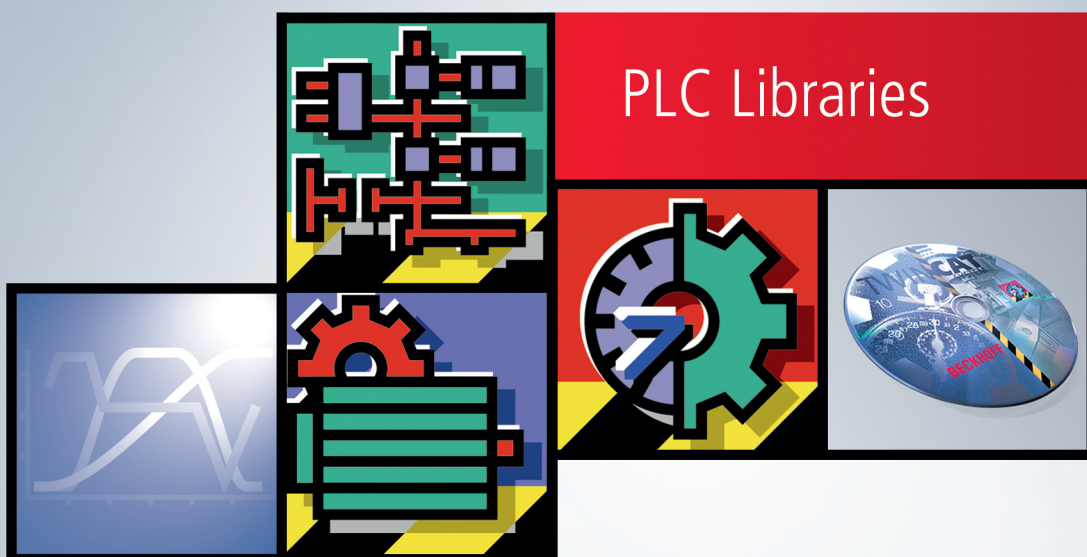


Manual | EN

TX1200

TwinCAT 2 | PLC Library: TcDMX



PLC Libraries

Table of contents

1 Foreword	5
1.1 Notes on the documentation	5
1.2 Safety instructions	6
1.3 Notes on information security.....	7
2 Target groups	8
3 DMX	9
4 Integration into TwinCAT (CX9020)	10
5 Programming	14
5.1 Overview function blocks	14
5.2 FB_DMXDiscovery	16
5.3 FB_DMXDiscovery512	17
5.4 FB_DMXSendRDMCommand	19
5.5 FB_EL6851Communication	20
5.6 FB_EL6851CommunicationEx	22
5.7 FB_DMXGetIdentifyDevice	24
5.8 FB_DMXSetIdentifyDevice.....	26
5.9 FB_DMXSetResetDevice	27
5.10 FB_DMXDiscMute.....	28
5.11 FB_DMXDiscUniqueBranch	29
5.12 FB_DMXDiscUnMute	30
5.13 FB_DMXGetLampHours	32
5.14 FB_DMXGetLampOnMode	33
5.15 FB_DMXSetLampHours.....	34
5.16 FB_DMXSetLampOnMode	35
5.17 FB_DMXGetDeviceInfo	36
5.18 FB_DMXGetDeviceLabel	37
5.19 FB_DMXGetDeviceModelDescription	38
5.20 FB_DMXGetManufacturerLabel	39
5.21 FB_DMXGetProductDetailIdList.....	40
5.22 FB_DMXGetSoftwareVersionLabel.....	42
5.23 FB_DMXSetDeviceLabel	43
5.24 FB_DMXClearStatusId	44
5.25 FB_DMXGetStatusIdDescription.....	45
5.26 FB_DMXGetStatusMessages	46
5.27 FB_DMXGetParameterDescription	47
5.28 FB_DMXGetSupportedParameters.....	48
5.29 FB_DMXGetSensorDefinition	49
5.30 FB_DMXGetSensorValue	50
5.31 FB_DMXGetDMX512PersonalityDescription	52
5.32 FB_DMXGetDMX512StartAddress	53
5.33 FB_DMXGetSlotDescription.....	54
5.34 FB_DMXGetSlotInfo.....	55
5.35 FB_DMXSetDMX512StartAddress	56

5.36	Data types	57
5.36.1	E_DMXCommandClass	57
5.36.2	E_DMXDataType	57
5.36.3	E_DMXLampOnMode	58
5.36.4	E_DMXParameterDescriptionCommandClass	58
5.36.5	E_DMXParameterId	58
5.36.6	E_DMXProductDetail	59
5.36.7	E_DMXResetDeviceType	61
5.36.8	E_DMXSensorType	61
5.36.9	E_DMXSensorUnit	61
5.36.10	E_DMXSensorUnitPrefix	62
5.36.11	E_DMXSlotDefinition	63
5.36.12	E_DMXSlotType	63
5.36.13	E_DMXStatusType	64
5.36.14	ST_DMX512Personality	64
5.36.15	ST_DMX512PersonalityDescription	64
5.36.16	ST_DMXCommandBuffer	64
5.36.17	ST_DMXDeviceInfo	65
5.36.18	ST_DMXMac	65
5.36.19	ST_DMXMessageQueue	65
5.36.20	ST_DMXMessageQueueItem	65
5.36.21	ST_DMXParameterDescription	66
5.36.22	ST_DMXProductCategory	66
5.36.23	ST_DMXRDMProtocolVersion	66
5.36.24	ST_DMXResponseTable	66
5.36.25	ST_DMXResponseTableItem	67
5.36.26	ST_DMXSensorDefinition	67
5.36.27	ST_DMXSensorValue	67
5.36.28	ST_DMXSlotInfo	67
5.36.29	ST_DMXStatusMessage	68
5.36.30	ST_EL6851InData	68
5.36.31	ST_EL6851InDataEx	68
5.36.32	ST_EL6851OutData	69
5.37	Error codes	69
6	Appendix	71
6.1	Transmission of cyclic process data as DMX master (EL6851)	71
6.2	Receipt of 64 bytes data to two DMX slaves (EL6851-0010) in each case	75
6.3	Configuration of DMX slaves via Remote Device Management (RDM)	78
6.4	DMX-Master with BC9191-0100	79
6.5	Support and Service	80

1 Foreword

1.1 Notes on the documentation

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

It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.

It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

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

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.

We reserve the right to revise and change the documentation at any time and without prior announcement. No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.

Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:

EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
with corresponding applications or registrations in various other countries.



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

Copyright

© 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.

1.3 Notes on information security

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

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

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

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

2 Target groups

The user of this library requires basic knowledge of the following.

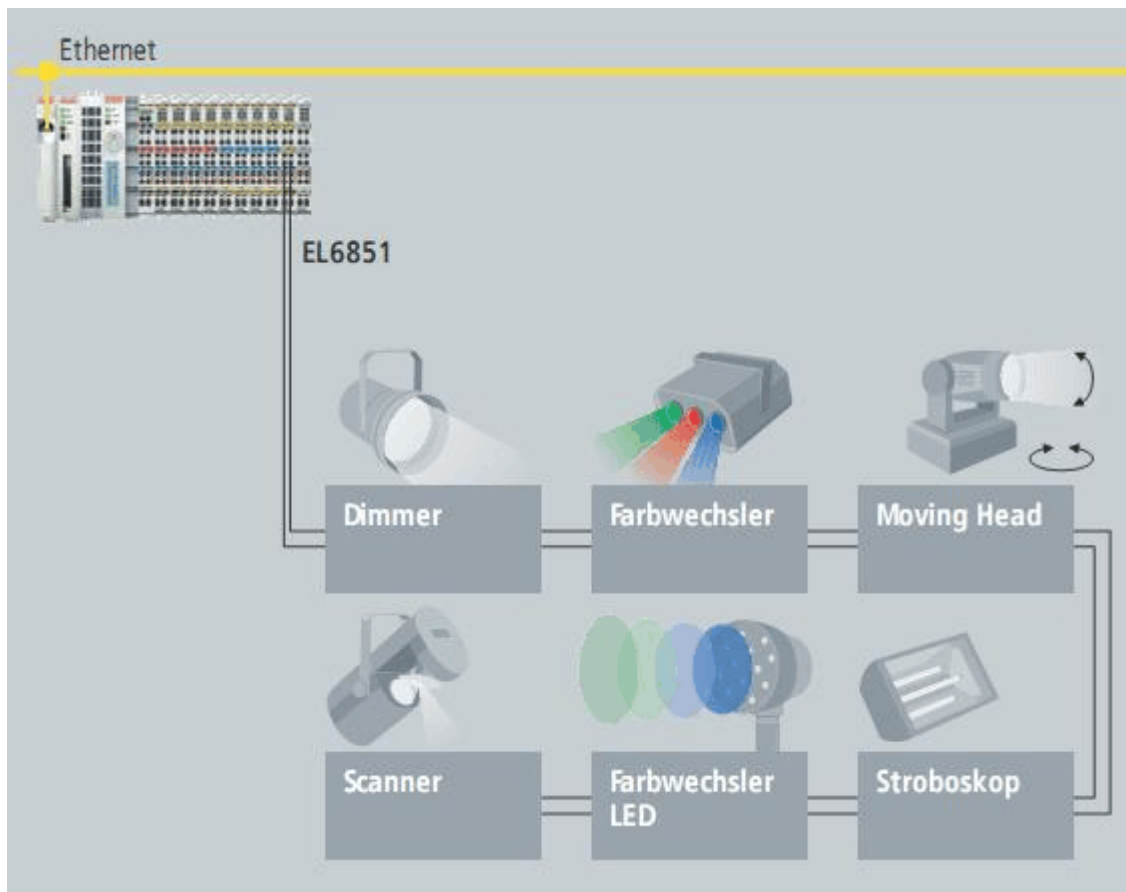
- TwinCAT PLC-Control
- TwinCAT System Manager
- PCs and networks
- Structure and properties of the Beckhoff Embedded PC and its Bus Terminal system
- Technology of DMX devices
- Relevant safety regulations for building technical equipment

This software library is intended for building automation system partners of Beckhoff Automation GmbH. The system partners operate in the field of building automation and are concerned with the installation, commissioning, expansion, maintenance and service of measurement, control and regulating systems for the technical equipment of buildings.

3 DMX

DMX is the standard protocol for controlling professional stage and effect lighting equipment, which is used, for example, for the dynamic lighting of showrooms and salesrooms as well as for exclusive plays of light and color in prestigious buildings, such as hotels and event centers. Color mixing and brightness values are transmitted to DMX devices that are static light sources, whilst moving sources of light additionally receive spatial coordinates. EtherCAT's high data transfer rate enables light settings to be updated at a higher rate, with the result that changes of light and color are perceived by the eye as being more harmonious. The [EL6851](#) can be used to control DMX devices with three axes, such as scanners, moving heads or spotlights; the implementation of the RDM protocol (**R**emote **D**evice **M**anagement) for DMX-internal diagnosis and parameterization is possible with TwinCAT function blocks.

The DMX master transmits new settings to the slaves cyclically at 250 kBaud to generate dynamic lighting changes and plays of color. In the DMX protocol, a maximum of 32 slaves are allowed in one strand without repeaters. The 512 byte long frame in the DMX protocol is termed a "Universe". 512 channels are available in it, each of which represents a device setting with 8-bit resolution, i.e. in 256 steps, e.g. for dimming, color, focus etc. In the case of moving light sources, additional settings such as inclination, swiveling and speed (with 8-bit or 16-bit resolution) occupy additional channels, so that the 512 channels are only indirectly sufficient for 32 devices. Furthermore, if the universe is fully utilized a frame will require 22 ms for internal DMX circulation, which means a refresh rate of 44 Hz. Light changes at this frequency are perceived to be unharmonious; the transitions only appear to be harmonious from a frequency of >200 Hz. The circulation period can be shortened by reducing the amount of user data; the optimum has proven to be a utilization of 64 bytes (frequency >300 Hz), with which 64 channels are available per universe.



The integration of several universes in a controller becomes simple with the EL6851: EtherCAT can transfer large amounts of data quickly, the EtherCAT protocol is retained until inside the terminal and the terminal supports various mapping sizes (64 to 512 bytes). Hence, if several master terminals are connected, each as its own universe, the time offset in transmitting from the controller to the master can be reduced significantly.

4 Integration into TwinCAT (CX9020)

This example explains how to write a simple PLC program for DMX in TwinCAT and how to link it with the hardware. Search for DMX devices.

<https://infosys.beckhoff.com/content/1033/tcplclibdmx/Resources/11977737739/.zip> <https://infosys.beckhoff.com/content/1033/tcplclibdmx/Resources/11977737739/.zip>

Hardware

Setting up the components

The following hardware is required:

- 1x Embedded PC CX9020
- 1x [DMX master terminal EL6851](#) [▶ 75]
- 1x end cap EL9111

Set up the hardware and the DMX components as described in the respective documents.

Software

Creation of the PLC program

Create a new PLC project for PC-based systems (ARM) and add the *TcDMX.lib* library.

Next, generate the following global variables:

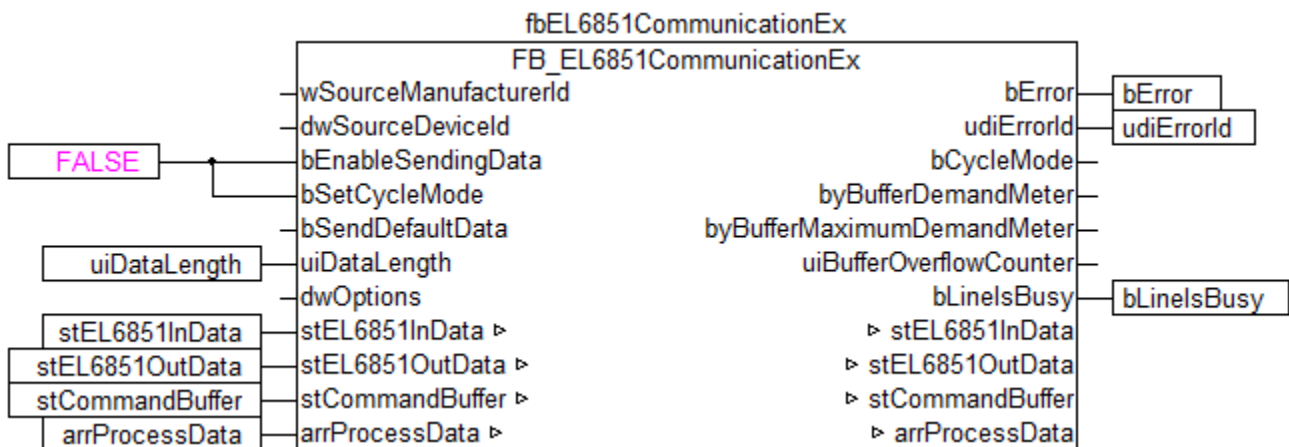
```
VAR_GLOBAL
    stEL6851InData      AT %I* : ST_EL6851InDataEx;
    stEL6851OutData     AT %Q* : ST_EL6851OutData;
    stCommandBuffer     : ST_DMXCommandBuffer;
END_VAR
```

stEL6851InData : Input variable for the DMX terminal.

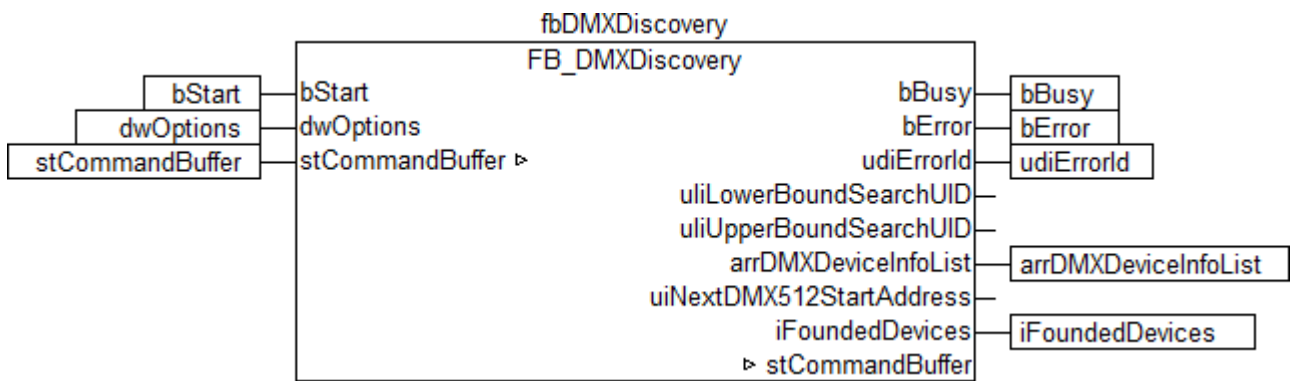
stEL6851OutData : Output variable for the DMX terminal.

stCommandBuffer : required for communication with DMX.

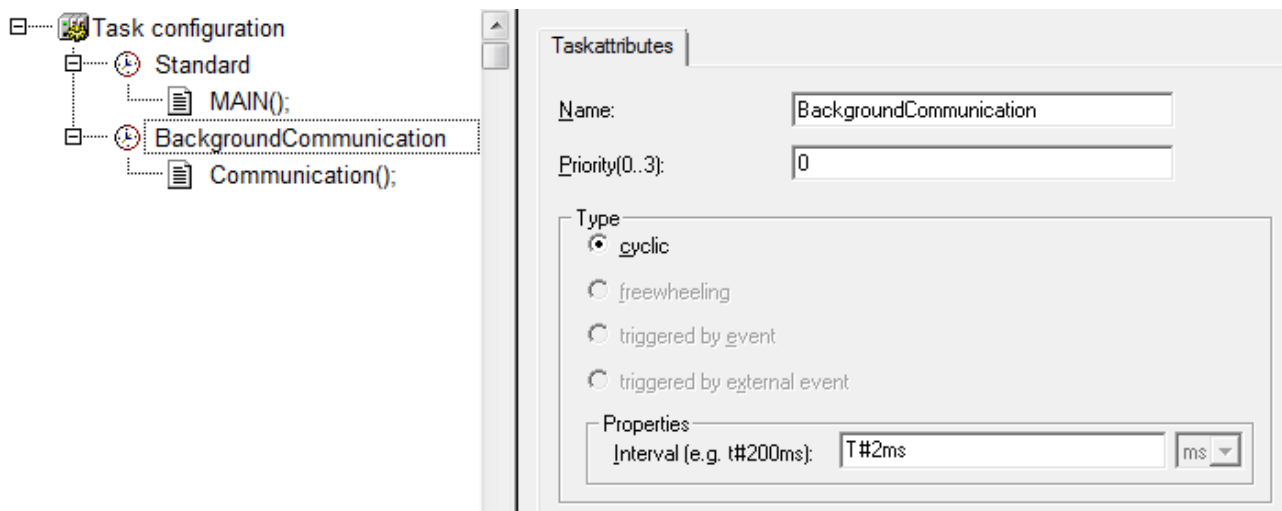
Then create a program (CFC) for background communication with DMX. The [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block is called in this program. Make sure to link the communication block with *stEL6851InData*, *stEL6851OutData* and *stCommandBuffer*.



Create a MAIN program (CFC) in which the block [FB_DMXDiscovery\(\)](#) [▶ 16] is called up. Connect the input *stCommandBuffer* of the block with the global variable *stCommandBuffer*.



Under Task Configuration create a new task for the background communication. Add the communication program to this task. Assign a higher priority (lower number) and shorter interval time to this task than for the standard task. More precise information on this can be found in the description of the [FB_EL6851CommunicationEx\(\)](#) [► 22] function block.



Load the project to the CX as the boot project and save it.

Configuration in the System Manager

Create a new TwinCAT System Manager project, select the CX as the target system, and search for the associated hardware.

Make the DMX inputs 1 to 64 available by opening the *Process data* tab of the EL6851 and selecting the option *0x1A01* under *Inputs*.

