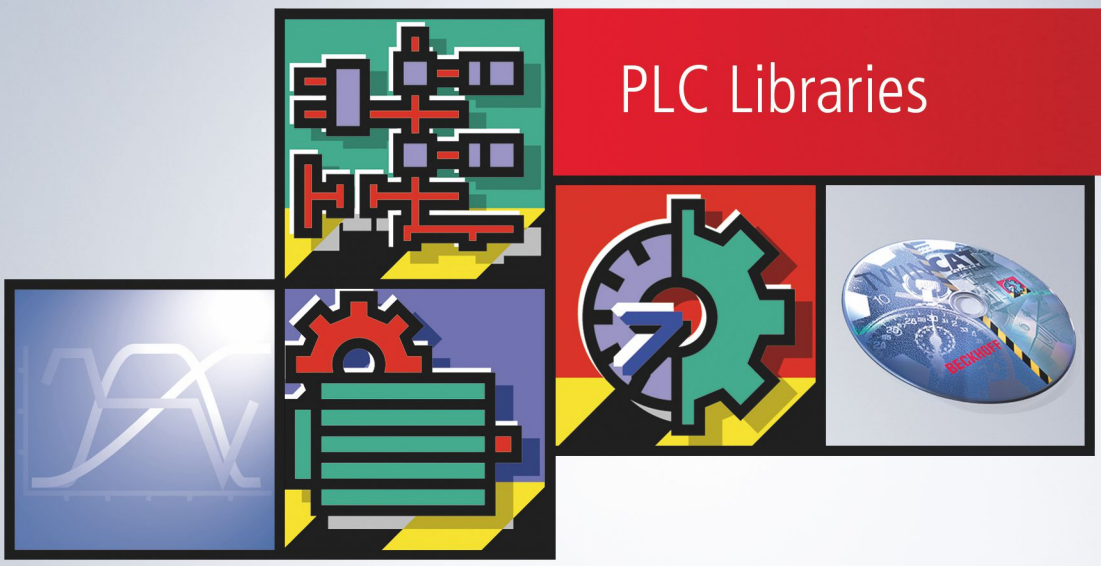
BECKHOFF New Automation Technology

Manual | EN

TX1200

TwinCAT 2 | PLC Library: TcMDP



2022-11-02 | Version: 1.1

Table of contents

1 Foreword

1.1 Notes on the documentation

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

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

Disclaimer

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

Trademarks

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

Patent Pending

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

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

Copyright

1.2 Safety instructions

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

⚠ DANGER

Serious risk of injury!

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

⚠ WARNING

Risk of injury!

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

⚠ CAUTION

Personal injuries!

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

NOTE

Damage to the environment or devices

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

Tip or pointer

This symbol indicates information that contributes to better understanding.

1.3 Notes on information security

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

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

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

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

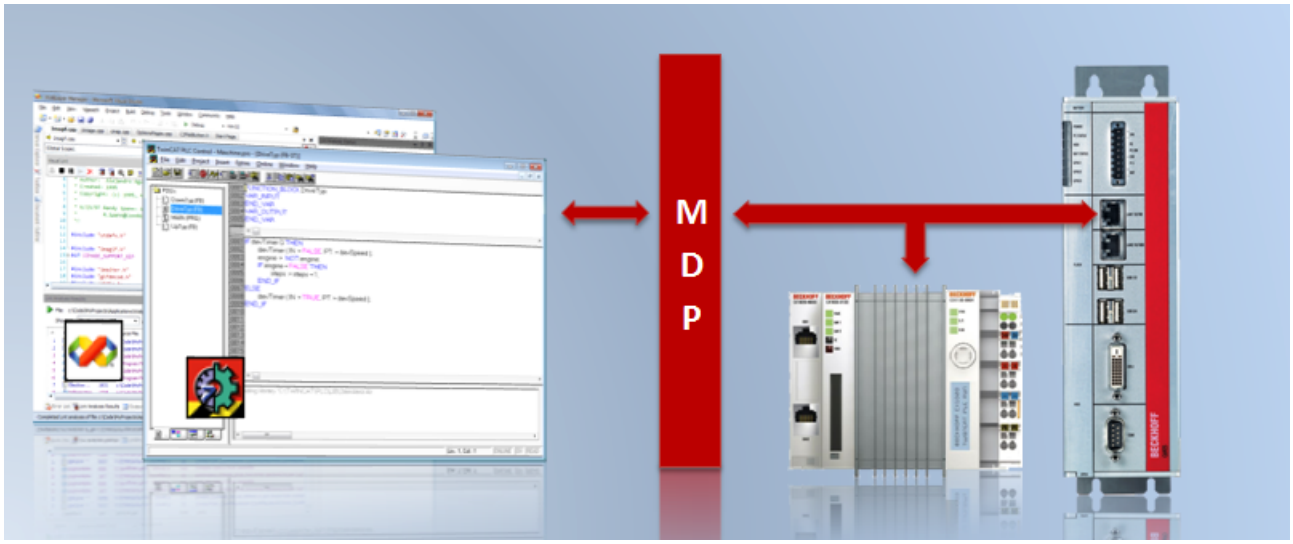
2 Overview

MDP (Modular Device Profile) information can be accessed from the PLC using the PLC function block.

The **Modular Device Profile for IPC** (MDP) is based on the Modular Device Profile specification of the EtherCAT Technology Group. All (software and hardware) components of the Industrial PC or Embedded PC are subdivided into modules. The list of available modules is generated dynamically depending upon the physically existing components.

The MDP standardises the method of accessing Beckhoff hardware and software, which up to now could be different depending on the Windows operating system used.

This documentation refers to the TwinCAT PLC library TcMDP, with which MDP information can be queried from the PLC. Further information on the general MDP and other interfaces is offered by the documentation on Beckhoff Device Manager.



System requirements

- Programming environment:
  - TwinCAT installation level: TwinCAT PLC or higher;
  - TwinCAT system version 2.11.0 build 1553 or higher; alternative: TwinCAT system version 2.11.0 R2 build 2025 or higher
  - **TcMDP.Lib** This PLC library must be integrated in the PLC project. All other libraries are added automatically. (Standard.Lib; TcBase.Lib; TcSystem.Lib; TcUtilities.Lib are included automatically)
- Target platform
  - PC or CX (x86): XP, XPe, CE (image v3.21c or higher);
  - CX (ARM): CE (image v3.21c or higher);
  - C69xx / CP62xx: CE (image v3.21f or higher);
  - TwinCAT PLC runtime system version 2.11.0 build 1541 or higher;
  - The system requirements of the Beckhoff Device Manager has to be attended too.

Further documentation

- Documentation on Beckhoff Device Manager

3 MDP element access options

The TwinCAT PLC MDP library offers the most diverse function blocks, to enable extensive access to MDP data.

There are two basic types of function blocks in the library.

On the one hand the general function blocks. They can be used to query and set arbitrary parameters in the MDP themselves by means of discrete access.

Furthermore, specific function blocks offer the possibility of accessing certain data as well as groupings of several data with one call. The function blocks available here offer fast access to the most important MDP information.

The type of MDP access and the differences between the two types of function blocks will be described in greater detail below. All function blocks have a uniform look & feel.

All function blocks are called by a rising edge on the *bExecute* input. Afterwards, cyclic calling of the function block (*bExecute* = FALSE) returns the result of the query at the output as soon as the processing of the query has been completed (*bBusy* = FALSE). The example [▶ 37] in this documentation supplies further handling tips. Each function block must be called (*bExecute* = FALSE) for as long as it takes for the internal processing (*bBusy* = FALSE) to be completed. During that time, all inputs of the function block must remain unchanged.

In general, the MDP is a model that describes hardware and software components in the form of modules. Information about these modules as well as about the device itself can be queried and changed.

A module consists of one or more tables. Each table consists of a fixed number of subindices. A subindex corresponds to a concrete element that can be accessed.

More detailed information about the structure of the MDP can be found in the MDP Information Model. Further options for accessing the MDP are also described there.

General function blocks

In order to be able to query or set an MDP parameter, the Dynamic Module ID of the module in which the parameter is located must be known.

This is determined with the aid of the function block FB_MDP_ScanModules [▶ 18].

Individual parameters can now be read or written by means of FB_MDP_Read [▶ 11] and FB_MDP_Write [▶ 13]. In addition to the dynamic Module ID, the number of the selected table (Table ID), the selected subindex within the table as well as further information is thereby specified for the query.

Likewise, the complete header of a module (ST_MDP_ModuleHeader [▶ 31]) can be queried with the function block FB_MDP_ReadModuleHeader [▶ 18].

The complete contents of a selected table within a module can be queried with the function block FB_MDP_ReadModuleContent [▶ 16].

The function block FB_MDP_ReadModule [▶ 15] bundles the above queries. The function block implicitly determines the Dynamic Module ID and queries both header and table.

The function block FB_MDP_ReadElement [▶ 13] also implicitly determines the Dynamic Module ID. Every single MDP element can be selected and queried.

For these both function blocks is a previous call of FB_MDP_ScanModules not necessary.

Specific function blocks

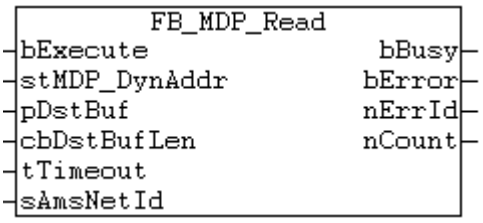
The function blocks available here offer fast access to the most important MDP information.

For example, calling the function block FB_MDP_NIC_Read [▶ 23] suffices in order to query all important information about a network adapter (see MDP NIC module). The module header is also queried and output in each case.

The specific function blocks likewise implicitly determine the dynamic Module ID, so that a prior call of FB_MDP_ScanModules [▶ 18] is superfluous.

4 Function blocks

4.1 FB_MDP_Read



The function block enables an element of an MDP module to be queried.

