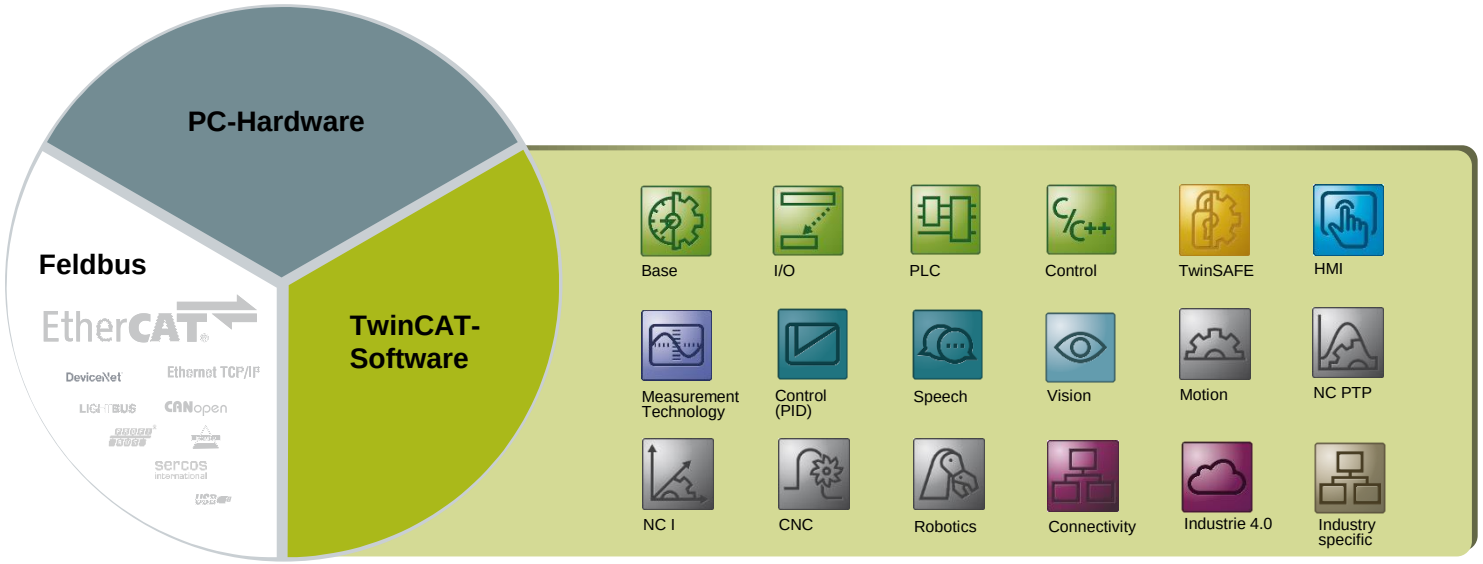
TwinCAT 3 C/C++

BECKHOFF

Motivation

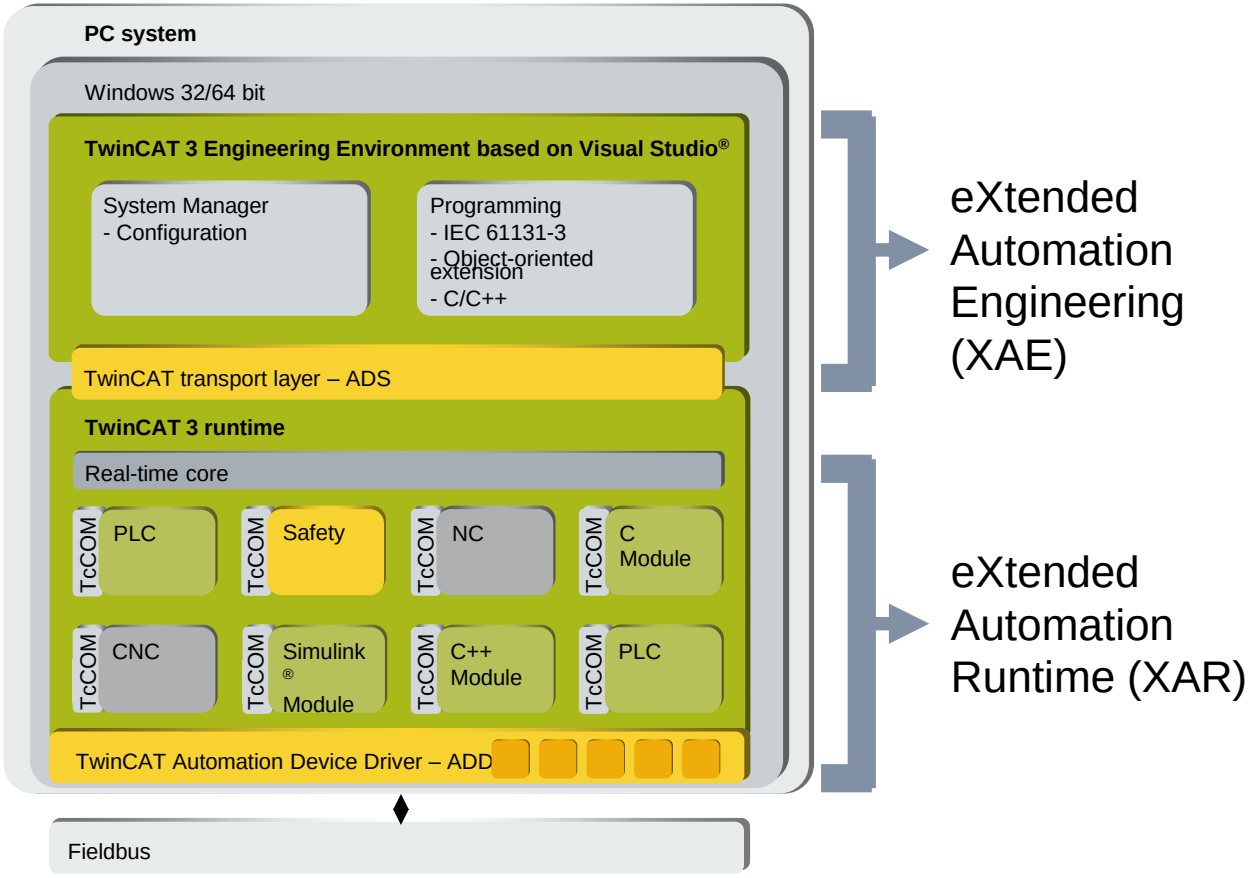
BECKHOFF

PC-based control technology from Beckhoff sets new standards in automation.



Architecture (XAA)

BECKHOFF



Multi-and many-core support

BECKHOFF

Overview

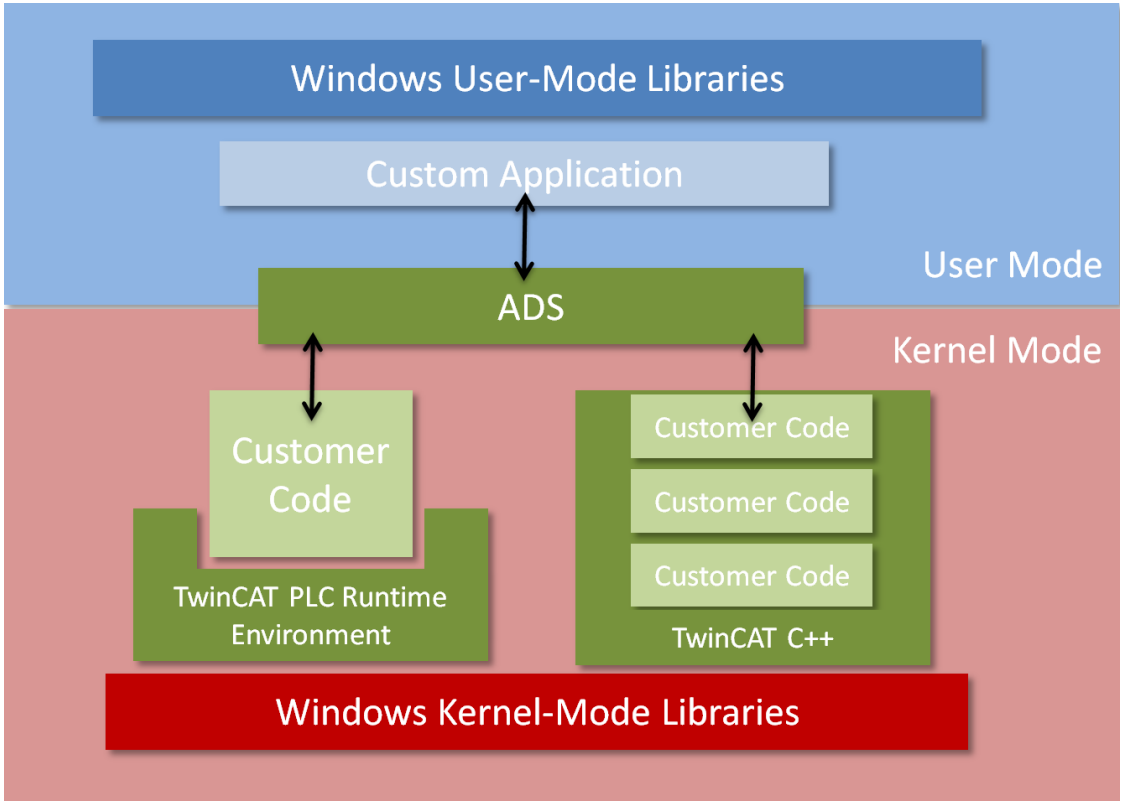
BECKHOFF

- TwinCAT C++
  - Quickstart
  - Coming from Usermode
  - Workflow
- TcCOM Modules
  - Concept
- Using TcCOM Interfaces
- TwinCAT C++ Development
- Documentation: Infosys
- Conclusion

Introduction – C++ and PLC Runtime

BECKHOFF

- TwinCAT C++ Modules directly use Windows Kernel-Mode Library thus are not in a „hosted“ environment like the PLC.
- Communication via ADS is completely equivalent
- C++ Modules use TcCOM concept as the PLC for integration into the TwinCAT XAR



Overview

BECKHOFF

- TwinCAT C++
  - Quickstart
  - Coming from Usermode
  - Workflow
- TcCOM Modules
  - Concept
- Using TcCOM Interfaces
- TwinCAT C++ Development
- Documentation: Infosys
- Conclusion

Content

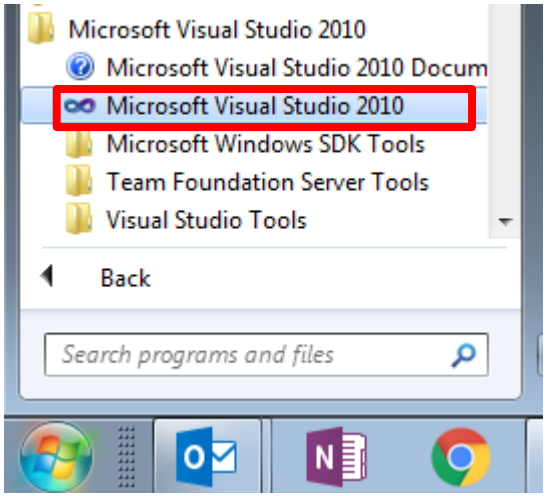
BECKHOFF

- Create a TwinCAT 3 project
- Create a TwinCAT 3 C++ project
- Implement TwinCAT 3 C++ project
- Create an instance
- Create a TwinCAT 3 task
- Debug the TwinCAT 3 C++ module

TwinCAT 3 - Start

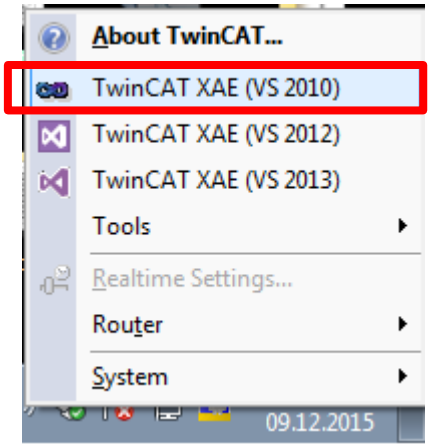
BECKHOFF

- Start Microsoft Visual Studio 2010/2012/2013
  - From Start->Programs



  or

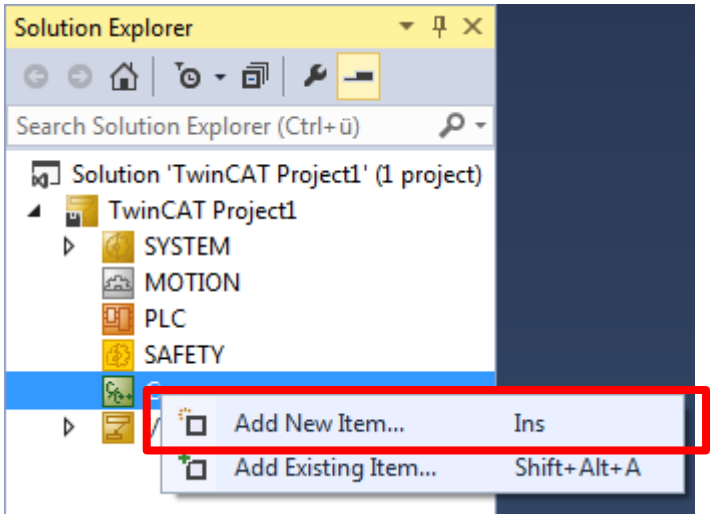
  - From TwinCAT Quickstart



TwinCAT 3 – C++ project creation

BECKHOFF

- Create a TwinCAT Project and
- Right click "C++" and select "Add New Item...“



- Please note: TwinCAT only shows the C++ node,
  - if system requirements are met
    or
  - a C++ project was added to the TwinCAT project before

Content

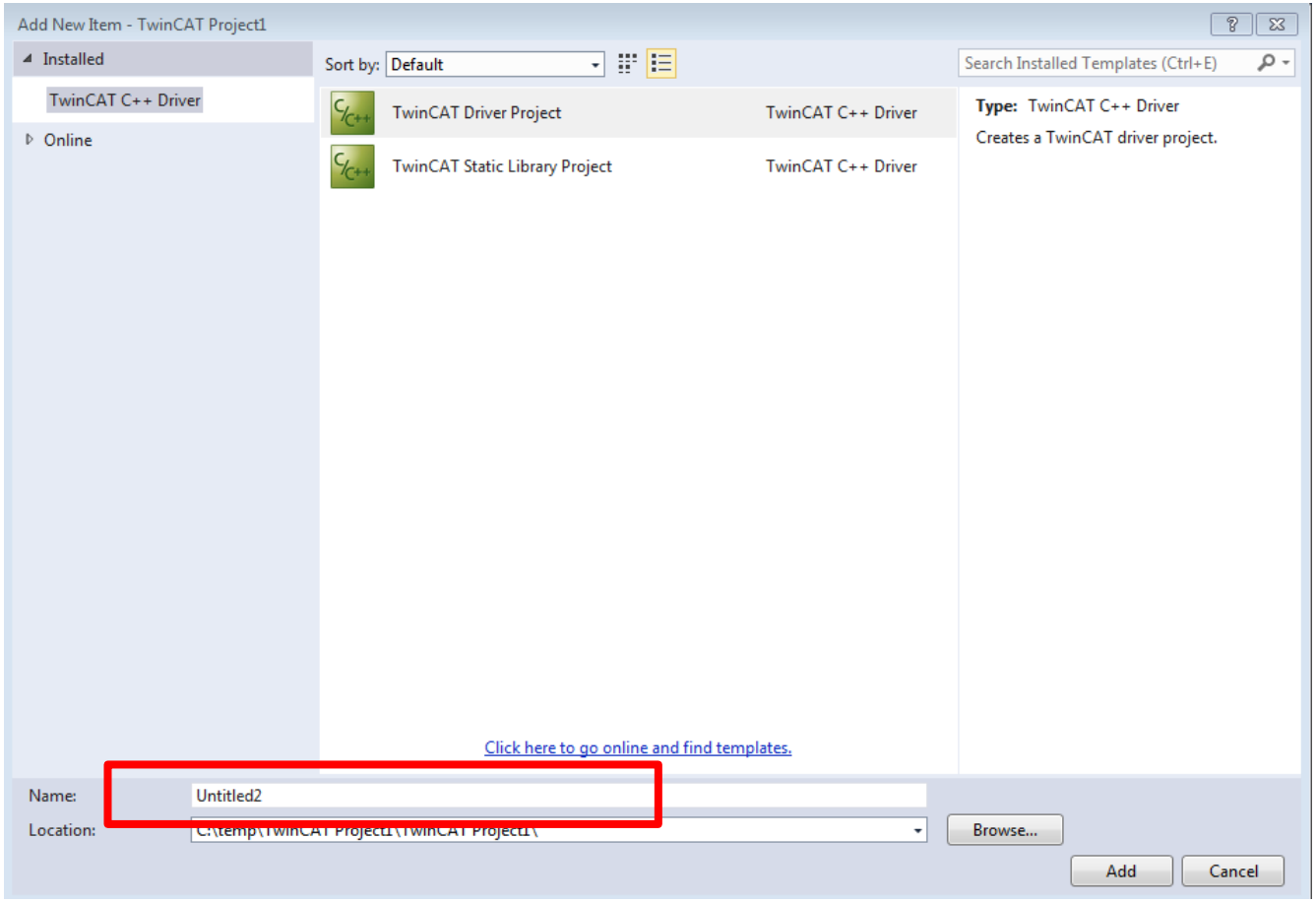
BECKHOFF

- Create a TwinCAT 3 project
- Create a TwinCAT 3 C++ project
- Implement TwinCAT 3 C++ project
- Create an instance
- Create a TwinCAT 3 task
- Debug the TwinCAT 3 C++ module

TwinCAT 3 – C++ project creation

BECKHOFF

- select "TwinCAT Driver Project", optional enter a project name and continue with "OK“



TwinCAT 3 – C++ project creation

BECKHOFF

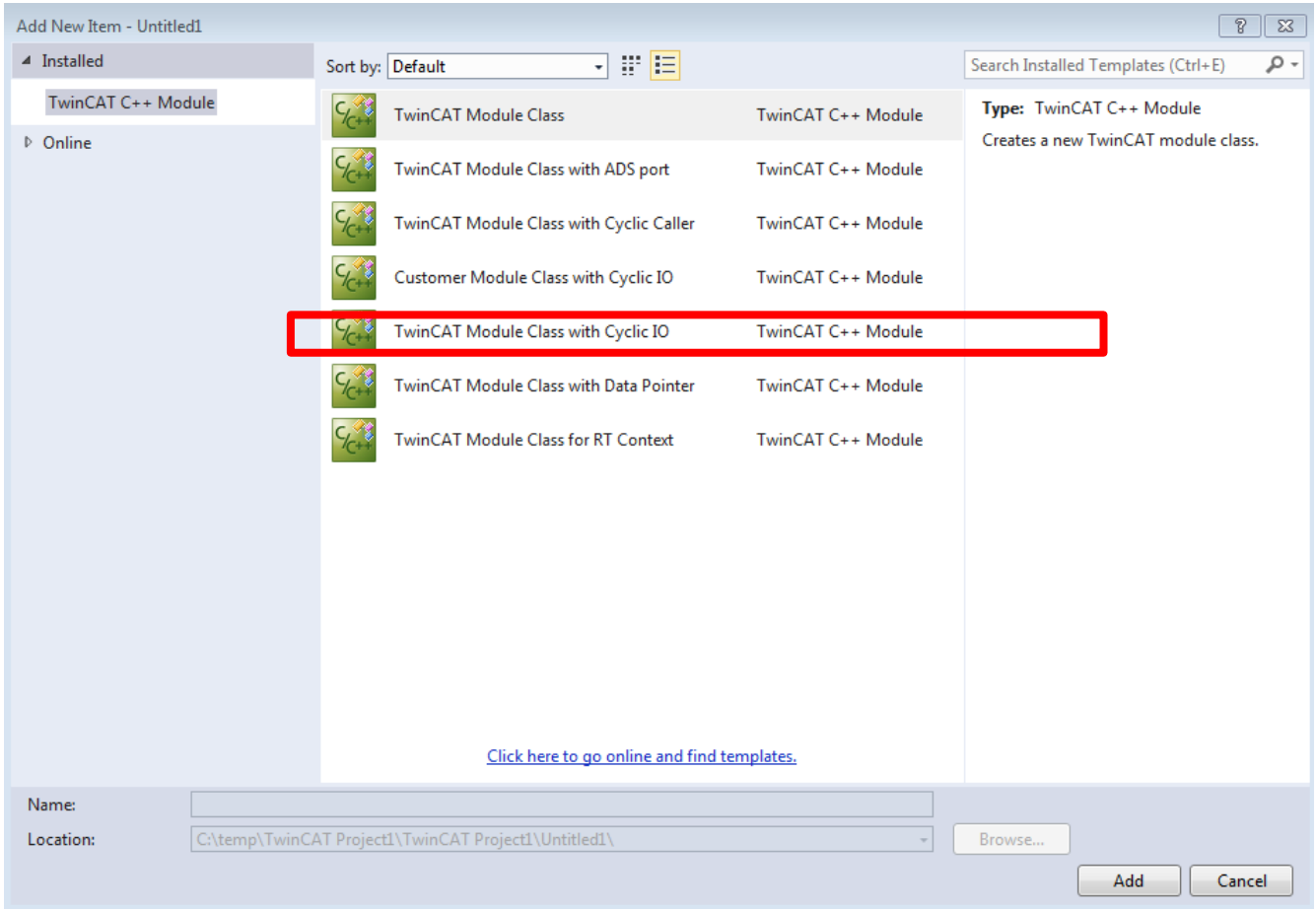
In the upcoming "TwinCAT Module Wizard" following templates are available:

- **"TwinCAT Module Class"**
  - Creates a new TwinCAT module class.
- **"TwinCAT Module Class with ADS Port"**
  - Creates a new TwinCAT module class which implements an ADS port
- **"TwinCAT Module Class with Cyclic Caller"**
  - Creates a new TwinCAT module class which implements the cyclic caller interface.
- **"TwinCAT Module Class with Cyclic IO“ (this will be used here)**
  - Creates a new TwinCAT module class which implements the cyclic caller interface and which has an input and output data area.
- **"TwinCAT Module Class with Data Pointer"**
  - Creates a new TwinCAT module class with a data pointer implementation

TwinCAT 3 – C++ project creation

BECKHOFF

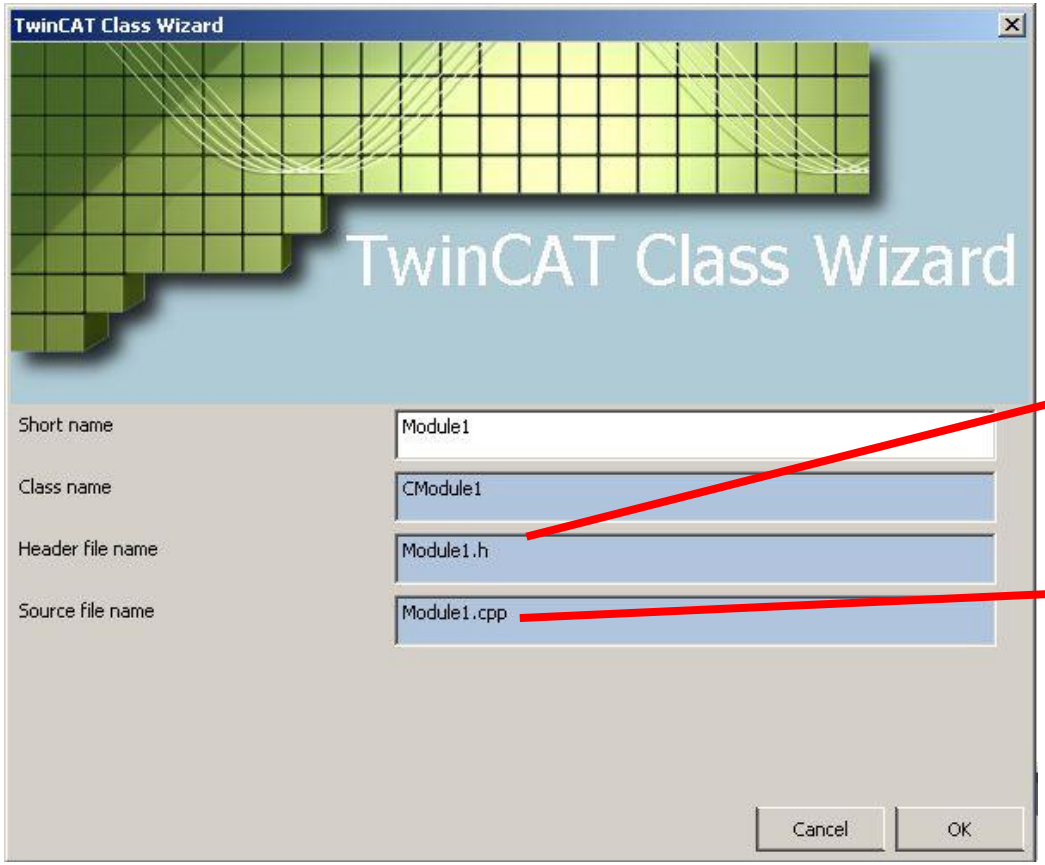
- In this case select "TwinCAT Module Class with Cyclic I/O”
- A name is not required and can not be entered here

