## Inhaltsverzeichnis

1 Foreword ............................................................................................................................. 5  
   1.1 Notes on the documentation ....................................................................................... 5  
   1.2 Safety instructions ........................................................................................................ 6  
2 Introduction........................................................................................................................ 7  
3 Function Blocks .................................................................................................................. 8  
   3.1 [obsolete] ...................................................................................................................... 8  
      3.1.1 FB_CxGetDeviceIdentification ............................................................................. 8  
      3.1.2 FB_CxGetDeviceIdentificationEx ....................................................................... 9  
      3.1.3 FB_CX1010SetWatchdog ..................................................................................... 9  
      3.1.4 FB_CX1020SetWatchdog ................................................................................... 10  
      3.1.5 FB_CX1030SetWatchdog ................................................................................... 11  
   3.2 FB_CXProfiler .............................................................................................................. 12  
   3.3 FB_CXsetTextDisplay ................................................................................................. 13  
   3.4 FB_CXsetTextDisplayUSB .......................................................................................... 14  
   3.5 FB_CXGetTextDisplayUSB ......................................................................................... 16  
   3.6 FB_CXSimpleUps .......................................................................................................... 18  
   3.7 FB_CX5010SetWatchdog ............................................................................................ 20  
   3.8 FB_CX5020SetWatchdog ............................................................................................ 20  
4 Functions ............................................................................................................................ 22  
   4.1 [obsolete] ...................................................................................................................... 22  
      4.1.1 F_GetVersionTcCXSystem ............................................................................... 22  
      4.1.2 F_GetVersionTcSystemCX1000 ......................................................................... 22  
      4.1.3 F_GetVersionTcSystemCX1010 ......................................................................... 22  
      4.1.4 F_GetVersionTcSystemCX1020 ......................................................................... 23  
      4.1.5 F_GetVersionTcSystemCX1030 ......................................................................... 23  
      4.1.6 F_GetVersionTcSystemCX5010 ......................................................................... 23  
      4.1.7 F_GetVersionTcSystemCX5020 ......................................................................... 24  
      4.1.8 F_GetVersionTcSystemCX9000 ......................................................................... 24  
      4.1.9 F_GetVersionTcSystemCX9010 ......................................................................... 24  
      4.1.10 F_CXSubTimeStamp ......................................................................................... 25  
      4.1.11 F_CX1000SetWatchdog .................................................................................... 26  
      4.1.12 F_CX9000SetWatchdog .................................................................................... 26  
      4.1.13 F_CX9010SetWatchdog .................................................................................... 27  
   4.2 F_CXNaviSwitch ............................................................................................................ 27  
   4.3 F_CXNaviSwitchUSB ..................................................................................................... 28  
   4.4 F_CX81xx_ADDRESS ..................................................................................................... 29  
   4.5 F_CX8180_LED_ERR .................................................................................................. 30  
   4.6 F_CX8180_LED_WD ................................................................................................. 30  
   4.7 F_CX8190_LED_ERR ................................................................................................. 31  
   4.8 F_CX8190_LED_WD ................................................................................................. 31  
5 [obsolete]............................................................................................................................ 33  
   5.1 FB_CxGetDeviceIdentification .................................................................................. 33  
   5.2 FB_CxGetDeviceIdentificationEx .............................................................................. 33
5.3 FB_CX1010SetWatchdog ................................................................. 34
5.4 FB_CX1020SetWatchdog ................................................................. 35
5.5 FB_CX1030SetWatchdog ................................................................. 36

6 [obsolete]............................................................................................................... 37
6.1 F_GetVersionTcCXSystem ........................................................................... 37
6.2 F_GetVersionTcSystemCX1000 ..................................................................... 37
6.3 F_GetVersionTcSystemCX1010 ..................................................................... 37
6.4 F_GetVersionTcSystemCX1020 ..................................................................... 38
6.5 F_GetVersionTcSystemCX1030 ..................................................................... 38
6.6 F_GetVersionTcSystemCX5010 ..................................................................... 38
6.7 F_GetVersionTcSystemCX5020 ..................................................................... 39
6.8 F_GetVersionTcSystemCX9000 ..................................................................... 39
6.9 F_GetVersionTcSystemCX9010 ..................................................................... 39
6.10 F_CXSubTimeStamp ..................................................................................... 40
6.11 F_CX1000SetWatchdog ................................................................................ 40
6.12 F_CX9000SetWatchdog ................................................................................ 41
6.13 F_CX9010SetWatchdog ................................................................................ 42

7 Data Types ........................................................................................................ 43
7.1 [obsolete] ................................................................................................. 43
7.1.1 ST_CX_DeviceIdentification .................................................................... 43
7.1.2 ST_CxDeviceIdentificationEx ................................................................. 44
7.2 ST_CX_ProfilerStruct .................................................................................. 44

8 Global constants............................................................................................. 46
8.1 Library version ............................................................................................ 46
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:


with corresponding applications or registrations in various other countries.

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.

The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited.

Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.
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.
# 2 Introduction

This library contains functions and function blocks that use features on the devices of the Embedded PC CX series.

## Function blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_CXProfiler [12]</td>
<td>Runtime measurement of PLC code via the CPU counter</td>
</tr>
<tr>
<td>FB_CXSetTextDisplay</td>
<td>Control of the two-line display of the CX1100</td>
</tr>
<tr>
<td>FB_CXSetTextDisplayUSB</td>
<td>Writing and deleting of rows on the two-line display of the CX 2100 or the EL6090 terminal.</td>
</tr>
<tr>
<td>FB_CXGetTextDisplayUSB</td>
<td>Reading of rows on the two-line display of the CX 2100 or the EL6090 terminal.</td>
</tr>
<tr>
<td>FB_CXSimpleUps [18]</td>
<td>Control of the UPS CX1190-UPS (device name CX1100-0900, CX1100-0910, CX1100-0920)</td>
</tr>
<tr>
<td>FB_CX5010SetWatchdog</td>
<td>Activates a hardware watchdog on the CX5010.</td>
</tr>
<tr>
<td>FB_CX5020SetWatchdog</td>
<td>Activates a hardware watchdog on the CX5020.</td>
</tr>
</tbody>
</table>

## Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_CXSubTimeStamp [25]</td>
<td>Calculates 64-bit subtraction (time A [100 ns] - time B [100 ns]) as result in μs; only for differences between 0 and 4294967295 us; see link.</td>
</tr>
<tr>
<td>F_CXNaviSwitch [27]</td>
<td>This function converts the value of the CX1100 navigation switch to an enum value.</td>
</tr>
<tr>
<td>F_CXNaviSwitchUSB [28]</td>
<td>This function converts the value of the CX2100 navigation switch to an enum value.</td>
</tr>
</tbody>
</table>

**Also see about this**
- F_CX1000SetWatchdog [26]
- F_CX9000SetWatchdog [26]
- F_CX9010SetWatchdog [27]
- FB_CxGetDeviceIdentificationEx [9]
- FB_CxGetDeviceIdentification [8]
- FB_CX1010SetWatchdog [9]
- FB_CX1020SetWatchdog [10]
- FB_CX1030SetWatchdog [11]
3 Function Blocks

3.1 [obsolete]

3.1.1 FB_CxGetDeviceIdentification

The function block FB_CxGetDeviceIdentification can be used to read device ID data of the CX.

**Obsolete functionality**

- Use the FB_GetDeviceIdentificationEx from the Tc2_Utility library.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bExecute : BOOL;</td>
</tr>
<tr>
<td>tTimeout : TIME;</td>
</tr>
</tbody>
</table>

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy : BOOL;</td>
</tr>
<tr>
<td>bError : BOOL;</td>
</tr>
<tr>
<td>nErrorID : UDINT;</td>
</tr>
<tr>
<td>stDevIdent : ST_CxDeviceIdentification;</td>
</tr>
</tbody>
</table>

**bBusy**: Data are read from the CX-Device. Data are stored in stDevIdent if bBusy = FALSE and bError = FALSE.

**bError**: gets TRUE, with any error.

**nErrorID**: contains the error id if bErr is TRUE.

**stDevIdent**: Contains the read data (type: ST_CxDeviceIdentification [43])

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1</td>
<td>CX (WES7/Win7/Win10: TC RT x86/x64, WEC6/7 :TC TR x86, WEC7: TC CE7 ARMV7)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>
3.1.2 FB_CxGetDeviceIdentificationEx

The function block FB_CxGetDeviceIdentificationEx can be used to read device ID data of the CX. The function block is an extension of the function block FB_CxGetDeviceIdentification. The read device data are stored in the variable stDevIdent of type ST_CxDeviceIdentificationEx.

Obsolete functionality

- Use the FB_GetDeviceIdentificationEx from the Tc2_Utilities library.

