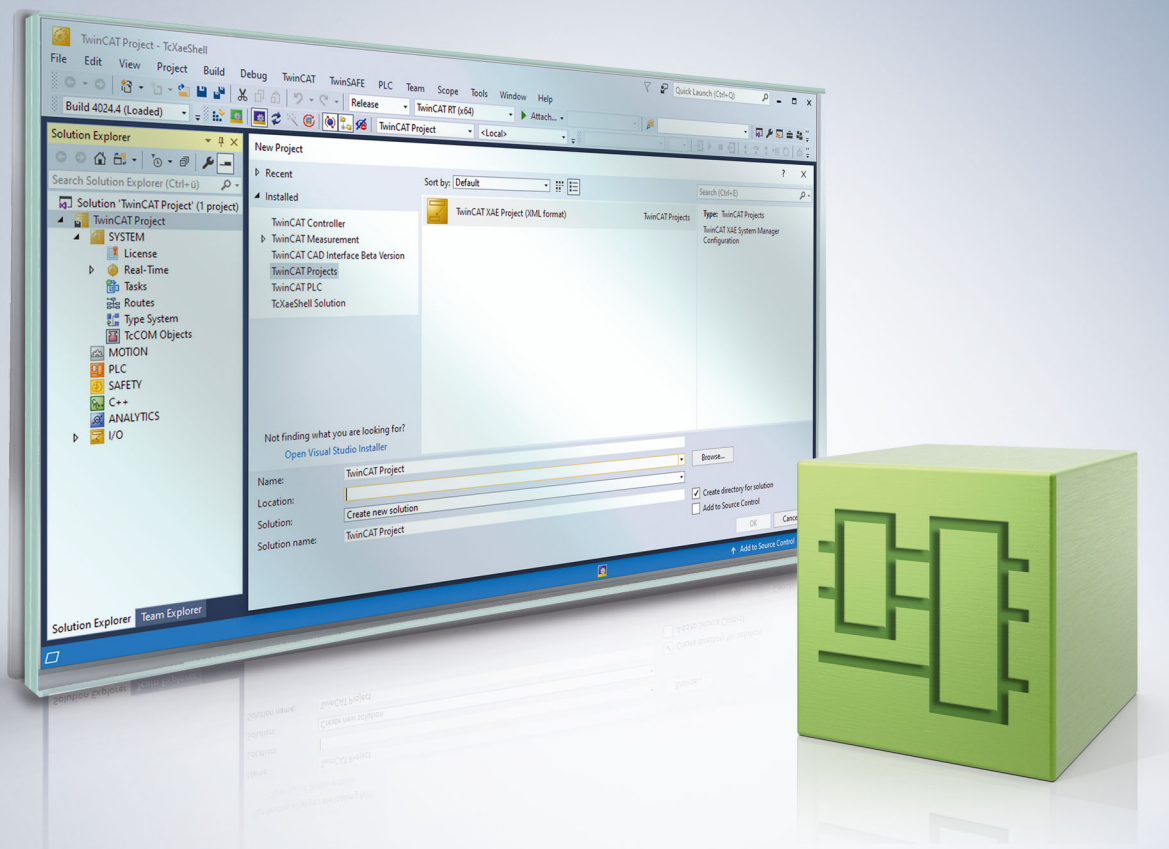
BECKHOFF New Automation Technology

Handbuch | DE

TE1000

TwinCAT 3 | PLC-Bibliothek: Tc2_MDP



18.10.2023 | Version: 1.7.1

Inhaltsverzeichnis

1 Vorwort ... 5
1.1 Hinweise zur Dokumentation ... 5
1.2 Zu Ihrer Sicherheit ... 6
1.3 Hinweise zur Informationssicherheit ... 7
2 Übersicht ... 8
3 Zugriffsvarianten auf MDP Elemente ... 9
4 Funktionsbausteine ... 10
4.1 Einführung ... 10
4.2 Generisch ... 11
4.2.1 Advanced ... 11
4.2.2 FB_MDP_ReadElement ... 14
4.2.3 FB_MDP_ReadModule ... 15
4.2.4 FB_MDP_ReadModuleContent ... 17
4.2.5 FB_MDP_ReadModuleHeader ... 18
4.2.6 FB_MDP_ScanModules ... 19
4.2.7 FB_MDP_SplitErrorID ... 20
4.3 Spezifisch ... 21
4.3.1 FB_MDP_CPU_Read ... 21
4.3.2 FB_MDP_Device_Read_DevName ... 22
4.3.3 FB_MDP_IdentityObj_Read ... 22
4.3.4 FB_MDP_NIC_Read ... 23
4.3.5 FB_MDP_NIC_Write_IP ... 24
4.3.6 FB_MDP_SiliconDrive_Read ... 25
4.3.7 FB_MDP_SW_Read_MdpVersion ... 26
4.3.8 FB_MDP_TwinCAT_Read ... 27
5 Datentypen ... 29
5.1 Allgemeine Datentypen ... 29
5.1.1 E_MDP_AddrArea ... 29
5.1.2 E_MDP_ModuleType ... 29
5.1.3 ST_MDP_Addr ... 29
5.1.4 ST_MDP_ModuleHeader ... 30
5.2 Strukturen spezifischer MDP Module ... 30
5.2.1 ST_MDP_CPU ... 30
5.2.2 ST_MDP_IdentityObject ... 31
5.2.3 ST_MDP_NIC_Properties ... 31
5.2.4 ST_MDP_SiliconDrive ... 31
5.2.5 ST_MDP_TwinCAT ... 32
6 Globale Konstanten ... 33
6.1 Global_Version ... 33
7 Fehlercodes ... 34
7.1 Übersicht der Fehlercodes ... 34
7.2 E_MDP_ErrGroup ... 34
7.3 E_MDP_ErrCodesPLC ... 35

8 Beispiele 37
8.1 Abfrage von CPU-Daten (generisch) 37
8.2 Abfrage von CPU-Daten (spezifisch) 46
8.3 Abfrage des Lüfterstatus (generisch) 50
8.4 IPC-Seriennummern lesen 52
8.5 IP-Adresse setzen 54

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, stets die aktuell gültige 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 weiterentwickelt.
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® ist eine eingetragene Marke und patentierte Technologie lizenziert durch die Beckhoff Automation GmbH, Deutschland

Copyright

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

Lesen Sie die folgenden Erklärungen zu Ihrer Sicherheit.
Beachten und befolgen Sie stets produktspezifische Sicherheitshinweise, die Sie gegebenenfalls an den entsprechenden Stellen in diesem Dokument vorfinden.

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.

Signalwörter

Im Folgenden werden die Signalwörter eingeordnet, die in der Dokumentation verwendet werden. Um Personen- und Sachschäden zu vermeiden, lesen und befolgen Sie die Sicherheits- und Warnhinweise.

Warnungen vor Personenschäden

⚠ GEFAHR

Es besteht eine Gefährdung mit hohem Risikograd, die den Tod oder eine schwere Verletzung zur Folge hat.

⚠ WARNUNG

Es besteht eine Gefährdung mit mittlerem Risikograd, die den Tod oder eine schwere Verletzung zur Folge haben kann.

⚠ VORSICHT

Es besteht eine Gefährdung mit geringem Risikograd, die eine mittelschwere oder leichte Verletzung zur Folge haben kann.

Warnung vor Umwelt- oder Sachschäden

HINWEIS

Es besteht eine mögliche Schädigung für Umwelt, Geräte oder Daten.

Information zum Umgang mit dem Produkt

Diese Information beinhaltet z. B.:
Handlungsempfehlungen, Hilfestellungen oder weiterführende Informationen zum Produkt.

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.

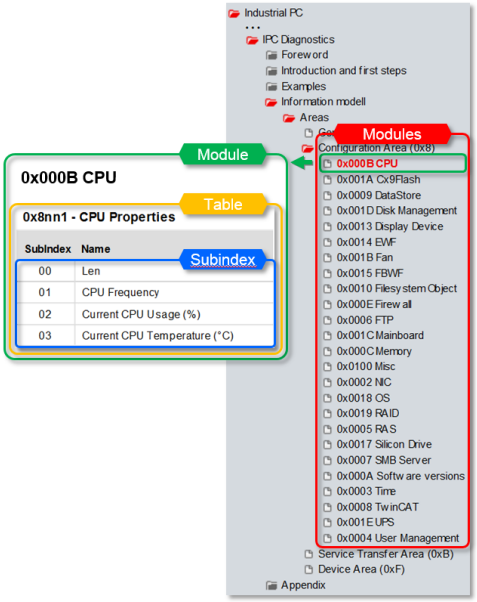
2 Übersicht

Update: Tc3_IPCDiag Bibliothek

Die TwinCAT 3 SPS Bibliothek Tc2_MDP ist der Vorgänger zur Tc3_IPCDiag. Mit der neuen Tc3_IPCDiag Bibliothek wurde zum einen die Menge lesbarer Parameter vergrößert und zum anderen die Anwenderschnittstelle optimiert. Es wird empfohlen die Tc3_IPCDiag Bibliothek zu verwenden.
Zukünftige Erweiterungen werden nicht mehr in der Tc2_MDP Bibliothek durchgeführt. Eine Verwendung der Tc2_MDP Bibliothek für neue Projekte wird nicht empfohlen. Alle Funktionalitäten der Tc2_MDP Bibliothek sind ebenfalls in der neuen Tc3_IPCDiag Bibliothek zu finden.

Die TwinCAT 3 SPS Bibliothek Tc2_MDP wird von der Beckhoff IPC Diagnose genutzt. Details finden Sie in der Dokumentation: Beckhoff Device Manager.

Die verfügbaren Daten der IPC-Diagnose sind in der Configuration Area in sogenannten „Modulen“ organisiert. Ein Modul beinhaltet alle Daten zu einem bestimmten Themenbereich, im Beispiel der IPC CPU.



Ein Modul kann wiederum Unterkategorien, sogenannte „Tables“, beinhalten. Ein Table organisiert seine in ihm enthaltenen Detailinformationen in sogenannten „Subindizes“. Da der Inhalt der Liste von der im aktuellen IPC vorhandenen Komponenten abhängt, wird die Liste dynamisch generiert – je nachdem, welche Komponenten der aktuelle PC beinhaltet, bzw. welche Informationstypen er unterstützt.

Beispiel: Der Zugriff auf Daten des Mainboards erfordert ein BIOS (und die entsprechende Hardware im PC), das diese Daten liefern kann.

Ein Modul kann also nicht über eine feste Adresse angesprochen werden, sondern es muss zuvor ermittelt werden, wo dieses Modul genau zu finden ist.

Eingeschränkter Zugriff zum Zeitpunkt des Systemstarts

MDP bildet eine Schnittstelle zur Hardware. Diese ist unabhängig von TwinCAT. Mit der SPS Bibliothek kann aus TwinCAT heraus auf MDP zugegriffen werden. Dies geschieht intern mittels ADS Kommunikation. Die Vielseitigkeit der Hardwarekonfiguration begründet eine unterschiedlich lange Initialisierungsphase des MDP Dienstes. Es ist möglich, dass erste SPS Zyklen ausgeführt werden während die MDP Initialisierung noch nicht abgeschlossen ist.
Entweder kann auf die möglichen Fehlerausgaben sowie Timeouts der Funktionsbausteine aus der Bibliothek reagiert werden und eine erneute Abfrage getriggert werden oder die Abfragen werden bewusst verzögert nach dem Systemstart ausgeführt.
Es wird empfohlen im SPS-Programm nicht sofort nach dem Systemstart Werte aus dem MDP abzufragen, sondern eine kleine Wartezeit zu berücksichtigen. Wie groß diese sein sollte, hängt von verschiedenen Parametern (wie der Performance Ihres Steuerungsrechners) ab, und kann daher nicht pauschal angegeben werden. Typischerweise liegt sie im Bereich von 10-60 Sekunden.

3 Zugriffsvarianten auf MDP Elemente

Die TwinCAT 3 SPS Tc2_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 **generischen 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 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). Jeder Funktionsbaustein muss solange 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 (Device Manager Dokumentation). Dort sind ebenso weitere Zugriffsmöglichkeiten auf das MDP beschrieben.

Generische Funktionsbausteine

Um einen IPC-Diagnose Parameter abfragen oder setzen zu können, muss die dynamische Modul ID des Moduls bekannt sein in dem sich der Parameter befindet.
Diese wird mithilfe des Funktionsbausteines FB_MDP_ScanModules [▶ 19] ermittelt.

Nun können einzelne Parameter mittels FB_MDP_Read [▶ 11] und FB_MDP_Write [▶ 12] 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 [▶ 30]) mit dem Funktionsbaustein FB_MDP_ReadModuleHeader [▶ 18] abgefragt werden.
Der komplette Inhalt einer ausgewählten Tabelle innerhalb eines Moduls kann mit dem Funktionsbaustein FB_MDP_ReadModuleContent [▶ 17] abgefragt werden.

Der Funktionsbaustein FB_MDP_ReadModule [▶ 15] bündelt obige Abfragen. Der Funktionsbaustein ermittelt implizit die dynamische Modul ID und fragt Header sowie Tabelle ab.
Der Funktionsbaustein FB_MDP_ReadElement [▶ 14] ermittelt ebenfalls die dynamische Modul ID bereits implizit. Mit ihm kann ein beliebiger einzelner IPC-Diagnose Parameter 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 IPC-Diagnose Informationen.

Beispielsweise reicht der Aufruf des Funktionsbausteines FB_MDP_NIC_Read [▶ 23] aus, um alle wichtigen Informationen über einen Netzwerkadapter abzufragen (siehe Device Manager Dokumentation Modul NIC).
Der Modul Header wird jeweils auch abgefragt und ausgegeben.
Ebenso ermitteln die spezifischen Funktionsbausteine implizit die dynamische Modul ID, sodass ein vorheriger Aufruf von FB_MDP_ScanModules [▶ 19] überflüssig ist.

4 Funktionsbausteine

4.1 Einführung

Update: Tc3_IPCDiag Bibliothek

Die TwinCAT 3 SPS Bibliothek Tc2_MDP ist der Vorgänger zur Tc3_IPCDiag. Mit der neuen Tc3_IPCDiag Bibliothek wurde zum einen die Menge lesbarer Parameter vergrößert und zum anderen die Anwenderschnittstelle optimiert. Es wird empfohlen die Tc3_IPCDiag Bibliothek zu verwenden.
Zukünftige Erweiterungen werden nicht mehr in der Tc2_MDP Bibliothek durchgeführt. Eine Verwendung der Tc2_MDP Bibliothek für neue Projekte wird nicht empfohlen. Alle Funktionalitäten der Tc2_MDP Bibliothek sind ebenfalls in der neuen Tc3_IPCDiag Bibliothek zu finden.

