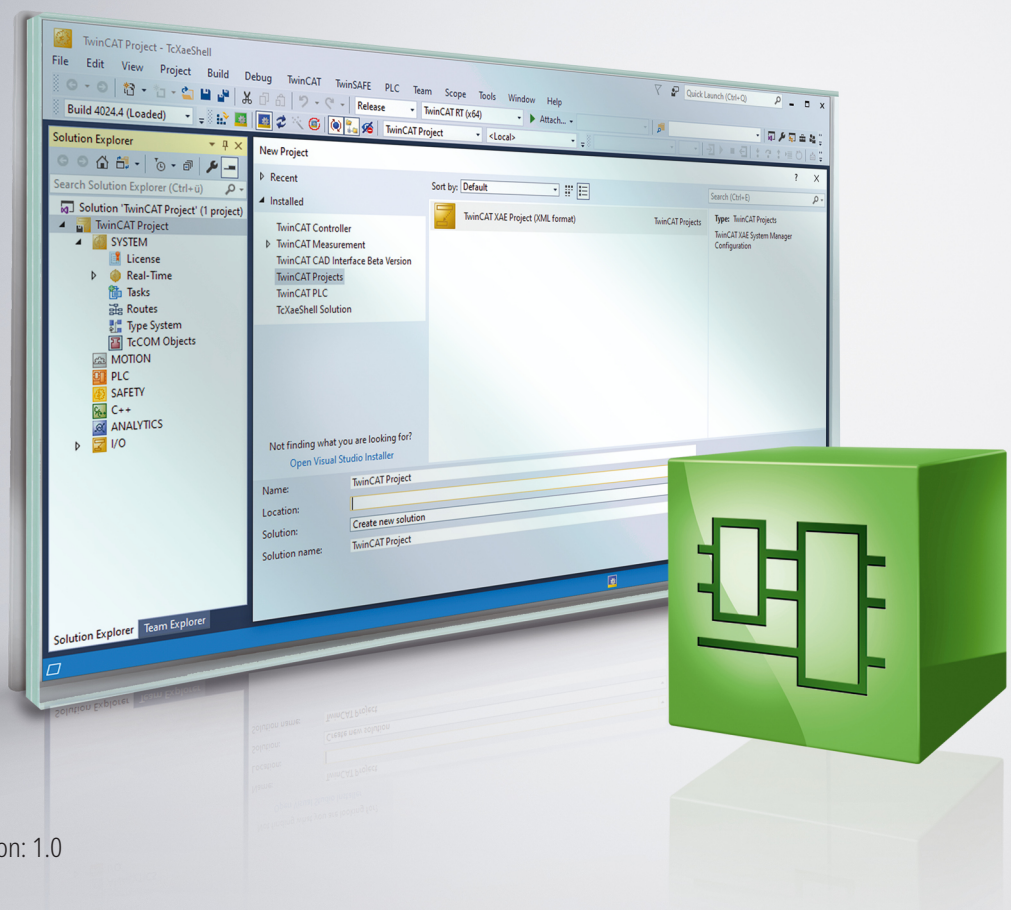
BECKHOFF New Automation Technology

Handbuch | DE

TE1000

TwinCAT 3 | PLC-Bibliothek: Tc3_DynamicMemory



20.08.2020 | Version: 1.0

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

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

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

Disclaimer

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

Marken

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

Patente

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

EtherCAT®

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

Copyright

1.2 Sicherheitshinweise

Sicherheitsbestimmungen

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

Haftungsausschluss

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

Qualifikation des Personals

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

Erklärung der Symbole

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

⚠ GEFAHR

Akute Verletzungsgefahr!

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

⚠ WARNUNG

Verletzungsgefahr!

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

⚠ VORSICHT

Schädigung von Personen!

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

HINWEIS

Schädigung von Umwelt oder Geräten

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

Tipp oder Fingerzeig

Dieses Symbol kennzeichnet Informationen, die zum besseren Verständnis beitragen.

2 Übersicht

Die SPS Bibliothek Tc3_DynamicMemory vereinfacht die Verwendung von dynamischem Speicher in der SPS. Die Speicherallokationen bedienen sich aus dem TwinCAT Router Speicher.

Es stehen Funktionsbausteine zur Verfügung, um Puffer für verschiedenste Daten anzulegen. Diese kümmern sich automatisch auch um die nötige Freigabe des Speichers zum Ende der Laufzeit.

Zudem können eigene Speicherallokationen mit einem Speicher-Manager überwacht werden, welcher Methoden analog zu den Operatoren __NEW() und __DELETE() anbietet. Bei Verwendung der Bibliothek kann und sollte also auf die Verwendung der zwei genannten Operatoren verzichtet werden.

