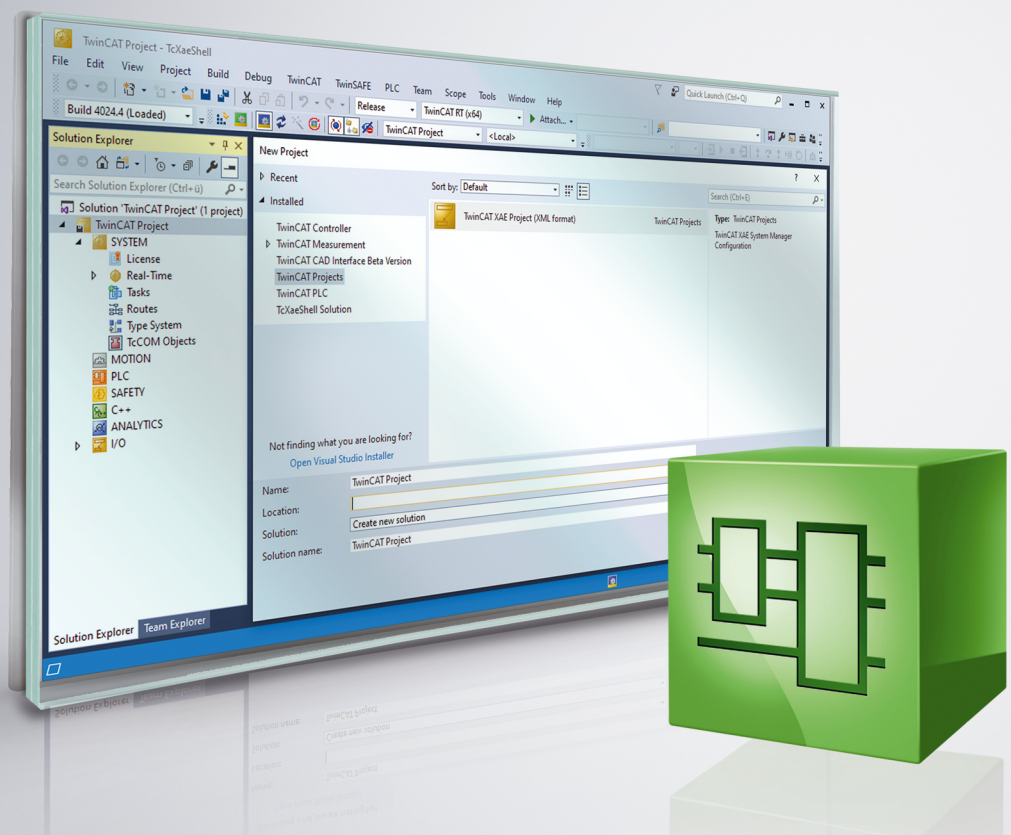
BECKHOFF New Automation Technology

Manual | EN

TE1000

TwinCAT 3 | PLC Library: Tc3_DynamicMemory



20.08.2020 | Version: 1.0

Table of contents

1 Foreword

1.1 Notes on the documentation

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

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

Disclaimer

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

Trademarks

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

Patent Pending

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

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

Copyright

1.2 Safety instructions

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

⚠ DANGER

Serious risk of injury!

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

⚠ WARNING

Risk of injury!

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

⚠ CAUTION

Personal injuries!

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

NOTE

Damage to the environment or devices

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

Tip or pointer

This symbol indicates information that contributes to better understanding.

2 Introduction

The PLC library Tc3_DynamicMemory simplifies the use of dynamic memory in the PLC. The memory allocations avail themselves of the TwinCAT Router memory.

Function blocks are available for creating buffers for the most diverse data. They also automatically take care of the necessary release of the memory at the end of runtime.

In addition, your own memory allocations can be monitored with a memory manager, which offers methods analogous to the operators __NEW() and __DELETE(). When using the library, therefore, you can and should refrain from using the two operators named above.

This results in a better overview of the size of memory already allocated when using this PLC library.

Advantages of dynamically allocated memory in the TwinCAT real-time

Only the size of memory required at runtime needs to be allocated. The previous creation of large reserves is no longer necessary.

It is possible to react during the runtime to an unexpectedly high memory requirement, e.g. during data exchange via a communication interface, and if necessary to make the memory available only temporarily.

Disadvantages of dynamically allocated memory in the TwinCAT real-time

- A memory allocation during the runtime is computationally intensive.
- A memory allocation can fail if no block of an appropriate size is free in the memory.
- Fragmentations in the TwinCAT Router memory are possible.
- Explicit memory releases are necessary.
- A type change of a dynamically allocated instance by Online Change is not possible.

>>> Memory should be allocated dynamically in the TwinCAT real-time only in the case of high benefit and where there is no sensible alternative.