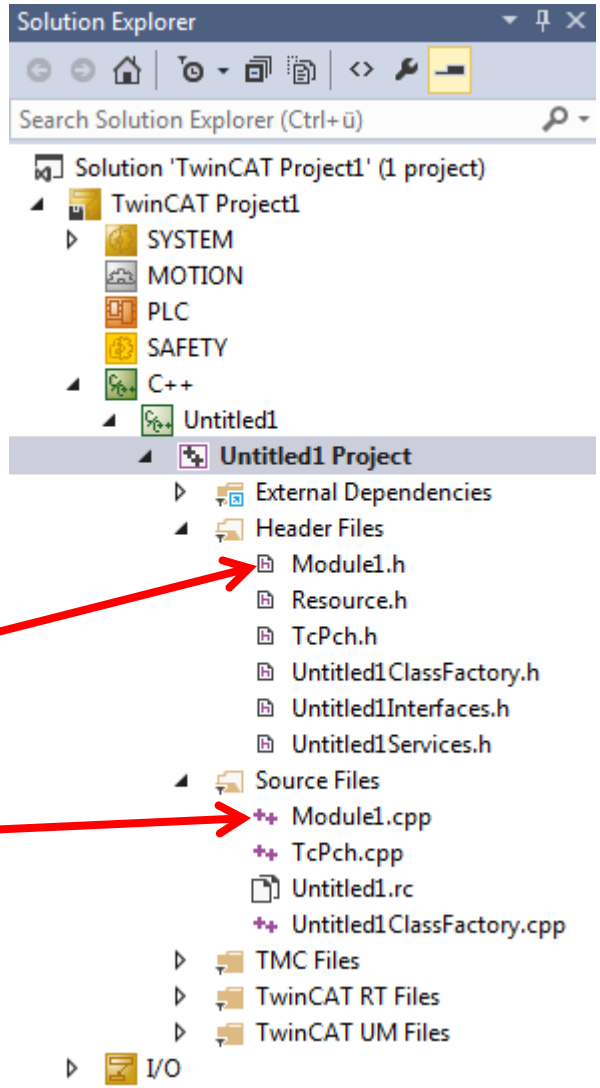


TwinCAT 3 – C++ project creation

BECKHOFF

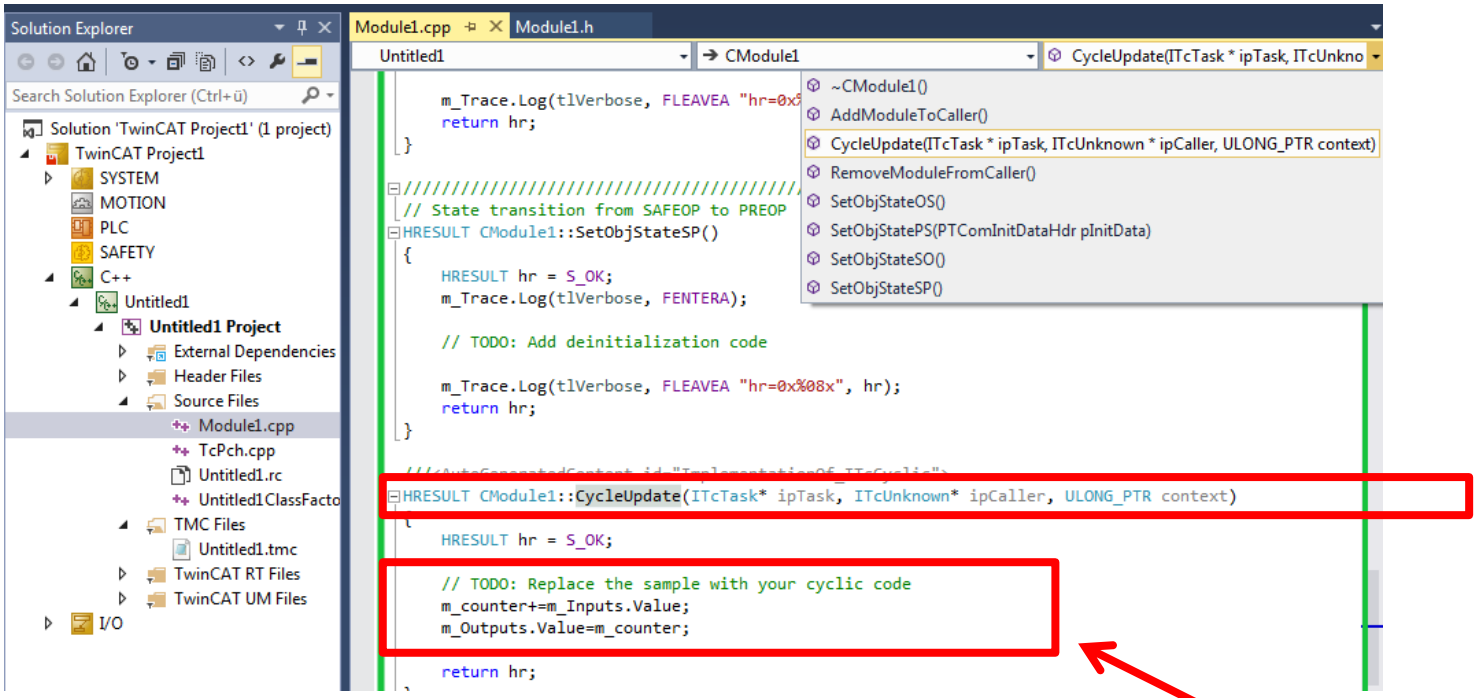
- In the upcoming "TwinCAT Class Wizard" enter a unique name or continue with proposal “Module1"





TwinCAT3 – Implement a TwinCAT C++ module

- Open automatically generated cpp-file
- The methode <MyClass>::CycleUpdate() is cyclically called - this is the place to implement the cyclic logic

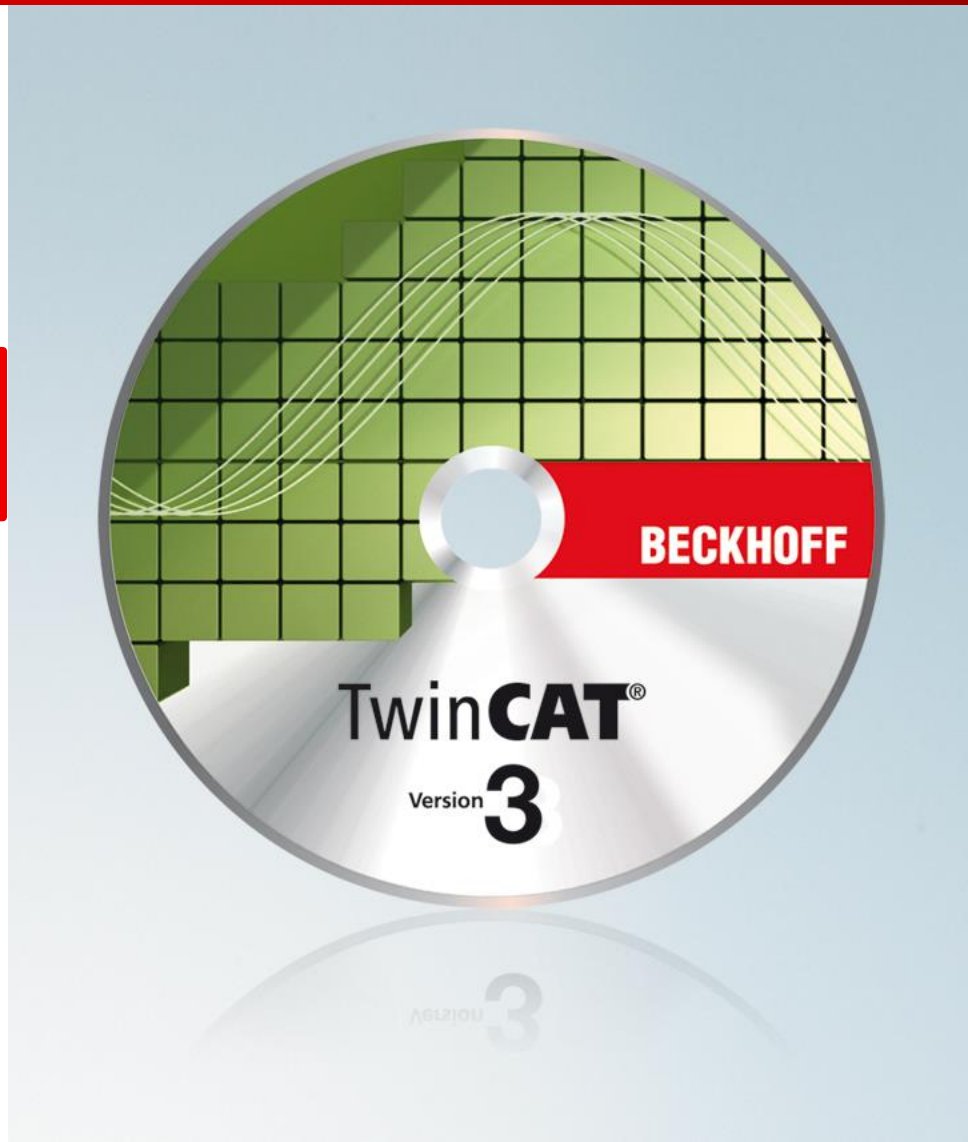


Modifying variables in the output area

Content

BECKHOFF

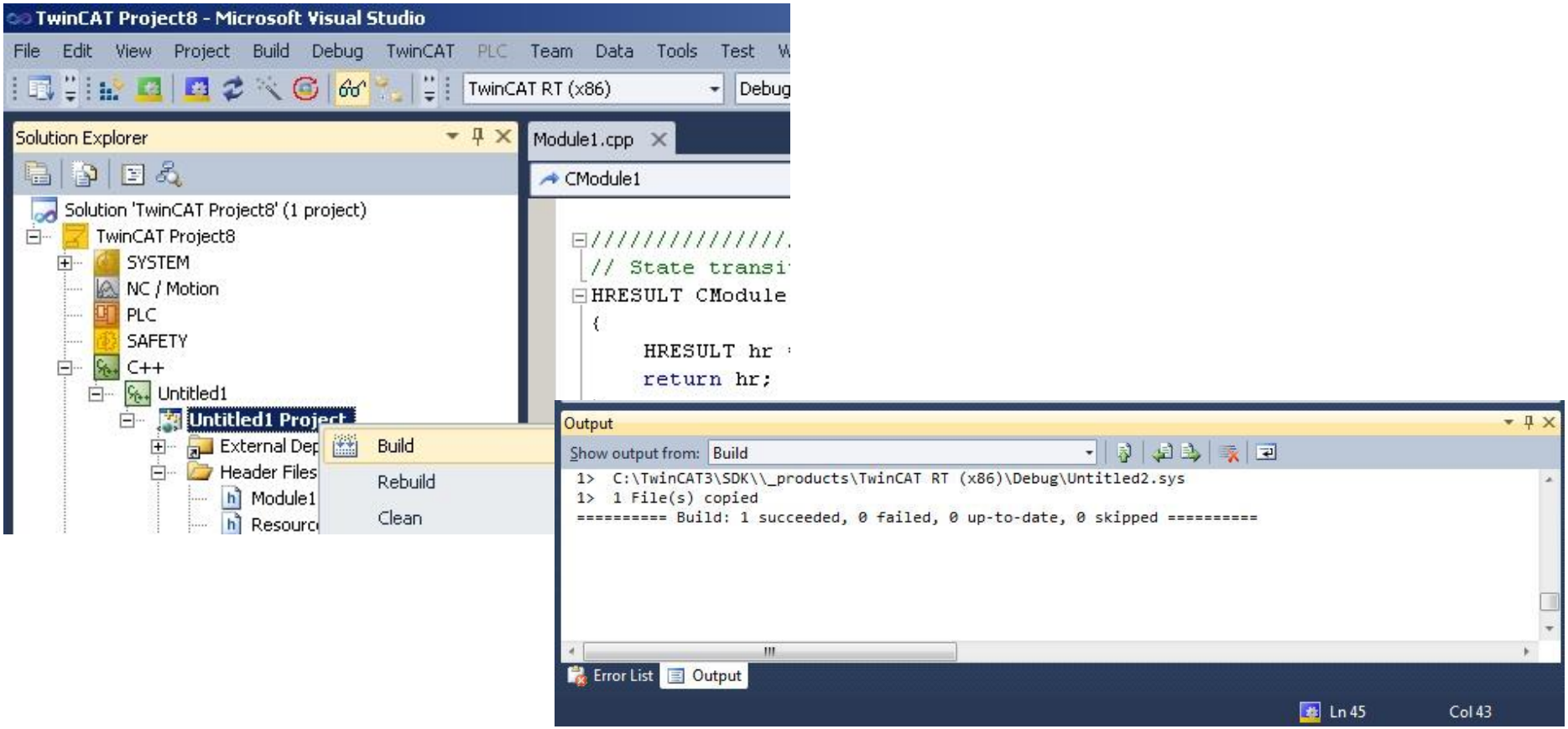
- Create a TwinCAT 3 project
- Create a TwinCAT 3 C++ project
- Implement TwinCAT 3 C++ project
- Create an instance
- Create a TwinCAT 3 task
- Debug the TwinCAT 3 C++ module



TwinCAT 3 – Compile a C++ project

BECKHOFF

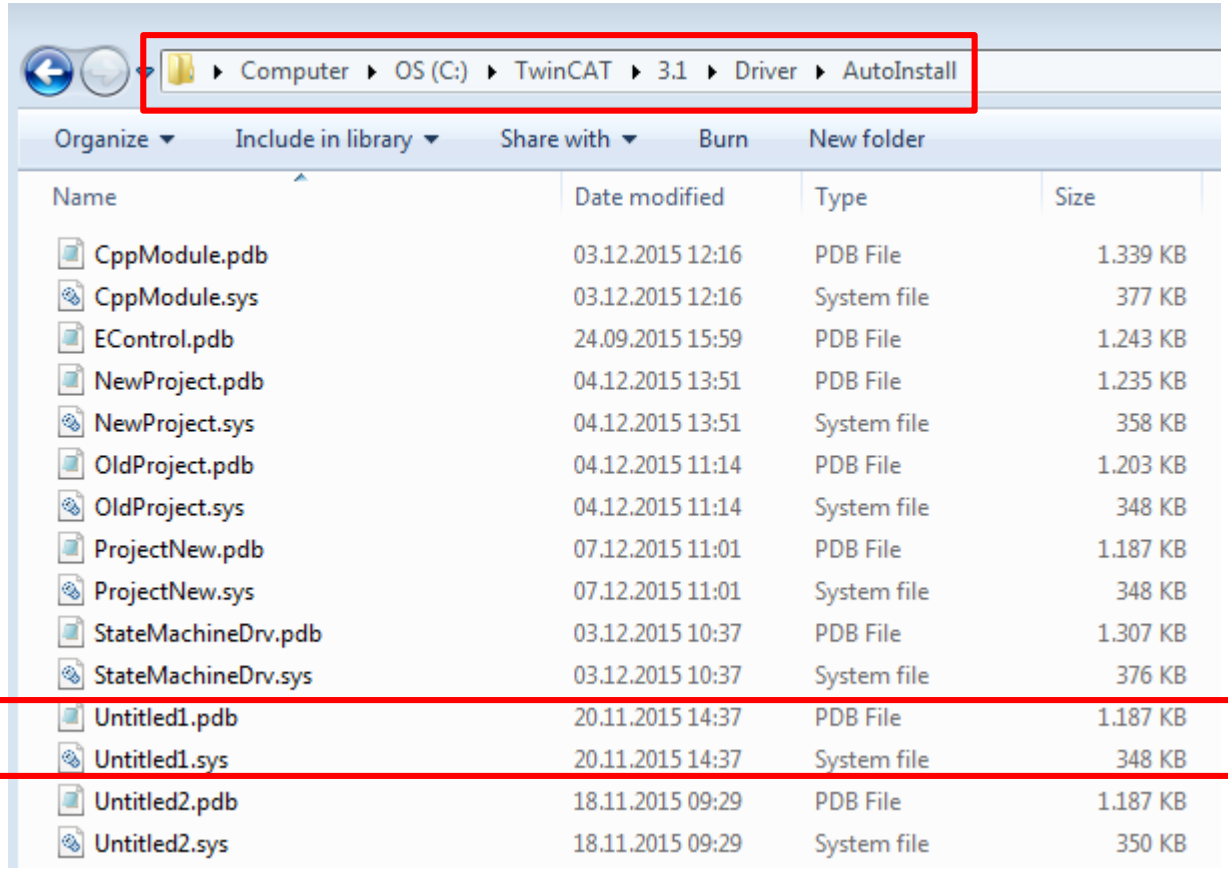
- Right click in the TwinCAT3 C++ project and select "Build" or "Rebuild“
- Check the result in the Output message window!



TwinCAT 3 – Compile a C++ project

BECKHOFF

- In case the build was successfully the new TwinCAT C++ module (driver, *.sys) is automatically deployed to the folder “C:\TwinCAT\3.x\Driver\AutoInstall"



Content

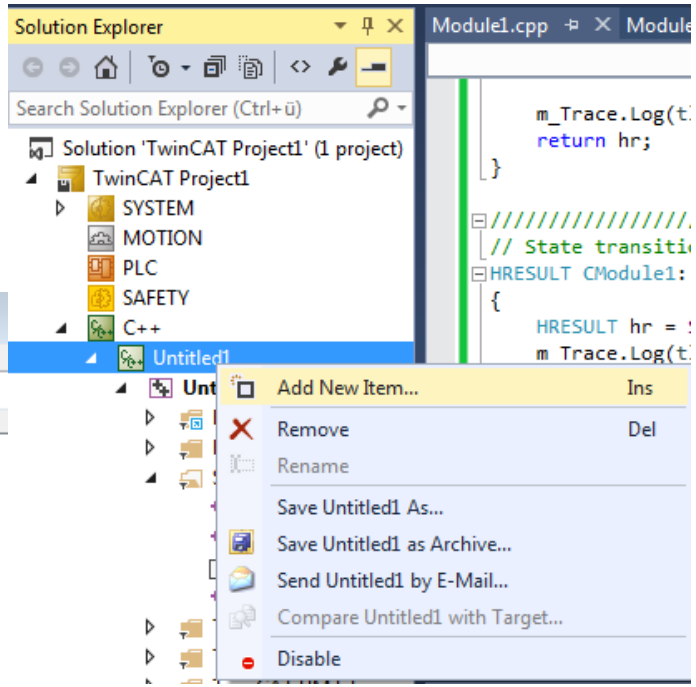
BECKHOFF

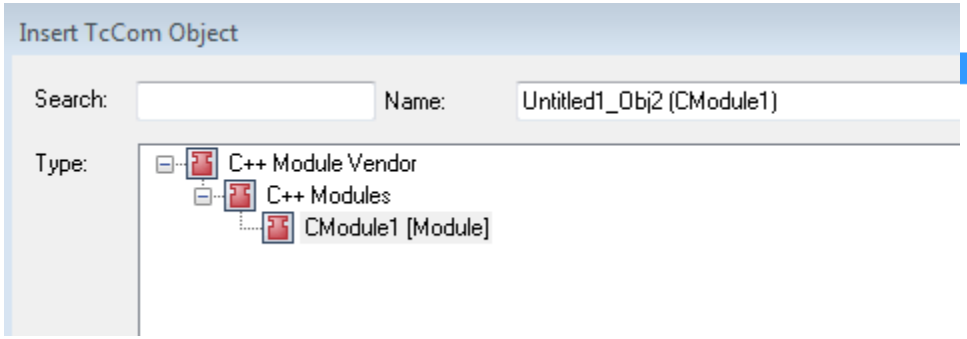
- Create a TwinCAT 3 project
- Create a TwinCAT 3 C++ project
- Implement TwinCAT 3 C++ project
- Create an instance
- Create a TwinCAT 3 task
- Debug the TwinCAT 3 C++ module

TwinCAT 3 – Create a C++ module instance

BECKHOFF

- After building the TwinCAT C++ project open the "C++ - Configuration" node
- Right click "C++ - project" and select "Add New Item..."
- All existing C++ modules are listed



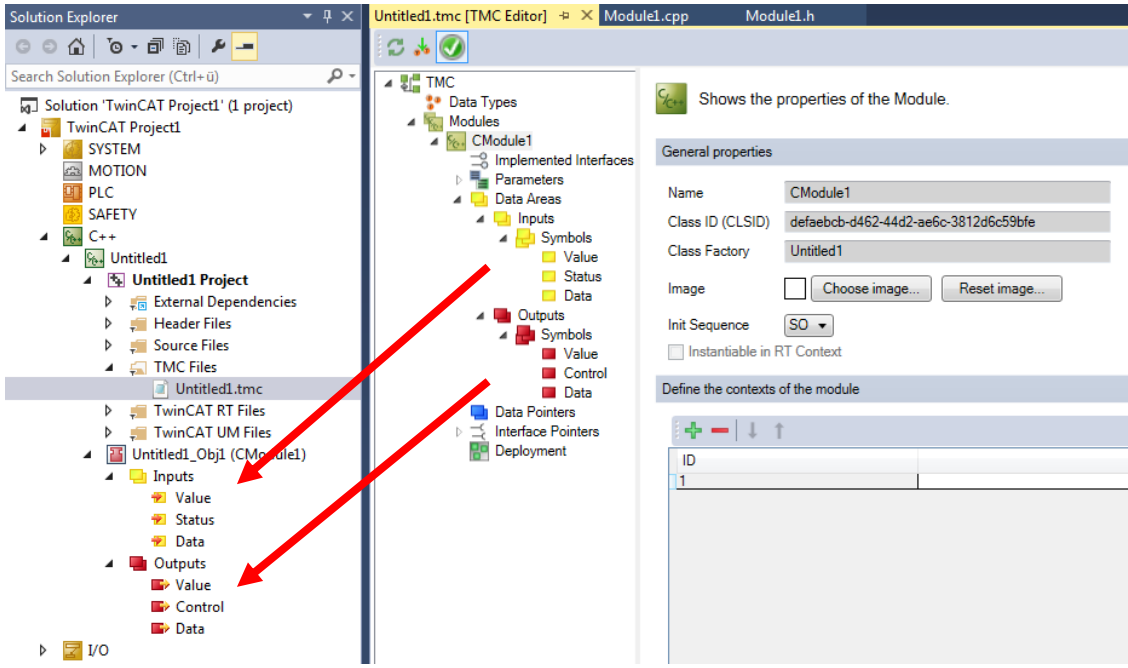


- Select on C++ module, optional enter an instance name and continue with "OK"

TwinCAT 3 – Create a C++ module instance

BECKHOFF

- The module provides a simple I/O interface with defined variables:
  - Inputs-Dataarea: Value, Status, Data
  - Outputs-Dataarea: Value, Control, Data