Bei Verwendung dieser SPS Bibliothek resultiert so ein besserer Überblick über die Menge des bereits allokierten Speichers.

Vorteile von dynamisch allokiertem Speicher in der TwinCAT Echtzeit

Es muss nur die Speichermenge allokiert werden, welche zur Laufzeit benötigt wird. Das vorherige Anlegen großer Reserven entfällt.

Auf unerwartet hohen Speicherbedarf, bspw. beim Datenaustausch über eine Kommunikationsschnittstelle, kann in der Laufzeit reagiert werden und dieser ggf. sogar nur temporär zur Verfügung gestellt werden.

Nachteile von dynamisch allokiertem Speicher in der TwinCAT Echtzeit

- Eine Speicherallokation während der Laufzeit ist rechenintensiv.
- Eine Speicherallokation kann fehlschlagen, wenn kein entsprechend großer Block mehr im Speicher frei ist.
- Fragmentierungen im TwinCAT Router Speicher sind möglich.
- Explizite Speicherfreigaben sind nötig.
- Eine Typänderung einer dynamisch allokierten Instanz per Online-Change ist nicht möglich.

>>> Nur bei hohem Nutzen und ohne sinnvolle Alternativen sollte in der TwinCAT Echtzeit Speicher dynamisch allokiert werden.

Systemvoraussetzung

Target System	Win7, WES7, WEC7, Win10 IPC oder CX (x86, x64, ARM)
Min. TwinCAT-Version	3.1.4024.7
Min. TwinCAT-Level	TC1200 TC3 PLC

3 Funktionsbausteine

3.1 FB_DynMem_Buffer

Der Funktionsbaustein stellt einen Puffer aus dynamisch allokiertem Speicher zur Verfügung. Dieser Puffer kann für individuelle Daten oder Datenblöcke verwendet werden.

Dem Funktionsbaustein wird bei der Deklaration eine Instanz des Funktionsbausteins FB_DynMem_Manager [▶ 9] übergeben, wodurch die Speicherallokationen überwacht werden.

Deklaration

```
fbMyBuffer : FB_DynMem_Buffer(ipMemMan := fbMyMemMan);
```

Properties

bAvailable: ist TRUE, falls der Puffer verfügbar ist.

nBufferSize: gibt die aktuelle Größe des Puffers in Bytes an.

pBuffer: stellt einen Pointer auf den internen dynamisch allokierten Puffer zur Verfügung.

Methoden

CreateBuffer():

Die Methode erzeugt den Puffer, indem Speicher zur Laufzeit allokiert wird. Neben der gewünschten Größe in Bytes wird angegeben, ob der Speicherblock hierbei genullt werden soll.
Bei erfolgreicher Abarbeitung liefert die Methode TRUE zurück.

```
METHOD CreateBuffer : BOOL
VAR_INPUT
    nSize    : UDINT;    // buffer size [in bytes]
    bReset   : BOOL;     // zero the allocated memory
END_VAR
```

Clear():

Die Methode leert den Puffer. Der komplette Pufferspeicher wird hierbei genullt.
Bei erfolgreicher Abarbeitung liefert die Methode TRUE zurück.

```
METHOD Clear : BOOL
VAR_INPUT
END_VAR
```

Resize():

Die Methode ändert die Größe des Puffers. Dabei kann angegeben werden, ob der bisherige Pufferinhalt erhalten bleiben soll.
Bei erfolgreicher Abarbeitung liefert die Methode TRUE zurück.

```
METHOD Resize : BOOL
VAR_INPUT
    nSize     : UDINT;   // new buffer size [in bytes]
    bPreserve : BOOL;    // TRUE => preserve old content, FALSE=> don't preserve old content
    bReset    : BOOL;    // zero the allocated memory (before preserving)
END_VAR
```

Geänderter Pointer

Bei Größenänderung des Puffers kann sich die Adresse des Puffers ändern. Es ist deshalb vor Verwendung des Puffers zwingend notwendig, sich die neue Adresse zu holen.

DeleteBuffer():

Die Methode entfernt den Puffer, indem der allokierte Speicher wieder freigegeben wird.
Bei erfolgreicher Abarbeitung liefert die Methode TRUE zurück.

```
METHOD DeleteBuffer : BOOL
VAR_INPUT
END_VAR
```

Ungültiger Pointer

Nach Entfernung des Puffers ist die zuvor verwendete Pufferadresse ungültig und darf nicht mehr verwendet werden. Es wird empfohlen noch existierende Pointer-Variablen explizit auf NULL zu setzen.

