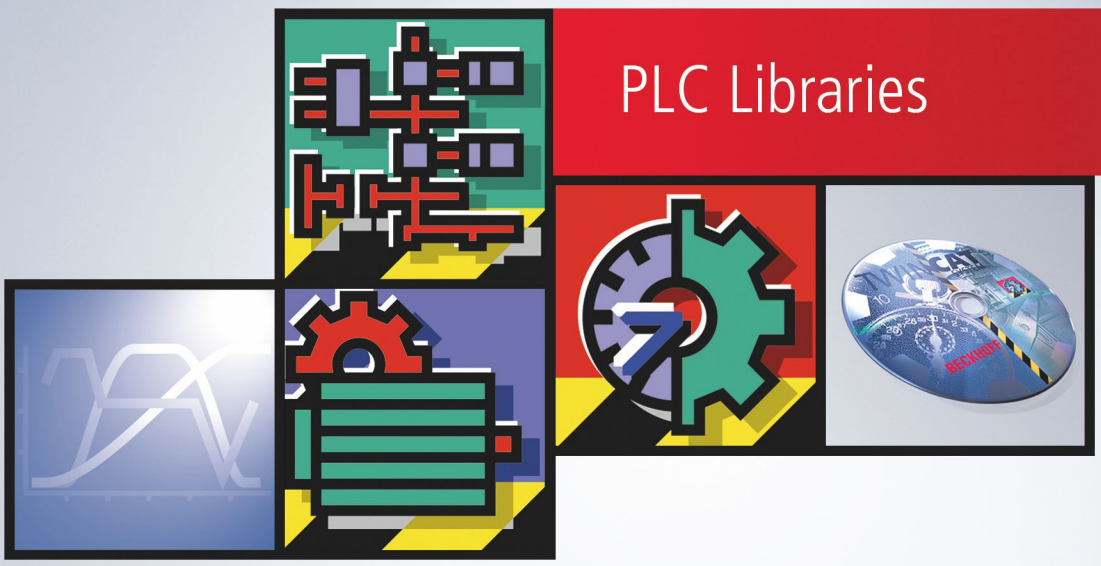
BECKHOFF New Automation Technology

Manual | EN

TX1200

TwinCAT 2 | PLC Library: TcPlcUtilitiesBC



2022-11-10 | Version: 1.1

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with applicable national standards.
It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.
It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

The responsible staff must ensure that the application or use of the products described satisfy all the requirements for safety, including all the relevant laws, regulations, guidelines and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without prior announcement.
No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.
Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
with corresponding applications or registrations in various other countries.

EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

Copyright

1.2 Safety instructions

Safety regulations

Please note the following safety instructions and explanations!
Product-specific safety instructions can be found on following pages or in the areas mounting, wiring, commissioning etc.

Exclusion of liability

All the components are supplied in particular hardware and software configurations appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation and drive engineering who are familiar with the applicable national standards.

Description of symbols

In this documentation the following symbols are used with an accompanying safety instruction or note. The safety instructions must be read carefully and followed without fail!

⚠ DANGER

Serious risk of injury!

Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.

⚠ WARNING

Risk of injury!

Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.

⚠ CAUTION

Personal injuries!

Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.

NOTE

Damage to the environment or devices

Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.

Tip or pointer

This symbol indicates information that contributes to better understanding.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

2 Overview

The library contains useful function blocks for the PLC controller (BCxxxx controller). In addition to the RTC blocks, the library contains a function block for decoding the DCF-77 time signal, along with a number of conversion functions. Internally, PLC Controller system functions are called.

Requirements

Some of the functions are only supported by PLC controllers with a newer firmware version. The required firmware versions are listed in the PlcSystemBC library documentation.

Content of the library

Function blocks

Name	Description
RTC [▶ 10]	Real Time Clock
RTC_EX [▶ 11]	Real Time Clock with an additional millisecond output
DCF77_TIME [▶ 13]	Read the DCF77 radio time
ReadWriteTerminalReg [▶ 16]	Access to the registers of the terminals
DRAND [▶ 18]	Random number generator
FB_BasicPID [▶ 19]	Simple PID controller

Functions

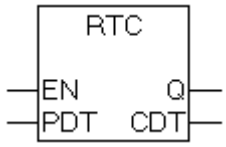
Name	Description
DT_TO_SYSTEMTIME [▶ 22]	Converting DATE_AND_TIME to Windows system time structure
SYSTEMTIME_TO_DT [▶ 22]	Converting Windows system time structure to DATE_AND_TIME
TIME_TO_OTSTRUCT [▶ 23]	Convert TIME to a structure with milliseconds, seconds, minutes, hours, days and weeks
OTSTRUCT_TO_TIME [▶ 24]	Convert a structure with milliseconds, seconds, minutes, hours, days and weeks to a TIME variable
SETBIT32 [▶ 25]	Sets a bit in a 32-bit variable to 1
CSETBIT32 [▶ 26]	Sets / resets a bit in a 32-bit variable
GETBIT32 [▶ 26]	Determines the value of a bit of a 32-bit variable
CLEARBIT32 [▶ 27]	Sets a bit in a 32-bit variable to zero
F_SwapReal [▶ 28]	Converts bus controller REAL numbers into the 4Byte REAL Intel format.
IsFinite [▶ 29]	Verifies the formatting of a floating-point number in accordance with the IEEE

Data structures

Name	Description
TYPE TIMESTRUCT [▶ 32]	Windows system time structure
TYPE OTSTRUCT [▶ 32]	Operating time structure

3 Function blocks

3.1 RTC



The "RTC" (Real Time Clock) function block allows an internal clock to be implemented within a PLC controller (BCxxxx). The clock must be initialized with a starting date and time. After initialization, the time and date are updated cyclically each time the function block is called. A controller system clock is used to calculate the current time and date. The system clock has a resolution of one millisecond. The function block should be called in every PLC cycle, so that the current time can be calculated. The RTC has a potential error of about 1 minute in each 24 hours. In order to avoid large errors, the RTC can be cyclically synchronized (e.g. with a radio clock or with a TwinCAT PC via the fieldbus). The current date and time are available in the usual DATE_AND_TIME (DT) format at the function block's output. Multiple instances of the RTC function block can be created within one PLC program.

