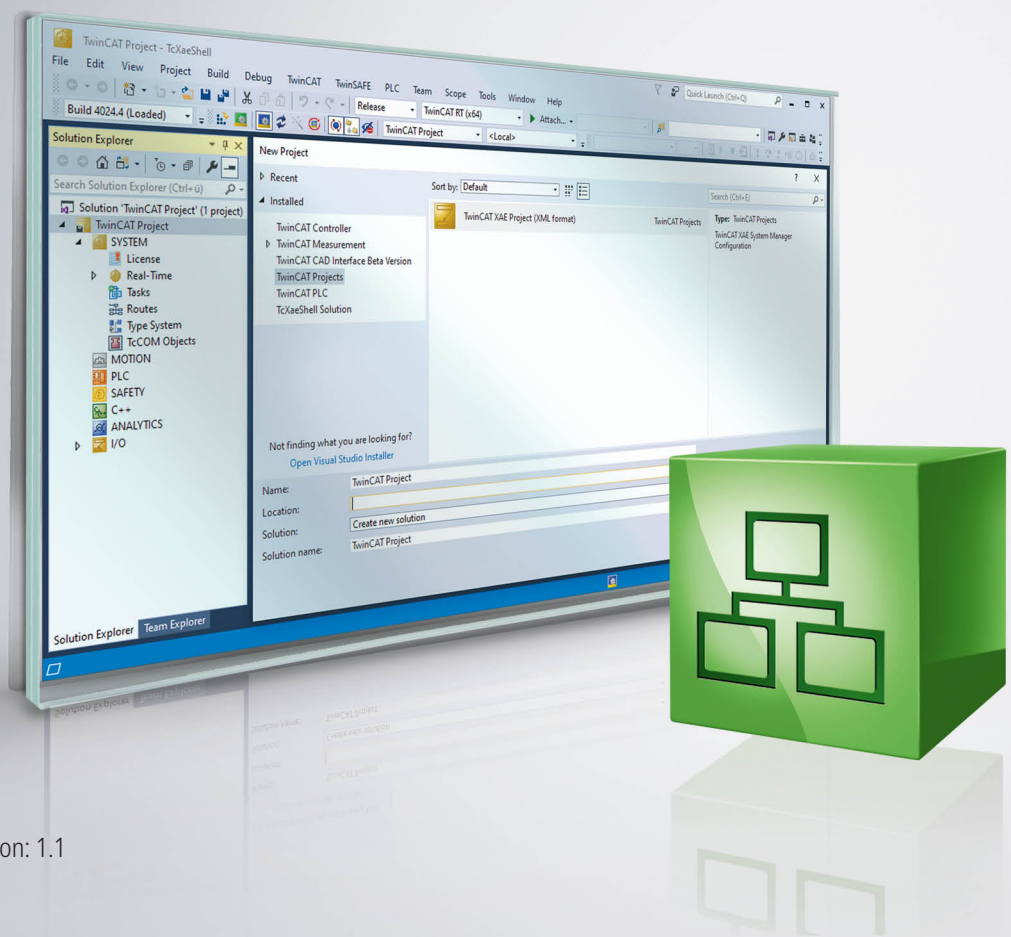
BECKHOFF New Automation Technology

Manual | EN

TF6710

TwinCAT 3 | IoT Functions



2021-05-06 | Version: 1.1

Table of contents

1 Foreword 5
1.1 Notes on the documentation 5
1.2 Safety instructions 6
2 Overview 7
3 Installation 8
3.1 System requirements 8
3.2 Setup scenarios 8
3.3 Installation 8
3.4 Licensing 11
4 Technical introduction 14
4.1 Reference Data Agent 15
4.2 Communication patterns 15
4.3 Programming workflow 16
4.4 Synchronizing message operations 17
4.5 Timeout settings 19
5 Configuration 21
5.1 Overview 21
5.2 Configurator 21
5.2.1 Topology view 23
5.2.2 Tree view 24
5.2.3 Mappings 25
5.2.4 Target Browser 25
5.2.5 Cascading Editor 25
5.2.6 Parameter Editor 26
5.2.7 Settings 27
5.2.8 Error logging 34
6 PLC API 35
6.1 Function blocks 35
6.1.1 FB_IotFunctions_Connector 35
6.1.2 FB_IotFunctions_Message 36
6.1.3 FB_IotFunctions_Request 37
6.2 Data types 38
6.2.1 ST_IotFunctionsEvent 38
6.2.2 ST_IotFunctionsMessage 38
6.2.3 ST_IotFunctionsRequest 39
6.2.4 ST_IotFunctionsRequestContainer 39
7 Samples 41
8 Appendix 42
8.1 Support and Service 42

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

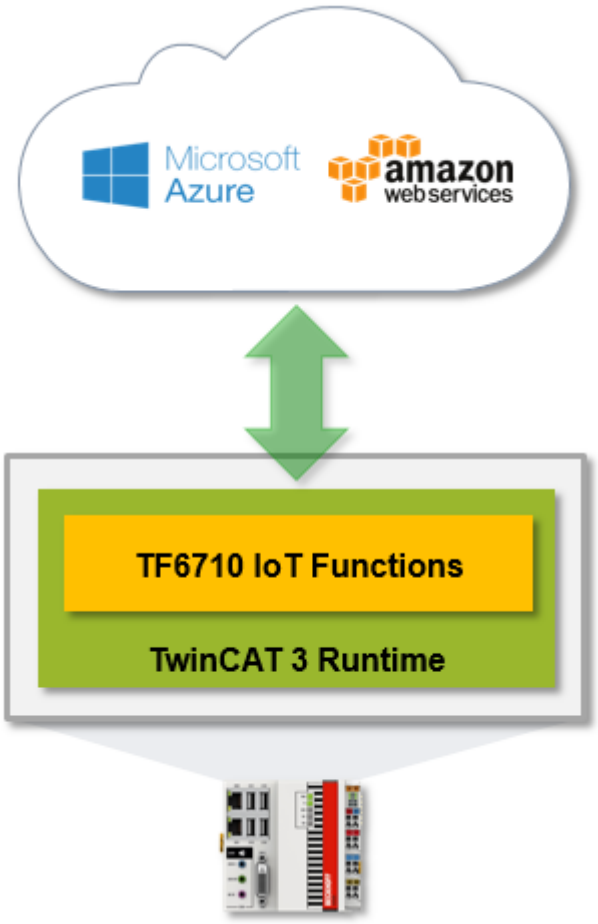
Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.