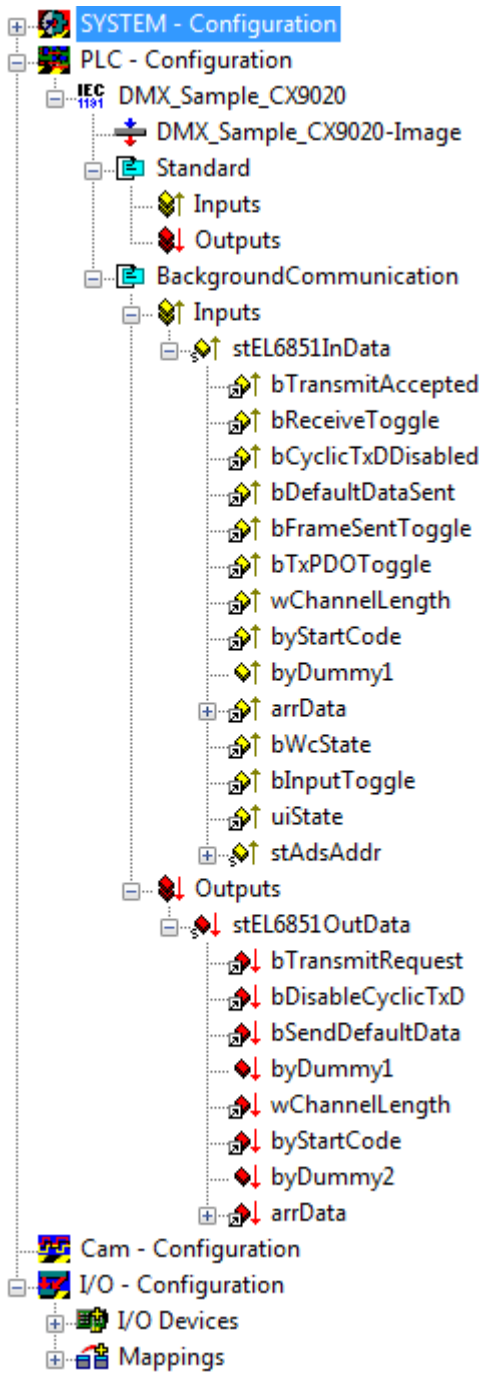
The screenshot displays the TwinCAT configuration environment. On the left, a tree view shows the project structure under 'I/O Devices', including 'Device 1 (EtherCAT)' with sub-items like 'Device 1-Image', 'Device 1-Image-Info', 'Inputs', 'Outputs', 'InfoData', and three terminal blocks: 'Term 1 (EK1200)', 'Term 2 (EL6851)', and 'Term 3 (EL9011)'. On the right, the 'Sync Manager' dialog is open, showing a table of synchronization manager (SM) entries. The entry for SM 3 (Inputs, 70) is highlighted with a red box. Below the table, the 'PDO Assignment (0x1C13)' section shows two entries, '0x1A00' and '0x1A01', both checked and highlighted with a red box.

SM	Size	Type	Flags
0	128	MbxOut	
1	128	MbxIn	
2	518	Outputs	
3	70	Inputs	

PDO Assignment (0x1C13):

- 0x1A00
- 0x1A01

Add the PLC program created above under PLC configuration. The two tasks are listed when the PLC project is expanded in the tree view. Expand the tasks – the global input and output variables *stEL6851InData* and *stEL6851OutData* should be allocated to the background communication task since the variables are to be processed faster. Move them via drag & drop.



Now link the global variables of the PLC program with the Bus Terminal inputs and outputs, create the allocations, and activate the configuration. Then start the device in run mode. Your CX is now ready for use.

Start the search of DMX devices by switching the variable *bStart* to TRUE. The number of found devices is stored in variable *iFoundedDevices* and additional information in *arrDMXDeviceInfoList*.

Also see about this

- ST_EL6851InDataEx [▶ 68]
- ST_EL6851OutData [▶ 69]
- ST_DMXCommandBuffer [▶ 64]

5 Programming

● Installation

i Beginning with TwinCAT 2.11 Build 2229 (R3 and x64 Engineering), the library "TcDMX.lib" will be installed automatically.

● Name of the library

i This library replaces the "TcEL6851.lib".

Hardware documentation in Beckhoff Information System: [EL6851 - DMX Master/Slave Terminal \[▶ 75\]](#)

Further libraries are required

For PC systems (x86) and Embedded-PCs (CXxxxx):

- Standard.lib
- TcBase.lib
- TcSystem.lib
- TcUtilities.lib

For Bus Terminal Controller of BCxx00, BCxx50, BCxx20, BC9191 and BXxx00 series:

- not available

● Memory usage

i By linking the library PLC program memory is already consumed. Depending on the application program the remaining memory cannot be sufficient.

The use of the TwinCAT libraries is recommended both for the RDM protocol and also for the transmission of the cyclic process data to the DMX slaves. Examples of both variants can be found in the appendix.

Description of the Library

Only one block is required for basic communication. The FB_EL6851CommunicationEx takes over communication with the EL6851. This block can be used to switch between the RDM and the DMX protocols. If the DMX protocol is in use, then no RDM communication can take place and vice versa.

5.1 Overview function blocks

High Level

Name	Description
FB_DMXDiscovery [▶ 16]	Searches for up to 50 DMX devices and optionally sets the start address automatically.
FB_DMXDiscovery512 [▶ 17]	Searches for up to 512 DMX devices and optionally sets the start address automatically.

Low Level

Base

Name	Description
FB_DMXSendRDMCommand [▶ 19]	Sends a single RDM-command defined by the command-number.
FB_EL6851Communication [▶ 20]	Access to the EL6851 [▶ 75] .
FB_EL6851CommunicationEx [▶ 22]	Access to the EL6851 [▶ 75] .

Device Control Parameter Messages

Name	Description
FB_DMXGetIdentifyDevice [▶ 24]	Queries whether or not the identification of a DMX device is active.
FB_DMXSetIdentifyDevice [▶ 26]	Activates or deactivates the identification of a DMX device.
FB_DMXSetResetDevice [▶ 27]	Activates a reset in a DMX device.

Discovery Messages

Name	Description
FB_DMXDiscMute [▶ 28]	Sets the mute flag of a DMX device.
FB_DMXDiscUniqueBranch [▶ 29]	Queries whether DMX devices are located within a certain address range.
FB_DMXDiscUnMute [▶ 30]	Resets the mute flag of a DMX device.

Power/Lamp Setting Parameter Messages

Name	Description
FB_DMXGetLampHours [▶ 32]	Reads the number of hours in which the lamp was on.
FB_DMXGetLampOnMode [▶ 33]	Reads the parameter that defines the switch-on characteristics of the DMX device
FB_DMXSetLampHours [▶ 34]	Sets the operating hours counter for the lamp.
FB_DMXSetLampOnMode [▶ 35]	Defines the switch-on characteristics of the DMX device.

Product Information Messages

Name	Description
FB_DMXGetDeviceInfo [▶ 36]	Queries all relevant information from a DMX device.
FB_DMXGetDeviceLabel [▶ 37]	Reads a text from the DMX device, which contains a more detailed description of the device.
FB_DMXGetDeviceModelDescription [▶ 38]	Queries the description of the device type.
FB_DMXGetManufacturerLabel [▶ 39]	Queries the description of the DMX device manufacturer.
FB_DMXGetProductDetailIdList [▶ 40]	Queries the categories to which the DMX device belongs.
FB_DMXGetSoftwareVersionLabel [▶ 42]	Queries the description of the software version of the DMX device.
FB_DMXSetDeviceLabel [▶ 43]	Writes a description text into the DMX device.

Queued and Status Messages

Name	Description
FB_DMXClearStatusId [▶ 44]	Clears the message buffer in the DMX device.
FB_DMXGetStatusIdDescription [▶ 45]	Reads the text of a certain status ID from the DMX device.
FB_DMXGetStatusMessages [▶ 46]	Collects status or error information from a DMX device.

RDM Information Messages

Name	Description
FB_DMXGetParameterDescription [▶ 47]	Retrieves the definition of manufacturer-specific PIDs.
FB_DMXGetSupportedParameters [▶ 48]	Queries all supported parameters from a DMX device.

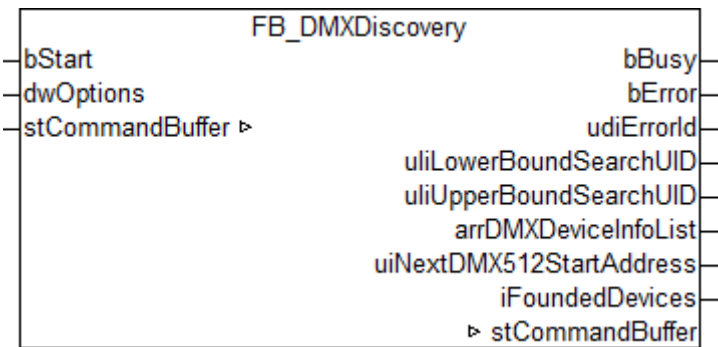
Sensor Parameter Messages

Name	Description
FB_DMXGetSensorDefinition [▶ 49]	Queries the definition of a specific sensor.
FB_DMXGetSensorValue [▶ 50]	Queries the current value of a sensor.

Setup Messages

Name	Description
FB_DMXGetDMX512PersonalityDescription [▶ 52]	Reads further <i>Personality</i> information from the DMX device.
FB_DMXGetDMX512StartAddress [▶ 53]	Queries the DMX512 start address.
FB_DMXGetSlotDescription [▶ 54]	Queries the text description for slot offsets.
FB_DMXGetSlotInfo [▶ 55]	Queries basic information about the functionality of the DMX512 slots from a DMX device.
FB_DMXSetDMX512StartAddress [▶ 56]	Sets the DMX512 start address.

5.2 FB_DMXDiscovery



This function block searches for up to 50 DMX devices and optionally sets the start address automatically. The most important information for the devices found is displayed in a structure.

VAR_INPUT

```
bStart          : BOOL;
dwOptions       : DWORD;
```

bStart: The block is activated by a positive edge at this input.

dwOptions: Options (see table). The individual constants must be linked with OR operators.

Constant	Description
DMX_OPTION_COMPLETE_NEW_DISCOVERY	All DMX devices are taken into account.
DMX_OPTION_SET_START_ADDRESS	The start address is set for all DMX devices found consecutively, starting from 1.
DMX_OPTION_OPTICAL_FEEDBACK	After a DMX device has been found, the IDENTIFY_DEVICE function is called for two seconds.

VAR_OUTPUT

```
bBusy           : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
uliLowerBoundSearchUID : T_ULARGE_INTEGER;
uliUpperBoundSearchUID : T_ULARGE_INTEGER;
```



```
arrDMXDeviceInfoList : ARRAY[1..50] OF ST_DMXDeviceInfo;
uiNextDMX512StartAddress : UINT;
iFoundedDevices : INT;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

uliLowerBoundSearchUID: During the search, the lower search address is sent to this output.

uliUpperBoundSearchUID: During the search, the upper search address is sent to this output.

arrDMXDeviceInfoList: Array with the most important information of the DMX devices found.

uiNextDMX512StartAddress: If the DMX_OPTION_SET_START_ADDRESS option is activated, then the start address that will be assigned to the next DMX device will be displayed at this output.

iFoundedDevices: During the search, the current number of devices found will be sent to this output.

VAR_IN_OUT

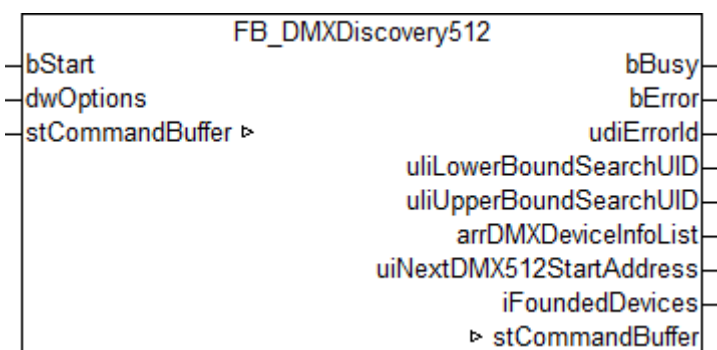
```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block.

Also see about this

- ▣ [ST_DMXDeviceInfo](#) [▶ 65]
- ▣ [ST_DMXCommandBuffer](#) [▶ 64]

5.3 FB_DMXDiscovery512



This function block searches for up to 512 DMX devices and optionally sets the start address automatically. The most important information for the devices found is displayed in a structure.

VAR_INPUT

```
bStart : BOOL;
dwOptions : DWORD;
```

bStart: The block is activated by a positive edge at this input.

dwOptions: Options (see table). The individual constants must be linked with OR operators.

Constant	Description
DMX_OPTION_COMPLETE_NEW_DISCOVERY	All DMX devices are taken into account.

Constant	Description
DMX_OPTION_SET_START_ADDRESS	The start address is set for all DMX devices found consecutively, starting from 1.
DMX_OPTION_OPTICAL_FEEDBACK	After a DMX device has been found, the IDENTIFY_DEVICE function is called for two seconds.

VAR_OUTPUT

```

bBusy          : BOOL;
bError         : BOOL;
udiErrorId     : UDINT;
uliLowerBoundSearchUID : T_ULARGE_INTEGER;
uliUpperBoundSearchUID : T_ULARGE_INTEGER;
arrDMXDeviceInfoList : ARRAY[1..512] OF ST_DMXDeviceInfo;
uiNextDMX512StartAddress : UINT;
iFoundedDevices : INT;

```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [► 69].

uliLowerBoundSearchUID: During the search, the lower search address is sent to this output.

uliUpperBoundSearchUID: During the search, the upper search address is sent to this output.

arrDMXDeviceInfoList: Array with the most important information of the DMX devices found.

uiNextDMX512StartAddress: If the DMX_OPTION_SET_START_ADDRESS option is activated, then the start address that will be assigned to the next DMX device will be displayed at this output.

iFoundedDevices: During the search, the current number of devices found will be sent to this output.

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [► 22] block.

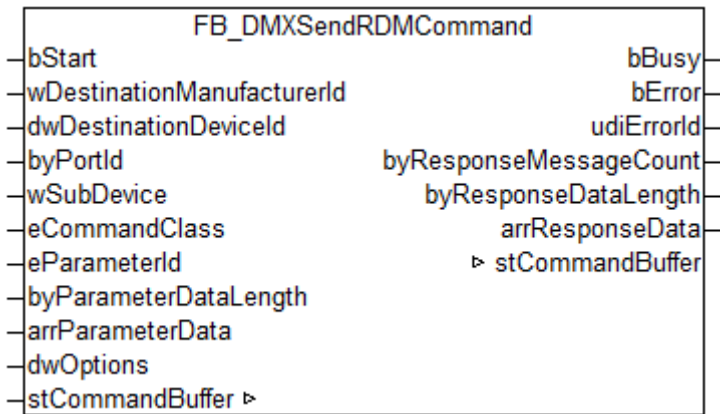
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2240	PC/CX	TcDMX-library higher than V1.1.0

Also see about this

- 📖 [ST_DMXDeviceInfo](#) [► 65]
- 📖 [ST_DMXCommandBuffer](#) [► 64]

5.4 FB_DMXSendRDMCommand



This function-block sends a single RDM-command defined by the command-number and, if applicable, by a parameter-value.

VAR_INPUT

```

bStart                : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId              : BYTE;
wSubDevice            : WORD;
eCommandClass        : E_DMXCommandClass;
eParameterId         : E_DMXParameterId;
byParameterDataLength : BYTE;
arrParameterData     : ARRAY[0..255] OF BYTE;
dwOptions             : DWORD := 0;
    
```

bStart: The block is activated by a rising edge at this input.

wDestinationManufacturerId: Unique manufacturer Id of the DMX device.

dwDestinationDeviceId: Unique device Id of the DMX device.

byPortId: The port Id field shall be set in the range of 1-255 identifying the controller port being used, such that the combination of source UID and port Id will uniquely identify the controller and port where the message originated.

wSubDevice: Sub-devices should be used in devices containing a repetitive number of similar modules, such as a dimmer rack. The Sub-Device field allows parameter messages to be addressed to a specific module within the device to set or get properties of that module.

eCommandClass: The command class (CC) specifies the action of the message.

eParameterId: The parameter Id is a 16-bit number that identifies a specific type of parameter data.

byParameterDataLength: The parameter data length (PDL) is the number of slots included in the parameter data area that it precedes. When this field is set to 0x00 it indicates that there is no parameter data following.

arrParameterData: The parameter data is of variable length. The content format is PID dependent.

dwOptions: Options (currently not used).

VAR_OUTPUT

```

bBusy                : BOOL;
bError               : BOOL;
udiErrorId           : UDINT;
byResponseMessageCount : BYTE;
byResponseDataLength : BYTE;
arrResponseData     : ARRAY[0..255] OF BYTE;
    
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

byResponseMessageCount: The message count is used by the DMX slave to indicate that additional messages are available. Messages can be read by the RDM command Get: QUEUED_MESSAGE.

byResponseDataLength: Contains the number of bytes returned by the RDM command.

arrResponseData: This output contains the response data of the RDM command. The length is variable and the content format is RDM command dependent.

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block.

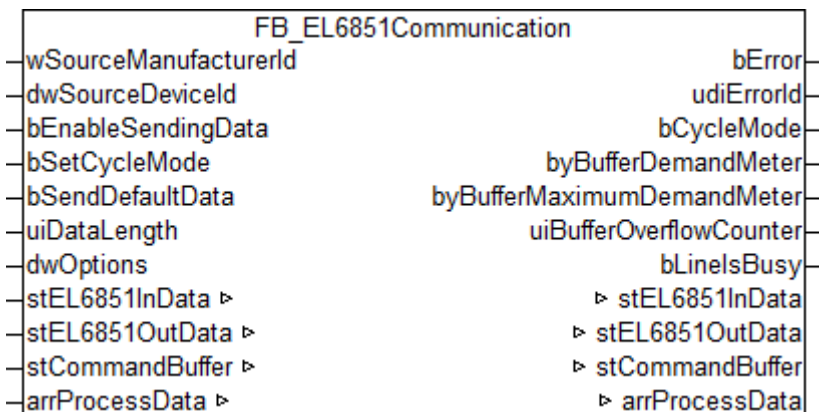
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

Also see about this

- ▣ [E_DMXCommandClass](#) [▶ 57]
- ▣ [E_DMXParameterId](#) [▶ 58]
- ▣ [ST_DMXCommandBuffer](#) [▶ 64]

5.5 FB_EL6851Communication



i This function block is obsolete. Instead, use [FB_EL6851CommunicationEx\(\)](#) [▶ 22].

The [EL6851](#) [▶ 75] should always be accessed via this block. This applies both to the transmission of the cyclic DMX data and to the transmission of the RDM commands.

If data is to be transmitted cyclically to the DMX devices, then set the *bEnableSendingData* input to TRUE, the *bSetCycleMode* input to TRUE, the *bSendDefaultData* input to FALSE and the *uiDataLength* input to the corresponding length (in bytes). The data to be transmitted can be specified via the *arrProcessData* variable.

If RDM commands are to be transmitted, then set the *bEnableSendingData* input to FALSE and the *bSetCycleMode* input to FALSE. The blocks for the DMX/RDM commands do not directly access the EL6851 process image, but store the individual DMX/RDM commands in a buffer instead. The `FB_EL6851Communication()` block reads the commands sequentially from this buffer and forwards them to the EL6851. This prevents multiple blocks accessing the EL6851 process image at the same time. The buffer in which the DMX/RDM commands are stored is contained in a variable of type `ST_DMXCommandBuffer`. There is one instance of the `FB_EL6851Communication()` block and one variable of type `ST_DMXCommandBuffer` per EL6851.

The extent to which the buffer is utilized can be determined from the outputs of the block. If the buffer is regularly overflowing, you should analyse the level of utilisation of the PLC task with the aid of the TwinCAT System Manager.

The `FB_EL6851Communication()` block can be called in a separate, faster task if necessary. In this case, the faster task in which the `FB_EL6851Communication()` block is called should have a higher priority than the TASK in which the block for the RDM commands is called.

You will find examples of both modes of operation in the appendix.

● Remarks concerning the IDs of DMX devices

i Each DMX device has a unique, fixed, 48-bit long address, also called Unique ID or UID for short. This address is composed of the manufacturer ID (16-bit) and the device ID (32-bit). The manufacturer ID identifies the manufacturer of the device and is issued by the ESTA (Entertainment Services and Technology Association). A list of all known manufacturer IDs can be found at http://www.esta.org/tsp/working_groups/CP/mfctrlIDs.php. The device ID is freely specified by the manufacturer. This is intended to ensure that each UID exists only once worldwide. The UID cannot normally be changed. The ESTA has given Beckhoff Automation the manufacturer ID 0x4241. Since the DMX master also has a UID, this should be specified in accordance with the ESTA (*wSourceManufacturerId* input).

VAR_INPUT

```
wSourceManufacturerId : WORD := 16#42_41;
dwSourceDeviceId      : DWORD := 16#12_13_14_15;
bEnableSendingData    : BOOL := TRUE;
bSetCycleMode         : BOOL := TRUE;
bSendDefaultData     : BOOL;
uiDataLength          : UINT;
dwOptions             : DWORD;
```

wSourceManufacturerId: Unique manufacturer ID of the DMX device. Should be 0x4241 according to the ESTA.

dwSourceDeviceId: Unique device ID of the DMX device. Can be freely assigned.

bEnableSendingData: If the terminal is in cycle mode (*bCycleMode* output = TRUE), then transmission can be activated (TRUE) or blocked (FALSE) with this block.

bSetCycleMode: Activates the cycle mode. The cyclic process data can be transmitted to the DMX devices in cycle mode. Cycle mode must be deactivated to transmit the RDM/DMX commands.

bSendDefaultData: The standard values will be transmitted in cycle mode if this input is active (TRUE).

uiDataLength: This input is only relevant if cycle mode is active. It indicates the length of the DMX512 frame in bytes.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bError                : BOOL;
udiErrorId            : UDINT;
bCycleMode            : BOOL;
byBufferDemandMeter   : BYTE;
byBufferMaximumDemandMeter : BYTE;
uiBufferOverflowCounter : UINT;
bLineIsBusy           : BOOL;
```

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

bCycleMode: Is TRUE if cycle mode is active (see also *bSetCycleMode* input).

byBufferDemandMeter: Buffer utilization (0 - 100%).

byBufferMaximumDemandMeter: Previous maximum utilization of the respective buffer (0 - 100%).

uiBufferOverflowCounter: Number of buffer overflows to date.

bLineIsBusy: This output is set as long as the `FB_EL6851Communication()` block is processing DMX/RDM commands.