VAR_INPUT

bExecute : BOOL;
tTimeout : TIME;

VAR_OUTPUT

bBusy : BOOL;
bError : BOOL;
nErrorID : UDINT;
stDevIdent : ST_CxDeviceIdentificationEx;

bBusy: The data are read from the CX. After error-free execution, the data are in the structure stDevIdent if bBusy = FALSE.

bError: Becomes TRUE, as soon as an error occurs.

nErrorID: Supplies the error number when the bError output is set.

stDevIdent: Contains the read device data (type: ST_CxDeviceIdentificationEx)

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1</td>
<td>CX (WES7/Win7/Win10: TC RT x86/x64, WECE6/7 :TC TR x86, WECE7: TC CE7 ARMV7)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

3.1.3 FB_CX1010SetWatchdog

The function block FB_CX1010SetWatchdog activates a hardware watchdog on the CX1010. The watchdog is activated via bEnable = TRUE and the tTimeOut time. The tTimeOut time can be a minimum of 2 seconds and a maximum of 255 seconds.
Once the watchdog has been activated, the function block instance must be called cyclically at shorter intervals than tTimeOut, since the CX1010 restarts automatically when tTimeOut has elapsed. The watchdog can therefore be used to automatically reboot systems, which have entered an infinite loop or where the PLC has become stuck.

The watchdog can be deactivated via bEnable = FALSE or tTimeOut = T#0s.

### NOTE
The watchdog must be deactivated before breakpoints are used, before a PLC reset or an overall reset, before a TwinCAT stop, before switching to Config mode or before the configuration is activated, because otherwise the CX1010 would reboot immediately once tTimeOut has elapsed.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>tTimeout</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEnable</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

**tTimeout:** Watchdog time, if expired a reboot is automatically executed.

**bEnable:** Activate or deactivate the watchdog.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>bEnabled</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

**bEnabled:** TRUE = Watchdog is activated, FALSE = Watchdog is not activated.

**bError:** Error when activate or deactivate the watchdog.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (x86)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

### 3.1.4 FB_CX1020SetWatchdog

The function block FB_CX1020SetWatchdog activates a hardware watchdog on the CX1020. The watchdog is activated via bEnable = TRUE and the tTimeOut time. The tTimeOut time can be a minimum of 2 seconds and a maximum of 255 seconds.

Once the watchdog has been activated, the function block instance must be called cyclically at shorter intervals than tTimeOut, since the CX1020 restarts automatically when tTimeOut has elapsed. The watchdog can therefore be used to automatically reboot systems, which have entered an infinite loop or where the PLC has become stuck.

The watchdog can be deactivated via bEnable = FALSE or tTimeOut = T#0s.

### NOTE
The watchdog must be deactivated before breakpoints are used, before a PLC reset or an overall reset, before a TwinCAT stop, before switching to Config mode or before the configuration is activated, because otherwise the CX1020 would reboot immediately once tTimeOut has elapsed.
**Function Blocks**

**VAR_INPUT**

```plaintext
VAR_INPUT
  tTimeout : TIME;
  bEnable : BOOL;
END_VAR
```

tTimeout: Watchdog time, if expired a reboot is automatically executed.

bEnable: Activate or deactivate the watchdog.

**VAR_OUTPUT**

```plaintext
VAR_OUTPUT
  bEnabled : BOOL;
  bError : BOOL;
END_VAR
```

bEnabled: TRUE = Watchdog is activated, FALSE = Watchdog is not activated.

bError: Error when activate or deactivate the watchdog.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (x86)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

**3.1.5 FB_CX1030SetWatchdog**

The function block FB_CX1030SetWatchdog activates a hardware watchdog on the CX1030. The watchdog is activated via bEnable = TRUE and the tTimeout time. The tTimeout time can be a minimum of 2 seconds and a maximum of 255 seconds.

Once the watchdog has been activated, the function block instance must be called cyclically at shorter intervals than tTimeout, since the CX1030 restarts automatically when tTimeout has elapsed. The watchdog can therefore be used to automatically reboot systems, which have entered an infinite loop or where the PLC has become stuck.

The watchdog can be deactivated via bEnable = FALSE or tTimeout = T#0s.

**NOTE**

The watchdog must be deactivated before breakpoints are used, before a PLC reset or an overall reset, before a TwinCAT stop, before switching to Config mode or before the configuration is activated, because otherwise the CX1030 would reboot immediately once tTimeout has elapsed.

**VAR_INPUT**

```plaintext
VAR_INPUT
  tTimeout : TIME;
  bEnable : BOOL;
END_VAR
```

tTimeout: Watchdog time, if expired a reboot is automatically executed.

bEnable: Activate or deactivate the watchdog.

**VAR_OUTPUT**

```plaintext
VAR_OUTPUT
  bEnabled : BOOL;
  bError : BOOL;
END_VAR
```
**bEnabled**: TRUE = Watchdog is activated, FALSE = Watchdog is not activated.

**bError**: Error when activate or deactivate the watchdog.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (x86)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

#### 3.2 FB_CXProfiler

The function block FB_CXProfiler can be used to measure the execution time of the PLC code under Microsoft Windows CE.

For other operating systems, refer to the Profiler function block in the Tc2_Utils library.

Internally, an instance of the GETCPUCOUNTER function block is called. The measurement is started by a rising edge at the bStart input, and is stopped by a falling edge. The measurements are evaluated internally, and are then made available for further processing at the stData output in a structure of type ST_CX_ProfilerStruct. As well as the current, minimum and maximum execution times, the function block calculates the mean execution time for the last 100 measurements. The times measured are given in microseconds. The output variable stData.dwMeasureCycle provides information about the number of measurements that have already been carried out. In order to measure the execution time for a specific segment of the PLC program the measurement must be started by a rising edge at the START input when the segment to be measured starts, and stopped by a falling edge at the START input at the end of the segment. All values at the DATA output can be reset if a rising edge is generated at the RESET input at the same time as the rising edge at START. The measurements in the DATA structure that have already been determined then become invalid, and are re-calculated when the function block is called again.

**Comment:**

The determined times can deviate from the actual values, since already for the calls of the GETCPUCOUNTER function block some time is needed. This time depends on the particular computer, and is included in the determined times. Task interruptions, e.g. by the NC, are not detected and lead to longer measuring times.

**VAR_INPUT**

```plaintext
VAR_INPUT
   bStart : BOOL;
   bReset : BOOL;
END_VAR
```

**bStart**: A rising edge at this input starts the measurement of the execution time. A falling edge at this input stops the measurement, and causes the current, minimum, maximum and mean execution times to be recalculated. The variable stData.dwMeasureCycle is incremented at the same time.

**bReset**: A rising edge at this input will reset the variables at the stData output. A rising edge at this input and simultaneous rising edge at START input will reset the variables at the DATA output.

**VAR_OUTPUT**

```plaintext
VAR_OUTPUT
   bBusy : BOOL;
   stData : ST_CX_ProfilerStruct;
END_VAR
```
bBusy: This output is set at the start of the measuring procedure, and remains set until the time measurement has been completed. Once the bBUSY output has been reset, the latest times are available at the stData output.

stData: structure of type ST_CX_ProfilerStruct [44] with the measured times [in µs].

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1</td>
<td>CX (WEC6/7: TC RT x86, WEC7: TC CE7 ARMV7)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

3.3 FB_CXSetTextDisplay

The functionblock FB_CXSetTextDisplay can be used to send text messages to the two line display of the CX1100.

VAR_INPUT

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bExecute  : BOOL;</td>
</tr>
<tr>
<td>nDevID    : UDINT;</td>
</tr>
<tr>
<td>nMode     : E_CX1100_DisplayModes;</td>
</tr>
<tr>
<td>stLine    : STRING(20);</td>
</tr>
<tr>
<td>nCursorPos: DWORD;</td>
</tr>
</tbody>
</table>

bExecute: Command is executed with rising edge.

nDevID: Device ID of the CX1100-Device.

nMode: Modeswitch (see Enumeration)

stLine: String with 20 characters. This String is displayed with the appropriate command.

nCursorPos: Cursor position. The string is being written beginning from this position.

VAR_OUTPUT

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy      : BOOL;</td>
</tr>
<tr>
<td>bErr       : BOOL;</td>
</tr>
<tr>
<td>nErrorID   : UDINT;</td>
</tr>
</tbody>
</table>

bBusy: Command is being transported via ADS. bBusy remains TRUE, while no new command is accepted.

bErr: gets TRUE, with any error.

nErrorID: contains the error id if bErr is TRUE.