Tip or pointer

This symbol indicates information that contributes to better understanding.

2 Overview

TC3 IoT Functions is a product for the TwinCAT 3 runtime that enables bi-directional data communication with cloud services from within the machine program. The transport mechanism depends on the cloud service. It can be MQTT, AMQP, ADS or OPC UA.



3 Installation

3.1 System requirements

TC3 IoT Functions requires TC3 IoT Data Agent to run in the background. For more information see the technical introduction [▶ 14], the setup scenarios [▶ 8] and the TC3 IoT Data Agent system requirements.

Technical data	Description
Operating system	Windows 7/10, Windows Embedded Standard 7, Windows Embedded Compact 7
Target platform	PC architecture (x86, x64, ARM)
.NET Framework	not required
TwinCAT version [1]	TwinCAT 3 Build 4022.20 (or higher)
TwinCAT installation level	TwinCAT 3 XAE, XAR
Required TwinCAT license [2]	TF6710 TC3 IoT Functions
Required setup	TF6720 TC3 IoT Data Agent Driver and PLC library for TC3 IoT Functions are automatically included in base TwinCAT installation.

[1] Version of the TwinCAT 3 runtime on which TC3 IoT Functions can be executed

[2] Although TC3 IoT Functions (TF6710) has a technical dependency on TC3 IoT Data Agent (TF6720), a license for TF6720 is not required.

3.2 Setup scenarios

TC3 IoT Functions and TC3 IoT Data Agent can either be installed on the same computer or separately from each other on different devices.

TC3 IoT Functions and TC3 IoT Data Agent on the controller

In this scenario, TC3 IoT Functions and TC3 IoT Data Agent are running on the (same) controller. This is the default setup scenario and no further settings have to be made. A TF6710 license is required on the controller to use the TC3 IoT Functions PLC library.

TC3 IoT Functions on the controller and TC3 IoT Data Agent on a gateway device

In this scenario, TC3 IoT Functions is running on the controller and TC3 IoT Data Agent is installed on a gateway device. The communication between TC3 IoT Functions and TC3 IoT Data Agent is based on ADS. The PLC function blocks of TC3 IoT Functions need to reference the gateway device on which the TC3 IoT Data Agent is installed (see Reference Data Agent [▶ 15]). The gateway device does not require any additional licenses. A TF6710 license is only required on the devices that use the TC3 IoT Functions PLC library.

TC3 IoT Functions and TC3 IoT Data Agent on a gateway device

In this scenario, TC3 IoT Functions and TC3 IoT Data Agent are installed and executed on a gateway device, e.g. in an edge scenario. Data ingest to the edge device can be anything, ranging from TCP based protocols like OPC UA to totally other ways of data communication. A TF6710 license is required on the gateway device to use the TC3 IoT Functions PLC library.

3.3 Installation

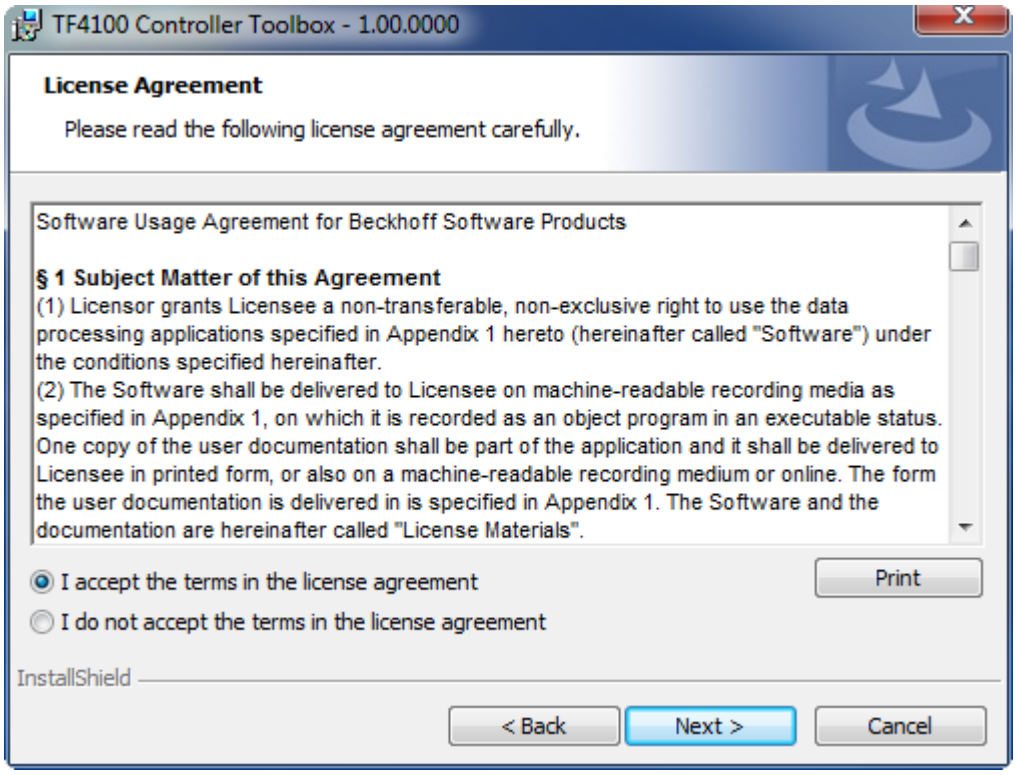
The following section describes how to install the TwinCAT 3 Function for Windows-based operating systems.

✓ The TwinCAT 3 Function setup file was downloaded from the Beckhoff website.