VAR_IN_OUT

```
stEL6851InData      : ST_EL6851InData;
stEL6851OutData     : ST_EL6851OutData;
stCommandBuffer     : ST_DMXCommandBuffer;
arrProcessData      : ARRAY[1..512] OF BYTE;
```

stEL6851InData: Structure in the EL6851 input process image. It is used for communication from the [EL6851](#) [▶ 75] to the PLC.

stEL6851OutData: Structure in the EL6851 output process image. It is used for communication from the PLC to the [EL6851](#) [▶ 75].

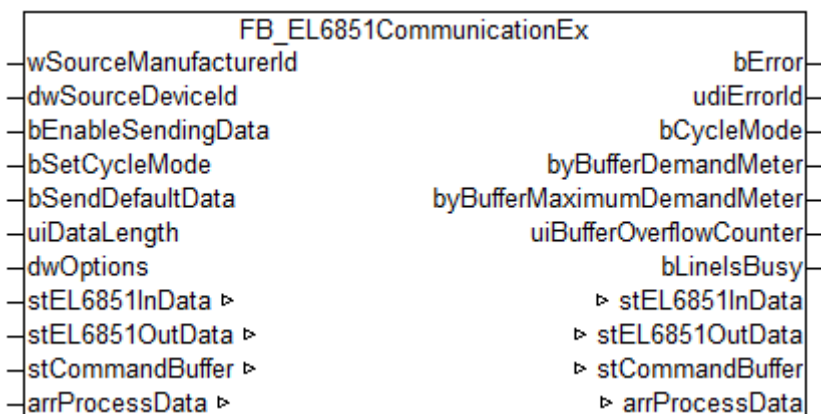
stCommandBuffer: A reference to the structure for communication (the buffer) with the `FB_EL6851Communication()`-block.

arrProcessData: The data that are to be transmitted cyclically to the DMX devices are transferred to the block via this variable. Cycle mode must be active for this to take place (see also *bSetCycleMode* input).

Also see about this

- ▣ [ST_EL6851InData](#) [▶ 68]
- ▣ [ST_EL6851OutData](#) [▶ 69]
- ▣ [ST_DMXCommandBuffer](#) [▶ 64]

5.6 FB_EL6851CommunicationEx



The [EL6851](#) [▶ 75] should always be accessed via this block. This applies both to the transmission of the cyclic DMX data and to the transmission of the RDM commands.

If data is to be transmitted cyclically to the DMX devices, then set the *bEnableSendingData* input to TRUE, the *bSetCycleMode* input to TRUE, the *bSendDefaultData* input to FALSE and the *uiDataLength* input to the corresponding length (in bytes). The data to be transmitted can be specified via the *arrProcessData* variable.

If RDM commands are to be transmitted, then set the *bEnableSendingData* input to FALSE and the *bSetCycleMode* input to FALSE. The blocks for the DMX/RDM commands do not directly access the EL6851 process image, but store the individual DMX/RDM commands in a buffer instead. The `FB_EL6851Communication()` block reads the commands sequentially from this buffer and forwards them to the EL6851. This prevents multiple blocks accessing the EL6851 process image at the same time. The buffer in which the DMX/RDM commands are stored is contained in a variable of type `ST_DMXCommandBuffer`. There is one instance of the `FB_EL6851Communication()` block and one variable of type `ST_DMXCommandBuffer` per EL6851.

The extent to which the buffer is utilized can be determined from the outputs of the block. If the buffer is regularly overflowing, you should analyse the level of utilisation of the PLC task with the aid of the TwinCAT System Manager.

The `FB_EL6851Communication()` block can be called in a separate, faster task if necessary. In this case, the faster task in which the `FB_EL6851Communication()` block is called should have a higher priority than the TASK in which the block for the RDM commands is called.

You will find examples of both modes of operation in the appendix.

i Remarks concerning the IDs of DMX devices

Each DMX device has a unique, fixed, 48-bit long address, also called Unique ID or UID for short. This address is composed of the manufacturer ID (16-bit) and the device ID (32-bit). The manufacturer ID identifies the manufacturer of the device and is issued by the ESTA ([Entertainment Services and Technology Association](http://www.esta.org/tsp/working_groups/CP/mfctrIDs.php)). A list of all known manufacturer IDs can be found at http://www.esta.org/tsp/working_groups/CP/mfctrIDs.php. The device ID is freely specified by the manufacturer. This is intended to ensure that each UID exists only once worldwide. The UID cannot normally be changed. The ESTA has given Beckhoff Automation the manufacturer ID 0x4241. Since the DMX master also has a UID, this should be specified in accordance with the ESTA (*wSourceManufacturerId* input).

VAR_INPUT

```
wSourceManufacturerId : WORD := 16#42_41;
dwSourceDeviceId     : DWORD := 16#12_13_14_15;
bEnableSendingData   : BOOL := TRUE;
bSetCycleMode        : BOOL := TRUE;
bSendDefaultData     : BOOL;
uiDataLength         : UINT;
dwOptions            : DWORD;
```

wSourceManufacturerId: Unique manufacturer ID of the DMX device. Should be 0x4241 according to the ESTA.

dwSourceDeviceId: Unique device ID of the DMX device. Can be freely assigned.

bEnableSendingData: If the terminal is in cycle mode (*bCycleMode* output = TRUE), then transmission can be activated (TRUE) or blocked (FALSE) with this block.

bSetCycleMode: Activates the cycle mode. The cyclic process data can be transmitted to the DMX devices in cycle mode. Cycle mode must be deactivated to transmit the RDM/DMX commands.

bSendDefaultData: The standard values will be transmitted in cycle mode if this input is active (TRUE).

uiDataLength: This input is only relevant if cycle mode is active. It indicates the length of the DMX512 frame in bytes.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bError           : BOOL;
udiErrorId       : UDINT;
bCycleMode       : BOOL;
byBufferDemandMeter : BYTE;
```

```
byBufferMaximumDemandMeter : BYTE;
uiBufferOverflowCounter    : UINT;
bLineIsBusy                : BOOL;
```

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

bCycleMode: Is TRUE if cycle mode is active (see also *bSetCycleMode* input).

byBufferDemandMeter: Buffer utilisation (0 – 100%).

byBufferMaximumDemandMeter: Previous maximum utilisation of the respective buffer (0 – 100%).

uiBufferOverflowCounter: Number of buffer overflows to date.

bLineIsBusy: This output is set as long as the `FB_EL6851Communication()` block is processing DMX/RDM commands.

VAR_IN_OUT

```
stEL6851InData      : ST_EL6851InDataEx;
stEL6851OutData     : ST_EL6851OutData;
stCommandBuffer     : ST_DMXCommandBuffer;
arrProcessData      : ARRAY[1..512] OF BYTE;
```

stEL6851InData: Structure in the EL6851 input process image. It is used for communication from the [EL6851](#) [▶ 75] to the PLC.

stEL6851OutData: Structure in the EL6851 output process image. It is used for communication from the PLC to the [EL6851](#) [▶ 75].

stCommandBuffer: A reference to the structure for communication (the buffer) with the `FB_EL6851Communication()`-block.

arrProcessData: The data that are to be transmitted cyclically to the DMX devices are transferred to the block via this variable. Cycle mode must be active for this to take place (see also *bSetCycleMode* input).

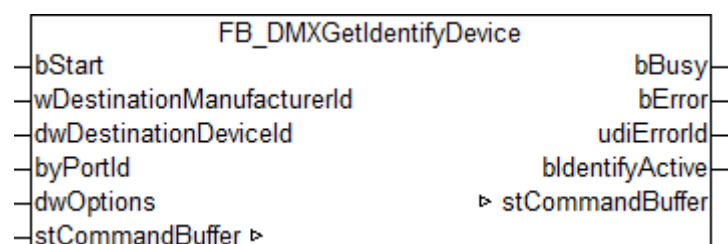
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

Also see about this

- ▣ [ST_EL6851InDataEx](#) [▶ 68]
- ▣ [ST_EL6851OutData](#) [▶ 69]
- ▣ [ST_DMXCommandBuffer](#) [▶ 64]

5.7 FB_DMXGetIdentifyDevice



This function block queries whether or not the identification of a DMX device is active.

Applying a positive edge to the *bStart* input starts the block, and the *bBusy* output goes TRUE. The *wDestinationManufacturerId* and *dwDestinationDeviceId* inputs address the DMX device. The *byPortId* input defines the channel within the addressed DMX device. If the execution of the command has been completed, the *bBusy* output goes back to FALSE. The *bError*, *udiErrorId* and *blIdentifyActive* outputs can now be processed. Further positive edges at the *bStart* input will be ignored as long as the block is active (*bBusy* is TRUE).

VAR_INPUT

```
bStart           : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId        : BYTE;
dwOptions       : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy           : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
blIdentifyActive : BOOL;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

blIdentifyActive: If the execution of the command has been completed (*bBusy* is FALSE), then the state of identification of the DMX device is displayed at this output.

VAR_IN_OUT

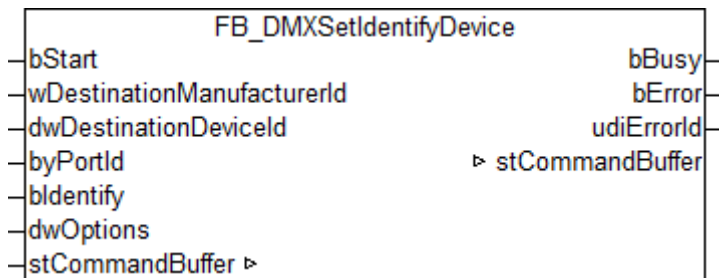
```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block.

Also see about this

📖 [ST_DMXCommandBuffer](#) [▶ 64]

5.8 FB_DMXSetIdentifyDevice



This function block activates or deactivates the identification of a DMX device.

Applying a positive edge to the `bStart` input starts the block, and the `bBusy` output goes TRUE. The `wDestinationManufacturerId` and `dwDestinationDeviceId` inputs address the DMX device. The `byPortId` input defines the channel within the addressed DMX device. If the execution of the command has been completed, the `bBusy` output goes back to FALSE. The `bError` and `udiErrorId` outputs can now be processed. Further positive edges at the `bStart` input will be ignored as long as the block is active (`bBusy` is TRUE).

VAR_INPUT

```
bStart           : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId        : BYTE;
bIdentify       : BOOL := FALSE;
dwOptions       : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

bIdentify: This specifies whether the identification is to be activated (TRUE) or deactivated (FALSE).

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy           : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in `udiErrorId`. Only valid if `bBusy` is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if `bBusy` is FALSE. See [Error codes](#) [▶ 69].

VAR_IN_OUT

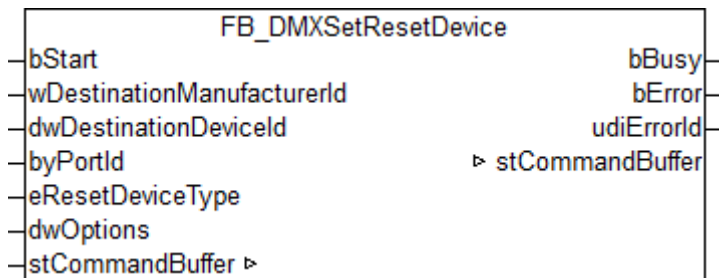
```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block.

Also see about this

■ [ST_DMXCommandBuffer](#) [▶ 64]

5.9 FB_DMXSetResetDevice



This function block activates a reset in a DMX device.

Applying a positive edge to the *bStart* input starts the block, and the *bBusy* output goes TRUE. The *wDestinationManufacturerId* and *dwDestinationDeviceId* inputs address the DMX device. The *byPortId* input defines the channel within the addressed DMX device. If the execution of the command has been completed, the *bBusy* output goes back to FALSE. The *bError* and *udiErrorId* outputs can now be processed. Further positive edges at the *bStart* input will be ignored as long as the block is active (*bBusy* is TRUE).

VAR_INPUT

```
bStart           : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId        : BYTE;
eResetDeviceType : E_DMXResetDeviceType := eDMXResetDeviceTypeWarm;
dwOptions       : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

eResetDeviceType: This specifies whether a warm start (*eDMXResetDeviceTypeWarm*) or a cold start (*eDMXResetDeviceTypeCold*) is to be performed. No other values are possible for this input.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy           : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block.

Also see about this

📖 [E_DMXResetDeviceType](#) [▶ 61]

ST_DMXCommandBuffer [▶ 64]

5.10 FB_DMXDiscMute



This function block sets the mute flag of a DMX device. The mute flag specifies whether a DMX device reacts to the `FB_DMXDiscUniqueBranch()` [▶ 29] command (mute flag is not set) or not (mute flag is set).

Applying a positive edge to the `bStart` input starts the block, and the `bBusy` output goes TRUE. The `wDestinationManufacturerId` and `dwDestinationDeviceId` inputs address the DMX device. The `byPortId` input defines the channel within the addressed DMX device. If the execution of the command has been completed, the `bBusy` output goes back to FALSE. The `bError`, `udiErrorId` and `wControlField` outputs can now be processed. Further positive edges at the `bStart` input will be ignored as long as the block is active (`bBusy` is TRUE).

VAR_INPUT

```
bStart           : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId        : BYTE;
dwOptions       : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy           : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
wControlField   : WORD;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in `udiErrorId`. Only valid if `bBusy` is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if `bBusy` is FALSE. See [Error codes](#) [▶ 69].

wControlField: If the execution of the command has been completed (`bBusy` is FALSE), then further information about the DMX device will be output at this output. The meaning of the individual bits is defined as follows:

Bit	Description
0 - Managed Proxy Flag	This bit is set if the DMX device is a proxy device.
1 - Sub-Device Flag	This bit is set if the DMX device supports sub-devices.

Bit	Description
2 - Boot-Loader Flag	This bit is set if the DMX device cannot receive any commands (e.g. whilst the firmware is being loaded).
3 - Proxied Device Flag	This bit is set if the response was transmitted by a proxy device.
4 - 15	Reserve (always 0).

VAR_IN_OUT

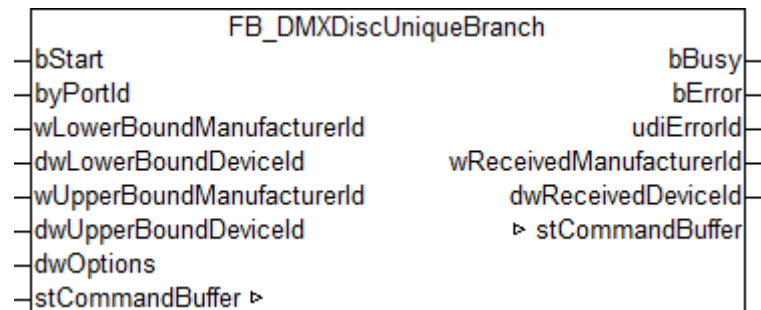
```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [► 22] block.

Also see about this

ST_DMXCommandBuffer [► 64]

5.11 FB_DMXDiscUniqueBranch



This function block queries whether DMX devices are located within a certain address range. This command is used for the discovery of the connected DMX devices.

Applying a positive edge to the *bStart* input starts the block, and the *bBusy* output goes TRUE. The *wLowerBoundManufacturerId*, *dwLowerBoundDeviceId*, *wUpperBoundManufacturerId* and *dwUpperBoundDeviceId* inputs define the address range which is searched for DMX devices. The *byPortId* input defines the channel within the addressed DMX device. If the execution of the command has been completed, the *bBusy* output goes back to FALSE. The *bError*, *udiErrorId*, *wReceivedManufacturerId* and *dwReceivedDeviceId* outputs can now be evaluated. Further positive edges at the *bStart* input will be ignored as long as the block is active (*bBusy* is TRUE).

If there is only one DMX device in the defined address range, then the 48-bit UID of the DMX device will be returned via the *wReceivedManufacturerId* and *dwReceivedDeviceId* outputs. If there are no DMX devices in this range, then the *bError* output is TRUE and *udiErrorId* is 0x8002 (no reply from the DMX device). If there are two or more DMX devices in the address range, then *bError* is TRUE and *udiErrorId* contains a 0x8006 (checksum error).

VAR_INPUT

```
bStart : BOOL;
byPortId : BYTE;
wLowerBoundManufacturerId : WORD;
dwLowerBoundDeviceId : DWORD;
wUpperBoundManufacturerId : WORD;
dwUpperBoundDeviceId : DWORD;
dwOptions : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

wLowerBoundManufacturerId: Unique manufacturer ID from the lower address range.

dwLowerBoundDeviceId: Unique device ID from the lower address range.

wUpperBoundManufacturerId: Unique manufacturer ID from the upper address range.

dwUpperBoundDeviceId: Unique device ID from the upper address range.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy          : BOOL;
bError         : BOOL;
udiErrorId    : UDINT;
wReceivedManufacturerId : WORD;
dwReceivedDeviceId : DWORD;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

wReceivedManufacturerId: If the execution of the command has been completed (*bBusy* is FALSE), then the state of identification of the DMX device is displayed at this output.

dwReceivedDeviceId: If the execution of the command has been completed (*bBusy* is FALSE), then the state of identification of the DMX device is displayed at this output.

VAR_IN_OUT

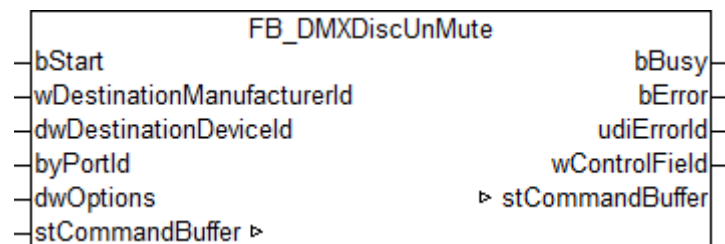
```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block.

Also see about this

ST_DMXCommandBuffer [▶ 64]

5.12 FB_DMXDiscUnMute



This function block resets the mute flag of a DMX device. The mute flag specifies whether a DMX device reacts to the [FB_DMXDiscUniqueBranch\(\)](#) [▶ 29] command (mute flag is not set) or not (mute flag is set).

Applying a positive edge to the *bStart* input starts the block, and the *bBusy* output goes TRUE. The *wDestinationManufacturerId* and *dwDestinationDeviceId* inputs address the DMX device. The *byPortId* input defines the channel within the addressed DMX device. If the execution of the command has been completed, the *bBusy* output goes back to FALSE. The *bError*, *udiErrorId* and *wControlField* outputs can now be processed. Further positive edges at the *bStart* input will be ignored as long as the block is active (*bBusy* is TRUE).

VAR_INPUT

```
bStart          : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId       : BYTE;
dwOptions      : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy          : BOOL;
bError         : BOOL;
udiErrorId     : UDINT;
wControlField  : WORD;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes \[▶ 69\]](#).

wControlField: If the execution of the command has been completed (*bBusy* is FALSE), then further information about the DMX device will be output at this output. The meaning of the individual bits is defined as follows:

Bit	Description
0 - Managed Proxy Flag	This bit is set if the DMX device is a proxy device.
1 - Sub-Device Flag	This bit is set if the DMX device supports sub-devices.
2 - Boot-Loader Flag	This bit is set if the DMX device cannot receive any commands (e.g. whilst the firmware is being loaded).
3 - Proxied Device Flag	This bit is set if the response was transmitted by a proxy device.
4 - 15	Reserve (always 0).

VAR_IN_OUT

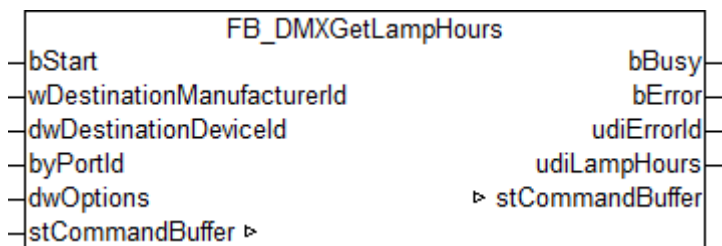
```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\) \[▶ 22\]](#) block.

