# 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 Introduction ................................................................................................................................................ 8

3 DMX .......................................................................................................................................................... 9

4 Programming ........................................................................................................................................... 10
   4.1 POU s ................................................................................................................................................ 10
       4.1.1 High Level ................................................................................................................................. 10
       4.1.2 Low Level .................................................................................................................................. 12
       4.1.3 Error codes ............................................................................................................................... 50
   4.2 DUTs ................................................................................................................................................ 51
       4.2.1 Enums ....................................................................................................................................... 51
       4.2.2 Structures .................................................................................................................................. 57
   4.3 Integration into TwinCAT ................................................................................................................. 63
       4.3.1 EL6851 with CX5120 ................................................................................................................ 63

5 Appendix .................................................................................................................................................. 68
   5.1 Example: Configuration by RDM ..................................................................................................... 68
   5.2 Example: DMX master ...................................................................................................................... 69
   5.3 Example: DMX slave .......................................................................................................................... 72
   5.4 Support and Service .......................................................................................................................... 75
1 Foreword

1.1 Notes on the documentation

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

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

Disclaimer

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

Trademarks

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

Patent Pending

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

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.
The reproduction, distribution and utilization of this document as well as the communication of its contents to
others without express authorization are prohibited.
Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a
patent, utility model or design.
1.2 Safety instructions

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔴 DANGER</td>
<td>Serious risk of injury! Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.</td>
</tr>
<tr>
<td>🔴 WARNING</td>
<td>Risk of injury! Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.</td>
</tr>
<tr>
<td>🔴 CAUTION</td>
<td>Personal injuries! Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.</td>
</tr>
<tr>
<td><strong>NOTE</strong></td>
<td>Damage to the environment or devices Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.</td>
</tr>
<tr>
<td>☰ Tip or pointer</td>
<td>This symbol indicates information that contributes to better understanding.</td>
</tr>
</tbody>
</table>
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 Introduction

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

- TwinCAT XAE
- PC and network knowledge
- Structure and properties of the Beckhoff Embedded PC and its Bus Terminal system
- Technology of DMX devices

The Tc2_DMX library is usable on all hardware platforms that support TwinCAT 3.1 or higher.

Hardware documentation EL6851 in the Beckhoff information system.
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. TwinCAT function blocks enable implementation of the RDM protocol (Remote Device Management) for internal diagnostics and parameterization in DMX.

The DMX master transmits new settings to the slaves cyclically at 250 kBaud in order 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 inharmonious. 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 Programming

4.1 POUs

4.1.1 High Level

4.1.1.1 FB_DMXDiscovery

This function block searches for up to 50 DMX devices and automatically sets the start address (optional). 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.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMX_OPTION_COMPLETE_NEW_DISCOVERY</td>
<td>All DMX devices are taken into account.</td>
</tr>
<tr>
<td>DMX_OPTION_SET_START_ADDRESS</td>
<td>The start address is set for all DMX devices that are found. Consecutive, starting with 1.</td>
</tr>
<tr>
<td>DMX_OPTION_OPTICAL_FEEDBACK</td>
<td>When a DMX device is found, the function IDENTIFY_DEVICE is called for two seconds.</td>
</tr>
</tbody>
</table>

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 function 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 [50]).

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 main information of the found DMX devices (see ST_DMXDeviceInfo [58]).

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: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.1.2 FB_DMXDiscovery512

This function block searches for up to 512 DMX devices and automatically sets the start address (optional). 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.

VAR_OUTPUT

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

The most important information for the devices found is displayed in a structure.
bBusy: When the function 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 [50]).

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 main information of the found DMX devices (see ST_DMXDeviceInfo [58]).

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: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.2 Low Level

4.1.2.1 Base

4.1.2.1.1 FB_DMXSendRDMCommand

This function block is for the general sending of a RDM command, defined by command number and, if necessary, transfer parameter.

VAR_INPUT

bStart : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId : DWORD;
byPortId : BYTE;
byResponseMessageCount : BYTE;
byResponseDataLength : BYTE;
arrResponseData : ARRAY [0..63] OF BYTE;
eCommandClass : SHORT;
byParameterId : BYTE;
byParameterDataLength : BYTE;
arrParameterData : ARRAY [0..1023] OF BYTE;
dwOptions : DWORD;
StCommandBuffer : ST_DMXCommandBuffer;
wSubDevice : WORD;
eCommandClass : E_DMXCommandClass;
eParameterId : E_DMXParameterId;
byParameterDataLength : BYTE;
arrParameterData : ARRAY [0..255] OF BYTE;
dwOptions := 0;

bStart: The block is activated 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.

wSubDevice: Sub-devices should be used in devices with a recurring number of modules, such as a dimmer rack. The sub-devices input enables parameter messages to be sent to a particular module within the device, in order to read or set module properties.

eCommandClass: Command Class (CC) indicates the message action (see E_DMXCommandClass [51]).

eParameterId: Parameter Id is a 16-bit number, which identifies a particular type of parameter data (see E_DMXParameterId [52]).

byParameterDataLength: The parameter data length (PDL) is the preceding number of slots, included in the parameter data area. If this input is 0x00, there is no parameter data to follow.

arrParameterData: Parameter data of variable length. The format of the content depends on the PID.

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 function 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.

byResponseMessageCount: This output indicates that the DMX slave contains further messages. The RDM command Get: QUEUES_MESSAG is used to read these messages.

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

arrResponseData: This output contains the data of the response from the RDM command. The length is variable, and the data format depends on the RDM command.

VAR_IN_OUT

stCommandBuffer : ST_DMXCommandBuffer;

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>
4.1.2.1.2 FB_EL6851Communication

**Outdated**

This function block is outdated. Use FB_EL6851CommunicationEx() instead.

The EL6851 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 analyze the level of utilization of the PLC task with the aid of the TwinCAT System Manager.

The function block `FB_EL6851Communication()` 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.

Examples for both operating modes can be found in the appendix.

**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). A list of all known manufacturer IDs can be found at [http://www.esta.org/tsp/working_groups/CP/mfctrID.php](http://www.esta.org/tsp/working_groups/CP/mfctrID.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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wSourceManufacturerId</code></td>
<td>WORD</td>
<td><code>16#42_41</code></td>
</tr>
<tr>
<td><code>dwSourceDeviceId</code></td>
<td>DWORD</td>
<td><code>16#12_13_14_15</code></td>
</tr>
<tr>
<td><code>bEnableSendingData</code></td>
<td>BOOL</td>
<td>TRUE</td>
</tr>
<tr>
<td><code>bSetCycleMode</code></td>
<td>BOOL</td>
<td>TRUE</td>
</tr>
<tr>
<td><code>uiDataLength</code></td>
<td>WORD</td>
<td></td>
</tr>
<tr>
<td><code>arrProcessData</code></td>
<td>ARRAY</td>
<td><code>ST_DMXCommandBuffer</code></td>
</tr>
</tbody>
</table>
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 (CycleMode 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 in order 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 [50]).
bCycleMode: Is TRUE if cycle mode is active (see also bSetCycleMode input).
byBufferDemandMeter: Occupancy of the respective buffer (0 - 100%).
byBufferMaximumDemandMeter: previous maximum occupancy 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 between the EL6851 and the PLC (see ST_EL6851InData [62]).
stEL6851OutData: Structure in the EL6851 output process image. It is used for communication between the PLC and the EL6851 (see ST_EL6851OutData [63]).
stCommandBuffer: reference to the structure for communication (buffer) with the FB_KL6851Communication() [14] function block (see ST_DMXCommandBuffer [58]).

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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

**4.1.2.1.3 FB_EL6851CommunicationEx**

The EL6851 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_EL6851CommunicationEx()` 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_EL6851CommunicationEx()` 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 analyze the level of utilization of the PLC task with the aid of the TwinCAT System Manager.

The function block `FB_EL6851CommunicationEx()` can be called in a separate faster task, if necessary. In this case, the faster task in which the `FB_EL6851CommunicationEx()` block is called should have a higher priority than the TASK in which the block for the RDM commands is called.