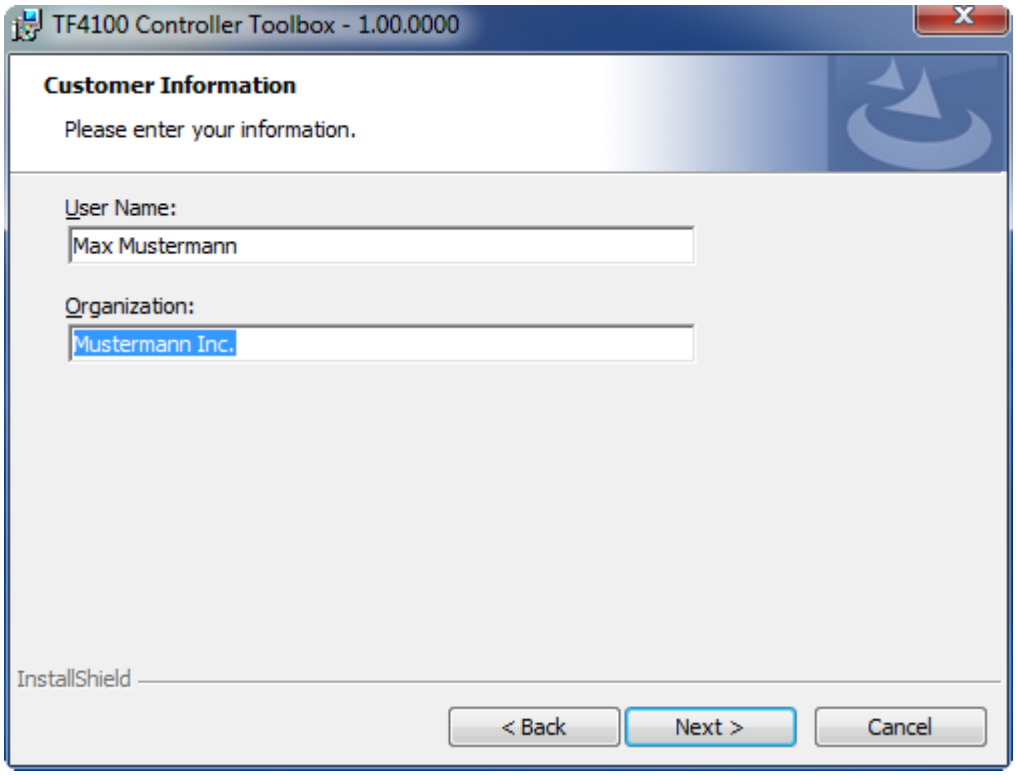
1. Run the setup file as administrator. To do this, select the command **Run as administrator** in the context menu of the file.

   ⇨ The installation dialog opens.

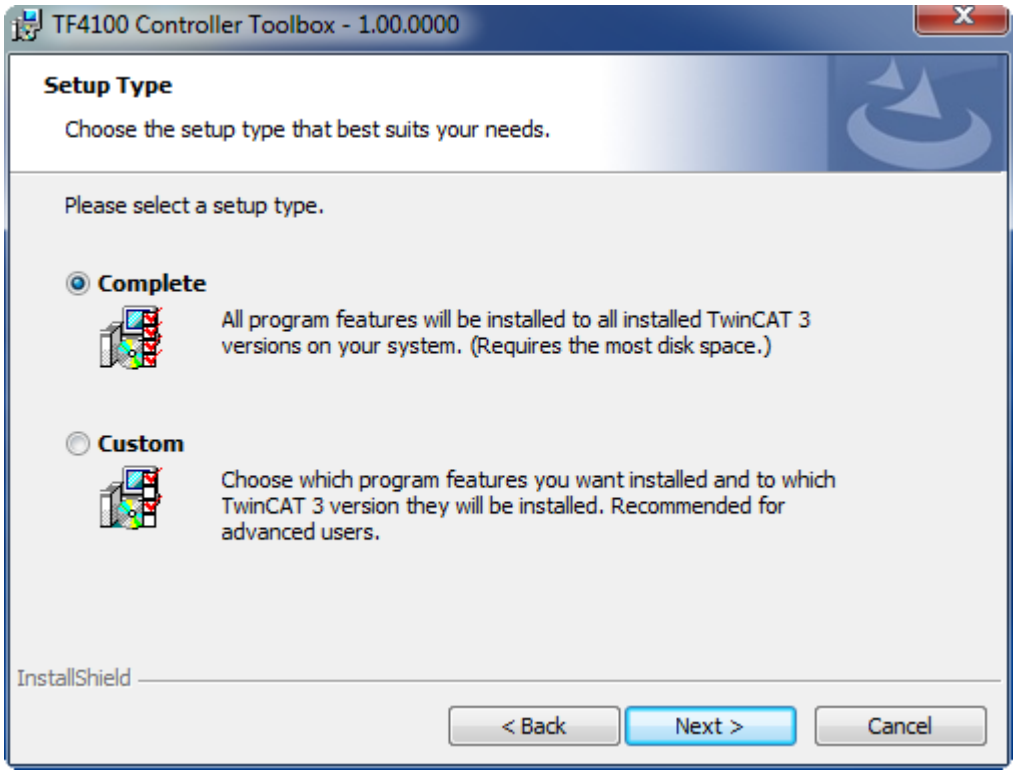
2. Accept the end user licensing agreement and click **Next**.