VAR_INPUT

```
VAR_INPUT

    EN  :  BOOL;

    PDT : DATE_AND_TIME;

END_VAR
```

EN: on a rising edge at this input, the function block is reinitialized with a preset time and date.
PDT: (Preset Date and Time) the initialization values for the date and time of the function block. A rising edge at the EN input will cause the function block to adopt this value.

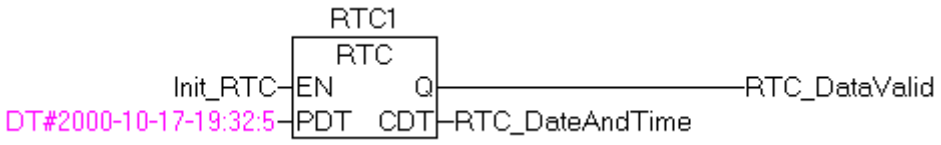
VAR_OUTPUT

```
VAR_OUTPUT

    Q  :   BOOL;

    CDT: DATE_AND_TIME;

END_VAR
```

Q: this output is set if the function block has been initialized at least once. If this output is set, then the values for the date and time at the PDT output are valid.
CDT: (Current Date and Time) current date and time from RTC. The CDT output is only updated when the function block is called. For this reason, instances of the function block should be called once in each PLC cycle.

Sample of a call in FBD:

```
PROGRAM MAIN
VAR
    RTC1           : RTC;
    Init_RTC       : BOOL;
    RTC_DataValid  : BOOL;
    RTC_DateAndTime: DT;
END_VAR
```

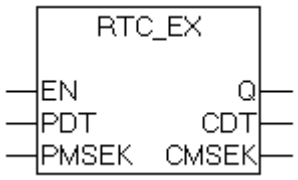


Requirements

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 and above	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

3.2 RTC_EX



The "RTC_EX" function block allows an internal clock to be implemented within a PLC controller (BCxxxx). The clock must be initialized with a starting date and time. After initialization, the time and date are updated cyclically each time the function block is called. A controller system clock is used to calculate the current time and date. The system clock has a resolution of one millisecond. So that the current time can be calculated, the function block should be called once in each cycle of the PLC.
The RTC_EX function block has a deviation of approx. 1 minute per 24 hours. In order to avoid large errors, the RTC_EX function block can be cyclically synchronized (e.g. with a radio clock or with a TwinCAT PC via the fieldbus). The current date and time are available in the usual DATE_AND_TIME (DT) format at the function block's output. In contrast to the RTC [▶ 10] function block, RTC_EX has a precision of one millisecond. Since DATE_AND_TIME variables only have a precision of one second, the milliseconds are output via the CMSEK output variable. Multiple instances of the RTC_EX function block can be created within one PLC program.

VAR_INPUT

```
VAR_INPUT

EN   :    BOOL;

PDT  :   DATE_AND_TIME;

PMSEK: DWORD;

END_VAR
```

EN: on a rising edge at this input, the RTC_EX function block is reinitialized with a preset time, date and milliseconds.
PDT: (Preset Date and Time) the initialization values for the date and time of the function block. With a rising edge at the EN input, this value is taken over by the function block.
PMSEK: (Preset Milliseconds) the initialization value for the milliseconds. A rising edge at the EN input will cause the function block to adopt this value.

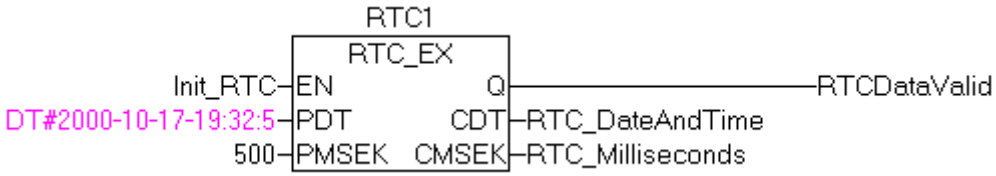
VAR_OUTPUT

```
VAR_OUTPUT

Q    : BOOL;

CDT  : DATE_AND_TIME;

CMSEK: DWORD;

END_VAR
```

Q: this output is set if the function block has been initialized at least once. If the output is set, the values for the date, time and milliseconds at the PDT and CMSEK outputs are valid.
CDT: (Current Date and Time) current date and time from RTC. The CDT output is only updated when the function block is called. Therefore the instances of the function block should be called once in each cycle of the PLC.
CMSEK: (Current Milliseconds) the millisecond output.

Sample of a call in FBD:

```
PROGRAM MAIN
VAR
    RTC1                : RTC_EX;
    Init_RTC            : BOOL;
    RTCDataValid        : BOOL;
    RTC_DateAndTime     : DT;
    RTC_Milliseconds    : DWORD;
END_VAR
```



Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 and above	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

3.3 DCF77_TIME

The "DCF77_TIME" function block can be used to decode the DCF-77 radio clock signal. A rising edge at the RUN input starts the decoding process, which continues as long as the RUN input remains set. In the worst case, the function block requires a maximum of one minute to synchronize itself, plus a further minute to decode the data for the following minute. During this time, the missing 59th second marker is waited for. Internally the function block is sampling the DCF-77 signal. In order to be able to sample the edges without error the function block should be called once in each PLC cycle. Satisfactory results can be obtained with a cycle time of <= 25 ms. If the DCF-77 signal is absent or faulty, the ERR output is set TRUE, and a corresponding error code is set at the ERRID output. The ERR and ERRID outputs are reset the next time a correct signal is received. Some receivers provide an inverted DCF-77 signal. In such cases the signal must first be inverted before being passed to the DCF_PULSE input. When operating without errors, the current time is updated at the CDT output every minute. Thereby the READY output is set to TRUE for one cycle of the PLC at the zeroth second. At this time the DCF-77 time at the CDT output is valid, and can be evaluated by the PLC program. The READY output is only set if the data for the following minute has been detected. The transferred parity bits are used for error detection. In the event of poor reception conditions, 100% error-free detection cannot be guaranteed. I.e. with two faulty (inverted) bits, the function block cannot detect an error and also sets the READY output to TRUE. In order to obtain reliable time information additional safeguards have to be implemented, e.g. redundancy analysis of the time information in consecutive minutes.