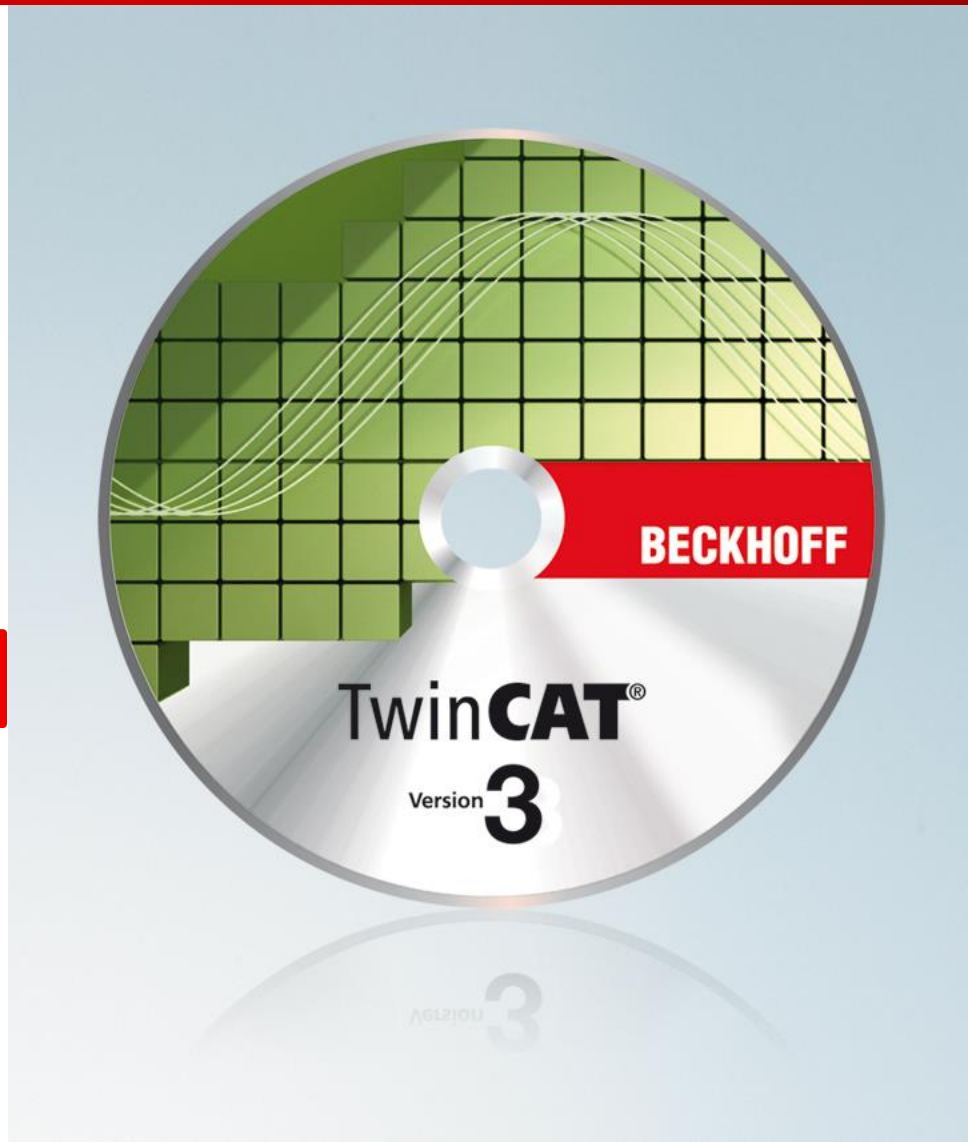
- The description is corresponding to each other at two places:
  - "<Classname>Services.h”
  - "TwinCAT Module Configuration" (TMC-)File

TMC-Files can be modified with the TMC-editor

Content

BECKHOFF

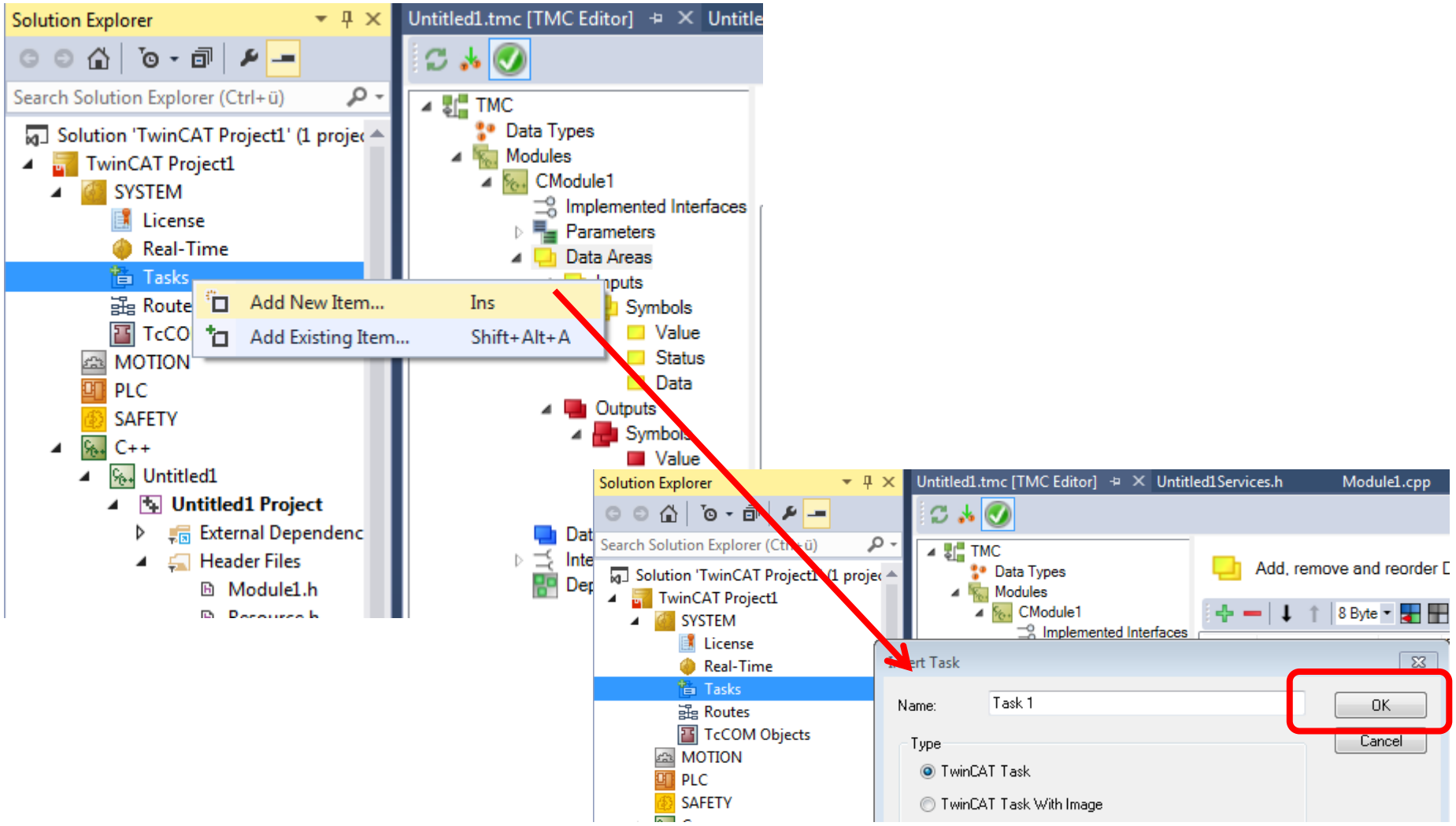
- Create a TwinCAT 3 project
- Create a TwinCAT 3 C++ project
- Implement TwinCAT 3 C++ project
- Create an instance
- Create a TwinCAT 3 task
- Debug the TwinCAT 3 C++ module



TwinCAT 3 – Create a task

BECKHOFF

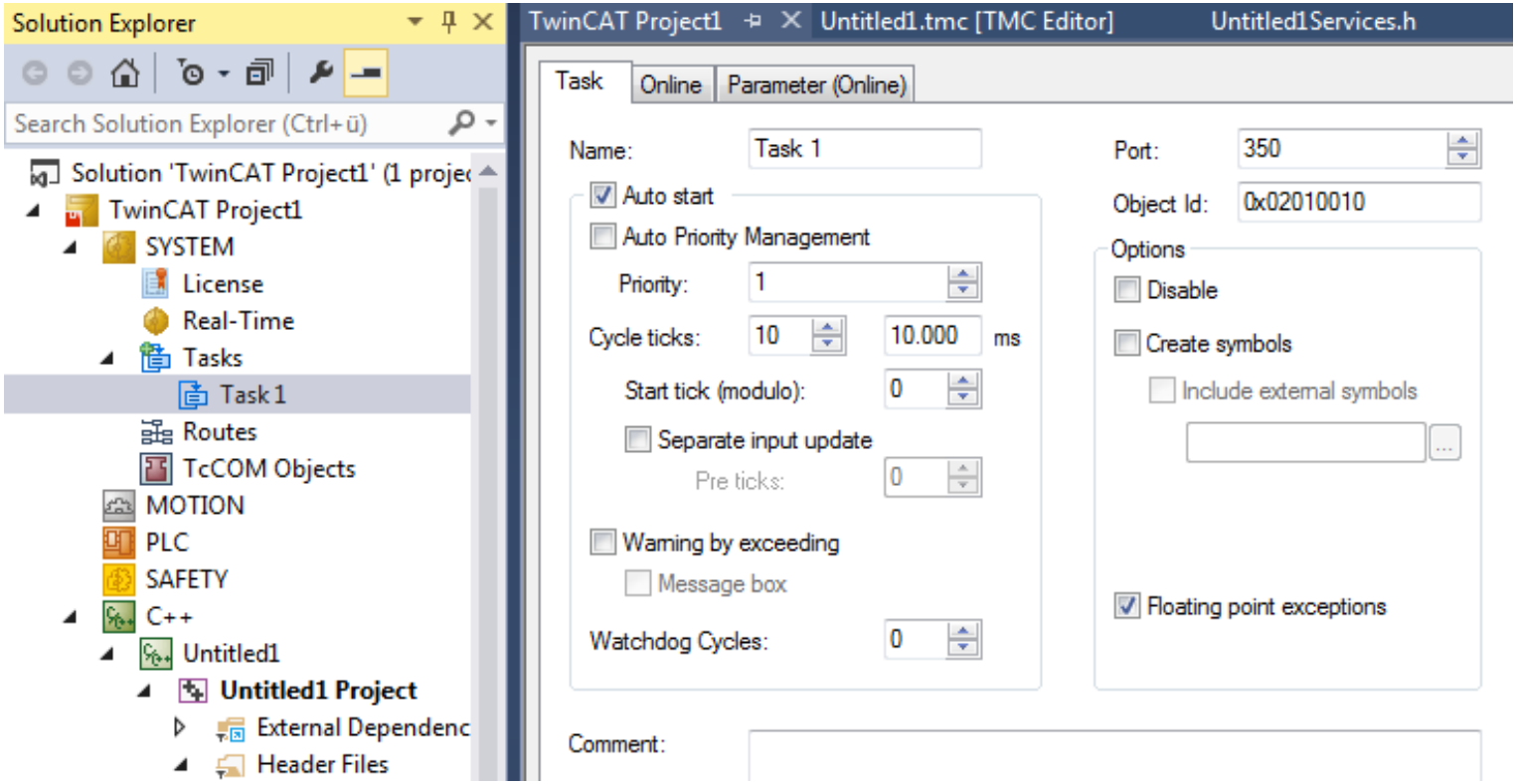
- Open "System", right click "Tasks" and select "Add New Item..."



TwinCAT 3 – Create a task

BECKHOFF

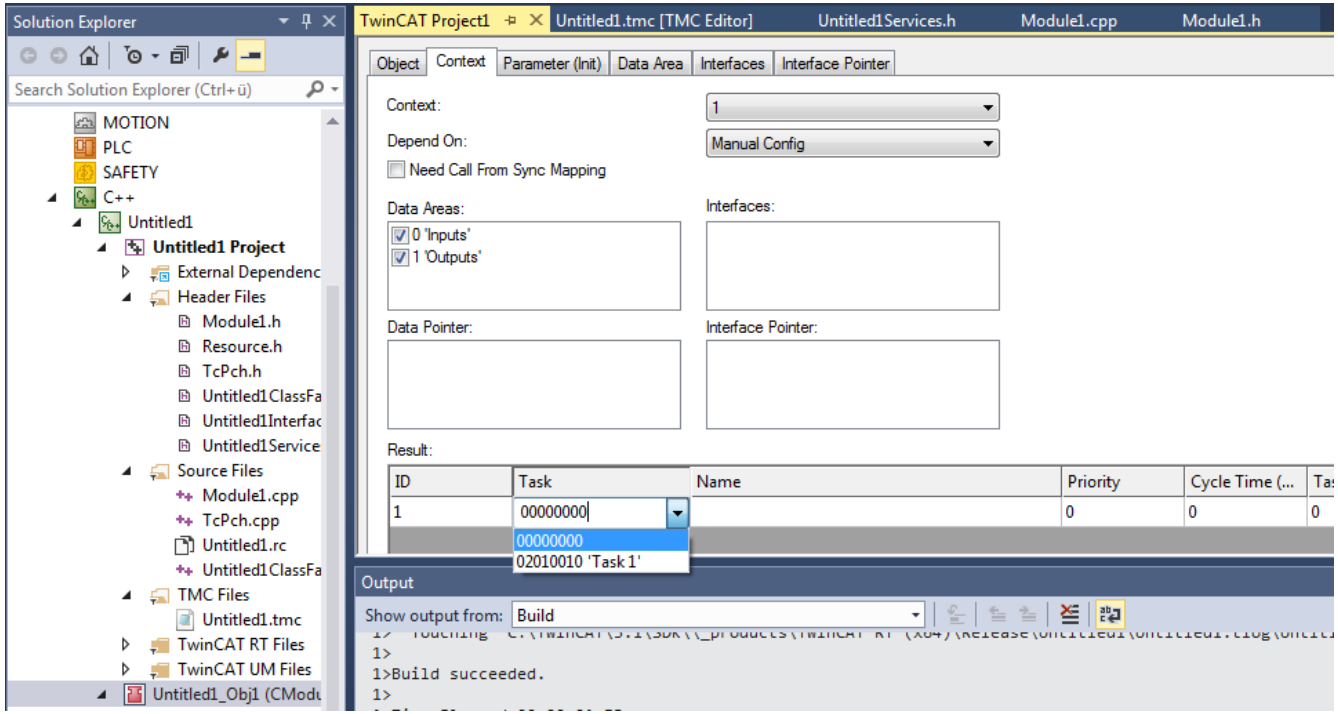
- Configure task, ‘Auto start’ is activated by default



TwinCAT 3 – Assign C++ module instance to the Task

BECKHOFF

- Select the C++ module instance in the solution tree
- On the right working area select tab "Context"
- Select the Task for the context.



Content

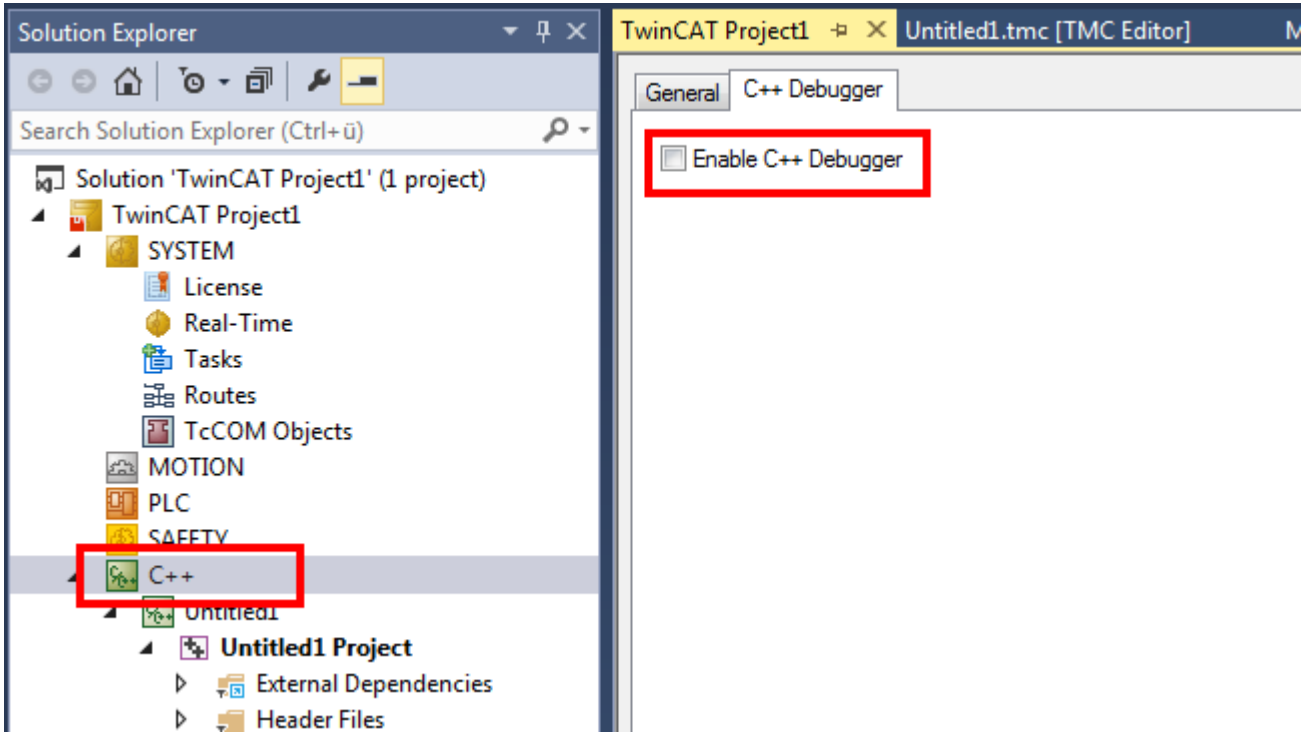
BECKHOFF

- Create a TwinCAT 3 project
- Create a TwinCAT 3 C++ project
- Implement TwinCAT 3 C++ project
- Create an instance
- Create a TwinCAT 3 task
- Debug the TwinCAT 3 C++ module

TwinCAT 3 – Debug the C++ module

BECKHOFF

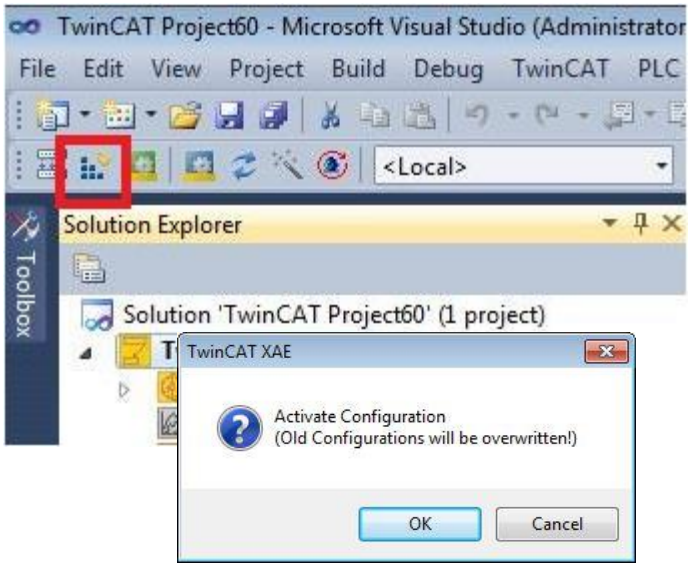
- The C++ debugger needs to be enabled
  - Navigate to "System -> Real-Time" and select tab "C++ Debugger“
  - activate the option "Enable C++ Debugger"

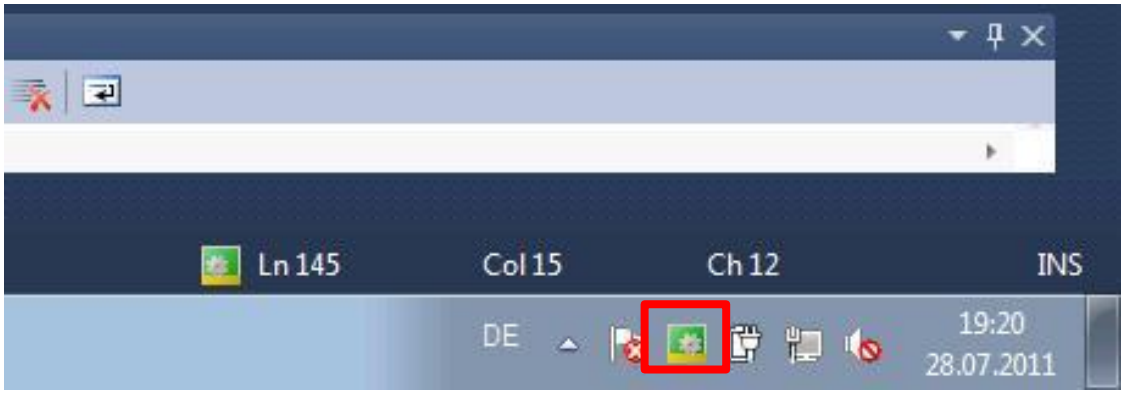


TwinCAT 3 – Start & Debug the C++ module

BECKHOFF

- Activate configuration and switch to Run-Mode

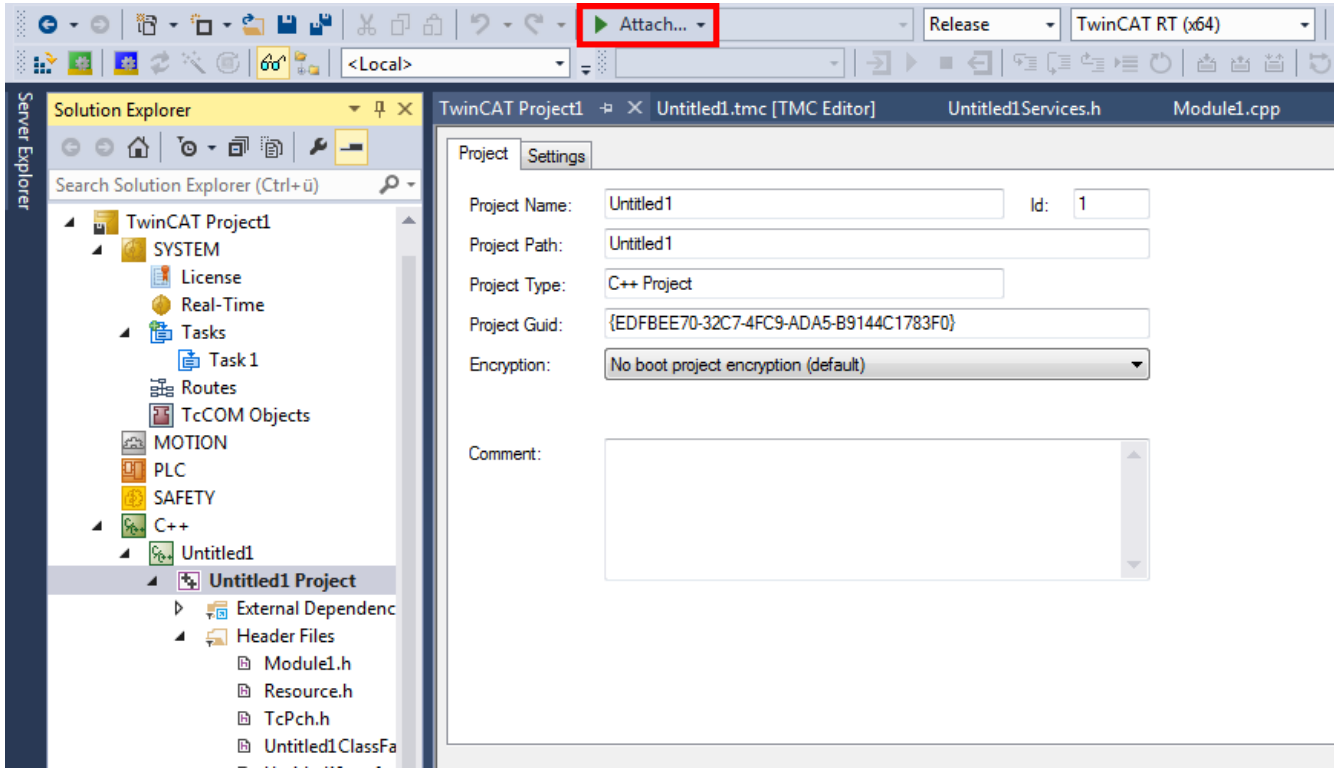




TwinCAT 3 – Debug the C++ module

BECKHOFF

- Attach to C++ module instance



TwinCAT 3 – working with breakpoints

BECKHOFF

TwinCAT3 – working with breakpoints

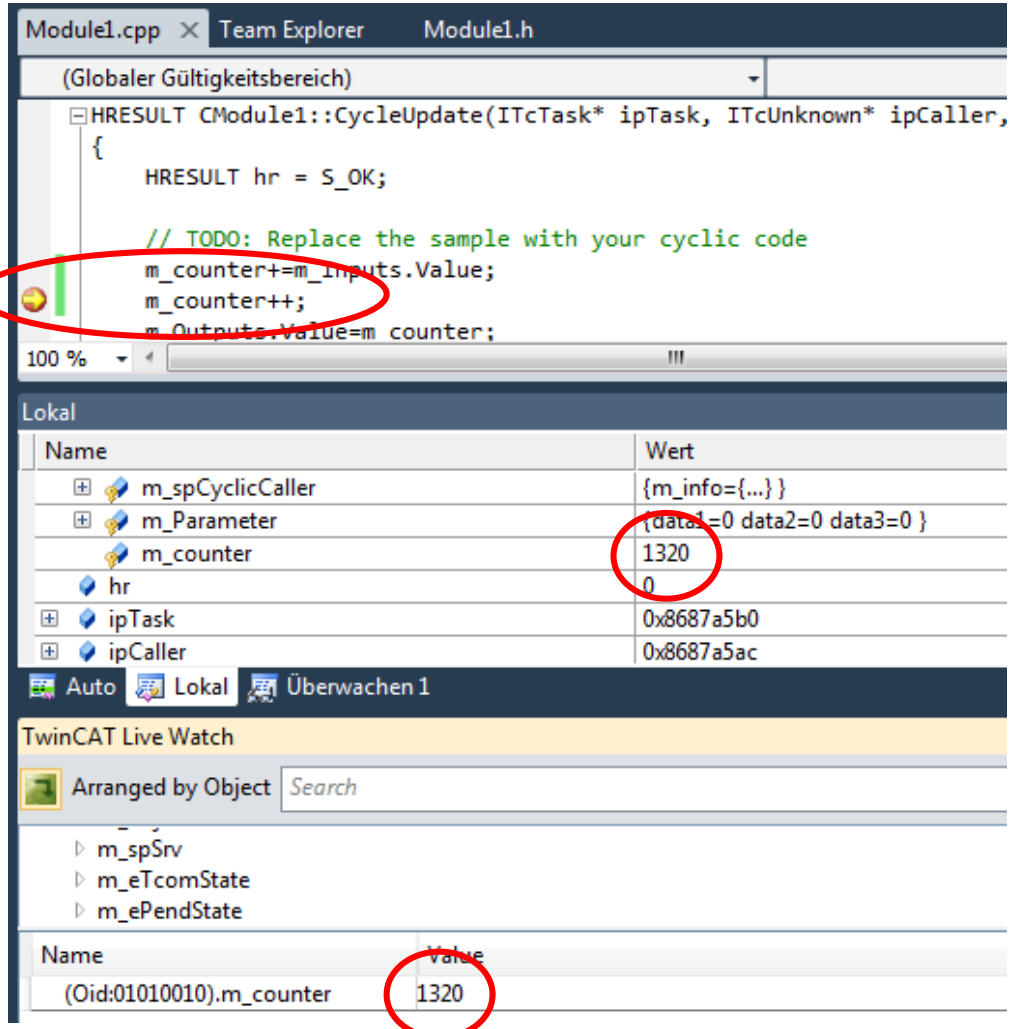
Controlled via keyboard:

F9 → toggle breakpoint

F10 → step over

F11 → step into

F5 → start debugging

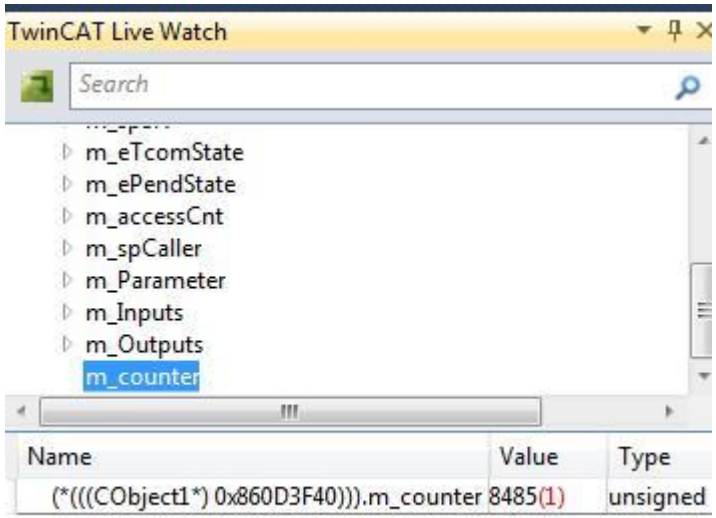


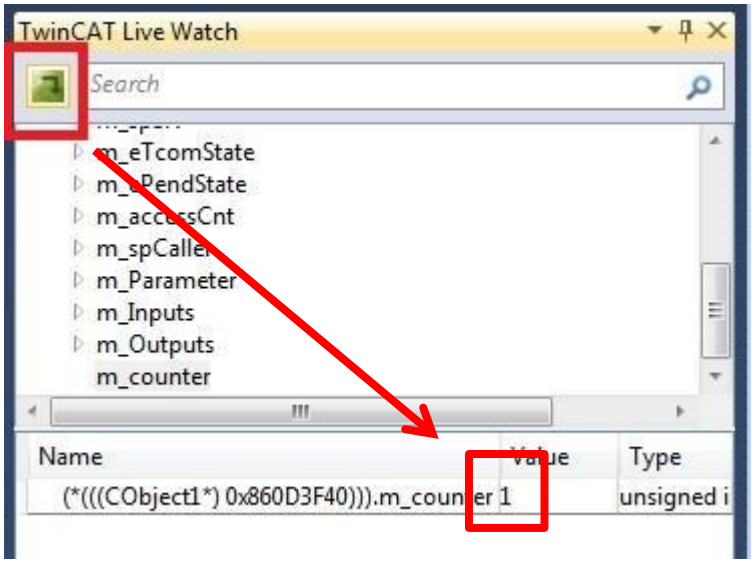
TwinCAT 3 – Debug the C++ module

BECKHOFF

At the TwinCAT Live Watch Window….

- Drag’n’drop variable to watch window
- See online values
- Prepare overwrite value (red brackets)
- Overwrite (green icon or right-click)





Overview

BECKHOFF

- TwinCAT C++
  - Quickstart
  - Coming from Usermode
  - Workflow
- TcCOM Modules
  - Concept
- Using TcCOM Interfaces
- TwinCAT C++ Development
- Documentation: Infosys
- Conclusion

Content

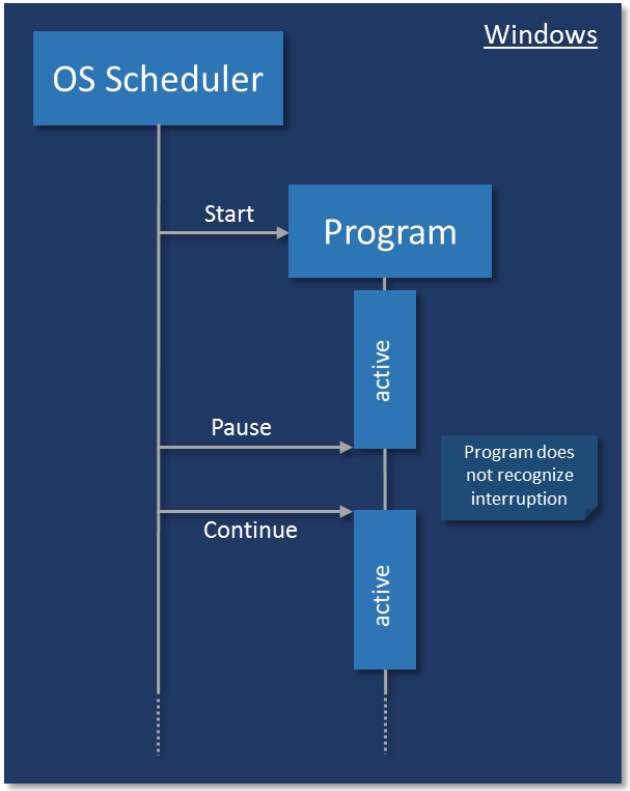
BECKHOFF

- Program Flow
- Concurrence
- UserMode vs. KernelMode

Program Flow

Usermode Programming:

- Coded program gets executed from OS
- Program will get interrupted and (normally) doesn't care about this interruption
- Program "waits" for OS and resources to become available.



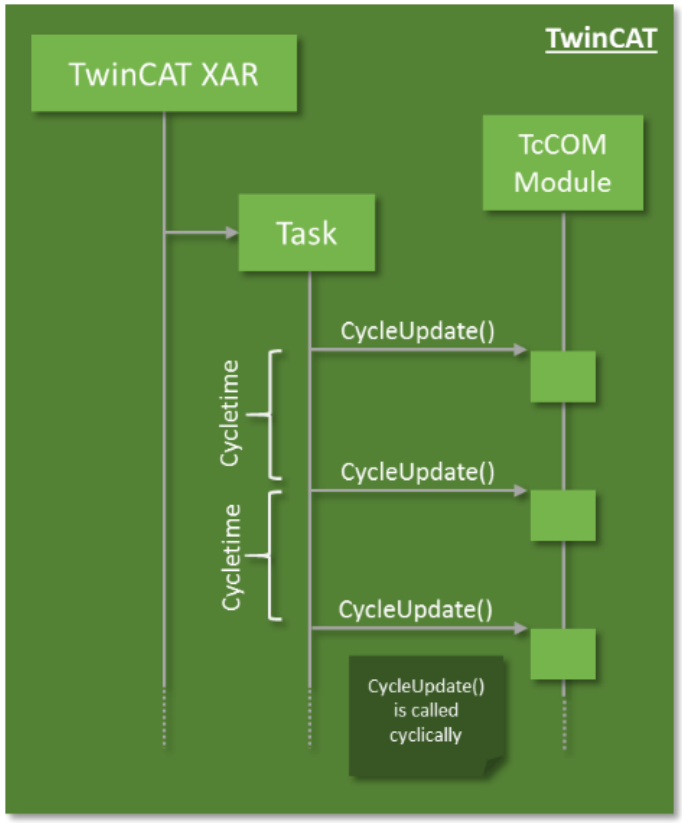
→ Usermode Programs are not Realtime-capabile

Program Flow

BECKHOFF

TwinCAT RT Programming:

- TwinCAT C++ Modules
  - provide a CycleUpdate() method
  - This gets called by TwinCAT as defined by task
  - …don't need to provide this if no cycle logic is implemented
- TwinCAT
  - Provides "Realtime-Tick"
  - Manages Resources



→ TwinCAT C++ Modules are Realtime-capable

Content

BECKHOFF

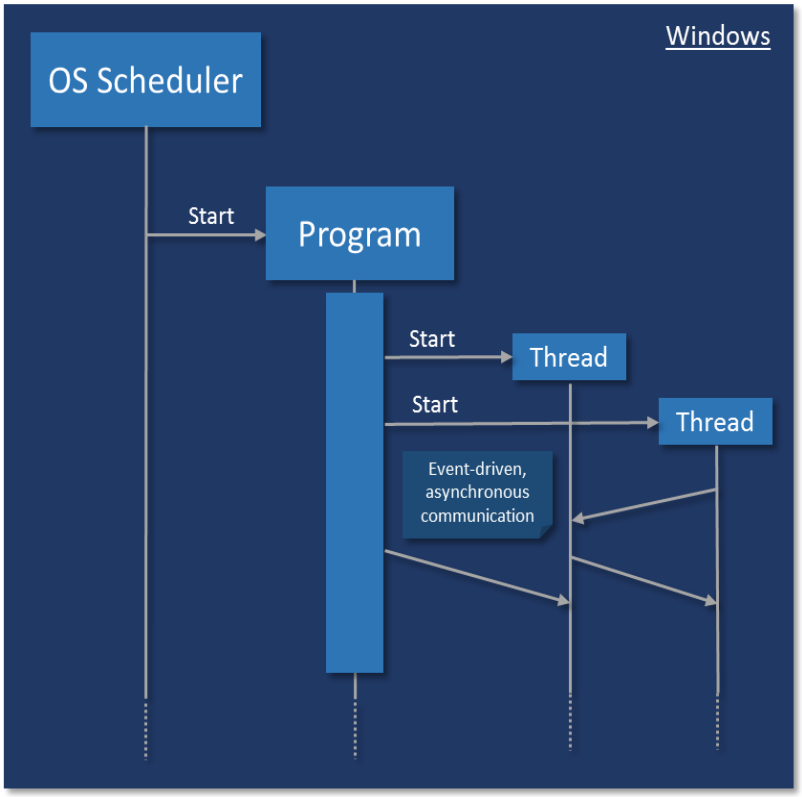
- Program Flow
- Concurrence
- UserMode vs. KernelMode

Concurrence

BECKHOFF

Usermode Programming:

- Program has control about “flow”: Starting/stopping threads
- This requires operations like allocation of resources, which is non-realtime compatible
- Operating system schedules all threads on all CPUs / cores
- Data exchange is event-driven and (often) blocking

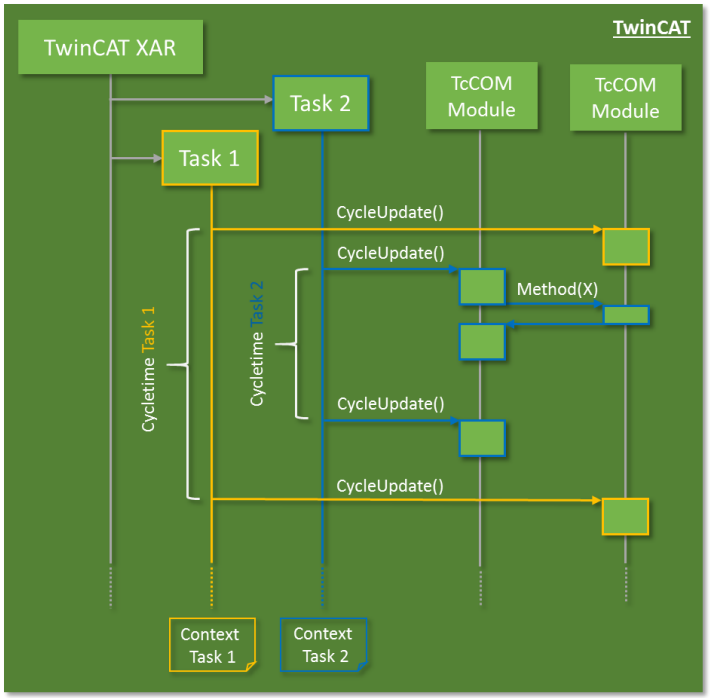


Concurrence

BECKHOFF

TwinCAT RT Programming:

- Tasks are calling modules cyclically
- Modules have a context
- Modules could communicate
  - Mapping, DataPointers, ADS and…
  - Methods
- Calling Methods (of Interfaces) on modules of other tasks requires to deal with the data consistency (for CriticalSections)



Content

BECKHOFF

- Program Flow
- Concurrence
- UserMode vs. KernelMode

UserMode vs. KernelMode

BECKHOFF

UserMode	TC-C++ in KernelMode
Interruption by OS	Realtime-capabile
CRT	TcCRT (RtlR0.h)
DLL	SYS
Paged Memory of OS	Nonpaged + TC managed Memory
STL / MFC	WDK / TcSTL
Program controls Workflow	Cyclic calling by TC

Overview

BECKHOFF

- TwinCAT C++
  - Quickstart
  - Coming from Usermode
  - Workflow
- TcCOM Modules
  - Concept
- Using TcCOM Interfaces
- TwinCAT C++ Development
- Documentation: Infosys
- Conclusion

Workflow

1. Type declarations
2. Programming
3. Instantiating Modules
4. Mapping of Variables
5. Building
6. Optional: Publishing
7. Optional: Signing
8. Activation

Overview

BECKHOFF

- TwinCAT C++
  - Quickstart
  - Coming from Usermode
  - Workflow
- TcCOM Modules
  - Concept
- Using TcCOM Interfaces
- TwinCAT C++ Development
- Documentation: Infosys
- Conclusion

Content

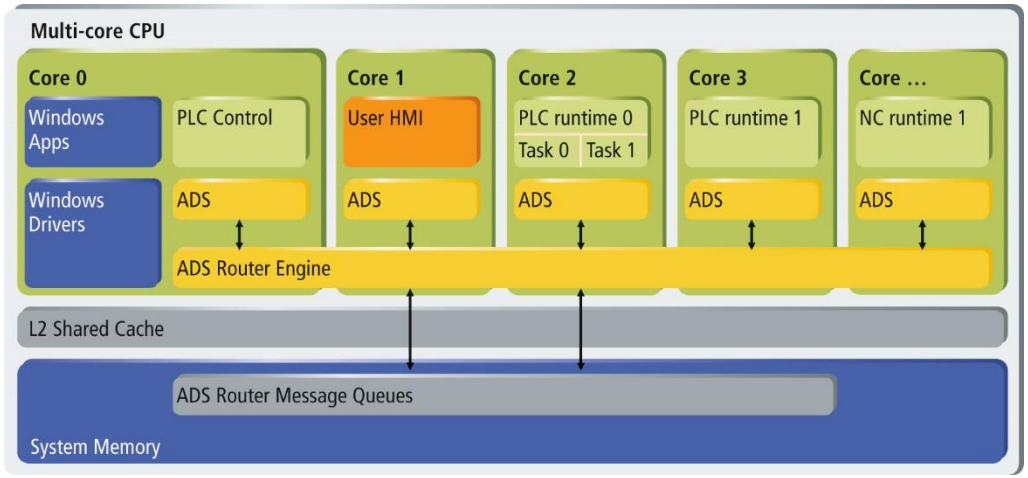
BECKHOFF

Modularization

BECKHOFF

- Standard units, methods:
  - Screws, bolts, tools, …
  - Cables, plugs, …

  →Reusability for Software



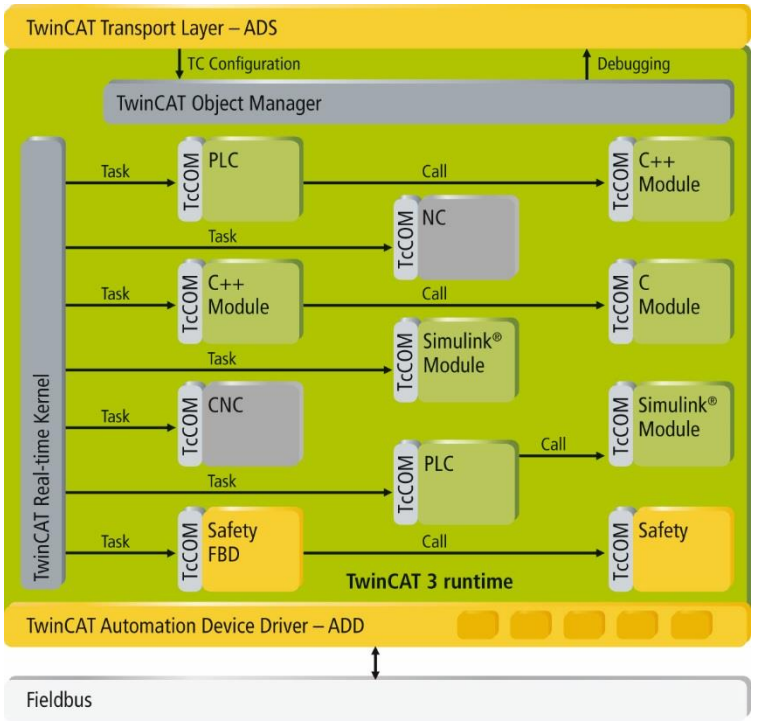
- Central issue, but complex:
  - How to harmonize?
  - How to solve overlapping issues?
  - How to create standardized interfaces?

Module overview

BECKHOFF

…to master the complexity of modern machines:

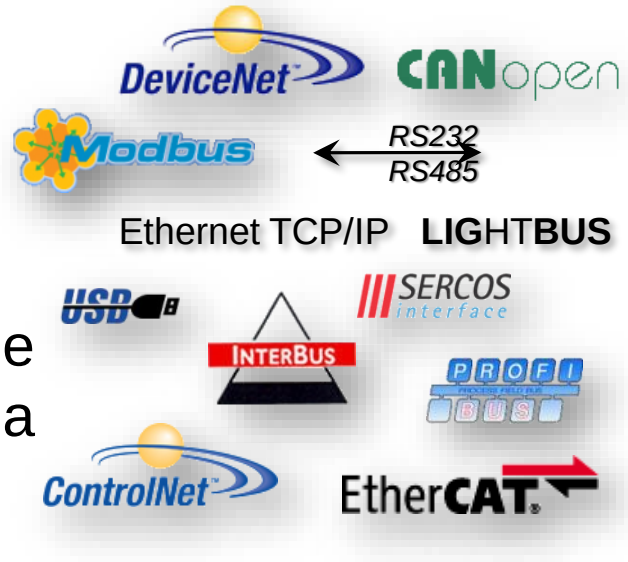
- software is independent of the hardware
- Individual functions (eg. assemblies or machine units) regarded as modules
- interactions between the controllers via communication networks (fieldbuses or Ethernet)
- reusability of this control software

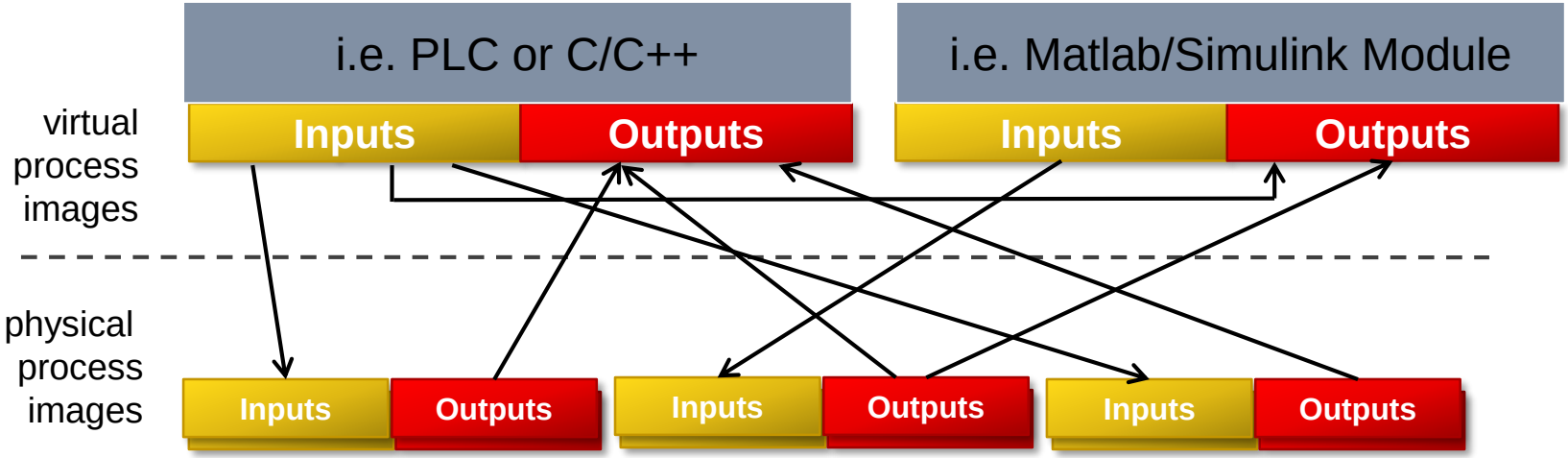


TwinCAT I/O – mapping of process images

BECKHOFF

- Abstraction as a main philosophy
- Open for all common field busses
- Support of all PC hardware interfaces
- Easy commissioning and diagnosis
- Assignment of logical/physically process image
  → Changes of the bus system do not require a change of the PLC code





Content

BECKHOFF

- Overview
- COM
- TcCOM
- Module Handling

COM – Component Object Model

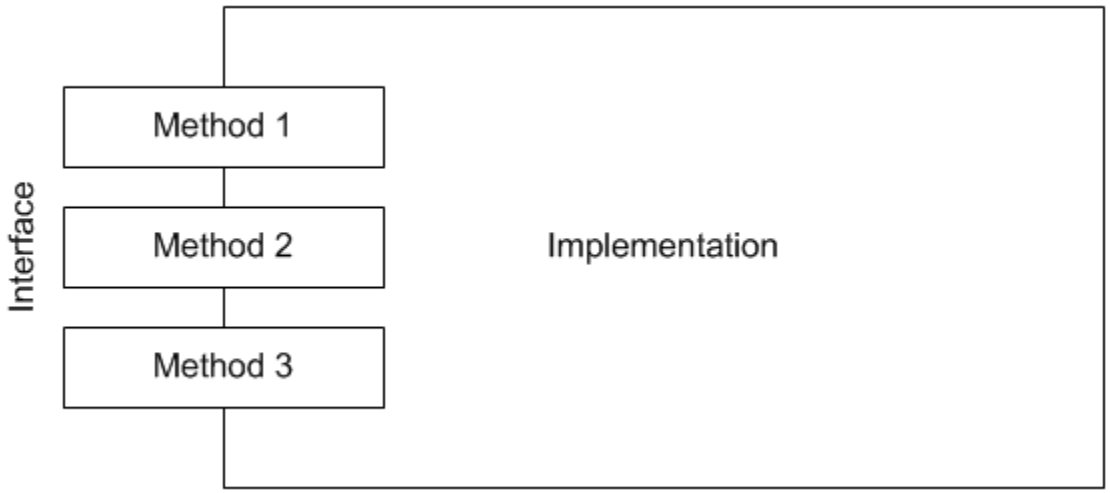
- Developed technology to design software components.

- Microsoft introduced this technology 1993.
- Integral part of Windows since Windows 3.1.

- Has influneced several other developments:
  - Mozilla products like the Firefox Browser.
  - SUN products like OpenOffice.
  - Philips and Samsung embedded products.
  - Symbian OS for mobile systems.