VAR_INPUT

```
VAR_INPUT
    bExecute        : BOOL;             (* Function block execution is triggered by a rising edge at
 this input.*)
    stMDP_DynAddr   : ST_MDP_Addr;
    pDstBuf         : POINTER TO BYTE;              (* Contains the address of the buffer for the re
ceived data. *)
    cbDstBufLen     : UDINT;                (* Contains the max. number of bytes to be received. *)
    tTimeout        : TIME := DEFAULT_ADS_TIMEOUT;  (* States the time before the function is cancel
led. *)
    sAmsNetId       : T_AmsNetId;           (* keep empty '' for the local device *)
END_VAR
```

bExecute	The function block is called by a rising edge on the input *bExecute*, if the block is not already active.
stMDP_DynAddr	The MDP addressing belonging to the selected network module is specified at this input. The structure is of the type ST_MDP_Addr [▶ 30]. The dynamic Module ID must already be specified with it.
pDstBuf	The memory address of the data buffer is specified at this input. The received data are stored there if the query is successful.
cbDstBufLen	The length of the data buffer in bytes is specified at this input.
tTimeout	Specifies a maximum length of time for the execution of the function block.
sAmsNetId	For local access don't specify this input or allocate an empty string. For remote access to another computer specify its AMS Net Id.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    nCount      : UDINT;
END_VAR
```

bBusy	This output is TRUE as long as the function block is active.
bError	Becomes TRUE as soon as an error situation occurs.
nErrID	Returns an error code [▶ 34] if the bError output is set.
nCount	This output indicates the number of bytes read.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib

4.2 FB_MDP_ReadIndex

The function block enables an element of the IPC diagnosis data to be queried. This can be part of the configuration area as well as the device area.

VAR_INPUT

```
VAR_INPUT
    bExecute        : BOOL;             (* Function block execution is triggered by a rising edge at
 this input.*)
    nIndex          : WORD;
    nSubIndex       : BYTE;
    pDstBuf         : POINTER TO BYTE;          (* Contains the address of the buffer for the receiv
ed data. *)
    cbDstBufLen     : UDINT;            (* Contains the max. number of bytes to be received. *)
    tTimeout        : TIME := DEFAULT_ADS_TIMEOUT;  (* States the time before the function is cancel
led. *)
    sAmsNetId       : T_AmsNetId;           (* keep empty '' for the local device *)
END_VAR
```

bExecute	The function block is called by a rising edge on the input *bExecute*, if the block is not already active.
nIndex	The first part of the addressing belonging to the selected element is specified at this input.
nSubIndex	The second part of the addressing belonging to the selected element is specified at this input.
pDstBuf	The memory address of the data buffer is specified at this input. The received data are stored there if the query is successful.
cbDstBufLen	The length of the data buffer in bytes is specified at this input.
tTimeout	Specifies a maximum length of time for the execution of the function block.
sAmsNetId	For local access don't specify this input or allocate an empty string. For remote access to another computer specify its AMS Net Id.

VAR_OUTPUT

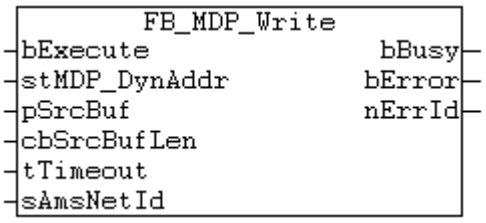
```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    nCount      : UDINT;
END_VAR
```

bBusy	This output is TRUE as long as the function block is active.
bError	Becomes TRUE as soon as an error situation occurs.
nErrID	Returns an error code [▶ 34] if the bError output is set.
nCount	This output indicates the number of bytes read.

Requirements

Development environment	Target system type	PLC libraries to include
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib [version 1.4.0 or higher]

4.3 FB_MDP_Write



The function block enables the MDP TwinCAT module to be queried.

VAR_INPUT

```
VAR_INPUT
    bExecute        : BOOL;             (* Function block execution is triggered by a rising edge at
 this input.*)
    stMDP_DynAddr   : ST_MDP_Addr;
    pSrcBuf     : POINTER TO BYTE;              (* Contains the address of the buffer for the sent d
ata. *)
    cbSrcBufLen     : UDINT;                (* Contains the max. number of bytes to be sent. *)
    tTimeout        : TIME := DEFAULT_ADS_TIMEOUT;  (* States the time before the function is cancel
led. *)
    sAmsNetId       : T_AmsNetId;           (* keep empty '' for the local device *)
END_VAR
```

bExecute	The function block is called by a rising edge on the input *bExecute*, if the block is not already active.
stMDP_DynAddr	The MDP addressing belonging to the selected network module is specified at this input. The structure is of the type ST_MDP_Addr [▶ 30]. The dynamic Module ID must already be specified with it.
pSrcBuf	The memory address of the data buffer is specified at this input. The data to be transmitted must be stored there.
cbSrcBufLen	The length of the data buffer in bytes is specified at this input.
tTimeout	Specifies a maximum length of time for the execution of the function block.
sAmsNetId	For local access don't specify this input or allocate an empty string. For remote access to another computer specify its AMS Net Id.

VAR_OUTPUT

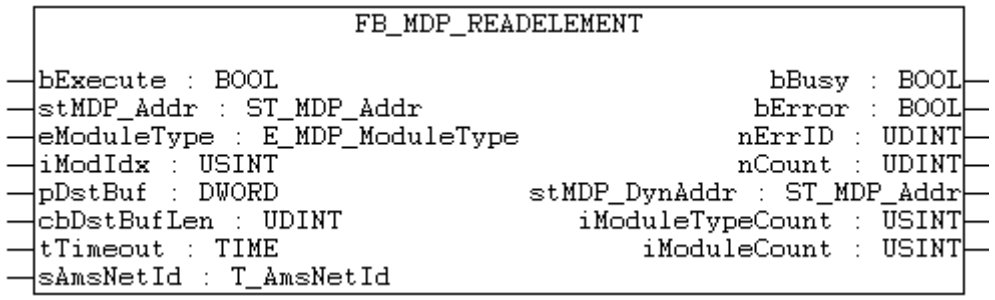
```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
END_VAR
```

bBusy	This output is TRUE if the function block is active.
bError	Becomes TRUE as soon as an error situation occurs.
nErrID	Returns an error code [▶ 34] if the bError output is set.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib

4.4 FB_MDP_ReadElement



The function block enables an individual MDP element to be queried. Every element out of every module of the Configuration Area can be read!
The device is scanned internally for the selected module and the element information are queried with the dynamic Module ID.

VAR_INPUT

```
VAR_INPUT
    bExecute        : BOOL;
    stMDP_Addr      : ST_MDP_Addr;            (* includes all address parameters without the Dynamic
 Module Id *)
    eModuleType     : E_MDP_ModuleType;      (* chosen module type out of the module type list *)
    iModIdx         : USINT;                 (* chosen index(0..n) of the demanded module type. E.g.
second NIC(idx 1) of three found NICs. *)
    pDstBuf         : POINTER TO BYTE;              (* Contains the address of the buffer for the re
ceived data. *)
    cbDstBufLen     : UDINT;                (* Contains the max. number of bytes to be received. *)
    tTimeout        : TIME  := DEFAULT_ADS_TIMEOUT; (* States the time before the function is cancel
led. *)
    sAmsNetId       : T_AmsNetId;          (* keep empty '' for the local device *)
END_VAR
```

bExecute	The function block is called by a rising edge on the input *bExecute*, if the block is not already active.
stMDP_Addr	The MDP addressing belonging to the selected module is specified at this input. The structure is of the type ST_MDP_Addr [▶ 30]. The area has to be the configuration area. The dynamic Module ID is only added internally and must not be allocated.
eModuleType	The MDP module type is specified at this input. The possible types are listed in the enumeration E_MDP_ModuleType [▶ 30]. (General information on the module type list
iModIdx	If several instances of an MDP module exist, a selection can be made by means of the input *iModIdx* (0,...,n).
pDstBuf	The memory address of the data buffer is specified at this input. The received data are stored there if the query is successful.
cbDstBufLen	The length of the data buffer in bytes is specified at this input.
tTimeout	Specifies a maximum length of time for the execution of the function block.
sAmsNetId	For local access don't specify this input or allocate an empty string. For remote access to another computer specify its AMS Net Id.

VAR_OUTPUT

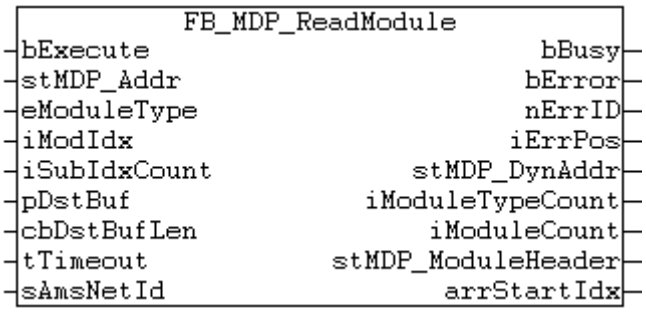
```
VAR_OUTPUT
    bBusy           : BOOL;
    bError          : BOOL;          (* indicates if Read was successfull or not *)
    nErrID          : UDINT;
    nCount          : UDINT;
    stMDP_DynAddr   : ST_MDP_Addr;        (* includes the new dynamic module type id. *)
    iModuleTypeCount: USINT;             (* returns the number of found modules equal the demanded mo
dule type. *)
    iModuleCount    : USINT;             (* returns the number of all detected MDP modules. *)
END_VAR
```

bBusy	This output is TRUE as long as the function block is active.
bError	Becomes TRUE as soon as an error situation occurs.
nErrID	Returns an error code [▶ 34] if the bError output is set.
nCount	This output indicates the number of bytes read.
stMDP_DynAddr	The MDP addressing belonging to the selected MDP module is specified at this output. The structure is of the type ST_MDP_Addr [▶ 30]. The dynamic Module ID was added by the function block.
iModuleTypeCount	The output *iModuleTypeCount* indicates the number of modules that correspond to the specified type.
iModuleCount	The output *iModuleCount* indicates the entire number of modules on the device.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib [version 1.2.0 or higher]

4.5 FB_MDP_ReadModule



The function block enables an MDP module to be queried.
The device is scanned internally for the selected module and the module header, and the module information are queried with the dynamic Module ID.

VAR_INPUT

```
VAR_INPUT
    bExecute        : BOOL;
    stMDP_Addr      : ST_MDP_Addr;            (* includes all address parameters without the Dynamic
 Module Id *)
    eModuleType     : E_MDP_ModuleType;      (* chosen module type out of the module type list *)
    iModIdx         : USINT;                (* chosen index(0..n) of the demanded module type. E.g.
second NIC(idx 1) of three found NICs. *)
    iSubIdxCount    : USINT;
    pDstBuf         : POINTER TO BYTE;              (* Contains the address of the buffer for the re
ceived data. *)
    cbDstBufLen     : UDINT;                (* Contains the max. number of bytes to be received. *)
    tTimeout        : TIME  := DEFAULT_ADS_TIMEOUT; (* States the time before the function is cancel
led. *)
    sAmsNetId       : T_AmsNetId;          (* keep empty '' for the local device *)
END_VAR
```

bExecute	The function block is called by a rising edge on the input *bExecute*, if the block is not already active.
stMDP_Addr	The MDP addressing belonging to the selected module is specified at this input. The structure is of the type ST_MDP_Addr [▶ 30]. The dynamic Module ID is only added internally.
eModuleType	The MDP module type is specified at this input. The possible types are listed in the enumeration E_MDP_ModuleType [▶ 30]. (General information on the module type list
iModIdx	If several instances of an MDP module exist, a selection can be made by means of the input *iModIdx* (0,...,n).

iSubIdxCount	The input *iSubIdxCount* is used to specify how many subindices of the selected Table ID are to be queried.
pDstBuf	The memory address of the data buffer is specified at this input. The received data are stored there if the query is successful.
cbDstBufLen	The length of the data buffer in bytes is specified at this input.
tTimeout	Specifies a maximum length of time for the execution of the function block.
sAmsNetId	For local access don't specify this input or allocate an empty string. For remote access to another computer specify its AMS Net Id.

VAR_OUTPUT

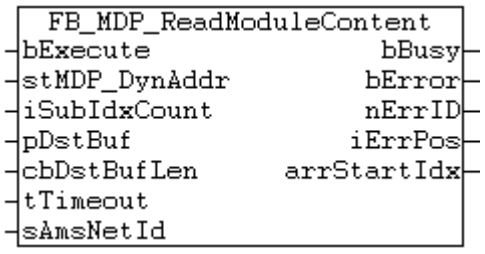
```
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;         (* indicates if Read was successfull or not *)
    nErrID              : UDINT;
    iErrPos             : USINT;
    stMDP_DynAddr       : ST_MDP_Addr;        (* includes the new dynamic module type id. *)
    iModuleTypeCount    : USINT;            (* returns the number of found modules equal the demande
d module type. *)
    iModuleCount        : USINT;            (* returns the number of all detected MDP modules. *)
    stMDP_ModuleHeader  : ST_MDP_ModuleHeader;
    arrStartIdx         : ARRAY[0..255] OF UINT;    (* startindexes in bytes of each subindex elemen
t *)
END_VAR
```

bBusy	This output is TRUE as long as the function block is active.
bError	Becomes TRUE as soon as an error situation occurs.
nErrID	Returns an error code [▶ 34] if the bError output is set.
iErrPos	If an error occurred and this refers to an individual element, then this output indicates the position (subindex of the element) at which an error first occurred.
stMDP_DynAddr	The MDP addressing belonging to the selected MDP module is specified at this output. The structure is of the type ST_MDP_Addr [▶ 30]. The dynamic Module ID was added by the function block.
iModuleTypeCount	The output *iModuleTypeCount* indicates the number of modules that corresponds to the specified type.
iModuleCount	The output *iModuleCount* indicates the entire number of modules on the device.
stMDP_ModuleHeader	The header information from the read MDP module is displayed at this output in the form of the structure ST_MDP_ModuleHeader [▶ 31].
arrStartIdx	This array describes how the individually queried subindices have been stored in the buffer. The array index zero indicates the position in bytes at which the data of subindex zero begins in the buffer. Subsequent subindices are handled analogously.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib

4.6 FB_MDP_ReadModuleContent



The function block enables the contents of an MDP module to be queried.

VAR_INPUT