E_CX1100_DisplayModes:

E_CX1100_DisplayModes : (  
e_CX1100_DisplayNoAction := 0,  
e_CX1100_DisplayOn := 1,  
e_CX1100_DisplayOff,  
e_CX1100_CursorOn,  
e_CX1100_CursorOff,  
e_CX1100_CursorBlinkOn,  
e_CX1100_CursorBlinkOff,  
)
Function Blocks

```plaintext
e_CX1100_BackLightOn,
e_CX1100_BackLightOff,
e_CX1100_ClearDisplay,
e_CX1100_WriteLine1,
e_CX1100_WriteLine2
```

e_CX1000_DisplayNoAction : No Action.
e_CX1000_DisplayOn : Switches display on.
e_CX1000_DisplayOff : Switches display off.
e_CX1000_CursorOn : Switches cursor on.
e_CX1000_CursorOff : Switches cursor off.
e_CX1000_CursorBlinkOn : Switches blinking of cursor on.
e_CX1000_CursorBlinkOff : Switches blinking of cursor off.
e_CX1000_BackLightOn : Switches background light on.
e_CX1000_BackLightOff : Switches background light off.
e_CX1000_ClearDisplay : Clears display.
e_CX1000_WriteLine1 : Write line 1.
e_CX1000_WriteLine2 : Write line 2.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (x86)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

3.4 FB_CXSetTextDisplayUSB

The function block is used for the CX2100 and for the EL6090 terminal. When the function block is called, only the matching NetID and port number for the respective device has to be allocated.

<table>
<thead>
<tr>
<th>nPort</th>
<th>sNetID</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX2100</td>
<td>Is displayed in TwinCAT on the ESB device tab.</td>
</tr>
<tr>
<td>EL6090</td>
<td>Is the EtherCAT address of the terminal</td>
</tr>
</tbody>
</table>

The function block FB_CXSetTextDisplayUSB is used to write and delete messages on the two-line display. The cursor is controlled on the display by switching it on and off or by making it flash. The function block is also used to switch the backlight on or off.

VAR_INPUT

```plaintext
VAR_INPUT
   bExecute : BOOL;
   sNetID : T_AmsNetID;
```
nPort : T_AmsPort;
eMode : E_CX2100_DisplayModesWr;
sLine1 : STRING(80);
sLine2 : STRING(80);
nCursorPosX : USINT;
nCursorPosY : USINT;
END_VAR

bExecute: The command is executed with the rising edge.

sNetID: AMSNetID of the device.
nPort: AmsPort of the device.
eMode: Mode switching.

sLine1: String with 80 characters. This string is displayed with the corresponding command in the first line. For strings with more than 16 characters, the text is displayed as scrolling text.

sLine2: String with 80 characters. This string is displayed with the corresponding command in the second line. For strings with more than 16 characters, the text is displayed as scrolling text.

nCursorPosX: Cursor position on the X axis. The string is written from this position in the display.
nCursorPosY: Cursor position on the Y axis. The string is written from this position in the display.

VAR_OUTPUT
VAR_OUTPUT
  bBusy : BOOL;
bError : BOOL;
nErrorID : UDINT;
END_VAR

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as “bBusy” remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.
nErrorID: Supplies the error number when the bError output is set.

E_CX2100_DisplayModesWr:

E_CX2100_DisplayModesWr : {
  eCX2100_DisplayNoActionWr := 0,
  eCX2100_CursorOn,
  eCX2100_CursorOff,
  eCX2100_CursorBlinkOn,
  eCX2100_CursorBlinkOff,
  eCX2100_BackLightOn,
  eCX2100_BackLightOff,
  eCX2100_ClearDisplay,
  eCX2100_WriteLine1,
  eCX2100_WriteLine2,
  eCX2100_WriteLines,
  eCX2100_CursorPosX,
  eCX2100_CursorPosY,
  eCX2100_CursorPosXY
};

eCX2100_DisplayNoActionWr: No action.
eCX2100_CursorOn: Switch on the cursor.
eCX2100_CursorOff: Switch off the cursor.
eCX2100_CursorBlinkOn: Switching on the cursor blinking.
eCX2100_CursorBlinkOff: Switch off the cursor blinking.
eCX2100_BackLightOn: Switch on the backlight.
eCX2100_BackLightOff: Switch off the backlight.
eCX2100_ClearDisplay: Clear display.
Function Blocks

**eCX2100_WriteLine1**: Write the first line.

**eCX2100_WriteLine2**: Write the second line.

**eCX2100_WriteLines**: Write lines.

**eCX2100_CursorPosX**: Cursor position on the X axis.

**eCX2100_CursorPosY**: Cursor position on the Y axis.

**eCX2100_CursorPosXY**: Cursor position on the XY axis.

## Requirements when using the EL6090

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1</td>
<td>PC or CX (WES7/Win7/Win10: TC RT x86/x64, WE C6/7: TC RT x86, WEC7: TC CE7 ARMV7, TC/BSD: TC RT x84, TC OS ARMT2)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

## Requirements when using the CX2100

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1</td>
<td>CX (WES7/Win7/Win10: TC RT x86/x64, WEC6/7: TC RT x86, TC/ BSD: TC RT x64)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

### 3.5 FB_CXGetTextDisplayUSB

The function block is used for the CX2100 and for the EL6090 terminal. When the function block is called, only the matching NetID and port number for the respective device has to be allocated.

<table>
<thead>
<tr>
<th>nPort:</th>
<th>sNetID</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX2100</td>
<td>Is displayed in TwinCAT on the ESB device tab.</td>
</tr>
<tr>
<td>EL6090</td>
<td>Is the EtherCAT address of the terminal.</td>
</tr>
</tbody>
</table>

The function block FB_CXGetTextDisplayUSB is used to read the lines on the display. In addition, the cursor status is read, i.e. cursor switched on or off, or is flashing. The function block also indicates whether the backlight is switched on or off.

#### VAR_INPUT

```
VAR_INPUT
  bExecute : BOOL;
  sNetID : T_AmsNetID;
  nPort : T_AmsPort;
  eMode : E_CX2100_DisplayModesRd;
END_VAR
```

**bExecute**: The command is executed with a rising edge.
sNetID: AMSNetID of the device.
nPort: AMSPort of the device.
eMode: Mode switching.

VAR_OUTPUT

VAR_OUTPUT
  bBusy      : BOOL;
  bError     : BOOL;
  nErrorID   : UDINT;
  sLine1     : STRING(80);
  sLine2     : STRING(80);
  nCursorPosX : USINT;
  nCursorPosY : USINT;
  nCursorMode : USINT;
  nBacklight : USINT;
END_VAR

bBusy: Command is being transmitted via ADS. No new command will be accepted as long as "bBusy" remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.
nErrorID: Returns the error number when the bError output is set.
sLine1: This string is read with the corresponding command.
sLine2: This string is read with the corresponding command.
nCursorPosX: Cursor position on the X axis
nnCursorPosY: Cursor position on the Y axis
nnCursorMode: Mode of the cursor
nBacklight: Backlight

E_CX2100_DisplayModesRd:

E_CX2100_DisplayModesRd : (  
  eCX2100_DisplayNoActionRd := 0,  
  eCX2100_ReadCursorInfo,  
  eCX2100_ReadBackLight,  
  eCX2100_ReadLine1,  
  eCX2100_ReadLine2,  
  eCX2100_ReadLines  
);

eCX2100_DisplayNoActionRd: No action.
eCX2100_ReadCursorInfo: Read values via the cursor.
eCX2100_ReadBackLight: Read backlight values.
eCX2100_ReadLine1: Read values from the first line.
eCX2100_ReadLine2: Read values from the second line.
eCX2100_ReadLines: Read values from lines.

Requirements when using the CX2100

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1</td>
<td>CX (WES7/Win7/Win10: TC RT x86/x64, WEC6/7: TC RT x86, TC/ BSD: TC RT x64)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>
Requirements when using the EL6090

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1</td>
<td>PC or CX (WES7/Win7/Win10: TC RT x86/x64, WEC6/7: TC RT x86, WEC7: TC CE7 ARMV7, TC/BSD: TC RT x64, TC OS ARMT2)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

### 3.6 FB_CXSimpleUps

The function block FB_CXSimpleUps can be used on the CX1000 or CX1020 in order to activate the UPS CX1190-UPS from the PLC. In this case the UPS settings must be deactivated in the TwinCAT System Manager.

```plaintext
VAR_INPUT
bDIPDisable : BOOL;
iDischargeLevel : USINT;
tDelay : TIME;
END_VAR
```
**Function Blocks**

**bDIPDisable** : if TRUE, then the orientation of the '1-2-3..8-9-0' switch on the UPS is ignored, in this case the iDischargeLevel is used instead.

**iDischargeLevel** : Discharge Switch Off Level: 0 = 100% (Maximum Discharge), 9 = 90%, 8 = 80%, ..., 2 = 20%, 1 = 10% (Minimum Discharge).

**tDelay** : Holding time, time before the Shutdown is executed. This time is used to overcome short power outages (up to 10s) without shutdown.
Once the holding time is elapsed, the FB finishes the holding period and waits internally for additional 2.5s.
If the power has returned by then, the FB continues with normal run mode, otherwise the powerfail shutdown is executed.
If the power returns during or after the shutdown, then the CX reboots after discharging and recharging the UPS.

