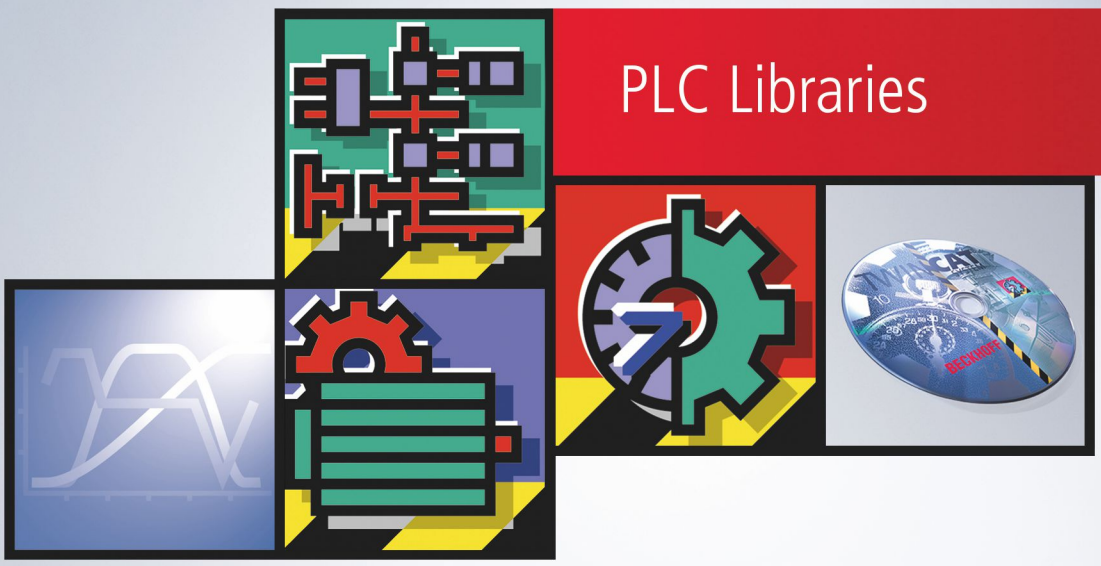
BECKHOFF New Automation Technology

Handbuch | DE

TX1200

TwinCAT 2 | PLC-Bibliothek: TcMDP



02.11.2022 | Version: 1.1

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, für jede Installation und Inbetriebnahme die zu dem betreffenden Zeitpunkt veröffentliche Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiter entwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

EtherCAT®

EtherCAT® ist eine eingetragene Marke und patentierte Technologie lizenziert durch die Beckhoff Automation GmbH, Deutschland

Copyright

1.2 Sicherheitshinweise

Sicherheitsbestimmungen

Beachten Sie die folgenden Sicherheitshinweise und Erklärungen!
Produktspezifische Sicherheitshinweise finden Sie auf den folgenden Seiten oder in den Bereichen Montage, Verdrahtung, Inbetriebnahme usw.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Erklärung der Symbole

In der vorliegenden Dokumentation werden die folgenden Symbole mit einem nebenstehenden Sicherheitshinweis oder Hinweistext verwendet. Die Sicherheitshinweise sind aufmerksam zu lesen und unbedingt zu befolgen!

⚠ GEFAHR

Akute Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht unmittelbare Gefahr für Leben und Gesundheit von Personen!

⚠ WARNUNG

Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht Gefahr für Leben und Gesundheit von Personen!

⚠ VORSICHT

Schädigung von Personen!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, können Personen geschädigt werden!

HINWEIS

Schädigung von Umwelt oder Geräten

Wenn der Hinweis neben diesem Symbol nicht beachtet wird, können Umwelt oder Geräte geschädigt werden.

Tipp oder Fingerzeig

Dieses Symbol kennzeichnet Informationen, die zum besseren Verständnis beitragen.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter https://www.beckhoff.de/secinfo.

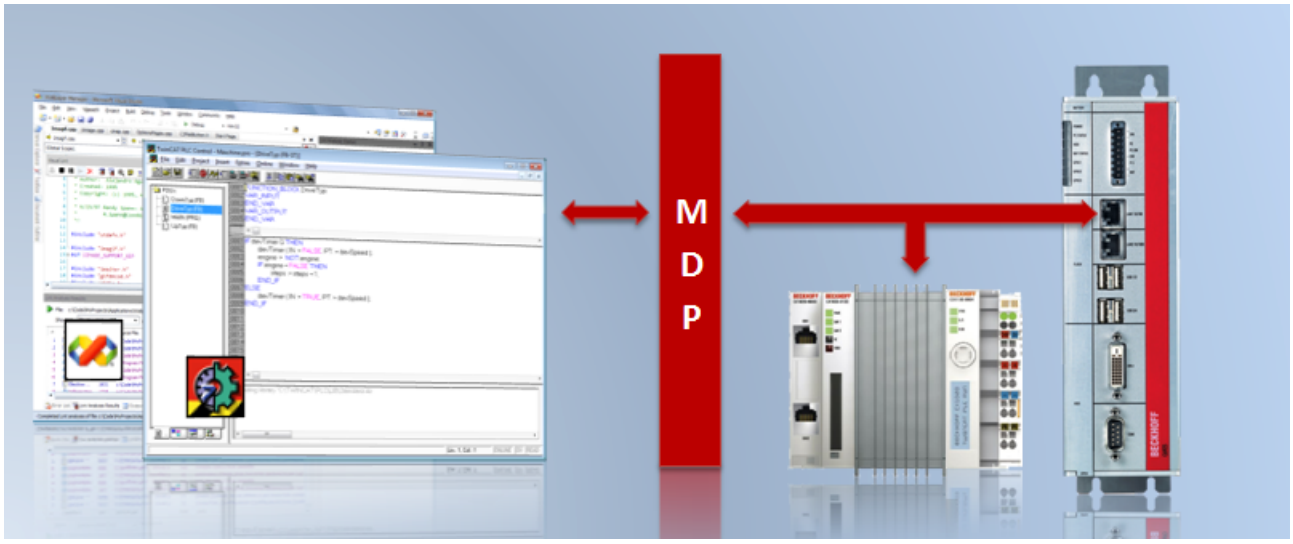
2 Übersicht

Mit den SPS-Funktionsbausteinen kann aus der PLC heraus auf MDP (Modular Device Profile) Informationen zugegriffen werden.

Das **Modular Device Profile for IPC** (MDP) basiert auf der Modular Device Profile Spezifikation der EtherCAT Technology Group. Sämtliche (Soft- und Hardware) Komponenten des Industrie PC bzw. Embedded PC werden in Module unterteilt. Die Liste der verfügbaren Module wird dynamisch je nach physikalisch vorhandenen Komponenten generiert.

Das MDP vereinheitlicht den Zugriff auf Beckhoff Hardware und Software, der bisher abhängig vom eingesetzten Windows Betriebsystem unterschiedlich ausfallen kann.

Die vorliegende Dokumentation bezieht sich auf die TwinCAT PLC Bibliothek TcMDP, mit der Informationen des MDP aus der SPS heraus abgefragt werden können. Weitere Informationen zum generellen MDP und anderen Schnittstellen bietet die Dokumentation zum Beckhoff Device Manager.



Systemvoraussetzungen

- Programmierumgebung:
  - TwinCAT Installation Level: TwinCAT PLC oder höher;
  - TwinCAT System Version 2.11.0 Build 1553 oder höher; alternativ: TwinCAT System Version 2.11.0 R2 Build 2025 oder höher
  - **TcMDP.Lib** Diese PLC Bibliothek muss in dem SPS-Projekt eingebunden sein. Alle anderen Bibliotheken werden automatisch hinzugefügt. (Standard.Lib; TcBase.Lib; TcSystem.Lib; TcUtilities.Lib werden automatisch eingebunden)
- Zielplattform:
  - PC oder CX (x86): XP, XPe, CE (image v3.21c oder höher);
  - CX (ARM): CE (image v3.21c oder höher);
  - C69xx / CP62xx: CE (image v3.21f oder höher);
  - TwinCAT SPS-Laufzeitsystem Version 2.11.0 Build 1541 oder höher;
  - Die Systemvoraussetzungen aus dem Beckhoff Device Manager, auf die sich diese Bibliothek bezieht, sind zu beachten.

Weiterführende Dokumentation

- Beckhoff Device Manager

3 Zugriffsvarianten auf MDP Elemente

Die TwinCAT SPS MDP Bibliothek bietet verschiedenste Funktionsbausteine, um einen umfangreichen Zugriff auf MDP-Daten zu ermöglichen.

Es gibt zwei Grundtypen von Funktionsbausteinen in der Bibliothek.

Zum einen die allgemeinen Funktionsbausteine. Mit ihnen lassen sich beliebige Parameter im MDP mittels diskreten Zugriffes selbst abfragen und setzen.

Des Weiteren bieten spezifische Funktionsbausteine die Möglichkeit, auf bestimmte Daten sowie Gruppierungen von mehreren Daten mit einem Aufruf zuzugreifen. Die hier zur Verfügung stehenden Funktionsbausteine bieten einen schnellen Zugriff auf die wichtigsten MDP-Informationen.

Die Art des MDP-Zugriffs und die Unterschiede beider Typen von Funktionsbausteinen werden im Folgenden näher erläutert. Die Funktionsbausteine besitzen ein einheitliches Erscheinungsbild.

Alle Funktionsbausteine werden mit einer positiven Flanke am Eingang *bExecute* aufgerufen. Danach liefert zyklisches Aufrufen des Funktionsbausteines (*bExecute* = FALSE) das Ergebnis der Abfrage am Ausgang, sobald die Bearbeitung der Abfrage abgeschlossen ist (*bBusy* = FALSE). Weitere Handhabungshinweise liefert das Beispiel [▶ 39] in dieser Dokumentation. Jeder Funktionsbaustein muss so lange aufgerufen werden (*bExecute* = FALSE) bis die interne Bearbeitung abgeschlossen (bBusy = FALSE) ist. Währenddessen sind alle Eingänge des Funktionsbausteins unverändert zu belassen.

Generell ist MDP ein Modell, welche Hardware- und Software Komponenten in Form von Modulen beschreibt. Informationen zu diesen Modulen sowie zum Gerät selbst können abgefragt und geändert werden.

Ein Modul besteht aus einer oder mehreren Tabellen. Jede Tabelle besteht aus einer festen Anzahl von Subindizes. Ein Subindex entspricht einem konkreten Element auf, das zugegriffen werden kann.

Zum Aufbau von MDP finden sich nähere Informationen im MDP Information Model. Dort sind ebenso weitere Zugriffsmöglichkeiten auf das MDP beschrieben.