From TwinCAT v2.10 Build > 1340 and TwinCAT v2.11 Build > 1543 a simple plausibility check of two consecutive telegrams was implemented in the DCF77_TIME function block. This functionality can be activated via a global Boolean variable for all instances of the DCF77_TIME block. When the plausibility check is activated the first synchronization is extended by a further minute to a maximum of 3 minutes.

```
GLOBAL_DCF77_SEQUENCE_CHECK : BOOL := FALSE;
```

TRUE = plausibility check is activated (two consecutive telegrams are checked)

FALSE = plausibility check is deactivated

Errors that occur during reception are registered by the function block. The ERRCNT output is an error counter. This counter indicates the number of errors that have occurred since the last correctly received signal. The counter is reset the next time a signal is correctly received.

Time code

During each minute, the numbers that encode the year, month, day, day of the week, hour and minute are transmitted in BCD format through pulse modulation of the second marks. The transmitted information always describes the subsequent minute. A second marker is transmitted each second. A second marker with a duration of 0.1 s represents a binary zero, while a duration of 0.2 s represents binary one. The information is extended with 3 check bits. At the 59th second, and a receiver can use this "gap" to synchronize itself.

From TwinCAT v2.10 Build > 1340 and TwinCAT v2.11 Build > 1543 the length of the short and long pulse signal can also be configured via a global variable. If the signal is poor the pulse widths are smaller. The receiver specifications usually contain information about the minimum and maximum pulse for the two

logic signals, with the higher value expected for higher field strengths and the lower value for low field strengths or in the event of interference. Problems may also occur near the sender (where the field strength is very large) if the pulse width of the logic zero becomes excessive. For this reason a fixed limit is set for differentiating between zero and one, depending on the receiver specification. **Check the specification of the receiver used and configure the pulse length accordingly.**

```
GLOBAL_DCF77_PULSE_SPLIT : TIME := T#140ms;
```

0 == pulse < 140 ms, 1 == pulse > 140

E.g.: in the specification of Atmel T4227 (Time Code Receiver) the following pulse length is given:
100 ms pulse (zero): min: 70 ms, typical: 95 ms, max: **130 ms**
200 ms pulse (one): min. **170 ms**, typical 195 ms, max. 235 ms
For this IC a limit value of 150 ms would be optimal = 130 + ((170 ms - 130 ms) / 2).

Tip:

If the configured limit value for the pulse length is too small, short pulses are detected as long. Conversely, if the configured limit value is too small, long pulses are detected as short. If the checksum is correct, the receiver cannot detect these errors. In the first case the receiver may supply times that are in future range, in the second case the times may be in the past.

VAR_INPUT

```
VAR_INPUT
    DCF_PULSE   :BOOL;
    RUN         :BOOL;
END_VAR
```

DCF_PULSE: the DCF-77 signal.
RUN: a rising edge at this input initializes the function block and starts decoding the DCF-77 signal. If this input is reset, the decoding process is stopped

VAR_OUTPUT

```
VAR_OUTPUT
    BUSY        :BOOL;
    ERR         :BOOL;
    ERRID       :UDINT;
    ERRCNT      :UDINT;
    READY       :BOOL;
    CDT         :DATE_AND_TIME;
END_VAR
```

BUSY: when the function block is activated, this output is set.
ERR: if an error occurred during decoding, this output is set.
ERRID: supplies the error number when the ERR output is set.
ERRCNT: number of errors that have occurred since the last error-free reception.
READY: if this output is set, then the data at the CDT output are valid.
CDT: the DCF-77 time in DATE_AND_TIME format.