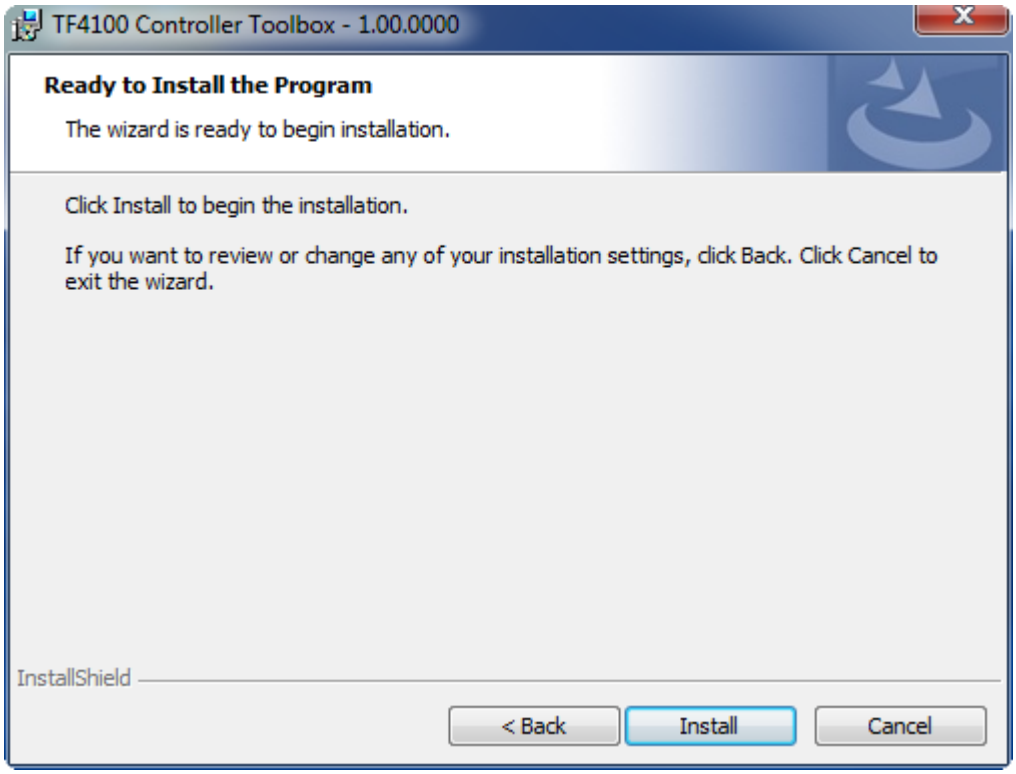
3. Enter your user data.



4. If you want to install the full version of the TwinCAT 3 Function, select **Complete** as installation type. If you want to install the TwinCAT 3 Function components separately, select **Custom**.

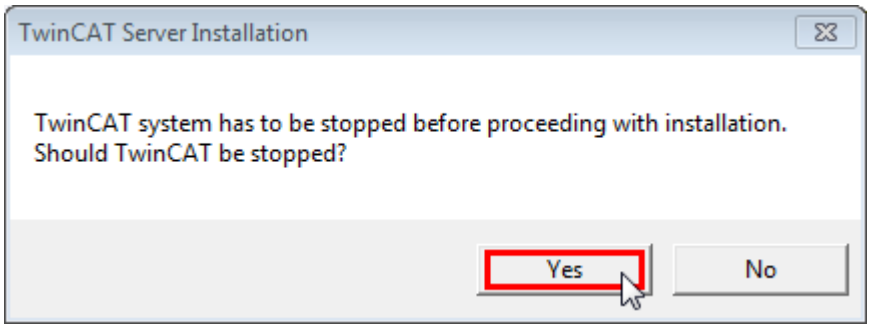


5. Select **Next**, then **Install** to start the installation.

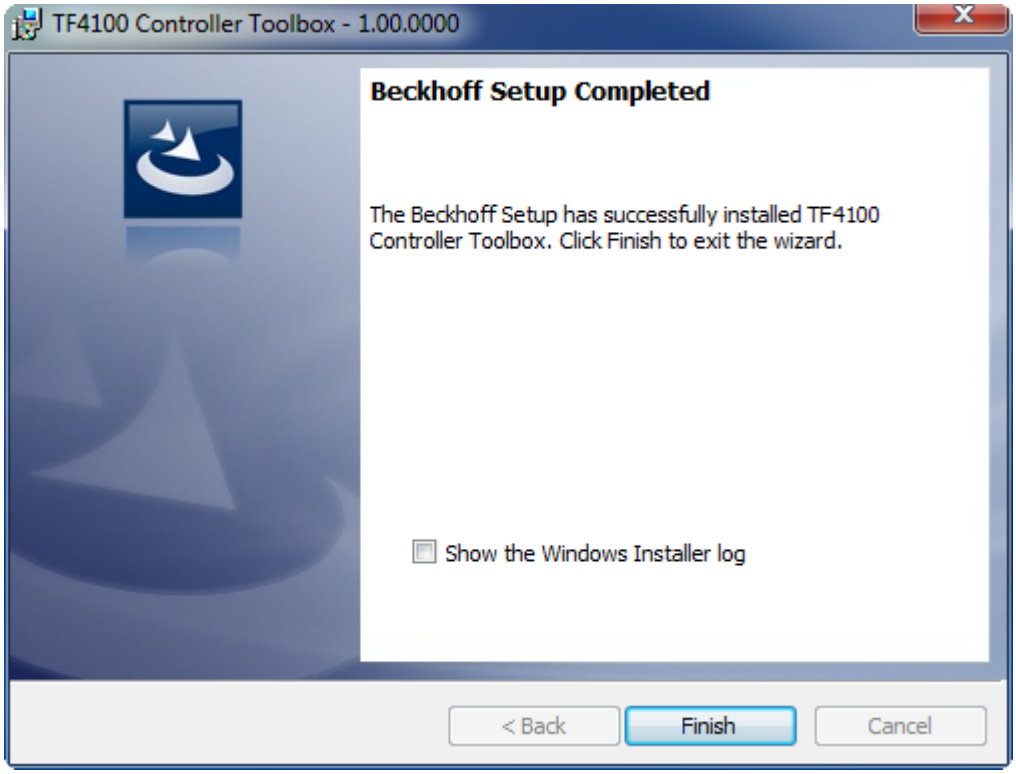


⇨ A dialog box informs you that the TwinCAT system must be stopped to proceed with the installation.

6. Confirm the dialog with **Yes**.



7. Select **Finish** to exit the setup.



⇨ The TwinCAT 3 Function has been successfully installed and can be licensed (see Licensing [▶ 11]).

3.4 Licensing

The TwinCAT 3 function can be activated as a full version or as a 7-day test version. Both license types can be activated via the TwinCAT 3 development environment (XAE).