Allgemeine Funktionsbausteine

Um einen MDP-Parameter abfragen oder setzen zu können, muss die dynamische Modul ID des Moduls bekannt sein in dem sich der Parameter befindet.

Diese wird mit Hilfe des Funktionsbausteines FB_MDP_ScanModules [▶ 20] ermittelt.

Nun können einzelne Parameter mittels FB_MDP_Read [▶ 12] und FB_MDP_Write [▶ 14] gelesen bzw. geschrieben werden. Dabei werden zur Abfrage, neben der dynamischen Modul ID, die Nummer der ausgewählten Tabelle (Table ID), der ausgewählte Subindex innerhalb der Tabelle, sowie weitere Informationen angegeben.

Ebenso kann der komplette Header eines Moduls (ST_MDP_ModuleHeader [▶ 32]) mit dem Funktionsbaustein FB_MDP_ReadModuleHeader [▶ 19] abgefragt werden.
Der komplette Inhalt einer ausgewählten Tabelle innerhalb eines Moduls kann mit dem Funktionsbaustein FB_MDP_ReadModuleContent [▶ 18] abgefragt werden.

Der Funktionsbaustein FB_MDP_ReadModule [▶ 16] bündelt obige Abfragen. Der Funktionsbaustein ermittelt implizit die dynamische Modul ID und fragt Header sowie Tabelle ab.

Der Funktionsbaustein FB_MDP_ReadElement [▶ 15] ermittelt ebenfalls die dynamische Modul ID bereits implizit. Mit ihm kann ein beliebiges einzelnes MDP-Element abgefragt werden.

Bei diesen beiden Funktionsbausteinen ist ein vorheriger Aufruf von FB_MDP_ScanModules demnach nicht nötig.

Spezifische Funktionsbausteine

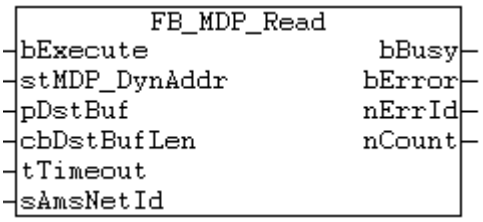
Die hier zur Verfügung stehenden Funktionsbausteine bieten schnellen Zugriff auf die wichtigsten MDP-Informationen.

Beispielsweise reicht der Aufruf des Funktionsbausteines FB_MDP_NIC_Read [▶ 24] aus, um alle wichtigen Informationen über einen Netzwerkadapter abzufragen (siehe MDP NIC Modul). Der Modul Header wird jeweils auch abgefragt und ausgegeben.

Ebenso ermitteln die spezifischen Funktionsbausteine implizit die dynamische Modul ID, so dass ein vorheriger Aufruf von FB_MDP_ScanModules [▶ 20] überflüssig ist.

4 Funktionsbausteine

4.1 FB_MDP_Read



Der Funktionsbaustein ermöglicht das Abfragen eines Elementes eines MDP Moduls.

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 received 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	Mit einer positiven Flanke am Eingang *bExecute* wird der Funktionsbaustein aufgerufen, sofern der Baustein nicht aktiv ist.
stMDP_DynAddr	An diesem Eingang wird die MDP Adressierung angegeben, welche zu dem gewählten Netzwerkmodul gehört. Die Struktur ist vom Typ ST_MDP_Addr [▶ 31]. Die dynamische Modul ID muss bereits mit angegeben werden.
pDstBuf	An diesem Eingang wird die Speicheradresse des Datenpuffers angegeben. Dort werden bei erfolgreicher Abfrage die empfangenen Daten abgelegt.
cbDstBufLen	An diesem Eingang wird die Länge in Byte des Datenpuffers angegeben.
tTimeout	Gibt eine maximale Zeitdauer für die Ausführung des Funktionsbausteines an.
sAmsNetId	Um die Anfrage auf dem lokalen Gerät durchzuführen, bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an einen anderern Computer zu richten kann hier dessen AMS Net Id angegeben werden.

VAR_OUTPUT

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

bBusy	Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.
bError	Wird TRUE, sobald eine Fehlersituation eintritt.
nErrID	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 35].
nCount	Dieser Ausgang gibt die Anzahl der gelesenen Bytes an.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib

4.2 FB_MDP_ReadIndex

Der Funktionsbaustein ermöglicht das Abfragen eines beliebigen Elementes der IPC Diagnose. Neben der Configuration-Area sind auch Daten aus der Device-Area zugänglich.

VAR_INPUT

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

bExecute	Mit einer positiven Flanke am Eingang *bExecute* wird der Funktionsbaustein aufgerufen, sofern der Baustein nicht aktiv ist.
nIndex	An diesem Eingang wird der erste Teil der Adressierung der geforderten IPC Diagnosedaten angegeben.
nSubIndex	An diesem Eingang wird der zweite Teil der Adressierung der geforderten IPC Diagnosedaten angegeben.
pDstBuf	An diesem Eingang wird die Speicheradresse des Datenpuffers angegeben. Dort werden bei erfolgreicher Abfrage die empfangenen Daten abgelegt.
cbDstBufLen	An diesem Eingang wird die Länge in Byte des Datenpuffers angegeben.
tTimeout	Gibt eine maximale Zeitdauer für die Ausführung des Funktionsbausteines an.
sAmsNetId	Um die Anfrage auf dem lokalen Gerät durchzuführen bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an einen anderern Computer zu richten kann hier dessen AMS Net Id angegeben werden.

VAR_OUTPUT

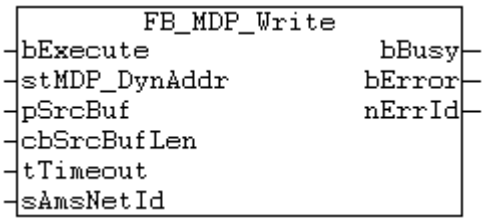
```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrId      : UDINT;
    nCount      : UDINT;
END_VAR
```

bBusy	Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.
bError	Wird TRUE, sobald eine Fehlersituation eintritt.
nErrID	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 35].
nCount	Dieser Ausgang gibt die Anzahl der gelesenen Bytes an.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib [version 1.4.0 oder höher]

4.3 FB_MDP_Write



Der Funktionsbaustein ermöglicht das Setzen eines Elementes eines MDP Moduls.

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 Mit einer positiven Flanke am Eingang *bExecute* wird der Funktionsbaustein aufgerufen, sofern der Baustein nicht aktiv ist.

stMDP_DynAddr An diesem Eingang wird die MDP Adressierung angegeben, welche zu dem gewählten Netzwerkmodul gehört. Die Struktur ist vom Typ ST_MDP_Addr [▶ 31]. Die dynamische Modul ID muss bereits mit angegeben werden.

pSrcBuf An diesem Eingang wird die Speicheradresse des Datenpuffers angegeben. Dort müssen die zu übertragenen Daten abgelegt sein.

cbSrcBufLen An diesem Eingang wird die Länge in Byte des Datenpuffers angegeben.

tTimeout Gibt eine maximale Zeitdauer für die Ausführung des Funktionsbausteines an.

sAmsNetId Um die Anfrage auf dem lokalen Gerät durchzuführen bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an einen anderern Computer zu richten kann hier dessen AMS Net Id angegeben werden.

VAR_OUTPUT

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

bBusy Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.

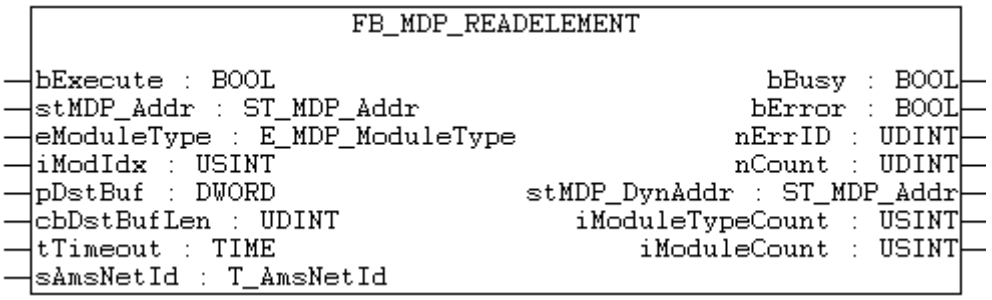
bError Wird TRUE, sobald eine Fehlersituation eintritt.

nErrID Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 35].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib

4.4 FB_MDP_ReadElement



Der Funktionsbaustein ermöglicht das Abfragen eines einzelnen MDP Elementes. Jedes Element aus jedem Modul der Configuration Area kann so gelesen werden!
Intern wird in dem Gerät nach dem gewählten Modul gescannt und mit der dynamischen Modul ID die Elementinformation abgefragt.

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	Mit einer positiven Flanke am Eingang *bExecute* wird der Funktionsbaustein aufgerufen, sofern der Baustein nicht aktiv ist.
stMDP_Addr	An diesem Eingang wird die MDP Adressierung angegeben, welche zu dem gewählten Modul gehört. Die Struktur ist vom Typ ST_MDP_Addr [▶ 31]. Die Area muss als Configuration Area angegeben werden. Die dynamische Modul ID wird erst intern hinzugefügt und darf nicht angegeben werden.
eModuleType	An diesem Eingang wird der MDP Modul Typ angegeben. Die möglichen Typen sind in der Enumeration E_MDP_ModuleType [▶ 31] aufgelistet. (allgemeine Informationen zur Modul Typen Liste)
iModIdx	Falls ein MDP Modul mehrfach vorhanden ist, kann mittels des Eingangs *iModIdx* eine Auswahl (0,...,n) getroffen werden.
pDstBuf	An diesem Eingang wird die Speicheradresse des Datenpuffers angegeben. Dort werden bei erfolgreicher Abfrage die empfangenen Daten abgelegt.
cbDstBufLen	An diesem Eingang wird die Länge in Byte des Datenpuffers angegeben.
tTimeout	Gibt eine maximale Zeitdauer für die Ausführung des Funktionsbausteines an.
sAmsNetId	Um die Anfrage auf dem lokalen Gerät durchzuführen, bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an einen anderen Computer zu richten kann hier dessen AMS Net Id angegeben werden.