Examples for both operating modes can be found in the appendix.

**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). A list of all known manufacturer IDs can be found at [http://www.esta.org/tsp/working_groups/CP/mfctrIDs.php](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 0x42_41. Since the DMX master also has a UID, this should be specified in accordance with the ESTA (`wSourceManufacturerId` input).

**VAR_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 (CycleMode 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 in order 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 [50]).

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

byBufferDemandMeter: Occupancy of the respective buffer (0 - 100%).

byBufferMaximumDemandMeter: previous maximum occupancy 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 between the EL6851 and the PLC (see ST_EL6851InDataEx [62]).

stEL6851OutData : Structure in the EL6851 output process image. It is used for communication between the PLC and the EL6851 (see ST_EL6851OutData[63]).

stCommandBuffer: Reference to the structure for communication (buffer) with the FB_EL6851CommunicationEx() [16] function block (see ST_DMXCommandBuffer [58]).

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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.2.2 Device Control Parameter Message

4.1.2.2.1 FB_DMXGetIdentifyDevice

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

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

\textbf{VAR_INPUT}

\begin{verbatim}
bStart             : BOOL;
wDestinationManufacturerId : WORD;
dwDestinationDeviceId    : DWORD;
byPortId             : BYTE;
dwOptions            : DWORD := 0;
\end{verbatim}

\texttt{bStart}: The command is started by a positive edge at this input.

\texttt{wDestinationManufacturerId}: Unique manufacturer Id of the DMX device (for details, see DMX device address).

\texttt{dwDestinationDeviceId}: Unique device Id of the DMX device (for details, see DMX device address).

\texttt{byPortId}: Channel within the addressed DMX device. Sub-devices are addressed through the Port Id. The root device always has the Port Id 0.

\texttt{dwOptions}: Options (currently not used).

\textbf{VAR_OUTPUT}

\begin{verbatim}
bBusy              : BOOL;
bError             : BOOL;
udiErrorId         : UDINT;
bIdentifyActive    : BOOL;
\end{verbatim}

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

\texttt{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 \texttt{udiErrorId}. Only valid if \texttt{bBusy} is FALSE.

\texttt{udiErrorId}: Contains the command-specific error code of the most recently executed command. Only valid if \texttt{bBusy} is FALSE (see error codes [501]).

\texttt{bIdentifyActive}: If the execution of the command has been completed (\texttt{bBusy} is FALSE), then the state of identification of the DMX device is displayed at this output.
VAR_IN_OUT

stCommandBuffer : ST_DMXCommandBuffer;

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.2.2.2 FB_DMXSetIdentifyDevice

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

Applying a positive edge to the bStart input starts the function 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 (for details, see DMX device address).

dwDestinationDeviceId: Unique device Id of the DMX device (for details, see DMX device address).

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 [50]).

**VAR_IN_OUT**

| stCommandBuffer | : | ST_DMXCommandBuffer; |

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th></th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td></td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

### 4.1.2.3 FB_DMXSetResetDevice

This function block activates a reset in a DMX device.

Applying a positive edge to the bStart input starts the function 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 := E_DMXResetDeviceTypeWarm; |
| dwOptions | : | DWORD := 0; |

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

wDestinationManufacturerId: Unique manufacturer Id of the DMX device (for details, see DMX device address).

dwDestinationDeviceId: Unique device Id of the DMX device (for details, see DMX device address).

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 (see E_DMXResetDeviceType [54]).

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 [50]).

**VAR_IN_OUT**

```plaintext
stCommandBuffer : ST_DMXCommandBuffer;
```

**stCommandBuffer**: Reference to the structure for communication (buffer) with the block `FB_EL6851Communication([14])` (see `ST_DMXCommandBuffer[58]`).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

### 4.1.2.3 Discovery Messages

#### 4.1.2.3.1 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([22])` command (mute flag is not set) or not (mute flag is set).