Beispiel

Folgender Beispiel-Code zeigt die Verwendung vom FB_DynMem_Buffer zur Allokation von Speicher während der Laufzeit. Der allokierte Speicher wird einmal als Strukturinstanz und ein andermal als Array verwendet.

```
PROGRAM MAIN
VAR
    bAllocateOnce   : BOOL := TRUE;
    fbMemMan        : FB_DynMem_Manager;

    fbBuffer_Struct : FB_DynMem_Buffer(ipMemMan:=fbMemMan);
    pStruct         : POINTER TO ST_MyStruct;

    fbBuffer_Array  : FB_DynMem_Buffer(ipMemMan:=fbMemMan);
    pArray          : POINTER TO LREAL;
    nArrayLength    : UINT := 10;
    bResizeArray    : BOOL;
END_VAR
```

```
IF bAllocateOnce THEN
    bAllocateOnce := FALSE;

    fbBuffer_Struct.CreateBuffer(nSize:=SIZEOF(ST_MyStruct), bReset:=TRUE);
    // after successfully creating the buffer fbMemMan shows an increased nAllocatedSize [bytes] and
nBufferCount=1
    pStruct := fbBuffer_Struct.pBuffer;

    fbBuffer_Array.CreateBuffer(nSize:=nArrayLength*SIZEOF(LREAL), bReset:=TRUE);
    // after successfully creating the buffer fbMemMan shows an increased nAllocatedSize [bytes] and
nBufferCount=2
    pArray := fbBuffer_Array.pBuffer;
END_IF

IF pStruct <> 0 THEN
    pStruct^.nCtrlValue := 7;
END_IF

IF pArray <> 0 THEN
    pArray[3] := 7.5;
END_IF

IF bResizeArray THEN
    bResizeArray := FALSE;
    nArrayLength := 15;
    fbBuffer_Array.Resize(nSize:=nArrayLength*SIZEOF(LREAL), bPreserve:=TRUE, bReset:=TRUE);
    // after successfully creating the buffer fbMemMan shows an increased nAllocatedSize [bytes] and
nBufferCount=2
    pArray := fbBuffer_Array.pBuffer;
END_IF
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4024.7	IPC oder CX (x86, x64, ARM)	Tc3_DynamicMemory

3.2 FB_DynMem_Manager

Der Funktionsbaustein stellt die Möglichkeit bereit, dynamisch Speicher zu allokieren und diesen wieder freizugeben. Hierbei geben integrierte Zähler den Speicherverbrauch an und ermöglichen so einen Überblick über den bereits allokierten Speicher.

Es wird dem Anwender freigestellt, ob eine Instanz über die gesamte Applikation hinweg genutzt wird, oder ob mehrere einzelne Instanzen für verschiedene Programmteile verwendet werden. Letzteres kann bei größeren Anwendungen von Vorteil im Diagnosefall sein. Dieser Funktionsbaustein ist MultiTasking-fähig und eine Instanz kann somit aus verschiedenen Task-Kontexten heraus verwendet werden.

Sollte der Funktionsbaustein beim Herunterfahren der SPS feststellen, dass nicht die gesamte Menge des allokierten Speichers bereits freigegeben ist, wird eine Fehlermeldung als Event an den TC3 Event Logger versendet.

Properties

nAllocatedSize: gibt die aktuelle Menge des mit dieser Funktionsbausteininstanz allokierten Speichers in Bytes an.

nBufferCount: gibt die Menge der mit dieser Funktionsbausteininstanz allokierten Puffer an.

nObjectCount: - future reserved -

Methoden

Alloc():

Die Methode allokiert dynamisch Speicher und liefert einen Pointer auf diesen Speicherblock zurück. Bei fehlgeschlagener Allokation, bspw. weil kein freier Speicher verfügbar ist, liefert die Methode NULL zurück.

Neben der gewünschten Größe in Bytes wird angegeben, ob der Speicherblock genullt werden soll.

```
METHOD Alloc : PVOID
VAR_INPUT
    nSize    : UDINT;    // requested size in bytes
    bReset   : BOOL;     // zero the allocated memory
END_VAR
```

Free():

Die Methode gibt einen zuvor allokierten Speicher wieder frei. Hierbei muss der Speicherblock zuvor mit derselben Funktionsbausteininstanz allokiert worden sein und die Größe dieses Speicherblocks muss angegeben werden.

```
METHOD Free
VAR_INPUT
    p       : REFERENCE TO PVOID; // the given pointer is reset to zero after deletion
    nSize   : UDINT;
END_VAR
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4024.7	IPC oder CX (x86, x64, ARM)	Tc3_DynamicMemory

3.3 FB_DynMem_Manager2

Der Funktionsbaustein stellt die Möglichkeit bereit, dynamisch Speicher zu allokieren und diesen wieder freizugeben. Hierbei geben integrierte Zähler den Speicherverbrauch an und ermöglichen so einen Überblick über den bereits allokierten Speicher.