**VAR_OUTPUT**

```
VAR_OUTPUT
bPowerFailure : BOOL;
bShutdownActive : BOOL;
bUpsReady : BOOL;
b24VinOK : BOOL;
bHolding : BOOL;
tTimeUntilShutdown : TIME;
eUpsState : E_UPS_STATE;
END_VAR
```

**bPowerFailure** : gets TRUE, if a power failure of the power supply is detected, gets FALSE, if the power supply is restored.

**bShutdownActive** : gets TRUE, if a Stop or Shutdown is being executed.

**bUpsReady** : gets TRUE, if the UPS supplies the voltage.

**b24VinOK** : gets TRUE, if the power supply supplies the UPS with 24V.

**bHolding** : gets TRUE, if a power failure of the power supply is detected, and the holding time has not yet elapsed.

**tTimeUntilShutdown** : Shows the time until the system shuts down after a power fault.

**eUpsState** : Shows the status of the UPS [UNDEF | CHARGING | CHARGED | DISCHARGE | DISCHARGE_RESTART | OUTPUT_OFF | OVERLOAD].

**VAR_CONFIG**

```
VAR_CONFIG
ii24VState AT %I* : BYTE;
iiChargeState AT %I* : USINT;
qiControl AT %Q* : BYTE;
qiDIPControl AT %Q* : USINT;
END_VAR
```

**ii24VState** : needs to be linked to input '24V State', see picture above.

**iiChargeState** : needs to be linked to input 'Charge State', see picture above.

**qiControl** : needs to be linked to output 'Control', see picture above.

**qiDIPControl** : needs to be linked to output 'DIP Ctrl', see picture above.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (x86)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>
For other CX and PC please refer to the function block FB_S_SUPS_BAPI in the Tc2_SUPS library.

### 3.7 FB_CX5010SetWatchdog

The function block FB_CX5010SetWatchdog activates a hardware watchdog on the CX5010. The watchdog is activated via bEnable = TRUE and the tTimeOut time. The tTimeOut time can be a minimum of 2 seconds and a maximum of 255 seconds.

Once the watchdog has been activated, the function block instance must be called cyclically at shorter intervals than tTimeOut, since the CX5010 restarts automatically when tTimeOut has elapsed. The watchdog can therefore be used to automatically reboot systems, which have entered an infinite loop or where the PLC has become stuck.

The watchdog can be deactivated via bEnable = FALSE or tTimeOut = T#0s.

**NOTE**

The watchdog must be deactivated before breakpoints are used, before a PLC reset or an overall reset, before a TwinCAT stop, before switching to Config mode or before the configuration is activated, because otherwise the CX5010 would reboot immediately once tTimeOut has elapsed.

**VAR_INPUT**

```plaintext
VAR_INPUT
  tTimeout : TIME;
  bEnable  : BOOL;
END_VAR
```

**tTimeout**: Watchdog time, if expired a reboot is automatically executed.

**bEnable**: Activate or deactivate the watchdog.

**VAR_OUTPUT**

```plaintext
VAR_OUTPUT
  bEnabled  : BOOL;
  bError    : BOOL;
END_VAR
```

**bEnabled**: TRUE = Watchdog is activated, FALSE = Watchdog is not activated.

**bError**: Error when activate or deactivate the watchdog.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (x86)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

### 3.8 FB_CX5020SetWatchdog

The function block FB_CX5020SetWatchdog activates a hardware watchdog on the CX5020. The watchdog is activated via bEnable = TRUE and the tTimeOut time. The tTimeOut time can be a minimum of 2 seconds and a maximum of 255 seconds.
Once the watchdog has been activated, the function block instance must be called cyclically at shorter intervals than tTimeOut, since the CX5020 restarts automatically when tTimeOut has elapsed. The watchdog can therefore be used to automatically reboot systems, which have entered an infinite loop or where the PLC has become stuck.

The watchdog can be deactivated via bEnable = FALSE or tTimeOut = T#0s.

**NOTE**
The watchdog must be deactivated before breakpoints are used, before a PLC reset or an overall reset, before a TwinCAT stop, before switching to Config mode or before the configuration is activated, because otherwise the CX5020 would reboot immediately once tTimeOut has elapsed.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>tTimeout</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEnable</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>bEnabled</th>
<th>BOOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

**tTimeOut**: Watchdog time, if expired a reboot is automatically executed.

**bEnable**: Activate or deactivate the watchdog.

**bEnabled**: TRUE = Watchdog is activated, FALSE = Watchdog is not activated.

**bError**: Error when activate or deactivate the watchdog.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (x86)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>
4 Functions

4.1 [obsolete]

4.1.1 F_GetVersionTcCXSystem

```
FUNCTION F_GetVersionTcSystemCX : UINT
VAR_INPUT
nVersionElement : INT;
END_VAR
```

The function returns library version info.

```
nVersionElement : Version element:
   • 1 : major number;
   • 2 : minor number;
   • 3 : revision number;
```

4.1.2 F_GetVersionTcSystemCX1000

```
FUNCTION F_GetVersionTcSystemCX1000 : UINT
VAR_INPUT
nVersionElement : INT;
END_VAR
```

The function returns library version info.

```
nVersionElement : Version element:
   • 1 : major number;
   • 2 : minor number;
   • 3 : revision number;
```

4.1.3 F_GetVersionTcSystemCX1010

```
FUNCTION F_GetVersionTcSystemCX1010 : UINT
VAR_INPUT
nVersionElement : INT;
END_VAR
```

The function returns library version info.
FUNCTION F_GetVersionTcSystemCX1010 : UINT
VAR_INPUT
nVersionElement : INT;
END_VAR

nVersionElement : Version element:
- 1 : major number;
- 2 : minor number;
- 3 : revision number;

4.1.4 F_GetVersionTcSystemCX1020

The function returns library version info.

FUNCTION F_GetVersionTcSystemCX1020 : UINT
VAR_INPUT
nVersionElement : INT;
END_VAR

nVersionElement : Version element:
- 1 : major number;
- 2 : minor number;
- 3 : revision number;

4.1.5 F_GetVersionTcSystemCX1030

The function returns library version info.

FUNCTION F_GetVersionTcSystemCX1030 : UINT
VAR_INPUT
nVersionElement : INT;
END_VAR

nVersionElement : Version element:
- 1 : major number;
- 2 : minor number;
- 3 : revision number;

4.1.6 F_GetVersionTcSystemCX5010

The function returns library version info.
FUNCTION F_GetVersionTcSystemCX5010 : UINT

VAR_INPUT
nVersionElement : INT;
END_VAR

nVersionElement : Version element:
  • 1 : major number;
  • 2 : minor number;
  • 3 : revision number;

4.1.7 F_GetVersionTcSystemCX5020

The function returns library version info.

FUNCTION F_GetVersionTcSystemCX5020 : UINT

VAR_INPUT
nVersionElement : INT;
END_VAR

nVersionElement : Version element:
  • 1 : major number;
  • 2 : minor number;
  • 3 : revision number;

4.1.8 F_GetVersionTcSystemCX9000

The function returns library version info.

FUNCTION F_GetVersionTcSystemCX9000 : UINT

VAR_INPUT
nVersionElement : INT;
END_VAR

nVersionElement : Version element:
  • 1 : major number;
  • 2 : minor number;
  • 3 : revision number;

4.1.9 F_GetVersionTcSystemCX9010
The function returns library version info.

FUNCTION F_GetVersionTcSystemCX9010 : UINT
VAR_INPUT
nVersionElement : INT;
END_VAR

nVersionElement : Version element:
• 1 : major number;
• 2 : minor number;
• 3 : revision number;

4.1.10  F_CXSubTimeStamp

For subtraction TwinCAT3.1 also offers 64-bit data types (LINT/ULINT, LWORD) that can be executed directly (A-B) or there are TC2-compatible functions for 64-bit operations that should be used as an alternative to F_CXSubTimeStamp.

The function F_CXSubTimeStamp executes a 64-bit subtraction time stamp A - time stamp B and converts the result to µs. The required 64-bit time stamps with 100 ns resolution can be read from the system with the function block GETCPUCOUNTER.

If the difference between time stamp A and time stamp B is negative or greater than 4294967295 us, the maximum value 4294967295 us is returned. This corresponds to 71 minutes, 34 seconds, 967 milliseconds and 295 microseconds. In such cases the function UInt64Sub64() from TcUtilities.lib can be used to execute a complete 64-bit subtraction with 64-bit result in [100 ns].

