SysLib Library

TXV 003 48.01
4th eddition
January 2010
subject to alterations

Changes history

Date	Eddition	Change description
February 2009	1	First eddition, description corresponds to library SysLib_v17
March 2009	2	Get_IP_address() and Set_IP_address() functions ommited Replaced by functions GetIPaddress() and SetIPaddress() in ComLib library Description corresponds to library SysLib_v18
November 2009	3	Description of CIBunitInfo(), SetAddressCIBunit() and ProgramLock() function added. Description corresponds to library SysLib_v20
Janurary 2010	4	Description of function Memcmp() and function block fbTick() added. Correction of constants MI2_CIB1 up to MI6_CIB2 and added constants MI0_CIB1 a MI0_CIB2 Description corresponds to library SysLib_v22

CONTENT

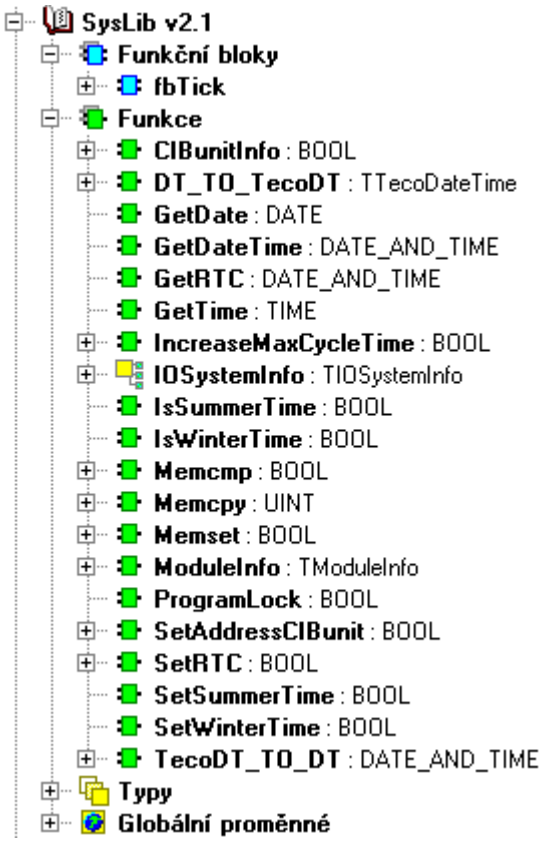
1 INTRODUCTION

Library of functions and function blocks are an integral part of instalation of Mosaic programmable environment. In term of their conception, it is possible to classify the libraries into following types:

- built-in libraries
- standardly supplied external libraries
- user defined libraries

The library can contain declarations of functions, function blocks, data types and global variables. The SysLib library is a standardly supplied libraries.

Following picture shows the structure of the SysLib library within the Mosaic environment



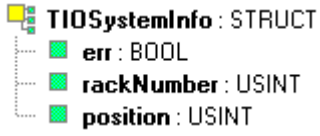
If we want to use the function from the SysLib library in the application program of the PLC, it is neccessary first to add this library into the project. The library is supplied as a part of Mosaic environment instalation.

2 DATA TYPES

In the SysLib library following data types are defined:

- *TIOSystemInfo* structure carrying information about PLC IO systém status
- *TmoduleInfo* structure with data about actual IO module status
- *TSYSTEM_S* structure enabling the access to the PLC systém registers (%Sxx)
- *TTecoDateTime* structure with date and time data
- *TCIBunitState* structure with actual CIB unit status data
- *TCIBunitInfo* structure with data about CIB unit

2.1 TIOSystemInfo type

Lib- 

rary : *SysLib*

TIOSystemInfo type is a structure carrying information about PLC IO system status which is returned by the function *IOSystemInfo*.

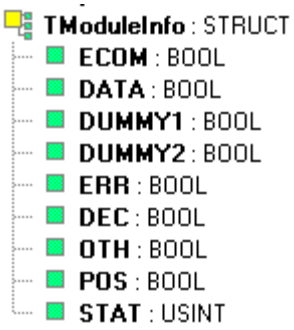
The signification of individual items of the *TIOSystemInfo* structure is as follows:

- *err* err = FALSE PLC IO systém without an error
  err = TRUE error in PLC IO system
- *rackNumber* with err = TRUE states the rack number of the PLC where the module signalling an error is located
- *position* with err = TRUE states the possition number in the rack of the PLC where the modul signalling an error is located
-

See also Function IOSystemInfo

2.2 TmoduleInfo type

Library : *SysLib*



TmoduleInfo type is a structure carrying information about the status of particular PLC IO module which is returned by the function Module*Info*.

The signification of individual items of the structure *TmoduleInfo* is as follows:

- *ECOM* communication error between CPU and IO module
  FALSE … communication is ok
  TRUE … communication error
- *DATA* data provided by the module valid
  FALSE … data are not valid
  TRUE … data are valid
- *DUMMY1* reserve
- *DUMMY2* reserve
- *ERR* IO module error
  FALSE … IO module without en error
  TRUE … IO module indicates error
- *DEC* declaration of IO module in PLC program
  FALSE … module is not declared in PLC program
  TRUE … module has a valid declaration in the PLC program
- *OTH* model type check error
- FALSE … type of the IO module corresponds to the declaration in the PLC program
- TRUE … type od the IO module does not correspond to the declaration in the PLC program
- *POS* module presence
  FALSE … no module is present in the set position
  TRUE … module is present
- *STAT* module status – above mentioned variables like 1 byte
  (ECOM = STAT.0, … , POS = STAT.7)

See also Function ModuleInfo

2.3 TSYSTEM_S type

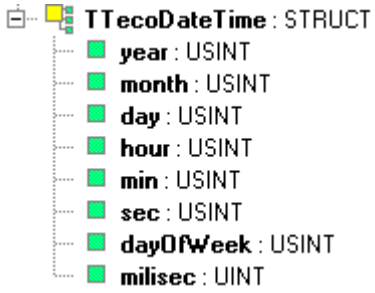
TSYSTEM_S type is a structure enabling the access to the PLC system registers. Its definition is as follows:

```
TYPE
  TSYSTEM_S : STRUCT
    S0                      : BYTE;  // %S0    arithmetic operation results
    S1                      : BYTE;  // %S1    logical operation results
    S2_0                    : BOOL;  // %S2.0  status of service input SP
    S2_1                    : BOOL;  // %S2.1  status of service input MS
    S2_2                    : BOOL;  // %S2.2  RUN mode
    S2_3                    : BOOL;  // %S2.3  warm restart
    S2_4                    : BOOL;  // %S2.4  cold restart
    OUTPUTS_ARE_ENABLED     : BOOL;  // %S2.5  outputs are enabled
    S2_6                    : BOOL;  // %S2.6  first PLC scan without restart
    CYCLE_TIME_WARNING      : BOOL;  // %S2.7  PLC scan first limit overtaken
    LAST_CYCLE_TIME_10MS    : USINT; // %S3    last cycle time [x 10 ms]
    CYCLE_COUNTER           : USINT; // %S4    counter of cycles
    COUNTER_10MS            : USINT; // %S5    10 miliseconds counter
    COUNTER_SECONDS         : USINT; // %S6    seconds counter from RTC
    COUNTER_MINUTES         : USINT; // %S7    minutes counter from RTC
    COUNTER_HOURS           : USINT; // %S8    hours counter from RTC
    COUNTER_DAYS_OF_WEEK    : USINT; // %S9    day of week counter from RTC
    COUNTER_DAYS_OF_MONTH   : USINT; // %S10   days of month counter from RTC
    COUNTER_MONTHS          : USINT; // %S11   months counter from RTC
    COUNTER_YEARS           : USINT; // %S12   years counter from RTC
    PERIOD_PULSE_100MS      : BOOL;  // %S13.0 periodical impulses 100 ms
    PERIOD_PULSE_500MS      : BOOL;  // %S13.1 periodical impulses 500 ms
    PERIOD_PULSE_1SEC       : BOOL;  // %S13.2 periodical impulses 1 s
    PERIOD_PULSE_10SEC      : BOOL;  // %S13.3 periodical impulses 10 s
    PERIOD_PULSE_1MIN       : BOOL;  // %S13.4 periodical impulses 1 min
    PERIOD_PULSE_10MIN      : BOOL;  // %S13.5 periodical impulses 10 min
    PERIOD_PULSE_1HOUR      : BOOL;  // %S13.6 periodical impulses 1 hod
    PERIOD_PULSE_1DAY       : BOOL;  // %S13.7 periodical impulses 1 den
    COUNTER_100MS           : UINT;  // %SW14  counter 100ms
    COUNTER_1SEC            : UINT;  // %SW16  counter 1s
    COUNTER_10SEC           : UINT;  // %SW18  counter 10s
    R_EDGE_100MS            : BOOL;  // %S20.0 rising edge 1x per 100 ms
    R_EDGE_500MS            : BOOL;  // %S20.1 rising edge 1x per 500 ms
    R_EDGE_1SEC             : BOOL;  // %S20.2 rising edge 1x per 1 s
    R_EDGE_10SEC            : BOOL;  // %S20.3 rising edge 1x per 10 s
    R_EDGE_1MIN             : BOOL;  // %S20.4 rising edge 1x per 1 min
    R_EDGE_10MIN            : BOOL;  // %S20.5 rising edge 1x per 10 min
    R_EDGE_1HOUR            : BOOL;  // %S20.6 rising edge 1x per 1 hod
    R_EDGE_1DAY             : BOOL;  // %S20.7 rising edge 1x per 1 den
    F_EDGE_100MS            : BOOL;  // %S21.0 falling edge 1x per 100 ms
    F_EDGE_500MS            : BOOL;  // %S21.1 falling edge 1x per 500 ms
    F_EDGE_1SEC             : BOOL;  // %S21.2 falling edge 1x per 1 s
    F_EDGE_10SEC            : BOOL;  // %S21.3 falling edge 1x per 10 s
    F_EDGE_1MIN             : BOOL;  // %S21.4 falling edge 1x per 1 min
    F_EDGE_10MIN            : BOOL;  // %S21.5 falling edge 1x per 10 min
    F_EDGE_1HOUR            : BOOL;  // %S21.6 falling edge 1x per 1 hod
    F_EDGE_1DAY             : BOOL;  // %S21.7 falling edge 1x per 1 den
    LAST_CYCLE_TIME_100US   : UINT;  // %SW22  last cycle time [x 100 µs]
    S24, S25, S26, S27, S28: BYTE;  // %S24,...,%S28 process control masks
    S29, S30, S31, S32, S33: BYTE;  // %S29,...,%S33 process control masks
    S34                     : BYTE;  // %S34   internal code of error
```

```
    BAT_ERR                 : BOOL;  // %S35.0 backup battery error (low voltage)
    S35_1                   : BOOL;  // %S35.1
    S35_2                   : BOOL;  // %S35.2
    S35_3                   : BOOL;  // %S35.3
    S35_4                   : BOOL;  // %S35.4
    S35_5                   : BOOL;  // %S35.5
    IS_SUMMER_TIME          : BOOL;  // %S35.6 daylight saving time indication
    SUMMER_TIME_REQUEST     : BOOL;  // %S35.7 daylight saving time request
    CPU_TEMPERATURE         : USINT; // %S36   CPU temperature [°C]
    S37                     : BYTE;  // %S37   system function flags
    S38                     : BYTE;  // %S38   user program version
    S39                     : BYTE;  // %S39   user program version
    S40                     : BYTE;  // %S40   firmware version
    S41                     : BYTE;  // %S41   firmware version
    S42                     : BYTE;  // %S42   CPU series
    S43                     : BYTE;  // %S43   flags of PLC behavior
    S44                     : BYTE;  // %S44   compiler type
    S45                     : BYTE;  // %S45   compiler type
    S46                     : BYTE;  // %S46   cycle time warning
    S47                     : BYTE;  // %S47   maximum cycle time
    S48, S49, S50, S51      : BYTE;  // %S48,...,%S51 full code of error
    COUNTER_1MS             : UDINT; // %SL52  counter with step 1 ms
  END_STRUCT;
END_TYPE
```