```
VAR_INPUT
    bExecute        : BOOL;
    stMDP_DynAddr   : ST_MDP_Addr;            (* includes the dynamic module type for which the modu
le content is requested. All subindexes of the chosen table are requested. *)
    iSubIdxCount    : USINT;                (* the number of SubIndexes to be requested *)
    pDstBuf         : POINTER TO BYTE;              (* Contains the address of the buffer for the re
ceived data. *)
    cbDstBufLen     : UDINT;                (* Contains the max. number of bytes to be received. *)
    tTimeout        : TIME  := DEFAULT_ADS_TIMEOUT; (* States the time before the function is cancel
led. *)
    sAmsNetId       : T_AmsNetId;          (* keep empty '' for the local device *)
END_VAR
```

bExecute	The function block is called by a rising edge on the input *bExecute*, if the block is not already active.
stMDP_DynAddr	The MDP addressing belonging to the selected module is specified at this input. The structure is of the type ST_MDP_Addr [▶ 30]. The dynamic Module ID must already be transferred with it.
iSubIdxCount	The input *iSubIdxCount* is used to specify how many subindices of the selected Table ID are to be queried.
pDstBuf	The memory address of the data buffer is specified at this input. The received data are stored there if the query is successful.
cbDstBufLen	The length of the data buffer in bytes is specified at this input.
tTimeout	Specifies a maximum length of time for the execution of the function block.
sAmsNetId	For local access don't specify this input or allocate an empty string. For remote access to another computer specify its AMS Net Id.

VAR_OUTPUT

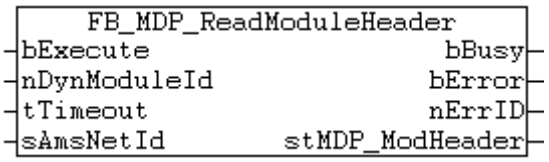
```
VAR_OUTPUT
    bBusy           : BOOL;
    bError          : BOOL;     (* indicates if Read was successfull or not *)
    nErrID          : UDINT;
    iErrPos         : USINT;
    arrStartIdx     : ARRAY[0..255] OF UINT;    (* startindexes in bytes of each subindex element *)
END_VAR
```

bBusy	This output is TRUE as long as the function block is active.
bError	Becomes TRUE as soon as an error situation occurs.
nErrID	Returns an error code [▶ 34] if the bError output is set.
iErrPos	If an error occurred and this refers to an individual element, then this output indicates the position (subindex of the element) at which an error first occurred.
arrStartIdx	This array describes how the individually queried subindices have been stored in the buffer. The array index zero indicates the position in bytes at which the data of subindex zero begins in the buffer. Subsequent subindices are handled analogously.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib

4.7 FB_MDP_ReadModuleHeader



The function block enables the header of an MDP module to be queried.

VAR_INPUT

```
VAR_INPUT
    bExecute        : BOOL;
    nDynModuleId    : BYTE;             (* the dynamic module id for which the module header is requ
ested *)
    tTimeout        : TIME  := DEFAULT_ADS_TIMEOUT; (* States the time before the function is cancel
led. *)
    sAmsNetId       : T_AmsNetId;          (* keep empty '' for the local device *)
END_VAR
```

bExecute	The function block is called by a rising edge on the input *bExecute*, if the block is not already active.
stMDP_DynAddr	The MDP addressing belonging to the selected network module is specified at this input. The dynamic Module ID must already be specified with it.
tTimeout	Specifies a maximum length of time for the execution of the function block.
sAmsNetId	For local access don't specify this input or allocate an empty string. For remote access to another computer specify its AMS Net Id.

VAR_OUTPUT

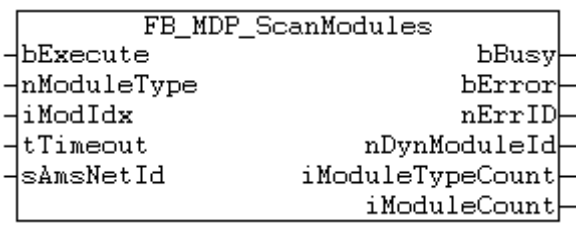
```
VAR_OUTPUT
    bBusy           : BOOL;
    bError          : BOOL;         (* indicates if Read was successfull or not *)
    nErrID          : UDINT;
    stMDP_ModHeader : ST_MDP_ModuleHeader;
END_VAR
```

bBusy	This output is TRUE as long as the function block is active.
bError	Becomes TRUE as soon as an error situation occurs.
nErrID	Returns an error code [▶ 34] if the bError output is set.
stMDP_ModuleHeader	The header information from the read MDP module is displayed at this output in the form of the structure ST_MDP_ModuleHeader [▶ 31].

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib

4.8 FB_MDP_ScanModules



The function block enables a device to be scanned for a certain MDP module.
Selection can be made if several instances of the module type are present. The dynamic Module ID for the selected module type is determined by the function block.
This is an important component of the MDP addressing, which is represented in the structure ST_MDP_Addr [▶ 30].

VAR_INPUT

```
VAR_INPUT
    bExecute        : BOOL;
    nModuleType     : WORD;             (* chosen module type out of the module type list *)
    iModIdx         : USINT;               (* chosen index(0..n) of the demanded module type. E.g.
second NIC(idx 1) of three found NICs. *)
    tTimeout        : TIME  := DEFAULT_ADS_TIMEOUT; (* States the time before the function is cancel
led. *)
    sAmsNetId       : T_AmsNetId;          (* keep empty '' for the local device *)
END_VAR
```

bExecute	The function block is called by a rising edge on the input *bExecute*, if the block is not already active.
nModuleType	The MDP module type is specified at this input. The possible types are listed in the enumeration E_MDP_ModuleType [▶ 30]. (General information on the module type list
iModIdx	If several instances of an MDP module exist, a selection can be made by means of the input *iModIdx* (0,...,n). In the case of uncertainty concerning the selection: information about which module is explicitly concerned can be queried via the function block FB_MDP_ReadModuleHeader [▶ 18] after scanning.
tTimeout	Specifies a maximum length of time for the execution of the function block.
sAmsNetId	For local access don't specify this input or allocate an empty string. For remote access to another computer specify its AMS Net Id.

VAR_OUTPUT

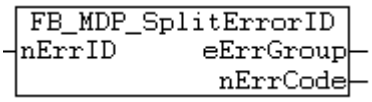
```
VAR_OUTPUT
    bBusy             : BOOL;
    bError            : BOOL;     (* indicates if Scan was successfull or not *)
    nErrID            : UDINT;
    nDynModuleId      : BYTE;     (* Dynamic Module Id *)
    iModuleTypeCount  : USINT;        (* returns the number of found modules equal the demanded modu
le type. *)
    iModuleCount      : USINT;        (* returns the number of all detected MDP modules. *)
END_VAR
```

bBusy	This output is TRUE as long as the function block is active.
bError	Becomes TRUE as soon as an error situation occurs.
nErrID	Returns an error code [▶ 34] if the bError output is set.
nDynModuleId	This output indicates the dynamic Module ID determined for the selected module.
iModuleTypeCount	The output *iModuleTypeCount* indicates the number of modules that correspond to the specified type.
iModuleCount	The output *iModuleCount* indicates the entire number of modules on the device.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib

4.9 FB_MDP_SplitErrorCode



The function block enables the *nErrID* to be split into an error group [▶ 34] and a specific error code. Accordingly, this function block can be referred to for the simplified evaluation of *nErrID*.

VAR_INPUT

```
VAR_INPUT
    nErrID  :UDINT;
END_VAR
```

nErrID nErrID is specified as an input on the function block. This 4-byte variable corresponds to the output nErrID on an MDP function block.

VAR_OUTPUT

```
VAR_OUTPUT
    eErrGroup    :E_MDP_ErrGroup;          (* type of transmitted error code *)
    nErrCode     :UINT;          (* error code [see specific error type table] *)
END_VAR
```

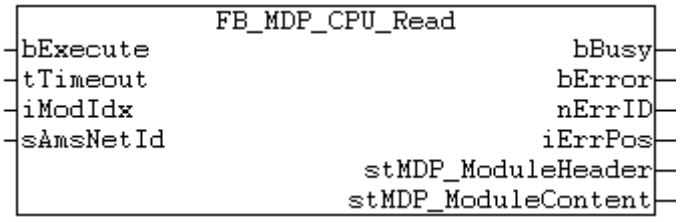
eErrGroup The output eErrGroup corresponds to a value of the enumeration E_MDP_ErrGroup [▶ 34]. It is possible with the aid of the error group to distinguish the type of error or the source of error concerned.

nErrCode The error code is specific for each error group.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib

4.10 FB_MDP_CPU_Read



The function block enables the MDP CPU module to be queried. (General information on the MDP CPU module)

VAR_INPUT