Generische Funktionsbausteine

Mit einem generischen Funktionsbaustein kann auf beliebige Module der IPC-Diagnose zugegriffen werden.

Die Anwendung ist komplexer und der Anwender benötigt ein gewisses Verständnis des MDP-(**M**odule **D**evice **P**rofile)-Informationsmodells.

Beispiel: Zugriff auf Netzwerkkarteninformation über FB_MDP_ReadElement:

Der generische Funktionsbaustein FB_MDP_ReadElement ermittelt die dynamische Modul-Adresse intern. Der Anwender muss nur angeben, auf welche Modulinstanz er zugreifen möchte. Das System hat mehrere Netzwerkkarten, und jede Netzwerkkarte ist durch eine eigene Modulinstanz repräsentiert.

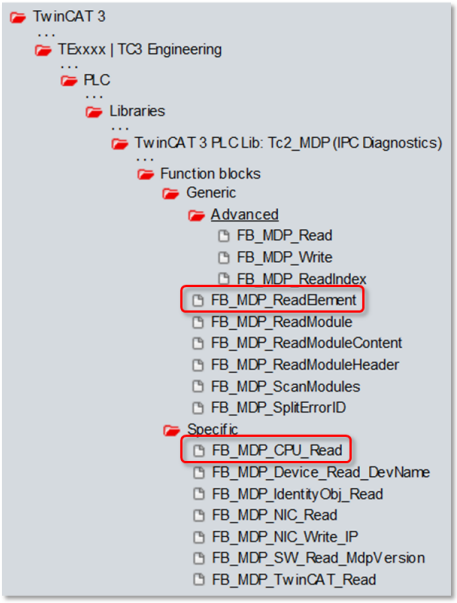
Spezifische Funktionsbausteine

Ein spezifischer Funktionsbaustein greift nur auf Informationen eines spezifischen Moduls zu. Er ist einfach zu verwenden und erfordert keine Kenntnis des verwendeten MDP-Informationsmodells.

Die verfügbaren spezifischen Funktionsbausteine ermitteln die dynamische Modul-Adresse intern.

Allerdings stehen spezifische Funktionsbausteine nur für eine Auswahl der Daten der IPC-Diagnose zur Verfügung.

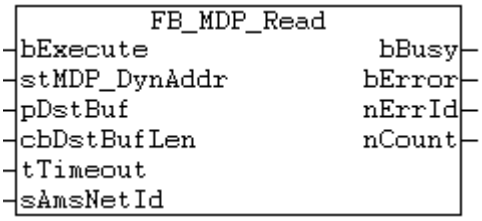
Beispiel: FB_MDP_CPU_Read



4.2 Generisch

4.2.1 Advanced

4.2.1.1 FB_MDP_Read



Der Funktionsbaustein ermöglicht das Abfragen eines Elementes eines IPC-Diagnose Moduls.

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;             (* Function block execution is triggered by a rising edge at t
his input.*)
    stMDP_DynAddr : ST_MDP_Addr;
    pDstBuf       : DWORD;            (* 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 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.

stMDP_DynAddr: An diesem Eingang wird die MDP Adressierung angegeben, welche zu dem gewählten Netzwerkmodul gehört. Die Struktur ist vom Typ ST_MDP_Addr [▶ 29]. 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 anderen Computer zu richten kann hier dessen AMS Net Id (vom Typ T_AmsNetId) 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.

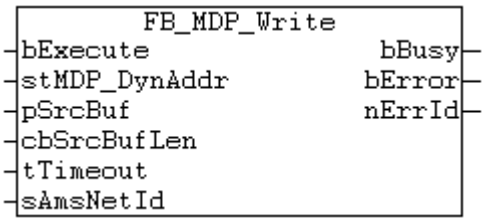
nErrID: Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 34].

nCount: Dieser Ausgang gibt die Anzahl der gelesenen Bytes an.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

4.2.1.2 FB_MDP_Write



Der Funktionsbaustein ermöglicht das Setzen eines Elementes eines IPC-Diagnose Moduls.

VAR_INPUT

```
VAR_INPUT
    bExecute       : BOOL;              (* Function block execution is triggered by a rising edge a
t this input.*)
    stMDP_DynAddr  : ST_MDP_Addr;
    pSrcBuf        : DWORD;             (* Contains the address of the buffer for the sent data. *)
    cbSrcBufLen    : UDINT;             (* Contains the max. number of bytes to be sent. *)
    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.

stMDP_DynAddr: An diesem Eingang wird die MDP Adressierung angegeben, welche zu dem gewählten Netzwerkmodul gehört. Die Struktur ist vom Typ ST_MDP_Addr [▶ 29]. 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 anderen Computer zu richten kann hier dessen AMS Net Id (vom Typ T_AmsNetId) 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 [▶ 34].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

4.2.1.3 FB_MDP_ReadIndex

Update: Tc3_IPCDiag Bibliothek

Die TwinCAT 3 SPS Bibliothek Tc2_MDP ist der Vorgänger zur Tc3_IPCDiag. Mit der neuen Tc3_IPCDiag Bibliothek wurde zum einen die Menge lesbarer Parameter vergrößert und zum anderen die Anwenderschnittstelle optimiert. Es wird empfohlen die Tc3_IPCDiag Bibliothek zu verwenden.
Zukünftige Erweiterungen werden nicht mehr in der Tc2_MDP Bibliothek durchgeführt. Eine Verwendung der Tc2_MDP Bibliothek für neue Projekte wird nicht empfohlen. Alle Funktionalitäten der Tc2_MDP Bibliothek sind ebenfalls in der neuen Tc3_IPCDiag Bibliothek zu finden.

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     : DWORD;            (* 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 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 (vom Typ T_AmsNetId) 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 [▶ 34].

nCount: Dieser Ausgang gibt die Anzahl der gelesenen Bytes an.

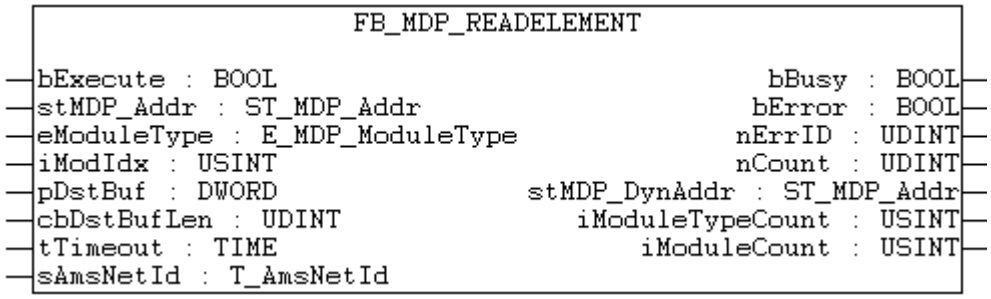
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4013	PC oder CX (x86, x64, ARM)	Tc2_MDP

4.2.2 FB_MDP_ReadElement

Update: Tc3_IPCDiag Bibliothek

Die TwinCAT 3 SPS Bibliothek Tc2_MDP ist der Vorgänger zur Tc3_IPCDiag. Mit der neuen Tc3_IPCDiag Bibliothek wurde zum einen die Menge lesbarer Parameter vergrößert und zum anderen die Anwenderschnittstelle optimiert. Es wird empfohlen die Tc3_IPCDiag Bibliothek zu verwenden.
Zukünftige Erweiterungen werden nicht mehr in der Tc2_MDP Bibliothek durchgeführt. Eine Verwendung der Tc2_MDP Bibliothek für neue Projekte wird nicht empfohlen. Alle Funktionalitäten der Tc2_MDP Bibliothek sind ebenfalls in der neuen Tc3_IPCDiag Bibliothek zu finden.



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     : DWORD;              (* 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 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.

stMDP_Addr: An diesem Eingang wird die MDP Adressierung angegeben, welche zu dem gewählten Modul gehört. Die Struktur ist vom Typ ST_MDP_Addr [▶ 29].
Die Area muss als Configuration Area angegeben werden.
Die dynamische Modul ID wird erst intern hinzugefügt und darf nicht angegeben werden.

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 (vom Typ T_AmsNetId) 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 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 [▶ 34].

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 [▶ 29]. Die dynamische Modul ID wurde durch den Funktionsbaustein hinzugefügt.

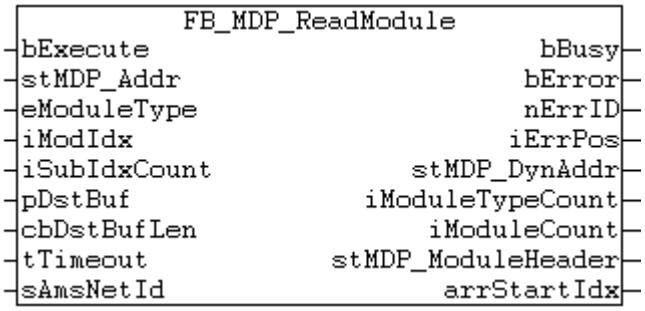
iModuleTypeCount: Der Ausgang *iModuleTypeCount* gibt die Anzahl der Module an, welche dem angegebenen Typ entsprechen.

iModuleCount: Der Ausgang *iModuleCount* gibt die komplette Anzahl der Module auf dem Gerät an.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

4.2.3 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. secon
d NIC(idx 1) of three found NICs. *)
    iSubIdxCount : USINT;
    pDstBuf      : DWORD;              (* 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 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.

stMDP_Addr: An diesem Eingang wird die MDP Adressierung angegeben, welche zu dem gewählten Modul gehört. Die Struktur ist vom Typ ST_MDP_Addr [▶ 29]. 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 [▶ 29] aufgelistet. (allgemeine Informationen: 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 (vom Typ T_AmsNetId) 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 mo
dule 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 [▶ 34].

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 [▶ 29]. 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.

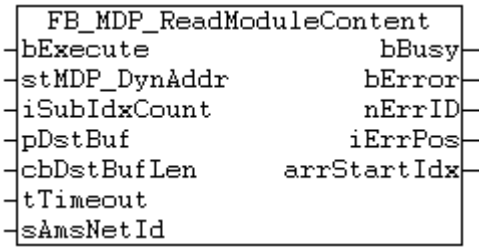
stMDP_ModuleHeader: An diesem Ausgang werden die Header Informationen des gelesenen MDP Moduls in Form der Struktur ST_MDP_ModuleHeader [▶ 30] angezeigt.

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

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

4.2.4 FB_MDP_ReadModuleContent



Der Funktionsbaustein ermöglicht das Abfragen des Inhaltes eines IPC-Diagnose Moduls.

VAR_INPUT

```
VAR_INPUT
    bExecute      : BOOL;
    stMDP_DynAddr : ST_MDP_Addr;    (* includes the dynamic module type for which the module content
 is requested. All subindexes of the chosen table are requested. *)
    iSubIdxCount  : USINT;          (* the number of SubIndexes to be requested *)
    pDstBuf       : DWORD;          (* 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 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.

stMDP_DynAddr: An diesem Eingang wird die MDP Adressierung angegeben, welche zu dem gewählten Modul gehört. Die Struktur ist vom Typ ST_MDP_Addr [▶ 29]. 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 anderen Computer zu richten kann hier dessen AMS Net Id (vom Typ T_AmsNetId) 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 [▶ 34].

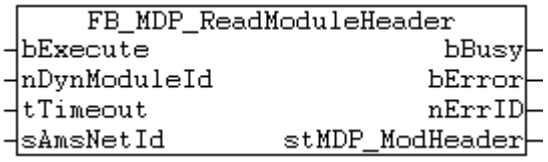
iErrPos: Falls ein Fehler auftrat und sich dieser auf ein einzelnes Element bezieht, gibt dieser Ausgang die Position (Subindex des Elementes) an, an welcher zuerst ein Fehler auftrat.

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 v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

4.2.5 FB_MDP_ReadModuleHeader



Der Funktionsbaustein ermöglicht das Abfragen des Headers eines IPC-Diagnose Moduls.