Licensing the full version of a TwinCAT 3 Function

A description of the procedure to license a full version can be found in the Beckhoff Information System in the documentation "TwinCAT 3 Licensing".

Licensing the 7-day test version of a TwinCAT 3 Function

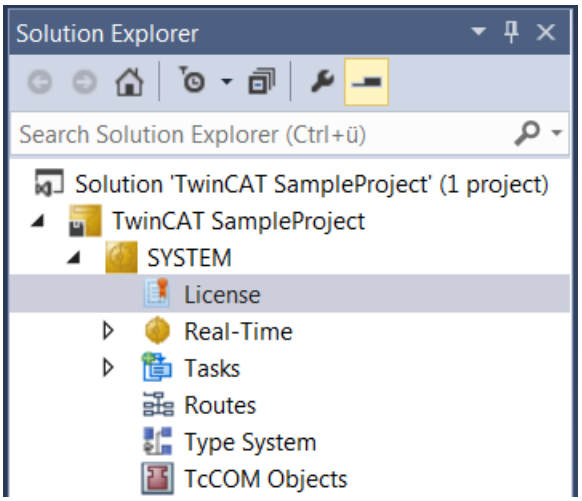
A 7-day test version cannot be enabled for a TwinCAT 3 license dongle.

1. Start the TwinCAT 3 development environment (XAE).
2. Open an existing TwinCAT 3 project or create a new project.

3. If you want to activate the license for a remote device, set the desired target system. To do this, select the target system from the **Choose Target System** drop-down list in the toolbar.

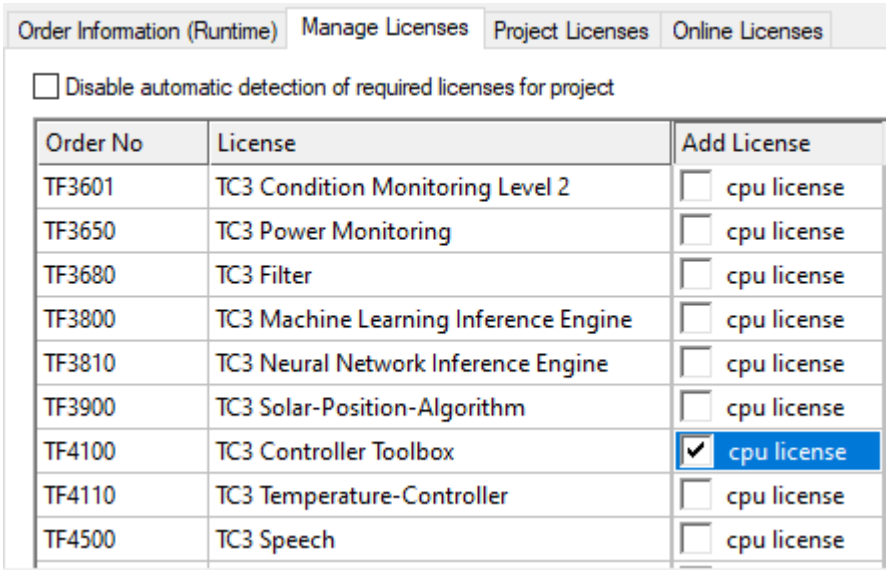
   ⇨ The licensing settings always refer to the selected target system. When the project is activated on the target system, the corresponding TwinCAT 3 licenses are automatically copied to this system.

4. In the **Solution Explorer**, double-click **License** in the **SYSTEM** subtree.



   ⇨ The TwinCAT 3 license manager opens.

5. Open the **Manage Licenses** tab. In the **Add License** column, check the check box for the license you want to add to your project (e.g. "TF4100 TC3 Controller Toolbox").

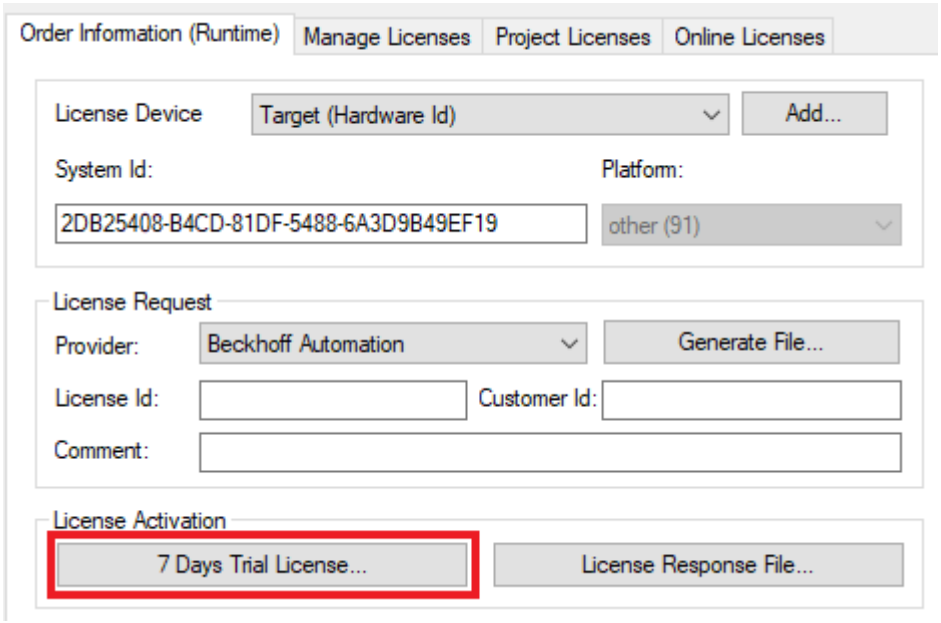


Order No	License	Add License
TF3601	TC3 Condition Monitoring Level 2	cpu license
TF3650	TC3 Power Monitoring	cpu license
TF3680	TC3 Filter	cpu license
TF3800	TC3 Machine Learning Inference Engine	cpu license
TF3810	TC3 Neural Network Inference Engine	cpu license
TF3900	TC3 Solar-Position-Algorithm	cpu license
TF4100	TC3 Controller Toolbox	cpu license
TF4110	TC3 Temperature-Controller	cpu license
TF4500	TC3 Speech	cpu license

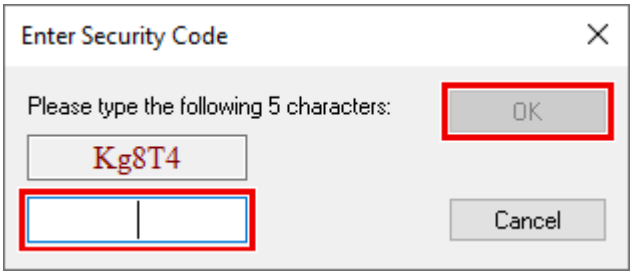
6. Open the **Order Information (Runtime)** tab.