```
VAR_INPUT
    bExecute  : BOOL;                (* Function block execution is triggered by a rising edge at t
his input.*)
    tTimeout  : TIME  := DEFAULT_ADS_TIMEOUT;     (* States the time before the function is cancelle
d. *)
    iModIdx   : USINT := 0;              (* Index number of chosen MDP module *)
    sAmsNetId : T_AmsNetId;              (* keep empty '' for the local device *)
END_VAR
```

bExecute The function block is called by a rising edge on the input *bExecute*, if the block is not already active.

tTimeout	Specifies a maximum length of time for the execution of the function block.
iModIdx	If several instances of an MDP module exist, a selection can be made by means of the input iModIdx (0,...,n).
sAmsNetId	For local access don't specify this input or allocate an empty string. For remote access to another computer specify its AMS Net Id.

VAR_OUTPUT

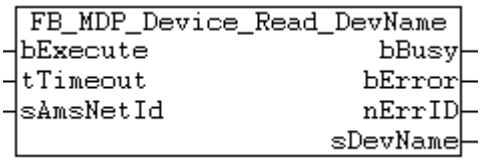
```
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
    nErrID              : UDINT;
    iErrPos             : USINT;
    stMDP_ModuleHeader  : ST_MDP_ModuleHeader;
    stMDP_ModuleContent : ST_MDP_CPU;
END_VAR
```

bBusy	This output is TRUE if the function block is active.
bError	Becomes TRUE as soon as an error situation occurs.
nErrID	Returns an error code [▶ 34] if the bError output is set.
iErrPos	If an error occurred and this refers to an individual element, then this output indicates the position (subindex of the element) at which an error first occurred.
stMDP_ModuleHeader	The header information from the read MDP module is displayed at this output in the form of the structure ST_MDP_ModuleHeader [▶ 31].
stMDP_ModuleContent	The information from TableID 1 of the read MDP module is displayed at this output in the form of the structure ST_MDP_CPU [▶ 31].

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib

4.11 FB_MDP_Device_Read_DevName



The function block enables the device name to be queried. This information is in the General Area of the MDP. (General information on the MDP information model)

VAR_INPUT

```
VAR_INPUT
    bExecute        : BOOL;             (* Function block execution is triggered by a rising edge at
 this input.*)
    tTimeout        : TIME  := DEFAULT_ADS_TIMEOUT; (* States the time before the function is cancel
led. *)
    sAmsNetId       : T_AmsNetId;          (* keep empty '' for the local device *)
END_VAR
```

bExecute	The function block is called by a rising edge on the input *bExecute*, if the block is not already active.
tTimeout	Specifies a maximum length of time for the execution of the function block.
sAmsNetId	For local access don't specify this input or allocate an empty string. For remote access to another computer specify its AMS Net Id.

VAR_OUTPUT

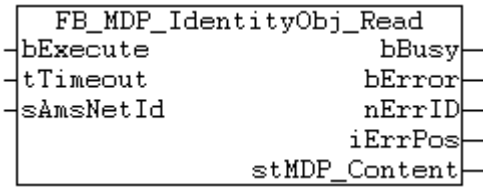
```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    sDevName    : T_MaxString;
END_VAR
```

bBusy	This output is TRUE as long as the function block is active.
bError	Becomes TRUE as soon as an error situation occurs.
nErrID	Returns an error code [▶ 34] if the bError output is set.
sDevName	The queried name is output as a string at this output.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib

4.12 FB_MDP_IdentityObj_Read



The function block enables the IdentityObject table to be queried. (General information on the MDP IdentityObject module from the General Area)

VAR_INPUT

```
VAR_INPUT
    bExecute  : BOOL;                (* Function block execution is triggered by a rising edge at t
his input.*)
    tTimeout  : TIME  := DEFAULT_ADS_TIMEOUT;     (* States the time before the function is cancelle
d. *)
    sAmsNetId : T_AmsNetId;               (* keep empty '' for the local device *)
END_VAR
```

bExecute	The function block is called by a rising edge on the input *bExecute*, if the block is not already active.
tTimeout	Specifies a maximum length of time for the execution of the function block.
sAmsNetId	For local access don't specify this input or allocate an empty string. For remote access to another computer specify its AMS Net Id.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
    nErrID              : UDINT;
    iErrPos             : USINT;
    stMDP_ModuleContent : ST_MDP_IdentityObject;
END_VAR
```

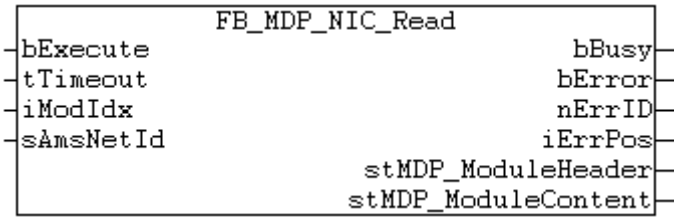
bBusy	This output is TRUE if the function block is active.
bError	Becomes TRUE as soon as an error situation occurs.
nErrID	Returns an error code [▶ 34] if the bError output is set.

iErrPos	If an error occurred and this refers to an individual element, then this output indicates the position (subindex of the element) at which an error first occurred.
stMDP_ModuleContent	The information from the table is displayed at this output in the form of the structure ST_MDP_IdentityObject [▶ 32].

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib

4.13 FB_MDP_NIC_Read



The function block enables the MDP NIC (Network Interface Card) module to be queried. (General information on the MDP NIC module)

VAR_INPUT

```
VAR_INPUT
    bExecute  : BOOL;                (* Function block execution is triggered by a rising edge at t
his input.*)
    tTimeout  : TIME  := DEFAULT_ADS_TIMEOUT;     (* States the time before the function is cancelle
d. *)
    iModIdx   : USINT := 0;              (* Index number of chosen MDP module *)
    sAmsNetId : T_AmsNetId;              (* keep empty '' for the local device *)
END_VAR
```

bExecute	The function block is called by a rising edge on the input *bExecute*, if the block is not already active.
tTimeout	Specifies a maximum length of time for the execution of the function block.
iModIdx	If several instances of an MDP module exist, a selection can be made by means of the input iModIdx (0,...,n).
sAmsNetId	For local access don't specify this input or allocate an empty string. For remote access to another computer specify its AMS Net Id.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
    nErrID              : UDINT;
    iErrPos             : USINT;
    stMDP_ModuleHeader  : ST_MDP_ModuleHeader;
    stMDP_ModuleContent : ST_MDP_NIC_Properties;
END_VAR
```

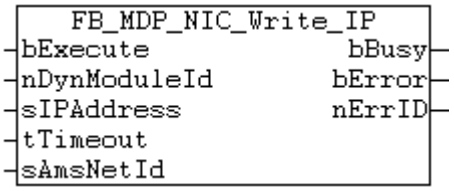
bBusy	This output is TRUE as long as the function block is active.
bError	Becomes TRUE as soon as an error situation occurs.
nErrID	Returns an error code [▶ 34] if the bError output is set.
iErrPos	If an error occurred and this refers to an individual element, then this output indicates the position (subindex of the element) at which an error first occurred.

stMDP_ModuleHeader	The header information from the read MDP module is displayed at this output in the form of the structure ST_MDP_ModuleHeader [▶ 31].
stMDP_ModuleContent	The information from TableID 1 of the read MDP module is displayed at this output in the form of the structure ST_MDP_NIC [▶ 32].

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib

4.14 FB_MDP_NIC_Write_IP



The function block enables a new IP address to be set. This element is part of the MDP NIC module. (General information on the MDP NIC module)

Please note that changes of this kind affect an existing network connection to the computer.

VAR_INPUT

```
VAR_INPUT
    bExecute        : BOOL;
    nDynModuleId    : BYTE;             (* the dynamic module id *)
    sIPAddress      : T_MaxString;          (* IP Address *)
    tTimeout        : TIME  := DEFAULT_ADS_TIMEOUT; (* States the time before the function is cancel
led. *)
    sAmsNetId       : T_AmsNetId;          (* keep empty '' for the local device *)
END_VAR
```

bExecute	The function block is called by a rising edge on the input *bExecute*, if the block is not already active.
nDynModuleId	The dynamic Module ID belonging to the selected network module is specified at this input.
sIPAddress	The IP address specified at this input in the form of a string is transmitted.
tTimeout	Specifies a maximum length of time for the execution of the function block.
sAmsNetId	For local access don't specify this input or allocate an empty string. For remote access to another computer specify its AMS Net Id.

VAR_OUTPUT

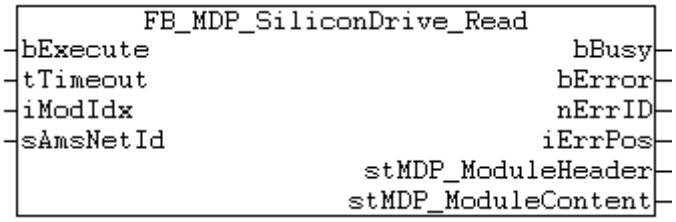
```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
END_VAR
```

bBusy	This output is TRUE if the function block is active.
bError	Becomes TRUE as soon as an error situation occurs.
nErrID	Returns an error code [▶ 34] if the bError output is set.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib

4.15 FB_MDP_SiliconDrive_Read



The function block enables the MDP SiliconDrive module to be queried. (General information on the MDP SiliconDrive module)

VAR_INPUT

```
VAR_INPUT
    bExecute  : BOOL;                (* Function block execution is triggered by a rising edge at t
his input.*)
    tTimeout  : TIME  := DEFAULT_ADS_TIMEOUT;     (* States the time before the function is cancelle
d. *)
    iModIdx   : USINT := 0;               (* Index number of chosen MDP module *)
    sAmsNetId : T_AmsNetId;               (* keep empty '' for the local device *)
END_VAR
```

bExecute	The function block is called by a rising edge on the input *bExecute*, if the block is not already active.
tTimeout	Specifies a maximum length of time for the execution of the function block.
iModIdx	If several instances of an MDP module exist, a selection can be made by means of the input iModIdx (0,...,n).
sAmsNetId	For local access don't specify this input or allocate an empty string. For remote access to another computer specify its AMS Net Id.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
    nErrID              : UDINT;
    iErrPos             : USINT;
    stMDP_ModuleHeader  : ST_MDP_ModuleHeader;
    stMDP_ModuleContent : ST_MDP_SiliconDrive;
END_VAR
```

bBusy	This output is TRUE as long as the function block is active.
bError	Becomes TRUE as soon as an error situation occurs.
nErrID	Returns an error code [▶ 34] if the bError output is set.
iErrPos	If an error occurred and this refers to an individual element, then this output indicates the position (subindex of the element) at which an error first occurred.
stMDP_ModuleHeader	The header information from the read MDP module is displayed at this output in the form of the structure ST_MDP_ModuleHeader [▶ 31].
stMDP_ModuleContent	The information from TableID 1 of the read MDP module is displayed at this output in the form of the structure ST_MDP_SiliconDrive [▶ 32].

NOTE

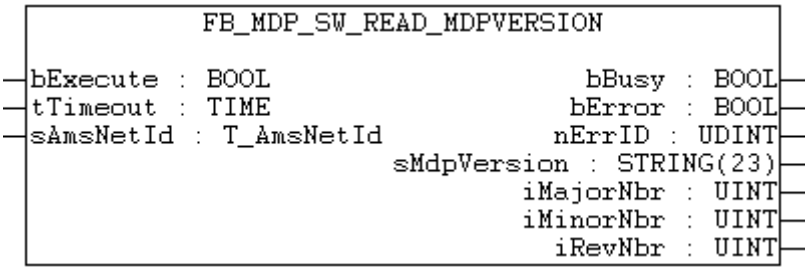
Timeout possible

The querying of the MDP Silicon Drive module is one of the more time-consuming processes. Hence, the Standard ADS Timeout can be exceeded. This can be remedied by increasing the period tTimeout applied to the input of the function block.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib

4.16 FB_MDP_SW_Read_MdpVersion



The function block enables the MDP version to be queried. This information is located in the module software in the configuration area of the MDP. (General information on the MDP information model)

The MDP version is independent of the PLC library version. The PLC library version is provided by the function F_GetVersionTcMDP [▶ 29].