VAR_INPUT

```
VAR_INPUT
    bExecute     : BOOL;
    nDynModuleId : BYTE;             (* the dynamic module id for which the module header is request
ed *)
    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.

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 (vom Typ T_AmsNetId) 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 [▶ 34].

stMDP_ModuleHeader: An diesem Ausgang werden die Header Informationen des gelesenen IPC-Diagnose Moduls in Form der Struktur ST_MDP_ModuleHeader [▶ 30] angezeigt.

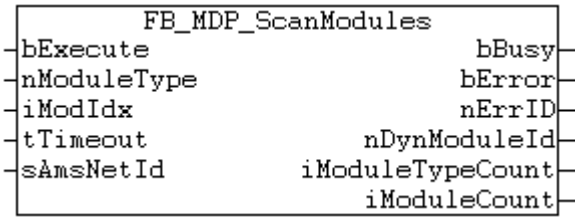
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

4.2.6 FB_MDP_ScanModules

Update: Tc3_IPCDiag Bibliothek

Die TwinCAT 3 SPS Bibliothek Tc2_MDP ist der Vorgänger zur Tc3_IPCDiag. Mit der neuen Tc3_IPCDiag Bibliothek wurde zum einen die Menge lesbarer Parameter vergrößert und zum anderen die Anwenderschnittstelle optimiert. Es wird empfohlen die Tc3_IPCDiag Bibliothek zu verwenden.
Zukünftige Erweiterungen werden nicht mehr in der Tc2_MDP Bibliothek durchgeführt. Eine Verwendung der Tc2_MDP Bibliothek für neue Projekte wird nicht empfohlen. Alle Funktionalitäten der Tc2_MDP Bibliothek sind ebenfalls in der neuen Tc3_IPCDiag Bibliothek zu finden.



Der Funktionsbaustein ermöglicht das Durchsuchen eines Gerätes nach einem bestimmten IPC-Diagnose 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 [▶ 29] 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 N
IC(idx 1) of three found NICs. *)
    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.

nModuleType: An diesem Eingang wird der IPC-Diagnose Modul Typ angegeben. Die möglichen Typen sind in der Enumeration E_MDP_ModuleType [▶ 29] aufgelistet. (allgemeine Informationen zu IPC-Diagnose Modultypen)

iModIdx: Falls ein IPC-Diagnose 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 [▶ 18] 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 anderen Computer zu richten kann hier dessen AMS Net Id (vom Typ T_AmsNetId) 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 module 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 [▶ 34].

nDynModuleId: Dieser Ausgang gibt die ermittelte dynamische Module ID für das gewählte Modul an.

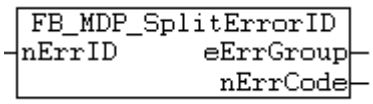
iModuleTypeCount: Der Ausgang *iModuleTypeCount* gibt die Anzahl der Module an, welche dem angegebenen Typ entsprechen.

iModuleCount: Der Ausgang *iModuleCount* gibt die komplette Anzahl der Module auf dem Gerät an.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

4.2.7 FB_MDP_SplitErrorID



Der Funktionsbaustein ermöglicht das Aufsplitten der *nErrID* zu einer Fehlergruppe [▶ 34] 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 [▶ 34]. Mit Hilfe der Fehlergruppe kann differenziert werden, um welche Art von Fehler bzw. um welche Fehlerquelle es sich handelt.

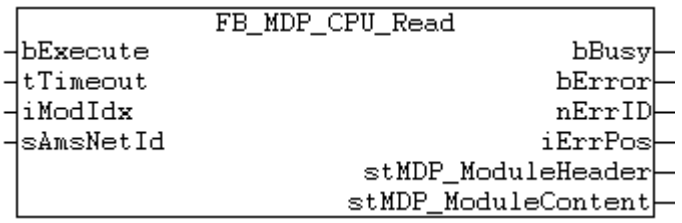
nErrCode: Der Fehlercode ist spezifisch für jede Fehlergruppe.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

4.3 Spezifisch

4.3.1 FB_MDP_CPU_Read



Der Funktionsbaustein ermöglicht die Abfrage des IPC-Diagnose Moduls CPU.

VAR_INPUT

```
VAR_INPUT
    bExecute  : BOOL;         (* Function block execution is triggered by a rising edge at this inpu
t.*)
    tTimeout  : TIME := DEFAULT_ADS_TIMEOUT; (* States the time before the function is cancelled. *)
    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 IPC-Diagnose 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 anderen Computer zu richten kann hier dessen AMS Net Id (vom Typ T_AmsNetId) 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 [▶ 34].

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.

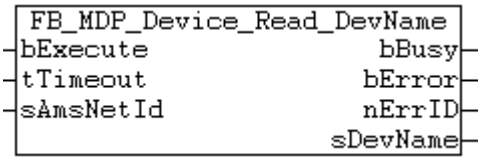
stMDP_ModuleHeader: An diesem Ausgang werden die Header Informationen des gelesenen IPC-Diagnose Moduls in Form der Struktur ST_MDP_ModuleHeader [▶ 30] angezeigt.

stMDP_ModuleContent: An diesem Ausgang werden die Informationen der TableID 1 des gelesenen IPC-Diagnose Moduls in Form der Struktur ST_MDP_CPU [▶ 30] angezeigt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

4.3.2 FB_MDP_Device_Read_DevName



Der Funktionsbaustein ermöglicht die Abfrage des Gerätenamens. Diese Information befindet sich in der IPC-Diagnose: General Area.

VAR_INPUT

```
VAR_INPUT
    bExecute  : BOOL;           (* Function block execution is triggered by a rising edge at this in
put.*)
    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.

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 (vom Typ T_AmsNetId) 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.

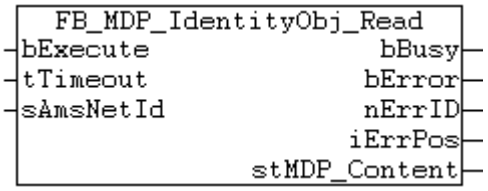
nErrID: Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 34].

sDevName: An diesem Ausgang wird der abgefragte Name als String ausgegeben.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

4.3.3 FB_MDP_IdentityObj_Read



Der Funktionsbaustein ermöglicht die Abfrage der Tabelle IdentityObject der General Area der IPC-Diagnose.

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 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.

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 (vom Typ T_AmsNetId) 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 [▶ 34].

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 [▶ 31] angezeigt.

Seriennummer wird nicht mehr unterstützt

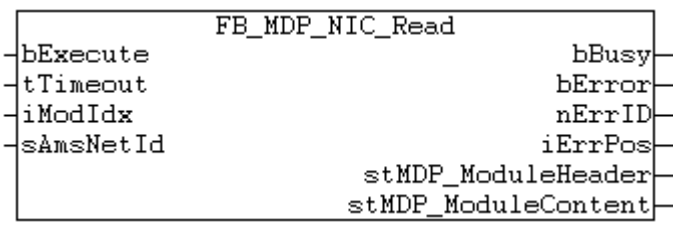
Veralteter Parameter führt zu Fehlersituation

Im Identity Object wird die Seriennummer des IPCs aus der MDP General Area ausgelesen. Dieser Parameter ist veraltet. Bei neueren Beckhoff IPC Geräten wird der Parameter nicht mehr unterstützt. Dies führt dazu, dass der Funktionsbaustein einen Fehler liefert und die Fehlerposition (`iErrPos = 4`) benennt, welche dem Parameter `iSerialNumber` entspricht.
Alternativ kann die Seriennummer aus der MDP Device Area ausgelesen werden. Siehe entsprechendes Beispiel [▶ 52].
Es wird die Verwendung der SPS Bibliothek Tc3_IPCDiag empfohlen, welche der Nachfolger zur SPS Bibliothek Tc2_MDP ist.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

4.3.4 FB_MDP_NIC_Read



Der Funktionsbaustein ermöglicht die Abfrage des IPC-Diagnose Moduls NIC (Network Interface Card).

VAR_INPUT

```
VAR_INPUT
    bExecute  : BOOL;           (* Function block execution is triggered by a rising edge at this in
put.*)
    tTimeout  : TIME := DEFAULT_ADS_TIMEOUT; (* States the time before the function is cancelled. *)
    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 IPC-Diagnose 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 anderen Computer zu richten kann hier dessen AMS Net Id (vom Typ T_AmsNetId) 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 [▶ 34].

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.

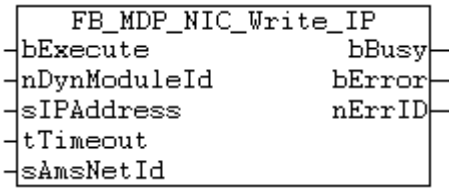
stMDP_ModuleHeader: An diesem Ausgang werden die Header Informationen des gelesenen IPC-Diagnose Moduls in Form der Struktur ST_MDP_ModuleHeader [▶ 30] angezeigt.

stMDP_ModuleContent: An diesem Ausgang werden die Informationen der TableID 1 des gelesenen IPC-Diagnose Moduls in Form der Struktur ST_MDP_NIC [▶ 31] angezeigt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

4.3.5 FB_MDP_NIC_Write_IP



Der Funktionsbaustein ermöglicht das Setzen einer neuen IP Adresse. Dieses Element ist Teil des IPC-Diagnose Moduls 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 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.

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 anderen Computer zu richten kann hier dessen AMS Net Id (vom Typ T_AmsNetId) angegeben werden.

VAR_OUTPUT

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

bBusy: Dieser Ausgang ist TRUE solange der Funktionsbaustein aktiv ist.

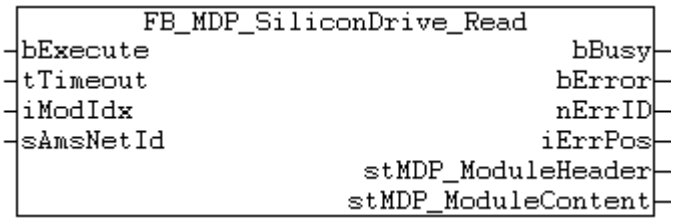
bError: Wird TRUE, sobald eine Fehlersituation eintritt.

nErrID: Liefert bei einem gesetzten bError-Ausgang einen Fehlercode [▶ 34].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

4.3.6 FB_MDP_SiliconDrive_Read



Der Funktionsbaustein ermöglicht die Abfrage des IPC-Diagnose Moduls SiliconDrive.

Veraltete Funktionalität

Die SiliconDrive Hardware ist durch neuere Speicherkarten-Typen ersetzt worden und die Funktionalität ist somit veraltet. Es wird stattdessen empfohlen die Abfrage des IPC-Diagnose Moduls Physical Drive SMART Parameters einzusetzen.

VAR_INPUT

```
VAR_INPUT
    bExecute  : BOOL;         (* Function block execution is triggered by a rising edge at this inpu
t.*)
    tTimeout  : TIME := DEFAULT_ADS_TIMEOUT; (* States the time before the function is cancelled. *)
```

```
    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 IPC-Diagnose 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 anderen Computer zu richten kann hier dessen AMS Net Id (vom Typ T_AmsNetId) 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 [▶ 34].

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 IPC-Diagnose Moduls in Form der Struktur ST_MDP_ModuleHeader [▶ 30]angezeigt.

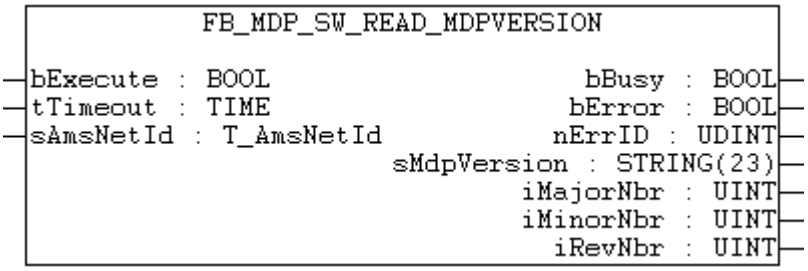
stMDP_ModuleContent: An diesem Ausgang werden die Informationen der TableID 1 des gelesenen IPC-Diagnose Moduls in Form der Struktur ST_MDP_SiliconDrive [▶ 31] angezeigt.

> Die Abfrage des IPC-Diagnose Moduls Silicon Drive gehört zu den zeitintensiveren Bearbeitungen. So kann möglicherweise 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 v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

4.3.7 FB_MDP_SW_Read_MdpVersion



Der Funktionsbaustein ermöglicht die Abfrage der MDP Version. Diese Information befindet sich im Modul Software in der Configuration Area der IPC-Diagnose.

Die MDP Version ist unabhängig von der Version der SPS Bibliothek. Um die Version der SPS Bibliothek abzufragen wird die Konstante stLibVersion_Tc2_MDP [▶ 33] verwendet.

VAR_INPUT

```
VAR_INPUT
    bExecute  : BOOL;           (* Function block execution is triggered by a rising edge at this in
put.*)
    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.

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 (vom Typ T_AmsNetId) 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 [▶ 34].

sMdpVersion: An diesem Ausgang wird die abgefragte MDP Version als String ausgegeben.

iMajorNbr: Die erste Position der Versionsnummer wird mit *iMajorNbr* als Zahl ausgegeben.

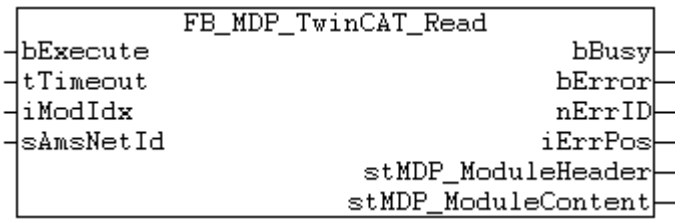
iMinorNbr: Die zweite Position der Versionsnummer wird mit *iMinorNbr* als Zahl ausgegeben.

iRevNbr: Die dritte Position der Versionsnummer wird mit *iRevNbr* als Zahl ausgegeben.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

4.3.8 FB_MDP_TwinCAT_Read



Der Funktionsbaustein ermöglicht die Abfrage des IPC-Diagnose Moduls TwinCAT.

VAR_INPUT

```
VAR_INPUT
    bExecute  : BOOL;         (* Function block execution is triggered by a rising edge at this inpu
t.*)
```

```
    tTimeout  : TIME := DEFAULT_ADS_TIMEOUT; (* States the time before the function is cancelled. *)
    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 IPC-Diagnose 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 anderen Computer zu richten kann hier dessen AMS Net Id (vom Typ T_AmsNetId) 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 [▶ 34].

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 IPC-Diagnose Moduls in Form der Struktur ST_MDP_ModuleHeader [▶ 30] angezeigt.

stMDP_ModuleContent: An diesem Ausgang werden die Informationen der TableID 1 des gelesenen IPC-Diagnose Moduls in Form der Struktur ST_MDP_TwinCAT [▶ 32] angezeigt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

5 Datentypen

5.1 Allgemeine Datentypen

5.1.1 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 der IPC-Diagnose.

5.1.2 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-Diagnose - Modultypen.

Dieser Modul Typ ist nicht gleichzusetzen mit der dynamischen Modul ID !

5.1.3 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 [▶ 29] 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 [▶ 19] 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 Dokumentation Device Manager

5.1.4 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 Modul Liste aufgezählt (Device Manager Dokumentation).
sName	Gibt den Namen dieses MDP Moduls an.
nDevType	Gibt den Typ des MDP Moduls als Code an.

5.2 Strukturen spezifischer MDP Module

5.2.1 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 IPC-Diagnose Modul CPU.

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

Die in dieser Struktur vorhandenen Parameter entsprechen den Subindizes der ersten Tabelle (Table ID 1) innerhalb des IPC-Diagnose Moduls CPU.

5.2.2 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 IPC-Diagnose General Area befindet.

Mittels des Funktionsbausteines FB_MDP_IdentityObj_Read [▶ 22] lassen sich diese kompletten Informationen abfragen.

Die in dieser Struktur vorhandenen Parameter entsprechen den Subindizes der Tabelle 'Identity Object' innerhalb der IPC-Diagnose General Area.

5.2.3 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 IPC-Diagnose Modul NIC (Network Interface Card).

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

Die in dieser Struktur vorhandenen Parameter entsprechen den Subindizes der ersten Tabelle (Table ID 1) innerhalb des IPC-Diagnose Moduls NIC.

5.2.4 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 [▶ 25] 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 IPC-Diagnose Moduls SiliconDrive.

Veraltete Funktionalität

Die SiliconDrive Hardware ist durch neuere Speicherkarten-Typen ersetzt worden und die Funktionalität ist somit veraltet. Es wird stattdessen empfohlen die Abfrage des IPC-Diagnose Moduls Physical Drive SMART Parameters einzusetzen.

5.2.5 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 [▶ 27] lassen sich diese kompletten Informationen abfragen.

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

6 Globale Konstanten

6.1 Global_Version

Alle Bibliotheken haben eine bestimmte Version. Diese Version ist u. a. im SPS-Bibliotheks-Repository zu sehen. Eine globale Konstante enthält die Information über die Bibliotheksversion:

Global_Version