VAR_OUTPUT

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

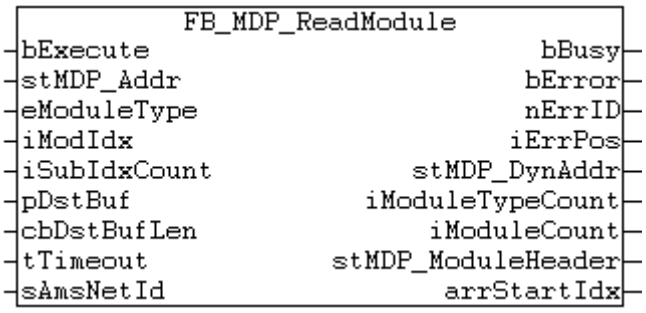
```
    nCount          : UDINT;
    stMDP_DynAddr   : ST_MDP_Addr;        (* includes the new dynamic module type id. *)
    iModuleTypeCount: USINT;            (* returns the number of found modules equal the demanded mo
dule type. *)
    iModuleCount    : USINT;            (* returns the number of all detected MDP modules. *)
END_VAR
```

bBusy	Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.
bError	Wird TRUE, sobald eine Fehlersituation eintritt.
nErrID	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 35].
nCount	Dieser Ausgang gibt die Anzahl der gelesenen Bytes an.
stMDP_DynAddr	An diesem Ausgang wird die MDP Adressierung angegeben, welche zu dem gewählten MDP Modul gehört. Die Struktur ist vom Typ ST_MDP_Addr [▶ 31]. Die dynamische Modul ID wurde durch den Funktionsbaustein hinzugefügt.
iModuleTypeCount	Der Ausgang *iModuleTypeCount* gibt die Anzahl der Module an, welche dem angegebenen Typ entsprechen.
iModuleCount	Der Ausgang *iModuleCount* gibt die komplette Anzahl der Module auf dem Gerät an.

Voraussetzungen

Entwicklungsumgebungg	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib [Version 1.2.0 oder höher]

4.5 FB_MDP_ReadModule



Der Funktionsbaustein ermöglicht das Abfragen eines MDP Moduls.
Intern wird in dem Gerät nach dem gewählten Modul gescannt und mit der dynamischen Modul ID der Modul Header sowie Modulinformationen abgefragt.

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 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	Mit einer positiven Flanke am Eingang *bExecute* wird der Funktionsbaustein aufgerufen, sofern der Baustein nicht aktiv ist.

stMDP_Addr	An diesem Eingang wird die MDP-Adressierung angegeben, welche zu dem gewählten Modul gehört. Die Struktur ist vom Typ ST_MDP_Addr [▶ 31]. Die dynamische Modul ID wird erst intern hinzugefügt.
eModuleType	An diesem Eingang wird der MDP Modul Typ angegeben. Die möglichen Typen sind in der Enumeration E_MDP_ModuleType [▶ 31] aufgelistet. (allgemeine Informationen zur Modul Typen Liste)
iModIdx	Falls ein MDP Modul mehrfach vorhanden ist, kann mittels des Eingangs *iModIdx* eine Auswahl (0,...,n) getroffen werden.
iSubIdxCount	Mit dem Eingang *iSubIdxCount* wird angegeben, wie viele Subindizes der gewählten Table ID abgefragt werden sollen.
pDstBuf	An diesem Eingang wird die Speicheradresse des Datenpuffers angegeben. Dort werden bei erfolgreicher Abfrage die empfangenen Daten abgelegt.
cbDstBufLen	An diesem Eingang wird die Länge in Byte des Datenpuffers angegeben.
tTimeout	Gibt eine maximale Zeitdauer für die Ausführung des Funktionsbausteines an.
sAmsNetId	Um die Anfrage auf dem lokalen Gerät durchzuführen, bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an einen anderen Computer zu richten kann hier dessen AMS Net Id angegeben werden.

VAR_OUTPUT

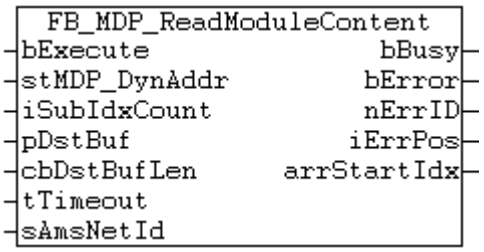
```
VAR_OUTPUT
    bBusy              : BOOL;
    bError             : BOOL;         (* indicates if Read was successfull or not *)
    nErrID             : UDINT;
    iErrPos            : USINT;
    stMDP_DynAddr      : ST_MDP_Addr;  (* includes the new dynamic module type id. *)
    iModuleTypeCount   : USINT;        (* returns the number of found modules equal the demanded mod
ule 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 element
 *)
END_VAR
```

bBusy	Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.
bError	Wird TRUE, sobald eine Fehlersituation eintritt.
nErrID	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 35].
iErrPos	Falls ein Fehler auftrat und sich dieser auf ein einzelnes Element bezieht, gibt dieser Ausgang die Position (Subindex des Elementes) an, an welcher zuerst ein Fehler auftrat.
stMDP_DynAddr	An diesem Ausgang wird die MDP-Adressierung angegeben, welche zu dem gewählten MDP Modul gehört. Die Struktur ist vom Typ ST_MDP_Addr [▶ 31]. Die dynamische Modul ID wurde durch den Funktionsbaustein hinzugefügt.
iModuleTypeCount	Der Ausgang *iModuleTypeCount* gibt die Anzahl der Module an, welche dem angegebenen Typ entsprechen.
iModuleCount	Der Ausgang *iModuleCount* gibt die komplette Anzahl der Module auf dem Gerät an.
stMDP_ModuleHeader	An diesem Ausgang werden die Header Informationen des gelesenen MDP Moduls in Form der Struktur ST_MDP_ModuleHeader [▶ 32]angezeigt.
arrStartIdx	Dieses Array beschreibt, wie die einzelnen abgefragten Subindizes in dem Puffer abgelegt wurden. Der Arrayindex null gibt die Position in Bytes an bei welcher die Daten des Subindex null im Puffer beginnen. Folgende Subindizes sind analog behandelt.

Voraussetzungen

Entwicklungsumgebungg	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib

4.6 FB_MDP_ReadModuleContent



Der Funktionsbaustein ermöglicht das Abfragen des Inhaltes eines MDP Moduls.

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	Mit einer positiven Flanke am Eingang *bExecute* wird der Funktionsbaustein aufgerufen, sofern der Baustein nicht aktiv ist.
stMDP_DynAddr	An diesem Eingang wird die MDP Adressierung angegeben, welche zu dem gewählten Modul gehört. Die Struktur ist vom Typ ST_MDP_Addr [▶ 31]. Die dynamische Modul ID muss bereits mit übergeben werden.
iSubIdxCount	Mit dem Eingang *iSubIdxCount* wird angegeben, wie viele Subindizes der gewählten Table ID abgefragt werden sollen.
pDstBuf	An diesem Eingang wird die Speicheradresse des Datenpuffers angegeben. Dort werden bei erfolgreicher Abfrage die empfangenen Daten abgelegt.
cbDstBufLen	An diesem Eingang wird die Länge in Byte des Datenpuffers angegeben.
tTimeout	Gibt eine maximale Zeitdauer für die Ausführung des Funktionsbausteines an.
sAmsNetId	Um die Anfrage auf dem lokalen Gerät durchzuführen, bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an einen anderern Computer zu richten kann hier dessen AMS Net Id angegeben werden.

VAR_OUTPUT

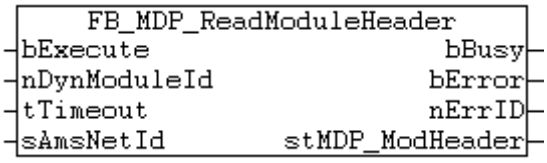
```
VAR_OUTPUT
    bBusy         : BOOL;
    bError        : BOOL;     (* indicates if Read was successfull or not *)
    nErrID        : UDINT;
    iErrPos       : USINT;
    arrStartIdx   : ARRAY[0..255] OF UINT;    (* startindexes in bytes of each subindex element *)
END_VAR
```

bBusy	Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.
bError	Wird TRUE, sobald eine Fehlersituation eintritt.
nErrID	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 35].
iErrPos	Falls ein Fehler auftrat und sich dieser auf ein einzelnes Element bezieht, gibt dieser Ausgang die Position (Subindex des Elementes) an, an welcher zuerst ein Fehler auftrat.
arrStartIdx	Dieses Array beschreibt, wie die einzelnen abgefragten Subindizes in dem Puffer abgelegt wurden. Der Arrayindex null gibt die Position in Bytes an bei welcher die Daten des Subindex null im Puffer beginnen. Folgende Subindizes sind analog behandelt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib

4.7 FB_MDP_ReadModuleHeader



Der Funktionsbaustein ermöglicht das Abfragen des Headers eines MDP Moduls.

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	Mit einer positiven Flanke am Eingang *bExecute* wird der Funktionsbaustein aufgerufen, sofern der Baustein nicht aktiv ist.
stMDP_DynAddr	An diesem Eingang wird die MDP Adressierung angegeben, welche zu dem gewählten Netzwerkmodul gehört. Die dynamische Modul ID muss bereits mit angegeben werden.
tTimeout	Gibt eine maximale Zeitdauer für die Ausführung des Funktionsbausteines an.
sAmsNetId	Um die Anfrage auf dem lokalen Gerät durchzuführen, bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an einen anderern Computer zu richten kann hier dessen AMS Net Id angegeben werden.

VAR_OUTPUT

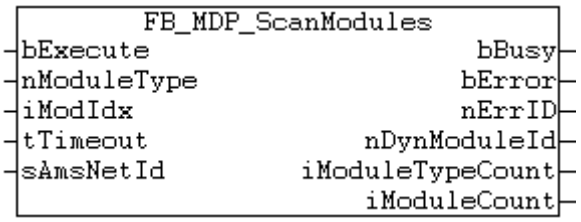
```
VAR_OUTPUT
    bBusy           : BOOL;
    bError          : BOOL;         (* indicates if Read was successfull or not *)
    nErrID          : UDINT;
    stMDP_ModHeader : ST_MDP_ModuleHeader;
END_VAR
```

bBusy	Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.
bError	Wird TRUE, sobald eine Fehlersituation eintritt.
nErrID	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 35].
stMDP_ModuleHeader	An diesem Ausgang werden die Header Informationen des gelesenen MDP Moduls in Form der Struktur ST_MDP_ModuleHeader [▶ 32] angezeigt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib

4.8 FB_MDP_ScanModules



Der Funktionsbaustein ermöglicht das Durchsuchen eines Gerätes nach einem bestimmten MDP Modul.

Bei mehrfachem Vorhandensein des Modultypen kann eine Auswahl getroffen werden. Für den gewählten Modultypen wird durch den Funktionsbaustein die dynamische Module ID ermittelt.