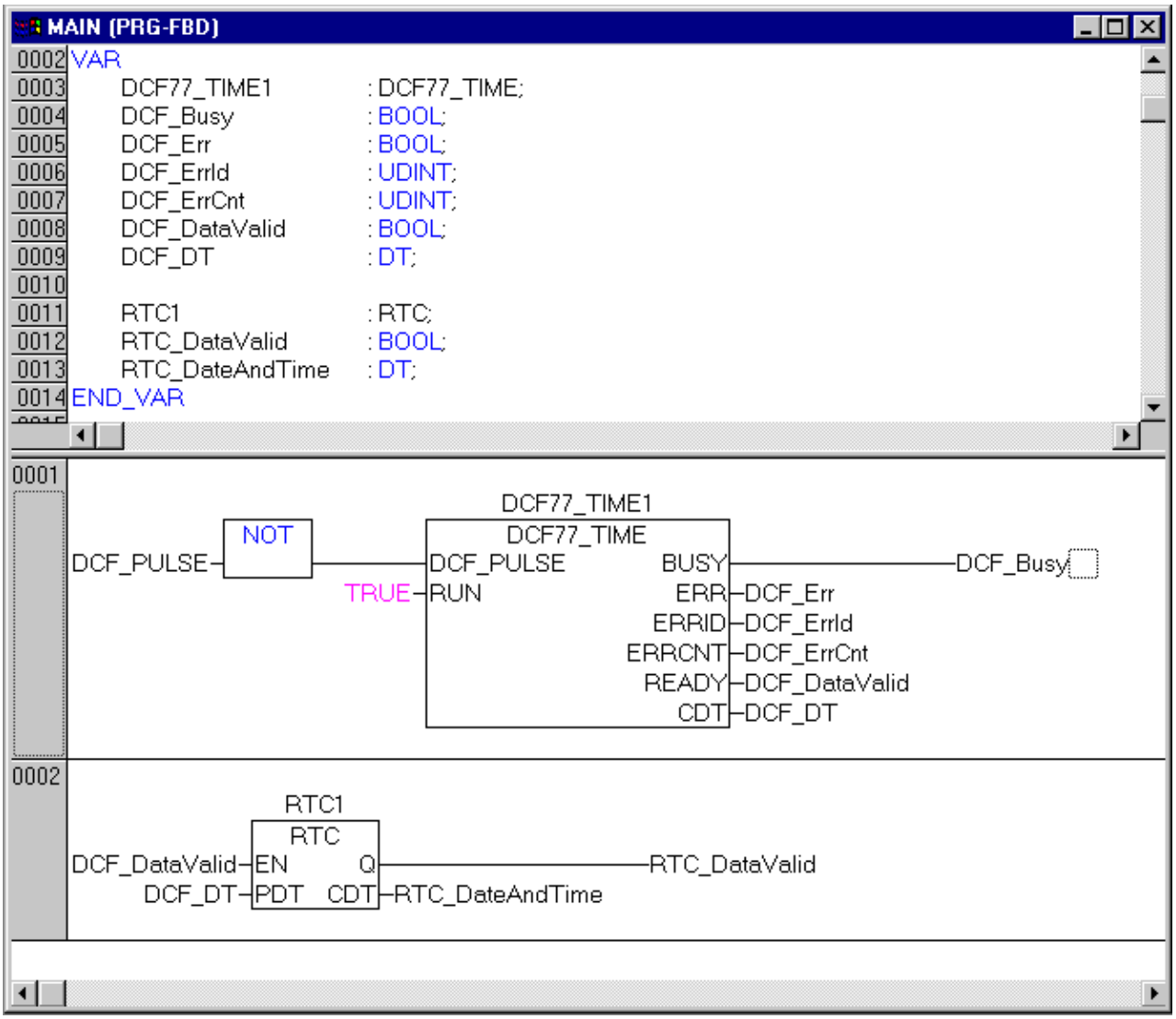
Error description:

Error Codes	Error description
0	No error
0x100	Timeout error. Possibly no DCF-77 signal detected.
0x200	Parity error. Incorrect bits were detected in the received data.
0x300	Incorrect data was received. Since the parity check can only detect one incorrect bit, the received data are checked again for validity (this error code will be generated, for instance, if month = 13).

Error Codes	Error description
0x400	The last decoding cycle was too long. This error can occur when reception is poor (not enough second markers were received).
0x500	The last decoding cycle was too short. This error can occur when reception is poor (additional edges were received).

Sample of a call in FBD

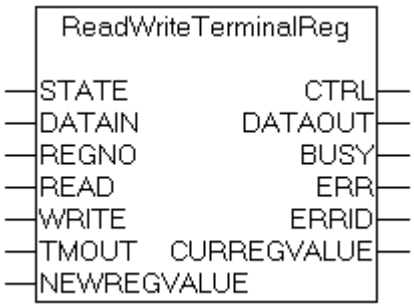
In the sample application the Real Time Clock is synchronized with the radio time signal as an error-free signal is received.



Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 and above	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

3.4 ReadWriteTerminalReg



The "ReadWriteTerminalReg" function block permits convenient access to the registers of the terminal via the terminal channel's status/control byte (register communication). In the standard operating mode, the data inputs and outputs of the intelligent terminal (e.g. an analog output terminal) are used to exchange the analog output data. A handshake via the status/control byte permits register access. The data input and output variables are used here to transfer the register values. The terminal must be mapped as a complex PLC terminal so that the status/control byte is visible in the process image. The terminal registers cannot be accessed if the terminal has been mapped as a compact PLC terminal or as a fieldbus terminal (process image of the terminal not visible to the bus controller).
A positive edge at the READ or WRITE input causes the register with number REGNO to be read or written to. Write protection of the register is disabled by the function block for a write access and enabled once more afterwards. When a register is written to, the new register value is read, and is available at the CURRREGVALUE output. If changes made to the register values are to be stored permanently, the power supply to the coupler must be interrupted or a software reset of the coupler must be executed.

VAR_INPUT

```
VAR_INPUT
    STATE           : BYTE;
    DATAIN          : WORD;
    REGNO           : BYTE;
    READ            : BOOL;
    WRITE           : BOOL;
    TMOUT           : TIME;
    NEWREGVALUE     : WORD;
END_VAR
```

STATE: status byte of the terminal channel.
DATAIN: data input word of the terminal channel.
REGNO: number of the register to which a read or write access is to be made.
READ: a positive edge at this input activates the function block and reads the current register value. If successful, the register value is available in the output variable CURRREGVALUE.
WRITE: a positive edge at this input activates the function block, and the value in the input variable NEWREGVALUE is written into the register REGNO. Subsequently, the current value of the register is read and is available in the output variable CURREGVALUE if successful.
TMOUT: specifies the timeout time that must not be exceeded when executing the function.
NEWREGVALUE: data word that is to be written to the register with the number REGNO during a write access.

VAR_OUTPUT

```
VAR_OUTPUT
    CTRL          :BYTE;
    DATAOUT       :WORD;
    BUSY          :BOOL;
    ERR           :BOOL;
    ERRID         :UDINT;
    CURREGVALUE   :WORD;
END_VAR
```

CONTROL: control byte of the terminal channel.
DATAOUT: data output word of the terminal channel.
BUSY: when the function block is activated, this output is set and remains set until the execution of the function has been completed.
ERR: if an error occurs during the execution of the function, then this output is set after the BUSY output has been reset.
ERRID: returns the error number when the ERR output is set.
CURREGVALUE: in case of a successful read or write access, the current register value is output via the variable.