FUNCTION F_CXSubTimeStamp: UDINT
VAR_INPUT
 nTimeStampLoDW_A : UDINT; (* 2*32 bit time stamp A: low DWORD *)
 nTimeStampHiDW_A : UDINT; (* 2*32 bit time stamp A: high DWORD *)
 nTimeStampLoDW_B : UDINT; (* 2*32 bit time stamp B: low DWORD *)
 nTimeStampHiDW_B : UDINT; (* 2*32 bit time stamp B: high DWORD *)
END_VAR

nTimeStampLoDW_A: lower 32bit of time stamp A.
nTimeStampHiDW_A: upper 32bit of time stamp A.
nTimeStampLoDW_B: lower 32bit of time stamp B.
nTimeStampHiDW_B: upper 32bit of time stamp B.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1</td>
<td>CX (WES7/Win7/Win10: TC RT x86/x64, WEC6/7: TC TR x86, WEC7: TC CE7 ARMv7)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>
4.1.11  F_CX1000SetWatchdog

The function F_CX1000SetWatchdog activates a hardware watchdog on the CX1000. The watchdog is activated via bEnable = TRUE and the tTimeout time. The tTimeout time can be a minimum of several PLC task cycles (multiple of the call time of the function F_CX1000SetWatchdog) and a maximum of 65 s and 535 ms.

Once the watchdog has been activated, the function must be called cyclically at shorter intervals than tTimeout, since the CX1000 restarts automatically when tTimeout has elapsed. The watchdog can therefore be used to automatically reboot systems, which have entered an infinite loop or where the PLC has become stuck.

The watchdog can be deactivated via bEnable = FALSE or tTimeout = 0.

NOTE
The watchdog must be deactivated before breakpoints are used, before a PLC reset or an overall reset, before a TwinCAT stop, before switching to Config mode or before the configuration is activated, because otherwise the CX1000 would reboot immediately once the timeout has elapsed.

FUNCTION F_CX1000SetWatchdog: BOOL
VAR_INPUT
  tTimeout : TIME;
  bEnable : BOOL;
END_VAR

tTimeout: Watchdog time, if expired a reboot is automatically executed.

bEnable: Activate or deactivate the watchdog.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (x86)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

4.1.12  F_CX9000SetWatchdog

The function F_CX9000SetWatchdog activates a hardware watchdog on the CX9000. The watchdog is activated via bEnable = TRUE and the tTimeout time. The tTimeout time can be a maximum of 65 seconds and 535 milliseconds.

Once the watchdog has been activated, the function must be called cyclically at shorter intervals than tTimeout, since the CX9000 restarts automatically when tTimeout has elapsed. The watchdog can therefore be used to automatically reboot systems, which have entered an infinite loop or where the PLC has become stuck.

The watchdog can be deactivated via bEnable = FALSE or tTimeout = T#0s.

NOTE
The watchdog must be deactivated before breakpoints are used, before a PLC reset or an overall reset, before a TwinCAT stop, before switching to Config mode or before the configuration is activated, because otherwise the CX9000 would reboot immediately once tTimeout has elapsed.
FUNCTION F_CX9000SetWatchdog: BOOL

VAR_INPUT
  tTimeout : TIME;
  bEnable : BOOL;
END_VAR

**tTimeout**: Watchdog time, if expired a reboot is automatically executed.

**bEnable**: Activate or deactivate the watchdog.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (ARM)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

#### 4.1.13 F_CX9010SetWatchdog

The function `F_CX9010SetWatchdog` activates a hardware watchdog on the CX9000. The watchdog is activated via `bEnable = TRUE` and the `tTimeout` time. The `tTimeout` time can be a maximum of 65 seconds and 535 milliseconds.

Once the watchdog has been activated, the function must be called cyclically at shorter intervals than `tTimeout`, since the CX9010 restarts automatically when `tTimeout` has elapsed. The watchdog can therefore be used to automatically reboot systems, which have entered an infinite loop or where the PLC has become stuck.

The watchdog can be deactivated via `bEnable = FALSE` or `tTimeout = T#0s`.

**NOTE**

The watchdog must be deactivated before breakpoints are used, before a PLC reset or an overall reset, before a TwinCAT stop, before switching to Config mode or before the configuration is activated, because otherwise the CX9010 would reboot immediately once `tTimeout` has elapsed.

FUNCTION F_CX9010SetWatchdog: BOOL

VAR_INPUT
  tTimeout : TIME;
  bEnable : BOOL;
END_VAR

**tTimeout**: Watchdog time, if expired a reboot is automatically executed.

**bEnable**: Activate or deactivate the watchdog.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (ARM)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

#### 4.2 F_CXNaviSwitch

The function `F_CXNaviSwitch` converts the value of the CX1100 navigation switch into an enum value of type `E_CX1100_NaviSwitch`.  

---

**TE1000**

Version: 1.3
FUNCTION F_CXNaviSwitch: E_CX1100_NaviSwitch

VAR_INPUT
   icx1100_IN : USINT
END_VAR

E_CX1100_NaviSwitch: Value of the CX1100 input 'IN'

Enum E_CX1100_NaviSwitch

TYPE E_CX1100_NaviSwitch : (  
   e_CX1100_NaviSwitch_IDLE := 0,
   e_CX1100_NaviSwitch_MIDDLE := 16,
   e_CX1100_NaviSwitch_TOP := 1,
   e_CX1100_NaviSwitch_TOPRIGHT := 9,
   e_CX1100_NaviSwitch_RIGHT := 8,
   e_CX1100_NaviSwitch_BOTTOMRIGHT := 10,
   e_CX1100_NaviSwitch_BOTTOM := 2,
   e_CX1100_NaviSwitch_BOTTOMLEFT := 6,
   e_CX1100_NaviSwitch_LEFT := 4,
   e_CX1100_NaviSwitch_TOPLEFT := 5,
   e_CX1100_NaviSwitch_MIDDLE_TOP := 17,
   e_CX1100_NaviSwitch_MIDDLE_TOPRIGHT := 25,
   e_CX1100_NaviSwitch_MIDDLE_RIGHT := 24,
   e_CX1100_NaviSwitch_MIDDLE_BOTTOMRIGHT := 26,
   e_CX1100_NaviSwitch_MIDDLE_BOTTOM := 18,
   e_CX1100_NaviSwitch_MIDDLE_BOTTOMLEFT := 22,
   e_CX1100_NaviSwitch_MIDDLE_LEFT := 20,
   e_CX1100_NaviSwitch_MIDDLE_TOPLEFT := 21  
)  
END_VAR  

Values other than those defined in enum (e.g. 11) are displayed as "*** INVALID: value ***" in online mode (e.g. *** INVALID: 11 ***). The function F_CXNaviSwitch then returns the invalid value (e.g. 11).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (x86)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

4.3 F_CXNaviSwitchUSB

The function F_CXNaviSwitchUSB converts the value of the CX2100 navigation switch or an EL6090 into an enum value of type E_CX2100_NaviSwitch.

FUNCTION F_CXNaviSwitchUSB: E_CX2100_NaviSwitch

VAR_INPUT
   icx2100_IN : USINT;
END_VAR

icx2100_IN: Value of the CX2100 input 'IN'

Enum E_CX2100_NaviSwitch

TYPE E_CX2100_NaviSwitch : (  
   e_CX2100_NaviSwitch_IDLE := 0,
   e_CX2100_NaviSwitch_MIDDLE := 15,
   e_CX2100_NaviSwitch_ENTER := 16,
   e_CX2100_NaviSwitch_ENTER_MIDDLE := 31,
   e_CX2100_NaviSwitch_TOP := 1,
   e_CX2100_NaviSwitch_TOPRIGHT := 9,
e_CX2100_NaviSwitch_RIGHT := 8,
e_CX2100_NaviSwitch_BOTTOMRIGHT := 10,
e_CX2100_NaviSwitch_BOTTOM := 2,
e_CX2100_NaviSwitch_BOTTOMLEFT := 6,
e_CX2100_NaviSwitch_LEFT := 4,
e_CX2100_NaviSwitch_TOPLEFT := 5,

(* clockwise in 90 degree, 3 switches on*)
e_CX2100_NaviSwitch_TOPLEFTRIGHT := 13,
e_CX2100_NaviSwitch_RIGHTTOPBOTTOM := 11,
e_CX2100_NaviSwitch_BOTTOMLEFTRIGHT := 14,
e_CX2100_NaviSwitch_LEFTTOPBOTTOM := 7,

(* clockwise in 45 degree steps with enter switch pressed, 1-2 switches on *)
e_CX2100_NaviSwitch_ENTER_TOP := 17,
e_CX2100_NaviSwitch_ENTER_TOPRIGHT := 25,
e_CX2100_NaviSwitch_ENTER_RIGHT := 24,
e_CX2100_NaviSwitch_ENTER_BOTTOMRIGHT := 26,
e_CX2100_NaviSwitch_ENTER_BOTTOM := 18,
e_CX2100_NaviSwitch_ENTER_BOTTOMLEFT := 20,
e_CX2100_NaviSwitch_ENTER_TOPLEFT := 21

(* clockwise in 90 degree steps with enter switch pressed, 3 switches on *)
e_CX2100_NaviSwitch_ENTER_TOPLEFTRIGHT := 29,
e_CX2100_NaviSwitch_ENTER_RIGHTTOPBOTTOM := 27,
e_CX2100_NaviSwitch_ENTER_BOTTOMLEFTRIGHT := 30,
e_CX2100_NaviSwitch_ENTER_LEFTTOPBOTTOM := 23,

Requirements when using the CX2100

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1</td>
<td>CX (WES7/Win7/Win10: TC RT x86/x64, WEC6/7: TC RT x86, TC/BSD: TC RT x64)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

Requirements when using the EL6090

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1</td>
<td>PC or CX (WES7/Win7/Win10: TC RT x86/x64, WEC6/7: TC RT x86, WEC7: TC CE7 ARMV7, TC/BSD: TC RT x84, TC OS ARMT2)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

4.4 F_CX81xx_ADDRESS

This function reads the position of the address selection switch of the CXxxxx. One possible application is that you can activate different program parts in the PLC depending on the switch position.

