
uiUpLimit:= ,
fReference:= ,
uiSampleCycle:= ,
bBusy=> ,
bValid=> ,
bError=> ,
ErrorID=> ,
fVaule=>);</td></tr>
</table>

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bEnable	Function block enable	BOOL	TRUE, FALSE	FALSE	TRUE: Function block executes; FALSE: Does not execute.
fSample	Input sample value	REAL		0	Input value to be filtered.
fDeviation	Input maximum allowed deviation between two samples	REAL		0	Input maximum allowed deviation between two samples.
uiUpLimit	Set filter counter upper limit	UINT			Set the filter counter upper limit.
fReference	Input reference value	REAL		0	Input reference value.
uiSampleCycle	Input sample cycle	UINT	1-1000	0	Input sample cycle.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
bBusy	Instruction executing	BOOL	TRUE, FALSE	TRUE: Function block is executing.
bValid	Output valid	BOOL	TRUE, FALSE	TRUE: Instruction execution is valid.
bError	Error	BOOL	TRUE, FALSE	TRUE: An exception occurred.
eErrorID	Error ID	WSTRING		
fValue	Filtered output valid value	LREAL		Filtered output valid value.

◆ Key points

• After the bEnable level is enabled, given the number of scan cycles uiSampleCycle, the filter counter upper limit uiUpLimit, the sampling reference value fReference, and the maximum allowed deviation between two samples fDeviation, input the sampling variable fSample.

• If the difference between the current value and the last value is <= fDeviation, the current value is valid and used as the sample value for debounce filtering.

• If the difference between the current value and the last value is > fDeviation, the current value is invalid. It is discarded, and the last value replaces it as the sample value for debounce filtering.

• After undergoing limiting, the value is used as the debounce filter's sample value and compared with the current valid value.

• If the sample value = the current valid value, then the current count value is cleared.

• If the sample value <> the current valid value, the count increments by 1, and a check is made for overflow (count > upper limit uiUpLimit). If the count overflows, the current sample value replaces the current filtered output value, and the count is cleared.

6.6.6 LimitingFilter (Limiting filter)

<table>
<tr><th>Name</th><th colspan="3">FB_LimitingFilter (Limiting filter)</th></tr>
<tr><td>Supported modes</td><td>CSP</td><td>CSV</td><td>CST</td></tr>
<tr><td colspan="2">Graphical representation</td><td colspan="2">ST representation</td></tr>
<tr><td colspan="2">
</td><td colspan="2">FB_LimitingFilter(
bEnable:= ,
fSample:= ,
fDeviation:= ,
fReference:= ,
uiSampleCycle:= ,
bBusy=> ,
bValid=> ,
bError=> ,
eError=> ,
fValue=>);</td></tr>
</table>

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bEnable	Function block enable	BOOL	TRUE, FALSE	FALSE	TRUE: Function block executes.
fSample	Input sample value	REAL		0	Input value to be filtered.
fDeviation	Input maximum allowed deviation between two samples	REAL		0	Input maximum allowed deviation between two samples.
fReference	Input reference value	REAL		0	Input sampling reference value.
uiSampleCycle	Input sample cycle	UINT	1-1000	0	Input sample cycle.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
bBusy	Instruction executing	BOOL	TRUE, FALSE	TRUE: Function block is executing.
bValid	Output valid	BOOL	TRUE, FALSE	TRUE: Instruction execution is valid.
bError	Error	BOOL	TRUE, FALSE	TRUE: An exception occurred.
eErrorID	Error ID	WSTRING		
fValue	Filtered output valid value	REAL		Filtered output valid value.

◆ Key points

• After the bEnable level is enabled, given the number of scan cycles uiSampleCycle, the sampling reference value fReference, and the maximum allowed deviation between two samples fDeviation, input the sample value fSample. Each time a new value is sampled, determine:

• If the difference between the current value and the last value is <= fDeviation, the current value is valid and output as the filtered value.

• If the difference between the current value and the last value is > fDeviation, the current value is invalid. It is discarded, and the last value replaces it as the filtered output.

• It can overcome interference caused by sporadic events; less effective against periodic interference.