   ⇨ In the tabular overview of licenses, the previously selected license is displayed with the status “missing”**.**

7. Click **7-Day Trial License...** to activate the 7-day trial license.



⇨ A dialog box opens, prompting you to enter the security code displayed in the dialog.



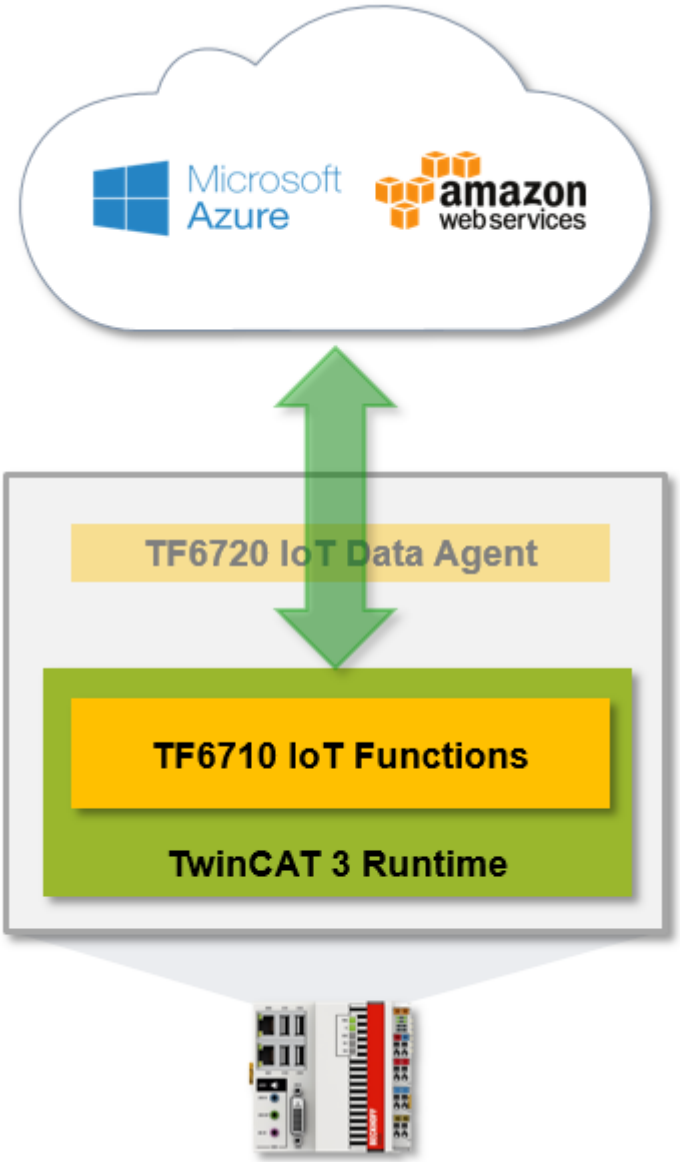
8. Enter the code exactly as it is displayed and confirm the entry.
9. Confirm the subsequent dialog, which indicates the successful activation.
   ⇨ In the tabular overview of licenses, the license status now indicates the expiry date of the license.
10. Restart the TwinCAT system.

⇨ The 7-day trial version is enabled.

4 Technical introduction

TC3 IoT Functions is a product for the TwinCAT 3 runtime that enables bi-directional data communication with the Cloud. To establish a connectivity channel with a cloud service, the product uses technical functionalities of the TC3 IoT Data Agent (TF6720) in the background. TC3 IoT Functions can therefore be used with every cloud service that is also supported by the TC3 IoT Data Agent.
Note that the TC3 IoT Data Agent is only used for providing the connectivity layer to the cloud and no TF6720 license has to be purchased to use TF6710.

TC3 IoT Functions and TC3 IoT Data Agent do not have to run on the same system, but can also run separately from each other on different systems (see Setup scenarios).