This function reads the position of the DIP switch of the CXxxxx. One possible application is that you can activate different program parts in the PLC depending on the switch position.

VAR_INPUT

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>iCX_Typ</td>
<td>INT</td>
</tr>
</tbody>
</table>

(* Use product code without 'CX' e.g.: CX8180 -> 8180 *)

END_VAR

VAR_OUTPUT

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F_CX80xx_ADDRESS</td>
<td>INT</td>
</tr>
</tbody>
</table>

F_CX80xx_ADDRESS : -1, non-implemented CX, address of the switch
Functions

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>Hardware</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.30</td>
<td>ARM</td>
<td>CXxxxx</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

4.5  

F_CX8180_LED_ERR

Since the CX8180 supports various protocols, the two LEDs WD and ERR on the CX8180 are not used by the firmware. This allows the user to create his own diagnosis messages. The LEDs can be used to indicate, for example, whether the CX8180 has received an IP address from the DHCP server or whether devices are exchanging data.

The F_CX8180_LED_ERR function controls the ERR LED on the CX8180. Various color and flashing modes can be used here. The possible LED colors are red and green.

VAR_INPUT

VAR_INPUT
  eMode : E_CX8180_LED;
END_VAR

eMode: The way in which the LED lights up.

VAR_OUTPUT

F_F_CX8180_LED_ERR : INT;

F_CX8180_LED_ERR: -1, not implemented flash code, 0 OK

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>Hardware</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.30</td>
<td>ARM</td>
<td>CXxxxx</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

4.6  

F_CX8180_LED_WD

Since the CX8180 supports various protocols, the two LEDs WD and ERR on the CX8180 are not used by the firmware. This allows the user to create his own diagnosis messages. The LEDs can be used to indicate, for example, whether the CX8180 has received an IP address from the DHCP server or whether devices are exchanging data.

The F_CX8180_LED_WD function controls the WD LED on the CX8180. Various color and flashing modes can be used here. The possible LED colors are red and green.

VAR_INPUT

VAR_INPUT
  eMode : E_CX8180_LED;
END_VAR

eMode: The way in which the LED lights up.
VAR_OUTPUT
F_ CX8180_LED_WD : INT;
F_ CX8180_LED_WD: -1, not implemented flash code, 0 OK

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>Hardware</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.30</td>
<td>ARM</td>
<td>CXxxxx</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

4.7  F_CX8190_LED_ERR

Since the CX8190 supports various protocols, the two LEDs WD and ERR on the CX8190 are not used by the firmware. This allows the user to create his own diagnosis messages. The LEDs can be used to indicate, for example, whether the CX8190 has received an IP address from the DHCP server or whether devices are exchanging data.

The F_CX8190_LED_ERR function controls the ERR LED on the CX8190. Various color and flashing modes can be used here. The possible LED colors are red and green.

VAR_INPUT
VAR_INPUT
  eMode : E_CX8190_LED;
END_VAR

eMode: The way in which the LED lights up.

VAR_OUTPUT
VAR_OUTPUT
F_ F_CX8190_LED_ERR : INT;
F_CX8190_LED_ERR: -1, not implemented flash code, 0 OK

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>Hardware</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.30</td>
<td>ARM</td>
<td>CXxxxx</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

4.8  F_CX8190_LED_WD

Since the CX8190 supports various protocols, the two LEDs WD and ERR on the CX8190 are not used by the firmware. This allows the user to create his own diagnosis messages. The LEDs can be used to indicate, for example, whether the CX8190 has received an IP address from the DHCP server or whether devices are exchanging data.

The F_CX8190_LED_WD function controls the WD LED on the CX8190. Various color and flashing modes can be used here. The possible LED colors are red and green.
**VAR_INPUT**

```plaintext
VAR_INPUT
    eMode : E_CX8190_LED;
END_VAR
```

eMode: The way in which the LED lights up.

**VAR_OUTPUT**

```plaintext
F_ CX8190_LED_WD : INT;
```

F_ CX8190_LED_WD: -1, not implemented flash code, 0 OK

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>Hardware</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.30</td>
<td>ARM</td>
<td>CXxxxx</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>
5 [obsolete]

5.1 FB_CxGetDeviceIdentification

The function block FB_CxGetDeviceIdentification can be used to read device ID data of the CX.

Obsolete functionality

• Use the FB_GetDeviceIdentificationEx from the Tc2_Utils library.

VAR_INPUT

VAR_INPUT
  bExecute : BOOL;
  tTimeout : TIME;
END_VAR

bExecute: The command is executed with the rising edge.
tTimeout: States the time before the function is canceled.

VAR_OUTPUT

VAR_OUTPUT
  bBusy : BOOL;
  bError : BOOL;
  nErrorID : UDINT;
  stDevIdent : ST_CxDeviceIdentification;
END_VAR

bBusy: Data are read from the CX-Device. Data are stored in stDevIdent if bBusy = FALSE and bError = FALSE.
bError: gets TRUE, with any error.
nErrorID: contains the error id if bErr is TRUE.
stDevIdent: Contains the read data (type: ST_CxDeviceIdentification [43])

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1</td>
<td>CX (WES7/Win7/Win10: TC RT x86/x64, WEC6/7: TC TR x86, WEC7: TC CE7 ARMV7)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

5.2 FB_CxGetDeviceIdentificationEx
The function block FB_CxGetDeviceIdentificationEx can be used to read device ID data of the CX. The function block is an extension of the function block FB_CxGetDeviceIdentification. The read device data are stored in the variable stDevIdent of type ST_CxDeviceIdentificationEx [44].

- **Obsolete functionality**
  - Use the FB_GetDeviceIdentificationEx from the Tc2_Utilities library.

### VAR_INPUT

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bExecute : BOOL;</td>
</tr>
<tr>
<td>tTimeout : TIME;</td>
</tr>
</tbody>
</table>

- **bExecute**: The command is executed with a rising edge.
- **tTimeout**: States the time before the function is cancelled.

### VAR_OUTPUT

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy : BOOL;</td>
</tr>
<tr>
<td>bError : BOOL;</td>
</tr>
<tr>
<td>nErrorId : UDINT;</td>
</tr>
<tr>
<td>stDevIdent : ST_CxDeviceIdentificationEx;</td>
</tr>
</tbody>
</table>

- **bBusy**: The data are read from the CX. After error-free execution, the data are in the structure stDevIdent if bBusy = FALSE.
- **bError**: Becomes TRUE, as soon as an error occurs.
- **nErrorID**: Supplies the error number when the bError output is set.
- **stDevIdent**: Contains the read device data (type: ST_CxDeviceIdentificationEx [44])

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1</td>
<td>CX (WES7/Win7/Win10: TC RT x86/x64, WEC6/7:TC TR x86, WEC7: TC CE7 ARMV7)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

### 5.3 FB_CX1010SetWatchdog

The function block FB_CX1010SetWatchdog activates a hardware watchdog on the CX1010. The watchdog is activated via bEnable = TRUE and the tTimeOut time. The tTimeOut time can be a minimum of 2 seconds and a maximum of 255 seconds.

Once the watchdog has been activated, the function block instance must be called cyclically at shorter intervals than tTimeOut, since the CX1010 restarts automatically when tTimeOut has elapsed. The watchdog can therefore be used to automatically reboot systems, which have entered an infinite loop or where the PLC has become stuck.

The watchdog can be deactivated via bEnable = FALSE or tTimeOut = T#0s.
The watchdog must be deactivated before breakpoints are used, before a PLC reset or an overall reset, before a TwinCAT stop, before switching to Config mode or before the configuration is activated, because otherwise the CX1010 would reboot immediately once tTimeOut has elapsed.

**NOTE**

VAR_INPUT

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>tTimeout : TIME;</td>
</tr>
<tr>
<td>bEnable : BOOL;</td>
</tr>
<tr>
<td>END_VAR</td>
</tr>
</tbody>
</table>

**tTimeout**: Watchdog time, if expired a reboot is automatically executed.

**bEnable**: Activate or deactivate the watchdog.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bEnabled : BOOL;</td>
</tr>
<tr>
<td>bError : BOOL;</td>
</tr>
<tr>
<td>END_VAR</td>
</tr>
</tbody>
</table>

**bEnabled**: TRUE = Watchdog is activated, FALSE = Watchdog is not activated.

**bError**: Error when activate or deactivate the watchdog.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (x86)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

**5.4 FB_CX1020SetWatchdog**

![FB_CX1020SetWatchdog]

The function block FB_CX1020SetWatchdog activates a hardware watchdog on the CX1020. The watchdog is activated via bEnable = TRUE and the tTimeout time. The tTimeout time can be a minimum of 2 seconds and a maximum of 255 seconds.