Also see about this

- ST_DMXCommandBuffer [▶ 64]

5.13 FB_DMXGetLampHours



This function block reads the number of hours in which the lamp was on. The block [FB_DMXSetLampHours\(\)](#) [► 34] can be used to edit the hour counter.

VAR_INPUT

```
bStart           : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId        : BYTE;
dwOptions       : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy           : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
udiLampHours    : UDINT;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [► 69].

udiLampHours: Number of hours in which the lamp was switched on.

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [► 22] block.

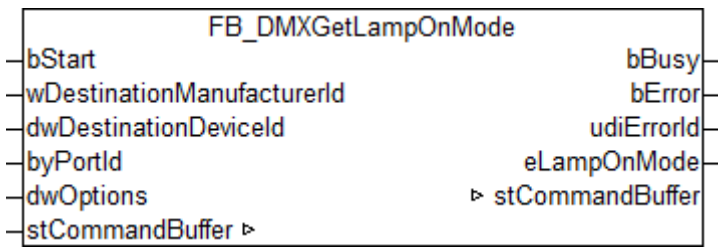
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2251	PC/CX	TcDMX-library higher than V1.2.0

Also see about this

📖 [ST_DMXCommandBuffer](#) [► 64]

5.14 FB_DMXGetLampOnMode



This function block reads the parameter that defines the switch-on characteristics of the DMX device. The block `FB_DMXSetLampOnMode()` [► 35] can be used to edit the value.

VAR_INPUT

```
bStart           : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId        : BYTE;
dwOptions       : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy           : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
eLampOnMode     : E_DMXLampOnMode;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [► 69].

eLampOnMode: Contains the current parameter that defines the switch-on characteristics of the DMX device.

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```


stCommandBuffer: A reference to the structure for communication (the buffer) with the `FB_EL6851CommunicationEx()` [► 22] block.

Requirements

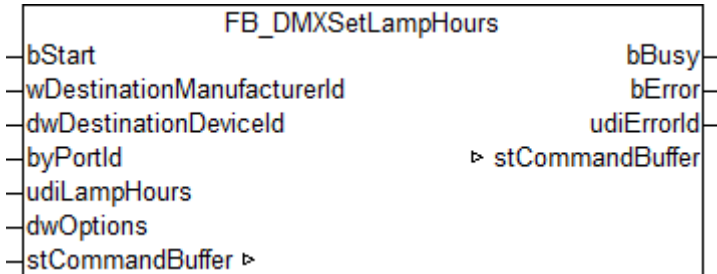
Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2251	PC/CX	TcDMX-library higher than V1.2.0

Also see about this

 E_DMXLampOnMode [[▶ 58](#)]

 ST_DMXCommandBuffer [[▶ 64](#)]

5.15 FB_DMXSetLampHours



This function block sets the operating hours counter for the lamp. The block [FB_DMXGetLampHours\(\) \[\[▶ 32\]\(#\)\]](#) can be used to read the counter.

VAR_INPUT

```
bStart           : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId        : BYTE;
udiLampHours    : UDINT := 0;
dwOptions       : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

udiLampHours: New value for the operating hours counter.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy           : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes \[\[▶ 69\]\(#\)\]](#).

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\) \[\[▶ 22\]\(#\)\]](#) block.

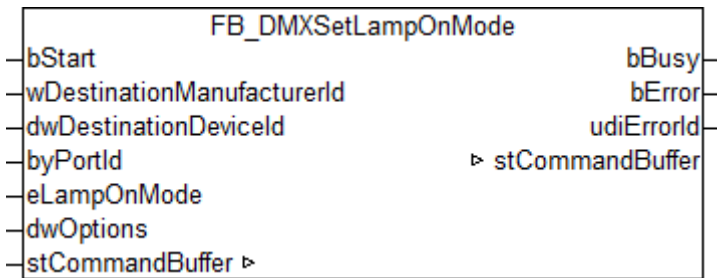
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2251	PC/CX	TcDMX-library higher than V1.2.0

Also see about this

ST_DMXCommandBuffer [▶ 64]

5.16 FB_DMXSetLampOnMode



This function block defines the switch-on characteristics of the DMX device. The block `FB_DMXGetLampOnMode()` [▶ 33] can be used to read the set value.

VAR_INPUT

```
bStart : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId : BYTE;
eLampOnMode : E_DMXLampOnMode := eDMXLampOnModeOff;
dwOptions : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

eLampOnMode: This parameter defines the switch-on characteristics of the DMX device.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy : BOOL;
bError : BOOL;
udiErrorId : UDINT;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the `FB_EL6851CommunicationEx()` [► 22] block.

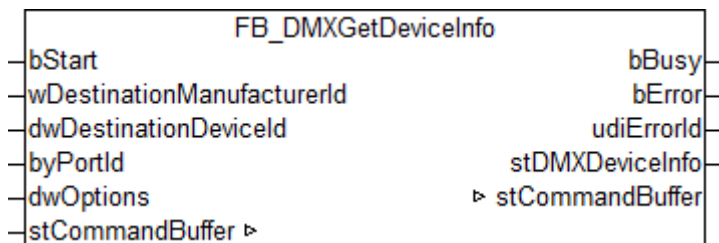
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2251	PC/CX	TcDMX-library higher than V1.2.0

Also see about this

- 📄 E_DMXLampOnMode [► 58]
- 📄 ST_DMXCommandBuffer [► 64]

5.17 FB_DMXGetDeviceInfo



This function block queries all relevant information from a DMX device.

Applying a positive edge to the *bStart* input starts the block, and the *bBusy* output goes TRUE. The *wDestinationManufacturerId* and *dwDestinationDeviceId* inputs address the DMX device. The *byPortId* input defines the channel within the addressed DMX device. If the execution of the command has been completed, the *bBusy* output goes back to FALSE. The *bError*, *udiErrorId* and *stDMXDeviceInfo* outputs can now be processed. Further positive edges at the *bStart* input will be ignored if the block is active (*bBusy* is TRUE).

VAR_INPUT

```
bStart           : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId         : BYTE;
dwOptions        : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy           : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
stDMXDeviceInfo : ST_DMXDeviceInfo;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

stDMXDeviceInfo: If the execution of the command has been completed (*bBusy* is FALSE), then all relevant information for the DMX device is sent to this output in a structure. See also the description of the [ST_DMXDeviceInfo](#) [▶ 65] structure.

uiUID: Unique 48-bit address of the DMX device. The lower 32 bits define the device ID and the upper 16 bits the manufacturer ID.

tRDMProtocolVersion: Version number of the supported RDM standard.

uiDeviceModelId: The device model ID is defined by the device manufacturer.

stProductCategory: Device type of the DMX device. The device types are defined in the RDM standard.

uiSoftwareVersionId: Version number of the software (firmware) in the DMX device.

uiDMX512Footprint: Number of DMX512 slots occupied by the device. Each sub-device and the root device occupy their own DMX512 slots.

stDMX512Personality: DMX devices can possess various *personalities*. *Personalities* are comparable with profiles. The current profile and the number of profiles are displayed in this element.

uiDMX512StartAddress: The DMX512 start address. This lies within the range from 1 – 512. If the *uiDMX512Footprint* parameter is 0, then the DMX start address is 0xFFFF (65535). Each sub-device and the root device occupy different DMX512 start addresses.

uiSubDeviceCount: Number of sub-devices.

bySensorCount: The number of sensors contained in the sub-device or the root device.

VAR_IN_OUT

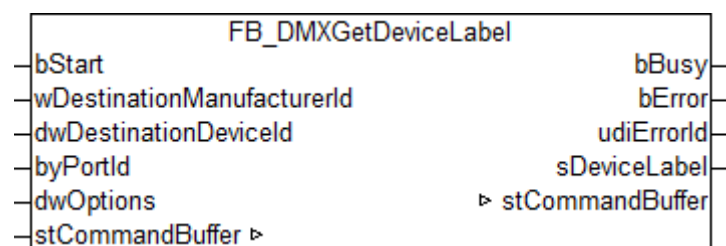
```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block.

Also see about this

- ST_DMXCommandBuffer [▶ 64]

5.18 FB_DMXGetDeviceLabel



This function block reads a text from the DMX device, which contains a more detailed description of the device. The block [FB_DMXSetDeviceLabel\(\)](#) [▶ 43] can be used to edit the text.

VAR_INPUT

```
bStart : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId : BYTE;
dwOptions : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy          : BOOL;
bError         : BOOL;
udiErrorId    : UDINT;
sDeviceLabel  : STRING;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [► 69].

sDeviceLabel: Description text for the DMX device.

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [► 22] block.

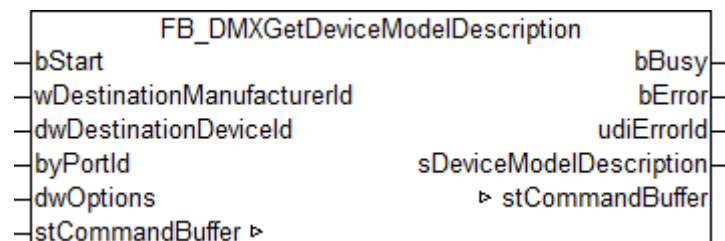
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2251	PC/CX	TcDMX-library higher than V1.2.0

Also see about this

📖 [ST_DMXCommandBuffer](#) [► 64]

5.19 FB_DMXGetDeviceModelDescription



This function block queries the description of the device type.

Applying a positive edge to the *bStart* input starts the block, and the *bBusy* output goes TRUE. The *wDestinationManufacturerId* and *dwDestinationDeviceId* inputs address the DMX device. The *byPortId* input defines the channel within the addressed DMX device. If the execution of the command has been completed, the *bBusy* output goes back to FALSE. The *bError*, *udiErrorId* and *sDeviceModelDescription* outputs can now be processed. Further positive edges at the *bStart* input will be ignored as long as the block is active (*bBusy* is TRUE).

VAR_INPUT

```
bStart          : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId       : BYTE;
dwOptions      : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy          : BOOL;
bError         : BOOL;
udiErrorId     : UDINT;
sDeviceModelDescription : STRING;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes \[▶ 69\]](#).

sDeviceModelDescription: If the execution of the command has been completed (*bBusy* is FALSE), then a description (maximum 32 characters) of the device type will be sent to this output. The contents are specified by the DMX device manufacturer.

VAR_IN_OUT

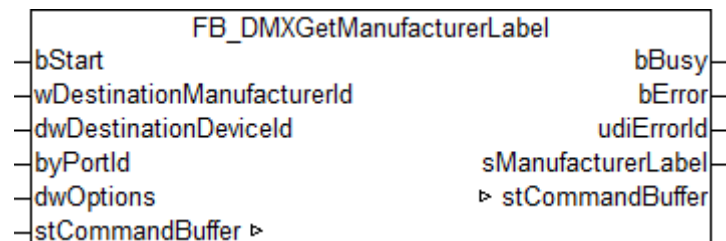
```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\) \[▶ 22\]](#) block.

Also see about this

ST_DMXCommandBuffer [▶ 64]

5.20 FB_DMXGetManufacturerLabel



This function block queries the description of the DMX device manufacturer.

Applying a positive edge to the *bStart* input starts the block, and the *bBusy* output goes TRUE. The *wDestinationManufacturerId* and *dwDestinationDeviceId* inputs address the DMX device. The *byPortId* input defines the channel within the addressed DMX device. If the execution of the command has been completed,

the *bBusy* output goes back to FALSE. The *bError*, *udiErrorId* and *sManufacturerLabel* outputs can now be processed. Further positive edges at the *bStart* input will be ignored as long as the block is active (*bBusy* is TRUE).

VAR_INPUT

```
bStart          : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId       : BYTE;
dwOptions      : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy          : BOOL;
bError         : BOOL;
udiErrorId     : UDINT;
sManufacturerLabel : STRING;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

sManufacturerLabel: If the execution of the command has been completed (*bBusy* is FALSE), then a description (maximum 32 characters) of the DMX device type will be sent to this output. The contents are specified by the DMX device manufacturer.

VAR_IN_OUT

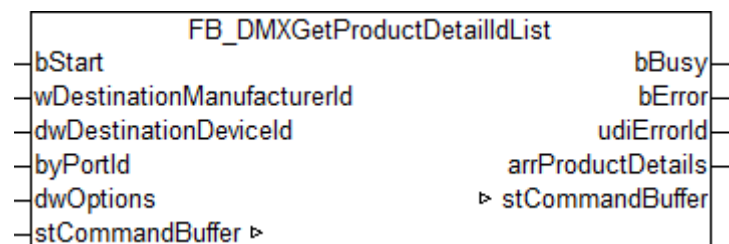
```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block.

Also see about this

ST_DMXCommandBuffer [▶ 64]

5.21 FB_DMXGetProductDetailIdList



This function block queries the categories to which the DMX device belongs.

RDM defines different device categories. Each DMX device can be assigned to up to 6 categories. The assignment is done by the device manufacturer and cannot be changed via RDM.

VAR_INPUT

```
bStart          : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId       : BYTE;
dwOptions      : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy          : BOOL;
bError         : BOOL;
udiErrorId     : UDINT;
arrProductDetails : ARRAY[1..6] OF E_DMXProductDetail;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

arrProductDetails: Contains the list with up to 6 device categories.

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block.

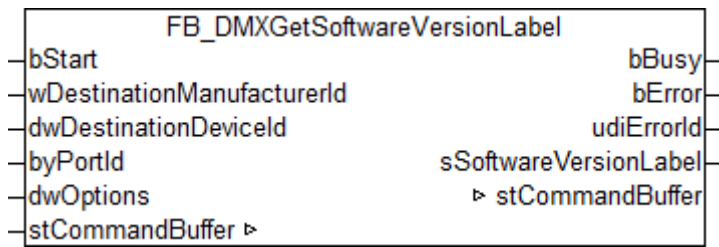
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2251	PC/CX	TcDMX-library higher than V1.2.0

Also see about this

- 📖 [E_DMXProductDetail](#) [▶ 59]
- 📖 [ST_DMXCommandBuffer](#) [▶ 64]

5.22 FB_DMXGetSoftwareVersionLabel



This function block queries the description of the software version of the DMX device.

Applying a positive edge to the *bStart* input starts the block, and the *bBusy* output goes TRUE. The *wDestinationManufacturerId* and *dwDestinationDeviceId* inputs address the DMX device. The *byPortId* input defines the channel within the addressed DMX device. If the execution of the command has been completed, the *bBusy* output goes back to FALSE. The *bError*, *udiErrorId* and *sSoftwareVersionLabel* outputs can now be processed. Further positive edges at the *bStart* input will be ignored as long as the block is active (*bBusy* is TRUE).

VAR_INPUT

```
bStart           : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId        : BYTE;
dwOptions       : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy           : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
sSoftwareVersionLabel : STRING;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

sSoftwareVersionLabel: If the execution of the command has been completed (*bBusy* is FALSE), then a description (maximum 32 characters) of the software version of the DMX device will be output at this output. The contents are specified by the DMX device manufacturer.

VAR_IN_OUT

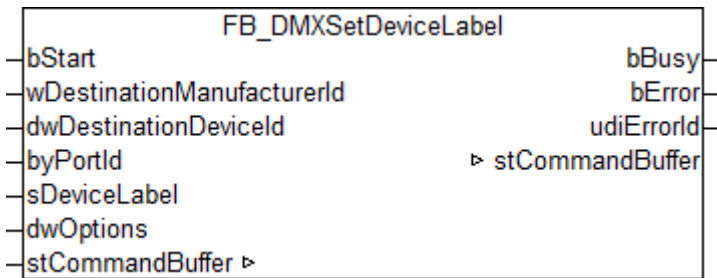
```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block.

Also see about this

ST_DMXCommandBuffer [▶ 64]

5.23 FB_DMXSetDeviceLabel



This function block writes a description text into the DMX device. The block [FB_DMXGetDeviceLabel\(\) \[▶ 37\]](#) can be used to read the text.

VAR_INPUT

```
bStart           : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId        : BYTE;
sDeviceLabel    : STRING := '';
dwOptions       : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

sDeviceLabel: New description text for the DMX device.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy           : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes \[▶ 69\]](#).

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\) \[▶ 22\]](#) block.

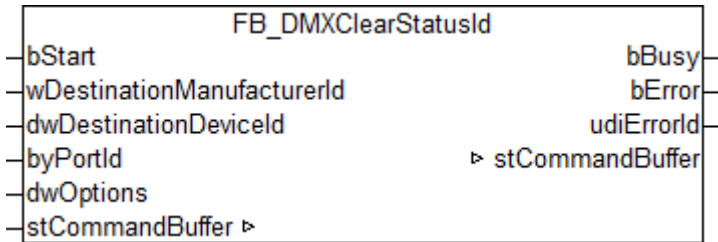
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2251	PC/CX	TcDMX-library higher than V1.2.0

Also see about this

📖 ST_DMXCommandBuffer [▶ 64]

5.24 FB_DMXClearStatusId



This function block clears the message buffer in the DMX device.

VAR_INPUT

```

bStart          : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId       : BYTE;
dwOptions      : DWORD := 0;
  
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

dwOptions: Options (currently not used).

VAR_OUTPUT

```

bBusy          : BOOL;
bError         : BOOL;
udiErrorId     : UDINT;
  
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes \[▶ 69\]](#).

VAR_IN_OUT

```

stCommandBuffer : ST_DMXCommandBuffer;
  
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\) \[▶ 22\]](#) block.

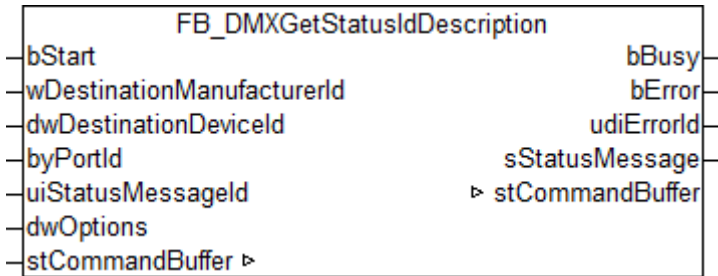
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2251	PC/CX	TcDMX-library higher than V1.2.0

Also see about this

▢ ST_DMXCommandBuffer [▶ 64]

5.25 FB_DMXGetStatusIdDescription



This function block reads the text of a certain status ID from the DMX device.

RDM defines some standard messages. Each of these messages has a unique status ID. This block can be used to read the corresponding text from the DMX device.

VAR_INPUT

```
bStart          : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId       : BYTE;
uiStatusMessageId : UINT := 1;
dwOptions      : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

uiStatusMessageId: Status ID for which the text is to be read.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy          : BOOL;
bError         : BOOL;
udiErrorId     : UDINT;
sStatusMessage : STRING;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes \[▶ 69\]](#).

sStatusMessage: Status message.

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the FB_EL6851CommunicationEx() [▶ 22] block.

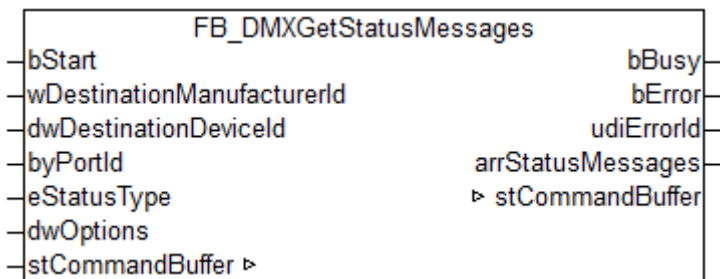
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2251	PC/CX	TcDMX-library higher than V1.2.0

Also see about this

ST_DMXCommandBuffer [▶ 64]

5.26 FB_DMXGetStatusMessages



This function block collects status or error information from a DMX device.

VAR_INPUT

```
bStart : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId : BYTE;
eStatusType : E_DMXStatusType := eDMXStatusTypeNone;
dwOptions : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer Id of the DMX device.

dwDestinationDeviceId: Unique device Id of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the port Id. The root device always has the port Id 0.

eStatusType: Status type.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy : BOOL;
bError : BOOL;
udiErrorId : UDINT;
arrStatusMessages : ARRAY[0..24] OF ST_DMXStatusMessage;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

arrStatusMessages: If the execution of the command has been completed (*bBusy* is FALSE), then all status/error information are sent to this output in an array.