Applying a positive edge to the `bStart` input starts the function 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**

```plaintext
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 (for details, see DMX device address).

**dwDestinationDeviceId**: Unique device Id of the DMX device (for details, see DMX device address).

**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 [50]).

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:

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

VAR_IN_OUT

stCommandBuffer : ST_DMXCommandBuffer;

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.2.3.2 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 function block, and the bBusy output goes TRUE. The inputs wLowerBoundManufacturerId, dwLowerBoundDeviceId, wUpperBoundManufacturerId and dwUpperBoundDeviceId define the address range in which DMX devices are searched. The byPortId input defines the channel within the addressed DMX device. If the execution of the command has been completed,
the \textit{bBusy} output goes back to FALSE. The \textit{bError}, \textit{udiErrorId}, \textit{wReceivedManufacturerId} and \textit{dwReceivedDeviceId} outputs can now be evaluated. Further positive edges at the \textit{bStart} input will be ignored as long as the function block is active (\textit{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 \textit{wReceivedManufacturerId} and \textit{dwReceivedDeviceId} outputs. If no DMX devices are found in this range, the output \textit{bError} is TRUE and \textit{udiErrorId} is 0x8002 (no response from DMX device). If two or more DMX devices are found in the address range, \textit{bError} is TRUE and \textit{udiError} contains an 0x8006 entry (checksum error).

\textbf{VAR\_INPUT}

\begin{verbatim}
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).
\end{verbatim}

\textbf{VAR\_OUTPUT}

\begin{verbatim}
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 \textit{udiErrorId}. Only valid if \textit{bBusy} is FALSE.
udiErrorId: Contains the command-specific error code of the most recently executed command. Only valid if \textit{bBusy} is FALSE (see error codes \cite{error_codes}.
wReceivedManufacturerId: If the execution of the command has been completed (\textit{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 (\textit{bBusy} is FALSE), then the state of identification of the DMX device is displayed at this output.
\end{verbatim}

\textbf{VAR\_IN\_OUT}

\begin{verbatim}
VAR_IN_OUT
stCommandBuffer : ST_DMXCommandBuffer;

stCommandBuffer: Reference to the structure for communication (buffer) with the block \texttt{FB\_EL6851\_Communication()} \cite{FB_EL6851\_Communication} (see \texttt{ST\_DMXCommandBuffer} \cite{ST\_DMXCommandBuffer}).
\end{verbatim}

\textbf{Requirements}

\begin{tabular}{|l|l|}
\hline
Development environment & required TC3 PLC library \\
\hline
TwinCAT from v3.1.4020.14 & Tc2\_DMX from 3.5.3.0 \\
\hline
\end{tabular}
4.1.2.3.3 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() command (mute flag is not set) or not (mute flag is set).

Applying a positive edge to the `bStart` input starts the function 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`: The command is started by a positive edge at this input.
- `wDestinationManufacturerId`: Unique manufacturer Id of the DMX device (for details, see DMX device address).
- `dwDestinationDeviceId`: Unique device Id of the DMX device (for details, see DMX device address).
- `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`: 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 [50]).
- `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:
<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Managed Proxy Flag</td>
</tr>
<tr>
<td>1</td>
<td>Sub-Device Flag</td>
</tr>
<tr>
<td>2</td>
<td>Boot-Loader Flag</td>
</tr>
<tr>
<td>3</td>
<td>Proxied Device Flag</td>
</tr>
<tr>
<td>4-15</td>
<td>Reserve (always 0).</td>
</tr>
</tbody>
</table>

**VAR_IN_OUT**

stCommandBuffer : ST_DMXCommandBuffer;

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

**4.1.2.4 Power and Lamp Setting Parameter Messages**

**4.1.2.4.1 FB_DMXGetLampHours**

This function block reads the number of hours in which the lamp was on. The block FB_DMXSetLampHours() [27] 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 (for details, see DMX device address).

dwDestinationDeviceId: Unique device Id of the DMX device (for details, see DMX device address).

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 function 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 [50]).

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

VAR_IN_OUT

stCommandBuffer : ST_DMXCommandBuffer;

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.2.4.2 FB_DMXGetLampOnMode

This function block reads the parameter that defines the switch-on characteristics of the DMX device. The block FB_DMXSetLampOnMode() [28] 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 (for details, see DMX device address).

dwDestinationDeviceId: Unique device Id of the DMX device (for details, see DMX device address).

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 function 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 \[50\]).

eLampOnMode: Contains the current parameter that defines the switch-on characteristics of the DMX device (see E_DMXLampOnMode \[51\]).

VAR_IN_OUT

stCommandBuffer : ST_DMXCommandBuffer;

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_E(L6851Communication) \[14\] (see ST_DMXCommandBuffer \[58\]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.2.4.3 FB_DMXSetLampHours

This function block sets the operating hours counter for the lamp. The block FB_DMXGetLampHours() \[25\] 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 block is activated 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 [50]).

VAR_IN_OUT

stCommandBuffer : ST_DMXCommandBuffer;

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.2.4.4 FB_DMXSetLampOnMode

This function block defines the switch-on characteristics of the DMX device. The block FB_DMXGetLampOnMode() [26] 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 block is activated 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 (see E_DMXLampOnMode [51]).

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 [50]).

VAR_IN_OUT

stCommandBuffer : ST_DMXCommandBuffer;

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.2.5 Product Information Messages

4.1.2.5.1 FB_DMXGetDeviceInfo

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

Applying a positive edge to the bStart input starts the function 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 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 (for details, see DMX device address).

dwDestinationDeviceId: Unique device Id of the DMX device (for details, see DMX device address).
**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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
<tr>
<td>udiErrorId</td>
<td>UDINT</td>
</tr>
<tr>
<td>stDMXDeviceInfo</td>
<td>ST_DMXDeviceInfo</td>
</tr>
</tbody>
</table>

**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 [50]).

**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 `ST_DMXDeviceInfo` [58]).

### VAR_IN_OUT

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>stCommandBuffer</td>
<td>ST_DMXCommandBuffer</td>
</tr>
</tbody>
</table>

**stCommandBuffer:** Reference to the structure for communication (buffer) with the block `FB_EL6851Communication()` [14] (see `ST_DMXCommandBuffer` [58]).

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

### 4.1.2.5.2 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()` [35] can be used to edit the text.

### VAR_INPUT

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bStart</td>
<td>BOOL</td>
</tr>
<tr>
<td>wDestinationId</td>
<td>WORD</td>
</tr>
<tr>
<td>dwDestinationId</td>
<td>DWORD</td>
</tr>
<tr>
<td>byPortId</td>
<td>BYTE</td>
</tr>
<tr>
<td>dwOptions</td>
<td>DWORD := 0</td>
</tr>
</tbody>
</table>

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

**wDestinationManufacturerId:** Unique manufacturer Id of the DMX device (for details, see DMX device address).

**dwDestinationDeviceId:** Unique device Id of the DMX device (for details, see DMX device address).

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

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

**bBusy**: When the function 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 [50]).

**sDeviceLabel**: Description text for the DMX device.

**VAR_IN_OUT**

stCommandBuffer : ST_DMXCommandBuffer;

**stCommandBuffer**: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

**4.1.2.5.3 FB_DMXGetDeviceModelDescription**

This function block queries the description of the device type.

Applying a positive edge to the bStart input starts the function 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 (for details, see DMX device address).

**dwDestinationDeviceId**: Unique device Id of the DMX device (for details, see DMX device address).
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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL;</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL;</td>
</tr>
<tr>
<td>udiErrorId</td>
<td>UDINT;</td>
</tr>
<tr>
<td>sDeviceModelDescription</td>
<td>STRING;</td>
</tr>
</tbody>
</table>

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 [50]).

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>stCommandBuffer</td>
<td>ST_DMXCommandBuffer;</td>
</tr>
</tbody>
</table>

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.2.5.4 FB_DMXGetManufacturerLabel

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

Applying a positive edge to the bStart input starts the function 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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bStart</td>
<td>BOOL;</td>
</tr>
<tr>
<td>wDestinationManufacturerId</td>
<td>WORD;</td>
</tr>
<tr>
<td>dwDestinationDeviceId</td>
<td>DWORD;</td>
</tr>
<tr>
<td>byPortId</td>
<td>BYTE;</td>
</tr>
<tr>
<td>dwOptions</td>
<td>DWORD := 0;</td>
</tr>
</tbody>
</table>

bStart: The command is started by a positive edge at this input.
wDestinationManufacturerId: Unique manufacturer Id of the DMX device (for details, see DMX device address).

dwDestinationDeviceId: Unique device Id of the DMX device (for details, see DMX device address).

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**

<table>
<thead>
<tr>
<th>bBusy</th>
<th>BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bError</td>
<td>BOOL;</td>
</tr>
<tr>
<td>udiErrorId</td>
<td>UDINT;</td>
</tr>
<tr>
<td>sManufacturerLabel</td>
<td>STRING;</td>
</tr>
</tbody>
</table>

**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 [50]).

**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:** Reference to the structure for communication (buffer) with the block `FB_EL6851Communication()` [14] (see `ST_DMXCommandBuffer` [58]).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

**4.1.2.5.5  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**

<table>
<thead>
<tr>
<th>bStart</th>
<th>BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>wDestinationManufacturerId</td>
<td>WORD;</td>
</tr>
<tr>
<td>dwDestinationDeviceId</td>
<td>DWORD;</td>
</tr>
<tr>
<td>byPortId</td>
<td>BYTE;</td>
</tr>
<tr>
<td>dwOptions</td>
<td>DWORD := 0;</td>
</tr>
</tbody>
</table>

**bStart:** The command is started by a positive edge at this input.
wDestinationManufacturerId: Unique manufacturer Id of the DMX device (for details, see DMX device address).

dwDestinationDeviceId: Unique device Id of the DMX device (for details, see DMX device address).

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 function 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 [50]).

arrProductDetails: Contains the list with up to 6 device categories (see E_DMXProductDetail [53]).

VAR_IN_OUT

stCommandBuffer : ST_DMXCommandBuffer;

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.2.5.6 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 function 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 (for details, see DMX device address).

dwDestinationDeviceId: Unique device Id of the DMX device (for details, see DMX device address).

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 if 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 [50]).

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: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.2.5.7 FB_DMXSetDeviceLabel

This function block writes a description text into the DMX device. The function block FB_DMXGetDeviceLabel() [30] can be used to read the text.
**VAR_INPUT**

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

**bStart:** The block is activated 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 \[50\]).

**VAR_IN_OUT**

stCommandBuffer : ST_DMXCommandBuffer;

**stCommandBuffer:** Reference to the structure for communication (buffer) with the block **FB_EL6851Communication\[14\]** (see **ST_DMXCommandBuffer\[58\]**).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

**4.1.2.6** Queued and Status Messages

**4.1.2.6.1** **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 (for details, see DMX device address).

dwDestinationDeviceId: Unique device Id of the DMX device (for details, see DMX device address).

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 [50]).

VAR_IN_OUT

stCommandBuffer : ST_DMXCommandBuffer;

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.2.6.2 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 block is activated 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 function 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 [50]).

sStatusMessage: Status message.

VAR_IN_OUT

stCommandBuffer : ST_DMXCommandBuffer;

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.2.6.3 FB_DMXGetStatusMessages

This function block collects the status or error information of an DMX device.
VAR_INPUT

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

bStart: The block is activated 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 (see E_DMXStatusType [57]).

dwOptions: Options (currently not used).

VAR_OUTPUT

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

bBusy: When the function 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 [50]).

arrStatusMessages: Once command execution is complete (bBusy is FALSE), all status/error information is available at this output (see ST_DMXStatusMessage [61]).

VAR_IN_OUT

stCommandBuffer : ST_DMXCommandBuffer;