Once the watchdog has been activated, the function block instance must be called cyclically at shorter intervals than tTimeout, since the CX1020 restarts automatically when tTimeout has elapsed. The watchdog can therefore be used to automatically reboot systems, which have entered an infinite loop or where the PLC has become stuck.

The watchdog can be deactivated via bEnable = FALSE or tTimeout = T#0s.

**NOTE**

The watchdog must be deactivated before breakpoints are used, before a PLC reset or an overall reset, before a TwinCAT stop, before switching to Config mode or before the configuration is activated, because otherwise the CX1020 would reboot immediately once tTimeout has elapsed.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>tTimeout : TIME;</td>
</tr>
<tr>
<td>bEnable : BOOL;</td>
</tr>
<tr>
<td>END_VAR</td>
</tr>
</tbody>
</table>

**tTimeout**: Watchdog time, if expired a reboot is automatically executed.

**bEnable**: Activate or deactivate the watchdog.
**VAR_OUTPUT**

```language=plaintext
VAR_OUTPUT
    bEnabled : BOOL;
    bError  : BOOL;
END_VAR
```

*bEnabled*: TRUE = Watchdog is activated, FALSE = Watchdog is not activated.

*bError*: Error when activate or deactivate the watchdog.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (x86)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

### 5.5 FB_CX1030SetWatchdog

The function block FB_CX1030SetWatchdog activates a hardware watchdog on the CX1030. The watchdog is activated via bEnable = TRUE and the tTimeOut time. The tTimeOut time can be a minimum of 2 seconds and a maximum of 255 seconds.

Once the watchdog has been activated, the function block instance must be called cyclically at shorter intervals than tTimeOut, since the CX1030 restarts automatically when tTimeOut has elapsed. The watchdog can therefore be used to automatically reboot systems, which have entered an infinite loop or where the PLC has become stuck.

The watchdog can be deactivated via bEnable = FALSE or tTimeOut = T#0s.

**NOTE**

The watchdog must be deactivated before breakpoints are used, before a PLC reset or an overall reset, before a TwinCAT stop, before switching to Config mode or before the configuration is activated, because otherwise the CX1030 would reboot immediately once tTimeOut has elapsed.

**VAR_INPUT**

```language=plaintext
VAR_INPUT
    tTimeout : TIME;
    bEnable  : BOOL;
END_VAR
```

*tTimeout*: Watchdog time, if expired a reboot is automatically executed.

*bEnable*: Activate or deactivate the watchdog.

**VAR_OUTPUT**

```language=plaintext
VAR_OUTPUT
    bEnabled : BOOL;
    bError  : BOOL;
END_VAR
```

*bEnabled*: TRUE = Watchdog is activated, FALSE = Watchdog is not activated.

*bError*: Error when activate or deactivate the watchdog.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (x86)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>
6.1 F_GetVersionTcCXSystem

FUNCTION F_GetVersionTcCXSystem : UINT
VAR_INPUT
nVersionElement : INT;
END_VAR

nVersionElement : Version element:
• 1 : major number;
• 2 : minor number;
• 3 : revision number;

6.2 F_GetVersionTcSystemCX1000

FUNCTION F_GetVersionTcSystemCX1000 : UINT
VAR_INPUT
nVersionElement : INT;
END_VAR

nVersionElement : Version element:
• 1 : major number;
• 2 : minor number;
• 3 : revision number;

6.3 F_GetVersionTcSystemCX1010

FUNCTION F_GetVersionTcSystemCX1010 : UINT
VAR_INPUT
nVersionElement : INT;
END_VAR

nVersionElement : Version element:
6.4 F_GetVersionTcSystemCX1020

The function returns library version info.

```
FUNCTION F_GetVersionTcSystemCX1020 : UINT
VAR_INPUT
nVersionElement : INT;
END_VAR
```

```
F_GetVersionTcSystemCX1020
nVersionElement
```

6.5 F_GetVersionTcSystemCX1030

The function returns library version info.

```
FUNCTION F_GetVersionTcSystemCX1030 : UINT
VAR_INPUT
nVersionElement : INT;
END_VAR
```

```
F_GetVersionTcSystemCX1030
nVersionElement
```

6.6 F_GetVersionTcSystemCX5010

The function returns library version info.

```
FUNCTION F_GetVersionTcSystemCX5010 : UINT
VAR_INPUT
nVersionElement : INT;
END_VAR
```

```
F_GetVersionTcSystemCX5010
nVersionElement
```
nVersionElement : Version element:
   • 1 : major number;
   • 2 : minor number;
   • 3 : revision number;

6.7  F_GetVersionTcSystemCX5020

The function returns library version info.

FUNCTION F_GetVersionTcSystemCX5020 : UINT
VAR_INPUT
   nVersionElement : INT;
END_VAR

nVersionElement : Version element:
   • 1 : major number;
   • 2 : minor number;
   • 3 : revision number;

6.8  F_GetVersionTcSystemCX9000

The function returns library version info.

FUNCTION F_GetVersionTcSystemCX9000 : UINT
VAR_INPUT
   nVersionElement : INT;
END_VAR

nVersionElement : Version element:
   • 1 : major number;
   • 2 : minor number;
   • 3 : revision number;

6.9  F_GetVersionTcSystemCX9010

The function returns library version info.
FUNCTION F_GetVersionTcSystemCX9010 : UINT
VAR_INPUT
nVersionElement : INT;
END_VAR

nVersionElement : Version element:
- 1 : major number;
- 2 : minor number;
- 3 : revision number;

6.10 F_CXSubTimeStamp

For subtraction TwinCAT3.1 also offers 64-bit data types (LINT/ULINT, LWORD) that can be executed directly (A-B) or there are TC2-compatible functions for 64-bit operations that should be used as an alternative to F_CXSubTimeStamp.

The function F_CXSubTimeStamp executes a 64-bit subtraction time stamp A - time stamp B and converts the result to µs. The required 64-bit time stamps with 100 ns resolution can be read from the system with the function block GETCPU_COUNTER.

If the difference between time stamp A and time stamp B is negative or greater than 4294967295 us, the maximum value 4294967295 us is returned. This corresponds to 71 minutes, 34 seconds, 967 milliseconds and 295 microseconds. In such cases the function UInt64Sub64() from TcUtilities.lib can be used to execute a complete 64-bit subtraction with 64-bit result in [100 ns].

FUNCTION F_CXSubTimeStamp : UDINT
VAR_INPUT
  nTimeStampLoDW_A : UDINT; (* 2*32 bit time stamp A: low DWORD *)
  nTimeStampHiDW_A : UDINT; (* 2*32 bit time stamp A: high DWORD *)
  nTimeStampLoDW_B : UDINT; (* 2*32 bit time stamp B: low DWORD *)
  nTimeStampHiDW_B : UDINT; (* 2*32 bit time stamp B: high DWORD *)
END_VAR

nTimeStampLoDW_A: lower 32bit of time stamp A.
nTimeStampHiDW_A: upper 32bit of time stamp A.
nTimeStampLoDW_B: lower 32bit of time stamp B.
nTimeStampHiDW_B: upper 32bit of time stamp B.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1</td>
<td>CX (WES7/Win7/Win10: TC RT x86/x64, WEC6/7 :TC TR x86, WEC7: TC CE7 ARMV7)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

6.11 F_CX1000SetWatchdog
The function F_CX1000SetWatchdog activates a hardware watchdog on the CX1000. The watchdog is activated via bEnable = TRUE and the tTimeout time. The tTimeout time can be a minimum of several PLC task cycles (multiple of the call time of the function F_CX1000SetWatchdog) and a maximum of 65 s and 535 ms.

Once the watchdog has been activated, the function must be called cyclically at shorter intervals than tTimeOut, since the CX1000 restarts automatically when tTimeOut has elapsed. The watchdog can therefore be used to automatically reboot systems, which have entered an infinite loop or where the PLC has become stuck.

The watchdog can be deactivated via bEnable = FALSE or tTimeOut = 0.

### NOTE

The watchdog must be deactivated before breakpoints are used, before a PLC reset or an overall reset, before a TwinCAT stop, before switching to Config mode or before the configuration is activated, because otherwise the CX1000 would reboot immediately once the timeout has elapsed.