Diese ist wichtiger Bestandteil der MDP Adressierung, welche in der Struktur ST_MDP_Addr [▶ 31] dargestellt ist.

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(i
dx 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	Mit einer positiven Flanke am Eingang *bExecute* wird der Funktionsbaustein aufgerufen, sofern der Baustein nicht aktiv ist.
nModuleType	An diesem Eingang wird der MDP Modul Typ angegeben. Die möglichen Typen sind in der Enumeration E_MDP_ModuleType [▶ 31] aufgelistet. (allgemeine Informationen zur Module Type Liste)
iModIdx	Falls ein MDP Modul mehrfach vorhanden ist, kann mittels des Eingangs *iModIdx* eine Auswahl (0,...,n) getroffen werden. Bei Unsicherheit bezüglich der Auswahl: Informationen um welches Modul es sich explizit handelt, können nach dem Scannen über den Funktionsbaustein FB_MDP_ReadModuleHeader [▶ 19] abgefragt werden.
tTimeout	Gibt eine maximale Zeitdauer für die Ausführung des Funktionsbausteines an.
sAmsNetId	Um die Anfrage auf dem lokalen Gerät durchzuführen, bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an einen anderern Computer zu richten kann hier dessen AMS Net Id angegeben werden.

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 modul
e type. *)
    iModuleCount    : USINT;        (* returns the number of all detected MDP modules. *)
END_VAR
```

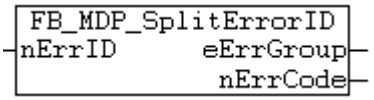
bBusy	Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.
bError	Wird TRUE, sobald eine Fehlersituation eintritt.
nErrID	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 35].
nDynModuleId	Dieser Ausgang gibt die ermittelte dynamische Module ID für das gewählte Modul an.

iModuleTypeCount	Der Ausgang *iModuleTypeCount* gibt die Anzahl der Module an, welche dem angegebenen Typ entsprechen.
iModuleCount	Der Ausgang *iModuleCount* gibt die komplette Anzahl der Module auf dem Gerät an.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib

4.9 FB_MDP_SplitErrorID



Der Funktionsbaustein ermöglicht das Aufsplitten der *nErrID* zu einer Fehlergruppe [▶ 35] und einem spezifischen Fehlercode.
Zur vereinfachten Auswertung der *nErrID* kann demnach dieser Funktionsbaustein herangezogen werden.

VAR_INPUT

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

nErrID	Als Eingang am Funktionsbaustein wird die nErrID angegeben. Diese 4 Byte Variable entspricht dem Ausgang nErrID an einem MDP Funktionsbaustein.

VAR_OUTPUT

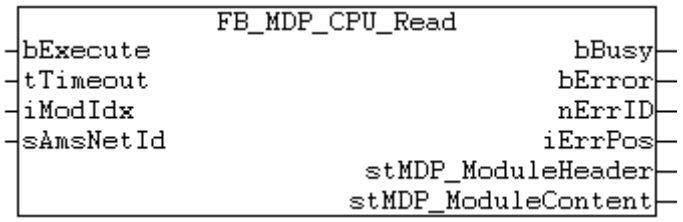
```
VAR_OUTPUT
    eErrGroup    :E_MDP_ErrGroup;  (* type of transmitted error code *)
    nErrCode     :UINT;  (* error code [see specific error type table] *)
END_VAR
```

eErrGroup	Der Ausgang eErrGroup entspricht einem Wert der Enumeration E_MDP_ErrGroup [▶ 35]. Mit Hilfe der Fehlergruppe kann differenziert werden, um welche Art von Fehler bzw. um welche Fehlerquelle es sich handelt.
nErrCode	Der Fehlercode ist spezifisch für jede Fehlergruppe.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib

4.10 FB_MDP_CPU_Read



Der Funktionsbaustein ermöglicht die Abfrage des MDP Moduls CPU. (allgemeine Informationen zum MDP Modul CPU)

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	Mit einer positiven Flanke am Eingang *bExecute* wird der Funktionsbaustein aufgerufen, sofern der Baustein nicht aktiv ist.
tTimeout	Gibt eine maximale Zeitdauer für die Ausführung des Funktionsbausteines an.
iModIdx	Falls ein MDP Modul mehrfach vorhanden ist, kann mittels des Eingangs iModIdx eine Auswahl (0,...,n) getroffen werden.
sAmsNetId	Um die Anfrage auf dem lokalen Gerät durchzuführen bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an einen anderern Computer zu richten kann hier dessen AMS Net Id angegeben werden.

VAR_OUTPUT

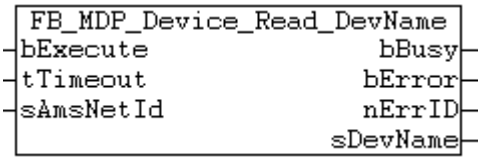
```
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
    nErrID              : UDINT;
    iErrPos             : USINT;
    stMDP_ModuleHeader  : ST_MDP_ModuleHeader;
    stMDP_ModuleContent : ST_MDP_CPU;
END_VAR
```

bBusy	Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.
bError	Wird TRUE, sobald eine Fehlersituation eintritt.
nErrID	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 35].
iErrPos	Falls ein Fehler auftrat und sich dieser auf ein einzelnes Element bezieht, gibt dieser Ausgang die Position (Subindex des Elementes) an, an welcher zuerst ein Fehler auftrat.
stMDP_ModuleHeader	An diesem Ausgang werden die Header Informationen des gelesenen MDP Moduls in Form der Struktur ST_MDP_ModuleHeader [▶ 32] angezeigt.
stMDP_ModuleContent	An diesem Ausgang werden die Informationen der TableID 1 des gelesenen MDP Moduls in Form der Struktur ST_MDP_CPU [▶ 32] angezeigt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib

4.11 FB_MDP_Device_Read_DevName



Der Funktionsbaustein ermöglicht die Abfrage des Gerätenamens. Diese Information befindet sich in der General Area des MDP. (allgemeine Informationen zum 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	Mit einer positiven Flanke am Eingang *bExecute* wird der Funktionsbaustein aufgerufen, sofern der Baustein nicht aktiv ist.
tTimeout	Gibt eine maximale Zeitdauer für die Ausführung des Funktionsbausteines an.
sAmsNetId	Um die Anfrage auf dem lokalen Gerät durchzuführen, bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an einen anderen Computer zu richten kann hier dessen AMS Net Id angegeben werden.

VAR_OUTPUT

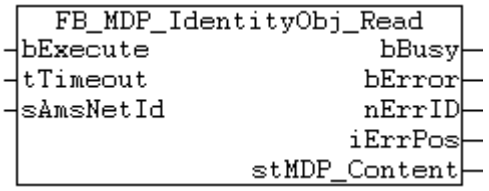
```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    sDevName    : T_MaxString;
END_VAR
```

bBusy	Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.
bError	Wird TRUE, sobald eine Fehlersituation eintritt.
nErrID	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 35].
sDevName	An diesem Ausgang wird der abgefragte Name als String ausgegeben.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib

4.12 FB_MDP_IdentityObj_Read



Der Funktionsbaustein ermöglicht die Abfrage der Tabelle IdentityObject. (allgemeine Informationen zum MDP Modul IdentityObject aus der GeneralArea)

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	Mit einer positiven Flanke am Eingang *bExecute* wird der Funktionsbaustein aufgerufen, sofern der Baustein nicht aktiv ist.
tTimeout	Gibt eine maximale Zeitdauer für die Ausführung des Funktionsbausteines an.
sAmsNetId	Um die Anfrage auf dem lokalen Gerät durchzuführen bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an einen anderern Computer zu richten kann hier dessen AMS Net Id angegeben werden.

VAR_OUTPUT

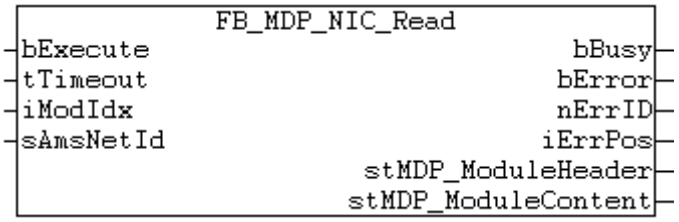
```
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
    nErrID              : UDINT;
    iErrPos             : USINT;
    stMDP_ModuleContent : ST_MDP_IdentityObject;
END_VAR
```

bBusy	Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.
bError	Wird TRUE, sobald eine Fehlersituation eintritt.
nErrID	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 35].
iErrPos	Falls ein Fehler auftrat und sich dieser auf ein einzelnes Element bezieht, gibt dieser Ausgang die Position (Subindex des Elementes) an, an welcher zuerst ein Fehler auftrat.
stMDP_ModuleContent	An diesem Ausgang werden die Informationen der Tabelle in Form der Struktur ST_MDP_IdentityObject [▶ 33] angezeigt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib

4.13 FB_MDP_NIC_Read



Der Funktionsbaustein ermöglicht die Abfrage des MDP-Moduls NIC (Network Interface Card). (allgemeine Informationen zum MDP Modul NIC)

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	Mit einer positiven Flanke am Eingang *bExecute* wird der Funktionsbaustein aufgerufen, sofern der Baustein nicht aktiv ist.
tTimeout	Gibt eine maximale Zeitdauer für die Ausführung des Funktionsbausteines an.
iModIdx	Falls ein MDP Modul mehrfach vorhanden ist, kann mittels des Eingangs iModIdx eine Auswahl (0,...,n) getroffen werden.

sAmsNetId	Um die Anfrage auf dem lokalen Gerät durchzuführen, bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an einen anderern Computer zu richten kann hier dessen AMS Net Id angegeben werden.

VAR_OUTPUT

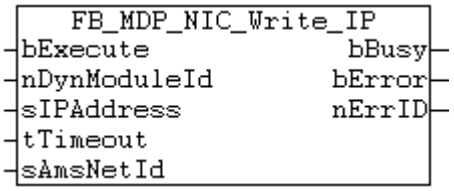
```
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
    nErrID              : UDINT;
    iErrPos             : USINT;
    stMDP_ModuleHeader  : ST_MDP_ModuleHeader;
    stMDP_ModuleContent : ST_MDP_NIC_Properties;
END_VAR
```

bBusy	Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.
bError	Wird TRUE, sobald eine Fehlersituation eintritt.
nErrID	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 35].
iErrPos	Falls ein Fehler auftrat und sich dieser auf ein einzelnes Element bezieht, gibt dieser Ausgang die Position (Subindex des Elementes) an, an welcher zuerst ein Fehler auftrat.
stMDP_ModuleHeader	An diesem Ausgang werden die Header Informationen des gelesenen MDP-Moduls in Form der Struktur ST_MDP_ModuleHeader [▶ 32] angezeigt.
stMDP_ModuleContent	An diesem Ausgang werden die Informationen der TableID 1 des gelesenen MDP-Moduls in Form der Struktur ST_MDP_NIC [▶ 33] angezeigt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib

4.14 FB_MDP_NIC_Write_IP



Der Funktionsbaustein ermöglicht das Setzen einer neuen IP-Adresse. Dieses Element ist Teil des MDP-Moduls NIC. (allgemeine Informationen zum MDP Modul NIC)

Beachten Sie, dass Änderungen dieser Art eine bestehende Netzwerkverbindung zu dem Rechner beeinflussen.

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 ca
```

```
ncelled. *)
    sAmsNetId           : T_AmsNetId;          (* keep empty '' for the local device *)
END_VAR
```