See also The handbook of PLC Tecomat programmer, chapter 5.3 System registers

2.4 TtecoDateTime type



Library : *SysLib*

TTecoDateTime type is a structure with date and time data. This structure is used by conversion function *TecoDT_TO_DT* and *DT_TO_TecoDT*.

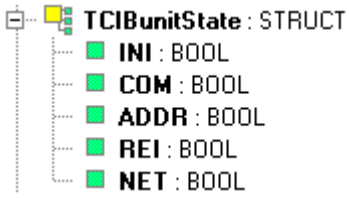
The signification of individual items of the structure *TTecoDateTime* is as follows:

- *year* year (last two digits)
- *month* month
- *day* day
- *hour* hours
- *min* minutes
- *sec* seconds
- *dayOfWeek* day of the week (1 = Monday, 2 = Tuesday, … , 7 = Sunday)
- *milisec* miliseconds

See also Function TecoDT_TO_DT, Function DT_TO_TecoDT

2.5 TCIBunitState type

Library: *SysLib*



TCIBunitState type is a structure carrying information about the status of the unit on the CIB bus which is returned by the function *CIBunitInfo()* as a part of information about the CIB unit.

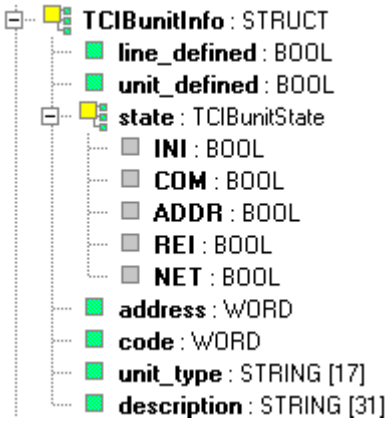
The signification of individual units of the structure *TCIBunitState* is as follows:

- *INI* CIB unit initialization status
  - TRUE CIB unit is initialized
  - FALSE CIB unit initialization failed
- *COM* communication status with the CIB unit
  - TRUE communication is runnig faultlessly
  - FALSE CIB unit does not communicate
- *ADR* status of the CIB unit addressing
  - TRUE unit addressing successful
  - FALSE unit logic address assignment failed
- *REI* flag of reinitialization of the CIB unit
  - TRUE unit reinitialization in progress
  - FALSE unit is under routine run
- *NET* status of CIB unit control
  - TRUE unit is controlled
  - FALSE unit is not controlled

See also Chyba: zdroj odkazu nenalezen, Function CIBunitInfo

2.6 TCIBunitInfo type

Library : *SysLib*



TCIBunitInfo type is a structure carrying information about the unit on the CIB bus which is returned by the function *CIBunitInfo()*.

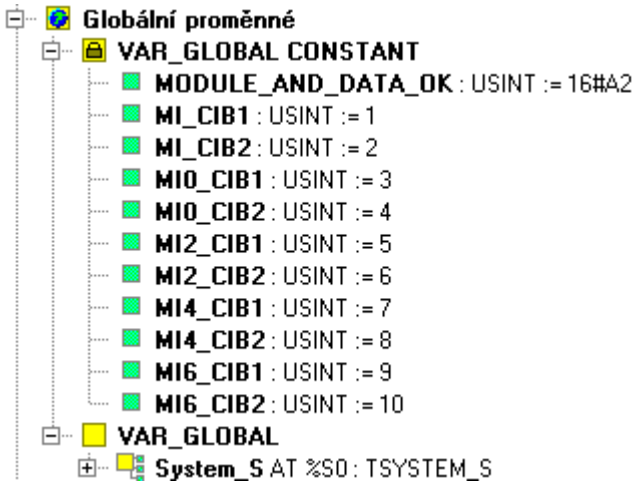
The signification of individual units of the sturcture *TCIBunitState* is as follows:

- *line_defined* CIB line is defined in HW configuration of the PLC
- *unit_defined* CIB unit is defined in HW configuration of the PLC
- *state* status of CIB unit (see TCIBunitState type)
- *address* currently set HW address of the CIB unit
- *code* code of the CIB unit
- *unit_type* CIB unit type attribution
- *description* CIB unit description

See also TCIBunitState type, Function CIBunitInfo

3 GLOBAL VARIABLES AND CONSTANTS

In the SysLib library following global variables and constants are defined:



Constants *MI_CIB1* up to *MI6_CIB2* are used for specification of CIB line within functions *CIBunitInfo* and *SetAddressCIBunit* and have the following meaning:

- *MI_CIB1* CIB line on the basic Foxtrot module (internal module MI2-01M)
- *MI_CIB2* this line is not used within Foxtrot systems
- *MI0_CIB1* external module MI2-02M, address 0, line CIB1
- *MI0_CIB2* external module MI2-02M, address 0, line CIB2
- *MI2_CIB1* external module MI2-02M, address 2, line CIB1
- *MI2_CIB2* external module MI2-02M, address 2, line CIB2
- *MI4_CIB1* external module MI2-02M, address 4, line CIB1
- *MI4_CIB2* external module MI2-02M, address 4, line CIB2
- *MI6_CIB1* external module MI2-02M, address 6, line CIB1
- *MI6_CIB2* external module MI2-02M, address 6, line CIB2

Constant *MODULE_AND_DATA_OK* have a value of 16#A2 and can be used for control of status of PLC IO modules. Each IO module within the PLC has assigned one system registry within the zone from %S100 upto %S227 that indicates immediate IO module status. Value 16#A2 indicates that the IO module is alright. For details see PLC Tecomat programmer handbook, chapter 5.3 System registry, status zone of the peripheral system.