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.2.7 RDM Information Messages

4.1.2.7.1 FB_DMXGetParameterDescription
This function block queries the definition of manufacturer-specific PIDs.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>bStart</td>
</tr>
<tr>
<td>WORD</td>
<td>wDestinationManufacturerId</td>
</tr>
<tr>
<td>DWORD</td>
<td>dwDestinationDeviceId</td>
</tr>
<tr>
<td>BYTE</td>
<td>byPortId</td>
</tr>
<tr>
<td>E_DMXParameterId</td>
<td>eParameterId</td>
</tr>
<tr>
<td>DWORD</td>
<td>dwOptions</td>
</tr>
</tbody>
</table>

**bStart:** The block is activated 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:** Requested manufacturer-specific PID (see [E_DMXParameterId](#52)).

**dwOptions:** Options (currently not used).

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>bBusy</td>
</tr>
<tr>
<td>BOOL</td>
<td>bError</td>
</tr>
<tr>
<td>UDINT</td>
<td>udiErrorId</td>
</tr>
<tr>
<td>ST_DMXParameterDescription</td>
<td>stParameterDescription</td>
</tr>
</tbody>
</table>

**bBusy:** When the function 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](#50)).

**stParameterDescription:** Once command execution is complete (*bBusy* is FALSE), the PID information is available at this output (see [ST_DMXParameterDescription](#59)).

**VAR_IN_OUT**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_DMXCommandBuffer</td>
<td>stCommandBuffer</td>
</tr>
</tbody>
</table>

**stCommandBuffer:** Reference to the structure for communication (buffer) with the block [FB_EL6851Communication()](#14) (see [ST_DMXCommandBuffer](#58)).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

**4.1.2.7.2 FB_DMXGetSupportedParameters**

This function block queries all relevant information 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 (for details, see DMX device address).

dwDestinationDeviceId: Unique device Id of the DMX device (for details, see DMX device address).

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 function 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 [50]).

arrParameters: Once command execution is complete (bBusy is FALSE), all supported parameters for the DMX device are available at this output (see E_DMXParameterId [52]).

VAR_IN_OUT

stCommandBuffer : ST_DMXCommandBuffer;

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.2.8 Sensor Parameter Messages

4.1.2.8.1 FB_DMXGetSensorDefinition

This function block queries the definition of a particular sensor.
VAR_INPUT

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

bStart: The block is activated 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 function 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 [50]).

stSensorDefinition: Once the command execution is complete (bBusy is FALSE), the sensor definition is available at this output (see ST_DMXSensorDefinition [61]).

VAR_IN_OUT

stCommandBuffer : ST_DMXCommandBuffer;

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.2.8.2 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 block is activated 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;
stDMXSensorValue : ST_DMXSensorValue;

bBusy: When the function 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 [50]).

stDMXSensorValue: Structure with information about the current state of the sensor (see ST_DMXSensorValue [61]).

VAR_IN_OUT

stCommandBuffer : ST_DMXCommandBuffer;

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.1.2.9 Setup Messages

4.1.2.9.1 FB_DMXGetDMX512PersonalityDescription

FB_DMXGetDMX512PersonalityDescription

- bStart
- wDestinationManufacturerId
- dwDestinationDeviceId
- byPortId
- bySensorNumber
- stCommandBuffer
- bBusy
- bError
- udiErrorId
- stDMX512PersonalityDescription

TE1000 Version: 1.7
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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bStart</td>
<td>BOOL</td>
</tr>
<tr>
<td>wDestinationManufacturerId</td>
<td>WORD</td>
</tr>
<tr>
<td>dwDestinationDeviceId</td>
<td>DWORD</td>
</tr>
<tr>
<td>byPortId</td>
<td>BYTE</td>
</tr>
<tr>
<td>byPersonality</td>
<td>BYTE := 0;</td>
</tr>
<tr>
<td>dwOptions</td>
<td>DWORD := 0;</td>
</tr>
</tbody>
</table>

**bStart:** The block is activated 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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
<tr>
<td>udiErrorId</td>
<td>UDINT</td>
</tr>
<tr>
<td>stDMX512PersonalityDescription</td>
<td>ST_DMX512PersonalityDescription;</td>
</tr>
</tbody>
</table>

**bBusy:** When the function 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 [50]).

**stDMX512PersonalityDescription:** Structure with information on the personality (see ST_DMX512PersonalityDescription [58]).

**VAR_IN_OUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>stCommandBuffer</td>
<td>ST_DMXCommandBuffer;</td>
</tr>
</tbody>
</table>

**stCommandBuffer:** Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

**4.1.2.9.2 FB_DMXGetDMX512StartAddress**

![FB_DMXGetDMX512StartAddress](image)
This function block queries the DMX512 start address. This lies within the range from 1 – 512. If the DMX device does not occupy any DMX slot, then the DMX512 start address is 0xFFFF (65535). Each sub-device and the root device occupy different DMX512 start addresses.

Applying a positive edge to the \textit{bStart} input starts the function block, and the \textit{bBusy} output goes TRUE. The \textit{wDestinationManufacturerId} and \textit{dwDestinationDeviceId} inputs address the DMX device. The \textit{byPortId} input defines the channel within the addressed DMX device. If the execution of the command has been completed, the \textit{bBusy} output goes back to FALSE. The \textit{bError}, \textit{udiErrorId} and \textit{iDMX512StartAddress} outputs can now be processed. Further positive edges at the \textit{bStart} input will be ignored as long as the block is active (\textit{bBusy} is TRUE).

\textbf{VAR_INPUT}
\begin{verbatim}
  bStart : BOOL;
  wDestinationManufacturerId : WORD;
  dwDestinationDeviceId : DWORD;
  byPortId : BYTE;
  dwOptions : DWORD := 0;
\end{verbatim}

\textbf{VAR_OUTPUT}
\begin{verbatim}
  bBusy : BOOL;
  bError : BOOL;
  udiErrorId : UDINT;
  iDMX512StartAddress : INT;
\end{verbatim}

\textbf{VAR_IN_OUT}
\begin{verbatim}
  stCommandBuffer : ST_DMXCommandBuffer;
\end{verbatim}

\textbf{Requirements}
\begin{tabular}{|l|l|}
\hline
Development environment & required TC3 PLC library \\
\hline
TwinCAT from v3.1.4020.14 & Tc2_DMX from 3.5.3.0 \\
\hline
\end{tabular}
4.1.2.9.3   **FB_DMXGetSlotDescription**

This function block queries the text description for slot offsets.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bStart</td>
<td>BOOL</td>
</tr>
<tr>
<td>wDestinationManufacturerId</td>
<td>WORD</td>
</tr>
<tr>
<td>dwDestinationDeviceId</td>
<td>DWORD</td>
</tr>
<tr>
<td>byPortId</td>
<td>BYTE</td>
</tr>
<tr>
<td>iDMX512SlotOffset</td>
<td>INT</td>
</tr>
<tr>
<td>dwOptions</td>
<td>DWORD</td>
</tr>
</tbody>
</table>

- **bStart**: The block is activated 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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
<tr>
<td>udiErrorId</td>
<td>UDINT</td>
</tr>
<tr>
<td>sSlotDescription</td>
<td>STRING</td>
</tr>
</tbody>
</table>

- **bBusy**: When the function 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 [50]).
- **sSlotDescription**: Once the command execution is complete (`bBusy` is FALSE), the slot description (maximum 32 characters) is available at this output.

**VAR_IN_OUT**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>stCommandBuffer</td>
<td>ST_DMXCommandBuffer</td>
</tr>
</tbody>
</table>

- **stCommandBuffer**: Reference to the structure for communication (buffer) with the block `FB_EL6851Communication()` [14] (see ST_DMXCommandBuffer [58]).

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>
4.1.2.9.4 FB_DMXGetSlotInfo

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

VAR_INPUT

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bStart</td>
<td>The command is started by a positive edge at this input.</td>
</tr>
<tr>
<td>wDestinationManufacturerId</td>
<td>Unique manufacturer Id of the DMX device (for details, see DMX device address).</td>
</tr>
<tr>
<td>dwDestinationDeviceId</td>
<td>Unique device Id of the DMX device (for details, see DMX device address).</td>
</tr>
<tr>
<td>byPortId</td>
<td>Channel within the addressed DMX device. Sub-devices are addressed through the Port Id. The root device always has the Port Id 0.</td>
</tr>
<tr>
<td>dwOptions</td>
<td>Options (currently not used).</td>
</tr>
</tbody>
</table>

VAR_OUTPUT

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>When the function block is activated the output is set, and it remains active until execution of the command has been completed.</td>
</tr>
<tr>
<td>bError</td>
<td>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.</td>
</tr>
<tr>
<td>udiErrorId</td>
<td>Contains the command-specific error code of the most recently executed command. Only valid if bBusy is FALSE (see error codes [50]).</td>
</tr>
<tr>
<td>arrSlotInfos</td>
<td>Once the command execution is complete (bBusy is FALSE), all relevant information of the DMX512 slots is available as array at this output (see ST_DMXSlotInfo [61]).</td>
</tr>
</tbody>
</table>

VAR_IN_OUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stCommandBuffer</td>
<td>Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>
4.1.2.9.5   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 function 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

<table>
<thead>
<tr>
<th>Var Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bStart</td>
<td>BOOL</td>
</tr>
<tr>
<td>wDestinationManufacturerId</td>
<td>WORD</td>
</tr>
<tr>
<td>dwDestinationDeviceId</td>
<td>DWORD</td>
</tr>
<tr>
<td>byPortId</td>
<td>BYTE</td>
</tr>
<tr>
<td>iDMX512Startaddress</td>
<td>INT</td>
</tr>
<tr>
<td>dwOptions</td>
<td>DWORD</td>
</tr>
</tbody>
</table>

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

wDestinationManufacturerId: Unique manufacturer Id of the DMX device (for details, see DMX device address).

dwDestinationDeviceId: Unique device Id of the DMX device (for details, see DMX device address).

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

<table>
<thead>
<tr>
<th>Var Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
<tr>
<td>udiErrorId</td>
<td>UDINT</td>
</tr>
</tbody>
</table>

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 [50]).

VAR_IN_OUT

<table>
<thead>
<tr>
<th>Var Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>stCommandBuffer</td>
<td>ST_DMXCommandBuffer;</td>
</tr>
</tbody>
</table>

stCommandBuffer: Reference to the structure for communication (buffer) with the block FB_EL6851Communication() [14] (see ST_DMXCommandBuffer [58]).
### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>
### 4.1.3 Error codes

<table>
<thead>
<tr>
<th>Value (hex)</th>
<th>Value (dez)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>0</td>
<td>No error.</td>
</tr>
<tr>
<td>0x8001</td>
<td>32769</td>
<td>No answer from the DMX terminal.</td>
</tr>
<tr>
<td>0x8002</td>
<td>32770</td>
<td>No answer from the DMX device.</td>
</tr>
<tr>
<td>0x8003</td>
<td>32771</td>
<td>Communication buffer overflow.</td>
</tr>
<tr>
<td>0x8004</td>
<td>32772</td>
<td>No answer from the communication block.</td>
</tr>
<tr>
<td>0x8005</td>
<td>32773</td>
<td>The byPortId parameter is outside the valid range.</td>
</tr>
<tr>
<td>0x8006</td>
<td>32774</td>
<td>Checksum error.</td>
</tr>
<tr>
<td>0x8007</td>
<td>32775</td>
<td>The eResetDeviceType parameter is outside the valid range.</td>
</tr>
<tr>
<td>0x8008</td>
<td>32776</td>
<td>Timeout.</td>
</tr>
<tr>
<td>0x8009</td>
<td>32777</td>
<td>The uliLowerBoundUID parameter is larger than the uliUpperBoundUID parameter.</td>
</tr>
<tr>
<td>0x800A</td>
<td>32778</td>
<td>No RDM commands can be transmitted, because the terminal is in cycle mode.</td>
</tr>
<tr>
<td>0x800B</td>
<td>32779</td>
<td>The iDMX512StartAddress parameter is outside the valid range (1 – 512).</td>
</tr>
<tr>
<td>0x800C</td>
<td>32780</td>
<td>Error in setting the DMX512 start address.</td>
</tr>
<tr>
<td>0x800D</td>
<td>32781</td>
<td>No process data can be transmitted, because the terminal is not in cycle mode.</td>
</tr>
<tr>
<td>0x800E</td>
<td>32782</td>
<td>It is a RDM telegram received with the data length 0.</td>
</tr>
<tr>
<td>0x800F</td>
<td>32783</td>
<td>RDM response: Reply of the RDM telegram is invalid.</td>
</tr>
<tr>
<td>0x8010</td>
<td>32784</td>
<td>RDM response: The DMX slave cannot comply with request because the message is not implemented in responder.</td>
</tr>
<tr>
<td>0x8011</td>
<td>32785</td>
<td>RDM response: The DMX slave cannot interpret request as controller data was not formatted correctly.</td>
</tr>
<tr>
<td>0x8012</td>
<td>32786</td>
<td>RDM response: The DMX slave cannot comply due to an internal hardware fault.</td>
</tr>
<tr>
<td>0x8013</td>
<td>32787</td>
<td>RDM response: Proxy is not the RDM line master and cannot comply with message.</td>
</tr>
<tr>
<td>0x8014</td>
<td>32788</td>
<td>RDM response: SET Command normally allowed but being blocked currently.</td>
</tr>
<tr>
<td>0x8015</td>
<td>32789</td>
<td>RDM response: Not valid for Command Class attempted. May be used where GET allowed but SET is not supported.</td>
</tr>
<tr>
<td>0x8016</td>
<td>32790</td>
<td>RDM response: Value for given Parameter out of allowable range or not supported.</td>
</tr>
<tr>
<td>0x8017</td>
<td>32791</td>
<td>RDM response: Buffer or Queue space currently has no free space to store data.</td>
</tr>
<tr>
<td>0x8018</td>
<td>32792</td>
<td>RDM response: Incoming message exceeds buffer capacity.</td>
</tr>
<tr>
<td>0x8019</td>
<td>32793</td>
<td>RDM response: Sub-Device is out of range or unknown.</td>
</tr>
<tr>
<td>0x801A</td>
<td>32794</td>
<td>The iDMX512SlotOffset parameter is outside the valid range (0-511).</td>
</tr>
<tr>
<td>0x801B</td>
<td>32795</td>
<td>The bySensorNumber parameter is outside the valid range (0-254).</td>
</tr>
<tr>
<td>0x801C</td>
<td>32796</td>
<td>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 FB_EL6851CommunicationEx() must be used.</td>
</tr>
<tr>
<td>0x801D</td>
<td>32797</td>
<td>The ADS address to access the PDOs is invalid. Was the structure AdsAddr of the KL6851 mapped to the corresponding variable?</td>
</tr>
<tr>
<td>0x801E</td>
<td>32798</td>
<td>During read access to the PDOs an ADS error has occurred.</td>
</tr>
</tbody>
</table>
4.2 DUTs

4.2.1 Enums

4.2.1.1 E_DMXCommandClass