bExecute	Mit einer positiven Flanke am Eingang *bExecute* wird der Funktionsbaustein aufgerufen, sofern der Baustein nicht aktiv ist.
nDynModuleId	An diesem Eingang wird die dynamische Modul ID angegeben, welche zu dem gewählten Netzwerkmodul gehört.
sIPAddress	Die an diesem Eingang angegebene IP-Adresse in Form eines String wird übermittelt.
tTimeout	Gibt eine maximale Zeitdauer für die Ausführung des Funktionsbausteines an.
sAmsNetId	Um die Anfrage auf dem lokalen Gerät durchzuführen, bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an einen anderer Computer zu richten kann hier dessen AMS Net Id angegeben werden.

VAR_OUTPUT

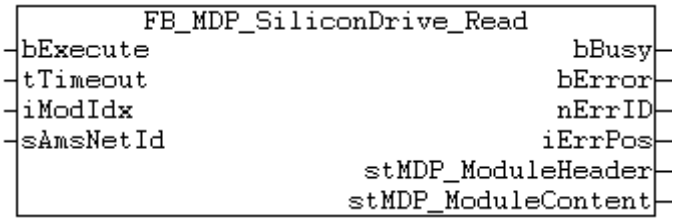
```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
END_VAR
```

bBusy	Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.
bError	Wird TRUE, sobald eine Fehlersituation eintritt.
nErrID	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 35].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib

4.15 FB_MDP_SiliconDrive_Read



Der Funktionsbaustein ermöglicht die Abfrage des MDP Moduls SiliconDrive. (allgemeine Informationen zum MDP Modul SiliconDrive)

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	Mit einer positiven Flanke am Eingang *bExecute* wird der Funktionsbaustein aufgerufen, sofern der Baustein nicht aktiv ist.
tTimeout	Gibt eine maximale Zeitdauer für die Ausführung des Funktionsbausteines an.

iModIdx	Falls ein MDP Modul mehrfach vorhanden ist, kann mittels des Eingangs iModIdx eine Auswahl (0,...,n) getroffen werden.
sAmsNetId	Um die Anfrage auf dem lokalen Gerät durchzuführen, bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an einen anderern Computer zu richten kann hier dessen AMS Net Id angegeben werden.

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	Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.
bError	Wird TRUE, sobald eine Fehlersituation eintritt.
nErrID	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 35].
iErrPos	Falls ein Fehler auftrat und sich dieser auf ein einzelnes Element bezieht, gibt dieser Ausgang die Position (Subindex des Elementes) an, an welcher zuerst ein Fehler auftrat.
stMDP_ModuleHeader	An diesem Ausgang werden die Header Informationen des gelesenen MDP Moduls in Form der Struktur ST_MDP_ModuleHeader [▶ 32]angezeigt.
stMDP_ModuleContent	An diesem Ausgang werden die Informationen der TableID 1 des gelesenen MDP Moduls in Form der Struktur ST_MDP_SiliconDrive [▶ 33] angezeigt.

HINWEIS

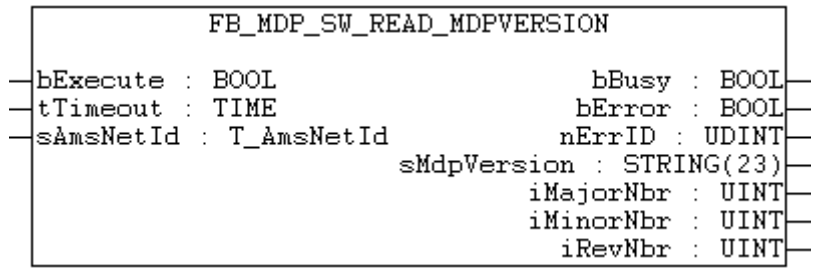
Mögliche Zeitüberschreitung

Die Abfrage des MDP Silicon Drive Moduls gehört zu den zeitintensiveren Bearbeitungen. So kann das Standard ADS Timeout überschritten werden. Eine Erhöhung der am Eingang des Funktionsbausteines angelegten Zeitdauer tTimeout kann Abhilfe schaffen.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib

4.16 FB_MDP_SW_Read_MdpVersion



Der Funktionsbaustein ermöglicht die Abfrage der MDP-Version. Diese Information befindet sich im Modul Software in der Configuration Area des MDP. (allgemeine Informationen zum MDP Information model)

Die MDP-Version ist unabhängig von der Version der SPS-Bibliothek. Um die Version der SPS-Bibliothek abzufragen wird die Funktion F_GetVersionTcMDP [▶ 30] verwendet.

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	Mit einer positiven Flanke am Eingang *bExecute* wird der Funktionsbaustein aufgerufen, sofern der Baustein nicht aktiv ist.
tTimeout	Gibt eine maximale Zeitdauer für die Ausführung des Funktionsbausteines an.
sAmsNetId	Um die Anfrage auf dem lokalen Gerät durchzuführen, bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an einen anderen Computer zu richten kann hier dessen AMS Net Id angegeben werden.

VAR_OUTPUT

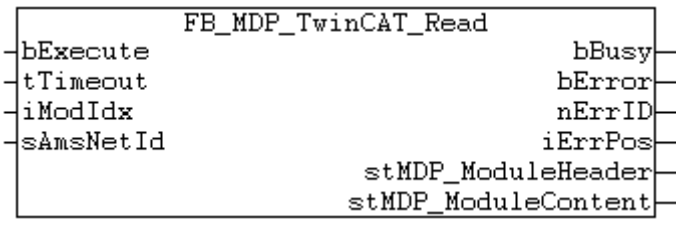
```
VAR_OUTPUT
    bBusy         :BOOL;
    bError        :BOOL;
    nErrID        :UDINT;
    sMdpVersion   :STRING(23);    (* complete MDP version as string [e.g.: '1, 0, 4, 47'] *)
    iMajorNbr     :UINT;      (* major number [e.g.: 1] *)
    iMinorNbr     :UINT;      (* minor number [e.g.: 4] *)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

bBusy	Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.
bError	Wird TRUE, sobald eine Fehlersituation eintritt.
nErrID	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 35].
sMdpVersion	An diesem Ausgang wird die abgefragte MDP Version als String ausgegeben.
iMajorNbr	Die erste Position der Versionsnummer wird mit *iMajorNbr* als Zahl ausgegeben.
iMinorNbr	Die zweite Position der Versionsnummer wird mit *iMinorNbr* als Zahl ausgegeben.
iRevNbr	Die dritte Position der Versionsnummer wird mit *iRevNbr* als Zahl ausgegeben.

Voraussetzungen

Entwicklungsumgebungg	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib [Version 1.2.0 oder höher]

4.17 FB_MDP_TwinCAT_Read



Der Funktionsbaustein ermöglicht die Abfrage des MDP Moduls TwinCAT. (allgemeine Informationen zum MDP Modul TwinCAT)

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	Mit einer positiven Flanke am Eingang *bExecute* wird der Funktionsbaustein aufgerufen, sofern der Baustein nicht aktiv ist.
tTimeout	Gibt eine maximale Zeitdauer für die Ausführung des Funktionsbausteines an.
iModIdx	Falls ein MDP Modul mehrfach vorhanden ist, kann mittels des Eingangs iModIdx eine Auswahl (0,...,n) getroffen werden.
sAmsNetId	Um die Anfrage auf dem lokalen Gerät durchzuführen, bedarf es keiner Angabe dieser Eingangsvariablen. Alternativ kann ein leerer String angegeben werden. Um die Anfrage an einen anderern Computer zu richten kann hier dessen AMS Net Id angegeben werden.

VAR_OUTPUT

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

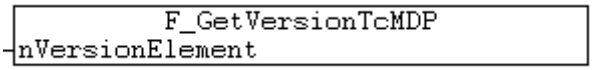
bBusy	Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.
bError	Wird TRUE, sobald eine Fehlersituation eintritt.
nErrID	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 35].
iErrPos	Falls ein Fehler auftrat und sich dieser auf ein einzelnes Element bezieht, gibt dieser Ausgang die Position (Subindex des Elementes) an, an welcher zuerst ein Fehler auftrat.
stMDP_ModuleHeader	An diesem Ausgang werden die Header Informationen des gelesenen MDP Moduls in Form der Struktur ST_MDP_ModuleHeader [▶ 32] angezeigt.
stMDP_ModuleContent	An diesem Ausgang werden die Informationen der TableID 1 des gelesenen MDP Moduls in Form der Struktur ST_MDP_TwinCAT [▶ 34] angezeigt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1551	PC oder CX (x86, ARM)	TcMDP.Lib

5 Funktionen

5.1 F_GetVersionTcMDP



Mit dieser Funktion können Versionsinformationen der SPS-Bibliothek ausgelesen werden.

Die MDP Version ist unabhängig von der Version der SPS-Bibliothek. Um die MDP Version abzufragen wird der Funktionsbaustein FB_MDP_SW_Read_MdpVersion [▶ 27] verwendet.

FUNCTION F_GetVersionTcMDP: UINT

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

nVersionElement : Versionselement, das gelesen werden soll. Mögliche Parameter:

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib

6 Datentypen

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
```

Die Enumeration *E_MDP_AddrArea* definiert konstante Werte für die unterschiedlichen Areas im MDP.