Global variable *System_S* has a structure *TSYSTEM_S* and enables the access to systém registry of the PLC. The signification of particular items is as follows:

```
System_S.S0                     BYTE   %S0     výsledky aritmetických operací
System_S.S1                     BYTE   %S1     výsledky logických operací
System_S.S2_0                   BOOL   %S2.0   stav služebního vstupu SP
System_S.S2_1                   BOOL   %S2.1   stav služebního vstupu MS
System_S.S2_2                   BOOL   %S2.2   režim RUN
System_S.S2_3                   BOOL   %S2.3   teplý restart
System_S.S2_4                   BOOL   %S2.4   studený restart
System_S.OUTPUTS_ARE_ENABLED    BOOL   %S2.5   výstupy odblokovány
System_S.S2_6                   BOOL   %S2.6   první průchod cyklem bez restartu
System_S.CYCLE_TIME_WARNING     BOOL   %S2.7   překrocena první mez doby cyklu
System_S.LAST_CYCLE_TIME_10MS   USINT  %S3     doba minulého cyklu v 10 ms
System_S.CYCLE_COUNTER          USINT  %S4     čítač cyklu
System_S.COUNTER_10MS           USINT  %S5     čítač desítek milisekund
System_S.COUNTER_SECONDS        USINT  %S6     čítač sekund systémového času
System_S.COUNTER_MINUTES        USINT  %S7     čítač minut systémového času
System_S.COUNTER_HOURS          USINT  %S8     čítač hodin systémového času
System_S.COUNTER_DAYS_OF_WEEK   USINT  %S9     čítač dnů v týdnu
System_S.COUNTER_DAYS_OF_MONTH  USINT  %S10    čítač dnů v měsíci
System_S.COUNTER_MONTHS         USINT  %S11    čítač měsiců
System_S.COUNTER_YEARS          USINT  %S12    čítač roků
System_S.PERIOD_PULSE_100MS     BOOL   %S13.0  pulz s periodou 100 ms
System_S.PERIOD_PULSE_500MS     BOOL   %S13.1  pulz s periodou 500 ms
System_S.PERIOD_PULSE_1SEC      BOOL   %S13.2  pulz s periodou 1 s
System_S.PERIOD_PULSE_10SEC     BOOL   %S13.3  pulz s periodou 10 s
System_S.PERIOD_PULSE_1MIN      BOOL   %S13.4  pulz s periodou 1 min
System_S.PERIOD_PULSE_10MIN     BOOL   %S13.5  pulz s periodou 10 min
System_S.PERIOD_PULSE_1HOUR     BOOL   %S13.6  pulz s periodou 1 hod
System_S.PERIOD_PULSE_1DAY      BOOL   %S13.7  pulz s periodou 1 den
System_S.COUNTER_100MS          UINT   %SW14   čítač v 100m
System_S.COUNTER_1SEC           UINT   %SW16   čítač v 1s
System_S.COUNTER_10SEC          UINT   %SW18   čítač v 10s
System_S.R_EDGE_100MS           BOOL   %S20.0  náběžná hrana 1x za 100 ms
System_S.R_EDGE_500MS           BOOL   %S20.1  náběžná hrana 1x za 500 ms
System_S.R_EDGE_1SEC            BOOL   %S20.2  náběžná hrana 1x za 1 s
System_S.R_EDGE_10SEC           BOOL   %S20.3  náběžná hrana 1x za 10 s
System_S.R_EDGE_1MIN            BOOL   %S20.4  náběžná hrana 1x za 1 min
System_S.R_EDGE_10MIN           BOOL   %S20.5  náběžná hrana 1x za 10 min
System_S.R_EDGE_1HOUR           BOOL   %S20.6 náběžná hrana 1x za 1 hod
System_S.R_EDGE_1DAY            BOOL   %S20.7 náběžná hrana 1x za 1 den
System_S.F_EDGE_100MS           BOOL   %S21.0 sestupná hrana 1x za 100 ms
System_S.F_EDGE_500MS           BOOL   %S21.1 sestupná hrana 1x za 500 ms
System_S.F_EDGE_1SEC            BOOL   %S21.2 sestupná hrana 1x za 1 s
System_S.F_EDGE_10SEC           BOOL   %S21.3 sestupná hrana 1x za 10 s
System_S.F_EDGE_1MIN            BOOL   %S21.4 sestupná hrana 1x za 1 min
System_S.F_EDGE_10MIN           BOOL   %S21.5 sestupná hrana 1x za 10 min
System_S.F_EDGE_1HOUR           BOOL   %S21.6 sestupná hrana 1x za 1 hod
System_S.F_EDGE_1DAY            BOOL   %S21.7 sestupná hrana 1x za 1 den
System_S.LAST_CYCLE_TIME_100US  UINT   %SW22  doba minulého cyklu v 100 µs
System_S.S24                    BYTE   %S24   řídící masky procesů
System_S.S25                    BYTE   %S25   řídící masky procesů
System_S.S26                    BYTE   %S26   řídící masky procesů
System_S.S27                    BYTE   %S27   řídící masky procesů
System_S.S28                    BYTE   %S28   řídící masky procesů
System_S.S29                    BYTE   %S29   řídící masky procesů
System_S.S30                    BYTE   %S30   řídící masky procesů
System_S.S31                    BYTE   %S31   řídící masky procesů
```

```
System_S.S32                    BYTE   %S32   řídící masky procesů
System_S.S33                    BYTE   %S33   řídící masky procesů
System_S.S34                    BYTE   %S34   interní kód chyby
System_S.BAT_ERR                BOOL   %S35.0 chyba zálohovací baterie
System_S.S35_1                  BOOL   %S35.1
System_S.S35_2                  BOOL   %S35.2
System_S.S35_3                  BOOL   %S35.3
System_S.S35_4                  BOOL   %S35.4
System_S.S35_5                  BOOL   %S35.5
System_S.IS_SUMMER_TIME         BOOL   %S35.6 indikace letniho času
System_S.SUMMER_TIME_REQUEST    BOOL   %S35.7 žádost o přechod na letní čas
System_S.CPU_TEMPERATURE        USINT  %S36   teplota procesorového modulu [°C]
System_S.S37                    BYTE   %S37   příznaky funkcí systému
System_S.S38                    BYTE   %S38   verze uživatelského programu
System_S.S39                    BYTE   %S39   verze uživatelského programu
System_S.S40                    BYTE   %S40   verze FW systému
System_S.S41                    BYTE   %S41   verze FW systému
System_S.S42                    BYTE   %S42   řada CPU
System_S.S43                    BYTE   %S43   příznaky chování PLC
System_S.S44                    BYTE   %S44   typ překladače
System_S.S45                    BYTE   %S45   typ překladače
System_S.S46                    BYTE   %S46   varovná mez doby cyklu
System_S.S47                    BYTE   %S47   max. mez doby doby cyklu
System_S.S48                    BYTE   %S48   úplný kód chyby PLC
System_S.S49                    BYTE   %S49   úplný kód chyby PLC
System_S.S50                    BYTE   %S50   úplný kód chyby PLC
System_S.S51                    BYTE   %S51   úplný kód chyby PLC
System_S.COUNTER_1MS            UDINT  %SL52  čítač po 1 ms
```

See also PLC Tecomat programmer handbook, chapter 5.3 System registry

4 FUNCTION

The SysLib library contains following functions for operations with actual date and time of the PLC:

- *SetSummerTime* sets the request on automatic daylight saving time change
- *IsSummerTime* tests if the summer time is on
- *SetWinterTime* switch off he request on automatic daylight saving time change
- *IsWinterTime* tests if the winter time is on
- *GetDate* returns actual date
- *GetTime* returns actual time
- *GetDateTime* returns actual date and time
- *GetRTC* upload the date and time from the RTC circuit
- *SetRTC* sets new date and time to the RTC circuit

Other functions are used for conversion of IEC date and time into the structure traditionally used within Tecomat systems:

- *TecoDT_TO_DT* transfers date and time from the structure TTecoDateTime into the IEC DATE_AND_TIME format
- *DT_TO_TecoDT* transfers date and time from the IEC format DATE_AND_TIME into the structure TTecoDateTime

The SysLib library further contains functions for PLC peripheral systém testing that are designed for such a case when the application uses the possibility of I/O modules removal in running or, rather, the possibility to ignore the IO module error:

- *IOSystemInfo* returns whole information about the status of the PLC I/O system
- *ModuleInfo* returns information about current status of one I/O module

Other functions are used for operation with units on the CIB bus:

- *CIBunitInfo* returns information about current status of one CIB unit
- *SetAddressCIBunit* sets new HW address of the CIB unit

Funkcion memcpy, memset and memcmp in the SysLib library is used for memory operations:

- *Memcpy* copies the memory block
- *Memset* fills the memory block with a set constant
- *Memcmp* compares two memory blocks

Funkce IncreaseMaxCycleTime is used to increase limits for cycle time watch:

- *IncreaseMaxCycleTime* single increase of watched PLC cycle time

The last function of the SysLib library is the lock of the PLC application program:

- *ProgramLock* switch on the program protection so, that it can not be decompiled

4.1 Function SetSummerTime

Library : *SysLib*

The *SetSummerTime* function sets the request on automatic transfer between summer and winter time. The *SetSummerTime* function has no input parameters. The output of this function is the value TRUE providing that the requirement was successfully set. In such case, the central unit starts to watch the daylight saving time transfer. The transfer from winter to summer time is undertaken the last Sunday in March so, that the time shifts forward from 02:00 a.m. To 03:00 a.m. The transfer from summer to winter time is undertaken the last Sunday in October so, that the time shifts back from 03:00 a.m. To 02:00 a.m. If the PLC system is not on at this time, the transfer will be undertaken during the nearest switch on sequence.

The example of the program with the SetSummerTime function call :

```
PROGRAM SummerTimeExample
  VAR
    tmp     : BOOL;
    message : STRING;
  END_VAR

  // request for atomatic change between summer and winter time
  tmp := SetSummerTime();

  IF IsSummerTime() THEN
    message := 'Now is summer time';
  ELSE
    message := 'Now is winter time';
  END_IF;

END_PROGRAM
```

See also Function IsSummerTime, Function SetWinterTime, Function IsWinterTime

4.2 Function IsSummerTime

Library : *SysLib*

The *IsSummerTime* function tests whether the summer time is currently on. The *IsSummer-Time* function has no input parameters. Function returns the value TRUE if the summer time is on. If the winter time is on, it returns the value FALSE.