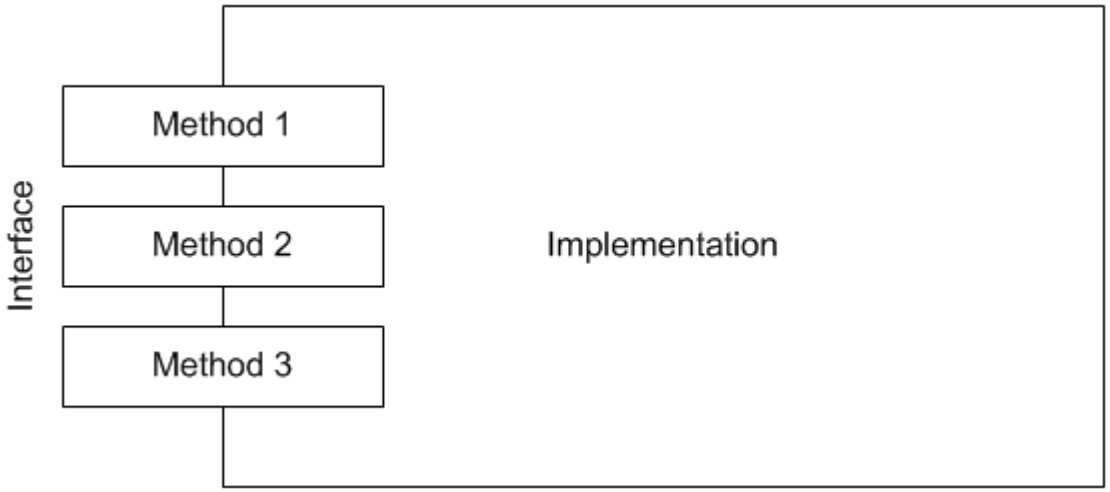
COM – Requirements for components.

- A software component has to solve an issue.
- A software component consists of two parts.
  - Implementation
    - Gets input data, proceeds something and produces output data.
  - Interface
    - Provides methods.



COM – Requirements for components

- Interface of the component is known.
  - Usage of the component by the Interface.
  - Component becomes replaceable.
- Implementation of the component is unknown.
  - Component is a Black-Box.
  - Implementation is able to change.



COM means Modularisation

BECKHOFF

- Modularisation to solve the trouble.
  - Individual functions, assemblies or units are regarded as modules.
  - Modules which are independent as possible.
  - Embedded into the overall system by uniform interfaces.
- Master the complexity of modern machines.
- Reduce the necessary engineering expenditure.

- Modular software on one central controller.
  - Individual modules within one runtime.
  - Lower costs.
  - All information are accessable.

Content

BECKHOFF

- Overview
- COM
- TcCOM
- Module Handling

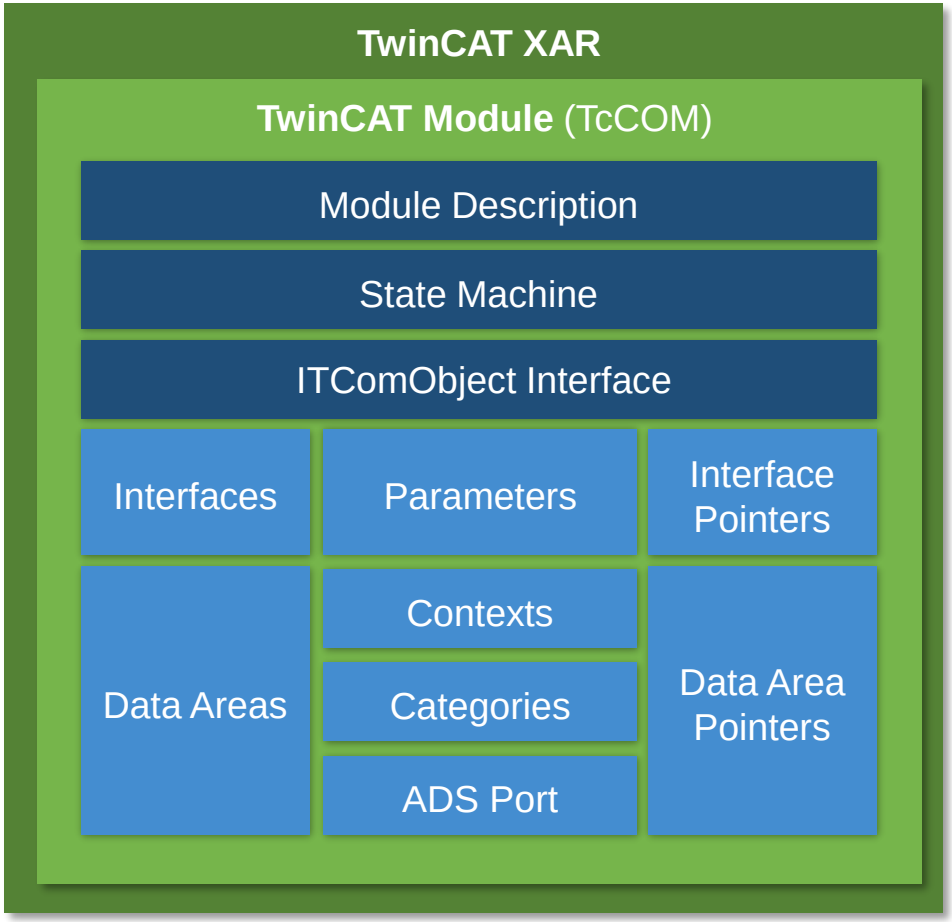
TcCOM Modules

- consists of a series of formally defined properties
- all of the tasks of an automation system can be and are packed into modules → capsulation
- Module consists of
  - module description file (XML format) → characteristics
  - Binary files → behaviour
- Central Module Manager handles all modules in TwinCAT → reachable and parameterisable.
- Modules can be compiled independently and designed in a very simple way
  …but they can also be very complex, internally.

TcCOM – TwinCAT Modules

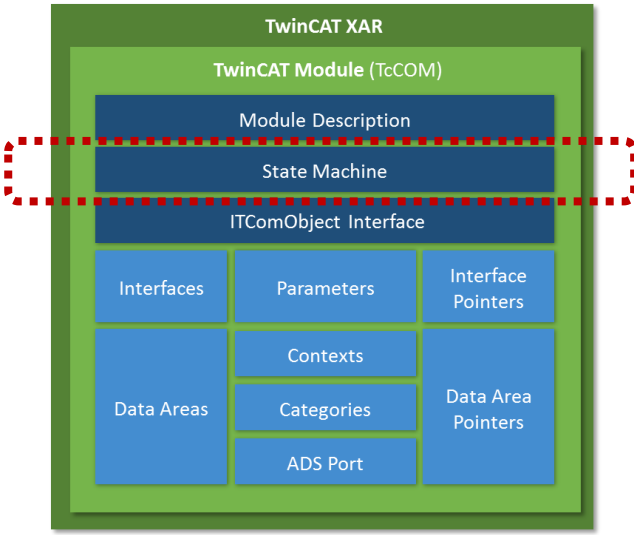
BECKHOFF

- TwinCAT Module
  - Predescribed Properties (dark blue)
    - Module Description
    - State Machine
    - ITComObject Interface
  - Optional Properties (blue)
    - Parameter
    - Interfaces
    - Interface Pointer
    - Data Areas
    - Data Area Pointer
    - Contexts
    - Categories
    - ADS



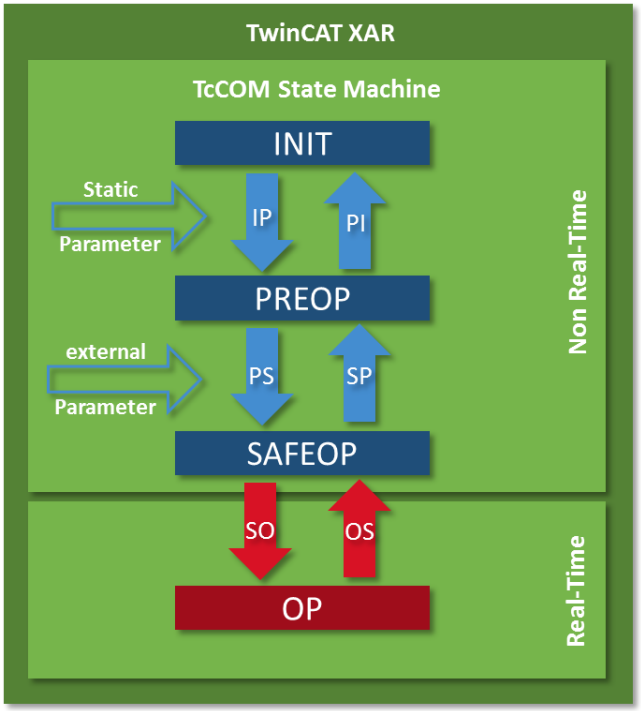
State machine

BECKHOFF



Prescribed: State machine

- describes the (de-)initialization state of a module



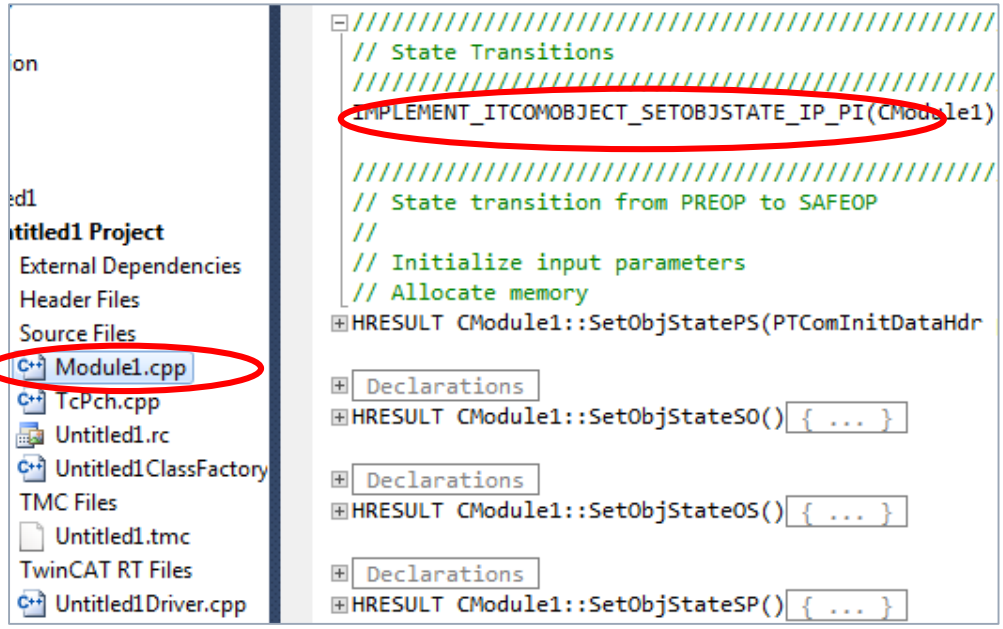
- Not the same as EtherCAT states

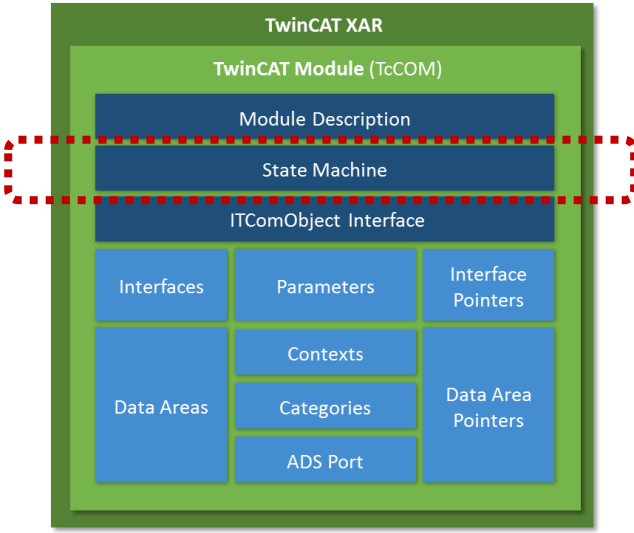
State machine

BECKHOFF

Prescribed: State Machine defines the sequence of the module generation, parameterizing and the creation of the connection to the other modules

- Accessible via TwinCAT C++





(C++ code is autogenerated;could be modified)

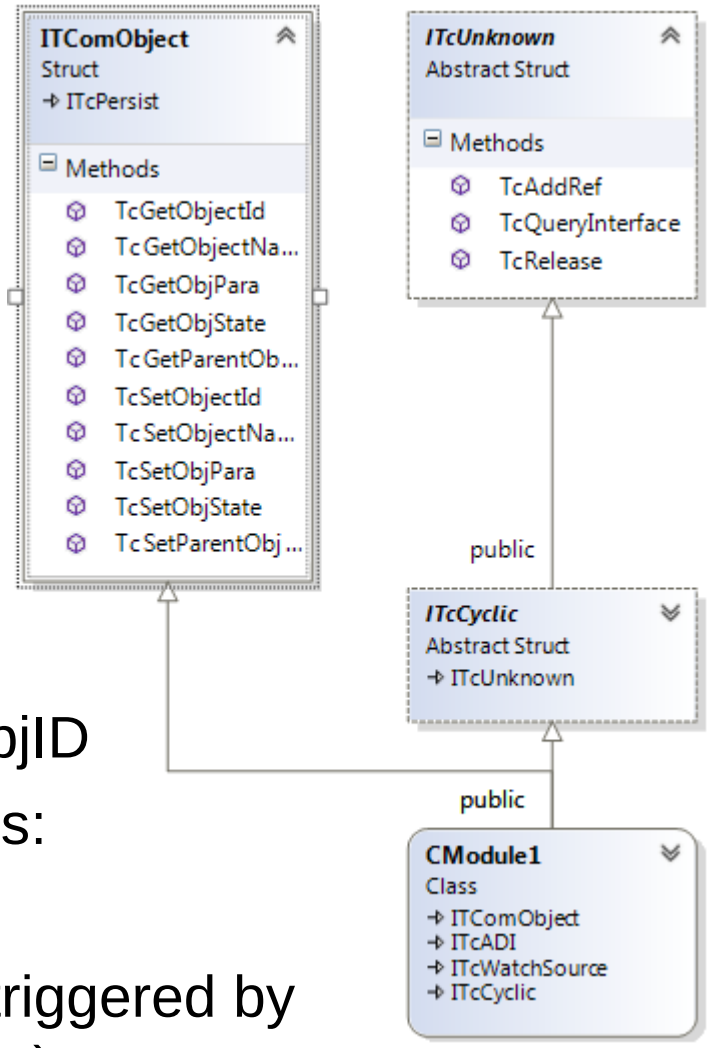
Content

BECKHOFF

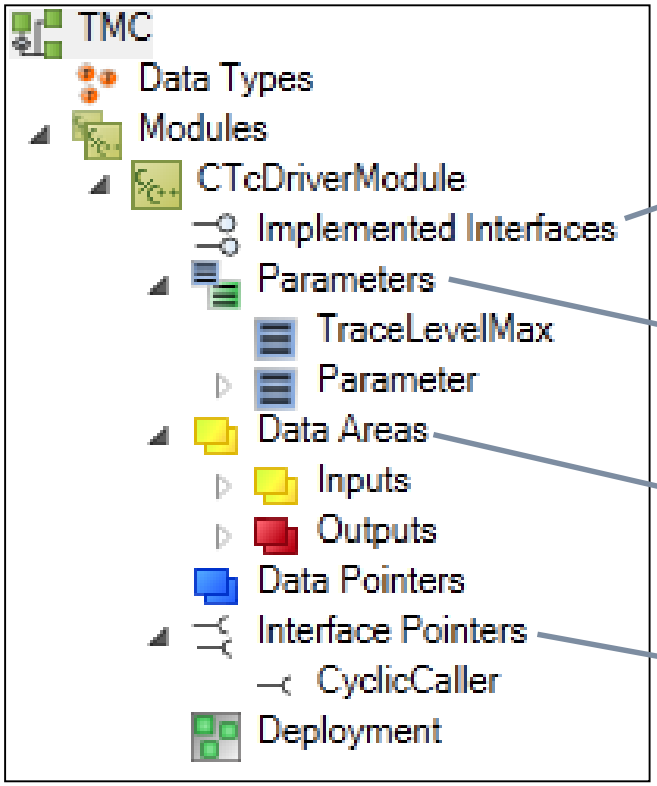
- Overview
- COM
- TcCOM
- Module Handling

TcCOM - Interfaces

- TcCOM modules use „basic“ interfaces for interaction with TwinCAT runtime
  - ITcUnknown
    - Reference counting
    - TcQueryInterface
  - ITComObject
    - Statemachine
    - Parameters
    - Name / ObjID / ClassID / ParentObjID
- One of the most common used interfaces:
  - ITcCyclic
    - Provides CycleUpdate(), which is triggered by CyclicCaller (implemented by Tasks)



- Editor of the TwinCAT Module Class (*.tmc)
  - Module is described in the „class description file“
  - Provides a Code Generator (C++) to implement the module



Name	Interface ID
ITComObject	{00000012-0000-0000-E000-000000000064}
ITcCyclic	{03000010-0000-0000-E000-000000000064}
ITcADI	{03000012-0000-0000-E000-000000000064}
ITcWatchSource	{03000018-0000-0000-E000-000000000064}

Name	Parameter ID	Specification	Size [Bits]	Is Aligned	Context
TraceLevelMax	#x03002103	Alias	32	Yes	1
Parameter	#x00000001	Struct	128	Yes	1

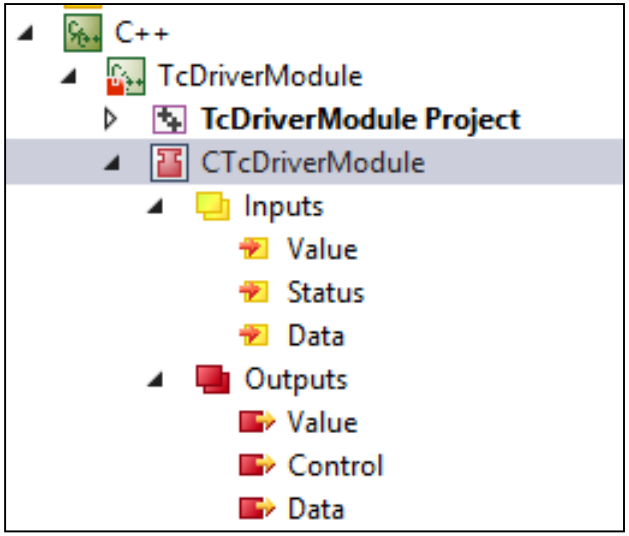
Number	Area Type	Name	Size [Bytes]	Is Aligned	Context
0	Input-Destination	Inputs	12	Yes	1
1	Output-Source	Outputs	12	Yes	1

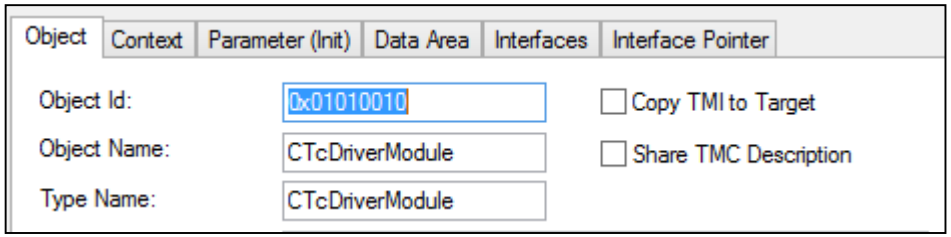
Name	Parameter ID	Type	Context
CyclicCaller	#x03002060		

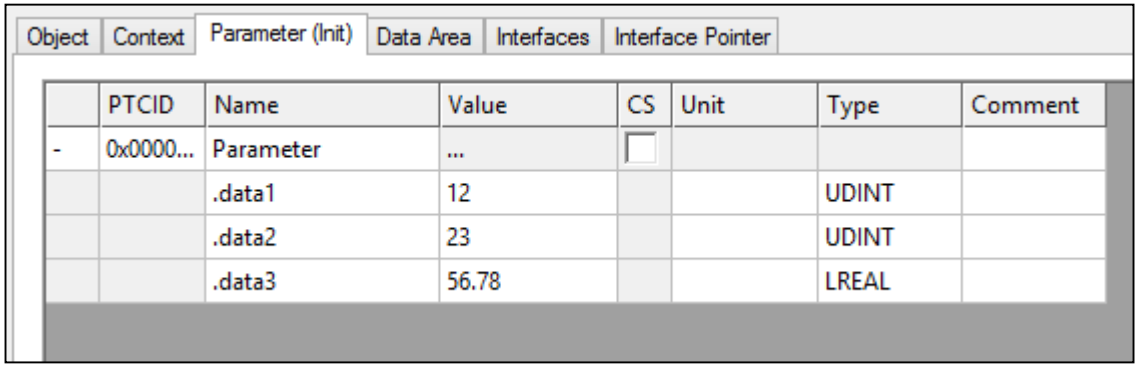
TMI Editor

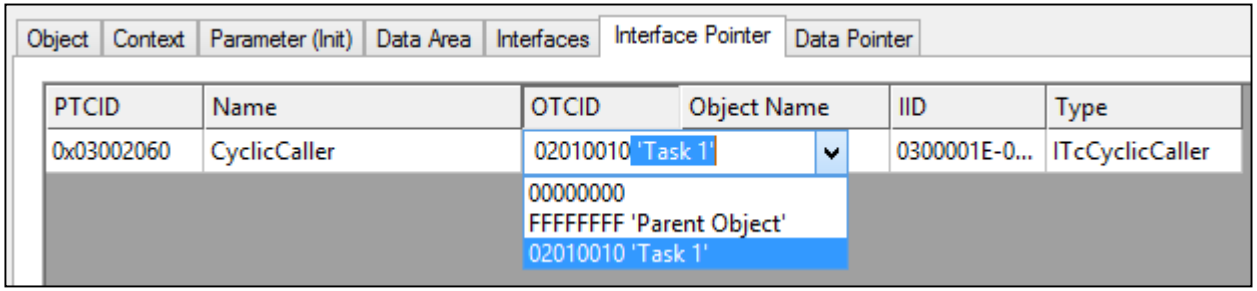
BECKHOFF

- Editor of the TwinCAT Module Instance (*.tmi)
  - Each instance is individually described in the „instance description file“