VAR_INPUT

```
VAR_INPUT
    bExecute        :BOOL;              (* Function block execution is triggered by a rising edge at
 this input.*)
    tTimeout        :TIME   :=DEFAULT_ADS_TIMEOUT;  (* States the time before the function is cancel
led. *)
    sAmsNetId       :T_AmsNetId;           (* keep empty '' for the local device *)
END_VAR
```

bExecute	The function block is called by a rising edge on the input *bExecute*, if the block is not already active.
tTimeout	Specifies a maximum length of time for the execution of the function block.
sAmsNetId	For local access don't specify this input or allocate an empty string. For remote access to another computer specify its AMS Net Id.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       :BOOL;
    bError      :BOOL;
    nErrID      :UDINT;
    sMdpVersion :STRING(23);    (* complete MDP version as string [e.g.: '1, 0, 4, 47'] *)
    iMajorNbr   :UINT;      (* major number [e.g.: 1] *)
    iMinorNbr   :UINT;      (* minor number [e.g.: 4] *)
    iRevNbr     :UINT;      (* revision number [e.g.: 47] *)
END_VAR
```

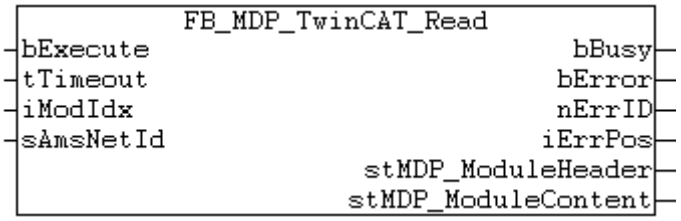
bBusy	This output is TRUE if the function block is active.
bError	Becomes TRUE as soon as an error situation occurs.
nErrID	Returns an error code [▶ 34] if the bError output is set.

sMdpVersion	The queried MDP version is output as a string at this output.
iMajorNbr	The first position of the MDP version is output as a number at this output.
iMinorNbr	The second position of the MDP version is output as a number at this output.
iRevNbr	The third position of the MDP version is output as a number at this output.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib [version 1.2.0 or higher]

4.17 FB_MDP_TwinCAT_Read



The function block enables the MDP TwinCAT module to be queried. (General information on the MDP TwinCAT module)

VAR_INPUT

```
VAR_INPUT
    bExecute  : BOOL;                (* Function block execution is triggered by a rising edge at t
his input.*)
    tTimeout  : TIME  := DEFAULT_ADS_TIMEOUT;     (* States the time before the function is cancelle
d. *)
    iModIdx   : USINT := 0;               (* Index number of chosen MDP module *)
    sAmsNetId : T_AmsNetId;               (* keep empty '' for the local device *)
END_VAR
```

bExecute	The function block is called by a rising edge on the input *bExecute*, if the block is not already active.
tTimeout	Specifies a maximum length of time for the execution of the function block.
iModIdx	If several instances of an MDP module exist, a selection can be made by means of the input iModIdx (0,...,n).
sAmsNetId	For local access don't specify this input or allocate an empty string. For remote access to another computer specify its AMS Net Id.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
    nErrID              : UDINT;
    iErrPos             : USINT;
    stMDP_ModuleHeader  : ST_MDP_ModuleHeader;
    stMDP_ModuleContent : ST_MDP_TwinCAT;
END_VAR
```

bBusy	This output is TRUE if the function block is active.
bError	Becomes TRUE as soon as an error situation occurs.
nErrID	Returns an error code [▶ 34] if the bError output is set.
iErrPos	If an error occurred and this refers to an individual element, then this output indicates the position (subindex of the element) at which an error first occurred.

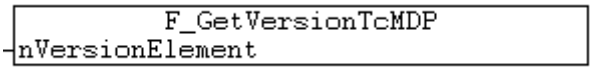
stMDP_ModuleHeader	The header information from the read MDP module is displayed at this output in the form of the structure ST_MDP_ModuleHeader [▶ 31].
stMDP_ModuleContent	The information from TableID 1 of the read MDP module is displayed at this output in the form of the structure ST_MDP_TwinCAT [▶ 33].

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1551	PC or CX (x86, ARM)	TcMDP.Lib

5 Functions

5.1 F_GetVersionTcMDP



This function can be used to read PLC library version information.

The MDP version is independent of the PLC library version. The MDP version is provided by the function block FB_MDP_SW_Read_MdpVersion [▶ 26].

FUNCTION F_GetVersionTcMDP: UINT

```
VAR_INPUT
    nVersionElement : INT;
END_VAR
```

nVersionElement : Version element to be read. Possible parameters:

- 1 : major number;
- 2 : minor number;
- 3 : revision number;

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib

6 Data types

6.1 E_MDP_AddrArea

```
TYPE E_MDP_AddrArea :(
    eMDP_Area_ConfigArea        := 16#8,
    eMDP_Area_ServiceArea       := 16#B,
    eMDP_Area_DeviceArea        := 16#F
);
END_TYPE
```

The enumeration *E_MDP_AddrArea* defines constant values for the different areas in the MDP.

A general description can be found in the Information Model.

6.2 E_MDP_ModuleType

```
TYPE E_MDP_ModuleType :(
    eMDP_ModT_NIC           := 16#0002,
    eMDP_ModT_Time          := 16#0003,
    eMDP_ModT_UserManagement    := 16#0004,
    eMDP_ModT_RAS           := 16#0005,
    eMDP_ModT_FTP           := 16#0006,
    eMDP_ModT_SMB           := 16#0007,
    eMDP_ModT_TwinCAT       := 16#0008,
    eMDP_ModT_Datastore     := 16#0009,
    eMDP_ModT_Software      := 16#000A,
    eMDP_ModT_CPU           := 16#000B,
    eMDP_ModT_Memory        := 16#000C,
    eMDP_ModT_Firewall      := 16#000E,
    eMDP_ModT_FileSystemObject := 16#0010,
    eMDP_ModT_PLC           := 16#0012,
    eMDP_ModT_DisplayDevice     := 16#0013,
    eMDP_ModT_EWF           := 16#0014,
    eMDP_ModT_FBWF          := 16#0015,
    eMDP_ModT_SiliconDrive      := 16#0017,
    eMDP_ModT_OS            := 16#0018,
    eMDP_ModT_Raid          := 16#0019,
    eMDP_ModT_Fan           := 16#001B,
    eMDP_ModT_Mainboard     := 16#001C,
    eMDP_ModT_DiskManagement    := 16#001D,
    eMDP_ModT_UPS           := 16#001E,
    eMDP_ModT_Misc          := 16#0100
);
END_TYPE
```

The enumeration *E_MDP_ModuleType* defines constant values for the different module types in the MDP. A module type can occur several times per device. Hence, a device with two Ethernet interfaces also has two MDP NIC modules.

Detailed information about the modules can be found in the documentation IPC Diagnostic - Module Types.

This module type is not to be equated with the dynamic Module ID !

6.3 ST_MDP_Addr

```
TYPE ST_MDP_Addr :
STRUCT
    nArea       : BYTE;     (* Area [range: 0x0-0xF] *)
    nModuleId   : BYTE;     (* Dynamic Module Id [range: 0x00-0xFF] *)
    nTableId    : BYTE;     (* Table Id [range: 0x0-0xF] *)
    nFlag       : BYTE;     (* Flags [range: 0x00-0xFF] *)
    nSubIdx     : BYTE;     (* SubIndex [range: 0x00-0xFF] *)
    arrReserved : ARRAY[0..2] OF BYTE;
END_STRUCT
END_TYPE
```

The structure contains information that is required for the MDP addressing.

nArea	Possible MDP areas are listed in E_MDP_AddrArea [▶ 30].
nModuleId	The Module ID is assigned dynamically. It does not correspond to the module types listed in E_MDP_ModuleType. The function block FB_MDP_ScanModules [▶ 18] can be used in order to determine a dynamic Module ID for a special type of module.
nTableId	This value specifies the number of the selected table of the selected module.
nFlag	This parameter is used internally only. It remains at the default value of 0x00.
nSubIdx	The Subindex parameter corresponds to the subindex in a table in an MDP module.

Detailed information on MDP addressing can be found in the documentaion IPC Diagnostic - Ads Overview.

6.4 ST_MDP_ModuleHeader

```
TYPE ST_MDP_ModuleHeader :
STRUCT
    iLen     :UINT;
    nAddr    :DWORD;
    sType    :T_MaxString;
    sName    :T_MaxString;
    nDevType :DWORD;
END_STRUCT
END_TYPE
```

The structure contains device information. This information always corresponds to the Table ID 0 of an MDP module. Each module possesses this module header.

iLen : Specifies the number of parameters in the Table ID, in this case the module header.

nAddr : Specifies the address of the module.

sType : Specifies the type of module. Possible types are listed in the MDP module list.

sName : Specifies the name of this MDP module.

nDevType : Specifies the type of MDP module as code.

6.5 ST_MDP_CPU

```
TYPE ST_MDP_CPU :
STRUCT
    iLen           :UINT;          (* Length *)
    iCPUfrequency  :UDINT;         (* CPU Frequency *)
    iCPUusage      :UINT;          (* Current CPU Usage [%] *)
END_STRUCT
END_TYPE
```

The structure contains information on the MDP CPU module.

This complete information can be queried by means of the function block FB_MDP_CPU_Read [▶ 20].

The parameters existing in this structure correspond to the subindices of the first table (Table ID 1) within the MDP CPU module.

6.6 ST_MDP_IdentityObject

```
TYPE ST_MDP_IdentityObject :
STRUCT
    iLen          : UINT;          (* Length *)
    iVendor       : UDINT;         (* Vendor *)
    iProductCode  : UDINT;         (* Product Code *)          (* not yet supported *)
    iRevNumber    : UDINT;         (* Revision Number *)   (* not yet supported *)
    iSerialNumber : UDINT;         (* Serial Number *)
END_STRUCT
END_TYPE
```

The structure contains information on the IdentityObject table, which is in the MDP General Area.

This complete information can be queried by means of the function block FB_MDP_IdentityObj_Read [▶ 22].

The parameters existing in this structure correspond to the subindices of the 'Identity Object' table within the MDP IdentityObject module in the General Area.

6.7 ST_MDP_NIC_Properties

```
TYPE ST_MDP_NIC_Properties :
STRUCT
    iLen           : UINT;      (* Length *)
    sMACAddress    : T_MaxString;   (* MAC Address *)
    sIPAddress     : T_MaxString;   (* IP Address *)
    sSubnetMask    : T_MaxString;   (* Subnet Mask *)
    bDHCP          : BOOL;      (* DHCP *)
END_STRUCT
END_TYPE
```

The structure contains information on the MDP NIC (Network Interface Card) module.

This complete information can be queried by means of the function block FB_MDP_NIC_Read [▶ 23].

The parameters existing in this structure correspond to the subindices of the first table (Table ID 1) within the MDP NIC module.