The example of the program with the *IsSummerTime* function call:

```
PROGRAM SummerTimeExample
  VAR
    tmp     : BOOL;
    message : STRING;
  END_VAR

  // request for atomatic change between summer and winter time
  tmp := SetSummerTime();

  IF IsSummerTime() THEN
    message := 'Now is summer time';
  ELSE
    message := 'Now is winter time';
  END_IF;

END_PROGRAM
```

See also Function SetSummerTime, Function SetWinterTime, Function IsWinterTime

4.3 Function SetWinterTime

Library : *SysLib*

The *SetWinterTime* function swithces off the request on automatic daylight saving time transfer. The *SetWinterTime* function has no input parameters. The output of this function is the value TRUE providing that the request on automatic time transfer was successfully switched off.

The example of the program with the *SetWinterTime* function call:

```
PROGRAM ExampleWinterTime
  VAR
    tmp     : BOOL;
    message : STRING;
  END_VAR

  // no atomatic change between summer and winter time
  tmp := SetWinterTime();

  IF IsWinterTime() THEN
    message := 'Now is winter time';
  ELSE
    message := 'Now is summer time';
  END_IF;
END_PROGRAM
```

See also Function SetSummerTime, Function IsWinterTime, Function IsWinterTime

4.4 Function IsWinterTime

Library : *SysLib*

The *IsWinterTime* function tests whether the winter time is currently on. The *IsWinterTime* function has no input parameters. Function returns the value TRUE if the winter time is on, otherwise, it returns the value FALSE.

The example of the program with the *IsWinterTime* function call :

```
PROGRAM ExampleWinterTime
  VAR
    tmp     : BOOL;
    message : STRING;
  END_VAR

  // no atomatic change between summer and winter time
  tmp := SetWinterTime();

  IF IsWinterTime() THEN
    message := 'Now is winter time';
  ELSE
    message := 'Now is summer time';
  END_IF;
END_PROGRAM
```

See also Function SetWinterTime, Function SetSummerTime, Function IsSummerTime

4.5 Function GetDate

Library : *SysLib*

The *GetDate* function returns actual date. The *GetDate* function has no input parameters. Date, that the function returns, is set at the beginning of each cycle of the PLC. During the cycle the value does not change.

The example of the program with the *GetDate* function call:

```
PROGRAM GetDateExample
  VAR
    presentDate : DATE;
  END_VAR

  presentDate   := GetDate();
  IF presentDate = D#2008-12-24 OR
     presentDate = D#2009-12-24 OR
     presentDate = D#2010-12-24
  THEN
    message := 'Today is Christmas Eve';
  END_IF;
END_PROGRAM
```

See also Function GetDateTime, Function GetTime

4.6 Function GetTime

Library : *SysLib*

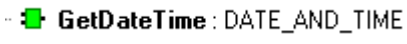
The *GetTime* function returns actual PLC time. The *GetTime* function has no input parameters. Time, that the function returns, is set at the beginning of each PLC cycle. During the cycle the value does not change.

The example of the program with the *GetTime* function call:

```
PROGRAM GetTimeExample
  VAR
    timePLC  : TIME;
    greeting : STRING;
  END_VAR

  timePLC := GetTime();
  IF timePLC > T#06:00:00 AND timePLC < T#12:00:00 THEN
    greeting := 'Good morning';
  ELSE
    IF timePLC >= T#12:00:00 AND timePLC < T#18:00:00 THEN
      greeting := 'Good afternoon';
    ELSE
      IF timePLC < T#23:59:59 THEN
        greeting := 'Good evening';
      ELSE
        greeting := 'Good night';
      END_IF;
    END_IF;
  END_IF;
END_PROGRAM
```

See also Function GetDate, Function GetDateTime

4.7 Function GetDateTime



Library : *SysLib*

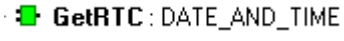
The *GetDateTime* function returns actual PLC date and time. The *GetDateTime* function has no input parameters. Time and date, the function returns, is set at the beginning of each PLC cycle. During the cycle the value does not change.

The example of the program with the *GetDateTime* function call:

```
PROGRAM GetDateTimeExample
  VAR
    dateTimePLC  : DATE_AND_TIME;
    dateTime     : TTecoDateTime;
    dayOfWeek    : STRING;
  END_VAR

  dateTimePLC := GetDateTime();
  // conversion DATE_AND_TIME to struct TTecoDateTime
  dateTime    := DT_TO_TecoDT(IEC_DT := dateTimePLC);
  CASE dateTime.dayOfWeek OF
    1 : dayOfWeek := 'Monday';
    2 : dayOfWeek := 'Tuesday';
    3 : dayOfWeek := 'Wednesday';
    4 : dayOfWeek := 'Thursday';
    5 : dayOfWeek := 'Friday';
    6 : dayOfWeek := 'Saturday';
    7 : dayOfWeek := 'Sunday';
  END_CASE;
END_PROGRAM
```

See also Function GetDate, Function GetTime, Function GetRTC

4.8 Function GetRTC



Library : *SysLib*

The *GetRTC* function upload date and time right from the RTC circuit in the PLC. The *GetRTC* function has no input parameters. Regarding the fact that time within the RTC circuit is changed continuously, two calls of the *GetRTC* function can return different value in the same cycle. The *GetRTC* function thus returns date and time that was at the moment of the call of this function (in comparison to the *GetDateTime* function that returns date and time that was at the beginning of the cycle where the function was called).

The example of the program with the *GetRTC* function call:

```
PROGRAM GetRTCexample
  VAR
    dateTimeRTC   : DATE_AND_TIME;
    message       : STRING;
  END_VAR

  dateTimeRTC := GetRTC();
  message     := 'RTC date and time : ' + DT_TO_STRING(dateTimeRTC);
END_PROGRAM
```

See also Function GetDateTime, Function SetRTC

4.9 Function SetRTC

Library : *SysLib*

The *SetRTC* function returns new date and time into the RTC circuit. The input parameter is new time and date. The function returns TRUE providing the new time is set.

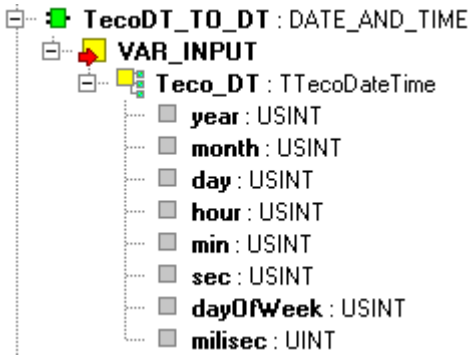
As the following example shows, the *SetRTC* function is neccessary to be called once only, the best on the edge of the variable that represents the requirement on the new time and date entry. The value of new time and date is then typically set from the operator´s panel.

The example of the program with the *SetRTC* function call:

```
PROGRAM SetRTCexample
  VAR
    newDateTimeRTC   : DT;
    setTime          : R_TRIG;
    setNewTime       : BOOL;
    tmp              : BOOL;
  END_VAR

  setTime( CLK := setNewTime);
  IF setTime.Q THEN
    tmp := SetRTC( PDT := newDateTimeRTC);
  END_IF;
END_PROGRAM
```

See also Function GetRTC, Function GetDateTime

4.10 Function TecoDT_TO_DT



Library : *SysLib*

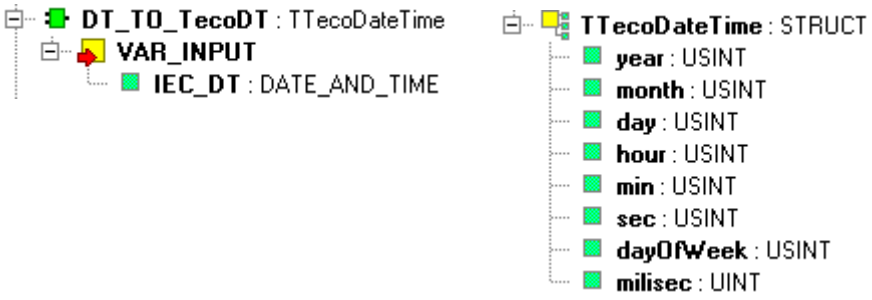
The *TecoDT_TO_DT* function transfers date and time from the structure TTecoDateTime to the IEC format DATE_AND_TIME.

The example of the program with the *TecoDT_TO_DT* function call:

```
PROGRAM TecoDT_TO_DT_example
  VAR
    dateTimePLC     : DATE_AND_TIME;
    dateTime        : TTecoDateTime;
  END_VAR

  dateTime.year     := 09;
  dateTime.month    := 11;
  dateTime.day      := 22;
  dateTime.hour     := 12;
  dateTime.min      := 34;
  dateTime.sec      := 56;
  dateTime.milisec  := 500;
  dateTimePLC       := TecoDT_TO_DT( dateTime);

END_PROGRAM
```

See also TtecoDateTime type, Function DT_TO_TecoDT

4.11 Function DT_TO_TecoDT



Library : *SysLib*

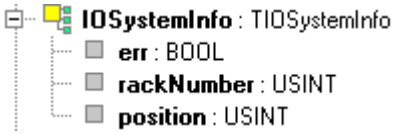
The *DT_TO_TecoDT* function transfers date and time from the IEC format DATE_AND_TIME into the structure TTecoDateTime. The advantage of this structure is that there can be operated individually with only some items of time and date as the following example shows.