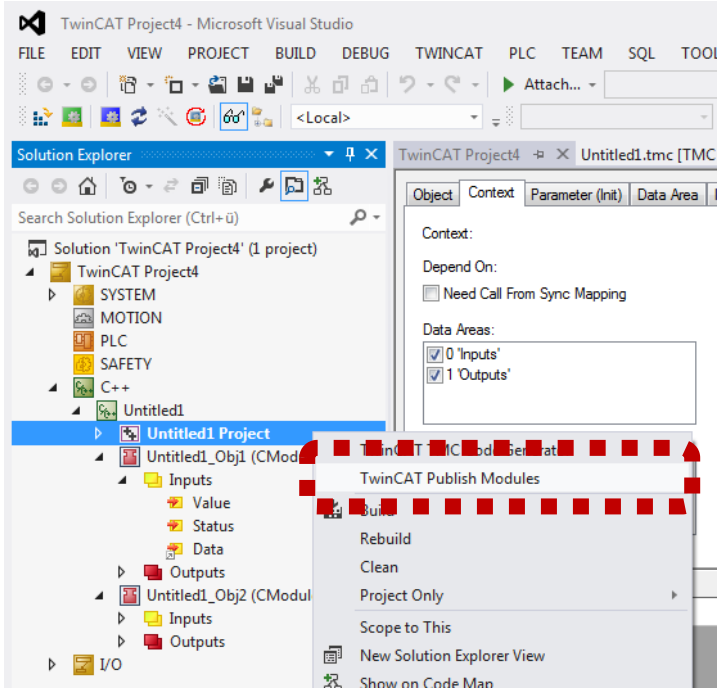






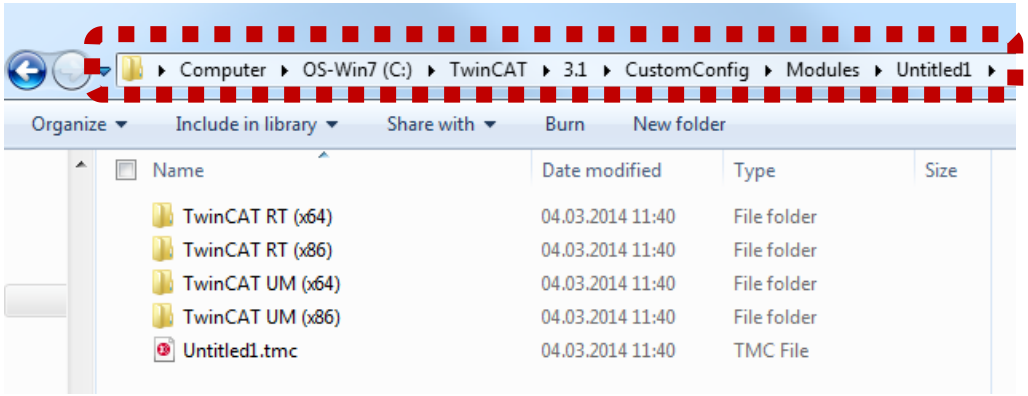
Export of modules

BECKHOFF



Publish (Export) of module

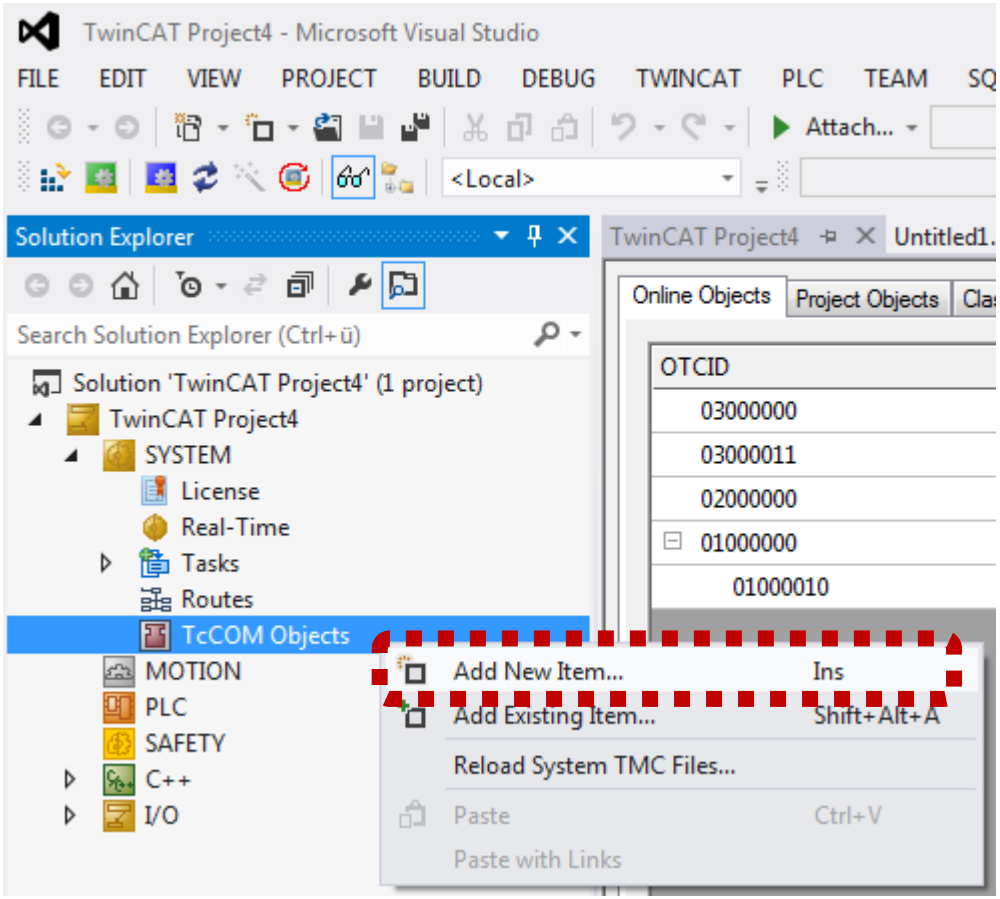
- Right click on C++ project and select “TwinCAT Publish Modules”
- Result:
  - Tmc class description with
  - Platform specific compiles



Could be distributed by simple file-copy

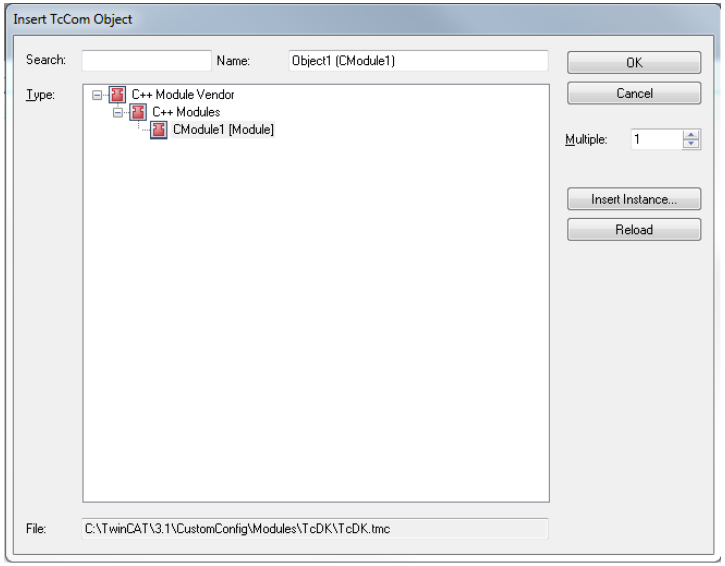
Instanciate / Import of modules

BECKHOFF



- Navigate and right click “System ->TcCOM Objects”

- Select “Add New Item…”
- Select the module to be inserted and number of instances to be created and finalize with “OK”



Overview

BECKHOFF

- TwinCAT C++
  - Quickstart
  - Coming from Usermode
  - Workflow
- TcCOM Modules
  - Concept
- Using TcCOM Interfaces
- TwinCAT C++ Development
- Documentation: Infosys
- Conclusion

Content

BECKHOFF

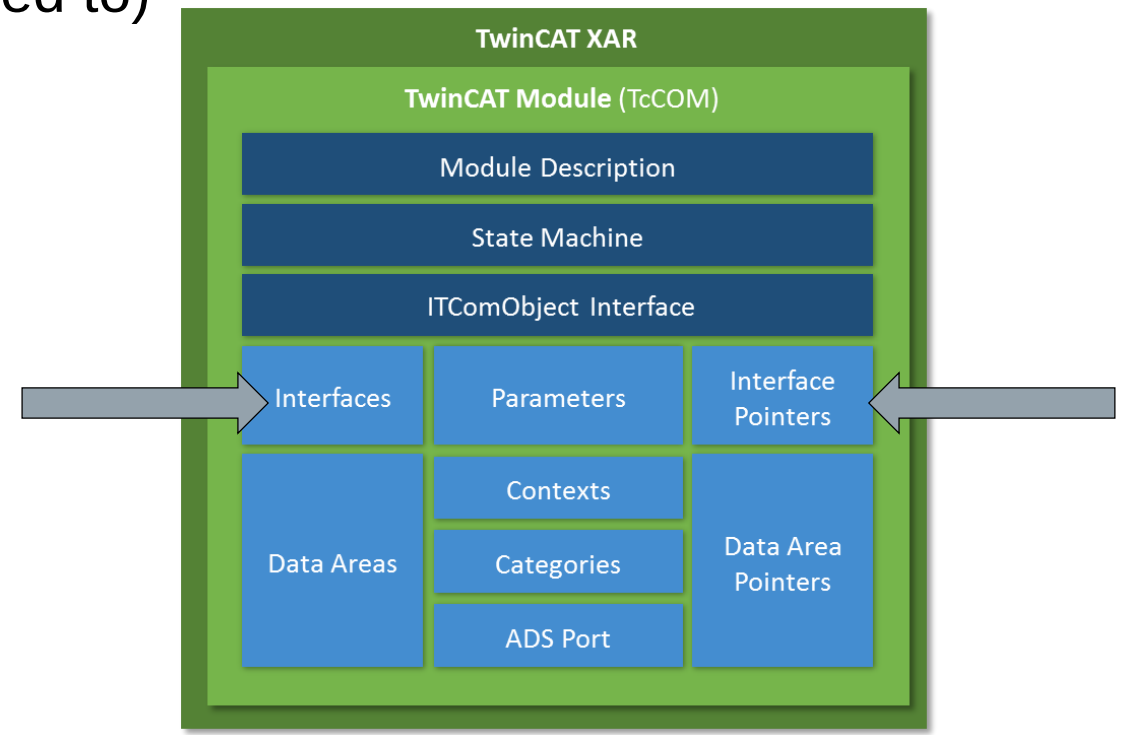
Overview

BECKHOFF

TcCOM Modules

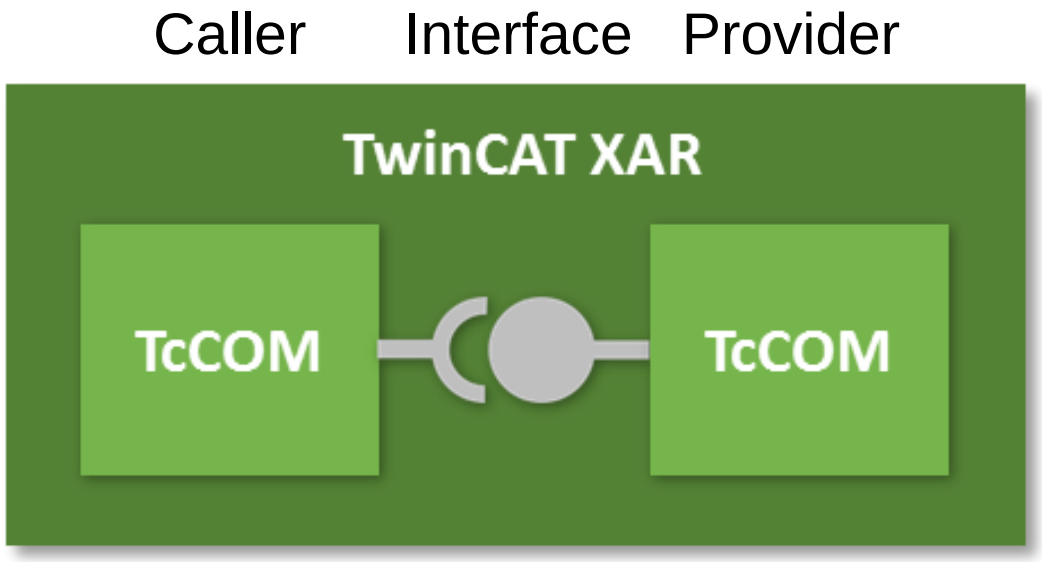
- Represent the building block of TwinCAT-based projects
- Encapsulate functionality
- Providing functionality / implementing interface(s)
- Consume other modules via Interfaces / Interface Pointers

- Could (but does not need to) get called cyclically



Overview

1. Interface-Definition
2. Provider TcCOM Module
   needs to implement the interface
3. Caller TcCOM Module
   needs reference to Provider



Content

BECKHOFF

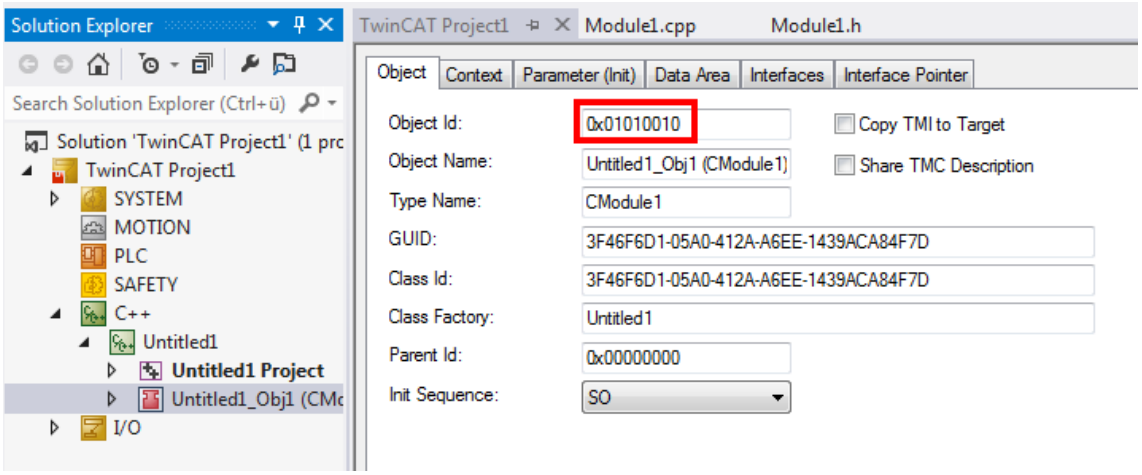
- Overview
  - References
- Creating the interface
  - TMC Editor
  - System Manager
- Creating the Provider
  - PLC
  - C++
- Creating the Caller
  - PLC
  - C++
- Setup and Startup

References: Using OIDs

BECKHOFF

- Caller needs to know the provider
  „On which module instance the method should to be called?“
- Modules are identified by their ObjectID (OID).

→OID of provider needs to be known at caller



→How to define OID at caller?

References: Using OIDs

BECKHOFF

Three ways:

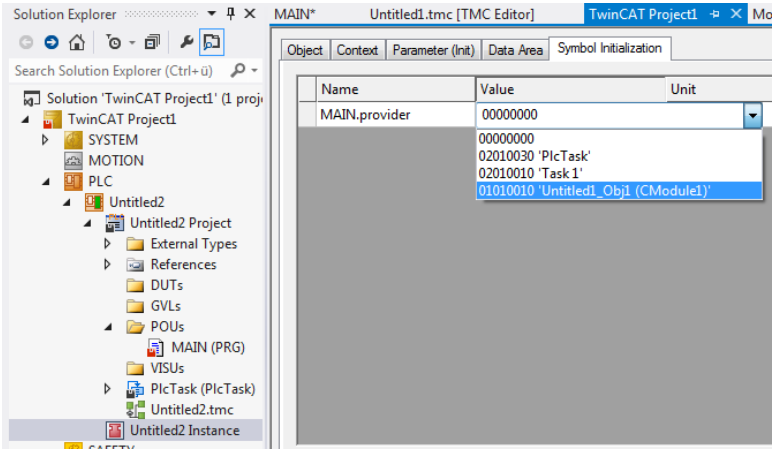
1) Hard coding

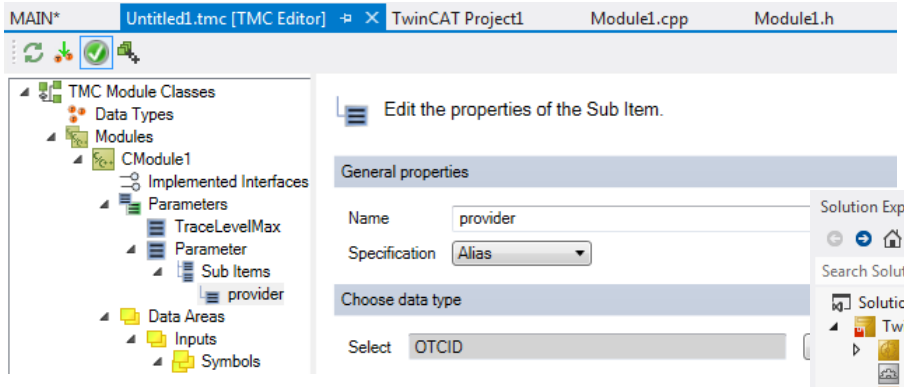
```
////////////////////////////////////////
CModule1::CModule1()
    : m_Trace(m_TraceLevelMax, m_spSrv)
    , m_counter(0)
    , m_Provider(0x01010010)
{
```

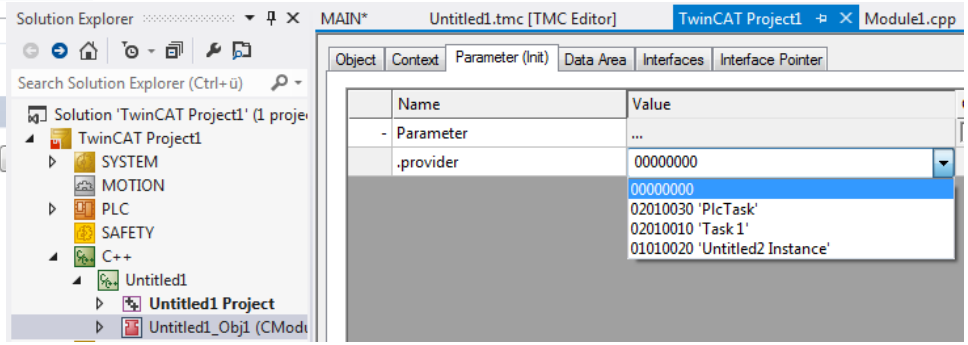
```
1  PROGRAM MAIN
2  VAR
3      provider : OTCID := 16#01010010;
4  END_VAR
```

2) Using Parameters: Define reference without recompile

```
1  PROGRAM MAIN
2  VAR
3      {attribute 'TcInitSymbol':= '1'}
4      provider : OTCID;
5  END_VAR
```





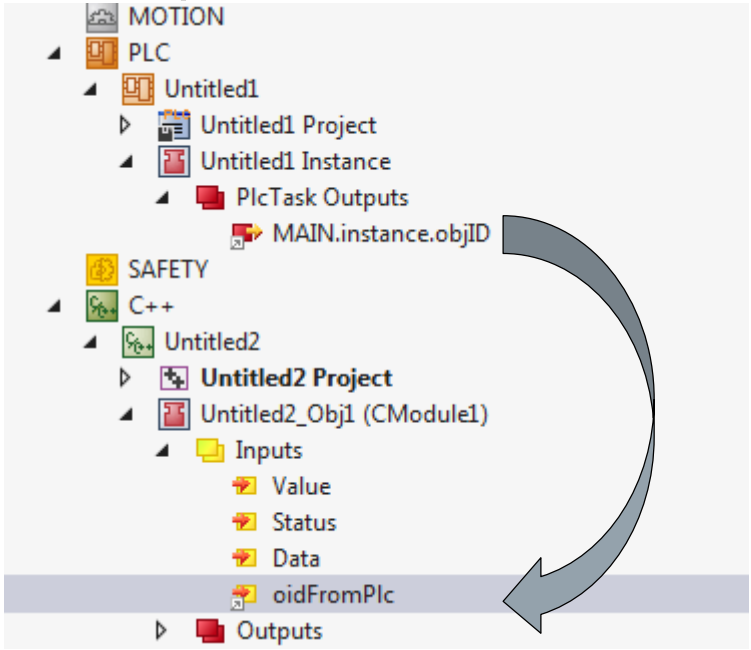


References: Using OIDs

BECKHOFF

3) Using Mapping
(The only way for PLC-provided TcCOM Objects)

- Provide the OID as an output and input
- Use TwinCAT Mapping:



- Pro:
  Don‘t have to deal with the OID value
- Con:
  Mapping will not provide OID during startup phase (in TcCOM statemachine) → Usage is implemented in Realtime (see later)

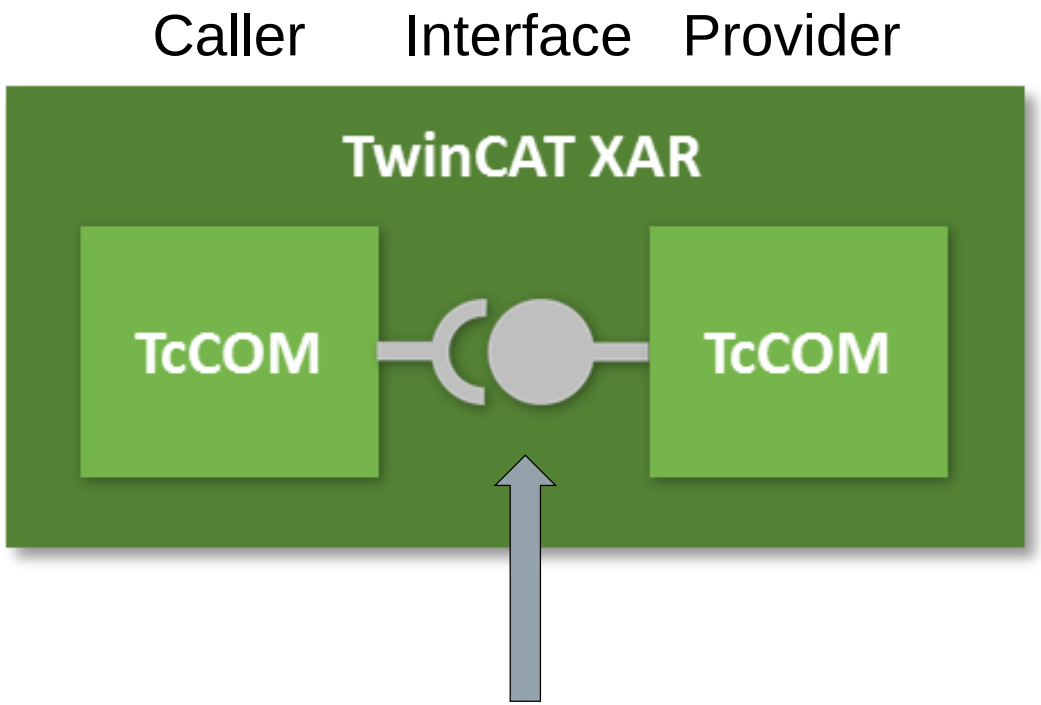
Content

BECKHOFF

- Overview
  - References
- Creating the interface
  - TMC Editor
  - System Manager
- Creating the Provider
  - PLC
  - C++
- Creating the Caller
  - PLC
  - C++
- Setup and Startup

Creating the interface

BECKHOFF

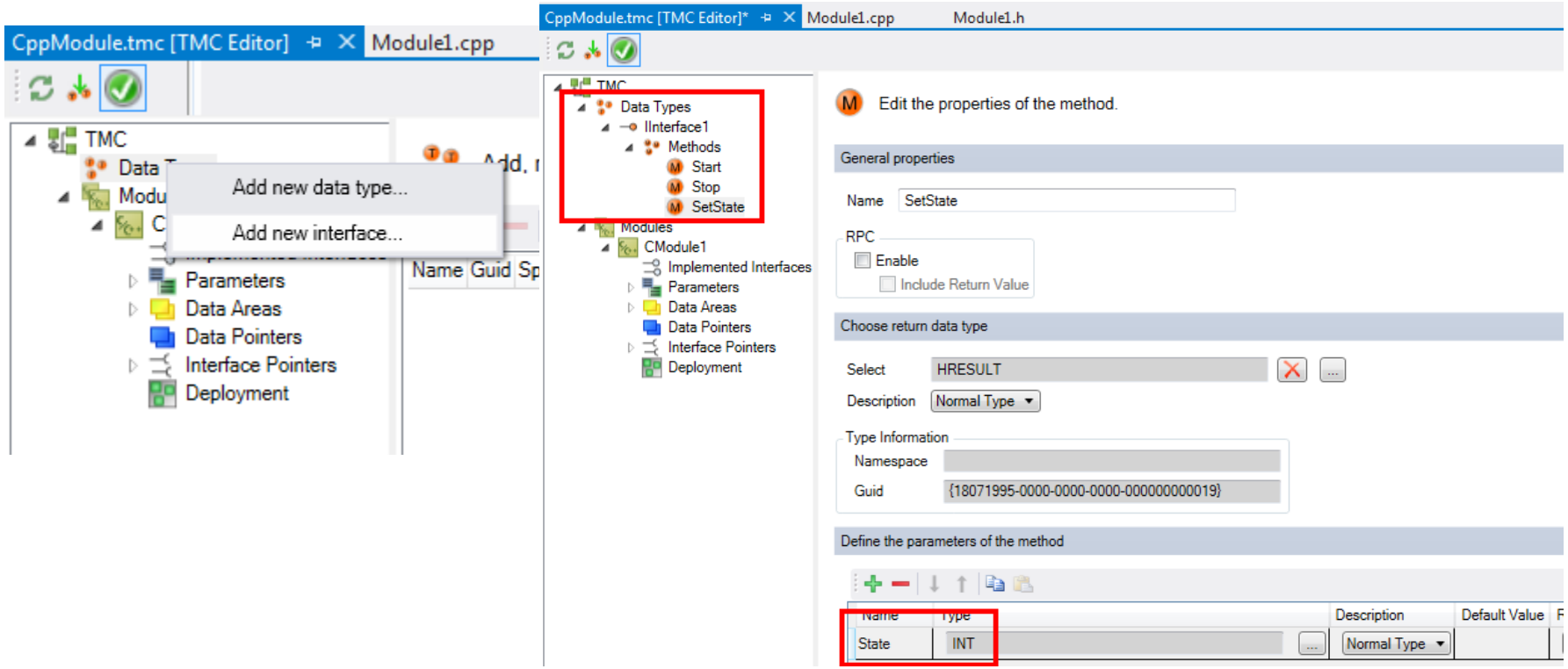


TwinCAT 3 - Creating the interface

BECKHOFF

For C++ modules, the TMC Editor is the prefered way to define the interface

- On „Data Types“ right-click and select „Add new interface…“

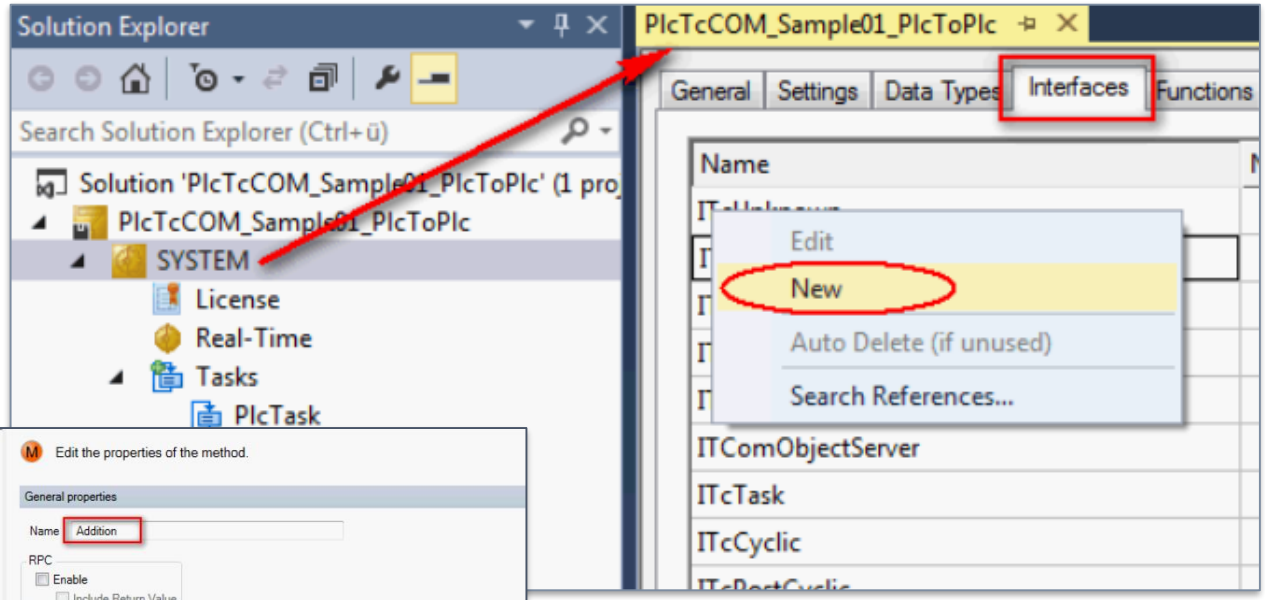


TwinCAT 3 - Creating the interface II

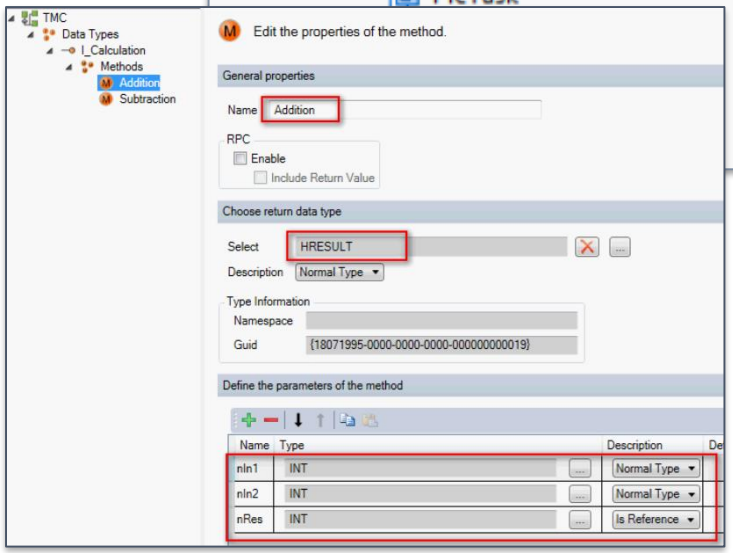
BECKHOFF

Interfaces could also be defined directly and independant of a C++ project.