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block.

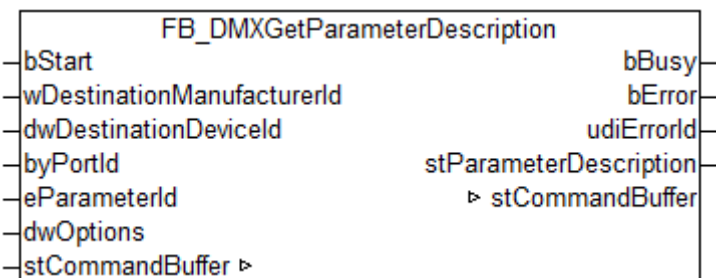
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

Also see about this

- ▣ [E_DMXStatusType](#) [▶ 64]
- ▣ [ST_DMXStatusMessage](#) [▶ 68]
- ▣ [ST_DMXCommandBuffer](#) [▶ 64]

5.27 FB_DMXGetParameterDescription



This function block retrieves the definition of manufacturer-specific PIDs.

VAR_INPUT

```
bStart : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId : BYTE;
eParameterId : E_DMXParameterId;
dwOptions : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer Id of the DMX device.

dwDestinationDeviceId: Unique device Id of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the port Id. The root device always has the port Id 0.

eParameterId: The requested manufacturer specific PID.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy          : BOOL;
bError         : BOOL;
udiErrorId     : UDINT;
stParameterDescription : ST_DMXParameterDescription;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

stParameterDescription: If the execution of the command has been completed (*bBusy* is FALSE), then information about the PID is sent to this output.

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block.

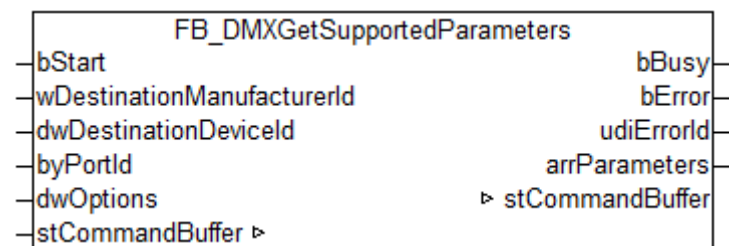
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

Also see about this

- ▶ E_DMXParameterId [▶ 58]
- ▶ ST_DMXParameterDescription [▶ 66]
- ▶ ST_DMXCommandBuffer [▶ 64]

5.28 FB_DMXGetSupportedParameters



This function block queries all supported parameters from a DMX device.

VAR_INPUT

```
bStart          : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId       : BYTE;
dwOptions      : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer Id of the DMX device.

dwDestinationDeviceId: Unique device Id of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the port Id. The root device always has the port Id 0.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy      : BOOL;
bError     : BOOL;
udiErrorId : UDINT;
arrParameters : ARRAY[0..114] OF E_DMXParameterId;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

arrParameters: If the execution of the command has been completed (*bBusy* is FALSE), then all supported parameters for the DMX device is sent to this output.

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block.

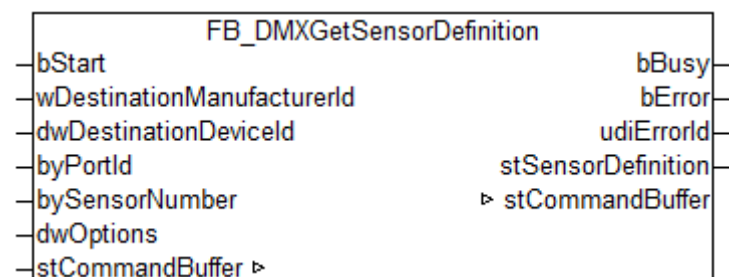
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

Also see about this

- ▶ E_DMXParameterId [▶ 58]
- ▶ ST_DMXCommandBuffer [▶ 64]

5.29 FB_DMXGetSensorDefinition



This function block queries the definition of a specific sensor.

VAR_INPUT

```
bStart      : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId    : BYTE;
bySensorNumber : BYTE := 0;
dwOptions   : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer Id of the DMX device.

dwDestinationDeviceId: Unique device Id of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the port Id. The root device always has the port Id 0.

bySensorNumber: DMX512 sensor number (0 - 254).

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy          : BOOL;
bError         : BOOL;
udiErrorId     : UDINT;
stSensorDefinition: ST_DMXSensorDefinition;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

stSensorDefinition: If the execution of the command has been completed (*bBusy* is FALSE), then the definition of the sensor will be sent to this output.

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block.

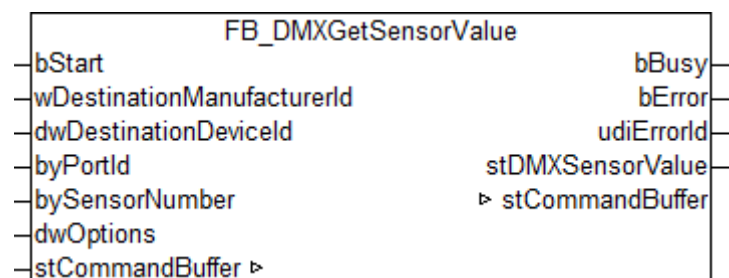
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

Also see about this

- ▣ [ST_DMXSensorDefinition](#) [▶ 67]
- ▣ [ST_DMXCommandBuffer](#) [▶ 64]

5.30 FB_DMXGetSensorValue



This function block queries the current value of a sensor.

VAR_INPUT

```
bStart : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId : BYTE;
bySensorNumber : BYTE := 0;
dwOptions : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

bySensorNumber: Number of the sensor whose values are queried.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy : BOOL;
bError : BOOL;
udiErrorId : UDINT;
stDMXSensorValue : ST_DMXSensorValue;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

stDMXSensorValue: Structure with information about the current state of the sensor.

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block.

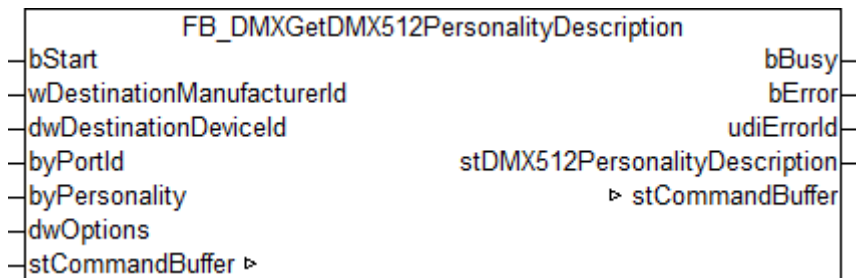
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2251	PC/CX	TcDMX-library higher than V1.2.0

Also see about this

- 📄 [ST_DMXSensorValue](#) [▶ 67]
- 📄 [ST_DMXCommandBuffer](#) [▶ 64]

5.31 FB_DMXGetDMX512PersonalityDescription



This function block reads further *Personality* information from the DMX device. Some DMX devices support so-called *Personalities*. Changing the *Personality* can influence certain RDM parameters.

VAR_INPUT

```
bStart           : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId        : BYTE;
byPersonality   : BYTE := 0;
dwOptions       : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

byPersonality: The *Personality* for which information is queried.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy           : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
stDMX512PersonalityDescription : ST_DMX512PersonalityDescription;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

stDMX512PersonalityDescription: Structure with information about the *Personality*.

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block.

Requirements

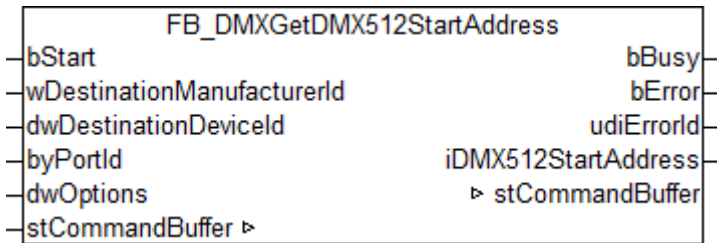
Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2251	PC/CX	TcDMX-library higher than V1.2.0

Also see about this

ST_DM512PersonalityDescription [▶ 64]

ST_DM512CommandBuffer [▶ 64]

5.32 FB_DM512GetDM512StartAddress



This function block queries the DM512 start address. This lies within the range from 1 – 512. If the DM512 device does not occupy any DM512 slot, then the DM512 start address is 0xFFFF (65535). Each sub-device and the root device occupy different DM512 start addresses.

Applying a positive edge to the *bStart* input starts the block, and the *bBusy* output goes TRUE. The *wDestinationManufacturerId* and *dwDestinationDeviceId* inputs address the DM512 device. The *byPortId* input defines the channel within the addressed DM512 device. If the execution of the command has been completed, the *bBusy* output goes back to FALSE. The *bError*, *udiErrorId* and *iDM512StartAddress* outputs can now be processed. Further positive edges at the *bStart* input will be ignored as long as the block is active (*bBusy* is TRUE).

VAR_INPUT

```

bStart           : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId        : BYTE;
dwOptions       : DWORD := 0;
  
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DM512 device.

dwDestinationDeviceId: Unique device ID of the DM512 device.

byPortId: Channel within the addressed DM512 device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

dwOptions: Options (currently not used).

VAR_OUTPUT

```

bBusy           : BOOL;
bError          : BOOL;
udiErrorId      : UDINT;
iDM512StartAddress : INT;
  
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes \[▶ 69\]](#).

iDM512StartAddress: If the execution of the command has been completed (*bBusy* is FALSE), then the DM512 start address of the DM512 device will be sent to this output.

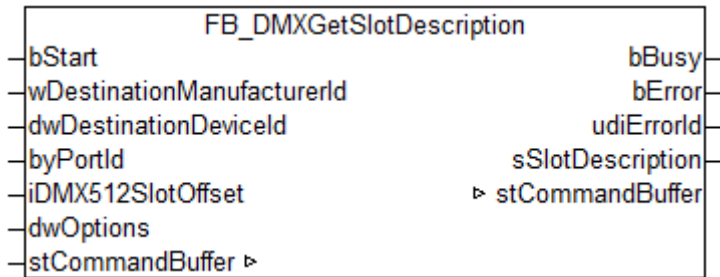
VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the `FB_EL6851CommunicationEx()` [► 22] block.

Also see about this

▣ `ST_DMXCommandBuffer` [► 64]

5.33 FB_DMXGetSlotDescription

This function block queries the text description for slot offsets.

VAR_INPUT

```
bStart : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId : BYTE;
iDMX512SlotOffset : INT := 0;
dwOptions : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer Id of the DMX device.

dwDestinationDeviceId: Unique device Id of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the port Id. The root device always has the port Id 0.

iDMX512SlotOffset: DMX512 slot offset (0 - 511).

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy : BOOL;
bError : BOOL;
udiErrorId : UDINT;
sSlotDescription : STRING;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [► 69].

sSlotDescription: If the execution of the command has been completed (*bBusy* is FALSE), then the description (maximum 32 characters) of the slot will be sent to this output.

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the FB_EL6851CommunicationEx() [▶ 22] block.

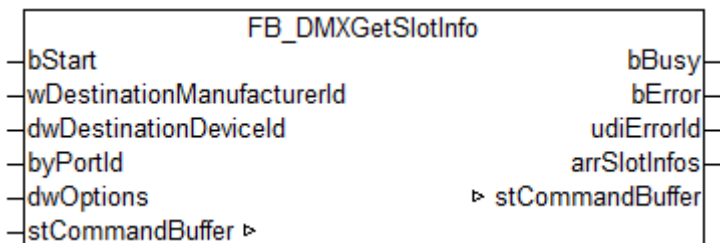
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

Also see about this

ST_DMXCommandBuffer [▶ 64]

5.34 FB_DMXGetSlotInfo



This function block queries basic information about the functionality of the DMX512 slots from a DMX device.

VAR_INPUT

```
bStart : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId : BYTE;
dwOptions : DWORD := 0;
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer Id of the DMX device.

dwDestinationDeviceId: Unique device Id of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the port Id. The root device always has the port Id 0.

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy : BOOL;
bError : BOOL;
udiErrorId : UDINT;
arrSlotInfos : ARRAY[0..45] OF ST_DMXSlotInfo;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See Error codes [▶ 69].

arrSlotInfos: If the execution of the command has been completed (*bBusy* is FALSE), then all relevant information for the DMX512 slots is sent to this output in a array.

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the `FB_EL6851CommunicationEx()` [► 22] block.

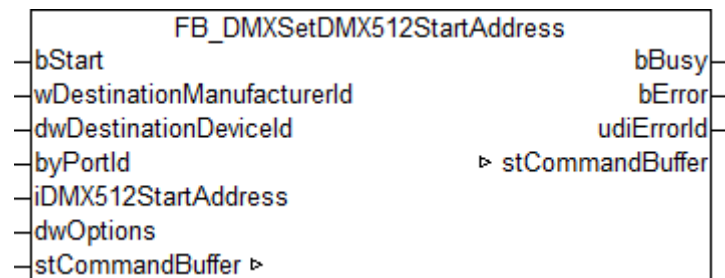
Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

Also see about this

- ▣ ST_DMXSlotInfo [► 67]
- ▣ ST_DMXCommandBuffer [► 64]

5.35 FB_DMXSetDMX512StartAddress



This function block sets the DMX512 start address. This lies within the range from 1 – 512. Each sub-device and the root device occupy different DMX512 start addresses.

Applying a positive edge to the *bStart* input starts the block, and the *bBusy* output goes TRUE. The *wDestinationManufacturerId* and *dwDestinationDeviceId* inputs address the DMX device. The *byPortId* input defines the channel within the addressed DMX device. If the execution of the command has been completed, the *bBusy* output goes back to FALSE. The *bError* and *udiErrorId* outputs can now be processed. Further positive edges at the *bStart* input will be ignored if the block is active (*bBusy* is TRUE).

VAR_INPUT

```

bStart : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId : BYTE;
iDMX512Startadresse : INT := 1;
dwOptions : DWORD := 0;
  
```

bStart: The command is started by a positive edge at this input.

wDestinationManufacturerId: Unique manufacturer ID of the DMX device.

dwDestinationDeviceId: Unique device ID of the DMX device.

byPortId: Channel within the addressed DMX device. Sub-devices are addressed through the Port ID. The root device always has the Port ID 0.

iDMX512StartAddress: DMX512 start address (1 to 152)

dwOptions: Options (currently not used).

VAR_OUTPUT

```
bBusy      : BOOL;
bError     : BOOL;
udiErrorId : UDINT;
```

bBusy: When the block is activated the output is set, and it remains active until execution of the command has been completed.

bError: This output is switched to TRUE as soon as an error occurs during the execution of a command. The command-specific error code is contained in *udiErrorId*. Only valid if *bBusy* is FALSE.

udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if *bBusy* is FALSE. See [Error codes](#) [▶ 69].

VAR_IN_OUT

```
stCommandBuffer : ST_DMXCommandBuffer;
```

stCommandBuffer: A reference to the structure for communication (the buffer) with the [FB_EL6851CommunicationEx\(\)](#) [▶ 22] block.

Also see about this

ST_DMXCommandBuffer [▶ 64]

5.36 Data types

5.36.1 E_DMXCommandClass

```
TYPE E_DMXCommandClass : (* Table A-1 *)
(
  eDMXCommandClassNotDefined      := 16#0000,      (* command class is not defined *)
  eDMXCommandClassDiscoveryCommand := 16#0010,
  eDMXCommandClassDiscoveryCommandResponse := 16#0011,
  eDMXCommandClassGetCommand      := 16#0020,
  eDMXCommandClassGetCommandResponse := 16#0021,
  eDMXCommandClassSetCommand      := 16#0030,
  eDMXCommandClassSetCommandResponse := 16#0031
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

5.36.2 E_DMXDataType

```
TYPE E_DMXDataType : (* Table A-15 *)
(
  eDMXDataTypeNotDefined      := 16#0000,      (* Data type is not defined *)
  eDMXDataTypeBitField        := 16#0001,      (* Data is bit packed *)
  eDMXDataTypeASCII           := 16#0002,      (* Data is a string *)
  eDMXDataTypeUnsignedByte    := 16#0003,      (* Data is an array of unsigned bytes *)
  eDMXDataTypeSignedByte     := 16#0004,      (* Data is an array of signed bytes *)
  eDMXDataTypeUnsignedWord    := 16#0005,      (* Data is an array of unsigned 16-bit words *)
  eDMXDataTypeSignedWord     := 16#0006,      (* Data is an array of signed 16-bit words *)
  eDMXDataTypeUnsignedDWord   := 16#0007,      (* Data is an array of unsigned 32-bit words *)
  eDMXDataTypeSignedDWord    := 16#0008,      (* Data is an array of signed 32-bit words *)
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

5.36.3 E_DMXLampOnMode

```

TYPE E_DMXLampOnMode :
(
  eDMXLampOnModeOff      := 0,    (* Lamp stays off until directly instructed to strike *)
  eDMXLampOnModeDMX     := 1,    (* Lamp strikes upon receiving a DMX512 signal *)
  eDMXLampOnModeOn      := 2,    (* Lamp strikes automatically at power-up *)
  eDMXLampOnModeAfterCal := 3    (* Lamp strikes after calibration or homing procedure *)
);
END_TYPE

```

5.36.4 E_DMXParameterDescriptionCommandClass

```

TYPE E_DMXParameterDescriptionCommandClass : (* Table A-16 *)
(
  eDMXParameterDescriptionCommandClassGet      := 16#0001, (* PID supports GET only *)
  eDMXParameterDescriptionCommandClassSet     := 16#0002, (* PID supports SET only *)
  eDMXParameterDescriptionCommandClassGetSet := 16#0003 (* PID supports GET & SET *)
);
END_TYPE

```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

5.36.5 E_DMXParameterId

```

TYPE E_DMXParameterId : (* Table A-3 *)
(
  eDMXParameterIdNone := 16#0000,

  (* Network Management *)
  eDMXParameterIdDiscUniqueBranch := 16#0001,
  eDMXParameterIdDiscMute         := 16#0002,
  eDMXParameterIdDiscUnMute       := 16#0003,
  eDMXParameterIdProxiedDevices   := 16#0010,
  eDMXParameterIdProxiedDeviceCount := 16#0011,
  eDMXParameterIdCommsStatus      := 16#0015,

  (* Status Collection *)
  eDMXParameterIdQueuedMessage := 16#0020,
  eDMXParameterIdStatusMessages := 16#0030,
  eDMXParameterIdStatusIdDescription := 16#0031,
  eDMXParameterIdClearStatusId := 16#0032,
  eDMXParameterIdSubDeviceStatusReportThreshold := 16#0033,

  (* RDM Information *)
  eDMXParameterIdSupportedParamaters := 16#0050,
  eDMXParameterIdParameterDescription := 16#0051,

  (* Product Information *)
  eDMXParameterIdDeviceInfo := 16#0060,
  eDMXParameterIdProductDetailIdList := 16#0070,
  eDMXParameterIdDeviceModelDescription := 16#0080,
  eDMXParameterIdManufacturerLabel := 16#0081,
  eDMXParameterIdDeviceLabel := 16#0082,
  eDMXParameterIdFactoryDefaults := 16#0090,
  eDMXParameterIdLanguageCapabilities := 16#00A0,
  eDMXParameterIdLanguage := 16#00B0,
  eDMXParameterIdSoftwareVersionLabel := 16#00C0,
  eDMXParameterIdBootSoftwareVersionId := 16#00C1,

```

```

eDMXParameterIdBootSoftwareVersionLabel      := 16#00C2,

(* DMX512 Setup *)
eDMXParameterIdDMXPersonality                := 16#00E0,
eDMXParameterIdDMXPersonalityDescription    := 16#00E1,
eDMXParameterIdDMXStartAddress              := 16#00F0,
eDMXParameterIdSlotInfo                     := 16#0120,
eDMXParameterIdSlotDescription              := 16#0121,
eDMXParameterIdDefaultSlotValue             := 16#0122,

(* Sensors *)
eDMXParameterIdSensorDefinition              := 16#0200,
eDMXParameterIdSensorValue                  := 16#0201,
eDMXParameterIdRecordSensors                := 16#0202,

(* Power/Lamp Settings *)
eDMXParameterIdDeviceHours                  := 16#0400,
eDMXParameterIdLampHours                    := 16#0401,
eDMXParameterIdLampStrikes                  := 16#0402,
eDMXParameterIdLampState                    := 16#0403,
eDMXParameterIdLampOnMode                   := 16#0404,
eDMXParameterIdDevicePowerCycles            := 16#0405,