6.6.7 MedianAverageFilter (Median average filter)

Name	FB_MedianAverageFilter (Median average filter)		
Supported modes	CSP	CSV	CST
Graphical representation		ST representation	
		FB_MedianAverageFilter(bEnable:= , fSample:= , uiSampleNum:= , uiSampleCycle:= , bBusy=> , bValid=> , bError=> , eError=> , fValue=>);	

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bEnable	Function block enable	BOOL	TRUE, FALSE	FALSE	TRUE: Function block executes,
fSample	Input sample value	REAL		0	Input value to be filtered.
uiSampleNum	Input sample count	UINT	3-3000	0	Input sample count.
uiSampleCycle	Input sample cycle	UINT	1-1000	0	Input sample cycle.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
bBusy	Instruction executing	BOOL	TRUE, FALSE	TRUE: Function block is executing.
bValid	Output valid	BOOL	TRUE, FALSE	TRUE: Instruction execution is valid.
bError	Error	BOOL	TRUE, FALSE	TRUE: An exception occurred.
eErrorID	Error ID	WSTRING		
fValue	Filtered output valid value	REAL		Filtered output valid value.

◆ Key points

• After the bEnable level is enabled, given the number of scan cycles uiSampleCycle and the sample count uiSampleNum, input the sampling variable fSample. Continuously sample uiSampleNum times, sort the data set, remove the maximum and minimum values, then take the average. This is equivalent to combining the "median filter method" and the "arithmetic mean filter method". uiSampleNum is typically chosen as 3~14.

• It can overcome interference caused by sporadic events, and has some effect against periodic interference. Suitable for high-frequency oscillating systems. However, the calculation speed is slow.

6.6.8 MedianFilter (Median filter)

<table>
<tr><th>Name</th><th colspan="3">FB_MedianFilter (Median filter)</th></tr>
<tr><td>Supported modes</td><td>CSP</td><td>CSV</td><td>CST</td></tr>
<tr><td colspan="2">Graphical representation</td><td colspan="2">ST representation</td></tr>
<tr><td colspan="2">
</td><td colspan="2">FB_MedianFilter(
bEnable:= ,
fSample:= ,
uiSampleNum:= ,
uiSampleCycle:= ,
bBusy=> ,
bValid=> ,
bError=> ,
eError=> ,
fValue=>);</td></tr>
</table>

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bEnable	Function block enable	BOOL	TRUE, FALSE	FALSE	TRUE: Function block executes.
fSample	Input sample value	REAL		0	Input value to be filtered.
uiSampleNum	Input sample count	UINT	1-1000	0	Input sample count.
uiSampleCycle	Input sample cycle	UINT	1-1000	0	Input sample cycle.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
bBusy	Instruction executing	BOOL	TRUE, FALSE	TRUE: Function block is executing.
bValid	Output valid	BOOL	TRUE, FALSE	TRUE: Instruction execution is valid.
bError	Error	BOOL	TRUE, FALSE	TRUE: An exception occurred.
eErrorID	Error ID	WSTRING		
fValue	Filtered output valid value	REAL		Filtered output valid value.

◆ Key points

- After the bEnable level is enabled, given the number of scan cycles uiSampleCycle and the sample count uiSampleNum, input the sampling variable fValue. Continuously sample uiSampleNum times, arrange the uiSampleNum sample values in order, and take the middle value as the current filter value.
- It can overcome interference caused by sporadic events, but is not effective for rapidly changing parameters.

6.6.9 RecursiveAverageFilter (Recursive average filter)

Name	FB_RecursiveAverageFilter (Recursive average filter)		
Supported modes	CSP	CSV	CST
Graphical representation		ST representation	
		FB_RecursiveAverageFilter(bEnable:= , fSample:= , uiSampleNum:= , uiSampleCycle:= , bBusy=> , bValid=> , bError=> , eError=> , fValue=>);	

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bEnable	Function block enable	BOOL	TRUE, FALSE	FALSE	TRUE: Function block executes.
fSample	Input sample value	REAL		0	Input value to be filtered.
uiSampleNum	Input queue length	UINT	1-3000	0	Input queue length.
uiSampleCycle	Input sample cycle	UINT	1-1000	0	Input sample cycle.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
bBusy	Instruction executing	BOOL	TRUE, FALSE	TRUE: Function block is executing.
bValid	Output valid	BOOL	TRUE, FALSE	TRUE: Instruction execution is valid.
bError	Error	BOOL	TRUE, FALSE	TRUE: An exception occurred.
eErrorID	Error ID	WSTRING		
fValue	Filtered output valid value	REAL		Filtered output valid value.