- System->Interfaces
- Right-Click „New“



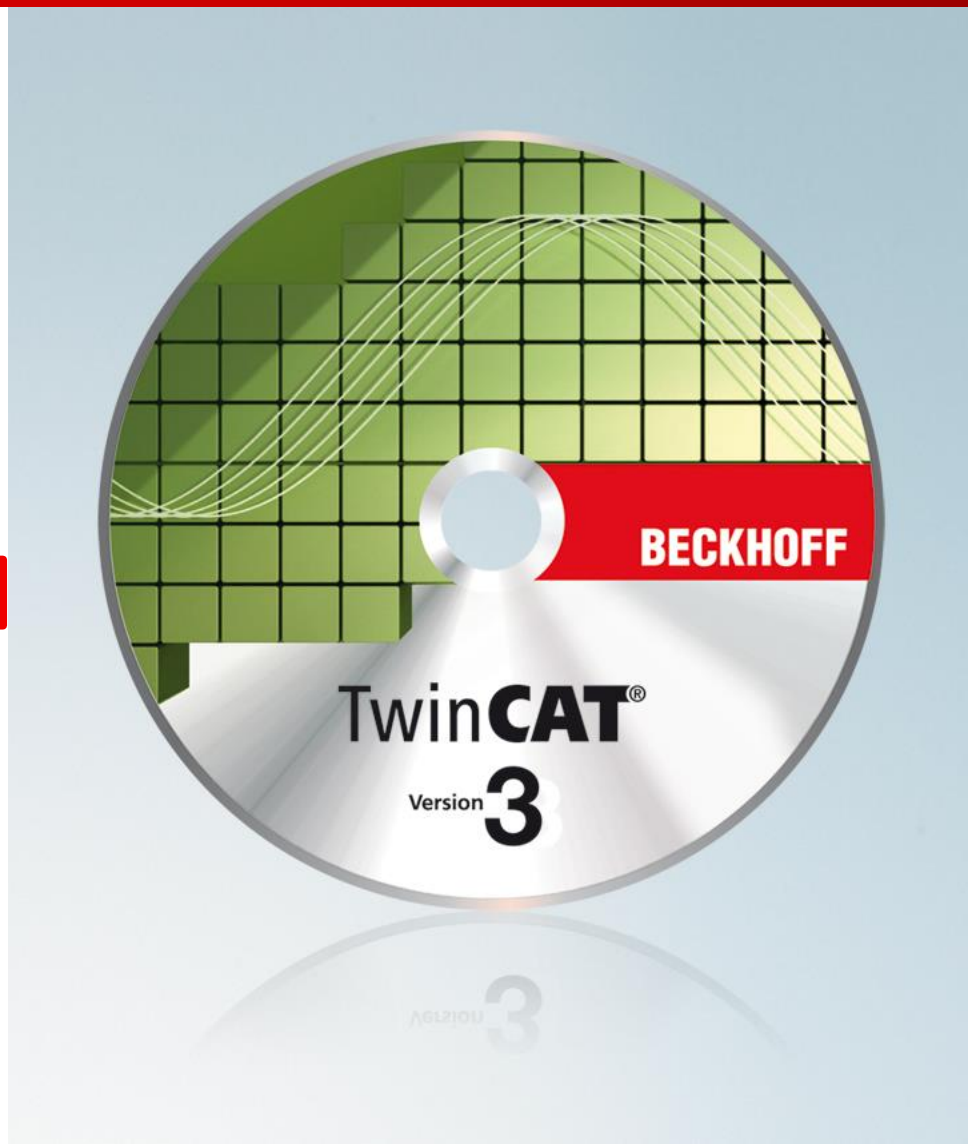
- Method Definition same as in TMC Editor



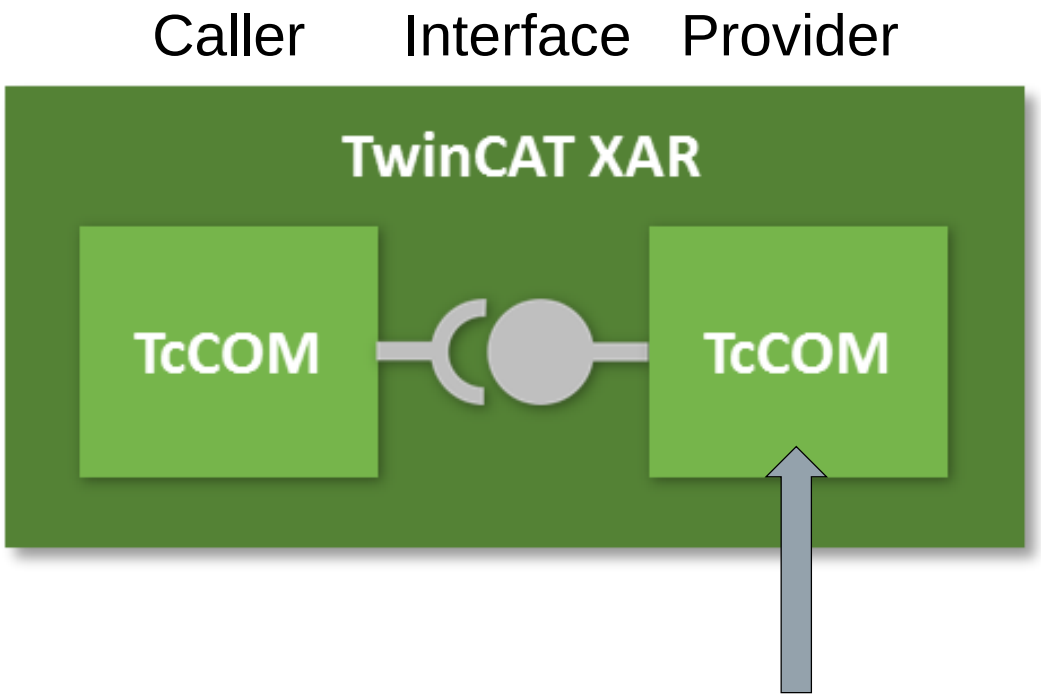
Content

BECKHOFF

- Overview
  - References
- Creating the interface
  - TMC Editor
  - System Manager
- Creating the Provider
  - PLC
  - C++
- Creating the Caller
  - PLC
  - C++
- Setup and Startup



Creating the Provider



Content

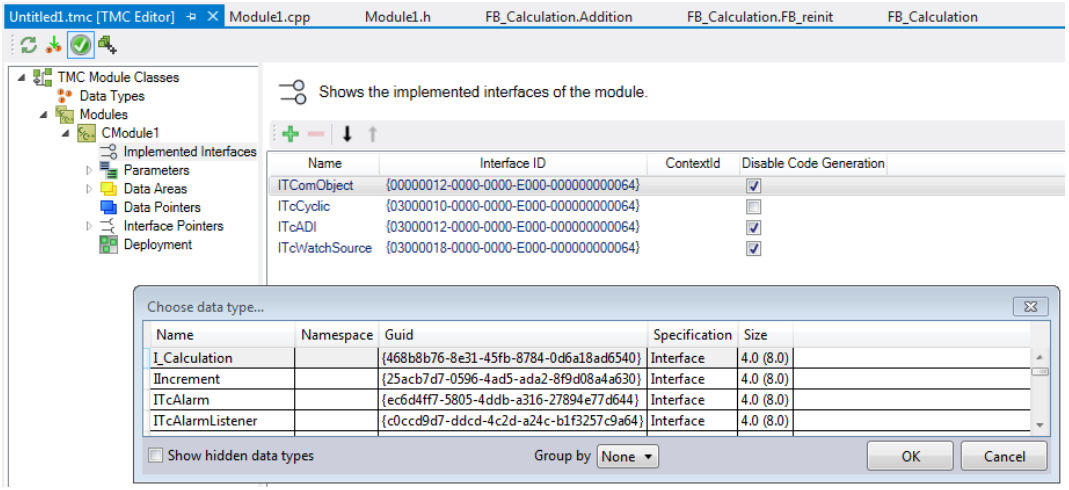
BECKHOFF

- Overview
  - References
- Creating the interface
  - TMC Editor
  - System Manager
- Creating the Provider
  - PLC
  - C++
- Creating the Caller
  - PLC
  - C++
- Setup and Startup

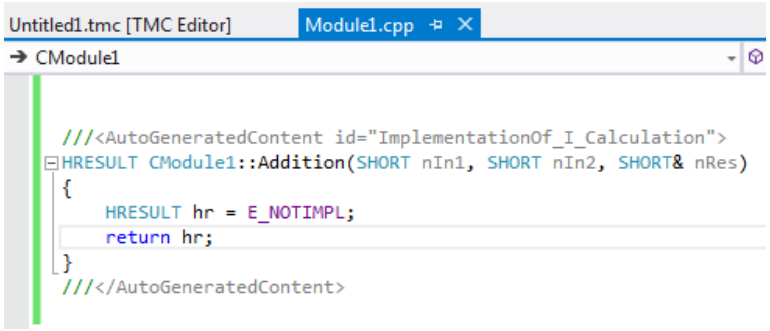
Creating the Provider – C++ I

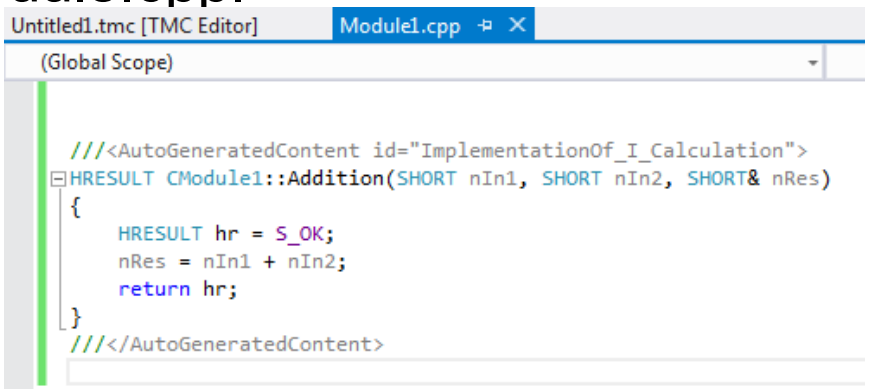
BECKHOFF

- Open TMC Editor and add the interface to „Implemented Interfaces“ of C++ Module



- Run CodeGen
- The Method is generated in Module.cpp. Fill with implementation



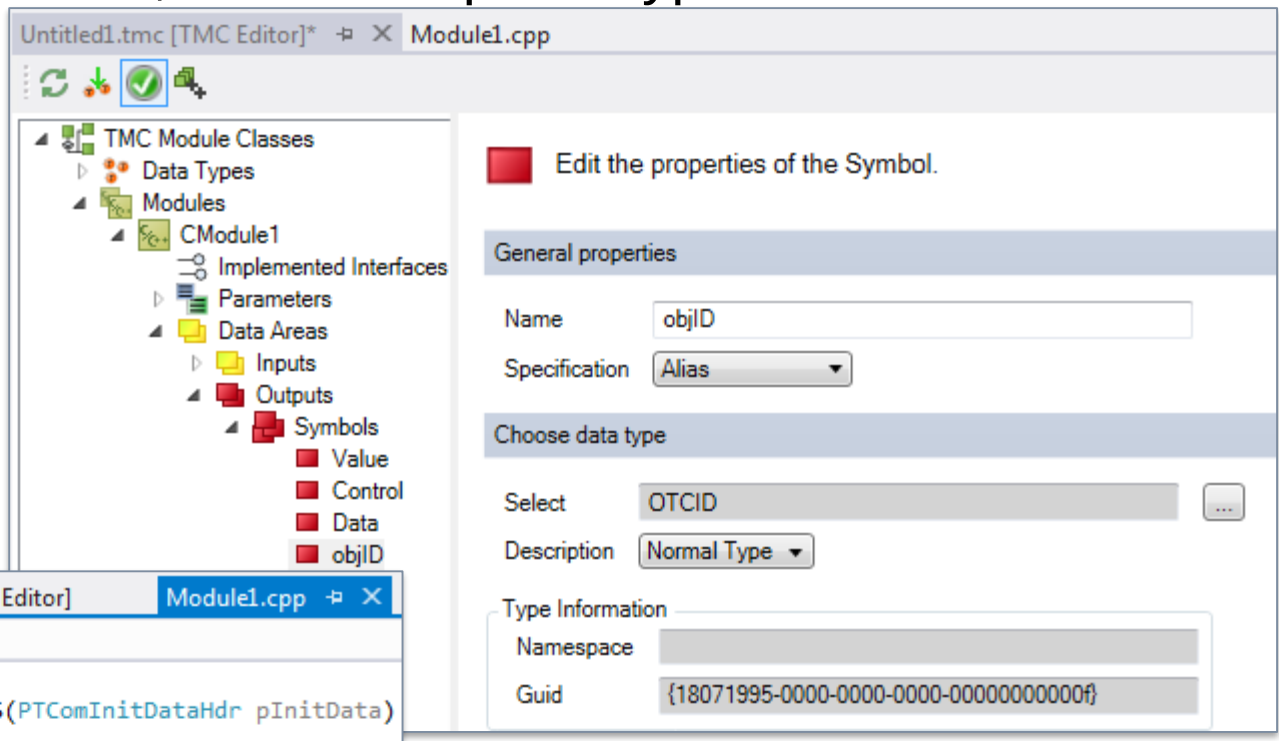


Creating the Provider – C++ II

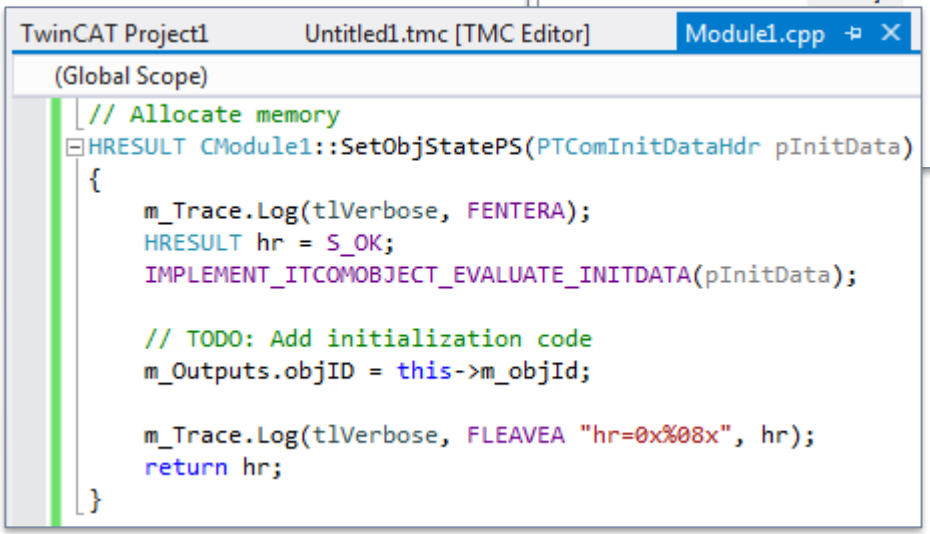
BECKHOFF

- If Mapping should be used, add an output of type OTCID

…and run CodeGen



- Assign ObjID to Output in PS-Transition

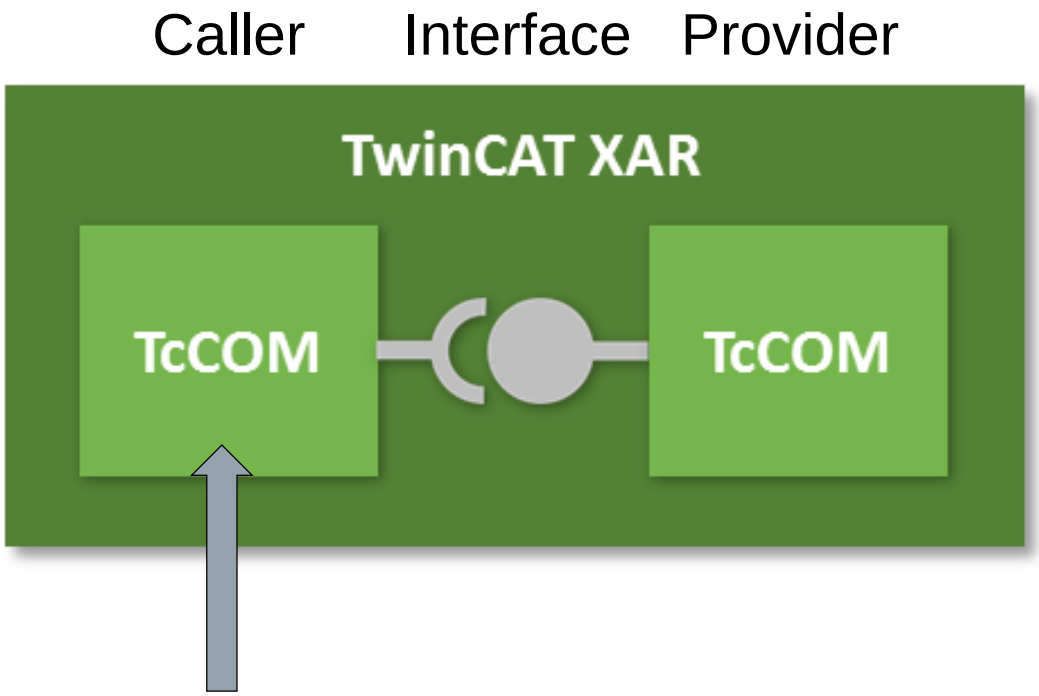


Content

BECKHOFF

- Overview
  - References
- Creating the interface
  - TMC Editor
  - System Manager
- Creating the Provider
  - PLC
  - C++
- Creating the Caller
  - PLC
  - C++
- Setup and Startup

Creating the Caller



Please note:

- Calling methods is a blocking call into the providing module.
- If modules run on different tasks, appropiate locking mechanisms need to be realized.