6.8 ST_MDP_SiliconDrive

```
TYPE ST_MDP_SiliconDrive :
STRUCT
    iLen                   : UINT;          (* Length *)
    iTotalEraseCounts      : UDINT;         (* Total EraseCounts (lower 4 bytes) *)
    iDriveUsage            : UINT;          (* Drive Usage (%) *)
    iNbrSpares             : UINT;          (* Number of Spares *)
    iNbrUsedSpares         : UINT;          (* Spares Used *)
    iTotalEraseCountsHigh  : UDINT;         (* Total EraseCounts (higher 4 bytes) *)
END_STRUCT
END_TYPE
```

The structure contains information on the MDP SiliconDrive module.

This complete information can be queried by means of the function block FB_MDP_SiliconDrive_Read [▶ 25].

iLen	iLen defines the number of MDP elements in the table of the MDP module.
iTotalEraseCounts	The entire sum of write/erase cycles of all blocks on a Silicon Drive. This element is defined as 64 Bit value.*iTotalEraseCounts* contains the lower 32 Bit.
iTotalEraseCountsHigh	The entire sum of write/erase cycles of all blocks on a Silicon Drive. This element is defined as 64 Bit value.*iTotalEraseCountsHigh* contains the higher 32 Bit.

iDriveUsage	The calculated usage of the silicon drive. It's based on maximal two million write/erase cycles per Block.
iNbrSpares	Spare Blocks replaces Blocks who had been wear out. *iNbrSpares* is the total number of Spare Blocks available.
iNbrUsedSpares	The number of spares already used on the Silicon Drive.

The parameters existing in this structure correspond to the subindices of the first table (Table ID 1) within the MDP SiliconDrive module.

6.9 ST_MDP_TwinCAT

```
TYPE ST_MDP_TwinCAT :
STRUCT
    iLen            : UINT;          (* Length *)
    iMajorVersion   : UINT;          (* Major Version *)
    iMinorVersion   : UINT;          (* Minor Version *)
    iBuild          : UINT;          (* Build *)
    sAmsNETid       : T_MaxString;       (* Ams NET ID *)
    iRegLevel       : UDINT;         (* TwinCAT registration level *)
    iStatus         : UINT;          (* TwinCAT status *)
    iRunAsDev       : UINT;          (* Run As Device *)         (* available for WindowsCE *)
    iShowTargetVisu : UINT;          (* show target visualization *) (* available for WindowsCE *)
    iLogFileSize    : UDINT;         (* log file size *)         (* available for WindowsCE *)
    sLogFilePath    : T_MaxString;       (* log file path *)         (* available for WindowsCE *)
END_STRUCT
END_TYPE
```

The structure contains information on the MDP TwinCAT module.

This complete information can be queried by means of the function block FB_MDP_TwinCAT_Read [▶ 27].

The parameters existing in this structure correspond to the subindices of the first table (Table ID 1) within the MDP TwinCAT module.

7 Error codes

The function blocks of the TcPlcLibMDP.Lib possess an output *nErrID*. This value is 4 bytes in size and returns the error code in the event of an error. *nErrID* consists of two parts:

(MSB) 2 bytes	2 bytes (LSB)
Error Group	**Error Code**
0x EC80	E_MDP_ErrCodesPLC [▶ 35]
0x ECA6	MDP general error
0x ECA7	MDP API error
0x ECA8	ADS error
0x ECAF	MDP module specific error

Error Group

The Error Group describes the type of error that has occurred. The different groups are listed in the enumeration E_MDP_ErrGroup [▶ 34].
All errors generated within the PLC library have the error group 0xEC80.

Error Code

The Error Code describes the concrete error.
For errors internal to the PLC library with the error group 0xEC80, the identifiers are listed in the enumeration E_MDP_ErrCodesPLC [▶ 35]. A description of the further error codes can be found in the Documentation of the IPC Diagnostics in the chapter MDP Error Codes.

General MDP-dependent errors are output in the error group 16#ECA6 “General error codes”. These errors sometimes indicate that an element from the module element list is not available. Example: 16#ECA60105 "No data available" If, in the case of a general or specific function block (see access options [▶ 9]), several elements are queried at the same time and one of these elements is not available or exhibits an error, then the output variable iErrPos indicates the index position (0..n) at which the error occurred for the first time. All elements below this index were queried successfully and are indicated despite the generation of an error at the output.

FB_MDP_SplitErrorID

The function block FB_MDP_SplitErrorID [▶ 20] enables the automatic separation of the variable *nErrID* into error group and error code.

Example:

nErrID = 0x ECA8 0745

The Error Group is 0x ECA8, therefore it is an Ads error.
The Error Code is 0x 0745, therefore it is a timeout error.

7.1 E_MDP_ErrGroup

```
TYPE E_MDP_ErrGroup :(
    eMDP_Err_NoError        := 16#0000,         (* Success - No Error *)
    eMDP_Err_PLC            := 16#EC80,         (* PLC library internal error codes *)
    eMDP_Err_GenErr         := 16#ECA6,         (* General error codes *)
    eMDP_Err_API            := 16#ECA7,         (* API error codes *)
    eMDP_Err_ADS            := 16#ECA8,         (* ADS error codes *)
    eMDP_Err_ModuleSpecific := 16#ECAF          (* Module specific error codes *)
);
END_TYPE
```

The enumeration *E_MDP_ErrGroup* defines constant values for the different error groups in the MDP. These indicate the type of error.

The values appear in the error codes [▶ 34], which are output by a PLC MDP function block in the event of an error.

A general description can be found in the MDP Information Model in the chapter Return Values. Individual error codes from the error groups 16#ECA6 - 16#ECAF are described there.

The error codes from group 16#EC80 are generated by the PLC MDP library and are described in chapter E_MDP_ErrCodesPLC [▶ 35].

7.2 E_MDP_ErrCodesPLC

NOTE

Timeout possible

The length of the processing time can vary depending on the MDP query. Due to the internal processes, the processing time can sometimes exceed the Standard ADS Timeout. This can be remedied by increasing the time period tTimeout applied to the input of the function block.

```
TYPE E_MDP_ErrCodesPLC :(
(* list of PLC library internal error codes *)
    eMDP_ErrPLC_NoError         := 16#0000,
    eMDP_ErrPLC_TimeOut         := 16#0001,
    eMDP_ErrPLC_ModuleNotFound  := 16#0002,
    eMDP_ErrPLC_BufferTooSmall  := 16#0003,
    eMDP_ErrPLC_ElementNotFound := 16#0004
);
END_TYPE
```

The enumeration *E_MDP_ErrCodesPLC* defines constant values for the different errors that can be generated internally in the library.

These values appear in the error codes [▶ 34], which are output by a PLC MDP function block in the event of an error.

eMDP_ErrPLC_TimeOut	The error *eMDP_ErrPLC_TimeOut* is generated if the time *tTimeout* applied to the input of the function block has expired.
eMDP_ErrPLC_ModuleNotFound	A list of active modules exists in the MDP. The function blocks in the PLC MDP library search this list for the queried module. If the list does not contain the module, then the error *eMDP_ErrPLC_ModuleNotFound* is output. This is the case when the module/device is not installed on the system or does not even exist.
eMDP_ErrPLC_BufferTooSmall	If a buffer has been specified at the input of the function block by means of pointers, then it is possible that this is not large enough for the existing data. In this case the error *eMDP_ErrPLC_BufferTooSmall* is output.
eMDP_ErrPLC_ElementNotFound	The request for a specific element was not successful. The element wasn't found. Maybe the specific module or element does not even exist on the system.

A general description can be found in the Information Model.

8 Samples

The following samples are available for the TwinCAT PLC Library Modular Device Profile.

Sample - Read access to MDP elements

This example offers an introduction to the handling of the function blocks that are available with the TcPlcMDP library.

This example is dedicated to the goal of determining the state of the Compact Flash card in the Embedded PC.
This can be found out via a parameter in the MDP Model. The querying of other parameters takes place analogously to this example.
Hence, this example can also be considered to be a guide to querying any MDP parameter from an MDP module.

Step-By-Step description of this example [▶ 37]

Download:
https://infosys.beckhoff.com/content/1033/tcplclibmdp/Resources/11941226379/.zip

Sample2 - Write access to MDP elements

This example shows that the write access to MDP elements can be implemented similarly.
In this sample a new IP address is configured. First the DHCP is deactivated and then a new IP address is set.

Download:
https://infosys.beckhoff.com/content/1033/tcplclibmdp/Resources/11941227787/.zip

Sample3 - Query of a module via two ways

This example shows two different ways to get information out of the MDP module CPU.

1. Request of the MDP module CPU with the specific function block.
2. Special request of single MDP elements out of the MDP module CPU with the general function block FB_MDP_ReadElement.
   Request of any possible element out of the MDP modules is possible in the same manner. Adaptation to another element is very easy! All available elements can be requested with this function block.

For Sample3 the PLC library version 1.2.0 or higher is required.

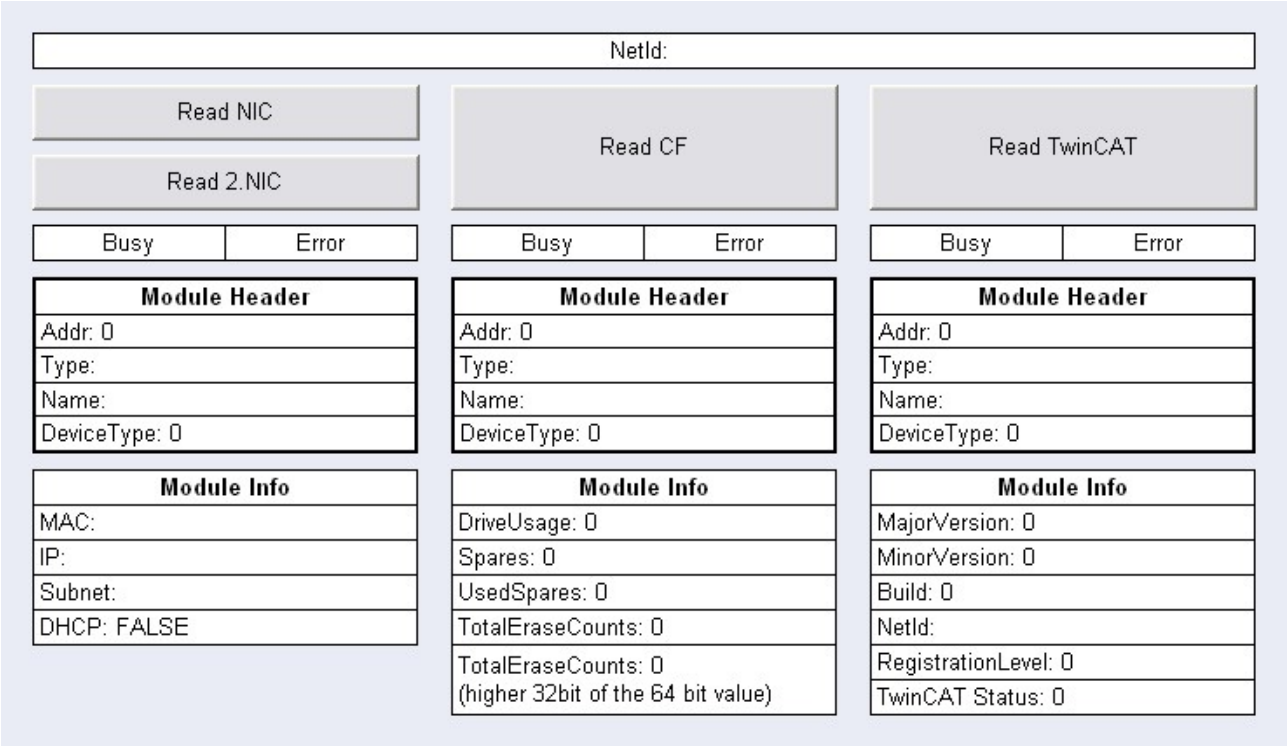
The CPU temperature is only available since MDP version 1.5.0. Also this parameter is not supported by every hardware.