(* Display Settings *)
eDMXParameterIdDisplayInvert                := 16#0500,
eDMXParameterIdDisplayLevel                 := 16#0501,

(* Configuration *)
eDMXParameterIdPanInvert                    := 16#0600,
eDMXParameterIdTiltInvert                   := 16#0601,
eDMXParameterIdPanTiltSwap                  := 16#0602,
eDMXParameterIdRealTimeClock                := 16#0603,

(* Control *)
eDMXParameterIdIdentifyDevice                := 16#1000,
eDMXParameterIdResetDevice                  := 16#1001,
eDMXParameterIdPowerState                   := 16#1010,
eDMXParameterIdPerformSelftest              := 16#1020,

eDMXParameterIdSelfTestDescription           := 16#1021,
eDMXParameterIdCapturePreset                := 16#1030,
eDMXParameterIdPresetPlayBack               := 16#1031
);
END_TYPE

```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

5.36.6 E_DMXProductDetail

```

TYPE E_DMXProductDetail :
(
  eDMXProductDetailNotDeclared              := 16#0000,

  (* Generally applied to fixtures / 0x00xx *)
  eDMXProductDetailArc                      := 16#0001,
  eDMXProductDetailMetalHalide              := 16#0002,
  eDMXProductDetailIncandescent             := 16#0003,
  eDMXProductDetailLED                      := 16#0004,
  eDMXProductDetailFluorescent               := 16#0005,
  eDMXProductDetailColdcathode              := 16#0006,
  eDMXProductDetailElectroluminescent       := 16#0007,
  eDMXProductDetailLaser                     := 16#0008,
  eDMXProductDetailFlashtube                 := 16#0009,

  (* Generally applied to fixture accessories / 0x01xx *)
  eDMXProductDetailColorscroller             := 16#0100,
  eDMXProductDetailColorwheel                := 16#0101,
  eDMXProductDetailColorchange              := 16#0102,
  eDMXProductDetailIrisDouser                := 16#0103,
  eDMXProductDetailDimmingShutter            := 16#0104,
  eDMXProductDetailProfileShutter            := 16#0105,

```

```

eDMXProductDetailBarndoorShutter      := 16#0106,
eDMXProductDetailEffectsDisc         := 16#0107,
eDMXProductDetailGoboRotator         := 16#0108,

(* Generally applied to projectors / 0x02xx *)
eDMXProductDetailVideo               := 16#0200,
eDMXProductDetailSlide               := 16#0201,
eDMXProductDetailFilm                := 16#0202,
eDMXProductDetailOilwheel            := 16#0203,
eDMXProductDetailLCDGate              := 16#0204,

(* Generally applied to atmospheric effects / 0x03xx *)
eDMXProductDetailFoggerGlycol        := 16#0300,
eDMXProductDetailFoggerMineraloil    := 16#0301,
eDMXProductDetailFoggerWater         := 16#0302,
eDMXProductDetailCO2                 := 16#0303,
eDMXProductDetailLN2                 := 16#0304,
eDMXProductDetailBubble               := 16#0305,
eDMXProductDetailFlamePropane        := 16#0306,
eDMXProductDetailFlameOther          := 16#0307,
eDMXProductDetailOlefactoryStimulator := 16#0308,
eDMXProductDetailSnow                := 16#0309,
eDMXProductDetailWaterJet            := 16#030A,
eDMXProductDetailWind                := 16#030B,
eDMXProductDetailConfetti            := 16#030C,
eDMXProductDetailHazard               := 16#030D,

(* Generally applied to dimmers/power controllers / 0x04xx *)
eDMXProductDetailPhaseControl        := 16#0400,
eDMXProductDetailReversePhaseControl := 16#0401,
eDMXProductDetailSine                := 16#0402,
eDMXProductDetailPWM                 := 16#0403,
eDMXProductDetailDC                  := 16#0404,
eDMXProductDetailHfballast           := 16#0405,
eDMXProductDetailHfhvNeonballast     := 16#0406,
eDMXProductDetailHfhvEl              := 16#0407,
eDMXProductDetailMhrBallast          := 16#0408,
eDMXProductDetailBitangleModulation  := 16#0409,
eDMXProductDetailFrequencyModulation := 16#040A,
eDMXProductDetailHighfrequency12V   := 16#040B,
eDMXProductDetailRelayMechanical     := 16#040C,
eDMXProductDetailRelayElectronic     := 16#040D,
eDMXProductDetailSwitchElectronic    := 16#040E,
eDMXProductDetailContactor           := 16#040F,

(* Generally applied to scenic drive / 0x05xx *)
eDMXProductDetailMirrorballRotator   := 16#0500,
eDMXProductDetailOtherRotator        := 16#0501,
eDMXProductDetailKabukiDrop          := 16#0502,
eDMXProductDetailCurtain             := 16#0503,
eDMXProductDetailLineset             := 16#0504,
eDMXProductDetailMotorControl        := 16#0505,
eDMXProductDetailDamperControl       := 16#0506,

(* Generally applied to data distribution / 0x06xx *)
eDMXProductDetailSplitter            := 16#0600,
eDMXProductDetailEthernetNode        := 16#0601,
eDMXProductDetailMerge               := 16#0602,
eDMXProductDetailDatapatch           := 16#0603,
eDMXProductDetailWirelessLink        := 16#0604,

(* Generally applied to data conversion and interfaces / 0x07xx *)
eDMXProductDetailProtocolConvertor    := 16#0701,
eDMXProductDetailAnalogDemultiplex    := 16#0702,
eDMXProductDetailAnalogMultiplex      := 16#0703,
eDMXProductDetailSwitchPanel          := 16#0704,

(* Generally applied to audio or video (AV) devices / 0x08xx *)
eDMXProductDetailRouter               := 16#0800,
eDMXProductDetailFader                := 16#0801,
eDMXProductDetailMixer                := 16#0802,

(* Generally applied to controllers, backup devices and test equipment / 0x09xx *)
eDMXProductDetailChangeoverManual     := 16#0900,
eDMXProductDetailChangeoverAuto       := 16#0901,
eDMXProductDetailTest                 := 16#0902,

(* Generally applied to any category / 0x0Axx *)
eDMXProductDetailGfiRcd               := 16#0A00,
eDMXProductDetailBattery               := 16#0A01,

```



```
eDMXProductDetailControllableBreaker := 16#0A02,

(* Manufacturer Specific Types / 0x8000 - 0xDFFF *)
eDMXProductDetailOther := 16#7FFF (* for use where the Manufacturer believes that no
of the defined details apply *)
);
END_TYPE
```

5.36.7 E_DMXResetDeviceType

```
TYPE E_DMXResetDeviceType :
(
  eDMXResetDeviceTypeWarm := 1,
  eDMXResetDeviceTypeCold := 255
);
END_TYPE
```

5.36.8 E_DMXSensorType

```
TYPE E_DMXSensorType : (* Table A-12 *)
(
  eDMXSensorTypeTemperature := 16#00,
  eDMXSensorTypeVoltage := 16#01,
  eDMXSensorTypeCurrent := 16#02,
  eDMXSensorTypeFrequency := 16#03,
  eDMXSensorTypeResistance := 16#04,
  eDMXSensorTypePower := 16#05,
  eDMXSensorTypeMass := 16#06,
  eDMXSensorTypeLength := 16#07,
  eDMXSensorTypeArea := 16#08,
  eDMXSensorTypeVolume := 16#09,
  eDMXSensorTypeDensity := 16#0A,
  eDMXSensorTypeVelocity := 16#0B,
  eDMXSensorTypeAcceleration := 16#0C,
  eDMXSensorTypeForce := 16#0D,
  eDMXSensorTypeEnergy := 16#0E,
  eDMXSensorTypePressure := 16#0F,
  eDMXSensorTypeTime := 16#10,
  eDMXSensorTypeAngle := 16#11,
  eDMXSensorTypePositionX := 16#12,
  eDMXSensorTypePositionY := 16#13,
  eDMXSensorTypePositionZ := 16#14,
  eDMXSensorTypeAngularVelocity := 16#15,
  eDMXSensorTypeLuminousIntensity := 16#16,
  eDMXSensorTypeLuminousFlux := 16#17,
  eDMXSensorTypeIlluminance := 16#18,
  eDMXSensorTypeChrominanceRed := 16#19,
  eDMXSensorTypeChrominanceGreen := 16#1A,
  eDMXSensorTypeChrominanceBlue := 16#1B,
  eDMXSensorTypeContacts := 16#1C,
  eDMXSensorTypeMemory := 16#1D,
  eDMXSensorTypeItems := 16#1E,
  eDMXSensorTypeHumidity := 16#1F,
  eDMXSensorTypeCounter16Bit := 16#20,
  eDMXSensorTypeOther := 16#7F
);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

5.36.9 E_DMXSensorUnit

```
TYPE E_DMXSensorUnit : (* Table A-13 *)
(
  eDMXSensorUnitNone := 16#00, (* CONTACTS *)
  eDMXSensorUnitCentigrade := 16#01, (* TEMPERATURE *)
  eDMXSensorUnitVoltsDC := 16#02, (* VOLTAGE *)
);
```

```

eDMXSensorUnitVoltsACPeak      := 16#03, (* VOLTAGE *)
eDMXSensorUnitVoltsACRms      := 16#04, (* VOLTAGE *)
eDMXSensorUnitAmpereDC        := 16#05, (* CURRENT *)
eDMXSensorUnitAmpereACPeak    := 16#06, (* CURRENT *)
eDMXSensorUnitAmpereACRms     := 16#07, (* CURRENT *)
eDMXSensorUnitHertz           := 16#08, (* FREQUENCY / ANG_VEL *)
eDMXSensorUnitOhm             := 16#09, (* RESISTANCE *)
eDMXSensorUnitWatt            := 16#0A, (* POWER *)
eDMXSensorUnitKilogram        := 16#0B, (* MASS *)
eDMXSensorUnitMeters          := 16#0C, (* LENGTH / POSITION *)
eDMXSensorUnitMetersSquared   := 16#0D, (* AREA *)
eDMXSensorUnitMetersCubed     := 16#0E, (* VOLUME *)
eDMXSensorUnitKilogrammesPerMeterCubed := 16#0F, (* DENSITY *)
eDMXSensorUnitMetersPerSecond := 16#10, (* VELOCITY *)
eDMXSensorUnitMetersPerSecondSquared := 16#11, (* ACCELERATION *)
eDMXSensorUnitNewton          := 16#12, (* FORCE *)
eDMXSensorUnitJoule           := 16#13, (* ENERGY *)
eDMXSensorUnitPascal          := 16#14, (* PRESSURE *)
eDMXSensorUnitSecond         := 16#15, (* TIME *)
eDMXSensorUnitDegree         := 16#16, (* ANGLE *)
eDMXSensorUnitSteradian       := 16#17, (* ANGLE *)
eDMXSensorUnitCandela         := 16#18, (* LUMINOUS_INTENSITY *)
eDMXSensorUnitLumen           := 16#19, (* LUMINOUS_FLUX *)
eDMXSensorUnitLux             := 16#1A, (* ILLUMINANCE *)
eDMXSensorUnitIre            := 16#1B, (* CHROMINANCE *)
eDMXSensorUnitByte           := 16#1C, (* MEMORY *)
);
END_TYPE

```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

5.36.10 E_DMXSensorUnitPrefix

```

TYPE E_DMXSensorUnitPrefix : (* Table A-14 *)
(
eDMXSensorUnitPrefixNone      := 16#00, (* Multiply by 1 *)
eDMXSensorUnitPrefixDeci      := 16#01, (* Multiply by 10^-1 *)
eDMXSensorUnitPrefixCenti     := 16#02, (* Multiply by 10^-2 *)
eDMXSensorUnitPrefixMilli     := 16#03, (* Multiply by 10^-3 *)
eDMXSensorUnitPrefixMicro     := 16#04, (* Multiply by 10^-6 *)
eDMXSensorUnitPrefixNano      := 16#05, (* Multiply by 10^-9 *)
eDMXSensorUnitPrefixPico      := 16#06, (* Multiply by 10^-12 *)
eDMXSensorUnitPrefixFempto     := 16#07, (* Multiply by 10^-15 *)
eDMXSensorUnitPrefixAtto      := 16#08, (* Multiply by 10^-18 *)
eDMXSensorUnitPrefixZepto     := 16#09, (* Multiply by 10^-21 *)
eDMXSensorUnitPrefixYocto     := 16#0A, (* Multiply by 10^-24 *)
eDMXSensorUnitPrefixDeca      := 16#11, (* Multiply by 10^1 *)
eDMXSensorUnitPrefixHecto     := 16#12, (* Multiply by 10^2 *)
eDMXSensorUnitPrefixKilo      := 16#13, (* Multiply by 10^3 *)
eDMXSensorUnitPrefixMega      := 16#14, (* Multiply by 10^6 *)
eDMXSensorUnitPrefixGiga      := 16#15, (* Multiply by 10^9 *)
eDMXSensorUnitPrefixTerra     := 16#16, (* Multiply by 10^12 *)
eDMXSensorUnitPrefixPeta      := 16#17, (* Multiply by 10^15 *)
eDMXSensorUnitPrefixExa       := 16#18, (* Multiply by 10^18 *)
eDMXSensorUnitPrefixZetta     := 16#19, (* Multiply by 10^21 *)
eDMXSensorUnitPrefixYotta     := 16#1A, (* Multiply by 10^24 *)
);
END_TYPE

```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

5.36.11 E_DMXXSlotDefinition

```

TYPE E_DMXXSlotDefinition : (* Table C-2 *)
(
  (* Intensity Functions / 0x00xx *)
  eDMXXSlotDefinitionIntensity      := 16#0001, (* Intensity *)
  eDMXXSlotDefinitionIntensityMaster := 16#0002, (* Intensity Master *)

  (* Movement Functions / 0x01xx *)
  eDMXXSlotDefinitionPan           := 16#0101, (* Pan *)
  eDMXXSlotDefinitionTilt          := 16#0102, (* Tilt *)

  (* Color Functions / 0x02xx *)
  eDMXXSlotDefinitionColorWheel    := 16#0201, (* Color Wheel *)
  eDMXXSlotDefinitionColorSubCyan   := 16#0202, (* Subtractive Color Mixer - Cyan/Blue *)
  eDMXXSlotDefinitionColorSubYellow := 16#0203, (* Subtractive Color Mixer - Yellow/Amber *)
  eDMXXSlotDefinitionColorSubMagenta := 16#0204, (* Subtractive Color Mixer - Magenta *)
  eDMXXSlotDefinitionColorAddRed    := 16#0205, (* Additive Color Mixer - Red *)
  eDMXXSlotDefinitionColorAddGreen  := 16#0206, (* Additive Color Mixer - Green *)
  eDMXXSlotDefinitionColorAddBlue   := 16#0207, (* Additive Color Mixer - Blue *)
  eDMXXSlotDefinitionColorCorrection := 16#0208, (* Color Temperature Correction *)
  eDMXXSlotDefinitionColorScroll    := 16#0209, (* Color Scroll *)
  eDMXXSlotDefinitionColorSemaphore := 16#0210, (* Color Semaphore *)

  (* Image Functions / 0x03xx *)
  eDMXXSlotDefinitionStaticGoboWheel := 16#0301, (* Static gobo wheel *)
  eDMXXSlotDefinitionRotoGoboWheel  := 16#0302, (* Rotating gobo wheel *)
  eDMXXSlotDefinitionPrismWheel     := 16#0303, (* Prism wheel *)
  eDMXXSlotDefinitionEffectsWheel   := 16#0304, (* Effects wheel *)

  (* Beam Functions / 0x04xx *)
  eDMXXSlotDefinitionBeamSizeIris    := 16#0401, (* Beam size iris *)
  eDMXXSlotDefinitionEdge            := 16#0402, (* Edge/Lens focus *)
  eDMXXSlotDefinitionFrost           := 16#0403, (* Frost/Diffusion *)
  eDMXXSlotDefinitionStrobe          := 16#0404, (* Strobe/Shutter *)
  eDMXXSlotDefinitionZoom            := 16#0405, (* Zoom lens *)
  eDMXXSlotDefinitionFramingShutter  := 16#0406, (* Framing shutter *)
  eDMXXSlotDefinitionShutterRotate   := 16#0407, (* Framing shutter rotation *)
  eDMXXSlotDefinitionDouser          := 16#0408, (* Douser *)
  eDMXXSlotDefinitionBarnDoor        := 16#0409, (* Barn Door *)

  (* Control Functions / 0x05xx *)
  eDMXXSlotDefinitionLampControl     := 16#0501, (* Lamp control functions *)
  eDMXXSlotDefinitionFixtureControl := 16#0502, (* Fixture control channel *)
  eDMXXSlotDefinitionFixtureSpeed    := 16#0503, (* Overall speed setting applied to multiple or a
  ll parameters *)
  eDMXXSlotDefinitionMacro           := 16#0504, (* Macro control *)

  eDMXXSlotDefinitionUndefined       := 16#FFFF (* No definition *)
);
END_TYPE

```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

5.36.12 E_DMXXSlotType

```

TYPE E_DMXXSlotType : (* Table C-1 *)
(
  eDMXXSlotTypePrimary      := 0, (* Slot directly controls parameter (represents Coarse for 16-bit parameters) *)
  eDMXXSlotTypeSecFine      := 1, (* Fine, for 16-bit parameters *)
  eDMXXSlotTypeSecTiming    := 2, (* Slot sets timing value for associated parameter *)
  eDMXXSlotTypeSecSpeed     := 3, (* Slot sets speed/velocity for associated parameter *)
  eDMXXSlotTypeSecControl   := 4, (* Slot provides control/mode info for parameter *)
  eDMXXSlotTypeSecIndex     := 5, (* Slot sets index position for associated parameter *)
  eDMXXSlotTypeSecRotation  := 6, (* Slot sets rotation speed for associated parameter *)
  eDMXXSlotTypeSecIndexRotate := 7, (* Combined index/rotation control *)
  eDMXXSlotTypeSecUndefined := 255 (* Undefined secondary type *)
);
END_TYPE

```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

5.36.13 E_DMXStatusType

```

TYPE E_DMXStatusType : (* Table A-4 *)
(
  eDMXStatusTypeNone           := 16#00,
  eDMXStatusTypeGetLastMessage := 16#01,
  eDMXStatusTypeAdvisory       := 16#02,
  eDMXStatusTypeWarning        := 16#03,
  eDMXStatusTypeError          := 16#04,
  eDMXStatusTypeAdvisoryCleared := 16#12,
  eDMXSensorTypeWarningCleared := 16#13,
  eDMXSensorTypeErrorCleared   := 16#14
);
END_TYPE

```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

5.36.14 ST_DM512Personality

```

TYPE ST_DM512Personality :
STRUCT
  byCurrentPersonality      : BYTE;
  byTotalPersonalities      : BYTE;
END_STRUCT
END_TYPE

```

5.36.15 ST_DM512PersonalityDescription

```

TYPE ST_DM512PersonalityDescription :
STRUCT
  iDMX512SlotsRequired      : INT;
  sDescription               : STRING;
END_STRUCT
END_TYPE

```

5.36.16 ST_DM512CommandBuffer

```

TYPE ST_DM512CommandBuffer :
STRUCT
  arrMessageQueue           : ST_DM512MessageQueue;
  stResponseTable           : ST_DM512ResponseTable;
  byTransactionNumber       : BYTE;
END_STRUCT
END_TYPE

```

Also see about this

- 📖 [ST_DM512MessageQueue](#) [▶ 65]
- 📖 [ST_DM512ResponseTable](#) [▶ 66]





5.36.17 ST_DMXDeviceInfo

```

TYPE ST_DMXDeviceInfo :
STRUCT
  uliUID                : ST_DMXMac;
  stRDMPProtocolVersion : ST_DMXRDMProtocolVersion;
  uiDeviceModelId       : UINT;
  stProductCategory     : ST_DMXProductCategory;
  udiSoftwareVersionId  : UDINT;
  uiDMX512Footprint     : UINT;
  stDMX512Personality   : ST_DMX512Personality;
  uiDMX512StartAddress  : UINT;
  uiSubDeviceCount      : UINT;
  bySensorCount         : BYTE;
END_STRUCT
END_TYPE

```

Also see about this

-  [ST_DMXMac](#) [▶ 65]
-  [ST_DMXRDMProtocolVersion](#) [▶ 66]
-  [ST_DMXProductCategory](#) [▶ 66]
-  [ST_DMX512Personality](#) [▶ 64]

5.36.18 ST_DMXMac

```

TYPE ST_DMXMac :
STRUCT
  wHighPart      : WORD;    (* Manufacturer ID / Higher word *)
  dwLowPart      : DWORD;   (* Device ID / Lower double word *)
END_STRUCT
END_TYPE