**FUNCTION F_CX1000SetWatchdog: BOOL**

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tTimeout</td>
<td>TIME</td>
</tr>
<tr>
<td>bEnable</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

**tTimeOut**: Watchdog time, if expired a reboot is automatically executed.

**bEnable**: Activate or deactivate the watchdog.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (x86)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

### 6.12 F_CX9000SetWatchdog

The function F_CX9000SetWatchdog activates a hardware watchdog on the CX9000. The watchdog is activated via bEnable = TRUE and the tTimeOut time. The tTimeOut time can be a maximum of 65 seconds and 535 milliseconds.

Once the watchdog has been activated, the function must be called cyclically at shorter intervals than tTimeOut, since the CX9000 restarts automatically when tTimeOut has elapsed. The watchdog can therefore be used to automatically reboot systems, which have entered an infinite loop or where the PLC has become stuck.

The watchdog can be deactivated via bEnable = FALSE or tTimeOut = T#0s.

### NOTE

The watchdog must be deactivated before breakpoints are used, before a PLC reset or an overall reset, before a TwinCAT stop, before switching to Config mode or before the configuration is activated, because otherwise the CX9000 would reboot immediately once tTimeOut has elapsed.

**FUNCTION F_CX9000SetWatchdog: BOOL**

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tTimeout</td>
<td>TIME</td>
</tr>
<tr>
<td>bEnable</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

**tTimeOut**: Watchdog time, if expired a reboot is automatically executed.
bEnable: Activate or deactivate the watchdog.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (ARM)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

### 6.13 F_CX9010SetWatchdog

The function F_CX9010SetWatchdog activates a hardware watchdog on the CX9000. The watchdog is activated via bEnable = TRUE and the tTimeOut time. The tTimeOut time can be a maximum of 65 seconds and 535 milliseconds.

Once the watchdog has been activated, the function must be called cyclically at shorter intervals than tTimeOut, since the CX9010 restarts automatically when tTimeOut has elapsed. The watchdog can therefore be used to automatically reboot systems, which have entered an infinite loop or where the PLC has become stuck.

The watchdog can be deactivated via bEnable = FALSE or tTimeOut = T#0s.

**NOTE**

The watchdog must be deactivated before breakpoints are used, before a PLC reset or an overall reset, before a TwinCAT stop, before switching to Config mode or before the configuration is activated, because otherwise the CX9010 would reboot immediately once tTimeOut has elapsed.

**FUNCTION F_CX9010SetWatchdog: BOOL**

```plaintext
VAR_INPUT
   tTimeout : TIME;
   bEnable  : BOOL;
END_VAR
```

tTimeout: Watchdog time, if expired a reboot is automatically executed.

bEnable: Activate or deactivate the watchdog.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (ARM)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>
7 Data Types

7.1 [obsolete]

7.1.1 ST_CX_DeviceIdentification

```
TYPE ST_CxDeviceIdentification :
  STRUCT
    strTargetType       : STRING(20);
    strHardwareModel    : STRING(10);
    strHardwareSerialNo : STRING(12);
    strHardwareVersion  : STRING(4);
    strHardwareDate     : STRING(10);
    strHardwareCPU      : STRING(10);
    strImageDevice      : STRING(20);
    strImageVersion     : STRING(10);
    strImageLevel       : STRING(10);
    strImageOsName      : STRING(20);
    strImageOsVersion   : STRING(8);
    strTwinCATVersion   : STRING(4);
    strTwinCATRevision  : STRING(4);
    strTwinCATBuild     : STRING(8);
    strTwinCATLevel     : STRING(20);
    strAmsNetId      : STRING(23);
  END_STRUCT
END_TYPE
```

**strTargetType**: Type of the target system, e.g. 'CX1000-CE', ...

**strHardwareModel**: Hardware model, e.g. '1001'

**strHardwareSerialNo**: Hardware serial number, e.g. '123'

**strHardwareVersion**: Hardware version, e.g. '1.7'

**strHardwareDate**: Hardware production date, e.g. '18.8.06'

**strHardwareCPU**: Hardware CPU architecture, e.g. 'INTELx86', 'ARM', 'UNKNOWN' or '' (empty string)

**strImageDevice**: Software platform, e.g. 'CX1000', ...

**strImageVersion**: Version of the software platform, e.g. '2.15'

**strImageRevision**: Version of the software platform, e.g. 'HMI'

**strImageOsName**: Name of the operating system, e.g. 'Windows CE'

**strImageOsVersion**: Version of the operating system, e.g. '5.0'

**strTwinCATVersion**: TwinCAT version, e.g. for TwinCAT 2.10.1307: '2'

**strTwinCATRevision**: TwinCAT Reversion, e.g. for TwinCAT 2.10.1307: '10'

**strTwinCATBuild**: TwinCAT Build, e.g. for TwinCAT 2.10.1307: '1307'

**strTwinCATLevel**: Registered TwinCAT level, e.g. 'PLC', 'NC-PTP', 'NC-I', ...

**strAmsNetId**: TwinCAT AMS-NetID, e.g. '5.0.252.31.1.1'

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (x86)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>
### 7.1.2 ST_CxDeviceIdentificationEx

```
TYPE ST_CxDeviceIdentificationEx:
  STRUCT
    strTargetType : STRING(30);
    strHardwareModel : STRING(16);
    strHardwareSerialNo : STRING(16);
    strHardwareVersion : STRING(8);
    strHardwareDate : STRING(12);
    strHardwareCPU : STRING(20);
    strImageDevice : STRING(48);
    strImageVersion : STRING(32);
    strImageLevel : STRING(32);
    strImageOsName : STRING(48);
    strImageOsVersion : STRING(8);
    strTwinCATVersion : STRING(4);
    strTwinCATRevision : STRING(4);
    strTwinCATBuild : STRING(8);
    strTwinCATLevel : STRING(20);
    strAmsNetId : T_AMSNetId;
  END_STRUCT
  END_TYPE
```

- **strTargetType**: Type of the target system, e.g. 'CX1000-CE', ...
- **strHardwareModel**: Hardware model, e.g. '1001'
- **strHardwareSerialNo**: Hardware serial number, e.g. '123'
- **strHardwareVersion**: Hardware version, e.g. '1.7'
- **strHardwareDate**: Hardware production date, e.g. '18.8.06'
- **strHardwareCPU**: Hardware CPU architecture, e.g. 'INTELx86', 'ARM', 'UNKNOWN' or '' (empty string)
- **strImageDevice**: Software platform, e.g. 'CX1000', ...
- **strImageVersion**: Version of the software platform, e.g. '2.15'
- **strImageVersion**: Version of the software platform, e.g. 'HMI'
- **strImageOsVersion**: Name of the operating system, e.g. 'Windows CE'
- **strImageOsVersion**: Version of the operating system, e.g. '5.0'
- **strTwinCATVersion**: TwinCAT version, e.g. for TwinCAT 2.10.1307: '2'
- **strTwinCATRevision**: TwinCAT Reversion, e.g. for TwinCAT 2.10.1307: '10'
- **strTwinCATBuild**: TwinCAT Build, e.g. for TwinCAT 2.10.1307: '1307'
- **strTwinCATLevel**: Registered TwinCAT level, e.g. 'PLC', 'NC-PTP', 'NC-I', ....
- **strAmsNetId**: TwinCAT AMS-NetID, e.g. '5.0.252.31.1.1'

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (x86)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>

### 7.2 ST_CX_ProfilerStruct

```
TYPE ST_CX_ProfilerStruct:
  STRUCT
    dwLastExecTime : DWORD;
    dwMinExecTime : DWORD;
    dwMaxExecTime : DWORD;
    dwAverageExecTime : DWORD;
    dwMeasureCycle : DWORD;
  END_STRUCT
  END_TYPE
```

- **dwLastExecTime**: Last execution time
- **dwMinExecTime**: Minimum execution time
- **dwMaxExecTime**: Maximum execution time
- **dwAverageExecTime**: Average execution time
- **dwMeasureCycle**: Measurement cycle
Data Types

**dwLastExecTime**: The most recently measured value for the execution time in [µs]

**dwMinExecTime**: The minimum execution time in [µs]

**dwMaxExecTime**: The maximum execution time in [µs]

**dwAverageExecTime**: The mean execution time for the last 100 measurements in [µs]

**dwMeasureCycle**: The number of measurements that have already been carried out

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.0</td>
<td>CX (x86)</td>
<td>Tc2_SystemCX</td>
</tr>
</tbody>
</table>
8 Global constants

8.1 Library version

All libraries have a specific version. This version is inter alia shown in the PLC library repository too. A global constant contains the library version information:

```
VAR_GLOBAL CONSTANT
  stLibVersion_Tc2_SystemCX : ST_LibVersion;
END_VAR
```

**stLibVersion_Tc2_SystemCX**: version information of the Tc2_SystemCX library (type: ST_LibVersion).

To compare the existing version to a required version the function `F_CmpLibVersion` (defined in Tc2_System library) is offered.

- All other possibilities known from TwinCAT 2 to query a library version are obsolete!
More Information:
www.beckhoff.de/te1000