Download:
https://infosys.beckhoff.com/content/1033/tcplclibmdp/Resources/11941229195/.zip

Sample4 - Query of complete modules with the specific function blocks

This example shows the easiest access to several MDP modules. Therefor the specific function blocks are used.

For Sample4 the PLC library version 1.3.0 or higher is recommended.



Download:
https://infosys.beckhoff.com/content/1033/tcplclibmdp/Resources/11941230603/.zip

8.1 Example

This example offers an introduction to the handling of the function blocks that are available with the TcPlcMDP library.

This example is dedicated to the goal of determining the state of the Compact Flash card in the Embedded PC.

This can be found out via a parameter in the MDP Model. The querying of other parameters takes place analogously to this example.

Hence, this example can also be a guide to querying any MDP parameter from an MDP module.

If you wish to go through the example on a PC that does not use a silicon Compact Flash card as memory, you can query the CPU utilization instead, for example. To do this, execute this example in the same way, adapting just a few points accordingly. Necessary values for this can be found in the general module description of the MDP CPU module.

Overview

The following steps are now performed:

1. Installation of the PLC library
2. Program structure
3. Determination of the dynamic Module ID
4. Querying of the MDP parameter
5. Test

1. Installation of the PLC library

Start TwinCAT PLC Control.

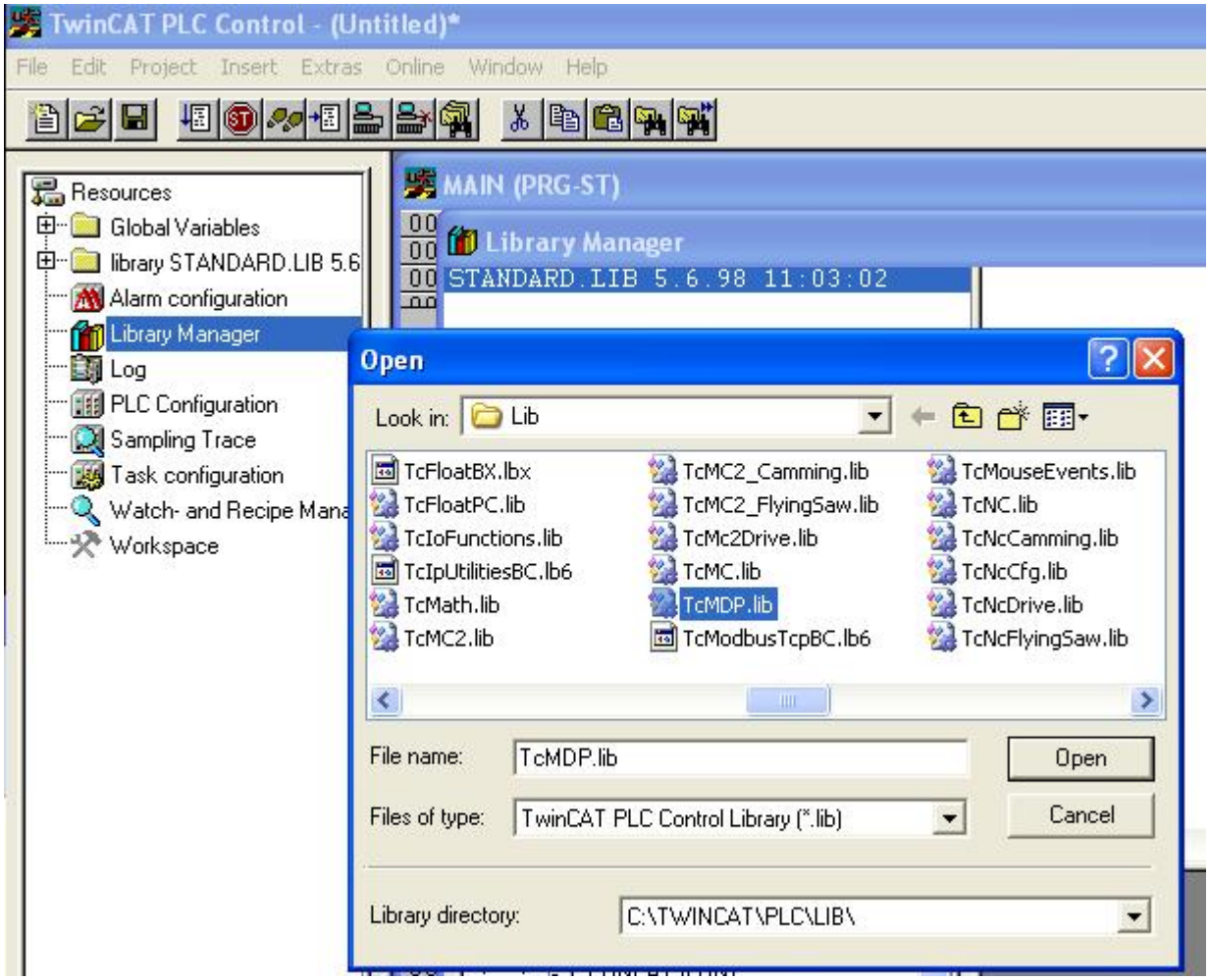
Create a new PLC project with 'File > New'.

Select your target platform PC and CX (x86) or CX (ARM).

Your first POU is a program called MAIN and in the programming language ST (Structured Text).

Open the Resources tab and the library manager.

Insert the library TcMDP.lib as shown in the picture below via 'Insert > Further library'.



All PLC blocks of the TwinCAT PLC MDP library are now available to you. All further implicitly required libraries have been automatically integrated with the TcMDP.lib.

2. Program structure

The state of the Compact Flash card is represented by a parameter in the MDP. To query this individual parameter, the dynamic Module ID of the module in which the parameter is located must be known.

This dynamic Module ID must be determined using the function block FB_MDP_ScanModules.

The parameters can then be queried with the function block FB_MDP_Read.

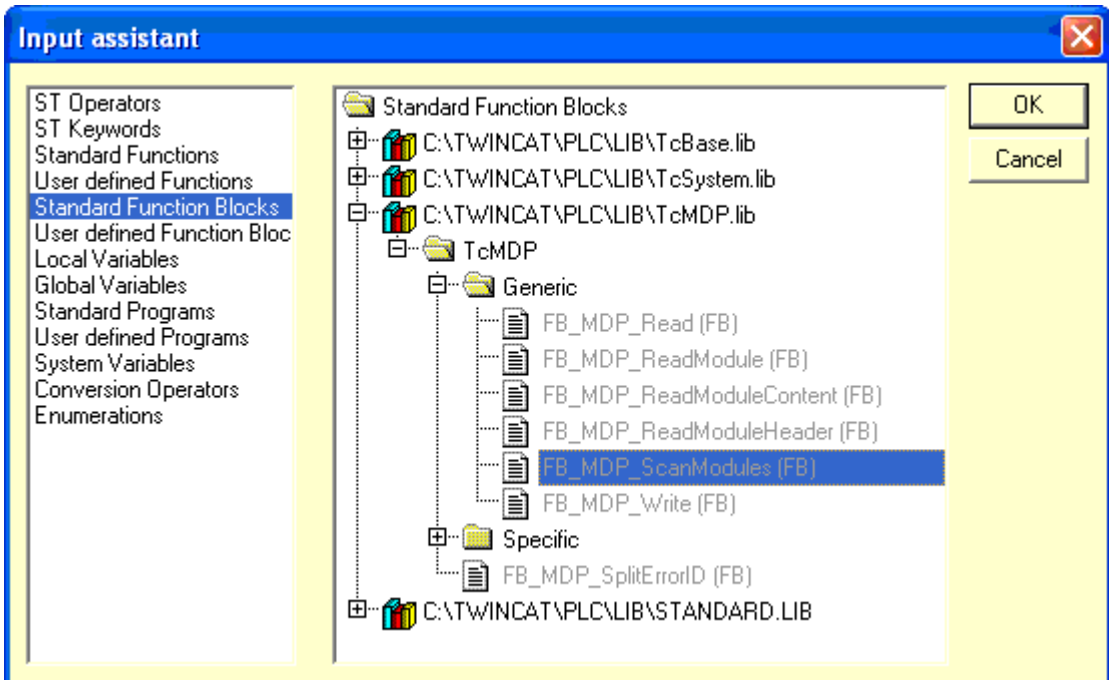
Generate a state machine in the MAIN program for this sequence.

```
MAIN (PRG-ST)
0001 PROGRAM MAIN
0002 VAR
0003     iState           : UINT   := 0;
0004     bStartRequest    : BOOL   := TRUE;
0005 END_VAR

0001 CASE iState  OF
0002 0: (* Idle *)
0003     IF bStartRequest THEN
0004         bStartRequest    := FALSE;
0005         iState           := 1;
0006     END_IF
0007 1: (* Scan for module *)
0008     ;
0009 2: (* Get received dynamic module id *)
0010     ;
0011 3: (* Request MDP Element *)
0012     ;
0013 4: (* Get received Information *)
0014     ;
0015 END_CASE
```

3. Determination of the dynamic Module ID

Insert the MDP function block FB_MDP_ScanModules [▶ 18] (press F2).



In state 1, start the function block by setting the input *bExecute* to TRUE.

The enumeration value eMDP_ModT_SiliconDrive [▶ 30], which allows accesses to information from Compact Flash cards, is specified at *nModuleType*. Further information on this module: General information on the MDP SiliconDrive moduleThere does no need to be any input on iModIdx. This way you can call the first found module of the selected module type automatically (default: iModIdx := 0).

Likewise, there does no need to be any input on tTimeout; instead, you can work with the default (DEFAULT_ADS_TIMEOUT).

```
1: (* Scan for module *)
    FB_MDP_ScanModules(
        bExecute:= TRUE,
        nModuleType:= eMDP_ModT_SiliconDrive,
        iModIdx:= ,
        tTimeout:= ,
```

```
        bBusy=> ,
        bError=> ,
        nErrID=> ,
        nDynModuleId=> ,
        iModuleTypeCount=> ,
        iModuleCount=>
    );
```

In state 2, call this function block cyclically by setting the input *bExecute* to FALSE. In this state, the function block is called if it is busy with the processing of the query.

The transition to the next state can be accomplished as soon as the output *bBusy* goes FALSE.