The example of the program with the *DT_TO_TecoDT* function call:

```
PROGRAM DT_TO_TecoDT_example
  VAR
    dateTimePLC   : DATE_AND_TIME;
    dateTime      : TTecoDateTime;
    message       : STRING;
  END_VAR

  dateTimePLC   := GetDateTime();
  dateTime      := DT_TO_TecoDT( dateTimePLC);
  IF dateTime.month = 05 AND dateTime.day = 30 THEN
    message := 'Birthday of my wife! Do not forget to buy flowers!';
  END_IF;
END_PROGRAM
```

See also TtecoDateTime type, Function TecoDT_TO_DT

4.12 Function IOSystemInfo



Library : *SysLib*

The *IOSystemInfo* function returns whole information about the status of the PLC I/O systém and is designed for such cases when the application uses the possibility of I/O modules removal in running or, rather, the possibility to ignore the IO module error.

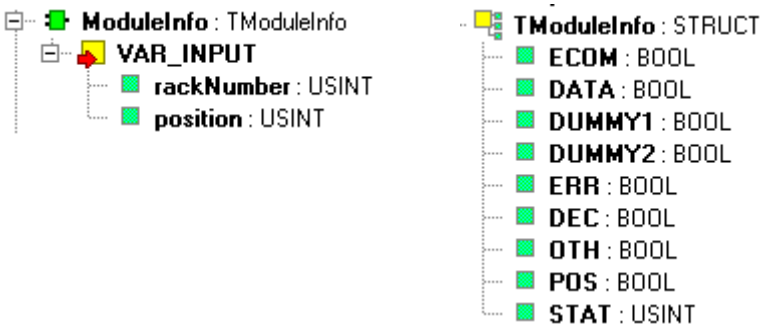
The *IOSystemInfo* function has no input parameters. The output of this function is structure of TIOSystemInfo type. The function check the PLC IO systém status according to what is set in HW configuration of the PLC. In case that IO system is alright, the return value on the output *err* is FALSE and outputs *rackNumber* and *position* have no meaning. If there is some problem within the IO system (i.e. IO module required in HW configuration is missing) then the output *err* is set to TRUE, output variable *rackNumber* indicates the rack number and variable *position* indicates the position on the rack where the problem was located. The function *ModuleInfo* can be used for detailed specificaton.

The example of the program with the *IOSystemInfo* function call:

```
PROGRAM IOSysteInfoExample
  VAR
    IO_Status     : TIOSystemInfo;
    ModuleStatus  : TModuleInfo;
  END_VAR

  // check IO system PLC
  IO_Status := IOSystemInfo();
  IF IO_Status.err THEN
    // any modul has problem
    ModuleStatus := ModuleInfo( IO_Status.rackNumber, IO_Status.position);
  END_IF;
END_PROGRAM
```

See also TIOSystemInfo type, TmoduleInfo type , Function ModuleInfo

4.13 Function ModuleInfo



Library : *SysLib*

The *ModuleInfo* function returns information about actual status of one I/O module. The input parameter *rackNumber* indicates the rack number and the parameter *position* indicates the position number for which the module information is returned. The function returns the structure TModuleInfo.

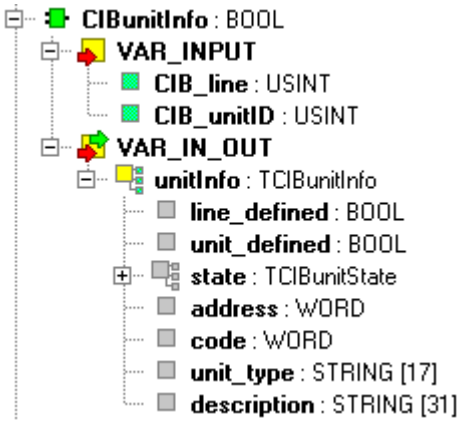
The example of the program with the *ModuleInfo* function call:

```
PROGRAM ModuleInfoExample
  VAR
    IO_Status     : TIOSystemInfo;
    ModuleStatus  : TModuleInfo;
  END_VAR

  // check IO system PLC
  IO_Status := IOSystemInfo();
  IF IO_Status.err THEN
    // any modul has problem
    ModuleStatus := ModuleInfo( IO_Status.rackNumber, IO_Status.position);
  END_IF;
END_PROGRAM
```

See also TIOSystemInfo type, TmoduleInfo type , Function IOSystemInfo

4.14 Function CIBunitInfo

Library : *SysLib*



The *CIBunitInfo* function is used for retrieving information about current status of one unit on the CIB bus. The input parameter *CIB_line* specify the CIB line (see constants *MI_CIB1* up to *MI6_CIB2*) and parameter *CIB_unitID* shows the position number for which information about the CIB unit is returned. The function returns the value TRUE providing information about the CIB unit are successfully retrieved. Obtained information are saved to the variable set by the parameter *unitInfo*. If the CIB unit is not declared within HW configuration of the PLC, function *CIBunitInfo* returns FALSE.

This function is supported within Foxtrot central units from the version 5.1. The function is filed to the library SysLib from version 19.

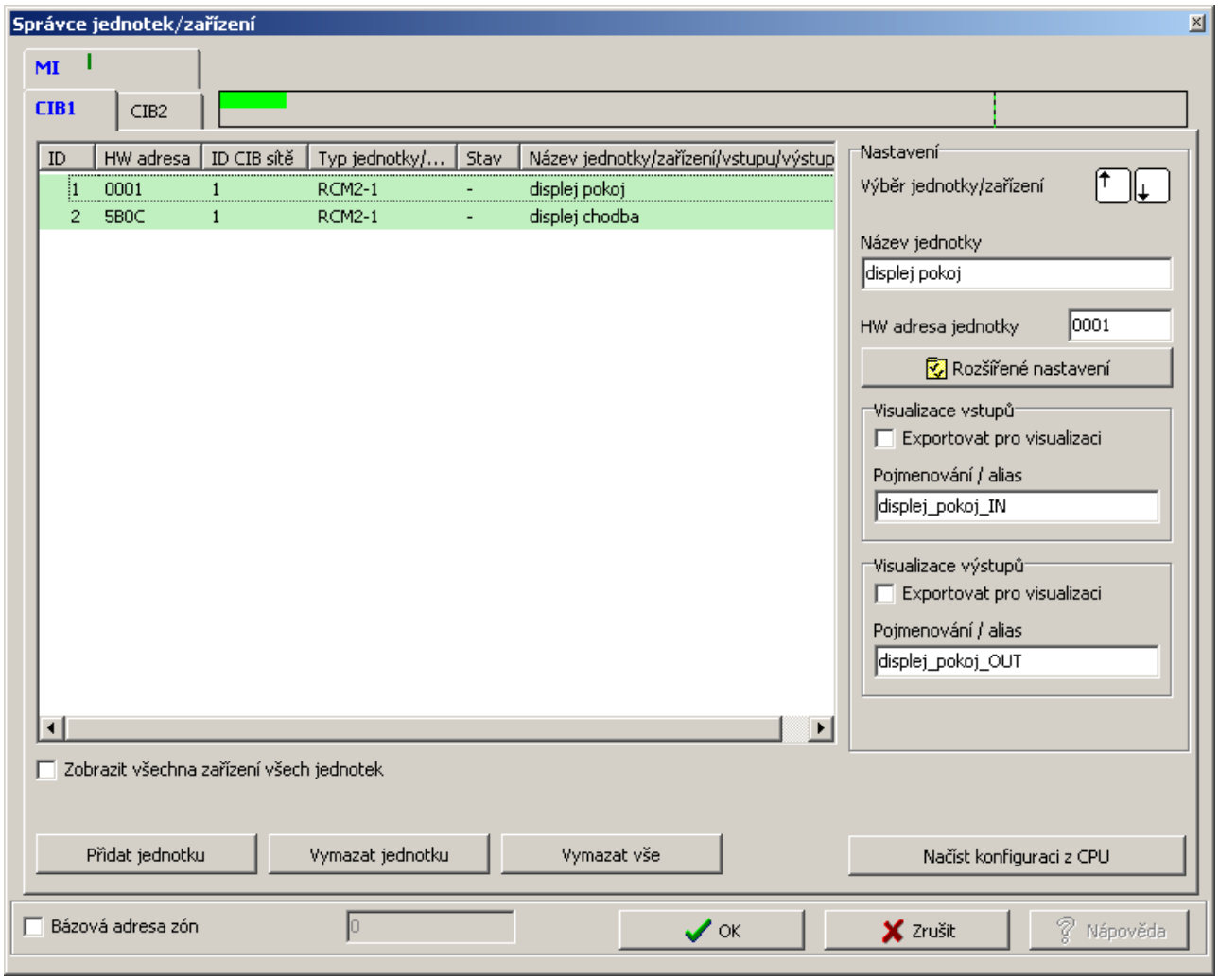
The example of the program with the *CIBunitInfo* function call:

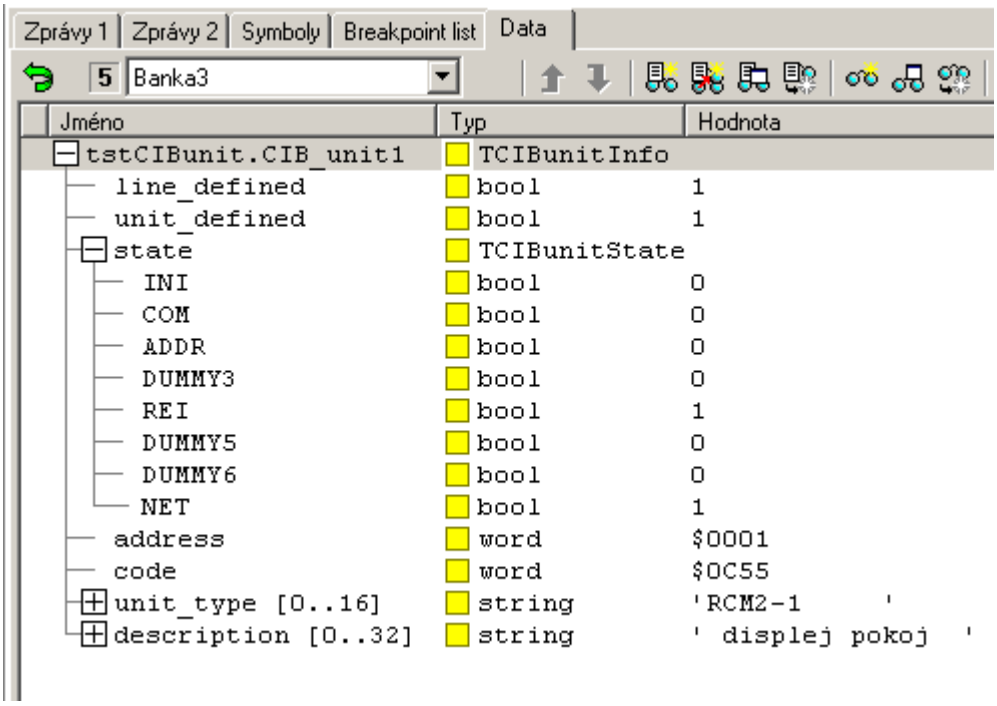
```
PROGRAM CIBunitInfoExample
  VAR
    CIB_unit1, CIB_unit2 : TCIBunitInfo;
    result1, result2     : BOOL;
  END_VAR

  result1 := CIBunitInfo( CIB_line := MI_CIB1, CIB_unitID := 1,
                          unitInfo := CIB_unit1);
  result2 := CIBunitInfo( CIB_line := MI_CIB1, CIB_unitID := 2,
                          unitInfo := CIB_unit2);
END_PROGRAM
```

The program shown here tests the status of the first two CIB units connected to the CIB line of the basic module of the Foxtrot systém. Units are within the PLC program set so as the picture of the Unit/device manager dialogue shows.

Another picture then shows values of the variable *CIB_unit1* that correspond to the status of the first defined CIB unit. Values corresponds to the situation when the CIB unit in the PLC program is defined but does not communicate (e.g. Because it is disconnected or has a different HW address than is set in the above mentioned dialogue).

If the declaration of CIB unit in the program is missing, the function *CIBunitInfo* will return FALSE and variable *CIB_unit1* will be zero.

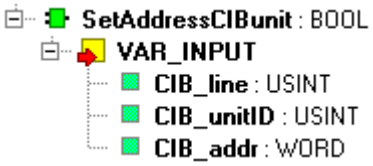

Správce jednotek/zařízení
MI
CIB1
CIB2
ID
HW adresa
ID CIB sítě
Typ jednotky/...
Stav
Název jednotky/zařízení/vstupu/výstup
1 0001 1 RCM2-1 - displej pokoj
2 5B0C 1 RCM2-1 - displej chodba
Nastavení
Výběr jednotky/zařízení
Název jednotky
displej pokoj
HW adresa jednotky
0001
Rozšířené nastavení
Visualizace vstupů
Exportovat pro visualizaci
Pojmenování / alias
displej_pokoj_IN
Visualizace výstupů
Exportovat pro visualizaci
Pojmenování / alias
displej_pokoj_OUT
Zobrazit všechna zařízení všech jednotek
Přidat jednotku
Vymazat jednotku
Vymazat vše
Načíst konfiguraci z CPU
Bázová adresa zón
0
OK
Zrušit
Nápověda


Zprávy 1
Zprávy 2
Symboly
Breakpoint list
Data
5
Banka3
Jméno
Typ
Hodnota
tstCIBunit.CIB_unit1 TCIBunitInfo
line_defined bool 1
unit_defined bool 1
state TCIBunitState
INI bool 0
COM bool 0
ADDR bool 0
DUMMY3 bool 0
REI bool 1
DUMMY5 bool 0
DUMMY6 bool 0
NET bool 1
address word $0001
code word $0C55
unit_type [0..16] string 'RCM2-1 '
description [0..32] string ' displej pokoj '

See also TCIBunitState type, Chyba: zdroj odkazu nenalezen

4.15 Function SetAddressCIBunit

Library : *SysLib*



The *SetAddressCIBunit* function sets new HW address of the CIB unit. It can be used in a situation when we want to use similar PLC program for more applications that differ from each other by HW address of CIB units.

The input parameter *CIB_line* specify the CIB line (see constants *MI_CIB1* up to *MI6_CIB2*) and parameter *CIB_unitID* indicates for which CIB unit the new address will be set. The function returns the value TRUE providing the CIB unit address is successfully set.

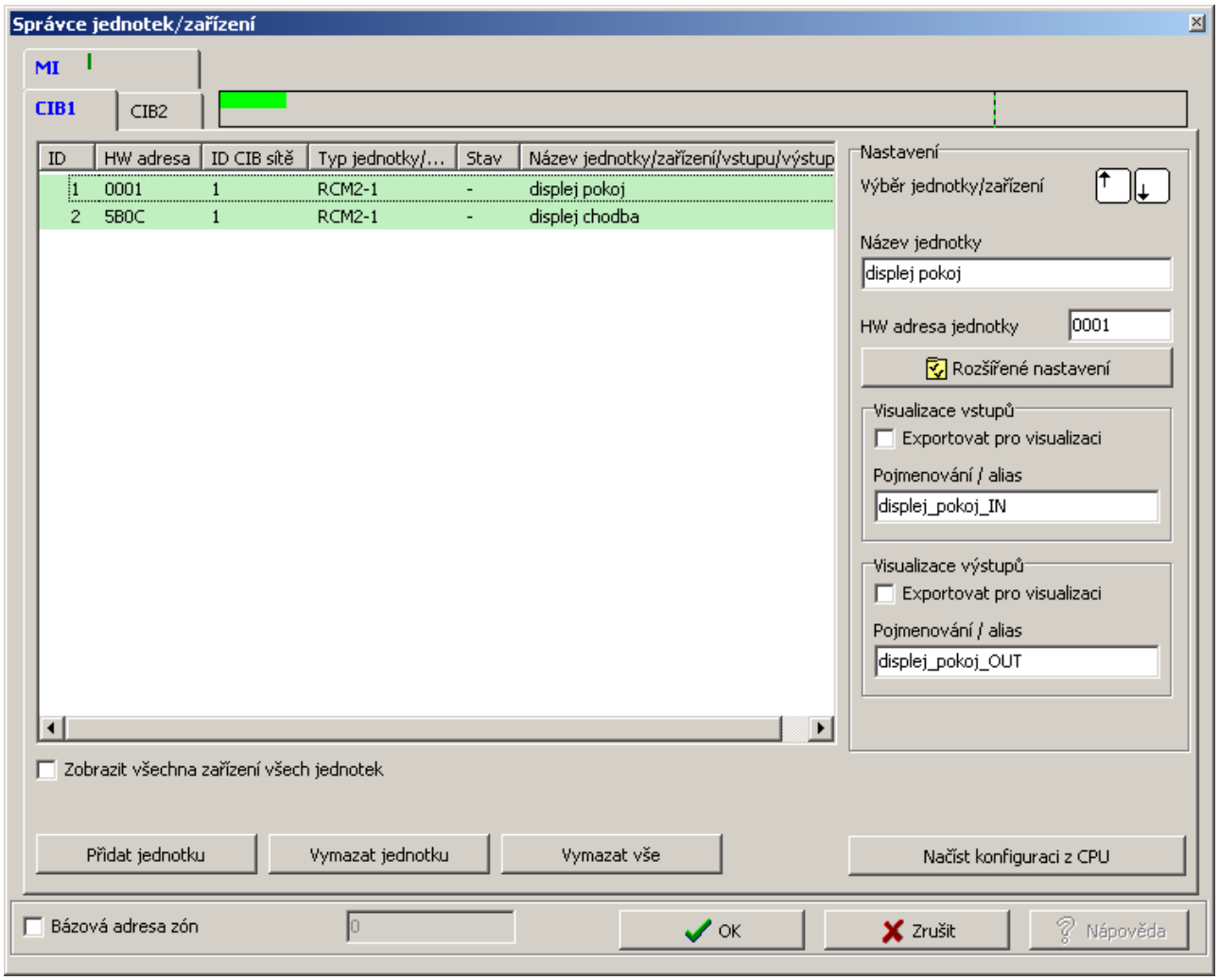
This function is supported within Foxtrot central units from the version 5.1. The function is filed to the library SysLib from version 19.