```plaintext
TYPE E_DMXCommandClass :
{
    eDMXCommandClassNotDefined := 16#0000,
    eDMXCommandClassDiscoveryCommand := 16#0010,
    eDMXCommandClassDiscoveryCommandResponse := 16#0011,
    eDMXCommandClassGetCommand := 16#0020,
    eDMXCommandClassGetCommandResponse := 16#0021,
    eDMXCommandClassSetCommand := 16#0030,
    eDMXCommandClassSetCommandResponse := 16#0031
};
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.2.1.2 E_DMXDataType

```plaintext
TYPE E_DMXDataType :
{
    eDMXDataTypeNotDefined := 16#0000,
    eDMXDataTypeBitField := 16#0001,
    eDMXDataTypeASCII := 16#0002,
    eDMXDataTypeUnsignedByte := 16#0003,
    eDMXDataTypeSignedByte := 16#0004,
    eDMXDataTypeUnsignedWord := 16#0005,
    eDMXDataTypeSignedWord := 16#0006,
    eDMXDataTypeUnsignedDWord := 16#0007,
    eDMXDataTypeSignedDWord := 16#0008
};
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.2.1.3 E_DMXLampOnMode

```plaintext
TYPE E_DMXLampOnMode :
{
    eDMXLampOnModeOff := 0,
    eDMXLampOnModeDMX := 1,
    eDMXLampOnModeOn := 2,
    eDMXLampOnModeAfterCal := 3
};
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>
4.2.1.4 E_DMXParameterDescriptionCommandClass

```
TYPE E_DMXParameterDescriptionCommandClass :
{
  eDMXParameterDescriptionCommandClassGet  := 16#0001,
  eDMXParameterDescriptionCommandClassSet  := 16#0002,
  eDMXParameterDescriptionCommandClassGetSet := 16#0003,
};
END_TYPE
```