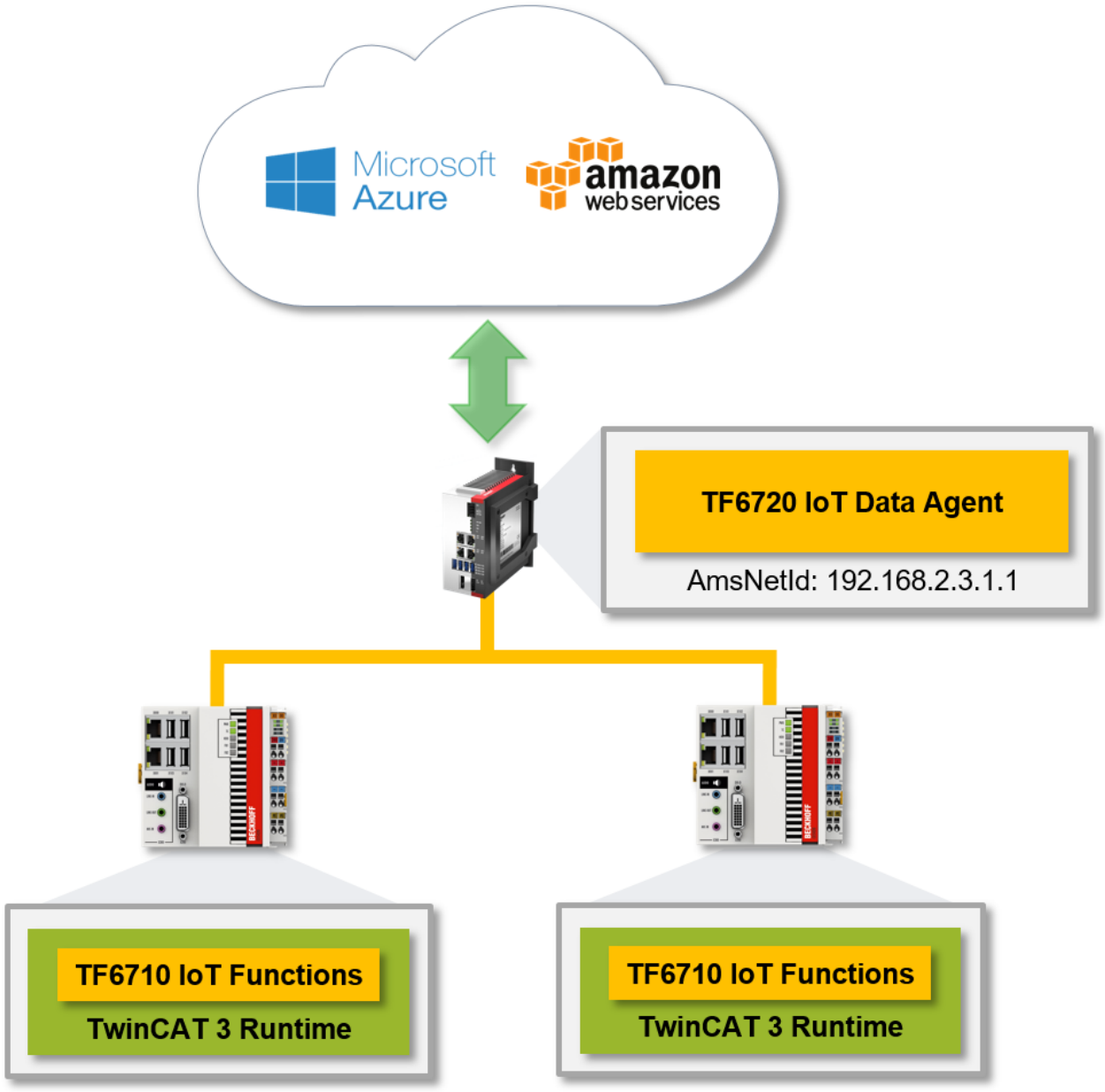
Features

TC3 IoT Functions includes the following features:

- Read/Write operations for messages that should be send/received to/from a cloud service
- Convenient timeout, error and retry handling

4.1 Reference Data Agent

If TC3 IoT Functions and TC3 IoT Data Agent are installed separately and the components are placed on different computers, the function blocks need to specify the location of the TC3 IoT Data Agent installation. This can be done via the AMS Net ID of the device that executes the TC3 IoT Data Agent. Simply enter the AMS Net ID of the device that executes TC3 IoT Data Agent in the corresponding input parameter of function block FB_IotFunctions_Connector [▶ 35]. Note that in this case an ADS route between the devices has to be created.



If TC3 IoT Data Agent and TC3 IoT Functions are running on the same device, the local AMS NET ID is used by default.

```
fbConnector : FB_IotFunctions_Connector := (sAmsNetId := 192.168.2.3.1.1');
```

4.2 Communication patterns

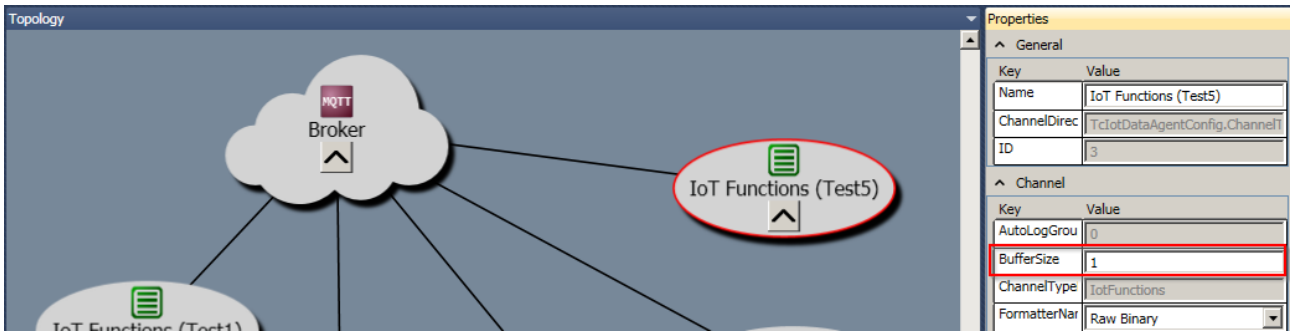
The upper connectivity layer to a cloud service is usually based on publisher/subscriber patterns but in some cases it can also be based on polling patterns. TC3 IoT Data Agent implements the connectivity with the cloud service and provides this access via an abstracted way of communication to TC3 IoT Functions. The PLC library of TC3 IoT Functions then uses this interface then via a polling read/write pattern.

Example

TC3 IoT Functions should be used to read messages from Azure IoT Hub. Connectivity with this cloud service is based on the publisher/subscriber pattern. This means that TC3 IoT Data Agent is configured with access credentials to the IoT Hub service and will therefore provides connectivity to it. TC3 IoT Functions then polls the TC3 IoT Data Agent for any incoming messages.

Message buffer

TC3 IoT Data Agent includes a message buffer for incoming messages that should be used by TC3 IoT Functions. This message buffer can be set directly on the corresponding TC3 IoT Functions channel.