Es wird dem Anwender freigestellt, ob eine Instanz über die gesamte Applikation hinweg genutzt wird, oder ob mehrere einzelne Instanzen für verschiedene Programmteile verwendet werden. Letzteres kann bei größeren Anwendungen von Vorteil im Diagnosefall sein. Dieser Funktionsbaustein ist MultiTasking-fähig und eine Instanz kann somit aus verschiedenen Task-Kontexten heraus verwendet werden.

Sollte der Funktionsbaustein beim Herunterfahren der SPS feststellen, dass nicht die gesamte Menge des allokierten Speichers bereits freigegeben ist, wird eine Fehlermeldung als Event an den TC3 Event Logger versendet.

Der Funktionsbaustein FB_DynMem_Manager2 erweitert FB_DynMem_Manager mit dem Wissen über die Größe der allokierten Speicherblöcke. Dies vereinfacht die Speicherfreigabe mittels der Methode Free2(), weil die Größe des Speicherblocks nicht angegeben werden muss.

Properties

nAllocatedSize: gibt die aktuelle Menge des mit dieser Funktionsbausteininstanz allokierten Speichers in Bytes an.

nBufferCount: gibt die Menge der mit dieser Funktionsbausteininstanz allokierten Puffer an.

nObjectCount: - future reserved -

Methoden

Alloc():

Die Methode allokiert dynamisch Speicher und liefert einen Pointer auf diesen Speicherblock zurück. Bei fehlgeschlagener Allokation, bspw. weil kein freier Speicher verfügbar ist, liefert die Methode NULL zurück.

Neben der gewünschten Größe in Bytes wird angegeben, ob der Speicherblock genullt werden soll.

```
METHOD Alloc : PVOID
VAR_INPUT
    nSize    : UDINT;    // requested size in bytes
    bReset   : BOOL;     // zero the allocated memory
END_VAR
```

Free2():

Die Methode gibt einen zuvor allokierten Speicher wieder frei. Hierbei muss der Speicherblock zuvor mit derselben Funktionsbausteininstanz allokiert worden sein.

```
METHOD Free2
VAR_INPUT
    p       : REFERENCE TO PVOID; // the given pointer is reset to zero after deletion
END_VAR
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4024.7	IPC oder CX (x86, x64, ARM)	Tc3_DynamicMemory

4 Beispiele

Folgender Beispiel-Code zeigt die Verwendung vom FB_DynMem_Buffer zur Allokation von Speicher während der Laufzeit. Der allokierte Speicher wird einmal als Strukturinstanz und ein andermal als Array verwendet.

```
PROGRAM MAIN
VAR
    bAllocateOnce   : BOOL := TRUE;
    fbMemMan        : FB_DynMem_Manager;

    fbBuffer_Struct : FB_DynMem_Buffer(ipMemMan:=fbMemMan);
    pStruct         : POINTER TO ST_MyStruct;

    fbBuffer_Array  : FB_DynMem_Buffer(ipMemMan:=fbMemMan);
    pArray          : POINTER TO LREAL;
    nArrayLength    : UINT := 10;
    bResizeArray    : BOOL;
END_VAR
```

```
IF bAllocateOnce THEN
    bAllocateOnce := FALSE;

    fbBuffer_Struct.CreateBuffer(nSize:=SIZEOF(ST_MyStruct), bReset:=TRUE);
    // after successfully creating the buffer fbMemMan shows an increased nAllocatedSize [bytes] and
nBufferCount=1
    pStruct := fbBuffer_Struct.pBuffer;

    fbBuffer_Array.CreateBuffer(nSize:=nArrayLength*SIZEOF(LREAL), bReset:=TRUE);
    // after successfully creating the buffer fbMemMan shows an increased nAllocatedSize [bytes] and
nBufferCount=2
    pArray := fbBuffer_Array.pBuffer;
END_IF

IF pStruct <> 0 THEN
    pStruct^.nCtrlValue := 7;
END_IF

IF pArray <> 0 THEN
    pArray[3] := 7.5;
END_IF

IF bResizeArray THEN
    bResizeArray := FALSE;
    nArrayLength := 15;
    fbBuffer_Array.Resize(nSize:=nArrayLength*SIZEOF(LREAL), bPreserve:=TRUE, bReset:=TRUE);
    // after successfully creating the buffer fbMemMan shows an increased nAllocatedSize [bytes] and
nBufferCount=2
    pArray := fbBuffer_Array.pBuffer;
END_IF
```

Mehr Informationen:
www.beckhoff.de/te1000

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