4.2.1.5 E_DMXParameterId

```
TYPE E_DMXParameterId :
{
  eDMXParameterIdNone              := 16#0000,
  eDMXParameterIdDiscUniqueBranch := 16#0001,
  eDMXParameterIdDiscMute          := 16#0002,
  eDMXParameterIdDiscUnMute        := 16#0003,
  eDMXParameterIdProxiedDevices    := 16#0010,
  eDMXParameterIdProxiedDeviceCount := 16#0011,
  eDMXParameterIdCommsStatus       := 16#0015,
  eDMXParameterIdQueuedMessage     := 16#0020,
  eDMXParameterIdStatusMessages    := 16#0030,
  eDMXParameterIdStatusIdDescription := 16#0031,
  eDMXParameterIdClearStatusId     := 16#0032,
  eDMXParameterIdSubDeviceStatusReportThreshold := 16#0033,
  eDMXParameterIdSupportedParamaters := 16#0050,
  eDMXParameterIdParameterDescription := 16#0051,
  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,
  eDMXParameterIdDMXPersonality    := 16#00E0,
  eDMXParameterIdDMXPersonalityDescription := 16#00E1,
  eDMXParameterIdDMXStartAddress   := 16#00F0,
  eDMXParameterIdSlotInfo          := 16#0120,
  eDMXParameterIdSlotDescription   := 16#0121,
  eDMXParameterIdDefaultSlotValue  := 16#0122,
  eDMXParameterIdSensorDefinition   := 16#0200,
  eDMXParameterIdSensorValue       := 16#0201,
  eDMXParameterIdRecordSensors     := 16#0202,
  eDMXParameterIdDeviceHours       := 16#0400,
  eDMXParameterIdLampHours         := 16#0401,
  eDMXParameterIdLampStrikes       := 16#0402,
  eDMXParameterIdLampState         := 16#0403,
  eDMXParameterIdLampOnMode        := 16#0404,
  eDMXParameterIdDevicePowerCycles := 16#0405,
  eDMXParameterIdDisplayInvert     := 16#0500,
  eDMXParameterIdDisplayLevel      := 16#0501,
  eDMXParameterIdPanInvert         := 16#0600,
  eDMXParameterIdTiltInvert        := 16#0601,
  eDMXParameterIdPanTiltSwap       := 16#0602,
  eDMXParameterIdRealTimeClock     := 16#0603,
  eDMXParameterIdIdentifyDevice    := 16#1000,
  eDMXParameterIdResetDevice       := 16#1001,
  eDMXParameterIdPowerState        := 16#1010,
  eDMXParameterIdPerformSelftest   := 16#1020,
  eDMXParameterIdSelfTestDescription := 16#1021,
  eDMXParameterIdCapturePreset     := 16#1030,
  eDMXParameterIdPresetPlayBack    := 16#1031,
};
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.2.1.6 E_DMXProductDetail

TYPE E_DMXProductDetail :
{
  eDMXProductDetailNotDeclared := #0000,
  eDMXProductDetailArc := #0001,
  eDMXProductDetailMetalHalide := #0002,
  eDMXProductDetailIncandescent := #0003,
  eDMXProductDetailLED := #0004,
  eDMXProductDetailFluorescent := #0005,
  eDMXProductDetailColdcathode := #0006,
  eDMXProductDetailElectroluminescent := #0007,
  eDMXProductDetailLaser := #0008,
  eDMXProductDetailFlashtube := #0009,
  eDMXProductDetailColorscroller := #0100,
  eDMXProductDetailColorwheel := #0101,
  eDMXProductDetailColorchange := #0102,
  eDMXProductDetailIrisDouser := #0103,
  eDMXProductDetailDimmingShutter := #0104,
  eDMXProductDetailProfileShutter := #0105,
  eDMXProductDetailBarndoorShutter := #0106,
  eDMXProductDetailEffectsDisc := #0107,
  eDMXProductDetailGoboRotator := #0108,
  eDMXProductDetailVideo := #0200,
  eDMXProductDetailSlide := #0201,
  eDMXProductDetailFilm := #0202,
  eDMXProductDetailOilwheel := #0203,
  eDMXProductDetailLCDGate := #0204,
  eDMXProductDetailFoggerGlycol := #0300,
  eDMXProductDetailFoggerMineraloil := #0301,
  eDMXProductDetailFoggerWater := #0302,
  eDMXProductDetailCO2 := #0303,
  eDMXProductDetailLN2 := #0304,
  eDMXProductDetailBubble := #0305,
  eDMXProductDetailFlamePropane := #0306,
  eDMXProductDetailFlameOther := #0307,
  eDMXProductDetailOlefactoryStimulator := #0308,
  eDMXProductDetailSnow := #0309,
  eDMXProductDetailWaterJet := #030A,
  eDMXProductDetailWind := #030B,
  eDMXProductDetailConfetti := #030C,
  eDMXProductDetailHazard := #030D,
  eDMXProductDetailPhaseControl := #0400,
  eDMXProductDetailReversePhaseControl := #0401,
  eDMXProductDetailSine := #0402,
  eDMXProductDetailPWM := #0403,
  eDMXProductDetailDC := #0404,
  eDMXProductDetailHfballast := #0405,
  eDMXProductDetailHfvNeonballast := #0406,
  eDMXProductDetailHfve1 := #0407,
  eDMXProductDetailHfhvBallast := #0408,
  eDMXProductDetailBitangleModulation := #0409,
  eDMXProductDetailFrequencyModulation := #040A,
  eDMXProductDetailHighfrequency12V := #040B,
  eDMXProductDetailRelayMechanical := #040C,
  eDMXProductDetailRelayElectronic := #040D,
  eDMXProductDetailSwitchElectronic := #040E,
  eDMXProductDetailContactor := #040F,
  eDMXProductDetailMirrorballRotator := #0500,
  eDMXProductDetailOtherRotator := #0501,
  eDMXProductDetailKabukiDrop := #0502,
  eDMXProductDetailCurtain := #0503,
  eDMXProductDetailLineset := #0504,
  eDMXProductDetailMotorControl := #0505,
  eDMXProductDetailDamperControl := #0506,
  eDMXProductDetailSplitter := #0507,
  eDMXProductDetailEthernetNode := #0508,
  eDMXProductDetailMerge := #0509,
  eDMXProductDetailDatapatch := #050A,
  eDMXProductDetailWirelessLink := #050B,
  eDMXProductDetailProtocolConvertor := #050C,
eDMXProductDetailAnalogDemultiplex := 16#0702,
eDMXProductDetailAnalogMultiplex := 16#0703,
eDMXProductDetailSwitchPanel := 16#0704,
eDMXProductDetailRouter := 16#0800,
eDMXProductDetailFader := 16#0801,
eDMXProductDetailMixer := 16#0802,
eDMXProductDetailChangeoverManual := 16#0900,
eDMXProductDetailChangeoverAuto := 16#0901,
eDMXProductDetailTest := 16#0902,
eDMXProductDetailGfiRcd := 16#0A00,
eDMXProductDetailBattery := 16#0A01,
eDMXProductDetailControllableBreaker := 16#0A02,
eDMXProductDetailOther := 16#7FFF;
END_TYPE

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.2.1.7 E_DMXTresetDeviceType

TYPE E_DMXTResetDeviceType :
{
eDMXTResetDeviceTypeWarm := 1,
eDMXTResetDeviceTypeCold := 255
};
END_TYPE

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.2.1.8 E_DMXTResponseType

TYPE E_DMXTResponseType :
{
eDMXTResponseTypeAck := 16#0000,
eDMXTResponseTypeAckTimer := 16#0001,
eDMXTResponseTypeNackReason := 16#0002,
eDMXTResponseTypeAckOverflow := 16#0003
};
END_TYPE

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.2.1.9 E_DMXTSensorType

TYPE E_DMXTSensorType :
{
eDMXTSensorTypeTemperature := 16#00,
eDMXTSensorTypeVoltage := 16#01,
eDMXTSensorTypeCurrent := 16#02,
eDMXTSensorTypeFrequency := 16#03,
eDMXTSensorTypeResistance := 16#04,
eDMXTSensorTypePower := 16#05,
eDMXTSensorTypeMass := 16#06,
eDMXTSensorTypeLength := 16#07,
eDMXTSensorTypeArea := 16#08,
eDMXTSensorTypeVolume := 16#09,
eDMXTSensorTypeDensity := 16#0A,
eDMXTSensorTypeVelocity := 16#0B,
eDMXTSensorTypeAcceleration := 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.2.1.10 E_DMXSensorUnit

TYPE E_DMXSensorUnit :
{
  eDMXSensorUnitNone := 16#00,
eDMXSensorUnitCentigrade := 16#01,
eDMXSensorUnitVoltsDC := 16#02,
eDMXSensorUnitVoltsACPeak := 16#03,
eDMXSensorUnitVoltsACRms := 16#04,
eDMXSensorUnitAmpereDC := 16#05,
eDMXSensorUnitAmpereACPeak := 16#06,
eDMXSensorUnitAmpereACRms := 16#07,
eDMXSensorUnitHertz := 16#08,
eDMXSensorUnitOhm := 16#09,
eDMXSensorUnitWatt := 16#0A,
eDMXSensorUnitKilogram := 16#0B,
eDMXSensorUnitMeters := 16#0C,
eDMXSensorUnitMetersSquared := 16#0D,
eDMXSensorUnitMetersCubed := 16#0E,
eDMXSensorUnitKilogrammesPerMeterCubed := 16#0F,
eDMXSensorUnitMetersPerSecond := 16#10,
eDMXSensorUnitMetersPerSecondSquared := 16#11,
eDMXSensorUnitNewton := 16#12,
eDMXSensorUnitJoule := 16#13,
eDMXSensorUnitPascal := 16#14,
eDMXSensorUnitSecond := 16#15,
eDMXSensorUnitDegree := 16#16,
eDMXSensorUnitSteradian := 16#17,
eDMXSensorUnitCandela := 16#18,
eDMXSensorUnitLumen := 16#19,
eDMXSensorUnitLux := 16#1A,
eDMXSensorUnitIre := 16#1B,
eDMXSensorUnitByte := 16#1C
};
END_TYPE

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>
4.2.1.11  E_DMXSensorUnitPrefix