```
VAR_GLOBAL CONSTANT
    stLibVersion_Tc2_MDP : ST_LibVersion;
END_VAR
```

stLibVersion_Tc2_MDP: Versionsnummer der Tc2_MDP-Bibliothek (Typ: ST_LibVersion).

Um zu sehen, ob die Version, die Sie haben auch die Version ist, die Sie brauchen, benutzen Sie die Funktion F_CmpLibVersion (definiert in der Tc2_System-Bibliothek).

Alle anderen Möglichkeiten Bibliotheksversionen zu vergleichen, die Sie von TwinCAT 2 kennen, sind veraltet!

7 Fehlercodes

7.1 Übersicht der Fehlercodes

Die Funktionsbausteine der Bibliothek Tc2_MDP besitzen einen Ausgang *nErrID*. Dieser Wert ist 4 Byte groß und liefert im Fehlerfall den Fehlercode. *nErrID* setzt sich aus zwei Teilen zusammen:

Error Group (MSB) 2 Byte	Error Code 2 Byte (LSB)
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

Der Funktionsbaustein FB_MDP_SplitErrorID [▶ 20] ermöglicht eine automatische Trennung der Variablen *nErrID* in Fehlergruppe und Fehlercode.

Error Group

Die Fehlergruppe beschreibt den Typ des aufgetretenen Fehlers. In der Enumeration E_MDP_ErrGroup [▶ 34] 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 [▶ 35] gelistet. Eine Beschreibung zu den restlichen Fehlercodes findet sich in der Dokumentation der IPC-Diagnose im Kapitel Fehlermeldungen.

> 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) 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.

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.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

7.2 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 [▶ 34] 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 [▶ 35] beschrieben.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

7.3 E_MDP_ErrCodesPLC

```
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 [▶ 34] 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.

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.

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 Pointern 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 MDP Information Model.(IPC Diagnose)

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.0	PC oder CX (x86, x64,ARM)	Tc2_MDP

8 Beispiele

Update: Tc3_IPCDiag Bibliothek

Die TwinCAT 3 SPS Bibliothek Tc2_MDP ist der Vorgänger zur Tc3_IPCDiag. Mit der neuen Tc3_IPCDiag Bibliothek wurde zum einen die Menge lesbarer Parameter vergrößert und zum anderen die Anwenderschnittstelle optimiert. Es wird empfohlen die Tc3_IPCDiag Bibliothek zu verwenden.
Zukünftige Erweiterungen werden nicht mehr in der Tc2_MDP Bibliothek durchgeführt. Eine Verwendung der Tc2_MDP Bibliothek für neue Projekte wird nicht empfohlen. Alle Funktionalitäten der Tc2_MDP Bibliothek sind ebenfalls in der neuen Tc3_IPCDiag Bibliothek zu finden.

Im Abschnitt Abfrage von CPU-Daten (generisch) [▶ 37] wird ein Beispielprogramm zum Lesen der CPU-Daten der IPC-Diagnose über den generischen Funktionsbaustein FB_MDP_ReadElement beschrieben. Es ist so aufgebaut, dass es mit den Grundkenntnissen des MDP-Informationsmodells einfach für den Zugriff auf andere Module der IPC-Diagnose erweitert werden kann. Das Beispiel zur Abfrage des Lüfterstatus (generisch) [▶ 50] baut auf dem Beispielprogramm zum Abfragen der CPU-Daten auf.

Im Abschnitt Abfrage von CPU-Daten (spezifisch) [▶ 46] wird ein Beispielprogramm zum Lesen der CPU-Daten der IPC-Diagnose über den spezifischen Funktionsbaustein FB_MDP_CPU_Read beschrieben. Das Beispiel kann nicht für den Zugriff auf andere Module der IPC-Diagnose, z. B. Lüfterdaten, erweitert werden.

Weitere Beispielprogramme sind in den Abschnitten IPC-Seriennummern lesen [▶ 52] und IP-Adresse setzen [▶ 54] beschrieben.

8.1 Abfrage von CPU-Daten (generisch)

Dieses Beispiel zeigt den Zugriff auf CPU-Daten der IPC-Diagnose über den generischen Funktionsbaustein FB_MDP_ReadElement [▶ 14].

Der Aufbau des Programms ist so gestaltet, dass das Programm leicht für den Zugriff auf andere Module der IPC-Diagnose angepasst werden kann. Programmzeilen, die für eine Anpassung des Programms auf andere Module der IPC-Diagnose geändert werden müssten, sind im Kommentar mit der Zeichenfolge //** markiert.

Nachfolgend auf den Programmcode des Beispiels finden Sie die textuelle Beschreibung des Beispielprogramms [▶ 38].

Beispiel zum Zugriff über den generischen Funktionsbaustein FB_MDP_ReadElement

Einzelne CPU-Daten sind über einen Subindex im Modul CPU in der Configuration Area der IPC-Diagnose auslesbar. Dazu wird der generische Funktionsbaustein FB_MDP_ReadElement verwendet.

Enumerationsdefinition

```
//** = simply adjust these lines if modifying code for own purposes

// central definition of state machine states
// (supports easy program modification)
{attribute 'qualified_only'}
TYPE E_State :
(
    Idle,
    ReadCPUusageInit,      //** initiate reading CPU usage
    ReadCPUusageProcess    //** process reading CPU usage
);
END_TYPE
```

Variablendeklaration

```
PROGRAM MAIN
VAR
    // internal use
    sAmsNetId           : STRING := '';       //** ADS Net ID (local = '')
    eState              : E_State;            // Enum with index for state machine
    bStart              : BOOL := TRUE;       // flag to trigger (re)start of statemachine
    nData               : UINT;               // data storage for unsigned integer
    stMDP_Addr          : ST_MDP_Addr;        // structure will include all address parameters
```

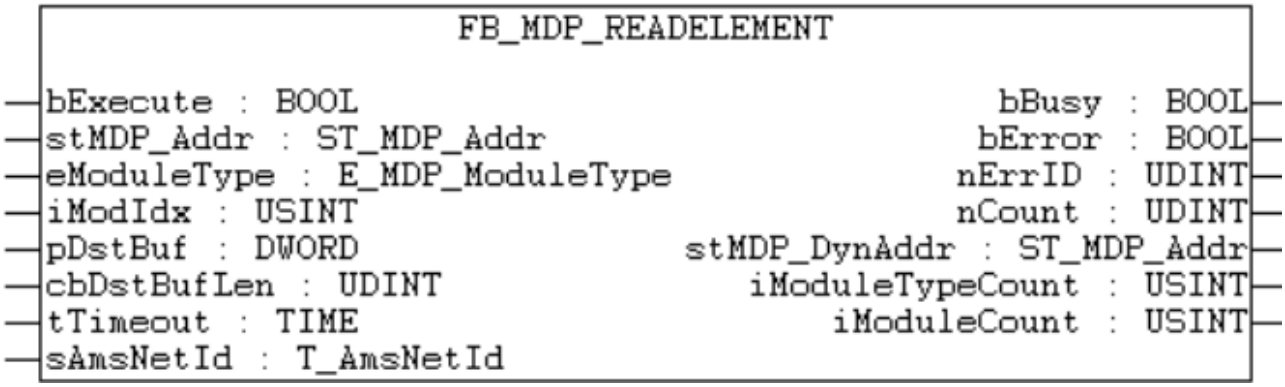
```
    // FB instances
    fbReadMDPElement    : FB_MDP_ReadElement; // instance of FB for reading MDP element

    // results of execution
    bError              : BOOL;               // error flag (indicator: error occured)
    nErrID              : UDINT;              // last error ID
    nCpuUsage           : UINT;               // buffer for CPU usage (%)
END_VAR
```

Programmcode

```
// For an easy re-use of the following code for own purposes, parts of this sample program use
// "general" data names (and copy the results in specific variables after processing the code).

CASE eState OF
    E_State.Idle:
        IF bStart THEN
            bStart := FALSE;
            eState := E_State.ReadCPUusageInit; //** initiate first state
        END_IF

    E_State.ReadCPUusageInit:
        stMDP_Addr.nArea    := INT_TO_BYTE(eMDP_Area_ConfigArea); //
** set area address to "Config Area"
        stMDP_Addr.nTableId := 1;             //** table ID in module for "cpu properties"
        stMDP_Addr.nSubIdx  := 2;             //
** subindex in table ID for "CPU usage"

        fbReadMDPElement(
            bExecute    := TRUE,              // Flag: trigger execution of FB
            eModuleType := eMDP_ModT_CPU,     //** desired module type = CPU
            stMDP_Addr  := stMDP_Addr,        // MDP address structure. Dynamic module ID added int
ernally.
            iModIdx     := 0,                 //
** instance of desired module type (0 = first instance)
            pDstBuf     := ADR(nData),        // buffer for storing data
            cbDstBufLen := SIZEOF(nData),     // length of buffer
            sAmsNetId   := sAmsNetId,         // AMS Net ID
            );                                //** Note: fbReadMDPElement.tTimeOut must be > cycle
time!

        eState := E_State.ReadCPUusageProcess;  //** next state: process FB

    E_State.ReadCPUusageProcess:               //** process FB: request CPU data
        fbReadMDPElement(bExecute := FALSE);   // Flag: Get execution state of FB
                                               //** Note: fbReadMDPElement.tTimeOut must be > cycle
time!

        IF NOT fbReadMDPElement.bBusy THEN     // FB executed?
            IF fbReadMDPElement.bError THEN    // Error?
                bError    := TRUE;             // set error flag
                nErrID    := fbReadMDPElement.nErrID; // store error id (16#ECA60105 = BIOS or HW do
es
                                               // not support this data (here: mainboard data))
                eState    := E_State.Idle;     // finish state machine
            ELSE                               // set parameters for next steps
                bError    := FALSE;            // turn off error flag
                nCpuUsage := nData;            //** store CPU usage in dedicated variable
                eState    := E_State.ReadCPUusageInit; //** next state
            END_IF
        END_IF

END_CASE
```

Beschreibung des Beispielprogramms

Der Funktionsbaustein FB_MDP_ReadElement erfordert minimal die im Folgenden aufgeführten Parameter:



VAR_INPUT

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. 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. (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.

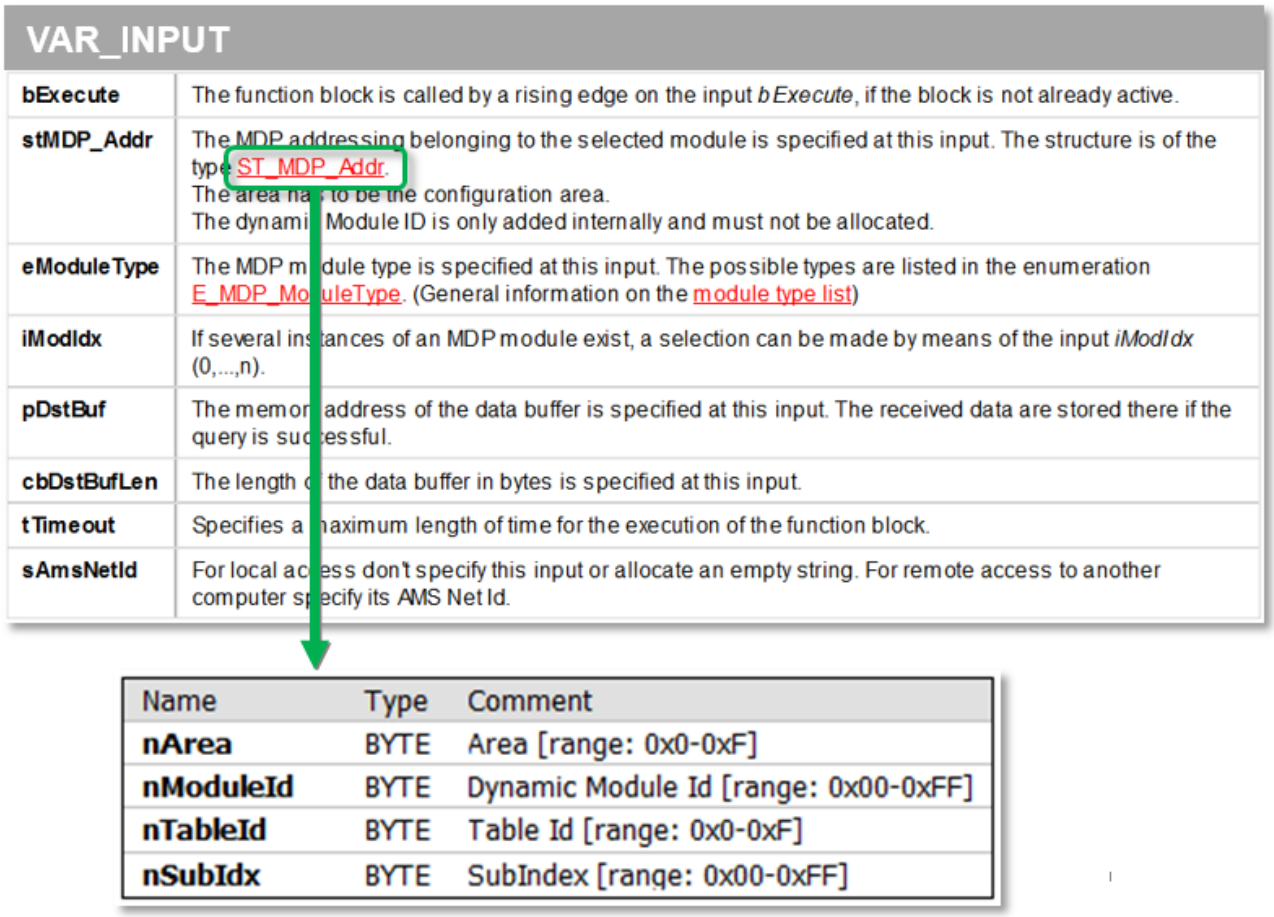
- Einen Speicherbereich für die MDP-Adressstruktur
- Den MDP-Modultypen
- Die Adresse und Größe des Speicherbereichs für die Ausgabedaten des Funktionsbausteins
- Die AMS-Adresse (AMS Net ID)

Zur Vollständigkeit sind hier noch die Ausgabewerte des Funktionsbausteins beschrieben:

VAR_OUTPUT

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 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. 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.