Eine allgemeine Beschreibung findet sich im 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
```

Die Enumeration *E_MDP_ModuleType* definiert konstante Werte für die unterschiedlichen Modul Typen im MDP.
Ein Modul Typ kann mehrfach pro Gerät vorkommen. So hat ein Gerät mit zwei Ethernet-Schnittstellen auch zwei MDP NIC Module.

Detailinformationen zu den Modulen finden sich in der Dokumentation der IPC Diagnostic - Modultypen.

Dieser Modul Typ ist nicht gleichzusetzen mit der dynamischen Modul 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
```

Die Struktur enthält Informationen, welche zur MDP Adressierung benötigt werden.

nArea	Mögliche MDP Areas sind in E_MDP_AddrArea [▶ 31] gelistet.
nModuleId	Die Module ID wird dynamisch zugewiesen. Sie entspricht nicht den Modul Typen, welche in E_MDP_ModuleType gelistet sind. Um für einen speziellen Modultyp eine dynamische Modul ID zu erfahren, kann der Funktionsbaustein FB_MDP_ScanModules [▶ 20] genutzt werden.
nTableId	Dieser Wert legt die Nummer der ausgewählten Tabelle des ausgewählten Moduls fest.
nFlag	Dieser Parameter wird nur intern verwendet. Er bleibt auf dem Defaultwert von 0x00.
nSubIdx	Der Parameter Subindex entsprecht dem Subindex in einer Tabelle in einem MDP Modul.

Detaillierte Informationen zur MDP Adressierung befinden sich in der Dokomentation MIPC 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
```

Die Struktur enthält Geräteinformationen. Diese Informationen entsprechen immer der Table ID 0 eines MDP Modules. Jedes Modul besitzt diesen Modul Header.

iLen : Gibt die Anzahl der Parameter der Table ID, in diesem Falle des Modul Headers, an.

nAddr : Gibt die Adresse des Moduls an.

sType : Gibt den Typ des Moduls an. Mögliche Typen sind in der MDP Module List aufgezählt.

sName : Gibt den Namen dieses MDP Moduls an.

nDevType : Gibt den Typ des MDP Moduls als Code an.

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
```

Die Struktur enthält Informationen zum MDP Modul CPU.

Mittels des Funktionsbausteines FB_MDP_CPU_Read [▶ 21] lassen sich diese kompletten Informationen abfragen.

Die in dieser Struktur vorhandenen Parameter entsprechenden Subindizes der ersten Tabelle (Table ID 1) innerhalb des MDP Moduls CPU.

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
```

Die Struktur enthält Informationen zur Tabelle IdentityObject, welche sich in der MDP General Area befindet.

Mittels des Funktionsbausteines FB_MDP_IdentityObj_Read [▶ 23] lassen sich diese kompletten Informationen abfragen.

Die in dieser Struktur vorhandenen Parameter entsprechenden Subindizes der Tabelle 'Identity Object' innerhalb des MDP Moduls IdentityObject in der 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
```

Die Struktur enthält Informationen zum MDP-Modul NIC (Network Interface Card).

Mittels des Funktionsbausteines FB_MDP_NIC_Read [▶ 24] lassen sich diese kompletten Informationen abfragen.

Die in dieser Struktur vorhandenen Parameter entsprechenden Subindizes der ersten Tabelle (Table ID 1) innerhalb des MDP Moduls NIC.

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
```

Die Struktur enthält Informationen zum MDP Modul Silicon Drive.

Mittels des Funktionsbausteines FB_MDP_SiliconDrive_Read [▶ 26] lassen sich diese kompletten Informationen abfragen.

iLen	iLen gibt die Zahl der MDP Elemente in der Tabelle im MDP Modul an.
iTotalEraseCounts	Dieser Wert gibt die Gesamtanzahl der Schreib- bzw. Löschzyklen von allen Speicherblöcken eines Silicon Drive an. Diese Anzahl liegt als 64 Bit Wert vor.*iTotalEraseCounts* enthält die unteren 32 Bit.
iTotalEraseCountsHigh	Dieser Wert gibt die Gesamtanzahl der Schreib- bzw. Löschzyklen von allen Speicherblöcken eines Silicon Drive an. Diese Anzahl liegt als 64 Bit Wert vor.*iTotalEraseCountsHigh* enthält die oberen 32 Bit.
iDriveUsage	Dies gibt die errechnete Abnutzung des Silicon Drive an. Der Wert basiert auf zwei Millionen Schreibzyklen pro Block als Maximalwert.
iNbrSpares	Spare Blöcke dienen dazu abgenutzte Speicherblöcke zu ersetzen. *iNbrSpares* gibt die Anzahl der Ersatzblöcke, welche auf dem Silicon Drive verfügbar sind, an.
iNbrUsedSpares	Der Wert gibt die Anzahl der Spare Blöcke an, welche bereits in Benutzung sind.

Die in dieser Struktur vorhandenen Parameter entsprechen den Subindizes der ersten Tabelle (Table ID 1) innerhalb des MDP Moduls SiliconDrive.

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
```

Die Struktur enthält Informationen zum MDP Modul TwinCAT.

Mittels des Funktionsbausteines FB_MDP_TwinCAT_Read [▶ 28] lassen sich diese kompletten Informationen abfragen.

Die in dieser Struktur vorhandenen Parameter entsprechenden Subindizes der ersten Tabelle (Table ID 1) innerhalb des MDP Moduls TwinCAT.

7 Fehlercodes

Die Funktionsbausteine der TcPlcLibMDP.Lib besitzen einen Ausgang *nErrID*. Dieser Wert ist 4 Byte groß und liefert im Fehlerfall den Fehlercode. *nErrID* setzt sich aus zwei Teilen zusammen:

(MSB) 2 Byte	2 Byte (LSB)
Error Group	**Error Code**
0x EC80	E_MDP_ErrCodesPLC [▶ 36]
0x ECA6	MDP general error
0x ECA7	MDP API error
0x ECA8	ADS error
0x ECAF	MDP module specific error

Error Group

Die Fehlergruppe beschreibt den Typ des aufgetretenen Fehlers. In der Enumeration E_MDP_ErrGroup [▶ 35] sind die unterschiedlichen Gruppen gelistet.
Alle Fehler die innerhalb der PLC Bibliothek generiert wurden, besitzen die Error Group 0xEC80.

Error Code

Der Fehlercode beschreibt den konkreten Fehler.

Für SPS bibliotheksinterne Fehler mit der Fehlergruppe 0xEC80 sind die Identifier in der Enumeration E_MDP_ErrCodesPLC [▶ 36] gelistet. Eine Beschreibung zu den restlichen Fehlercodes findet sich in der Dokumentation der IPC Diagnose in dem Kapitel MDP Fehlernummern.

> In der Fehlergruppe 16#ECA6 "General error codes" werden allgemeine MDP abhängige Fehler ausgegeben. Teilweise geben diese Fehler an, dass ein Element aus der Elementliste des Moduls nicht verfügbar ist. Bsp.: 16#ECA60105 "No data available" Falls bei einem allgemeinem oder spezifischen Funktionsbaustein (siehe Zugriffsvarianten [▶ 10]) mehrere Elemente zugleich abgefragt werden und eines dieser Elemente nicht verfügbar ist oder einen Fehler aufweist, so zeigt die Ausgangsvariable iErrPos an, an welcher Indexposition (0..n) der Fehler das erste Mal auftrat. Alle Elemente unterhalb dieses Index wurden erfolgreich abgefragt und sind trotz Fehlerausgabe am Ausgang angegeben.

FB_MDP_SplitErrorID

Der Funktionsbaustein FB_MDP_SplitErrorID [▶ 21] ermöglicht eine automatische Trennung der Variablen *nErrID* in Fehlergruppe und Fehlercode.

Beispiel:

nErrID = 0x ECA8 0745

Die Error Group ist 0x ECA8, es handelt sich demnach um einen Ads Fehler.
Der Error Code ist 0x 0745, es handelt sich demnach um einen Timeout Fehler.

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
```

Die Enumeration *E_MDP_ErrGroup* definiert konstante Werte für die unterschiedlichen Fehlergruppen im MDP. Diese geben den Fehlertyp an.

Die Werte finden sich in den Fehlercodes [▶ 35] wieder, welche im Fehlerfall am Ausgang eines PLC MDP Funktionsbausteines liegen.

Eine allgemeine Beschreibung findet sich im MDP Information Model in dem Kapitel Return Values. Dort sind einzelne Fehlercodes aus den Fehlergruppen 16#ECA6 - 16#ECAF beschrieben.

Die Fehlercodes der Gruppe 16#EC80 sind von der PLC MDP Bibliothek erzeugt und werden im Kapitel E_MDP_ErrCodesPLC [▶ 36] beschrieben.

7.2 E_MDP_ErrCodesPLC

HINWEIS

Mögliche Zeitüberschreitung

Je nach MDP-Abfrage kann die Bearbeitung unterschiedlich lange dauern. Aufgrund der internen Prozesse kann die Bearbeitungszeit teilweise das Standard ADS Timeout überschreiten. Eine Erhöhung der am Eingang des Funktionsbausteines angelegten Zeitdauer tTimeout kann Abhilfe schaffen.