```cpp
TYPE E_DMXSensorUnitPrefix : (
  eDMXSensorUnitPrefixNone := 16#00,
  eDMXSensorUnitPrefixDeci := 16#01,
  eDMXSensorUnitPrefixCenti := 16#02,
  eDMXSensorUnitPrefixMilli := 16#03,
  eDMXSensorUnitPrefixMicro := 16#04,
  eDMXSensorUnitPrefixNano := 16#05,
  eDMXSensorUnitPrefixPico := 16#06,
  eDMXSensorUnitPrefixPempto := 16#07,
  eDMXSensorUnitPrefixAtto := 16#08,
  eDMXSensorUnitPrefixZepto := 16#09,
  eDMXSensorUnitPrefixYocto := 16#0A,
  eDMXSensorUnitPrefixDeca := 16#11,
  eDMXSensorUnitPrefixHecto := 16#12,
  eDMXSensorUnitPrefixKilo := 16#13,
  eDMXSensorUnitPrefixMega := 16#14,
  eDMXSensorUnitPrefixGiga := 16#15,
  eDMXSensorUnitPrefixTerra := 16#16,
  eDMXSensorUnitPrefixPeta := 16#17,
  eDMXSensorUnitPrefixExa := 16#18,
  eDMXSensorUnitPrefixZetta := 16#19,
  eDMXSensorUnitPrefixYotta := 16#1A
); END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.2.1.12  E_DMXSlotDefinition

```cpp
TYPE E_DMXSlotDefinition : (
  eDMXSlotDefinitionIntensity := 16#0001,
  eDMXSlotDefinitionIntensityMaster := 16#0002,
  eDMXSlotDefinitionPan := 16#0101,
  eDMXSlotDefinitionTilt := 16#0102,
  eDMXSlotDefinitionColorWheel := 16#0201,
  eDMXSlotDefinitionColorSubCyan := 16#0202,
  eDMXSlotDefinitionColorSubYellow := 16#0203,
  eDMXSlotDefinitionColorAddRed := 16#0205,
  eDMXSlotDefinitionColorAddGreen := 16#0206,
  eDMXSlotDefinitionColorAddBlue := 16#0207,
  eDMXSlotDefinitionColorCorrection := 16#0208,
  eDMXSlotDefinitionColorSemaphore := 16#0210,
  eDMXSlotDefinitionStaticGoboWheel := 16#0301,
  eDMXSlotDefinitionRotoGoboWheel := 16#0302,
  eDMXSlotDefinitionPrismWheel := 16#0303,
  eDMXSlotDefinitionEffectsWheel := 16#0304,
  eDMXSlotDefinitionBeamSizeIris := 16#0401,
  eDMXSlotDefinitionEdge := 16#0402,
  eDMXSlotDefinitionFrost := 16#0403,
  eDMXSlotDefinitionStrobe := 16#0404,
  eDMXSlotDefinitionZoom := 16#0405,
  eDMXSlotDefinitionFramingShutter := 16#0406,
  eDMXSlotDefinitionShutterRotate := 16#0407,
  eDMXSlotDefinitionDouser := 16#0408,
  eDMXSlotDefinitionBarnDoor := 16#0409,
  eDMXSlotDefinitionLampControl := 16#0501,
  eDMXSlotDefinitionFixtureControl := 16#0502,
  eDMXSlotDefinitionFixtureSpeed := 16#0503,
  eDMXSlotDefinitionMacro := 16#0504,
  eDMXSlotDefinitionUndefined := 16#FFFF
); END_TYPE
```
4.2.13 E_DMXSlotType

```pascal
TYPE E_DMXSlotType : 
{
  eDMXSlotTypePrimary := 0,
  eDMXSlotTypeSecFine := 1,
  eDMXSlotTypeSecTiming := 2,
  eDMXSlotTypeSecSpeed := 3,
  eDMXSlotTypeSecControl := 4,
  eDMXSlotTypeSecIndex := 5,
  eDMXSlotTypeSecRotation := 6,
  eDMXSlotTypeSecIndexRotate := 7,
  eDMXSlotTypeSecUndefined := 255
};
END_TYPE
```

4.2.14 E_DMXStatusType

```pascal
TYPE E_DMXStatusType : 
{
  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
```

4.2.2 Structures

4.2.2.1 ST_DMX512Personality

```pascal
TYPE ST_DMX512Personality :
STRUCT
  byCurrentPersonality : BYTE;
  byTotalPersonalities : BYTE;
END_STRUCT
END_TYPE
```
4.2.2.2 ST_DMX512PersonalityDescription

TYPE ST_DMX512PersonalityDescription :
STRUCT
  iDMX512SlotsRequired : INT;
  sDescription         : STRING;
END_STRUCT
END_TYPE

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.2.2.3 ST_DMXCommandBuffer

TYPE ST_DMXCommandBuffer :
STRUCT
  arrMessageQueue     : ST_DMXMessageQueue;
  stResponseTable     : ST_DMXResponseTable;
  byTransactionNumber : BYTE;
END_STRUCT
END_TYPE

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.2.2.4 ST_DMXDeviceInfo

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

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.2.2.5 ST_DMXMac

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

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>
4.2.2.6  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