Eine sehr wichtige Rolle hat die Struktur zur Adressierung der gewünschten Information. Sie ist durch den Datentyp ST_MDP_Addr beschrieben:



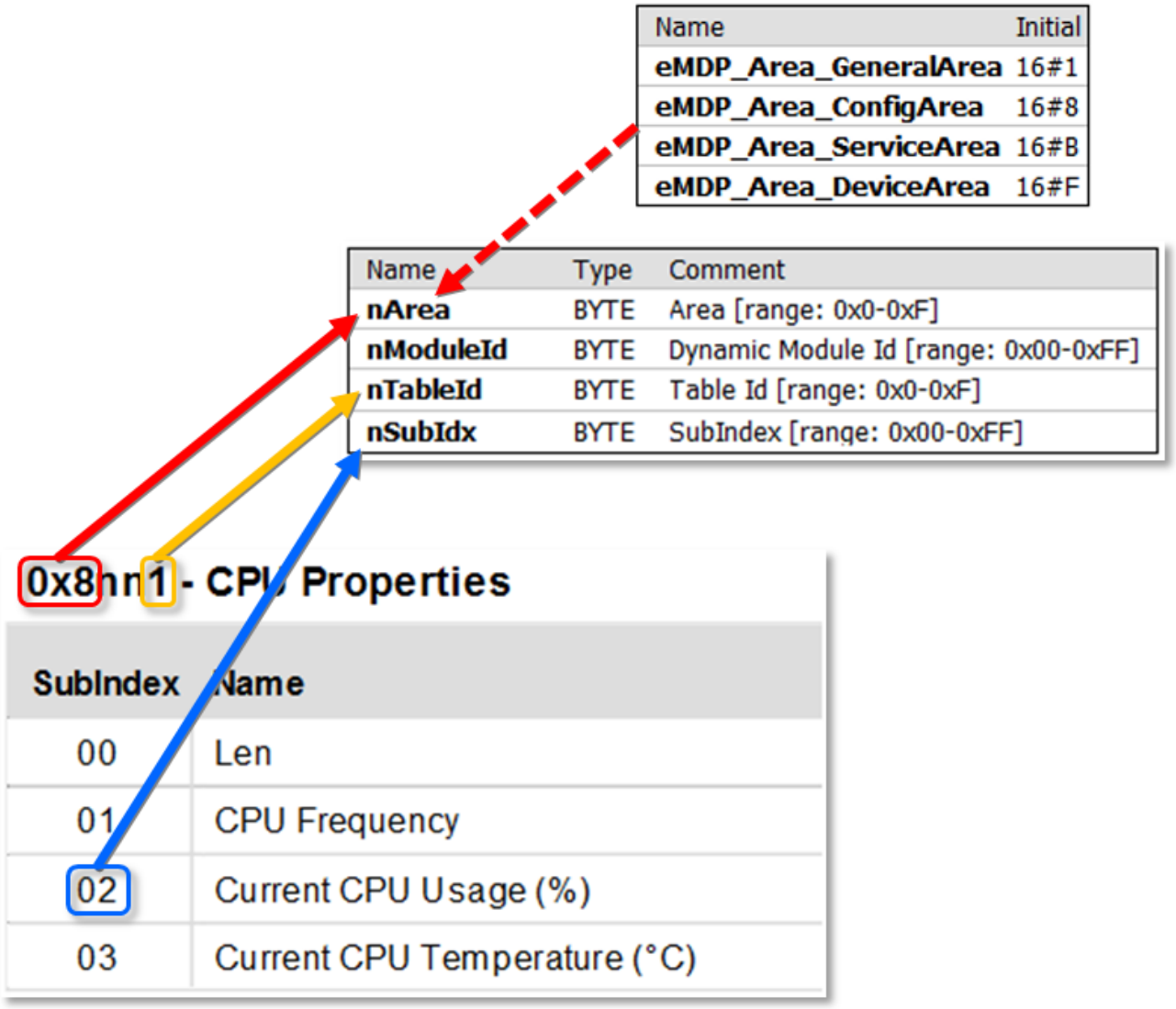
VAR_INPUT

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. 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. (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.

Name	Type	Comment
nArea	BYTE	Area [range: 0x0-0xF]
nModuleId	BYTE	Dynamic Module Id [range: 0x00-0xFF]
nTableId	BYTE	Table Id [range: 0x0-0xF]
nSubIdx	BYTE	SubIndex [range: 0x00-0xFF]

Für die Ermittlung des Parameters „nArea“ gibt es zwei Möglichkeiten:

- Die einzelnen Areas sind in der Tc2_MDP-Bibliothek als Enumeration hinterlegt. Dieser Eintrag kann als leicht lesbarer Eingangsparameter genutzt werden (gestrichelter roter Pfeil).
- Der Wert für die Area kann alternativ auch dem Index eines Tables entnommen werden (linke 4 Bit - roter Pfeil)

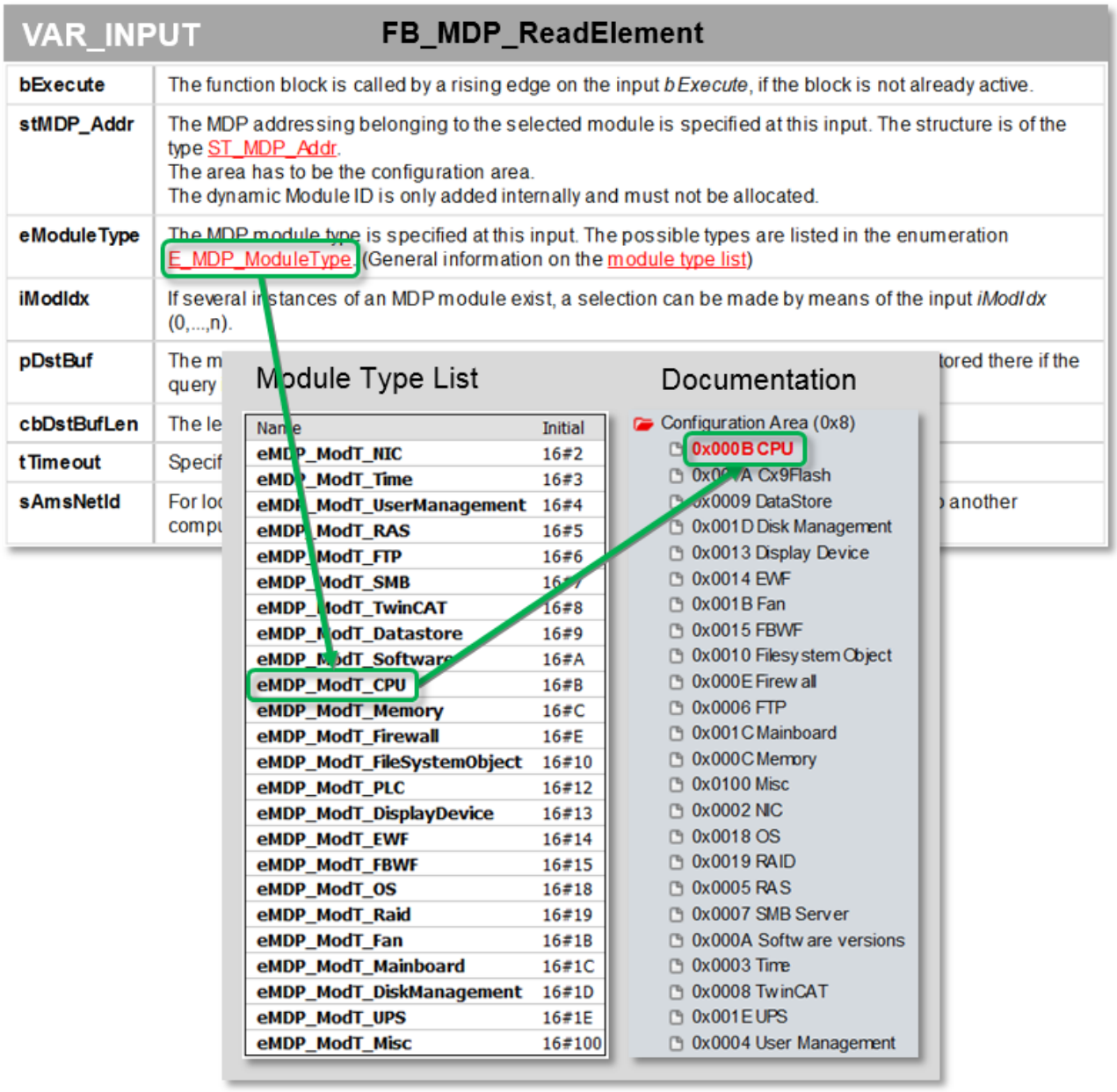


Der Parameter „nModule“ ist KEIN Eingabeparameter. Ihm wird die vom Funktionsbaustein ermittelte Moduladresse (=ModuleID) zugewiesen, nachdem dieser die gewünschte Modulinstanz ermittelt hat.

„nTableId“ entsprechen den rechten 4 Bit des Table-Index (gelber Pfeil).

„Subindex“ ist die Nummer des Eintrags in der Tabelle (blauer Pfeil).

Der Parameter „eModuleType“ gibt den Typ des Moduls an. Auch für den Modultyp gibt es eine Enumeration, die zwecks besserer Lesbarkeit des Programmes verwendet werden kann:



Alternativ kann der Wert auch der Modulbeschreibung entnommen und direkt eingetragen werden.

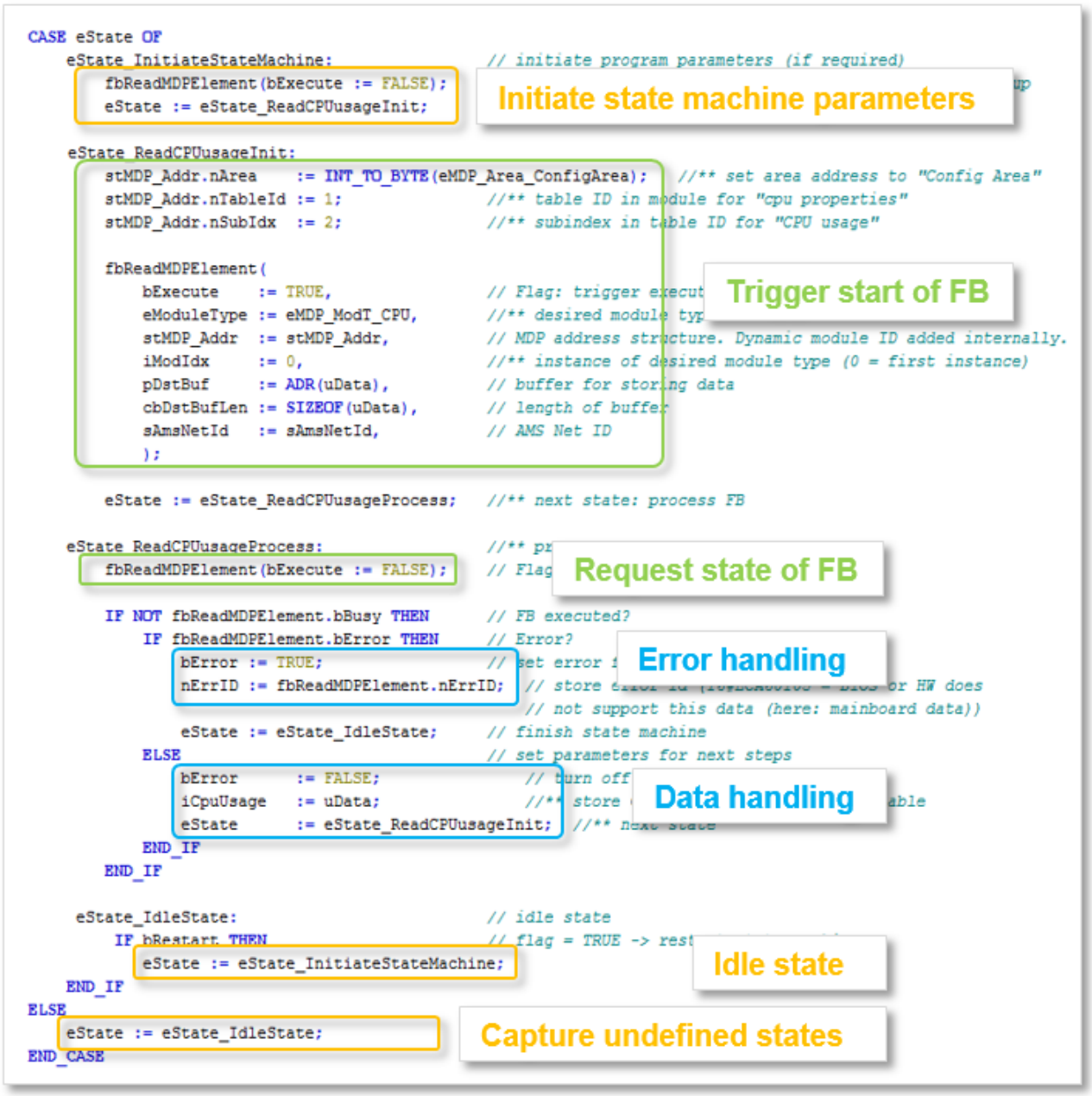
Aufbau der State Machine

Die Status der State Machine sind über Enumerationswerte in einem DUT definiert und können so zentral einfach angepasst bzw. erweitert werden.

```
CASE eState OF
    eState_InitiateStateMachine:                    // initiate program parameters (if required)
        fbReadMDPElement(bExecute := FALSE);        // ensure parameter 'bExecute' has FALSE state at startup
        eState := eState_ReadCPUusageInit;          //** initiate first state

    eState_ReadCPUusageInit:
        stMDP_Addr.nArea    := INT_TO_BYTE(eMDP_Area_ConfigArea);   //** set area address to "Config Area"
        stMDP_Addr.nTableId := 1;                   //** table ID in module for "cpu properties"
        stMDP_Addr.nSubIdx  := 2;                   //** subindex in table ID for "CPU usage"

        fbReadMDPElement(
            bExecute    := TRUE,
            eModuleType := eMDP_ModT_CPU,
            stMDP_Addr  := stMDP_Addr,
            iModIdx     := 0,
            pDstBuf     := ADR(uData),
            cbDstBufLen := SIZEOF(uData),
            sAmsNetId   := sAmsNetId,
            );

        eState := eState_ReadCPUusageProcess;       //** next state: process FB

    eState_ReadCPUusageProcess:                     //** process FB: request CPU data
        fbReadMDPElement(bExecute := FALSE);        // Flag: Get execution state of FB

        IF NOT fbReadMDPElement.bBusy THEN          // FB executed?
            IF fbReadMDPElement.bError THEN         // Error?
                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 := eState_IdleState;         // finish state machine
            ELSE                                    // set parameters for next steps
                bError      := FALSE;                   // turn off error flag
                iCpuUsage   := uData;                   //** store CPU usage in dedicated variable
                eState      := eState_ReadCPUusageInit; //** next state
            END_IF
        END_IF

    eState_IdleState:                               // idle state
        IF bRestart THEN                            // flag = TRUE -> restart state machine
            eState := eState_InitiateStateMachine;
    END_IF
ELSE
    eState := eState_IdleState;                     // capture undefined states
END_CASE
```

```
TYPE E_State :
(
    eState_InitiateStateMachine := 0,   // initiate state machine
    eState_ReadCPUDataInit      := 20,  //** initiate reading
    eState_ReadCPUDataProcess   := 21,  //** read
    eState_IdleState            := 100  // idle state
) UINT;
END_TYPE
```