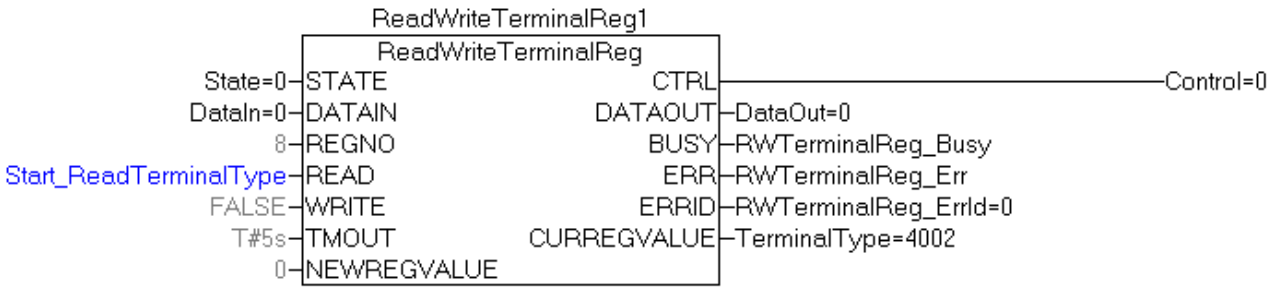
Error description:

Error number	Error description
0	No error
0x100	Timeout error. The time permitted for execution has been exceeded.
0x200	Parameter error (e.g. an invalid register number).
0x300	The read value differs from the written value (write access to this register may not be enabled or have failed)

Samples of a call in FBD:

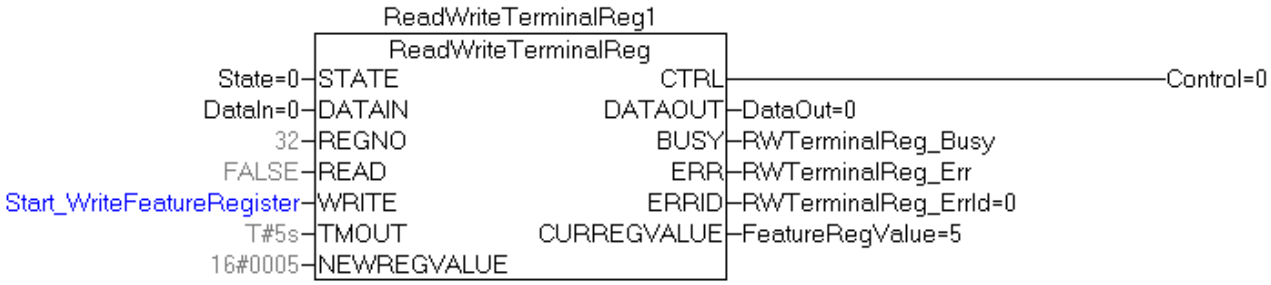
```
VAR
ReadWriteTerminalReg1     : ReadWriteTerminalReg;
State AT%IB0              : BYTE;
Control AT%QB0            : BYTE;
DataIn AT%IW2             : WORD;
DataOut AT%QW2            : WORD;
Start_ReadTerminalType    : BOOL;
Start_WriteFeatureRegister: BOOL;
RWTerminalReg_Busy        : BOOL;
RWTerminalReg_Err         : BOOL;
RWTerminalReg_ErrId       : UDINT;
TerminalType              : WORD;
FeatureRegValue           : WORD;
END_VAR
```

Sample 1



In Sample 1 the terminal name is read from register 8 of an analog output terminal. The variables *State*, *Control*, *DataIn* and *DataOut* are linked to the terminal's corresponding I/O variables in the TwinCAT System Manager. The terminal name is KL**4002**.

Sample 2

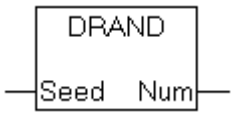


In Sample 2 the user-scaling is activated in the feature register (register 32) of a KL4002 analog output terminal. The new value in the feature register is then read by the function block, and can be checked through the output variable **CURREGVALUE**.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 and above	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

3.5 DRAND



Instances of function blocks are created according to IEC61131-3, and then called, or otherwise accessed, from with the PLC program using the instance names. The function block DRAND permits generation of a (pseudo-) random number of type REAL.

VAR_INPUT

```
VAR_INPUT
    Seed    : INT;
END_VAR
```

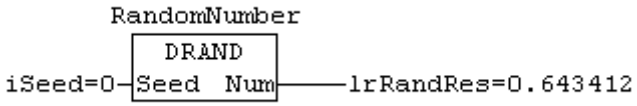
Seed: initial value to define the random number sequence.

VAR_OUTPUT

```
VAR_OUTPUT
    Num     : REAL;
END_VAR
```

Num: this output returns a pseudo-random number in the range 0.0 ... 1.0 with single precision. The generator here creates a number series with 1075 stochastic values per period.

Sample of calling the function block in FBD:

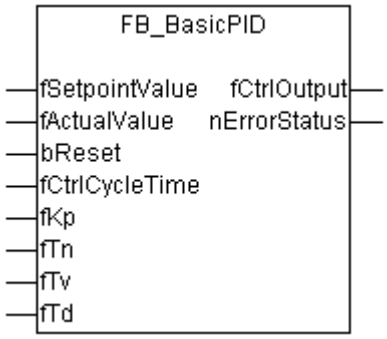


In the sample the REAL value 0.643412 is generated and returned. The input parameter ”Seed" affects the initial value of the series. If, for instance, a deterministically reproducible random number series is desired in different sessions, and identical ”Seed" value must be used.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 and above	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

3.6 FB_BasicPID

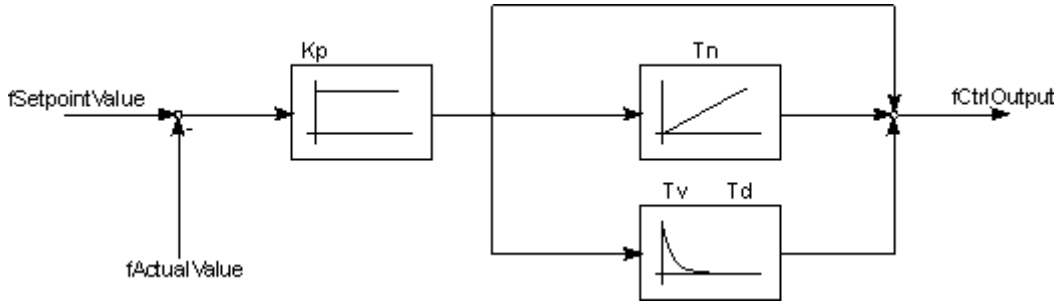


The function block is a simple discretized PID element.