System requirements

Target System	Win7, WES7, WEC7, Win10 IPC or CX (x86, x64, ARM)
Min. TwinCAT version	3.1.4024.7
Min. TwinCAT level	TC1200 TC3 PLC

3 Function blocks

3.1 FB_DynMem_Buffer

This function block provides a buffer of dynamically allocated memory. This buffer can be used for individual data or data blocks.

During the declaration, an instance of the function block FB_DynMem_Manager [▶ 9] is transferred to the function block, as a result of which the memory allocations are monitored.

Declaration

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

Properties

bAvailable: TRUE if the buffer is available.

nBufferSize: indicates the current size of the buffer in bytes.

pBuffer: provides a pointer to the internal dynamically allocated buffer.

Methods

CreateBuffer():

The method generates the buffer by allocating memory at runtime. In addition to the desired size in bytes, an indication is given as to whether the memory block should be zeroed during this process.
The method returns TRUE if execution was successful.

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

Clear():

This method clears the buffer. The complete buffer is zeroed in the process.
The method returns TRUE if execution was successful.

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

Resize():

This method changes the size of the buffer. When doing so, it is possible to specify whether the previous buffer contents should be retained.
The method returns TRUE if execution was successful.

```
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
```

Changed pointer

If the size of the buffer changes, the address of the buffer can also change. It is therefore absolutely necessary to obtain the new address before using the buffer.

DeleteBuffer():

This method deletes the buffer by releasing the allocated memory again.
The method returns TRUE if execution was successful.

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

Invalid pointer

After deleting the buffer, the previously used buffer address is invalid and may no longer be used. It is recommended to explicitly set any pointer variables still existing to ZERO.

Example

The following sample code shows the use of FB_DynMem_Buffer to allocate memory during the runtime. The allocated memory is used once as a structure instance and another time as an array.

```
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
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.7	IPC or CX (x86, x64, ARM)	Tc3_DynamicMemory

3.2 FB_DynMem_Manager

This function block provides the possibility to dynamically allocate memory and to release it again. Integrated counters indicate the memory consumption and thus enable an overview of the already allocated memory.

It is entirely up to the user to decide whether an instance is used across the entire application or whether several individual instances are used for different program parts. With larger applications, the latter may be advantageous in the case of diagnosis. This function block is capable of multi-tasking and an instance can thus be used from various task contexts.

If the function block determines that not all of the allocated memory has already been released when shutting down the PLC, an error message is sent as an event to the TC3 Event Logger.

Properties

nAllocatedSize: indicates the current size in bytes of memory allocated with this function block instance.

nBufferCount: indicates the current size of buffer allocated with this function block instance.

nObjectCount: - future reserved -

Methods

Alloc():

This method dynamically allocates memory and returns a pointer to this memory block. If allocation fails, e.g. because no free memory is available, the method returns ZERO.

An indication is given next to the desired size in bytes as to whether the memory block should be zeroed.

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

Free():

This method releases previously allocated memory again. The memory block must be allocated with the same function block instance beforehand and the size of this memory block must be specified.

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

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.7	IPC or CX (x86, x64, ARM)	Tc3_DynamicMemory

3.3 FB_DynMem_Manager2

This function block provides the possibility to dynamically allocate memory and to release it again. Integrated counters indicate the memory consumption and thus enable an overview of the already allocated memory.

It is entirely up to the user to decide whether an instance is used across the entire application or whether several individual instances are used for different program parts. With larger applications, the latter may be advantageous in the case of diagnosis. This function block is capable of multi-tasking and an instance can thus be used from various task contexts.

If the function block determines that not all of the allocated memory has already been released when shutting down the PLC, an error message is sent as an event to the TC3 Event Logger.

The function block FB_DynMem_Manager2 extends FB_DynMem_Manager by the knowledge of the size of the allocated memory blocks. This simplifies the memory release by means of the method Free2(), because the size of the memory block does not need to be specified.

Properties

nAllocatedSize: indicates the current size in bytes of memory allocated with this function block instance.

nBufferCount: indicates the current size of buffer allocated with this function block instance.

nObjectCount: - future reserved -

Methods

Alloc():

This method dynamically allocates memory and returns a pointer to this memory block. If allocation fails, e.g. because no free memory is available, the method returns ZERO.

An indication is given next to the desired size in bytes as to whether the memory block should be zeroed.

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

Free2():

This method releases previously allocated memory again. The memory block must have been allocated with the same function block instance beforehand.

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

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.7	IPC or CX (x86, x64, ARM)	Tc3_DynamicMemory

4 Samples

The following sample code shows the use of FB_DynMem_Buffer to allocate memory during the runtime. The allocated memory is used once as a structure instance and another time as an array.

```
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
```

More Information:
www.beckhoff.com/te1000

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