Funktionsbereiche der State Machine




```
CASE eState OF
    eState_InitiateStateMachine:                // initiate program parameters (if required)
        fbReadMDPElement(bExecute := FALSE);
        eState := eState_ReadCPUusageInit;

    eState_ReadCPUusageInit:
        stMDP_Addr.nArea    := INT_TO_BYTE(eMDP_Area_ConfigArea);  //** set area address to "Config Area"
        stMDP_Addr.nTableId := 1;               //** table ID in module for "cpu properties"
        stMDP_Addr.nSubIdx  := 2;               //** subindex in table ID for "CPU usage"

        fbReadMDPElement(
            bExecute    := TRUE,                // Flag: trigger execut
            eModuleType := eMDP_ModT_CPU,       //** desired module typ
            stMDP_Addr  := stMDP_Addr,          // MDP address structure. Dynamic module ID added internally.
            iModIdx     := 0,                   //** instance of desired module type (0 = first instance)
            pDstBuf     := ADR(uData),          // buffer for storing data
            cbDstBufLen := SIZEOF(uData),       // length of buffer
            sAmsNetId   := sAmsNetId,           // AMS Net ID
            );

        eState := eState_ReadCPUusageProcess;   //** next state: process FB

    eState_ReadCPUusageProcess:                 //** pr
        fbReadMDPElement(bExecute := FALSE);    // Flag

        IF NOT fbReadMDPElement.bBusy THEN      // FB executed?
            IF fbReadMDPElement.bError THEN     // Error?
                bError := TRUE;                 //  set error f
                nErrID := fbReadMDPElement.nErrID;  // store error id (0x8C40xxxx - BIOS or HW does
                                                    // not support this data (here: mainboard data))
                eState := eState_IdleState;     // finish state machine
            ELSE                                // set parameters for next steps
                bError      := FALSE;           // turn off
                iCpuUsage   := uData;           //** store
                eState      := eState_ReadCPUusageInit;  //** next state
            END_IF
        END_IF

    eState_IdleState:                           // idle state
        IF bRestart THEN                        // flag = TRUE -> rest
            eState := eState_InitiateStateMachine;
    END_IF
ELSE
    eState := eState_IdleState;
END_CASE
```

Ein- und Ausgabeparameter des Beispielprogrammes

```
TYPE E_State :
(
    eState_InitiateStateMachine     := 0,   // initiate state machine (set parameters)
    eState_ReadCPUusageInit         := 20,  //** initiate reading serial number of mainboard
    eState_ReadCPUusageProcess      := 21,  //** process reading serial number of mainboard
    eState_IdleState                := 100  // idle state
) UINT;
END_TYPE
```

```
PROGRAM MAIN
VAR
    // internal use
    sAmsNetId           : STRING := '';     //** ADS Net ID (local = '')
    eState              : E_State;          // Enum with index for state machine
    bRestart            : BOOL;             // flag to trigger restart of statemachine
    uData               : UINT;             // data storage for unsigned integer
    stMDP_Addr          : ST_MDP_Addr;      // structure will include all address parameters

    // FB instances
    fbReadMDPElement    : FB_MDP_ReadElement;   // instance of FB for reading MDP element

    // results of execution
    bError              : BOOL;             // error flag (indicator: error occured)
    nErrID              : UDINT;            // last error ID
    iCpuUsage           : UINT;             // buffer for CPU usage (%)
END_VAR
```

```
CASE eState OF
    eState_InitiateStateMachine:                // initiate program parameters (if required)
        fbReadMDPElement(bExecute := FALSE);    // ensure parameter 'bEx
        eState := eState_ReadCPUusageInit;      //** initiate first st

eState_ReadCPUusageInit:
        stMDP_Addr.nArea    := INT_TO_BYTE(eMDP_Area_ConfigArea);   //**
        stMDP_Addr.nTableId := 1;               //** table ID in module
        stMDP_Addr.nSubIdx  := 2;               //** subindex in table ID for "CPU usage"

        fbReadMDPElement(
            bExecute    := TRUE,                // Flag: trigger execution of FB
            eModuleType := eMDP_ModT_CPU,       //** desired module type
            stMDP_Addr  := stMDP_Addr,          // MDP address structure
            iModIdx     := 0,                   //** instance of desired
            pDstBuf     := ADR(uData),          // buffer for storing dat
            cbDstBufLen := SIZEOF(uData),       // length of buffer
            sAmsNetId   := sAmsNetId,           // AMS Net ID
            );

        eState := eState_ReadCPUusageProcess;   //** next state: process

    eState_ReadCPUusageProcess:                 //** process FB: request CPU data
        fbReadMDPElement(bExecute := FALSE);    // Flag: Get execution state of FB

        IF NOT fbReadMDPElement.bBusy THEN      // FB executed?
            IF fbReadMDPElement.bError THEN     // Error?
                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 := eState_IdleState;     // finish state machine
            ELSE                                // set parameters for next steps
                bError      := FALSE;               // turn off error flag
                iCpuUsage   := uData;               //** store CPU usage in dedicated variable
                eState      := eState_ReadCPUusageInit;  //** next state
            END_IF
        END_IF

    eState_IdleState:                           // idle state
        IF bRestart THEN                        // flag = TRUE -> restart
            eState := eState_InitiateStateMachine;
    END_IF
ELSE
    eState := eState_IdleState;                 // capture undefined states
END_CASE
```

Set area

Set table ID

Set subindex

Structure with address data

Set AMS net address (`` = local)

Store result

8.2 Abfrage von CPU-Daten (spezifisch)

Dieses Beispiel zeigt den Zugriff auf CPU-Daten der IPC-Diagnose über den spezifischen Funktionsbaustein FB_MDP_CPU_Read [▶ 21].

Dieses Beispiel kann nicht für den Zugriff auf andere Module der IPC-Diagnose, z. B. Lüfterdaten, modifiziert werden.

Nachfolgend auf den Programmcode des Beispiels finden Sie die textuelle Beschreibung des Beispielprogramms [▶ 47].

Beispiel zum Zugriff über den spezifischen Funktionsbaustein FB_MDP_CPU_Read

Der spezifische Funktionsbaustein FB_MDP_CPU_Read ermöglicht einen einfachen Zugriff auf ausgewählte Daten des Moduls CPU in der Configuration Area der IPC-Diagnose.

Enumerationsdefinition

```
//** = simply adjust these lines if modifying code for own purposes

// central definition of state machine states
// (supports easy program modification)
{attribute 'qualified_only'}
TYPE E_State :
(
    Idle,
    ReadCPUDataInit,        //** initiate reading CPU data
    ReadCPUDataProcess      //** process reading CPU data
);
END_TYPE
```

Variablendeklaration

```
PROGRAM MAIN
VAR
    // internal use
    sAmsNetId           : STRING := '';       //** ADS Net ID (local = '')
    eState              : E_State;            // Enum with index for state machine
    bStart              : BOOL := TRUE;       // flag to trigger (re)start of statemachine

    // FB instances
    fbReadCPUData       : FB_MDP_CPU_Read;    // instance of FB for reading CPU data

    // results of execution
    bError              : BOOL;               // error flag (indicator: error occured)
    nErrID              : UDINT;              // last error ID
    stHeaderCpuMod      : ST_MDP_ModuleHeader; // buffer for header data of CPU module
    stCPUData           : ST_MDP_CPU;         // structure which will contain CPU data
END_VAR
```

Programmcode

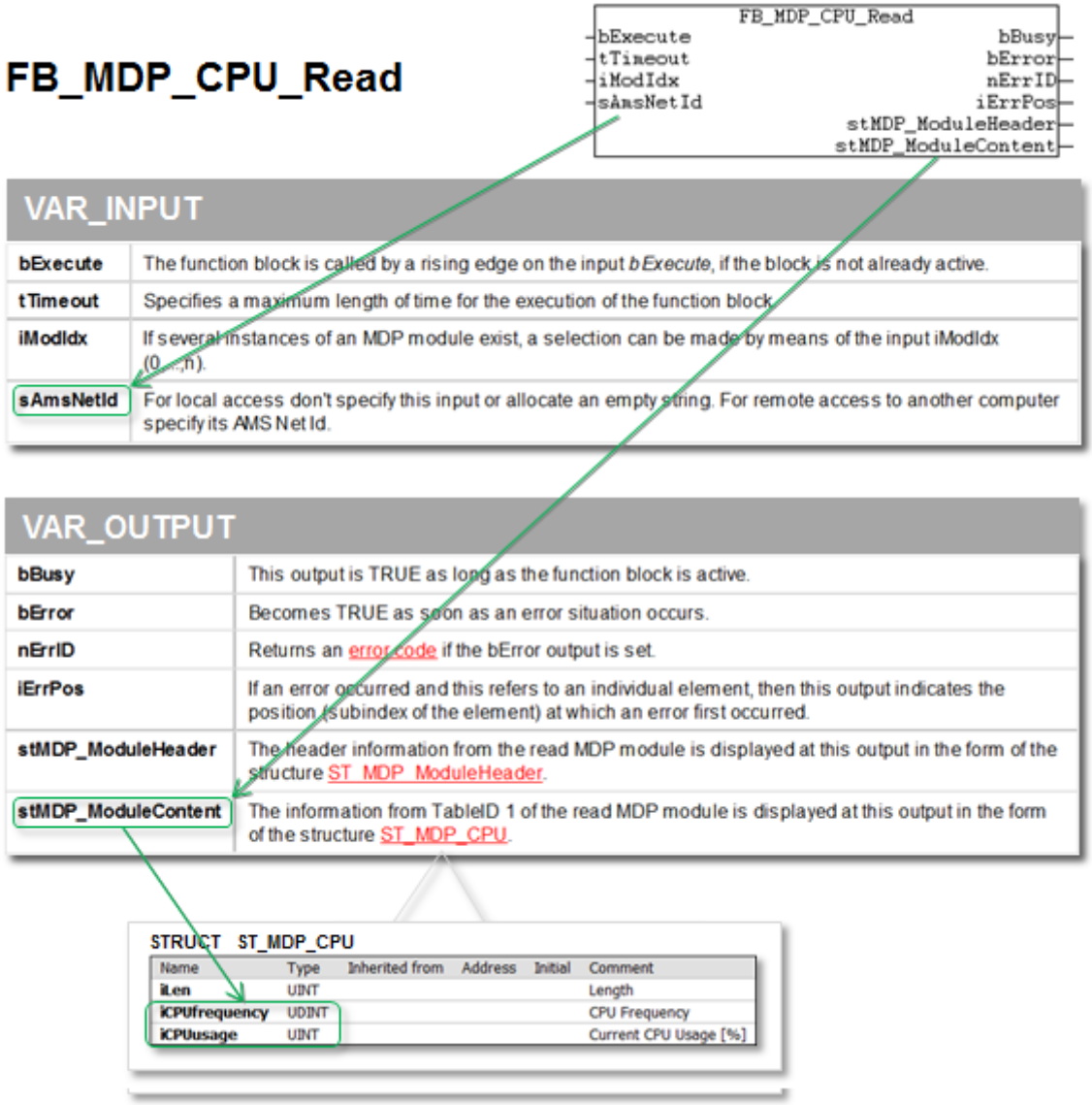
```
// For an easy reuse of the following code for own purposes, parts of this sample program use
// "general" data names (and copy the results in specific variables after processing the code).

CASE eState OF
    E_State.Idle:
        IF bStart THEN
            bStart := FALSE;
            eState := E_State.ReadCPUDataInit; //** initiate first state
        END_IF

    E_State.ReadCPUDataInit:                    //** trigger FB: request CPU data
        fbReadCPUData(
            bExecute    := TRUE,                // Flag: trigger execution of FB
            iModIdx     := 0,                   //
** Instance of desired module type (0 = first instance)
            sAmsNetId   := sAmsNetId);          // AMS Net ID

        eState := E_State.ReadCPUDataProcess;   //** next state: process FB

    E_State.ReadCPUDataProcess:                 //** process FB: request CPU data
        fbReadCPUData(bExecute := FALSE);       // Flag: Get execution state of FB

        IF NOT fbReadCPUData.bBusy THEN         // FB executed?
            IF fbReadCPUData.bError THEN        // Error?
```

```
                bError := TRUE;                 // set error flag
                nErrID := fbReadCPUData.nErrID; // store error id
                eState := E_State.Idle;          // finish state machine
            ELSE                                // set parameters for next steps
                bError         := FALSE;        // turn off error flag
                stHeaderCpuMod := fbReadCPUData.stMDP_ModuleHeader;  //
** store CPU module header data
                stCPUData      := fbReadCPUData.stMDP_ModuleContent; //** store CPU data
                eState         := E_State.ReadCPUDataInit;           //** read next set of CPU data
            END_IF
        END_IF

END_CASE
```