Transfer function:

$$G(s) = K_p(1 + \frac{1}{T_n s} + \frac{T_v s}{1 + T_d s})$$

Functional diagram:



VAR_INPUT

```
VAR_INPUT
    fSetpointValue  : REAL;
    fActualValue    : REAL;
    bReset          : BOOL;
    fCtrlCycleTime  : REAL;
    fKp             : REAL;
    fTn             : REAL;
    fTv             : REAL;
    fTd             : REAL;
END_VAR
```

fSetpointValue: setpoint of the controlled variable.

fActualValue: actual value of the controlled variable.

bReset: TRUE at this input resets the internal state variables and the controller output.

fCtrlCycleTime: cycle time with which the function block is called and with which the control loop is processed [s].

fKp : controller amplification / controller coefficient

fTn : integral action time [s]

fTv: derivative action time [s]

fTd : damping time [s]

VAR_OUTPUT

```
VAR_OUTPUT
    fCtrlOutput     : REAL;
    nErrorStatus    : UINT
END_VAR
```

fCtrlOutput : output of the PID element.

nErrorStatus : indicates the error number in the event of an error (nErrorStatus <> 0).

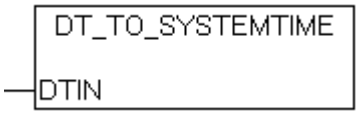
0 = nERR_NOERROR : no error.
1 = nERR_INVALIDPARAM : invalid parameters
2 = nERR_INVALIDCYCLETIME : invalid cycle time.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build > 947	PC (i386)	TcUtilities.Lib (Standard.Lib; TcBase.Lib; TcSystem.Lib are included automatically)
TwinCAT v2.7.0 Build > 522 TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build > 947	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

4 Functions

4.1 DT_TO_SYSTEMTIME



The "DT_TO_SYSTEMTIME" function allows a PLC variable in DATE_AND_TIME format (DT) to be converted to a Windows system time structure. The system time has a resolution of 1ms, while the resolution of DATE_AND_TIME is 1s. The "wMilliseconds" variable in the system time structure therefore always returns the value zero.

FUNCTION DT_TO_SYSTEMTIME : TIMESTRUCT

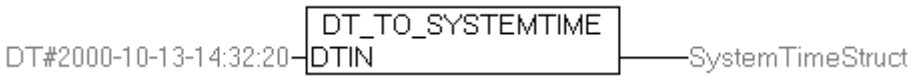
TIMESTRUCT [▶ 32]

```
VAR_INPUT
    DTIN        : DT;
END_VAR
```

DTIN:: the date and time to be converted, in DATE_AND_TIME format.

Sample of a call in FBD:

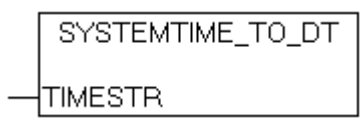
```
PROGRAM SystemTimeTest
VAR
    SystemTimeStruct  :TIMESTRUCT;
END_VAR
```



Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 and above	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

4.2 SYSTEMTIME_TO_DT



The "SYSTEMTIME_TO_DT" function allows the Windows system time structure to be converted to the DATE_AND_TIME format (DT) usual in a PLC. The system time has a resolution of 1ms, while the resolution of DATE_AND_TIME is 1s. The milliseconds from the system time are used in the course of the conversion to determine the direction of rounding for the returned DATE_AND_TIME value.

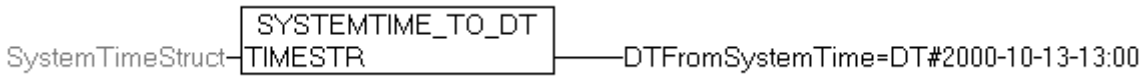
FUNCTION SYSTEMTIME_TO_DT : DT

```
VAR_INPUT
    TIMESTR         :TIMESTRUCT;
END_VAR
```

TIMESTR: structure with the Windows system time to be converted.

Sample of a call in FBD:

```
PROGRAM SystemTimeTest
VAR
    SystemTimeStruct  :TIMESTRUCT;
    DTFromSystemTime  :DT;
END_VAR
```



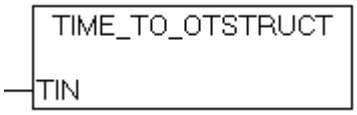
Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 and above	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

Also see about this

TYPE TIMESTRUCT [▶ 32]

4.3 TIME_TO_OTSTRUCT



The function "TIME_TO_OTSTRUCT" can be used to convert a TIME constant or variable into a structure with the resolved milliseconds, seconds, minutes, hours, days and weeks.

FUNCTION TIME_TO_OTSTRUCT : OTSTRUCT

OTSTRUCT [▶ 32]

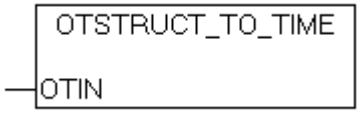
```
VAR_INPUT
    TIN             :TIME;
END_VAR
```

TIN: the TIME variable to be converted.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 and above	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

4.4 OTSTRUCT_TO_TIME



The function "OTSTRUCT_TO_TIME" can be used to convert a structure with resolved milliseconds, seconds, minutes, hours, days and weeks into a TIME variable.

FUNCTION OTSTRUCT_TO_TIME : TIME

```
VAR_INPUT
    OTIN        : OTSTRUCT;
END_VAR
```

OTIN: the structure to be converted.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 and above	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

Also see about this

TYPE OTSTRUCT [▶ 32]

4.5 SETBIT32

The function sets the bit specified by a bit number in the 32-bit value that is passed to it and returns the resulting value.