The example of the program with the *SetAddressCIBunit* function call:

```
VAR_GLOBAL CONSTANT
  NUM_CIB_UNIT : USINT := 4;
END_VAR

TYPE
  TcustomCIB : STRUCT
    valid    : BOOL;
    address  : WORD;
    code     : WORD;
  END_STRUCT;
END_TYPE

VAR_GLOBAL RETAIN
  customCIB : ARRAY[1..NUM_CIB_UNIT] OF TcustomCIB;
END_VAR

PROGRAM TestCIB
  VAR
    i        : USINT;
    CIB_unit : ARRAY[1..NUM_CIB_UNIT] OF TCIBunitInfo;
    id       : USINT;
    newAdr   : WORD;
    setAdr   : BOOL;
    result   : BOOL;
  END_VAR

  // read info about CIB units and set customer address
  FOR i := 1 TO NUM_CIB_UNIT DO
    IF CIBunitInfo( CIB_line := MI_CIB1, CIB_unitID := i,
                    unitInfo := CIB_unit[i]) THEN
      IF not customCIB[i].valid OR customCIB[i].code <> CIB_unit[i].code THEN
        customCIB[i].address := CIB_unit[i].address;
        customCIB[i].code    := CIB_unit[i].code;
        customCIB[i].valid   := TRUE;
```

```
      ELSE
        IF CIB_unit[i].address <> customCIB[i].address THEN
          SetAddressCIBunit(CIB_line := MI_CIB1, CIB_unitID := i,
                            CIB_addr := customCIB[i].address);
        END_IF;
      END_IF;
    END_IF;
  END_FOR;

  // set new customer address of CIB unit
  IF setAdr THEN
    IF id > 0 AND id < NUM_CIB_UNIT+1 THEN
      IF newAdr <> CIB_unit[id].address THEN
        result := SetAddressCIBunit(CIB_line := MI_CIB1,
           CIB_unitID := id, CIB_addr := newAdr);
        IF result THEN
          customCIB[id].address := newAdr;
        END_IF;
      END_IF;
    END_IF;
    setAdr := FALSE;
  END_IF;

END_PROGRAM
```

Lets suppose, there are two RCM2-1 units in the PLC program defined connected to the CIB bus of the basic module of the Foxtrot system (see the following dialogue of the Mosaic environment).

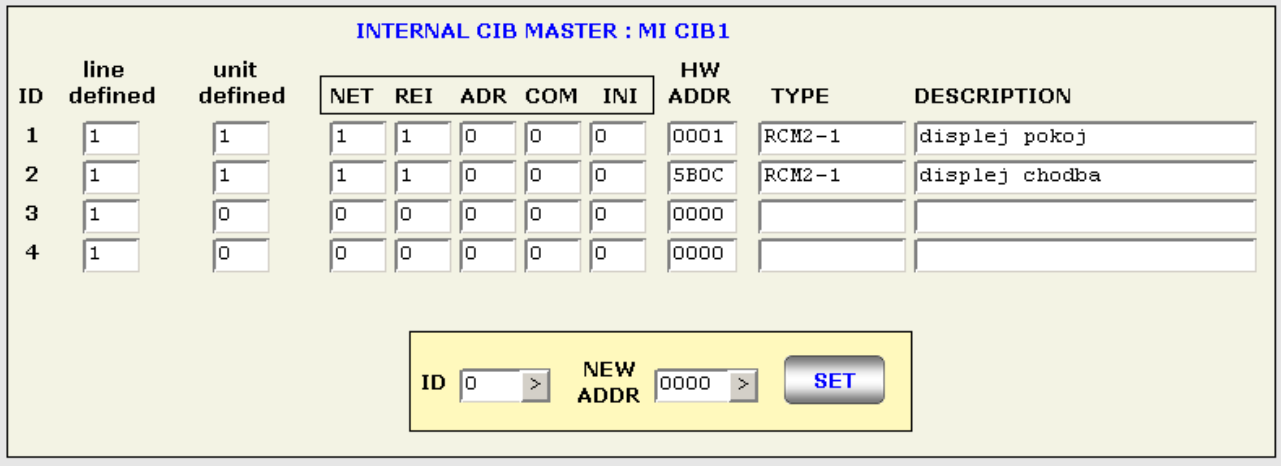


The task of the program in the example is to enable the change of the HW address of CIB units from the operator´s panel or from the web page so, that is is not neccessary while changing the address to compile the program for PLC again. There is a back-up variable *customCIB* set up in the program that contains information about current setup of four CIB units. During the cold restart, there is saved information about the code and the HW address of CIB units that correspond to the setup of the above stated dialogue „Units/devices manager“ so the content of the variable *customCIB* corresponds to the PLC program. On the contrary, during the warm restart, the HW address of CIB units is set according to the variable *customCIB*.

The address of the CIB unit can be changed if we set from the operator´s panel a new address into the variable *newAdr*. The number of the CIB unit, the address of which we want to change, shall be entered into the variable *id* and the variable *setAdr* shall be set to TRUE. If the CIB unit is defined within the PLC program, the function *SetAddressCIBunit* sets new HW address according to the variable *newAdr* and simultaneously enters the new address into the variable *customCIB*. Therefore, during the next warm restart of the PLC, the HW address from the operator´s panel will be used and not the address given by the PLC program. Consequently, even during PLC program changes when the new code is loaded into the PLC, HW addresses saved in the variable *customCIB* will be used for CIB units.

The given example allows to set (change) HW addresses of CIB units without the neccessity to change the PLC program. Therefore, we can use the same PLC program for more applications that differs in HW addresses of CIB units.

The new HW address of CIB unit can also be set, for example, vrom the web page shown in the following picture. On this web page, there is displayed the status of four CIB units on the MI_CIB1 line and setting fields in a yellow frame enable to set the new HW address.

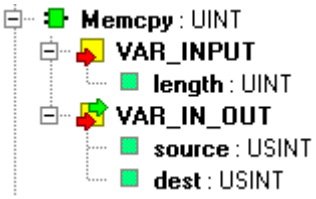


In the column „line defined“ are displayed variables *TestCIB.CIB_unit[i].line_defined*, in the column „unit defined“ are displayed variables *TestCIB.CIB_unit[i].unit_defined*, columns „NET“ „REI“ „ADR“ „COM“ and „INI“ contain corresponding items of the variable *TestCIB.-CIB_unit[i].state* (so *TestCIB.CIB_unit[i].state.NET*, etc.). Column „HW ADR“ displays variables *TestCIB.CIB_unit[i].address*, column „TYPE“ displays variables *TestCIB.CIB_unit[i].type* and finally column „DESCRIPTION“ displays variables *TestCIB.CIB_unit[i].description*. The status of these variables is constantly updated by function *CIBunitInfo()*. All hitherto stated variables are „Read Only“.

The setting of the new HW address of the CIB unit is undertaken via the setting field „ID“ which is coupled with the variable *TestCIB.id*, further via the setting field „NEW ADR“ that enables the change of the variable *TestCIB.newAdr* and finally via two condition picture „SET“ that enables to set the variable *TestCIB.setAdr* to the value TRUE.

See also TCIBunitState type, Chyba: zdroj odkazu nenalezen, Function CIBunitInfo

4.16 Function Memcpy



Library : *SysLib*

The *Memcpy* function will copy the memory block. Input variable *length* determines the lenght of the coppied block in bytes, the variable *source* specifies where the copy will be from and variable *dest* determines where it will be coppied to.

Variables description :

	Variable	*Type*	*Signification*
VAR_INPUT			
	length	UINT	The lenght of the coppied block in bytes
VAR_IN_OUT			
	source	USINT	The source where to copy from
	dest	USINT	The destination where to copy to
Memcpy			
	Release value	UINT	The number of bytes coppied

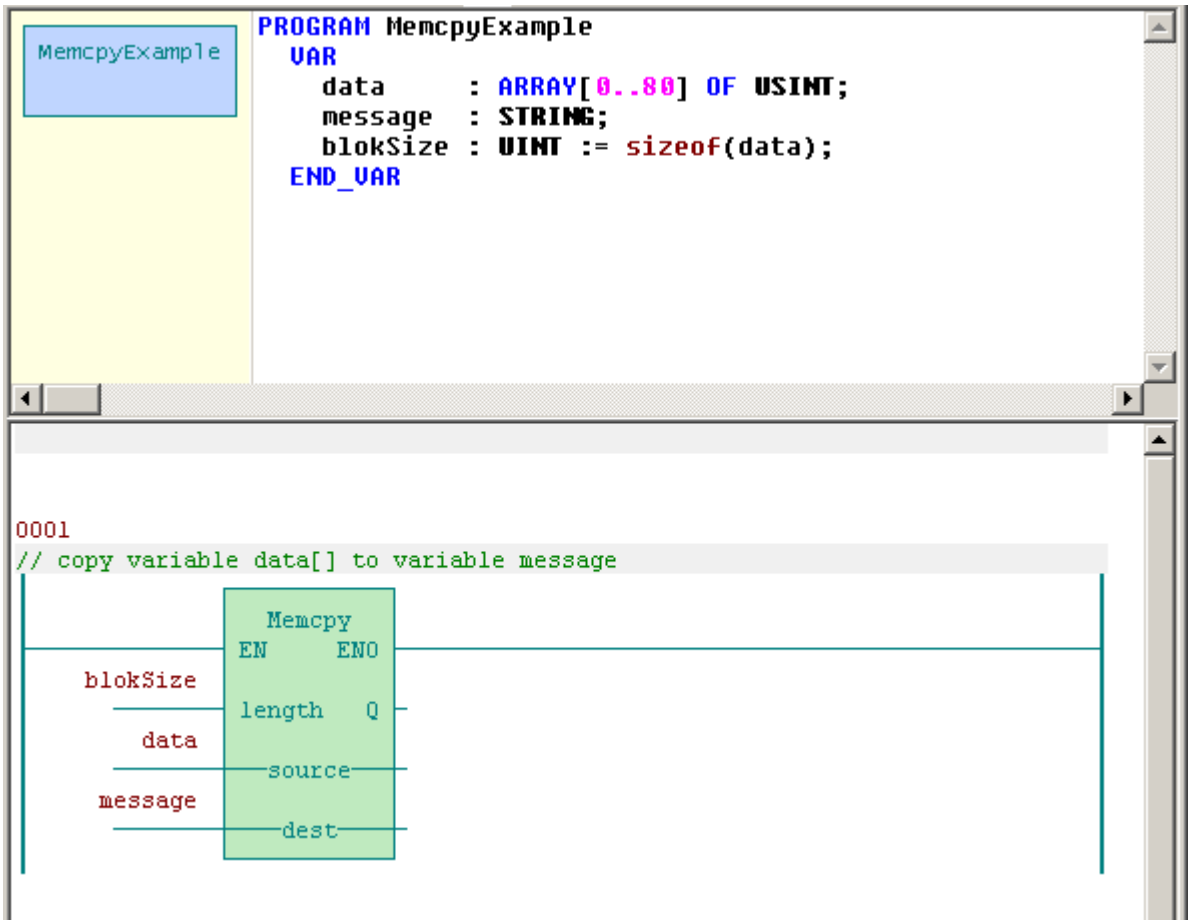
With regards that variables *source* and *dest* are of the USINT type, it can appear at the first sight that the *Memcpy* function can not be used for nothing else than for copying of one memory byte what practically makes no sense. Mentioned variables, however, are of the class VAR_IN_OUT which means that during the *Memcpy* function call they will transfer addresses of variables associated to *source* and *dest* (not values of these variables). Therefore, the problem how to persuade the ST compiler not to control the data type of variables *source* and *dest* is only in question. The function *void()* is used for this case which reset the check of the data type within variables of the class VAR_IN_OUT. The *void()* function thus enables copying variables of an optionable type using the function *Memcpy*. On the other hand, it is vital to carefully choose the size of the memory block coppied. If the size of the variable *dest* will be lower than the value transferred within the parameter *length*, the *Memcpy* function will overwrite variables located behind the variable *dest* in the memory. The compiler can not indicate this problem because the type check is switched off by the function *void()*. Only the programmer is responsible for the correctness.