Your program should now look as follows:

MAIN (PRG-ST)

```
0001 PROGRAM MAIN
0002 VAR
0003     iState           : UINT  := 0;
0004     bStartRequest    : BOOL  := TRUE;
0005     fbScanMDP        : FB_MDP_ScanModules;
0006 END_VAR
0007 1: (* Scan for module *)
0008     fbScanMDP(
0009         bExecute:= TRUE,
0010         nModuleType:= eMDP_ModT_SiliconDrive,
0011         iModIdx:= ,
0012         tTimeout:= ,
0013
0014         bBusy=> ,
0015         bError=> ,
0016         nErrID=> ,
0017         nDynModuleId=> ,
0018         iModuleTypeCount=> ,
0019         iModuleCount=>
0020     );
0021     iState  := 2;
0022 2: (* Get received dynamic module id *)
0023     fbScanMDP(
0024         bExecute:= FALSE,
0025         nModuleType:= eMDP_ModT_SiliconDrive,
0026         iModIdx:= ,
0027         tTimeout:= ,
0028
0029         bBusy=> ,
0030         bError=> ,
0031         nErrID=> ,
0032         nDynModuleId=> ,
0033         iModuleTypeCount=> ,
0034         iModuleCount=>
0035     );
0036     IF NOT fbScanMDP.bBusy THEN
0037         IF NOT fbScanMDP.bError THEN
0038             iState  := 3;
0039         ELSE
0040             iState  := 0;
0041         END_IF
0042     END_IF
0043 3: (* Request MDP Element *)
```

4. Querying of the MDP parameter

The MDP parameter that you would like to query is in a certain table.
This is in turn located in a certain module, which belongs to an area.
Take these values from the MDP description:

TwinCAT ADS Modular Device Profile - Configuration Area

SiliconDrive

0x8nn0

SubIndex	Type	Name	Value	Type
00	VAR	Len		UNSIGNE
01	VAR	Address	0x0017 00nn	UNSIGNE
02	VAR	Type	SiliconDrive	Vis-String
03	VAR	Name	SiliconDrive	Vis-String
04	VAR	Dev Type	0x0017 2710	UNSIGNE

0x8nn1

SubIndex	Type	Name	Type
00	VAR	Len	UNSIGNED8
01	VAR	Total EraseCounts	UNSIGNED64
02	VAR	Drive Usage (%)	UNSIGNED16
03	VAR	Number of Spares	UNSIGNED16
04	VAR	Spares Used	UNSIGNED16

Excerpt from the general information on the MDP SiliconDrive module.

In order to query an MDP element, declare an instance of the function block FB_MDP_Read [▶ 11].
Likewise define a variable iDriveUsage, whereby this is an Unsigned16 variable.

```
    fbReadMDP           : FB_MDP_Read;
    iDriveUsage     : UINT;
```

Transfer the determined values for the sought MDP parameter to the function block. To do this, select the input variable *stMDP_DynAddr* of the type ST_MDP_Addr [▶ 30] of the function block and assign the values.

```
3: (* Request MDP Element *)
    fbReadMDP.stMDP_DynAddr.nArea       := eMDP_Area_ConfigArea;
    fbReadMDP.stMDP_DynAddr.nModuleId           := fbScanMDP.nDynModuleId;
    fbReadMDP.stMDP_DynAddr.nTableId        := 1;
    fbReadMDP.stMDP_DynAddr.nSubIdx     := 2;
```

Further, in state 3, call the function block and start it by setting the input *bExecute* to TRUE.
You have already explicitly assigned the input *stMDP_DynAddr*.
As a data buffer, enter the address and length of your variable *iDriveUsage* for *pDstBuf* and *cbDstBufLen*.
As with the above function block, there does not need to be any input on tTimeout; instead, you can work with the default (DEFAULT_ADS_TIMEOUT).

The program section should now look as follows:

```
3: (* Request MDP Element *)
    fbReadMDP.stMDP_DynAddr.nArea        := INT_TO_BYTE(eMDP_Area_ConfigArea);
    fbReadMDP.stMDP_DynAddr.nModuleId    := fbScanMDP.nDynModuleId;
    fbReadMDP.stMDP_DynAddr.nTableId     := 1;
    fbReadMDP.stMDP_DynAddr.nSubIdx      := 2;

    fbReadMDP(
        bExecute:= TRUE,
        stMDP_DynAddr:= ,
        pDstBuf:= ADR(iDriveUsage),
        cbDstBufLen:= SIZEOF(iDriveUsage),
        tTimeout:= ,

        bBusy=> ,
        bError=> ,
        nErrId=> ,
        nCount=>
    );
    iState  := 4;
4: (* Get received Information *)
    fbReadMDP(
        bExecute:= FALSE,
        stMDP_DynAddr:= ,
        pDstBuf:= ADR(iDriveUsage),
        cbDstBufLen:= SIZEOF(iDriveUsage),
        tTimeout:= ,

        bBusy=> ,
        bError=> ,
        nErrId=> ,
        nCount=>
    );
    IF NOT fbReadMDP.bBusy THEN
        iState  := 0;
    END_IF
END_CASE
```

5. Test

Compile the created PLC program. Make sure that TwinCAT is in the Run mode on the desired system.

Login to the desired run-time system from TwinCAT PLC Control. Start the PLC program.

Upon the initialisation of bStartRequest with TRUE (see 2. Program structure) all conditions of the state machine are implemented once immediately at the program start.

If executed without error, the queried value is now stored in your variable *iDriveUsage*. In this example, this value indicates the percentage of the statistically possible number of writing cycles already performed by the Compact Flash card and thus provides useful information on the service life of your CF card.

If you have performed this example with the goal of querying the CPU utilization, then the utilization of the CPU in % will now be in your variable.

To start the complete query again, set your variable *bStartRequest* to TRUE again (for example by Online Write).

This example can also serve as a general guide. Each MDP parameter can be queried from an MDP module in the same way.

Click here to save this example program:
https://infosys.beckhoff.com/content/1033/tcplclibmdp/Resources/11941226379/.zip.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v2.11.0 Build >= 1541	PC or CX (x86, ARM)	TcMDP.Lib

8.2 Reading IP Serial numbers

This sample illustrates access to the serial number of the IPCs and the serial number of the IPC's mainboard.

- The serial number of the mainboard can be read via a subindex in module Mainboard in the Configuration Area of the IPC diagnostics. The general function block FB_MDP_ReadElement is used for this purpose
- The serial number of the IPC can be read via index 0xF9F0 in the Device Area of the IPC diagnostics. The general function block FB_MDP_ReadIndex is used for this purpose.

Sample: querying the serial number of a Beckhoff IPC

Enumeration definition

```
(* central definition of state machine states *)
TYPE E_State :
(
    Idle,
    ReadSnoMainboardInit,
    ReadSnoMainboardProcess,
    ReadSnoIPCInit,
    ReadSnoIPCProcess
);
END_TYPE
```

Variable declaration

```
PROGRAM MAIN
VAR
    sAmsNetId            : STRING := ''; (* ADS Net ID (local = '') *)
    eState               : E_State;      (* Enum with index for state machine *)
    bStart               : BOOL := TRUE; (* flag to trigger restart of statemachine *)
    sData                : STRING;       (* data storage for string variable *)
    stMDP_Addr           : ST_MDP_Addr;  (* structure which will include all address parameters *)


    (* FB instances *)
    fbReadMDPElement     : FB_MDP_ReadElement;
    fbReadMDPIndex       : FB_MDP_ReadIndex;

    (* results of execution *)
    bError               : BOOL;     (* error flag *)
    nErrID               : UDINT;    (* last error ID *)
    sSerialNoMainboard   : STRING;       (* buffer for serial number of mainboard *)
    sSerialNoIPC         : STRING;       (* buffer for serial number of IPC *)
END_VAR
```

Program code

```
CASE eState OF
    Idle:
        IF bStart THEN
           bStart := FALSE;
              eState := ReadSnoMainboardInit; (* initiate first state *
        END_IF

    (* read serial number of mainboard ************************************************************
* *)
    ReadSnoMainboardInit:
        sData := '';            (* clear data buffer *)
        sSerialNoMainboard := '';           (* clear buffer for serial number of mainboard *)
        stMDP_Addr.nArea := INT_TO_BYTE(eMDP_Area_ConfigArea); (* set area address to "Config Area"
*)
        stMDP_Addr.nTableId := 1;           (* table ID in index for "mainboard information" *)
        stMDP_Addr.nSubIdx := 2;        (* subindex in table ID for "serial number" *)

        fbReadMDPElement(
            bExecute := TRUE,
```

```
            eModuleType := eMDP_ModT_Mainboard,
            stMDP_Addr := stMDP_Addr,       (* MDP address structure. Dynamic module ID will be adde
d internally. *)
            iModIdx := 0,       (* Instance of desired module type (default: 0 = first instance) *)
            pDstBuf := ADR(sData),
            cbDstBufLen := SIZEOF(sData),
            sAmsNetId := sAmsNetId,
            );

        eState := ReadSnoMainboardProcess;

    ReadSnoMainboardProcess:
        fbReadMDPElement(bExecute := FALSE);

        IF NOT fbReadMDPElement.bBusy THEN
            IF fbReadMDPElement.bError THEN
               bError := TRUE;              (* set error flag *)
               nErrID := fbReadMDPElement.nErrID;   (* store error id (16#ECA60105 = BIOS or HW does
 not support this data (here: mainboard data)) *)
               eState := Idle;
            ELSE                    (* set parameters for next steps *)
               bError := FALSE;             (* turn off error flag *)
               sSerialNoMainboard := sData;         (* store serial number of mainboard in dedicated
 variable *)
               eState := ReadSnoIPCInit;
            END_IF
        END_IF

    (* read serial number of IPC ************************************************************** *)
    ReadSnoIPCInit:
        sData := '';        (* clear data buffer *)
        sSerialNoIPC := '';     (* clear buffer for serial number of IPC *)

        fbReadMDPIndex(
            bExecute := TRUE,
            nIndex := 16#F9F0,      (* index: read serial number IPC (-
> see docu 'MDP device area') *)
            nSubIndex := 0,         (* first subdindex (there is only one available for index 16#F9F0
) *)
            pDstBuf := ADR(sData), cbDstBufLen := SIZEOF(sData),
            sAmsNetId := sAmsNetId,
            );

        eState := ReadSnoIPCProcess;

    ReadSnoIPCProcess:

        fbReadMDPIndex(bExecute := FALSE);

        IF NOT fbReadMDPIndex.bBusy THEN
            IF fbReadMDPIndex.bError THEN
               bError := TRUE;          (* set error flag *)
               nErrID := fbReadMDPIndex.nErrID;     (* store error id (16#ECA60105 = BIOS or HW does
 not support this data (here: IPC serial number)) *)
               eState := Idle;
            ELSE                (* set parameters for next steps *)
               bError := FALSE;         (* turn off error flag *)
               sSerialNoIPC := sData;           (* store serial number of mainboard *)
               eState := Idle;
            END_IF
        END_IF
    END_CASE
```

Returning of the mainboard serial number instead of the IPC serial number

In older BIOS version (before Q4/2013) the serial number was not stored in the IPC BIOS. In these cases the return value is the serial number of the IPC mainboard. With older Beckhoff Automation Device Driver versions, the return value is also the serial number of the IPC mainboard. The serial number of the IPC mainboard can always be read via the mainboard module.

More Information:
www.beckhoff.com/tx1200

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