◆ Key points

• After the bEnable level is enabled, given the number of scan cycles uiSampleCycle and the queue length uiQueueCount, input the sampling variable fSample. Treat a continuous sequence of uiQueueCount samples as a queue with a fixed length of uiQueueCount. Each time a new data sample is obtained, it is placed at the tail of the queue, and the data at the head of the queue is discarded (First-In-First-Out principle). Then, an arithmetic average is calculated on the N data in the queue to obtain a new filter result.

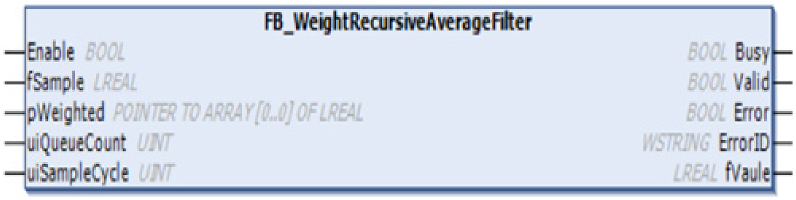
• A larger uiQueueCount value results in higher signal smoothness but lower sensitivity.

• A smaller uiQueueCount value results in lower signal smoothness but higher sensitivity.

• Typical selection for uiQueueCount: Flow: 12; Pressure: 4; Temperature: 4.

• It is effective for systems with high-frequency oscillation and periodic interference, but has low sensitivity and poor ability to overcome sporadic interference.

6.6.10 WeightRecursiveAverageFilter (Weighted recursive average filter)

Name	FB_WeightRecursiveAverageFilter (Weighted recursive average filter)		
Supported modes	CSP	CSV	CST
Graphical representation		ST representation	
		FB_WeightRecursiveAverageFilter(Enable:= , fSample:= , pWeighted:= , uiQueueCount:= , uiSampleCycle:= , Busy=> , Valid=> , Error=> , ErrorID=> , fVaule=>);	

(1) Input variable

Input variable	Name	Data type	Valid range	Initial value	Description
bEnable	Function block enable	BOOL	TRUE, FALSE	FALSE	TRUE: Function block executes.
fSample	Input sample value	REAL		0	Input value to be filtered.
pfWeighted	Input weight coefficients, stored in an array	POINTER TO REAL		0	Input weight coefficients, stored in an array; cannot all be zero.
uiQueueCount	Input queue length	UINT	1-3000	0	Input valid value.
uiSampleCycle	Input sample cycle	UINT	1-1000		Input sample cycle.

(2) Output variable

Output variable	Name	Data type	Valid range	Description
bBusy	Instruction executing	BOOL	TRUE, FALSE	TRUE: Function block is executing.
bValid	Output valid	BOOL	TRUE, FALSE	TRUE: Instruction execution is valid.
bError	Error	BOOL	TRUE, FALSE	TRUE: An exception occurred.
eErrorID	Error ID	WSTRING		
fValue	Filtered output valid value	REAL		Filtered output valid value.

◆ **Key points**

• After the bEnable level is enabled, given the number of scan cycles uiSampleCycle, the queue length uiQueueCount, input the sampling variable fSample and the address pfWeighted of the corresponding weight coefficient array.

• Data from different times are given different weights. Typically, data closer to the current time receive larger weights. Giving a larger weight coefficient to newer samples increases sensitivity but reduces signal smoothness.

• It is suitable for objects with short sampling periods or objects with large lag time constants.

6.7 PID auto-tuning function block

Note: The detailed description of this function block is lengthy and has its own comprehensive manual. Please refer to the manual: ATC Temperature Control PID User Manual.

Innovation Integrity Service

LinkedIn YouTube

Zhejiang Hechuan Technology Co., Ltd.
No.5, Qinshan Road, Longyou Industrial Zone, Quzhou City, Zhejiang Province

R&D Center (Hangzhou)
No. 299, Lixin Road, Qingshanhu Road, Lin'an District, Hangzhou City, Zhejiang Province, P.R. China

HCFA Official Website - www.hcfaglobal.com