FUNCTION SETBIT32 : DWORD

VAR_INPUT

```
VAR_INPUT
    inVal32         :DWORD;
    bitNo           :SINT;
END_VAR
```

inVal32: the 32-bit value to be changed.

bitNo: the number of the bit to be set (0-31). This number is internally converted to a modulo 32 value prior to execution.

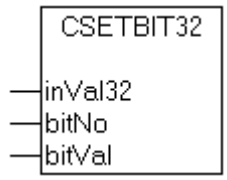
Sample of calling the function in FBD:

Bit 31 is set to the input value '0'. The result is the (hex) value '80000000'.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 and above	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

4.6 CSETBIT32



The function sets/resets the bit specified by a bit number in the 32-bit value that is passed to it and returns the resulting value.

FUNCTION CSETBIT32 : DWORD

VAR_INPUT

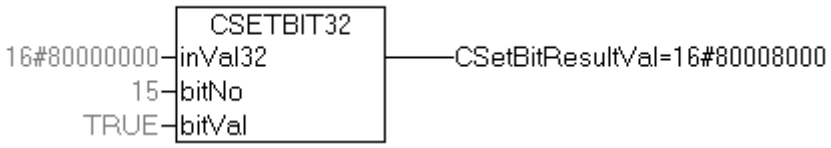
```
VAR_INPUT
    inVal32         :DWORD;
    bitNo           :SINT;
    bitVal          :BOOL;
END_VAR
```

inVal32: a 32-bit value;

bitNo: the number of the bit to be set or reset (0-31). This number is internally converted to a modulo 32 value prior to execution;

bitVal: value to which the bit is to be set or reset (TRUE = 1, FALSE = 0);

Sample of calling the function in FBD:



Bit 15 in the input value '16#80000000' is set to 1. The result (16#80008000) is assigned to the variable *CSetBitResultVal*.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 and above	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

4.7 GETBIT32

The function returns the status of the bit specified by a bit number in the 32-bit value that is passed to it as a boolean resulting value. The input value is not altered.

FUNCTION GETBIT32 : BOOL

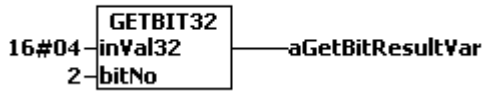
VAR_INPUT

```
VAR_INPUT
    inVal32         :DWORD;
    bitNo           :SINT;
END_VAR
```

inVal32: the 32-bit value;

bitNo: the number of the bit to be read (0-31). This number is internally converted to a modulo 32 value prior to execution;

Sample of calling the function in FBD:

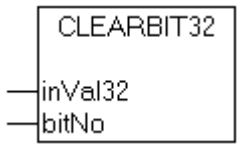


Bit 2 in the input value '16#04' is queried and assigned to the boolean variable *aGetBitResultVar*. The query returns TRUE in this sample.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 and above	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

4.8 CLEARBIT32



The function resets the bit specified by a bit number in the 32-bit value that is passed to it to zero and returns the resulting value.

FUNCTION CLEARBIT32 : DWORD

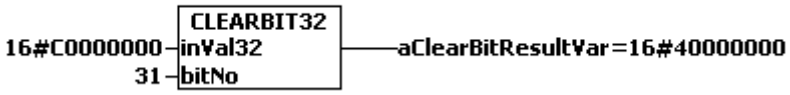
VAR_INPUT

```
VAR_INPUT
    inVal32         :DWORD;
    bitNo           :SINT;
END_VAR
```

inVal32: the 32-bit value to be changed;

bitNo: the number of the bit to be set (0-31). This number is internally converted to a modulo 32 value prior to execution;

Sample of calling the function in FBD:

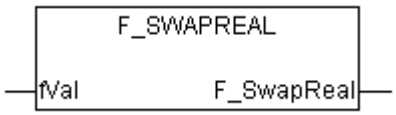


Bit 31 in the input value 'C0000000' is reset. The result is the (hex) value '40000000'.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 and above	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

4.9 F_SwapReal



The way in which a REAL number is represented in the memory of a bus controller (165) is different from the way a REAL number is represented in the memory of an Intel system (PC). In order to represent a bus controller's REAL number correctly in a PC, it is necessary for the high and low words of the REAL number to be swapped. Under the programming environment this is already done in online or simulation mode. To be able to request the REAL data of a Bus Controller via the network (ADS protocol, ADSDLL, AdsOcx etc.) and represent them properly on an Intel PC, the REAL data have to be converted into the correct format. This can either be done in the bus controller or in the PC. The function F_SwapReal can be used to convert the REAL variables (e.g. variables to be read by a VB application or recorded with TwinCAT Scope View) into a suitable format on the PC side. The *fVal* parameter that is passed is not affected by the conversion. The function returns a new REAL number with a modified representation in memory.

FUNCTION F_SwapReal : REAL

VAR_INPUT

```
VAR_INPUT
    fVal    :REAL;
END_VAR
```

fVal: the REAL value to be converted.

Sample in ST:

```
PROGRAM MAIN
VAR
    Real_165        : REAL;
    Real_i386 AT%MB0: REAL;
END_VAR
```

```
(* ADSREAD(.... ADR(Real_165), SIZEOF(Real_165)... );
...
...
...
*)
Real_i386 := F_SwapReal( Real_165 );
Real_i368 := Real_i368 + 0.001;
```

In the sample, a 4-byte REAL variable in the bus controller is converted into the 4-byte REAL Intel format. The Real_i386 variable can, for instance, be correctly displayed on a VB form.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 Build > 518 TwinCAT v2.8.0 Build > 735	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

4.10 IsFinite

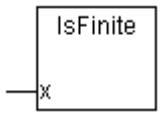


Fig. 1: IsFinite

The function IsFinite() returns TRUE, if its argument has a finite value (INF < x < +INF). The function returns FALSE, if the argument is infinite or NaN (NaN = Not a number). IsFinite() checks whether the formatting of an LREAL or REAL variable complies with IEEE.