ATTENTION ! Variables *source* and *dest* must not be of the BOOL type **!** Function *Memcpy* will not function properly if some of the parameters *source* or *dest* is of the BOOL type.

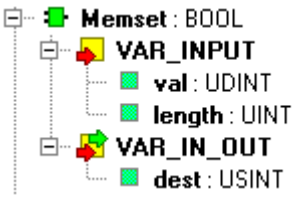
The following example shows how to copy the variable *data[]*, which is an USINT type that have 81 items, into the variable *message* that is of a STRING type (default size is 80 characters plus one for the terminating zero of the string). Variables *data* and *message* must have the same size.

```
PROGRAM MemcpyExample
  VAR
    data      : ARRAY[0..80] OF USINT;
    message   : STRING;
    size      : UINT;
  END_VAR

  // copy variable data[] to variable message
  size := Memcpy( length := sizeof(data),
                  source := void(data[0]),
                  dest   := void(message));
END_PROGRAM
```

The same program will appear in the LD language as follows:



4.17 Function Memset



Library : *SysLib*

The *Memset* function will fill the memory block with the set value.

Variable description :

	Variable	*Type*	*Signification*
VAR_INPUT			
	val	UDINT	Value that is entered to the memory block
	length	UINT	The lenght od the memory block in bytes
VAR_IN_OUT			
	dest	USINT	Variable that will be filled with a constant
Memset			
	Release value	BOOL	TRUE after successful function fulfilment

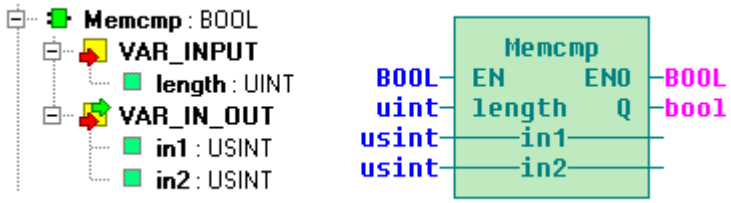
In the following example the *Memset* function will fill the variable *message* (STRING type) with the sign 16#20 which is the space code. The disagreement of data types in the parameter *dest,* where the parameter of USINT type is required and the variable *message* of the STRING type is necessary to be filled, is solved by using the function *void()* during parameter transfer. This function enables the transfer of the parameter *dest* and other data type, not only of the USINT type as the *Memset* function defines.

ATTENTION ! Variable *dest* must not be of the BOOL type **!** The *Memset* function will not function properly if the parameter *dest* is of the BOOL type.

```
PROGRAM MemsetExample
  VAR
    tmp       : BOOL;
    message   : STRING;
  END_VAR

  tmp := Memset( val    := 16#20202020,
                 length := sizeof(message)-1,
                 dest   := void(message));
END_PROGRAM
```

4.18 Function Memcmp

Library : *SysLib*



The *Memcmp* function compares together two memory blocks.

This function is located in the SysLib library from version 2.1 and is supported within central units of C,G and K ranks (all procesors of systems TC650, TC700 and Foxtrot) in all versions.

Variable description :

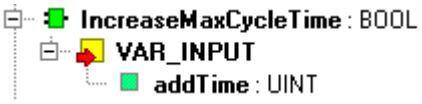
	Variable	*Type*	*Signification*
VAR_INPUT			
	length	UINT	Lenght of compared block in bytes
VAR_IN_OUT			
	in1	USINT	First compared block
	in2	USINT	Second compared block
Memcmp			
	Release value	BOOL	TRUE if the memory block are identical

In the following example the *Memsmp* function compares the variable *message* (STRING type) with the variable *data* (ARRAY OF USINT type). . The disagreement of data types in parameters *in1* and *in2,* where the parameter of USINT type is required is solved by using the function *void()* during parameter transfer.

ATTENTION ! Variables *in1* and *in2* must not be of the BOOL type**!** The *Memsmp* function will not function properly if some of parameters *in1 or in2* is of the BOOL type.

```
PROGRAM MemcmpExample
  VAR
    data      : ARRAY[0..80] OF USINT;
    message   : STRING;
    result    : BOOL;
  END_VAR

  // compare variable data[] and variable message
  result := Memcmp( length := sizeof(data),
                    in1 := void(data[0]), in2 := void(message));
END_PROGRAM
```

4.19 Function IncreaseMaxCycleTime



Library : *SysLib*

The *IncreaseMaxCycleTime* function will increase by a single application the watched PLC cycle time. This function is designed for such cases when, for example, the initialization of the application program lasts longer time then the set PLC cycle time and from the point of view of the application it is not suitable to use the watched cycle time. The input parameter addTime determines how many miliseconds should be added to increase the watched time of currently running PLC cycle.

This function is located in the SysLib library from version 1.5 and is supported within central units of K rank (TC700 CP-7004, Foxtrot) from version 2.8. Maximum cycle time within these systems is 750 ms.

The example of the program with the *IncreaseMaxCycleTime* function call:

```
PROGRAM RestartExample
  VAR
    tmp : BOOL;
  END_VAR

  // increase max. limit for current PLC cycle time
  tmp := IncreaseMaxCycleTime( addTime := 200);   // plus 200 ms

  // next restart activities
  // ...
END_PROGRAM
```

4.20 Function ProgramLock

Library : *SysLib*

The *ProgramLock* function locks the PLC application program so, that it is not possible to undertake the reverse compilation in the Mosaic environment. The function has no input parameters. Release value is always TRUE.

This function is located in the SysLib library from version 2.0 and is supported within central units of C,G and K ranks (all procesors of systems TC650, TC700 and Foxtrot) in all versions.

The example of the program with the *ProgramLock* function call:

```
PROGRAM prgMainExample
  VAR
    tmp : BOOL;
  END_VAR

  // lock application program
  tmp := ProgramLock();       // guard program

  // next activities
  // ...
END_PROGRAM
```

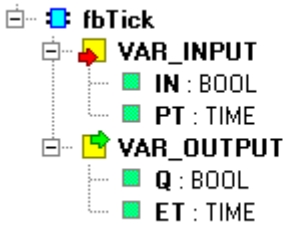
5 FUNCTION BLOCKS

The SysLib library contains following function blocks:

Function block	*Description*
fbTick	Timer periodically generating a puls for the period of one cycle

5.1 Function block fbTick

Library : *SysLib*



The function block *fbTick* is a timer that generates periodically pulses for the period of one cycle. The input in allows pulse generating, period of generation pulses is determined by the input PT.

This function block is supported within all central units of K rank (TC700 CP-7004, Foxtrot) from version 1.0. In the SysLib library is the block located from version SysLib_v21.

Variable description :

	Variable	*Type*	*signification*
VAR_INPUT			
	IN	BOOL	Pulses on the outpu Q are generated if the input IN has the value TRUE
	PT	TIME	Output pulses period
VAR_OUTPUT			
	Q	BOOL	Output puls with duration of one PLC cycle (scan)
	ET	TIME	Continuous time within the period

The example of the program with the *fbTick* function call :

```
PROGRAM TickExample
  VAR
    tickEnable : BOOL;
    tick       : fbTick;
    output     : BOOL;
  END_VAR

  tick( IN := tickEnable, PT := T#1s);
  IF tick.Q THEN
    output := NOT output;
  END_IF;
END_PROGRAM
```

The following picture shows the behaviour of the function block fbTick in graphical appearance.