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.2.2.7  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

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.2.2.8  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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>
4.2.2.9 ST_DMXPProductCategory

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

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.2.2.10 ST_DMXRDMProtocolVersion

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

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.2.2.11 ST_DMXPResponseTable

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

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.2.2.12 ST_DMXPResponseTableItem

TYPE ST_DMXPResponseTableItem:
  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
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

**4.2.2.13 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

**4.2.2.14 ST_DMXSensorValue**

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

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

**4.2.2.15 ST_DMXSlotInfo**

```
TYPE ST_DMXSlotInfo :
   STRUCT
     bEntryIsValid : BOOL;
     eSlotType : E_DMXSlotType;
     eSlotDefinition : E_DMXSlotDefinition;
   END_STRUCT
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

**4.2.2.16 ST_DMXStatusMessage**

```
TYPE ST_DMXStatusMessage :
   STRUCT
     bEntryIsValid : BOOL;
     iSubDeviceId : INT;
     eStatusType : E_DMXStatusType;
   END_STRUCT
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>
iStatusMessageId : INT;
iDataValue01 : INT;
iDataValue02 : INT;
END_STRUCT
END_TYPE

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

### 4.2.2.17 ST_EL6851AdsAddr

TYPE ST_EL6851AdsAddr :
STRUCT
  arrNetId : ARRAY [0..5] OF USINT;
iPort   : UINT;
END_STRUCT
END_TYPE

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

### 4.2.2.18 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

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

### 4.2.2.19 ST_EL6851InDataEx

TYPE ST_EL6851InDataEx :
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;
bWcState         : BOOL;
bInputToggle     : BOOL;
uiState          : UINT;
stAdsAddr        : ST_EL6851AdsAddr;
END_STRUCT
END_TYPE
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.2.2.20 ST_EL6851OutData

```plaintext
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
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>required TC3 PLC library</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT from v3.1.4020.14</td>
<td>Tc2_DMX from 3.5.3.0</td>
</tr>
</tbody>
</table>

4.3 Integration into TwinCAT

4.3.1 EL6851 with CX5120

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

Sample: https://infosys.beckhoff.com/content/1033/tcplclib_tc2_dmx/Resources/zip/6202185099.zip

Hardware

Setting up the components

The following hardware is required:

- 1x CX5120 Embedded PC
- 1x EL6851 DMX Master terminal
- 1x EL9011 end cap

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

Software

Creation of the PLC program

Create a new "TwinCAT XAE project" and a "Standard PLC project".

Add the library Tc2_DMX in the PLC project under "References".

Generate a global variable list with the name GVL_EIB and create the following variables:

```plaintext
var_global
  stEL6851InData   AT %I*: ST_EL6851InDataEx;
  stEL6851OutData  AT %Q*: ST_EL6851OutData;
  stCommandBuffer  : ST_DMXCommandBuffer;
end_var
```
stEL6851InData : Input variable for the DMX terminal (see ST_EL6851InDataEx [62]).

stEL6851OutData : Output variable for the DMX terminal (see ST_EL6851OutData [63]).

stCommandBuffer: Required for the communication with DMX (see ST_DMXCommandBuffer [58]).

Then create a program (CFC) for background communication with DMX. The function block FB_EL6851CommunicationEx [16] is called in this program. Make sure to link the communication function block with stEL6851InData, stEL6851OutData and stCommandBuffer.

Create a MAIN program (CFC) in which the function block FB_DMXDiscovery [10] is called up. The input stCommandBuffer of the function block is linked with the global variable stCommandBuffer. If the variable bStart is TRUE, the search for DMX devices is started. The number of found devices is stored in variable iFoundDevices and additional information in arrDMXDeviceInfoList.

Navigate to the task configuration section and configure the PlcTask. By way of example, the task is assigned priority 16 and a cycle time of 6 ms.

Create a further task for the background communication. Assign a higher priority (smaller number) and a lower interval time to this task than the PlcTask.
Add the program for the communication to this task. Further information on the task configuration can be found in the description of the function block FB_EL6851CommunicationEx [16].

I/O configuration

Select the CX as target system and initiate a search for its 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.
In the project instance within the PLC section, you can see that the input and output variables are assigned to the corresponding tasks.

Link the global variables of the PLC program with the inputs and outputs of the EtherCAT Terminal.
Create the Solution and enable the configuration.
5 Appendix

5.1 Example: Configuration by RDM

Configuration of DMX slaves via Remote Device Management (RDM)


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.

EtherCAT functionality

The dialogue is only functioning when the 'EtherCAT OK' LED lights up green. A red LED indicates a fault in the EtherCAT communication.
5.2 Example: DMX master

Sending the cyclic process data as a DMX master (EL6851)


Preparation

The application samples 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 were used for the test configuration:

- TwinCAT master PC with Windows 7 operating system, TwinCAT version 3.1 (Build 4014.2) and TwinCAT Intel PCI Ethernet adapter (Gigabit).
- Beckhoff EtherCAT coupler EK1100, EL6851 and EL9011 terminals
- RGB-LED DMX slave with 3 channels (one for each color). One slot is occupied per channel.

Starting the sample program

1. Save the https://infosys.beckhoff.com/content/1033/tcplclib_tc2_dmx/Resources/zip/18014399039931019.zip on your hard disk and unzip it.
2. Open the file DMX_Master.sln with TwinCAT XAE.
3. Connect the hardware accordingly and connect the Ethernet adapter of your PCs to the EtherCAT coupler.
4. Select the local Ethernet adapter (with real-time driver, if required) under DMX_Demo > I/O > Devices > Device 1 (EtherCAT). Then select the corresponding adapter under “Search...” in the “Adapter” tab and actuate it.

![Image of TwinCAT configuration interface]

*Solution Explorer*
5. Activate the configuration and confirm.


7. Compile the project in TwinCAT XAE by selecting the command “Build DMX_Demo” in the “Build” menu.
8. Log in and confirm loading of the program.

9. Start the program.

**Visualization**

Specification of the control values for the three colors of the DMX slave in TwinCAT XAE:
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 colors. If the “Start/Stop” button is pressed, then automatically generated data are 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 horizontal sliders.

5.3 Example: DMX slave

Receipt of 64 bytes of data by each of two DMX slaves (EL6851-0010)


Arrays with 64 bytes each in full configuration (all PDOs selected):
An EL6851-0010 can read max. 512 bytes (64 bytes in each one of eight arrays, see fig. 1). The arrays can be assigned in the TwinCAT XAE (Process data tab) via the PDO 0x1C13.

Example:

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

DMX channels 1 - 64 (default) by selection of PDO 0x1A01:

DMX channels 65 - 128 by selection of PDO 0x1A02:
In the example program the first DMX Slave receives the first 64 bytes sent and the second slave the next 64 bytes (figs. 2 + 3; it is also possible for one EL6851-0010 to receive the entire 128 bytes, the division is chosen deliberately in the example).

### DMX status object:

- **Receive toggle**
- **Framing error**
- **Overrun error**
- **TxPDO State**
- **TxPDO Toggle**
- **Channel length**
- **Start code**

In the DMX status object (index 0x6000, *DMX status*, Fig. 4) a copy counter is created with index 0x6000:11 (channel length).

**Example:**

If PDO 0x1A01 is activated, the value of *Channel length* is 64\textsubscript{dec}. If PDO 0x1A02 is activated, the value is 128\textsubscript{dec}. If both PDOS are activated (0x1A01 and 0x1A02) the value is likewise 128\textsubscript{dec}.

- **Watchdog DMX Slave 1**
- **Watchdog DMX Slave 2**

DMX slave 1 receives 64 bytes of data on channel 1 of the first array (*DMX channels 1 - 64*)

DMX slave 2 receives 64 bytes of data on channel 1 of the second array (*DMX channels 65 - 128*)

The *Receive toggle* bit (index 0x6000:02) is evaluated and displayed in each case via the FB *bMonitorToggleBit* (Watchdog DMX Slave).
5.4 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