INF numbers may occur in a runtime system if the result of a mathematical operation falls outside the range that can be represented. E.g.:

```
PROGRAM MAIN
VAR
    fSingle : REAL := 12.34;
END_VAR
(*Cyclic called program code*)
fSingle := fSingle*2;
```

NaN numbers may occur in the runtime system if their actual formatting (memory content) was overwritten through illegal access (e.g. by using the MEMCPY of MEMSET functions). E.g.:

```
PROGRAM MAIN
VAR
    fSingle : REAL := 12.34;
END_VAR
(*Cyclic called program code*)
MEMSET( ADR( fSingle ), 16#FF, SIZEOF( fSingle ) ); (* Invalid initialization of REAL variable *)
```

Calling a conversion function with an NaN or INF number as parameter causes an FPU exception on a PC system (i368). This exception subsequently leads to the PLC being stopped. The function IsFinite() enables the value of the variables to be checked, and therefore the FPU exception to be avoided and program execution to be continued.

FUNCTION IsFinite : BOOL

```
VAR_INPUT
    x :T_Arg;
END_VAR
```

x: an auxiliary structure with information about the REAL or LREAL variables to be checked. The structure parameters have to be generated when IsFinite() is called from help functions F_REAL [▶ 31] or F_LREAL [▶ 31] and transferred as parameters.

Sample of a call in ST:

In the following sample, the formatting of a REAL and an LREAL variable is checked, and an FPU exception is avoided.

```
PROGRAM MAIN
VAR
    fSingle         : REAL := 12.34;
    fDouble         : LREAL := 56.78;
    singleAsString  : STRING;
    doubleAsString  : STRING;
END_VAR
fSingle := fSingle*2;
IF IsFinite( F_REAL( fSingle ) ) THEN
    singleAsString := REAL_TO_STRING( fSingle );
ELSE
  (* report error !*)
 fSingle := 12.34;
END_IF

fDouble := fDouble*2;
IF IsFinite( F_LREAL( fDouble ) ) THEN
    doubleAsString := LREAL_TO_STRING( fDouble );
ELSE
  (* report error !*)
 fDouble := 56.78;
END_IF
```

In the following case, an FPU exception cannot be avoided through checking with IsFinite():

```
PROGRAM MAIN
VAR
    bigFloat : LREAL := 3.0E100;
    smallDigit: INT;
END_VAR
IF IsFinite( F_LREAL( bigFloat ) ) THEN
    smallDigit := LREAL_TO_INT( bigFloat );
END_IF
```

While the bigFloat variable has the right formatting, the variable value is too large for conversion into an INT type. An exception is triggered on a PC system (i368), and the runtime system is stopped.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 Build > 522	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

Development environment	Target platform	PLC libraries to include
TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build > 947		

Also see about this

TYPE T_Arg [▶ 32]

4.11 F_REAL

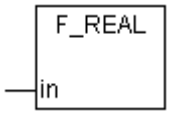


Fig. 2: F_REAL

This is a help function that returns information about a REAL variable with a certain structure.

FUNCTION F_REAL : T_Arg

T_Arg [▶ 32]

VAR_IN_OUT

```
VAR_IN_OUT
    in      :REAL;
END_VAR
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 Build > 522 TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build > 947	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

4.12 F_LREAL

Not present on the BCxxxx (165)

5 Data structures

5.1 TYPE TIMESTRUCT

```
TYPE TIMESTRUCT
STRUCT
   wYear         : WORD;
   wMonth        : WORD;
   wDayOfWeek    : WORD;
   wDay          : WORD;
   wHour         : WORD;
   wMinute       : WORD;
   wSecond       : WORD;
   wMilliseconds: WORD;
END_STRUCT
END_TYPE
```

wYear: the year: 1970 ~ 2106;

wMonth : the month: 1 ~ 12 (January = 1, February = 2, etc.);

wDayOfWeek: the day of the week: 0 ~ 6 (Sunday = 0, Monday = 1 etc.);

wDay : the day of the month: 1 ~ 31;

wHour: hour: 0 ~ 23;

wMinute : minute: 0 ~ 59;

wSecond : second: 0 ~ 59;

wMilliseconds : millisecond: 0 ~ 999;

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 and above	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

5.2 TYPE OTSTRUCT

```
TYPE OTSTRUCT
STRUCT
   wWeek         : WORD;
   wDay          : WORD;
   wHour         : WORD;
   wMinute       : WORD;
   wSecond       : WORD;
   wMilliseconds: WORD;
END_STRUCT
END_TYPE
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 and above	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

5.3 TYPE T_Arg

```
TYPE T_Arg :
STRUCT
    eType   : E_ArgType     := ARGTYPE_UNKNOWN
```

```
    cbLen   : UDINT         := 0
    pData   : UDINT         := 0
END_STRUCT
END_TYPE
```

eType: data type identifier;

cbLen : number of bytes allocated in the memory;

pData : address pointer;

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 Build > 522 TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build > 947	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

Also see about this

TYPE E_ArgType [▶ 33]

5.4 TYPE E_ArgType

```
TYPE E_ArgType :(

    ARGTYPE_UNKNOWN         := 0,
    ARGTYPE_BYTE,
    ARGTYPE_WORD,
    ARGTYPE_DWORD,
    ARGTYPE_REAL,
    ARGTYPE_LREAL,
    ARGTYPE_SINT,
    ARGTYPE_INT,
    ARGTYPE_DINT,
    ARGTYPE_USINT,
    ARGTYPE_UINT,
    ARGTYPE_UDINT,
    ARGTYPE_STRING,
    ARGTYPE_BOOL,
    ARGTYPE_BIGTYPE
);
END_TYPE
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v2.7.0 Build > 522 TwinCAT v2.8.0 Build > 747 TwinCAT v2.9.0 Build > 947	BCxxxx (165)	Standard.Lb6, PlcSystemBC.Lb6, TcPlcUtilitiesBC.Lb6

More Information:
www.beckhoff.com/tx1200

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com