```

5.36.19 ST_DMXMessageQueue

```

TYPE ST_DMXMessageQueue :
STRUCT
  arrBuffer                : ARRAY [1..20] OF ST_DMXMessageQueueItem;
  byBufferReadPointer      : BYTE;
  byBufferWritePointer     : BYTE;
  byBufferDemandCounter    : BYTE;
  byBufferMaximumDemandCounter : BYTE;
  uiBufferOverflowCounter  : UINT;
  bLockSemaphore          : BOOL;
END_STRUCT
END_TYPE

```

Also see about this

-  [ST_DMXMessageQueueItem](#) [▶ 65]

5.36.20 ST_DMXMessageQueueItem

```

TYPE ST_DMXMessageQueueItem :
STRUCT
  bEntryIsEngaged        : BOOL;
  byMessageLength        : BYTE;
  wDestinationManufacturerId : WORD;
  dwDestinationDeviceId   : DWORD;
  byTransactionNumber     : BYTE;
  byPortId                : BYTE;
  byMessageCount          : BYTE;
  wSubDevice              : WORD;
  byCommandClass          : BYTE;
  wParameterId            : WORD;
  byParameterDataLength   : BYTE;
  arrParameterData        : ARRAY [0..255] OF BYTE;
  bWaitingForDMXSlaveResponse : BOOL;
END_STRUCT
END_TYPE

```

5.36.21 ST_DMXParameterDescription

```

TYPE ST_DMXParameterDescription :
STRUCT
  byParameterDataLength  : BYTE;
  eDataType               : E_DMXDataType;
  ePDCommandClass        : E_DMXParameterDescriptionCommandClass;
  eType                   : E_DMXSensorType;
  eUnit                   : E_DMXSensorUnit;
  eUnitPrefix             : E_DMXSensorUnitPrefix;
  dwMinValidValue        : DWORD;
  dwMaxValidValue        : DWORD;
  dwDefaultValue         : DWORD;
  sDescription            : STRING;
END_STRUCT
END_TYPE

```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

Also see about this

- [E_DMXDataType \[▶ 57\]](#)
- [E_DMXParameterDescriptionCommandClass \[▶ 58\]](#)
- [E_DMXSensorType \[▶ 61\]](#)
- [E_DMXSensorUnit \[▶ 61\]](#)
- [E_DMXSensorUnitPrefix \[▶ 62\]](#)

5.36.22 ST_DMXProductCategory

```

TYPE ST_DMXProductCategory :
STRUCT
  byCoarse  : BYTE;
  byFine    : BYTE;
END_STRUCT
END_TYPE

```

5.36.23 ST_DMXRDMProtocolVersion

```

TYPE ST_DMXRDMProtocolVersion :
STRUCT
  byMajorVersion  : BYTE;
  byMinorVersion  : BYTE;
END_STRUCT
END_TYPE

```

5.36.24 ST_DMXResponseTable

```

TYPE ST_DMXResponseTable :
STRUCT
  arrResponseTable      : ARRAY [1..20] OF ST_DMXResponseTableItem;
  byResponseTableCounter : BYTE;
  byResponseTableMaxCounter : BYTE;
  uiResponseTableOverflowCounter : UINT;
  bLockSemaphore        : BOOL;
END_STRUCT
END_TYPE

```

Also see about this

- [ST_DMXResponseTableItem \[▶ 67\]](#)

5.36.25 ST_DMXResponseTableItem

```

TYPE ST_DMXResponseTableItem :
STRUCT
  bEntryIsEngaged      : BOOL;
  uiErrorId            : UINT;
  iErrorParameter      : INT;
  byMessageLength      : BYTE;
  wDestinationManufacturerId : WORD;
  dwDestinationDeviceId : DWORD;
  wSourceManufacturerId : WORD;
  dwSourceDeviceId     : DWORD;
  byTransactionNumber  : BYTE;
  byResponseType       : BYTE;
  byMessageCount       : BYTE;
  wSubDevice           : WORD;
  byCommandClass       : BYTE;
  wParameterId         : WORD;
  byParameterDataLength : BYTE;
  arrParameterData     : ARRAY [0..255] OF BYTE;
END_STRUCT
END_TYPE
    
```

5.36.26 ST_DMXSensorDefinition

```

TYPE ST_DMXSensorDefinition :
STRUCT
  eSensorType          : E_DMXSensorType;
  eSensorUnit          : E_DMXSensorUnit;
  eSensorUnitPrefix    : E_DMXSensorUnitPrefix;
  iRangeMinimumValue   : INT;
  iRangeMaximumValue   : INT;
  iNormalMinimumValue  : INT;
  iNormalMaximumValue  : INT;
  byRecordValueSupport : BYTE;
  sDescription         : STRING;
END_STRUCT
END_TYPE
    
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

Also see about this

- 📖 [E_DMXSensorType](#) [▶ 61]
- 📖 [E_DMXSensorUnit](#) [▶ 61]
- 📖 [E_DMXSensorUnitPrefix](#) [▶ 62]

5.36.27 ST_DMXSensorValue

```

TYPE ST_DMXSensorValue :
STRUCT
  iPresentValue        : INT;
  iLowestDetectedValue : INT;
  iHighestDetectedValue : INT;
  iRecordedValue       : INT;
END_STRUCT
END_TYPE
    
```

5.36.28 ST_DMXSlotInfo

```

TYPE ST_DMXSlotInfo :
STRUCT
  bEntryIsValid        : BOOL;
  eSlotType            : E_DMXSlotType;
END_STRUCT
    
```





```
eSlotDefinition : E_DMxSlotDefinition;
END_STRUCT
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

Also see about this

-  [E_DMxSlotType \[▶ 63\]](#)
-  [E_DMxSlotDefinition \[▶ 63\]](#)

5.36.29 ST_DMxStatusMessage

```
TYPE ST_DMxStatusMessage :
STRUCT
  bEntryIsValid      : BOOL;
  iSubDeviceId       : INT;
  eStatusType        : E_DMxStatusType;
  iStatusMessageId   : INT;
  iDataValue01       : INT;
  iDataValue02       : INT;
END_STRUCT
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

Also see about this

-  [E_DMxStatusType \[▶ 64\]](#)

5.36.30 ST_EL6851InData

```
TYPE ST_EL6851InData :
STRUCT
  bTransmitAccepted  : BOOL;
  bReceiveToggle     : BOOL;
  bCyclicTxDDisabled : BOOL;
  bDefaultDataSent   : BOOL;
  bFrameSentToggle   : BOOL;
  bTxPDOToggle       : BOOL;
  wChannelLength      : WORD;
  byStartCode        : BYTE;
  byDummy             : BYTE;
  arrData             : ARRAY [1..64] OF BYTE;
END_STRUCT
END_TYPE
```

5.36.31 ST_EL6851InDataEx

```
TYPE ST_EL6851InDataEx :
STRUCT
  bTransmitAccepted  : BOOL;
  bReceiveToggle     : BOOL;
  bCyclicTxDDisabled : BOOL;
  bDefaultDataSent   : BOOL;
  bFrameSentToggle   : BOOL;
  bTxPDOToggle       : BOOL;
  wChannelLength      : WORD;
  byStartCode        : BYTE;
```

```

byDummy1      : BYTE;
arrData       : ARRAY [1..64] OF BYTE;
bWcState      : BOOL;
bInputToggle  : BOOL;
uiState       : UINT;
stAdsAddr     : ST_EL6851AdsAddr;
END_STRUCT
END_TYPE

```

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT 2.11 R3/x64 higher than Build 2256	PC/CX	TcDMX-library higher than V1.3.0

5.36.32 ST_EL6851OutData