```
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
```

Die Enumeration *E_MDP_ErrCodesPLC* definiert konstante Werte für die unterschiedlichen Fehler, welche Bibliotheksintern generiert, werden können.

Diese Werte finden sich in den Fehlercodes [▶ 35] wieder, welche im Fehlerfall am Ausgang eines PLC MDP Funktionsbausteines liegen.

eMDP_ErrPLC_TimeOut	Der Fehler *eMDP_ErrPLC_TimeOut* wird generiert, wenn die am Eingang des Funktionsbausteines angelegte Zeitdauer *tTimeout* abgelaufen ist.
eMDP_ErrPLC_ModuleNotFound	Im MDP existiert eine Liste von aktiven Modulen. Die Funktionsbausteine der PLC MDP Bibliothek suchen diese Liste nach dem gefragten Modul ab. Falls die Liste das Modul nicht enthält, so wird der Fehler *eMDP_ErrPLC_ModuleNotFound* ausgegeben. Dies ist der Fall, wenn das bestimmte Modul/Gerät nicht auf dem System installiert oder gar nicht vorhanden ist.
eMDP_ErrPLC_BufferTooSmall	Wurde am Eingang des Funktionsbausteines ein Puffer mittels Pointer angegeben, so ist es möglich, dass dieser nicht ausreichend groß ist für die vorhandenen Daten. In diesem Fall wird der Fehler *eMDP_ErrPLC_BufferTooSmall* ausgegeben.

eMDP_ErrPLC_ElementNotFound	Die Abfrage eines bestimmten Elementes war nicht erfolgreich. Das Element wurde nicht gefunden. Möglicherweise ist das bestimmte Modul oder Element gar nicht auf dem System vorhanden.

Eine allgemeine Beschreibung findet sich im Information Model.

8 Beispiele

Folgende Beispiele werden zur TwinCAT SPS Bibliothek Modular Device Profile angeboten:

Sample - Lesezugriff auf MDP Elemente

Dieses Beispiel bietet eine Einführung in die Handhabung der Funktionsbausteine, welche mit der TcPlcMDP Bibliothek zur Verfügung stehen.

Das Beispiel widmet sich dem Ziel, den Zustand der Compact Flash Karte im Embedded-PC zu ermitteln. Dies kann über einen Parameter im MDP Model herausgefunden werden. Die Abfrage anderer Parameter findet analog zu diesem Beispiel statt.
So kann dieses Beispiel auch als Anleitung gesehen werden, um einen beliebigen MDP Parameter aus einem MDP Modul abzufragen.

Schritt-für-Schritt Erläuterung zu diesem Beispiel [▶ 39]

Download:
https://infosys.beckhoff.com/content/1031/tcplclibmdp/Resources/11941226379.zip

Sample2 - Schreibzugriff auf MDP Elemente

Dieses Beispiel zeigt, dass der Schreibzugriff auf MDP Elemente auf ähnliche Weise zu implementieren ist. In diesem Beispiel wird eine neue IP-Adresse vergeben. Dazu wird zuerst DHCP deaktiviert und danach eine beliebige neue IP-Adresse gesetzt.

Download:
https://infosys.beckhoff.com/content/1031/tcplclibmdp/Resources/11941227787.zip

Sample3 - Abfrage eines Moduls auf zwei Arten

Dieses Beispiel zeigt zwei verschiedene Wege auf, um Informationen aus dem MDP Modul CPU abzufragen.

1. Die Abfrage des MDP Moduls CPU mittels des spezifischen Funktionsbausteines.
2. Die konkrete Abfrage einzelner Elemente aus dem MDP Modul CPU mittels des generellen Funktionsbausteines FB_MDP_ReadElement.
   Die Abfrage eines beliebigen Elementes aus MDP Modulen ist analog möglich. Die Anpassung an ein beliebiges anderes Element ist sehr einfach!

Für das Sample3 wird die SPS-Bibliothek Version 1.2.0 oder höher benötigt.

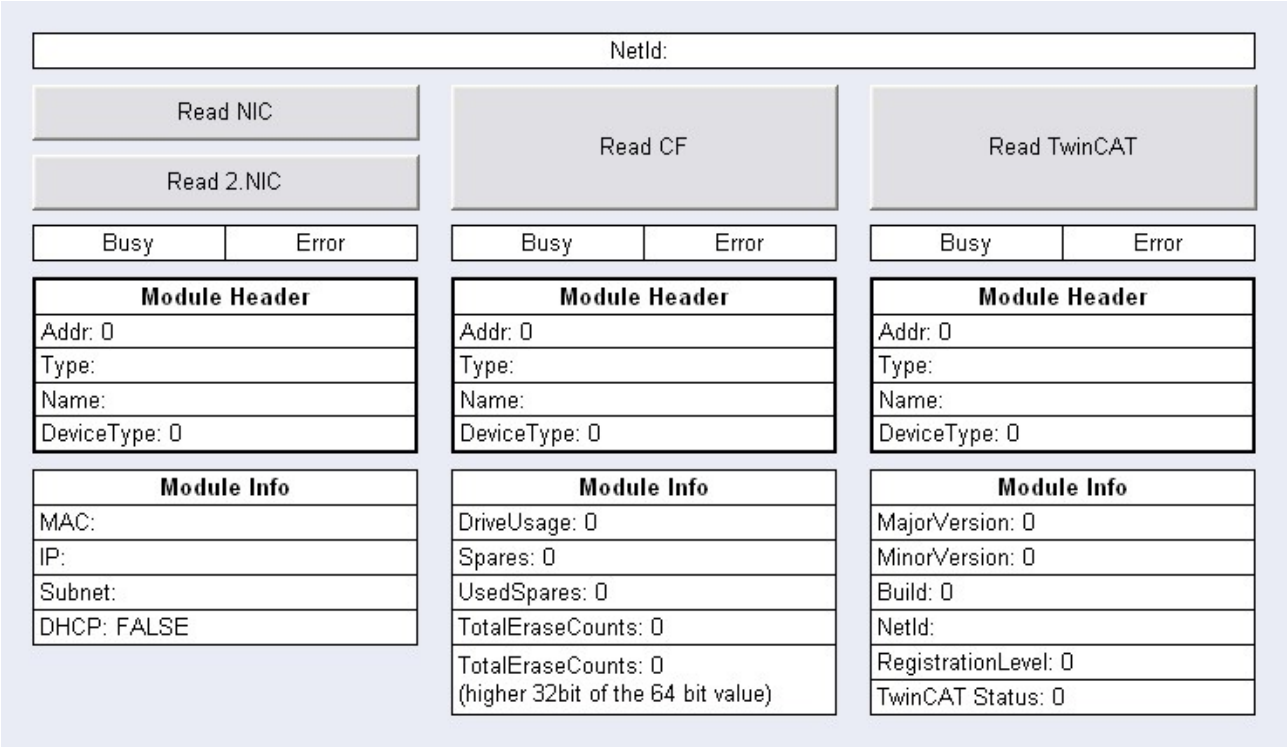
Die CPU Temperatur ist erst ab MDP Version 1.5.0 integriert. Ebenso wird dieser Parameter nicht von jeder Hardware unterstützt.

Download:
https://infosys.beckhoff.com/content/1031/tcplclibmdp/Resources/11941229195.zip

Sample4 - Abfrage kompletter Module mittels der spezifischen Funktionsbausteine

Dieses Beispiel zeigt den einfachsten Zugriff auf verschiedene MDP Module. Dabei werden die spezifischen Funktionsbausteine verwendet, um die Modulinformationen abzufragen.

Für das Sample4 wird die SPS-Bibliothek Version 1.3.0 oder höher empfohlen.



Download:
https://infosys.beckhoff.com/content/1031/tcplclibmdp/Resources/11941230603.zip

8.1 Beispiel

Dieses Beispiel bietet eine Einführung in die Handhabung der Funktionsbausteine, welche mit der TcPlcMDP Bibliothek zur Verfügung stehen.

Dieses Beispiel widmet sich dem Ziel, den Zustand der Compact Flash Karte im Embedded-PC zu ermitteln.

Dies kann über einen Parameter im MDP-Model herausgefunden werden. Die Abfrage anderen Parameters findet analog zu diesem Beispiel statt.

So kann dieses Beispiel auch als Anleitung gesehen werden, um einen beliebigen MDP-Parameter aus einem MDP-Modul abzufragen.

Falls Sie das Beispiel an einem PC durchgehen möchten, welcher keine Silicon Compact Flash Karte als Speicher nutzt, können Sie anstatt dessen beispielsweise die CPU-Auslastung abfragen. Dazu führen Sie dieses Beispiel auf gleiche Weise aus und passen nur wenige Stellen entsprechend an. Nötige Werte dazu finden Sie in der allgemeinen Modulbeschreibung zum MDP Modul CPU.

Übersicht

Folgende Schritte werden nun durchgeführt:

1. Installation der PLC Bibliothek
2. Programmstruktur
3. Dynamische Modul ID ermitteln
4. Abfrage des MDP Parameters
5. Test

1. Installation der PLC Bibliothek

Starten Sie TwinCAT PLC Control.

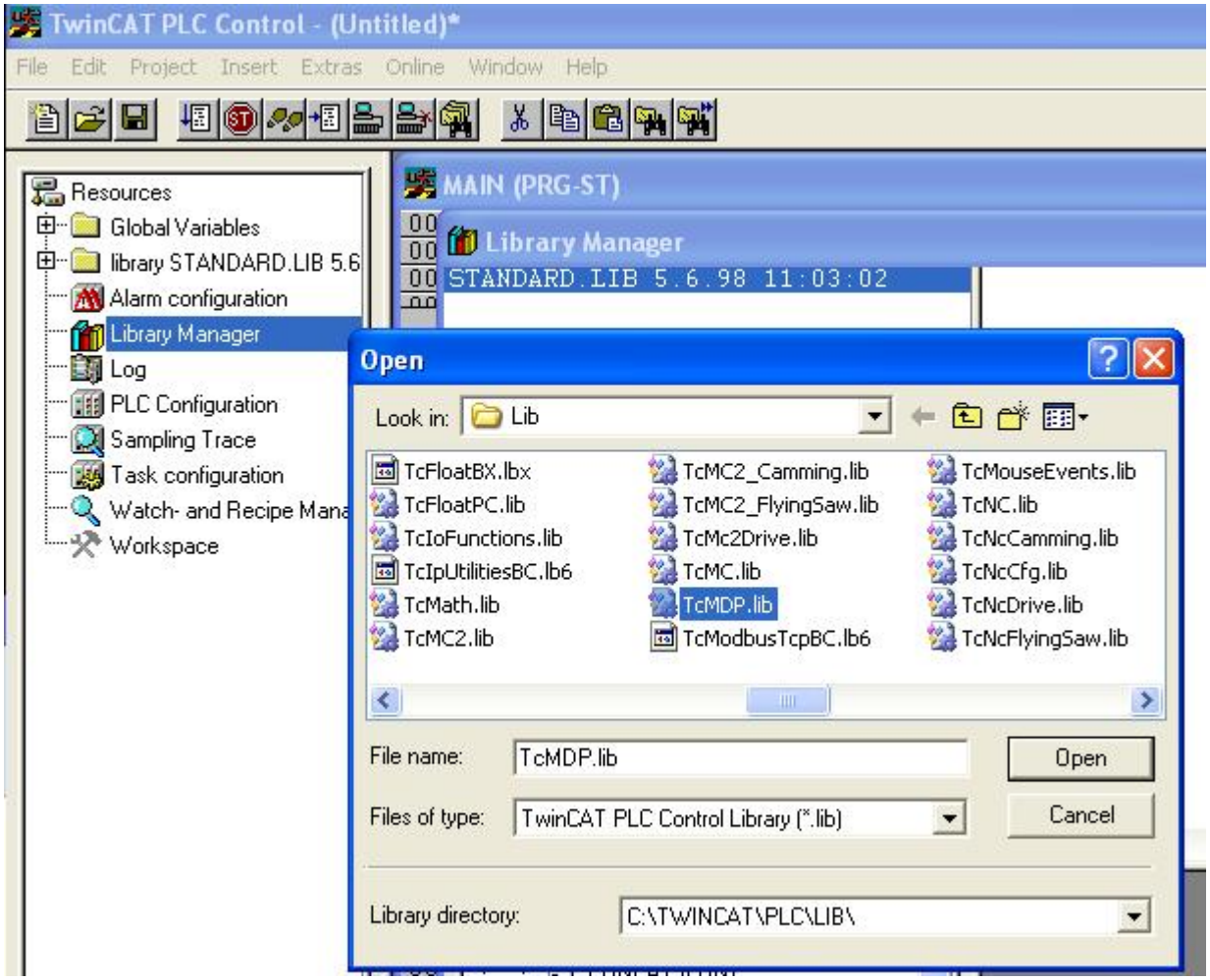
Mit 'Datei > Neu' legen Sie ein neues PLC/SPS Projekt an.

Wählen Sie Ihre Zielplattform PC und CX (x86) oder CX (ARM).

Ihre erste POU ist ein Programm namens MAIN und in der Programmiersprache ST (Strukturierter Text).

Öffnen Sie den Karteireiter Ressourcen und den Bibliotheksverwalter.

Fügen Sie wie im Bild unten dargestellt mit 'Einfügen > Weitere Bibliothek' die Bibliothek TcMDP.lib ein.



Jetzt stehen Ihnen alle SPS-Bausteine der TwinCAT PLC MDP Bibliothek zur Verfügung. Alle weiteren implizit benötigten Bibliotheken wurden automatisch mit der TcMDP.lib eingebunden.

2. Programmstruktur

Der Zustand der Compact Flash Karte wird von einem Parameter im MDP repräsentiert. Um diesen einzelnen Parameter abzufragen, muss die dynamische Modul ID des Moduls bekannt sein, in dem sich der Parameter befindet.

Mit dem Funktionsbaustein FB_MDP_ScanModules muss diese dynamische Modul ID ermittelt werden.

Daraufhin kann mit dem Funktionsbaustein FB_MDP_Read der Parameter abgefragt werden.

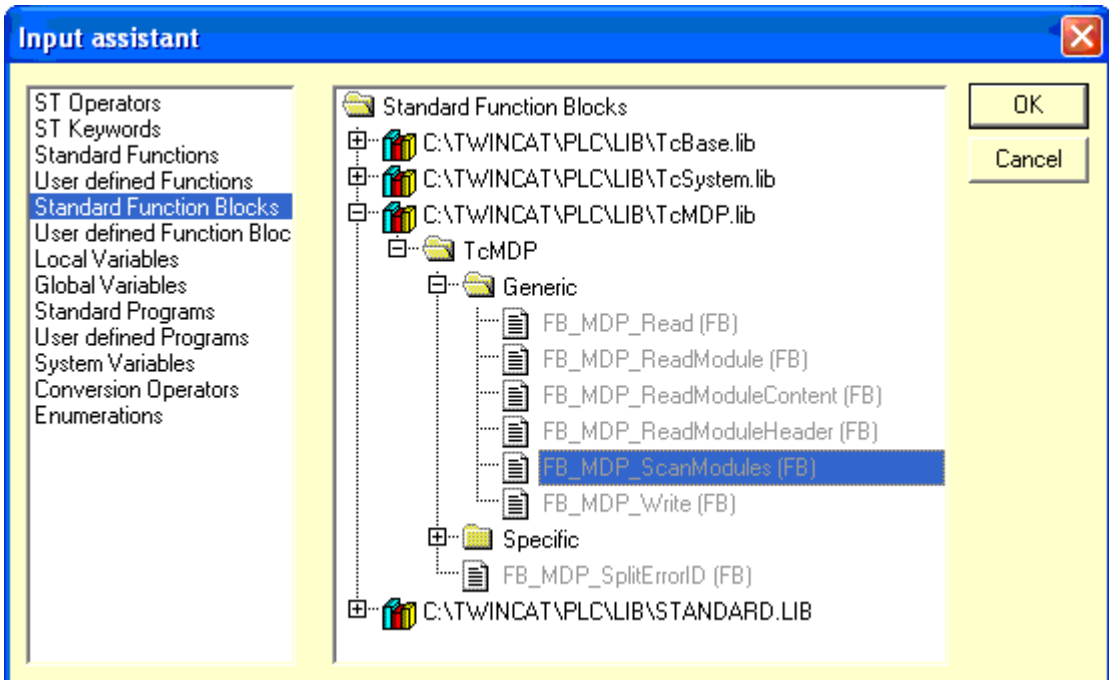
Erzeugen Sie für diesen Ablauf im MAIN Programm eine Zustandsmaschine.