Beschreibung des Beispielprogramms

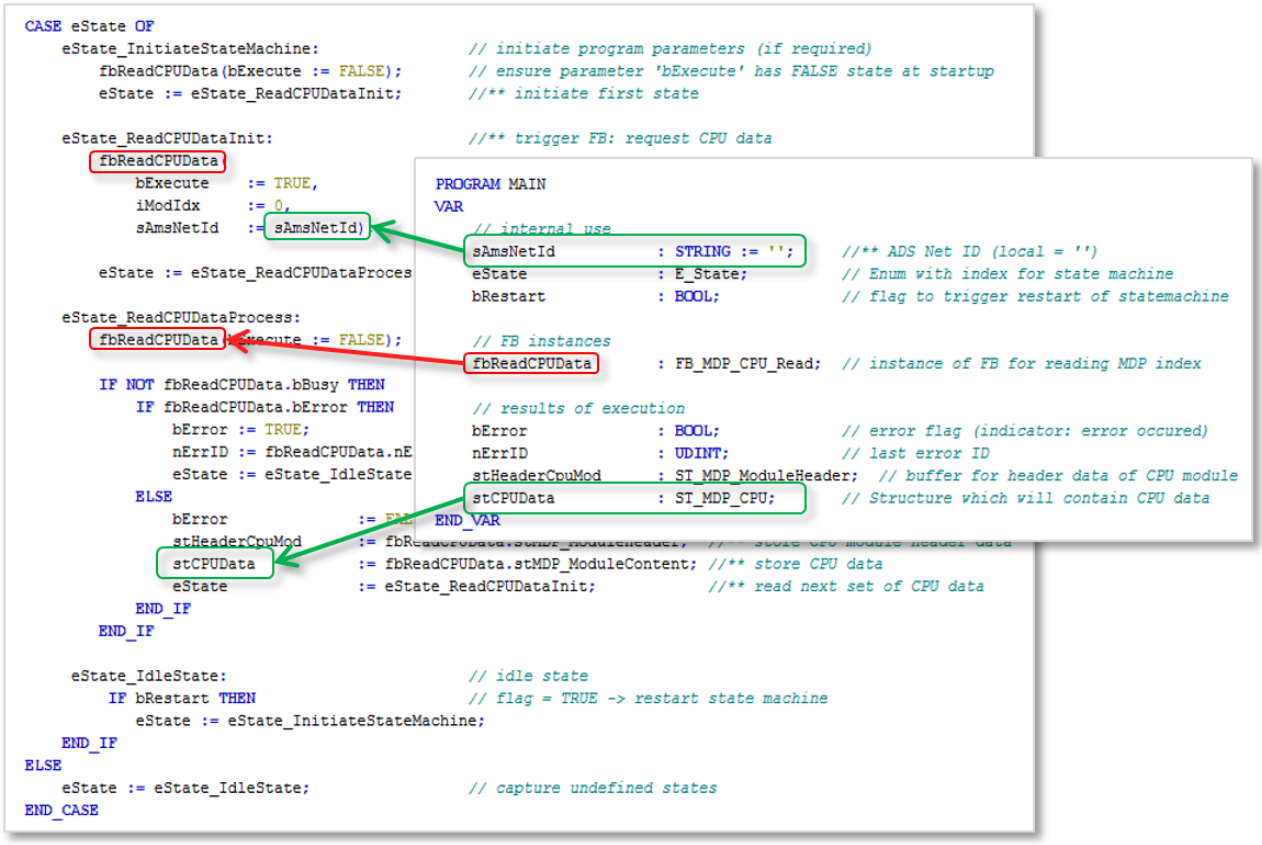
Der Funktionsbaustein FB_MDP_CPU_READ erfordert minimal zwei Parameter:

- Die AMS Net ID als Eingabeparameter für die Adresse des IPCs (lokal: ‘’)
- Eine Struktur stMDP_ModuleContent, die nach dem Aufruf des Funktionsbausteins die Daten enthält.



Der Baustein liefert nicht die CPU-Temperatur, diese kann nur über das generische Beispielprogramm ausgelesen werden. Der Wert „CPU-Temperatur“ wird nicht von allen IPCs unterstützt.

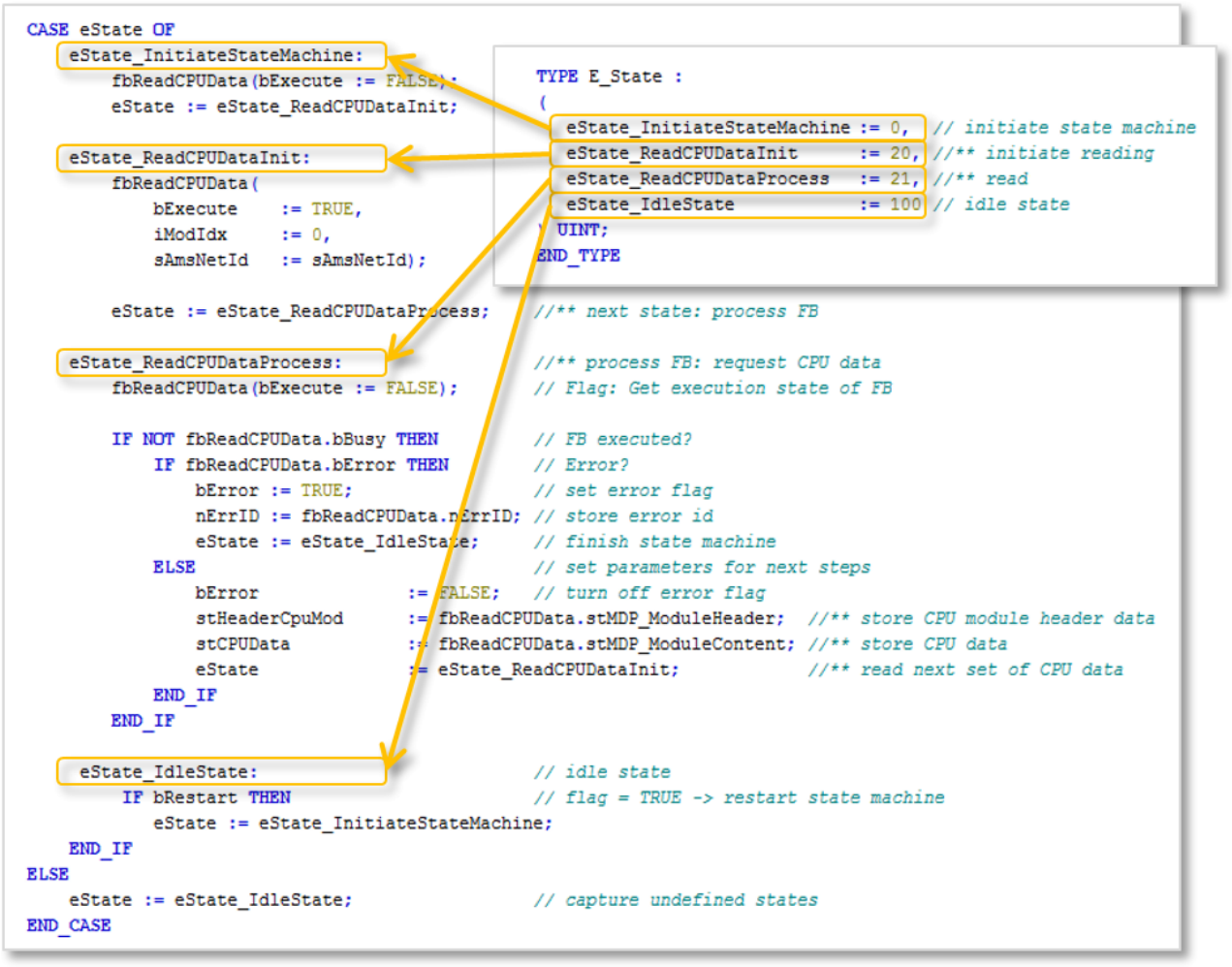
Die Parameter und der Funktionsbaustein werden im Programm an diesen Stellen genutzt:



```
CASE eState OF
    eState_InitiateStateMachine:                // initiate program parameters (if required)
        fbReadCPUData(bExecute := FALSE);       // ensure parameter 'bExecute' has FALSE state at startup
        eState := eState_ReadCPUDataInit;       //** initiate first state

    eState_ReadCPUDataInit:                     //** trigger FB: request CPU data
        fbReadCPUData(
            bExecute    := TRUE,
            iModIdx     := 0,
            sAmsNetId   := sAmsNetId);

        eState := eState_ReadCPUDataProces

    eState_ReadCPUDataProcess:
        fbReadCPUData(bExecute := FALSE);

        IF NOT fbReadCPUData.bBusy THEN
            IF fbReadCPUData.bError THEN
                bError := TRUE;
                nErrID := fbReadCPUData.nE
                eState := eState_IdleState
            ELSE
                bError              := FAL
                stHeaderCpuMod      := fbRe
                stCPUData           := fbReadCPUData.stMDP_ModuleContent; //** store CPU data
                eState              := eState_ReadCPUDataInit;            //** read next set of CPU data
            END_IF
        END_IF

     eState_IdleState:                          // idle state
         IF bRestart THEN                       // flag = TRUE -> restart state machine
            eState := eState_InitiateStateMachine;
    END_IF
ELSE
    eState := eState_IdleState;                 // capture undefined states
END_CASE
```

```
PROGRAM MAIN
VAR
    // internal use
    sAmsNetId           : STRING := '';     //** ADS Net ID (local = '')
    eState              : E_State;          // Enum with index for state machine
    bRestart            : BOOL;             // flag to trigger restart of statemachine

    // FB instances
    fbReadCPUData       : FB_MDP_CPU_Read;  // instance of FB for reading MDP index

    // results of execution
    bError              : BOOL;             // error flag (indicator: error occured)
    nErrID              : UDINT;            // last error ID
    stHeaderCpuMod      : ST_MDP_ModuleHeader;  // buffer for header data of CPU module
    stCPUData           : ST_MDP_CPU;       // Structure which will contain CPU data
END_VAR
```

Die State Machine

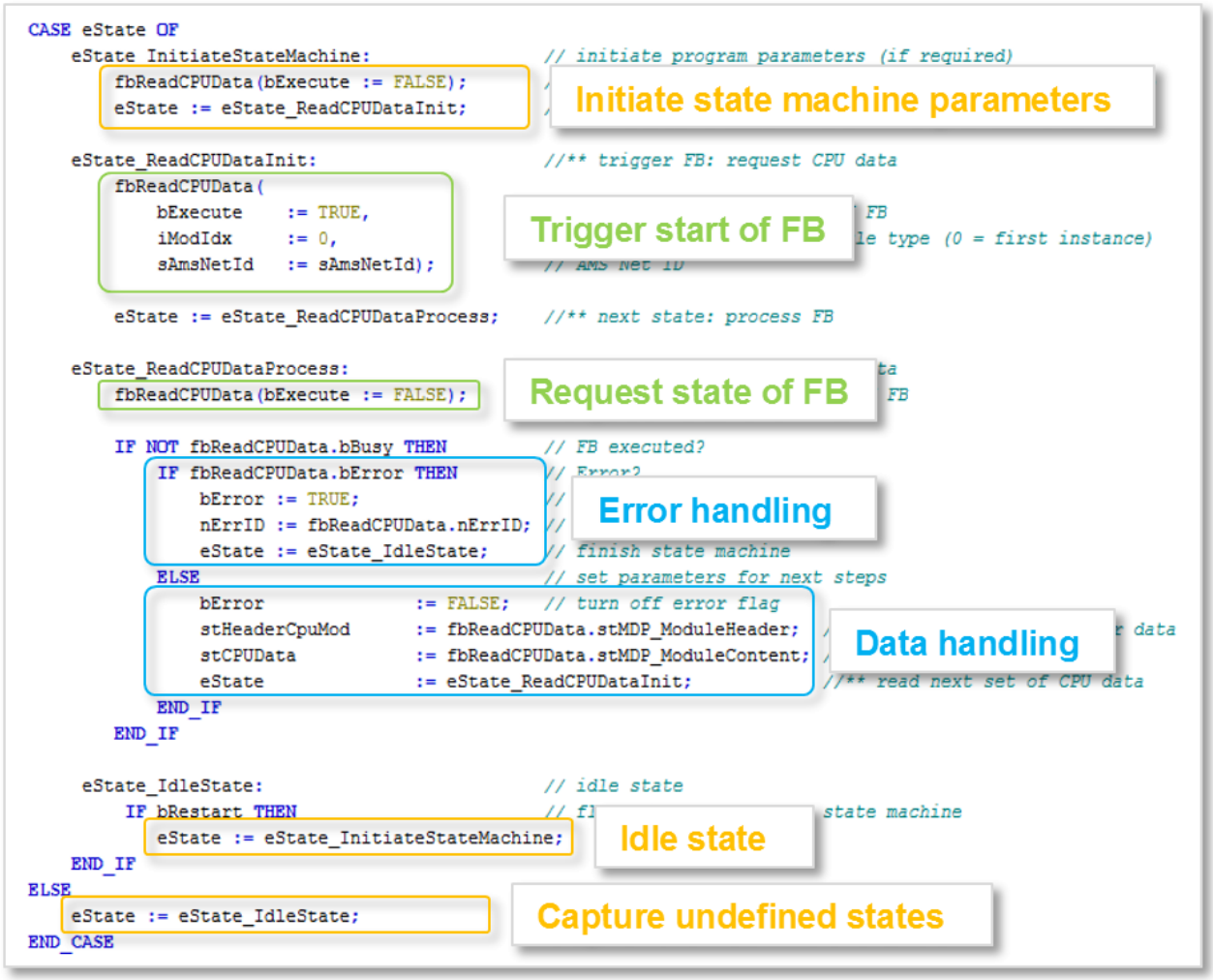


```
CASE eState OF
    eState_InitiateStateMachine:
        fbReadCPUData(bExecute := FALSE);
        eState := eState_ReadCPUDataInit;

    eState_ReadCPUDataInit:
        fbReadCPUData(
            bExecute    := TRUE,
            iModIdx     := 0,
            sAmsNetId   := sAmsNetId);

        eState := eState_ReadCPUDataProcess;    //** next state: process FB

    eState_ReadCPUDataProcess:                  //** process FB: request CPU data
        fbReadCPUData(bExecute := FALSE);       // Flag: Get execution state of FB

        IF NOT fbReadCPUData.bBusy THEN         // FB executed?
            IF fbReadCPUData.bError THEN        // Error?
                bError := TRUE;                 // set error flag
                nErrID := fbReadCPUData.nErrID; // store error id
                eState := eState_IdleState;      // finish state machine
            ELSE                                // set parameters for next steps
                bError              := FALSE;   // turn off error flag
                stHeaderCpuMod      := fbReadCPUData.stMDP_ModuleHeader;  //** store CPU module header data
                stCPUData           := fbReadCPUData.stMDP_ModuleContent; //** store CPU data
                eState              := eState_ReadCPUDataInit;            //** read next set of CPU data
            END_IF
        END_IF

    eState_IdleState:                           // idle state
         IF bRestart THEN                       // flag = TRUE -> restart state machine
            eState := eState_InitiateStateMachine;
    END_IF
ELSE
    eState := eState_IdleState;                 // capture undefined states
END_CASE
```

```
TYPE E_State :
(
    eState_InitiateStateMachine := 0,  // initiate state machine
    eState_ReadCPUDataInit      := 20, //** initiate reading
    eState_ReadCPUDataProcess   := 21, //** read
    eState_IdleState            := 100 // idle state
) UINT;
END_TYPE
```

Die Status der State Machine sind als Konstante ausgeführt, um eine einfache Anpassung des Programms zu ermöglichen. Damit muss der gewünschte „State“-Wert nur einmal zentral geändert werden. Die Status werden als Enumeratiosdeklaration im SPS-Projekt im Unterordner „DUTs“ als DUT unter dem Namen „E-State“ definiert.