```

TYPE ST_EL6851OutData :
STRUCT
  bTransmitRequest : BOOL;
  bDisableCyclicTxD : BOOL;
  bSendDefaultData : BOOL;
  byDummy1         : BYTE;
  wChannelLength   : WORD;
  byStartCode      : BYTE;
  byDummy2         : BYTE;
  arrData          : ARRAY [1..512] OF BYTE;
END_STRUCT
END_TYPE

```

5.37 Error codes

Error codes of TwinCAT PLC Library TcDMX

Value (hex)	Value (dec)	Description
0x0000	0	No error.
0x8001	32769	No answer from the DMX terminal.
0x8002	32770	No answer from the DMX device.
0x8003	32771	Communication buffer overflow.
0x8004	32772	No answer from the communication block.
0x8005	32773	The <i>byPortId</i> parameter is outside the valid range.
0x8006	32774	Checksum error.
0x8007	32775	The <i>eResetDeviceType</i> parameter is outside the valid range.
0x8008	32776	Timeout.
0x8009	32777	The <i>uliLowerBoundUID</i> parameter is larger than the <i>uliUpperBoundUID</i> parameter.
0x800A	32778	No RDM commands can be transmitted because the terminal is in cycle mode.
0x800B	32779	The <i>iDMX512StartAddress</i> parameter is outside the valid range (1 - 512).
0x800C	32780	Error in setting the DMX512 start address.
0x800D	32781	No process data can be transmitted because the terminal is not in cycle mode.
0x800E	32782	It is a RDM telegram received with the data length 0.
0x800F	32783	RDM response: Reply of the RDM telegram is invalid.
0x8010	32784	RDM response: The DMX slave cannot comply with request because the message is not implemented in responder.
0x8011	32785	RDM response: The DMX slave cannot interpret request as controller data was not formatted correctly.
0x8012	32786	RDM response: The DMX slave cannot comply due to an internal hardware fault.
0x8013	32787	RDM response: Proxy is not the RDM line master and cannot comply with message.

Value (hex)	Value (dec)	Description
0x8014	32788	RDM response: SET Command normally allowed but being blocked currently.
0x8015	32789	RDM response: Not valid for Command Class attempted. May be used where GET allowed but SET is not supported.
0x8016	32790	RDM response: Value for given Parameter out of allowable range or not supported.
0x8017	32791	RDM response: Buffer or Queue space currently has no free space to store data.
0x8018	32792	RDM response: Incoming message exceeds buffer capacity.
0x8019	32793	RDM response: Sub-Device is out of range or unknown.
0x801A	32794	The <i>iDMX512SlotOffset</i> parameter is outside the valid range (0-511).
0x801B	32795	The <i>bySensorNumber</i> parameter is outside the valid range (0-254).
0x801C	32796	RDM-Response: The field Parameter Data (PD) is too long. It was not possible to receive all the data of the reply. For this purpose, the function block <code>FB_EL6851CommunicationEx()</code> [► 22] must be used.
0x801D	32797	The ADS address to access the PDOs is invalid. Was the structure <i>AdsAddr</i> of the KL6851 mapped to the corresponding variable?
0x801E	32798	During read access to the PDOs an ADS error has occurred.

6 Appendix

6.1 Transmission of cyclic process data as DMX master (EL6851)

<https://infosys.beckhoff.com/content/1033/tcplclibdmx/Resources/11977739147/.zip> <https://infosys.beckhoff.com/content/1033/tcplclibdmx/Resources/11977739147/.zip>

Starting the example program

The application examples have been tested with a test configuration and are described accordingly. Certain deviations when setting up actual applications are possible.

The following hardware and software was used for the test configuration:

- TwinCAT master PC with TwinCAT version 2.11 (Build 2229) or newer and INTEL PRO/100 VE Ethernet adapter
- Beckhoff EtherCAT Coupler EK1100, [EL6851 \[▶ 75\]](#) and EL9011 Terminals
- RGB-LED DMX slave with 3 channels (one for each colour). One slot is occupied per channel.

The DMX slave is to be wired in accordance with the [connection diagram \[▶ 75\]](#).

Procedure for starting the program

- Save the <https://infosys.beckhoff.com/content/1033/tcplclibdmx/Resources/11977739147/.zip> for the TwinCAT System Manager and the PRO file for TwinCAT PLC Control <https://infosys.beckhoff.com/content/1033/tcplclibdmx/Resources/11977739147/.zip> locally on your hard drive.
- Start the *.TSM file and the *.PRO file; the TwinCAT System Manager and TwinCAT PLC Control open.
- Connect the hardware in accordance with fig. 1 and connect the Ethernet adapter of your PC to the EtherCAT coupler (further information on this can be found in the corresponding coupler manuals)
- Select the local Ethernet adapter (with real-time driver, if one) under System configuration, I/O configuration, I/O devices, Device (EtherCAT); on the "Adapter" tab choose "Search...", select the appropriate adapter and confirm (see Fig. 2a + 2b)

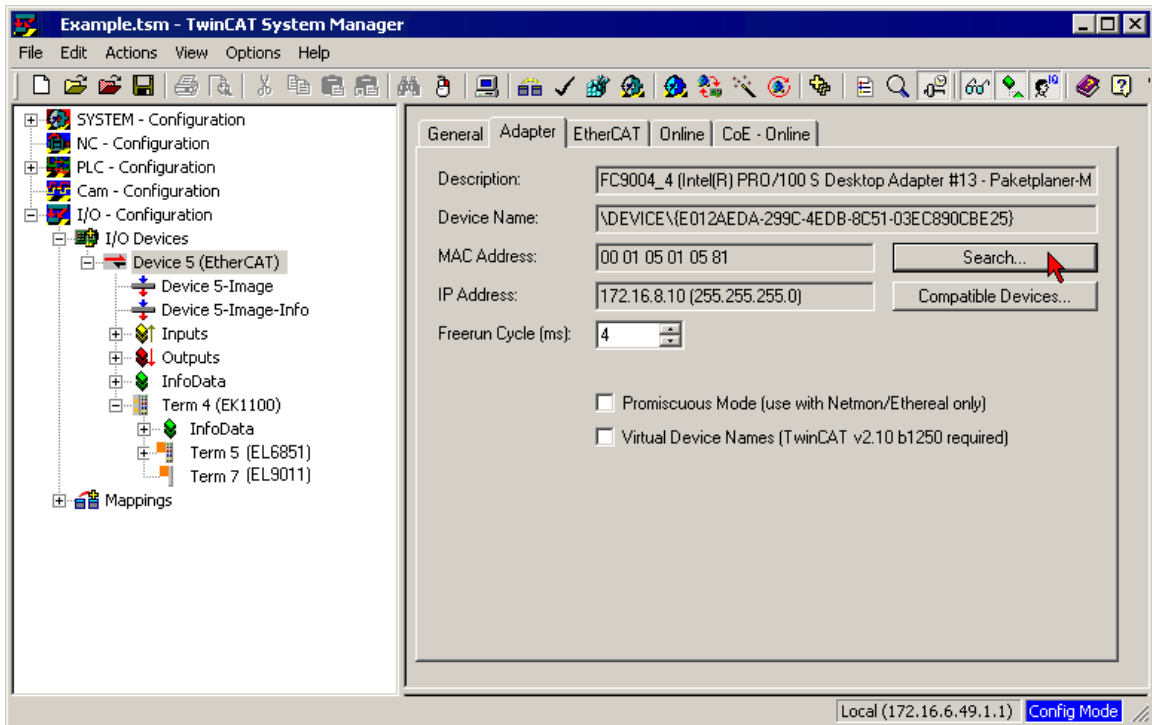


Fig. 2a: Searching the Ethernet adapter



Fig. 2b: Selection and confirmation of the Ethernet adapter

- Activation of the configuration and confirmation (Fig. 3a +3b)



Fig. 3a: Activation of the configuration

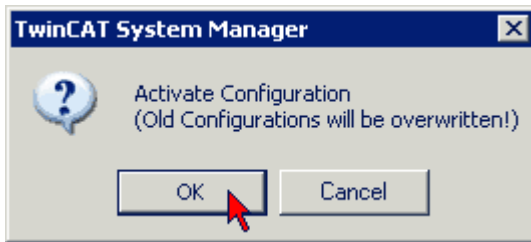


Fig. 3b: Confirming the activation of the configuration

- Confirming new variable mapping, restart in RUN mode (Fig. 4a + 4b)

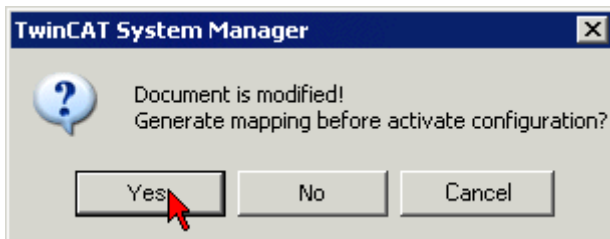


Fig. 4a: Generating variable mapping

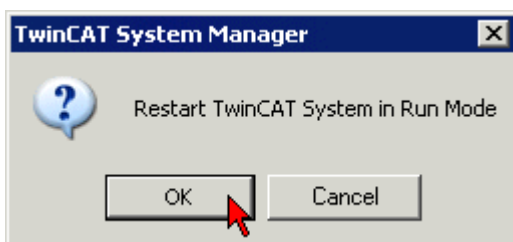


Fig. 4b: Restarting TwinCAT in RUN mode

- In TwinCAT PLC Control, under the 'Project' menu, select 'Compile all' to compile the project (Fig. 5)

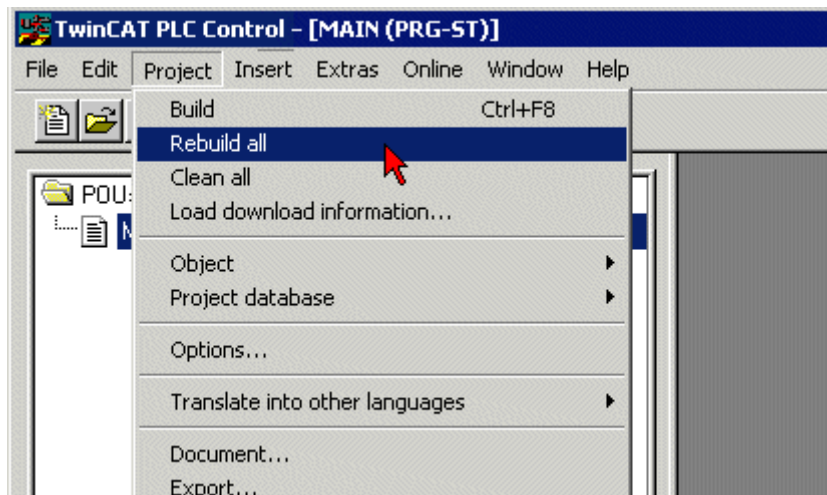


Fig. 5: Compile project

- In TwinCAT PLC Control: log in with the "F11" button, confirm loading the program (Fig. 6), run the program with the "F5" button

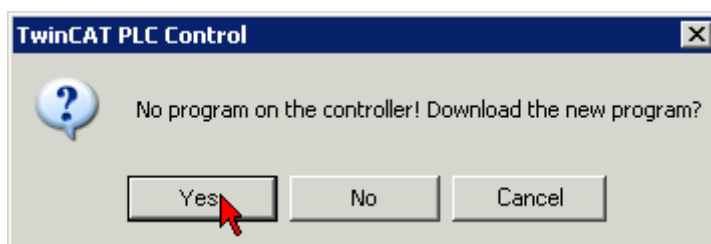


Fig. 6: Confirming program start

Visualization

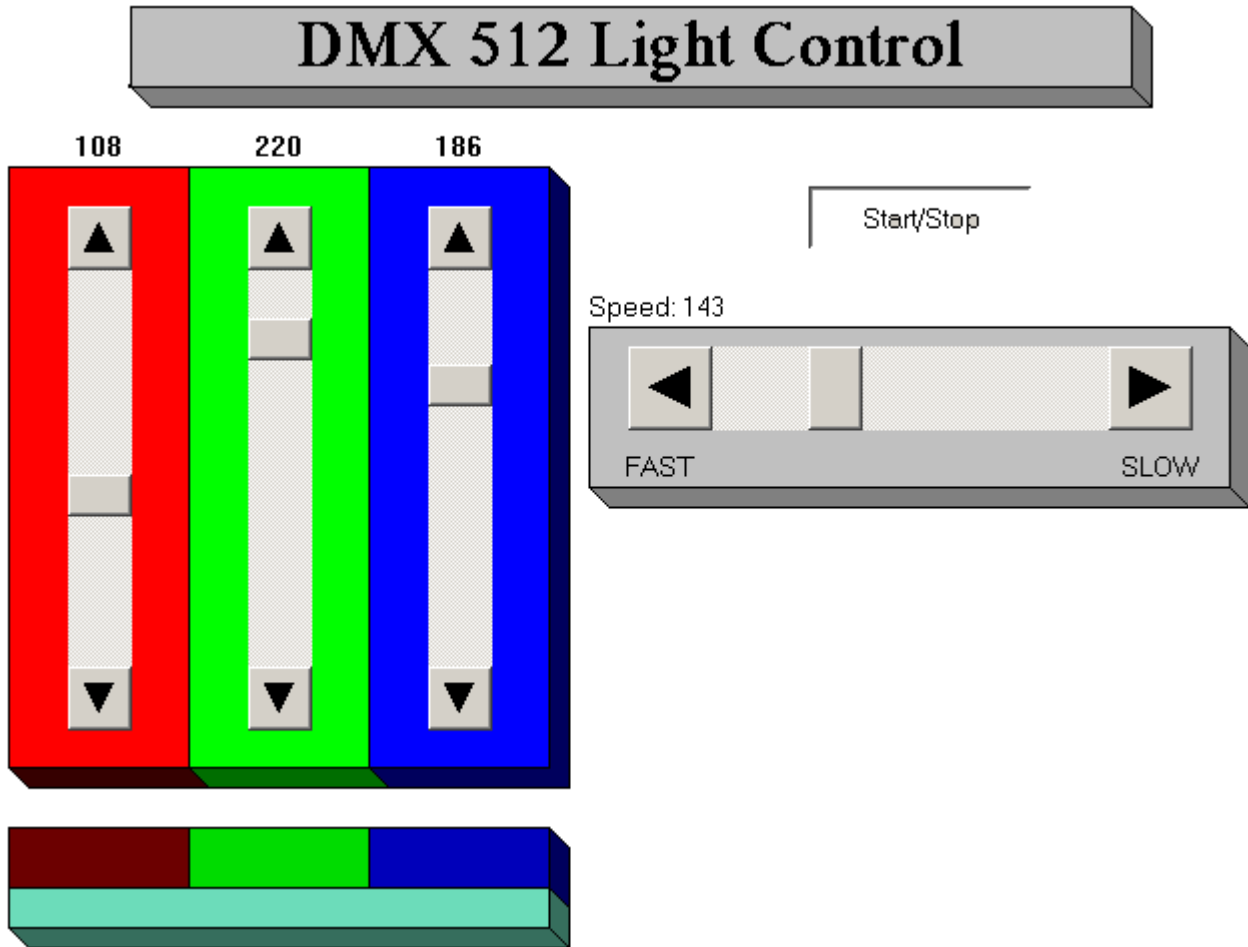


Fig. 7: Specification of the setting variables for the three colours of the DMX slave in TwinCAT PLC Control

The example transmits the DMX data cyclically to a DMX slave. The DMX device used here occupies three slots (bytes) in the DMX 512 frame. Each slot addresses one of the three colours. If the 'Start/Stop' button is pressed, then automatically generated data is transmitted to the DMX device. The speed of the changes can be altered using the horizontal slide control. If the 'Start/Stop' button is not pressed, you can change the values manually using the three vertical slide controls.

6.2 Receipt of 64 bytes data to two DMX slaves (EL6851-0010) in each case

Unpacking the example files [▶ 75] <https://infosys.beckhoff.com/content/1033/tcplclibdmx/Resources/11977740555.zip>

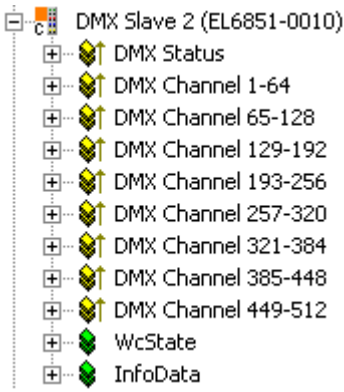


Fig. 3: 8 arrays each with 64 bytes, full range (all PDO activated)

The EL6851-0010 [► 75] is able to read max. 512 bytes (64 bytes in eight arrays in each case, see fig. 3). The arrays can assigned via PDO 0x1C13 in the TwinCAT System Manger (tab "process data").

Example:

- DMX Channel 1 - 64 -> Index 0x1A01
- DMX Channel 65 - 128 -> Index 0x1A02
-
- DMX Channel 449 - 512 -> Index 0x1A08

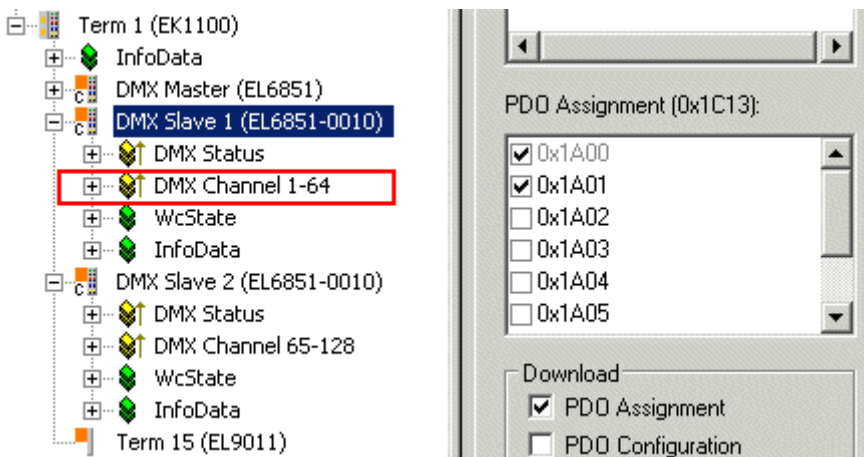


Fig. 4: DMX Channel 1 - 64 (default), select PDO 0x1A01

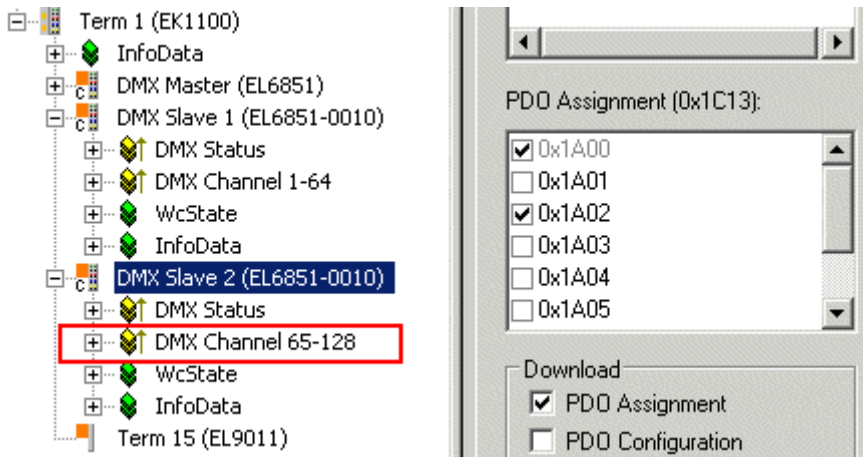


Fig. 5: DMX Channel 65 - 128, select PDO 0x1A02

In the sample program the first DMX slave receives the first sent 64 bytes and the second one the next 64 bytes (fig. 4+ 5, it is possible to receive all 128 bytes with one EL6851-0010, the segmentation in the example is deliberately selected).

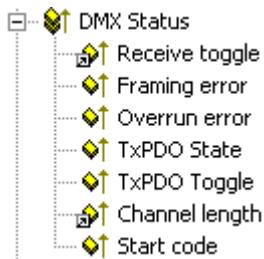


Abb. 6: DMX status object

In the DMX status object (Index 0x6000, "DMX-Status", fig. 6) a copy counter is established with Index 0x6000:11 ("Channel length").

Example: If the PDO 0x1A01 is activated, the value of "Channel length" is 64_{dez}. If the PDO 0x1A02 is activated, the value is 128_{dez}. If both PDOs are activated (0x1A01 and 0x1A02) the value is also 128_{dez}.

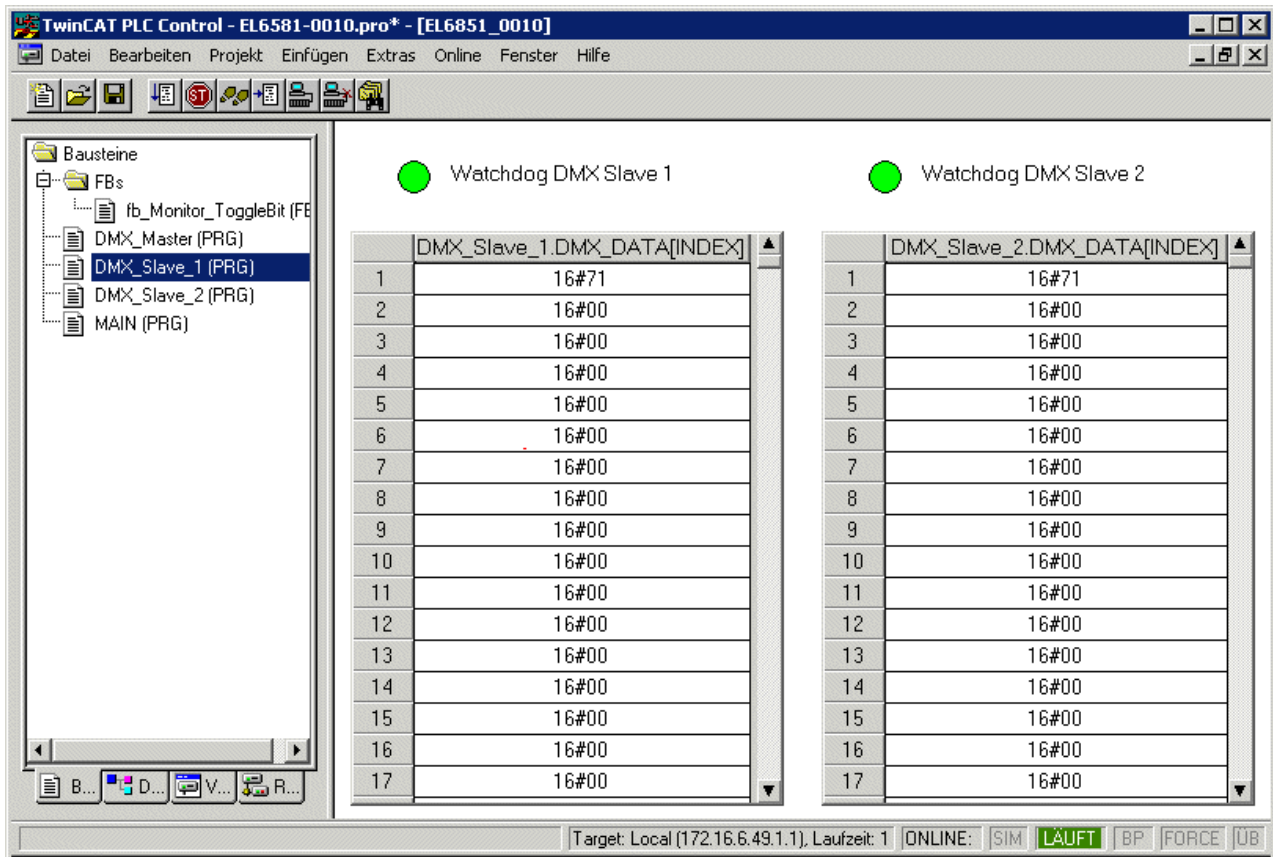


Fig. 6: Visualization example 3 in the TwinCAT PLC

DMX Slave 1 receives 64 bytes data on channel 1 of the first array ("DMX Channel 1 - 64")
 DMX Slave 1 receives 64 bytes data on channel 1 of the second array ("DMX Channel 65 - 128")

The "Receive toggle" Bit (Index 0x6000:02) is evaluated with the FB "fb_Monitor_ToggleBit" and displayed in the PLC visualization (Watchdog DMX Slave).

6.3 Configuration of DMX slaves via Remote Device Management (RDM)

<https://infosys.beckhoff.com/content/1033/tcplclibdmx/Resources/11977741963/.zip> <https://infosys.beckhoff.com/content/1033/tcplclibdmx/Resources/11977741963/.zip>

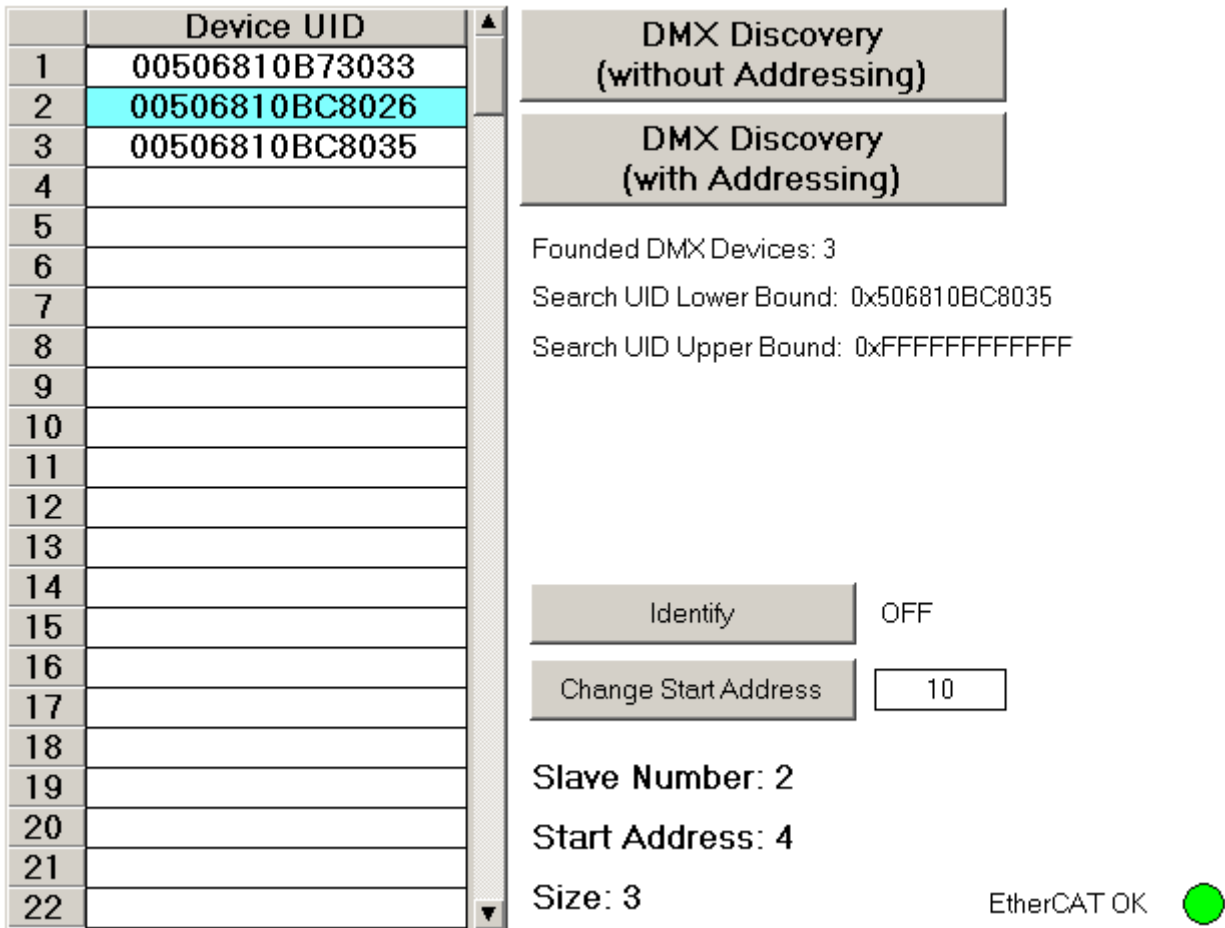


Fig. 2: Visualization example 2 in the TwinCAT PLC

● EtherCAT functionality

i The dialogue is only functioning when the 'EtherCAT OK' LED lights up green. A red LED indicates a fault in the EtherCAT communication.

DMX slaves are sought on the DMX line with the 'DMX Discovery (without Addressing)' button. All DMX slaves found are displayed in the list on the left. An entry is selected by clicking it. After confirming 'Identify', the RDM command is sent to identify the respective DMX device. The start address can be entered in the input field adjacent to the 'Change Start Address' button. After pressing the button, the new start address will be transmitted to the selected DMX device. The number of the DMX device, the start address in the DMX512 frame and the slot size (number of bytes in the DMX frame) for the selected device are displayed in the lower area.

6.4 DMX-Master with BC9191-0100

This example describes an simple PLC program how to communicate with DMX devices. The three analog values of the BC9191-0100 analog inputs are read and then output to the DMX channels 1-3 with the DMX interface of terminal strip X1.

<https://infosys.beckhoff.com/content/1033/tcplclibdmx/Resources/11977743371/.zip> <https://infosys.beckhoff.com/content/1033/tcplclibdmx/Resources/11977743371/.zip>

Hardware

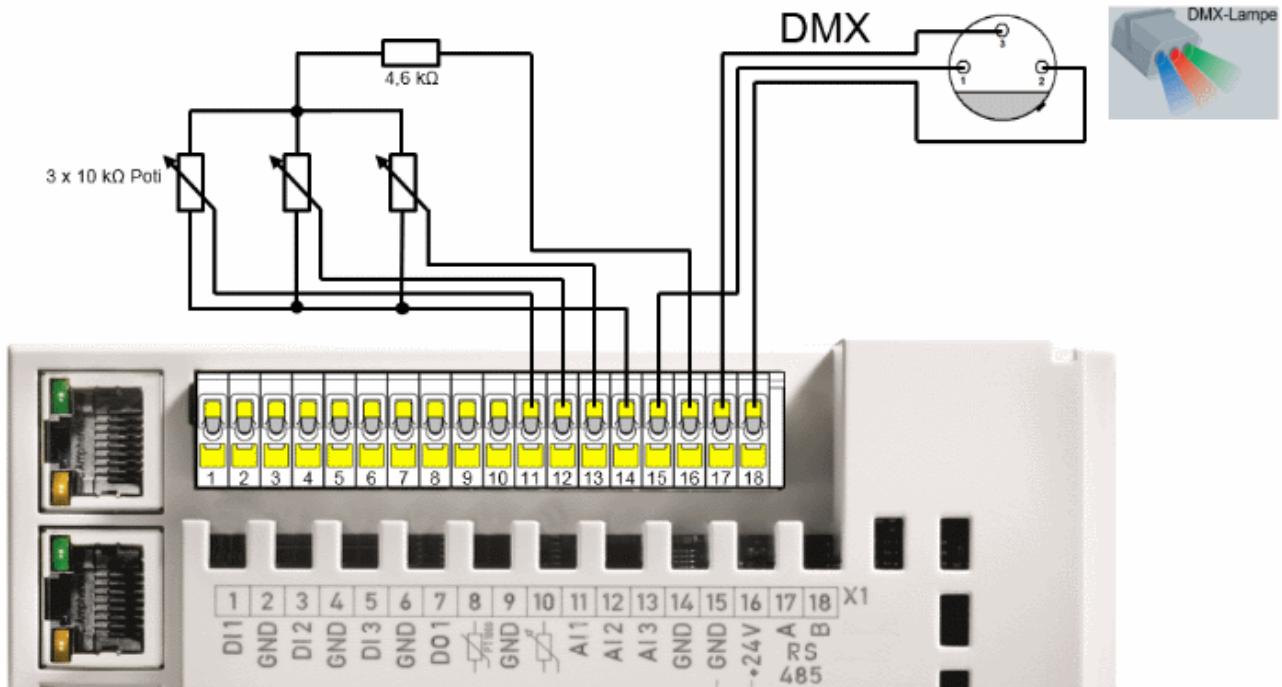
Setting up the components

The following hardware is required:

- 1x Bus Terminal Controller BC9191 with RS485
- 1x end terminal KL9010
- 1x DMX device

DMX assignment

Terminal strip X1	3-pol. XLR socket	5-pol. XLR socket
X1 / 15 (ground/screen)	Pin 1	Pin 1
X1 / 17 (DMX +)	Pin 3	Pin 3
X1 / 18 (DMX -)	Pin 2	Pin 2



Set up the DMX devices as described in the associated documentation.

Software

Explanations concerning the PLC program can be found as comments in the source code.

6.5 Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

Beckhoff's branch offices and representatives

Please contact your Beckhoff branch office or representative for local support and service on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on her internet pages: <https://www.beckhoff.com>

You will also find further documentation for Beckhoff components there.

Beckhoff Support

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

- support
- design, programming and commissioning of complex automation systems
- and extensive training program for Beckhoff system components

Hotline: +49 5246 963 157
Fax: +49 5246 963 9157
e-mail: support@beckhoff.com

Beckhoff Service

The Beckhoff Service Center supports you in all matters of after-sales service:

- on-site service
- repair service
- spare parts service
- hotline service

Hotline: +49 5246 963 460
Fax: +49 5246 963 479
e-mail: service@beckhoff.com

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

Phone: +49 5246 963 0
Fax: +49 5246 963 198
e-mail: info@beckhoff.com
web: <https://www.beckhoff.com>

More Information:
www.beckhoff.com/tx1200

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