```
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. Dynamische Modul ID ermitteln

Fügen Sie den MDP Funktionsbaustein FB_MDP_ScanModules [▶ 20] ein (F2 drücken).



Im Zustand 1 starten Sie den Funktionsbaustein, indem Sie am Eingang *bExecute* TRUE definieren.
Bei *nModuleType* wird der Enumerationswert eMDP_ModT_SiliconDrive [▶ 31] angegeben, welcher Zugriff auf Informationen von Compact Flash Karten erlaubt. Weitere Informationen zu diesem Modul: allgemeine Informationen zum MDP Modul SiliconDrive
Den Eingang iModIdx müssen Sie nicht belegen. Sie rufen so automatisch das erste gefundene Modul des gewählten Modultypen auf (default: iModIdx := 0).
Ebenso müssen Sie den Eingang tTimeout nicht belegen, sondern können mit dem Default (DEFAULT_ADS_TIMEOUT) arbeiten.

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

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

Im Zustand 2 rufen Sie diesen Funktionsbaustein zyklisch mit dem Eingang *bExecute* FALSE auf. Der Funktionsbaustein wird so lange in diesem Zustand aufgerufen, wie er nicht mit der Bearbeitung der Abfrage beschäftigt ist.
Sobald der Ausgang *bBusy* FALSE ist, kann in den nächsten Zustand übergegangen werden.

Ihr Programm sollte nun folgendermaßen aussehen:

```
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. Abfrage des MDP Parameters

Der MDP Parameter den Sie abfragen möchten ist befindet sich in einer bestimmten Tabelle.
Diese wiederum befindet sich in einem bestimmten Modul, welches einer Area angehört.
Diese Werte entnehmen Sie der MDP Beschreibung:

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

Ausschnitt aus den allgemeinen Informationen zum MDP Modul SiliconDrive.

Zur Abfrage eines MDP Elementes deklarieren Sie eine Instanz des Funktionsbausteines FB_MDP_Read [▶ 12].
Ebenso deklarieren Sie eine Variable iDriveUsage, wobei es sich um eine Unsigned16 Variable handelt.

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

Die ermittelten Werte für den gesuchten MDP Parameter übergeben Sie dem Funktionsbaustein. Dazu wählen Sie die Eingangsvariable *stMDP_DynAddr* vom Typ ST_MDP_Addr [▶ 31] des Funktionsbausteines und weisen die Werte zu.

```
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;
```

Im Zustand 3 rufen Sie des Weiteren den Funktionsbaustein auf und starten ihn, indem Sie am Eingang *bExecute* TRUE anlegen.
Den Eingang *stMDP_DynAddr* haben Sie bereits explizit zugewiesen.
Als Datenpuffer geben Sie bei *pDstBuf* und *cbDstBufLen* die Adresse und die Länge Ihrer Variablen *iDriveUsage* an.
Wie bei obigem Funktionsbaustein müssen Sie den Eingang tTimeout nicht belegen, sondern können mit dem Default (DEFAULT_ADS_TIMEOUT) arbeiten.

Der Programmteil sollte nun folgendermaßen aussehen:

```
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

Kompilieren Sie das erstellte PLC-Programm.
Stellen Sie sicher, dass sich TwinCAT auf dem gewünschten System im Run Modus befindet.
Führen Sie von TwinCAT PLC Control aus einem Login auf dem gewünschten Run-Time System durch.
Starten Sie das SPS-Programm.

Durch die Initialisierung von bStartRequest mit TRUE (siehe 2.Programmstruktur) werden alle Zustände der Zustandsmaschine zum Programmstart sofort einmalig ausgeführt.
In Ihrer Variablen *iDriveUsage* ist nun bei fehlerfreier Ausführung der abgefragte Wert abgelegt. Dieser Wert gibt in diesem Beispiel an, zu welchem Anteil in % die Compact Flash Karte bereits die statistisch mögliche Anzahl von Schreibzyklen vollzogen hat und ist somit eine sinnvolle Information bzgl. der Lebensdauer Ihrer CF-Karte.
Falls Sie dieses Beispiel durchgeführt haben mit dem Ziel die CPU-Auslastung abzufragen, befindet sich nun in Ihrer Variablen die Auslastung der CPU in %.

Um die komplette Abfrage erneut zu starten, setzen Sie Ihre Variable *bStartRequest* erneut auf TRUE (beispielsweise per Online Write).

Dieses Beispiel kann auch als allgemeine Anleitung dienen. Auf analoge Weise lässt sich jeder MDP Parameter aus einem MDP Modul abfragen.

Zum Speichern dieses Beispielprogramms hier klicken:
https://infosys.beckhoff.com/content/1031/tcplclibmdp/Resources/11941226379.zip.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v2.11.0 Build >= 1541	PC oder CX (x86, ARM)	TcMDP.Lib

8.2 IPC-Seriennummern lesen

Dieses Beispiel zeigt den Zugriff auf die Seriennummer des IPCs sowie die Seriennummer des Mainboards des IPC

- Die Seriennummer des Mainboards ist über einen SubIndex im Modul Mainboard in der Configuration Area der IPC-Diagnose auslesbar. Dazu wird der generelle Funktionsbaustein FB_MDP_ReadElement verwendet
- Die Seriennummer des IPCs ist über den Index 0xF9F0 der Device Area der IPC-Diagnose auslesbar. Dazu wird der generelle Funktionsbaustein FB_MDP_ReadIndex verwendet.

Beispiel zur Abfrage der Seriennummern eines Beckhoff IPCs

Enumerationsdefinition

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

Variablendeklaration

```
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
```

Programmcode

```
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
```

Rückgabe Seriennummer des Mainboards statt Seriennummer des IPCs

Bei älteren BIOS-Versionen (vor Q4/2013) wurde die Seriennummer noch nicht im IPC BIOS gespeichert. In diesen Fällen ist der Rückgabewert die Seriennummer des IPC Mainboards. Bei älteren Beckhoff Automation Device-Driver-Versionen ist der Rückgabewert ebenfalls die Seriennummer des IPC Mainboards. Die Seriennummer des IPC Mainboards kann immer über das Mainboard Modul gelesen werden.

Mehr Informationen:
www.beckhoff.de/tx1200

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Deutschland
Telefon: +49 5246 9630
info@beckhoff.de
www.beckhoff.de