Nachfolgend sind noch die verschiedenen Bereiche der State Machine und ihre Funktionen erläutert:



8.3 Abfrage des Lüfterstatus (generisch)

Dieses Beispiel zeigt den Zugriff auf die Daten der Lüftergeschwindigkeit über den generischen Funktionsbaustein FB_MDP_ReadElement [▶ 14]. Es kann zur Diagnose eines Lüfterausfalls verwendet werden (Lüftergeschwindigkeit = 0).

Voraussetzung ist das Vorhandensein eines Lüfters. Wenn kein Lüfter im IPC vorhanden ist, wird der angesprochene Modultyp vom IPC nicht unterstützt, und der Zugriffsversuch liefert die Fehlermeldung 16#EC800002 zurück.

Der Inhalt des FAN-Moduls ist in der Configuration Area der IPC-Diagnose beschrieben. Für den Zugriff wird der generische Funktionsbaustein FB_MDP_ReadElement verwendet.

Eine Beschreibung des Aufbaus dieses Beispielprogramms finden Sie im Beispiel Abfrage von CPU-Daten (generisch) [▶ 37].

Beispiel zum Zugriff über den generischen Funktionsbaustein FB_MDP_ReadElement

Enumeratiosdefinition

```
//** = simply adjust these lines if modifying code for own purposes

// central definition of state machine states
// (supports easy program modification)
{attribute 'qualified_only'}
TYPE E_State :
(
    Idle,                   // idle state
    ReadFanSpeedInit,       //** initiate reading fan speed
```

```
    ReadFanSpeedProcess      //** process reading fan speed
);
END_TYPE
```

```
PROGRAM MAIN
VAR
    // internal use
    sAmsNetId           : STRING := '';        //** ADS Net ID (local = '')
    eState              : E_State;             // Enum with index for state machine
    bStart              : BOOL := TRUE;        // flag to trigger restart of statemachine
    nData               : UINT;                // data storage for unsigned integer
    stMDP_Addr          : ST_MDP_Addr;         // structure will include all address parameters
    nModuleIndex        : USINT := 0;          //** Fan index (no. of fan)

    // FB instances
    fbReadMDPElement    : FB_MDP_ReadElement;  // instance of FB for reading MDP element

    // results of execution
    bError              : BOOL;                // error flag (indicator: error occured)
    nErrID              : UDINT;               // last error ID
    aFanSpeed           : ARRAY[0..1] OF UINT; //** buffer for speed of fans
END_VAR
```

Programmcode

```
// For an easy re-use of the following code for own purposes, parts of this sample program use
// "general" data names (and copy the results in specific variables after processing the code).
// Remark: Error 16#EC800002 means module type not supported (IPC does not provide this type of
information, e.g. does not have fans)

CASE eState OF
    E_State.Idle:
        IF bStart THEN
            bStart := FALSE;
            eState := E_State.ReadFanSpeedInit;    //** initiate first state
        END_IF

    E_State.ReadFanSpeedInit:                  //**
        stMDP_Addr.nArea := INT_TO_BYTE(eMDP_Area_ConfigArea); //** set area address to "Config
Area"
        stMDP_Addr.nTableId := 1;              //** table ID in module for "Fan properties"
        stMDP_Addr.nSubIdx := 1;               //** subindex in table ID for "Fan speed"

        fbReadMDPElement(
            bExecute := TRUE,                  // Flag: trigger execution of FB
            eModuleType := eMDP_ModT_Fan,      //** desired module type = Fan
            stMDP_Addr := stMDP_Addr,          // MDP address structure. Dynamic module ID added
internally.
            iModIdx := nModuleIndex,           //** instance of desired module type (0 = first
instance)
            pDstBuf := ADR(nData),             // buffer for storing data
            cbDstBufLen := SIZEOF(nData),      // length of buffer
            sAmsNetId := sAmsNetId,            // AMS Net ID
            );                                 //** Note: fbReadMDPElement.tTimeOut must be > cycle
time!


        eState := E_State.ReadFanSpeedProcess;//** next state: process FB

    E_State.ReadFanSpeedProcess:               //** process FB: request fan data
        fbReadMDPElement(bExecute := FALSE);   // Flag: Get execution state of FB
                                               //** Note: fbReadMDPElement.tTimeOut must be > cycle
time!

        IF NOT fbReadMDPElement.bBusy THEN     // FB executed?
            IF fbReadMDPElement.bError THEN    // Error?
                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 := E_State.Idle;        // finish state machine
            ELSE                               // set parameters for next steps
                bError := FALSE;               // turn off error flag
                aFanSpeed[nModuleIndex] := nData; //** store fan speed in dedicated array
                IF nModuleIndex = 0 THEN       //** Current fan = fan 1?
                    nModuleIndex := 1;         //** Read fan 2 (= second module instance) in next
loop
                ELSE                           //**
                    nModuleIndex := 0;         //** Read fan 1 (= first module instance) in next loop
                END_IF                         //**
                eState := E_State.ReadFanSpeedInit; //** next state
            END_IF
```

```
        END_IF

END_CASE
```

8.4 IPC-Seriennummern lesen

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

- 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 [▶ 14] 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 [▶ 13] verwendet.

Beispiel zur Abfrage der Seriennummern eines Beckhoff IPCs

Enumerationsdefinition

```
//** = simply adjust these lines if modifying code for own purposes

// central definition of state machine states
// (supports easy program modification)
{attribute 'qualified_only'}
TYPE E_State :
(
    Idle,                       // idle state
    ReadSnoMainboardInit,       //** initiate reading serial number of mainboard
    ReadSnoMainboardProcess,    //** process reading serial number of mainboard
    ReadSnoIPCInit,             //** initiate reading serial number of IPC
    ReadSnoIPCProcess           //** process reading serial number of IPC
);
END_TYPE
```

Variablendeklaration

```
PROGRAM MAIN
VAR
    // internal use
    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;  // instance of FB for reading MDP element
    fbReadMDPIndex           : FB_MDP_ReadIndex;    // instance of FB for reading MDP index

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

Programmcode

```
// For an easy re-use of the following code for own purposes, parts of this sample program use
// "general" data names (and copy the results in specific variables after processing the code).

CASE eState OF
     E_State.Idle:
         IF bStart THEN
           bStart := FALSE;
            eState := E_State.ReadSnoMainboardInit;  //** initiate first state
         END_IF

    //
** read serial number of mainboard ************************************************************

    E_State.ReadSnoMainboardInit:               //** trigger FB: request mainboard serial number
        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,                // Flag: trigger execution of FB
            eModuleType := eMDP_ModT_Mainboard, //
** desired module type / index = Mainboard
            stMDP_Addr  := stMDP_Addr,          // MDP address structure. Dynamic module ID will be
added internally.
            iModIdx     := 0,                   //
** Instance of desired module type (default: 0 = first instance)
            pDstBuf     := ADR(sData),          // buffer for storing data
            cbDstBufLen := SIZEOF(sData),       // length of buffer
            sAmsNetId   := sAmsNetId,           // AMS Net ID
            );

        eState := E_State.ReadSnoMainboardProcess; //** next state: process FB

    E_State.ReadSnoMainboardProcess:            //** process FB: request mainboard serial number
        fbReadMDPElement(bExecute := FALSE);    // Flag: Get execution state of FB

        IF NOT fbReadMDPElement.bBusy THEN      // FB executed?
            IF fbReadMDPElement.bError THEN     // Error?
                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 := E_State.Idle;         // finish state machine
            ELSE                                // set parameters for next steps
                bError := FALSE;                // turn off error flag
                sSerialNoMainboard := sData;    //
** store serial number of mainboard in dedicated variable
                eState := E_State.ReadSnoIPCInit; //** next state
            END_IF
        END_IF

    //
** read serial number of IPC ****************************************************************
    E_State.ReadSnoIPCInit:                     //
** trigger FB: request single index in MDP Device Area, IPC serial number
        sData           := '';                  // clear data buffer
        sSerialNoIPC    := '';                  //** clear buffer for serial number of IPC

        fbReadMDPIndex(
            bExecute    := TRUE,                // Flag: trigger execution of FB
            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),          // buffer for storing serial number
            cbDstBufLen := SIZEOF(sData),       // length of buffer
            sAmsNetId   := sAmsNetId,           // AMS Net ID
            );

        eState := E_State.ReadSnoIPCProcess;    //** next state: process FB

    E_State.ReadSnoIPCProcess:                  //
** process FB:: request single index in MDP Device Area, IPC serial number

        fbReadMDPIndex(bExecute := FALSE);      // flag: Get execution state of FB

        IF NOT fbReadMDPIndex.bBusy THEN        // FB executed?
            IF fbReadMDPIndex.bError THEN       // error?
                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 := E_State.Idle;         // finish state machine
            ELSE                                // set parameters for next steps
                bError := FALSE;                // turn off error flag
                sSerialNoIPC := sData;          //** store serial number of mainboard
                eState := E_State.Idle;         //
** set here next state if expanding the state machine
            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

8.5 IP-Adresse setzen

Dieses Beispiel zeigt den schreibenden Zugriff auf Daten der IPC-Diagnose über generelle Funktionsbausteine. In der gleichen Art und Weise kann auch auf alle anderen Elemente der Module der IPC-Diagnose zugegriffen werden.

Aktive Netzwerkverbindung

Die Änderung dieser Einstellungen erfordert jeweils eine aktive Netzwerkverbindung für den gewählten Ethernet/EtherCAT-Adapter. Ohne aktive Netzwerkverbindung können also keine Parameter (vor-) eingestellt werden.

Beispiel zum Setzen der IP-Adresse eines Beckhoff IPCs

Variablendeklaration

```
PROGRAM MAIN
// First DHCP is set off and then the given new IP address is set.
// The program code is executed only one time. To restart the program set bStart TRUE again.
VAR
    bStart     : BOOL := TRUE;
    nState     : INT := 100;
    fbScan     : FB_MDP_ScanModules;
    fbWrite    : FB_MDP_Write;
    stMDPAddr  : ST_MDP_Addr;
    bDHCP      : BOOL;
    sIP        : T_IPv4Addr := '174.18.3.154'; // the new ip address
    bError     : BOOL;
    nErrID     : UDINT;
END_VAR
```

Programmcode

```
CASE nState OF
100: // idle
    IF bStart THEN
        bStart := FALSE;
        nState := 00;
    END_IF

00: (* scan MDP module list for dyn.module id of NIC module *)
    fbScan(
        bExecute    := TRUE,
        nModuleType := eMDP_ModT_NIC,
        iModIdx     := 0, (* index of NIC module / network port *)
        sAmsNetId   := ''
        );

    nState := 01;

01:
    fbScan( bExecute:= FALSE );

    IF NOT fbScan.bBusy THEN
        IF NOT fbScan.bError THEN
            stMDPAddr.nArea     := INT_TO_BYTE(eMDP_Area_ConfigArea);
            stMDPAddr.nModuleId := fbScan.nDynModuleId;
            stMDPAddr.nTableId  := 1;
            nState              := 10;
        ELSE
            bError              := TRUE;
            nErrID              := fbScan.nErrID;
            nState              := 00;
        END_IF
    END_IF
```

```
10: // set DHCP off
    stMDPAddr.nSubIdx := 4; // DHCP
    bDHCP             := FALSE;

    fbWrite(
        bExecute      := TRUE,
        stMDP_DynAddr := stMDPAddr,
        pSrcBuf       := ADR(bDHCP),
        cbSrcBufLen   := SIZEOF(bDHCP),
        sAmsNetId     := ''
         );

    nState:= 11;

11:
    fbWrite( bExecute:= FALSE );
    IF NOT fbWrite.bBusy THEN
        IF NOT fbWrite.bError THEN
            nState     := 20;
        ELSE
            bError     := TRUE;
            nErrID     := fbWrite.nErrID;
            nState     := 10;
        END_IF
    END_IF

20: // set new IP address
    stMDPAddr.nSubIdx := 2; // IP address
    fbWrite(
        bExecute      := TRUE,
        stMDP_DynAddr := stMDPAddr,
        pSrcBuf       := ADR(sIP),
        cbSrcBufLen   := LEN(sIP),
        sAmsNetId     := ''
    );

    nState := 21;

21:
    fbWrite( bExecute:= FALSE );
    IF NOT fbWrite.bBusy THEN
        IF NOT fbWrite.bError THEN
            nState     := 100; (* NIC settings executed *)
        ELSE
            bError     := TRUE;
            nErrID     := fbWrite.nErrID;
            nState     := 20;
        END_IF
    END_IF
END_CASE
```

Mehr Informationen:
www.beckhoff.com/te1000

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