Content

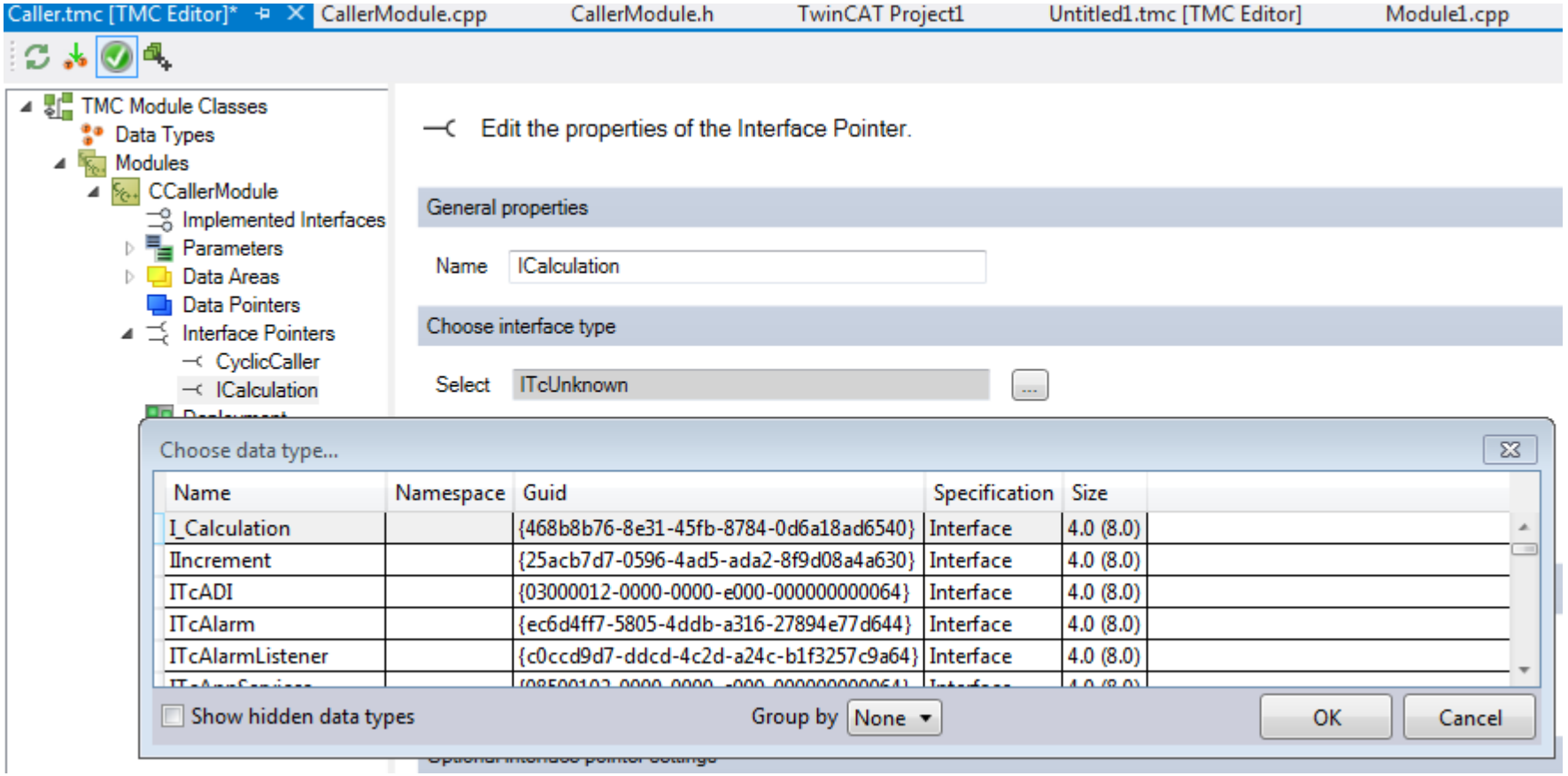
BECKHOFF

- Overview
  - References
- Creating the interface
  - TMC Editor
  - System Manager
- Creating the Provider
  - PLC
  - C++
- Creating the Caller
  - PLC
  - C++
- Setup and Startup

Creating the Caller – C++ I

BECKHOFF

- Open TMC Editor and add the Interface to „Interface Pointers“ of C++ Module



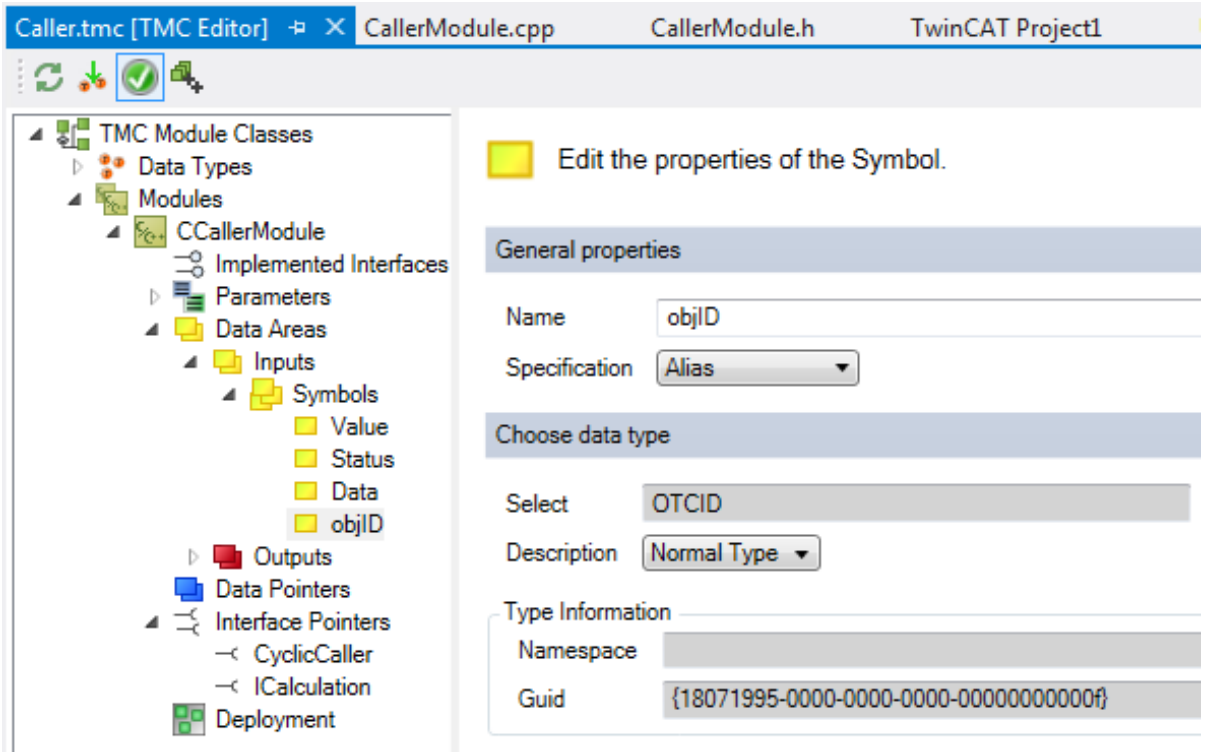
- …and run CodeGen

Creating the Caller – C++ II

BECKHOFF

- If mapping should be used:
  - add an input of type OTCID

- …and run CodeGen



Creating the Caller – C++ III

BECKHOFF

- In CycleUpdate() implement to get reference of provider object

```
///<AutoGeneratedContent id="ImplementationOf_ITcCyclic">
HRESULT CCallerModule::CycleUpdate(ITcTask* ipTask, ITcUnknown* ipCaller, ULONG_PTR context)
{
    HRESULT hr = S_OK;

    // TODO: Replace the sample with your cyclic code
    m_counter+=m_Inputs.Value;

    if(m_Inputs.objID != 0 && m_spICalculation == NULL)
    {
        m_spICalculation.SetOID(m_Inputs.objID);
        m_spSrv->TcQuerySmartObjectInterface(m_spICalculation);
    }
    if(m_spICalculation != NULL) {
        m_spICalculation->Addition(m_counter, m_counter, m_Outputs.Value);
    }

    return hr;
}
///</AutoGeneratedContent>
```

```
HRESULT CCallerModule::SetObjStateOS()
{
    m_Trace.Log(tlVerbose, FENTERA);
    HRESULT hr = S_OK;
    RemoveModuleFromCaller();

    // TODO: Add any additional deinitialization
    m_spICalculation = NULL;

    m_Trace.Log(tlVerbose, FLEAVEA "hr=0x%08x", hr);
    return hr;
}
```

- Interface pointer could then be used like any other local pointer

- Releasing the interface pointer could be done in transition OS
- In case no mapping is used for referencing the provider, the TcQueryInterface could be done in state machine already

Content

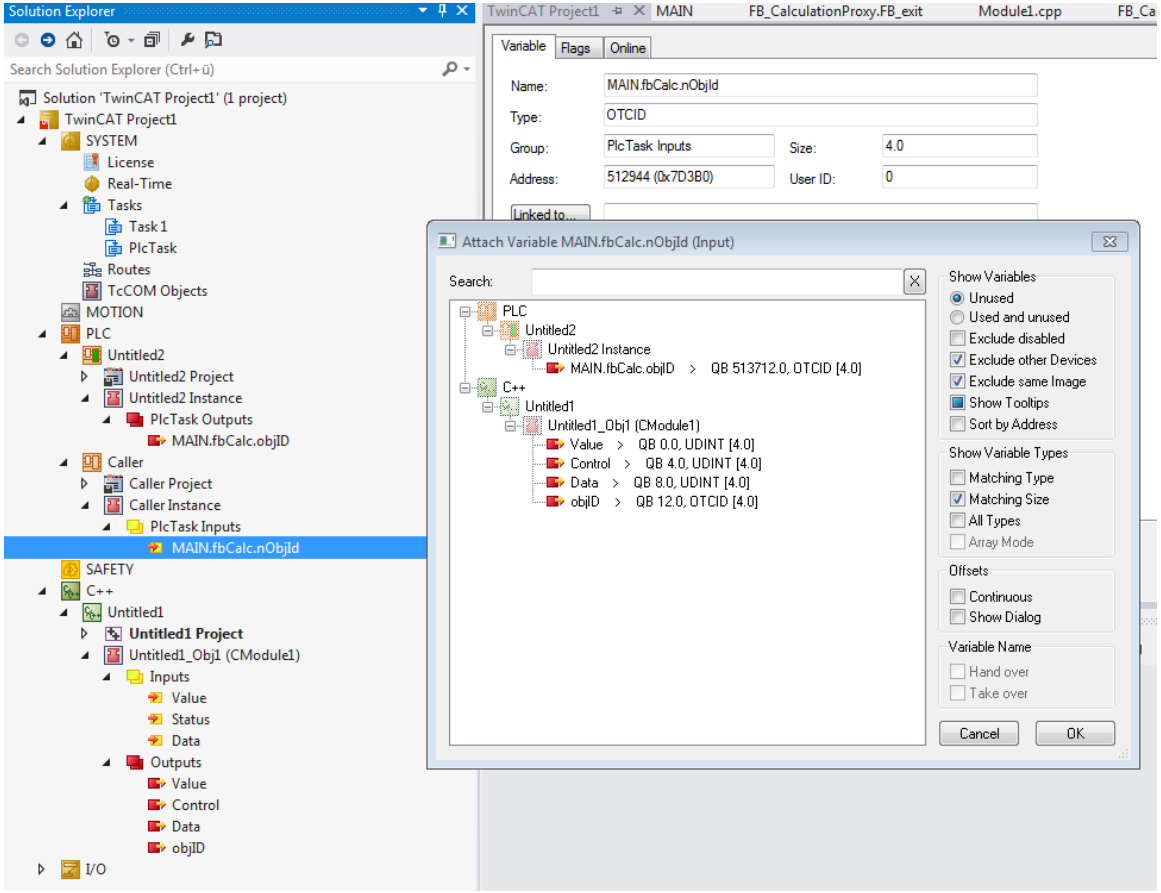
BECKHOFF

- Overview
  - References
- Creating the interface
  - TMC Editor
  - System Manager
- Creating the Provider
  - PLC
  - C++
- Creating the Caller
  - PLC
  - C++
- Setup and Startup

Setup

BECKHOFF

- If mapping is used for referencing provider from caller, setup is simply done by linking the symbols:



- Otherwise the OIDs need to be propagated

Startup

BECKHOFF

- PLC calling C++:

TwinCAT Project1 | MAIN [Online] | FB_CalculationProxy.FB_exit [Online] | Module1.cpp

TwinCAT_Device.Caller.MAIN

Expression	Type	Value	Prepared value	Ad
fbCalc	FB_CalculationProxy			
hrCalc	HRESULT	16#00000000		
a	INT	10		
b	INT	7		
nSum	INT	17		

```
1  IF fbCalc.ip 16#FFFFFA801361C510 = 0 THEN
2      hrCalc 0 := fbCalc.GetInterfacePointer();
3  END_IF
4  IF fbCalc.ip 16#FFFFFA801361C510 <> 0 THEN
5      hrCalc 0 := fbCalc.ip.Addition(a 10 ,b 7 ,nSum 17 );
6  END_IFRETURN
```

- C++ calling PLC:

```
if (m_spICalculation != NULL)
{
    m_spICalculation->Addition(m_counter, m_counter, m_Outputs.Value);
}

return hr;
```

100 %

Autos

Name	Value
hr	0
m_Outputs	{Value=3692 Control=0 Data=0 }
m_Outputs.Value	3692
m_counter	1846
m_spICalculation	{m_pInterface={...} m_oid=16842784 }
this	0xfffffa8019c1b428

Overview

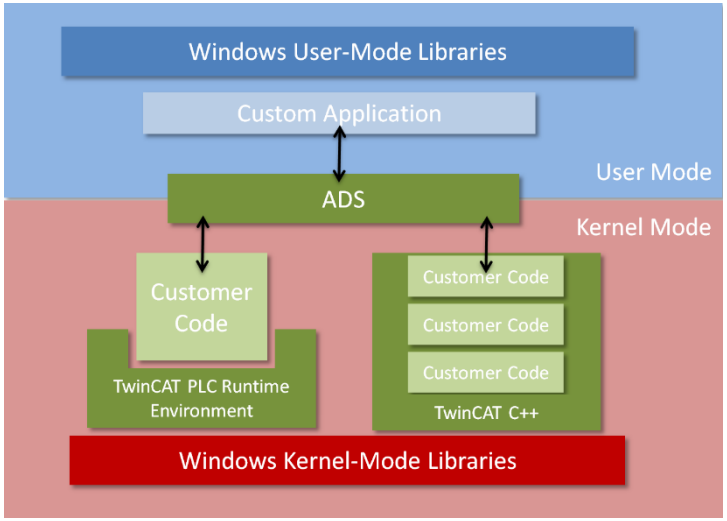
BECKHOFF

- TwinCAT C++
  - Quickstart
  - Coming from Usermode
  - Workflow
- TcCOM Modules
  - Concept
- Using TcCOM Interfaces
- TwinCAT C++ Development
- Documentation: Infosys
- Conclusion

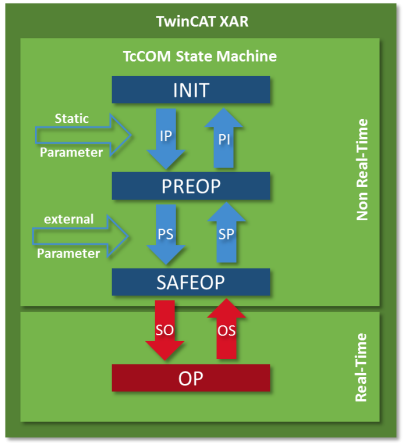
Startup/Shutdown

BECKHOFF

- TwinCAT Modules are Windows Drivers
- TwinCAT Modules get executed
  - in Windows Kernel Context and
  - in TwinCAT Realtime Context



- Code for *Initialitsation* and *Deinitialisation* is executed in Windows Kernel Context
- Last Phase and cyclic execution is done in TwinCAT Realtime Context

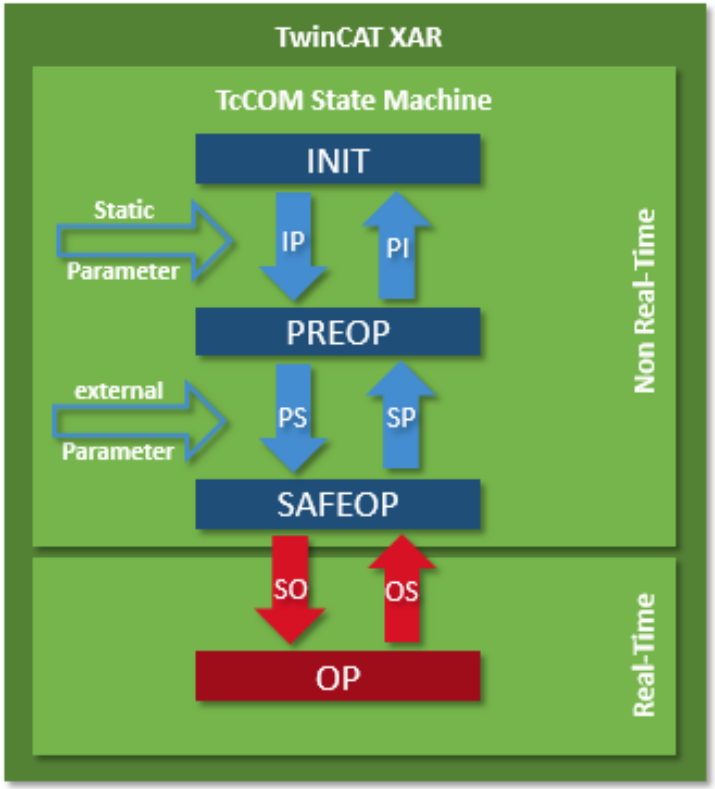


TwinCAT Memory Management

BECKHOFF

- In general memory could be reserved from
  - Member variables of modules
  - TMC Editor‘s Dataareas.

- (Dynamic) Memory can be allocated in statemachine
- Recommended to always release the memory in the "symmetric" transition, e.g. allocation in PS, release in SP
- Memory is taken from
  - Non-paged pool of OS (blue)
  - TcRouter memory (red)



TwinCAT C++ SDK

BECKHOFF

- TwinCAT C++ SDK is installed to C:\TwinCAT\3.x\sdk\Include

- Important parts of the TwinCAT C++ SDK
  - TcInterfaces.h and TcServices.h:
    TcCOM Framework
  - TcIoInterfaces.h:
    Tasks and Dataarea-Access
  - TcMath/*:
    mathematical functions
  - ADS communication
  - TwinCAT Runtime →
  - TwinCAT STL →

TwinCAT Runtime

BECKHOFF

- TwinCAT has its own implementation of Runtime Library (so its own „CRT“)
- Available via RtlR0.h
- Overview
  - Memset
  - Memcpy
  - Memcmp
  - Scanf
  - Vsprintf
  - …and more

TwinCAT STL

BECKHOFF

- STL is supported in TwinCAT
  - In folder Stl/*
  - Partly implemented, continuously enhanced
  - List
    Map
    Set
    Stack
    String
    Vector
    Wstring
    …
  - Algorithm (some; like binary_search etc.)

Overview

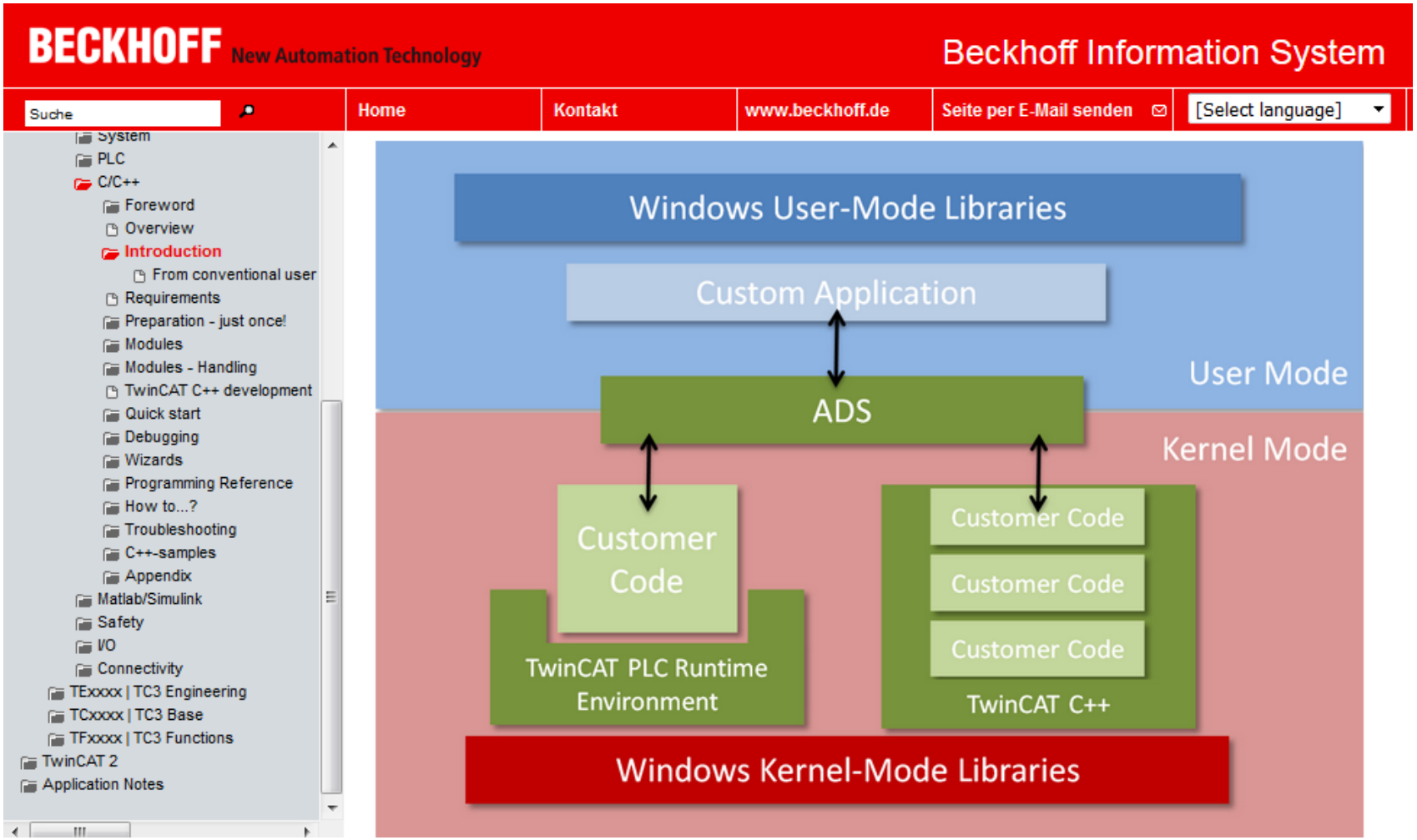
BECKHOFF

- TwinCAT C++
  - Quickstart
  - Coming from Usermode
  - Workflow
- TcCOM Modules
  - Concept
- Using TcCOM Interfaces
- TwinCAT C++ Development
- Documentation: Infosys
- Conclusion

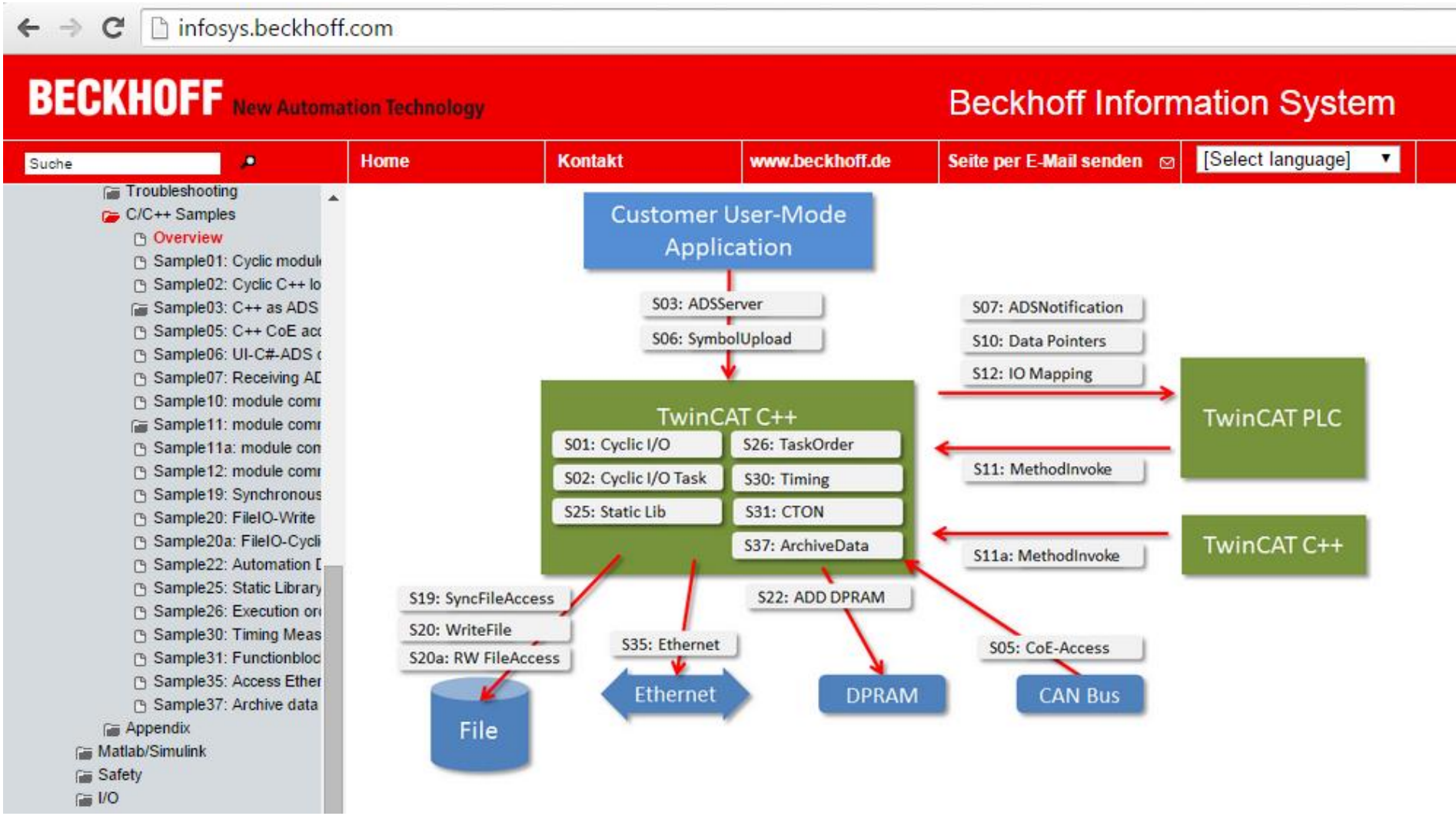
Documentation: Infosys

BECKHOFF

Please visit: http://infosys.beckhoff.com



Samples Samples Samples

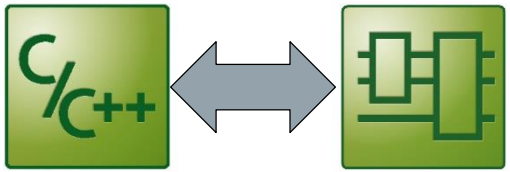


Overview

BECKHOFF

- TwinCAT C++
  - Quickstart
  - Coming from Usermode
  - Workflow
- TcCOM Modules
  - Concept
- Using TcCOM Interfaces
- TwinCAT C++ Development
- Documentation: Infosys
- Conclusion

PLC vs C++



- „Cyclic thinking“ in both worlds
- Each project is a module; modules could communicate in different ways
- No difference on IO Level
- Debugging via Breakpoints and online monitoring in C++ as well as in PLC

- Noticeable differences to PLC
  - Fine graduated access to startup / shutdown
  - Direct access to hardware via DPRAM
  - ~~No Online Change~~ not any more