4.3 Programming workflow

This section describes how to use the function blocks of the TC3 IoT Functions PLC library from a best practice point-of-view. The programming workflow includes the following steps:

- Invoking FB_IotFunctions_Connector.Execute() [▶ 16]
- Checking for general errors [▶ 16]
- Checking for read/write errors [▶ 17]
- Reading data [▶ 17]
- Writing data [▶ 17]

The code snippets in this section are based on the following declarations:

```
PROGRAM MAIN
VAR
    fbConnector : FB_IotFunctions_Connector;
    fbRead      : FB_IotFunctions_Message;
    fbWrite     : FB_IotFunctions_Message;
    nReadError  : UINT;
    nWriteError : UINT;
    nReadData   : UINT;
    nIn         : UINT;
    nOut        : UINT;
    bWrite      : BOOL;
END_VAR
```

Invoking FB_IotFunctions_Connector.Execute()

It is highly recommended to invoke the Execute method on the FB_IotFunctions_Connector function block instance as one of the first instructions. This method is responsible for Online Change handling, timeout handling and the communication with TC3 IoT Data Agent.

```
fbConnector.Execute();
```

Checking for general errors

Once the Execute method has been triggered, check the bError output of the FB_IotFunctions_Connector function block to figure out if any error has occurred during the communication with TC3 IoT Data Agent.

```
IF fbConnector.bError THEN
  ...
END_IF
```

Checking for read/write errors

If the bError output of the connector's function block instance is TRUE, check the individual error message of the request function block for errors to handle them properly. To acknowledge an error and to prevent it from recurring, call the Reset method of the function block.

```
IF fbConnector.bError THEN
  IF fbRead.bError THEN
    nReadError := nReadError + 1;
    fbRead.Reset();
  END_IF
  IF fbWrite.bError THEN
    nWriteError := nWriteError + 1;
    fbWrite.Reset();
  END_IF
END_IF
```

Before starting a new operation like read or write, check all relevant status queries, as these operations will reset all status information in the structures ST_IotFunctionsMessage and ST_IotFunctionsRequest.

Reading data

The Read operation is receiving data from the buffer and stores this data in a symbol. The value of the specified symbol is used to compare new data with previous data. When new data has been received, bDataAvailable is set to TRUE. If the current value of the symbol is different from the previous value, the data has changed and bDataChanged is set to TRUE.

This means, if you want to act on new data that differs from the previously received data, check the bDataChanged output. This output will only be set to TRUE if the receive buffer indicates a change to the previously received packet.

```
IF fbRead.bDataChanged THEN
  ...
END_IF
```

If you are interested in receiving data regardless of the data differing from the previously received payload, check the bDataAvailable output instead.

```
IF fbRead.bDataAvailable THEN
  ...
END_IF
```

Writing data

The following sample demonstrates how to set up a conditional write operation that will only be executed if the bWrite flag is set to TRUE. Before calling the Write method ensure that the function block is not busy. If you write to a busy function block instance, the call will return without starting the write operation.

```
IF bWrite THEN
  IF NOT fbWrite.bBusy THEN
    bWrite := FALSE;
    fbWrite.Write(ADR(nOut), SIZEOF(nOut));
  END_IF
 END_IF
```

4.4 Synchronizing message operations

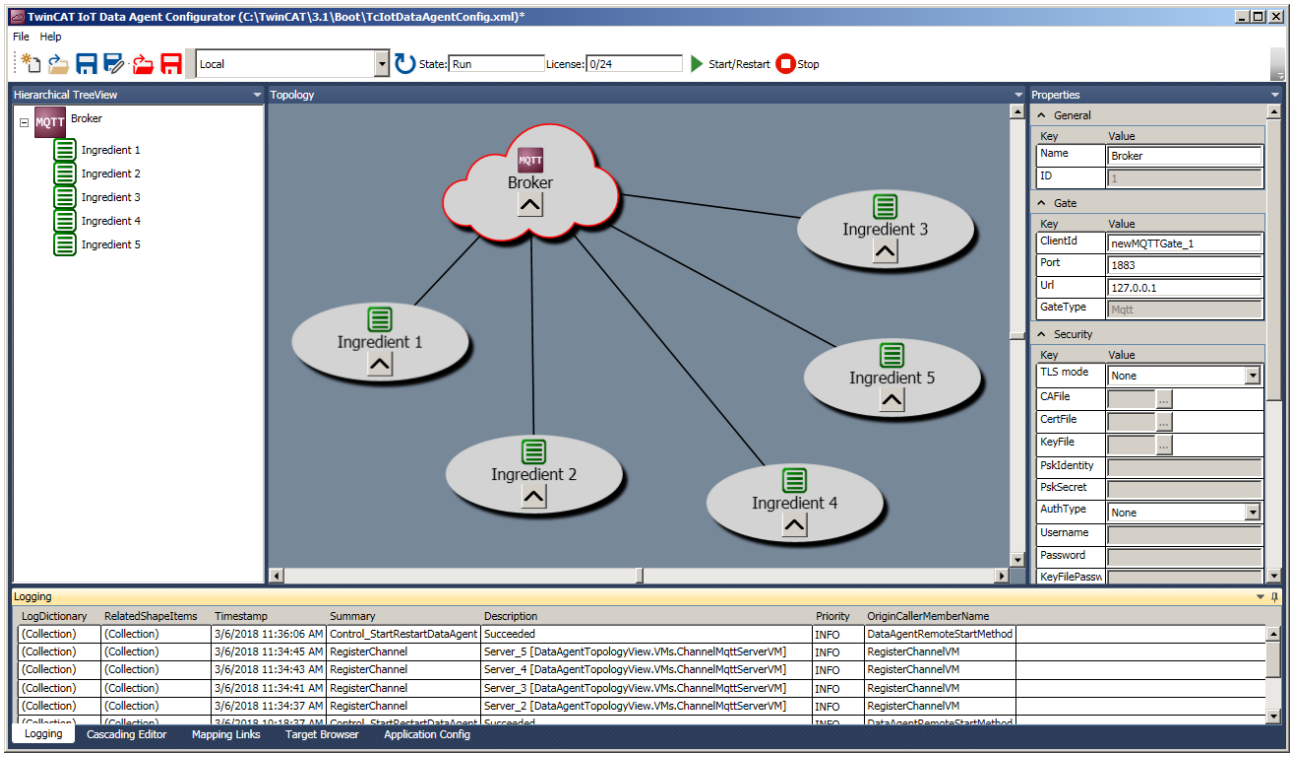
TC3 IoT Functions includes functionalities for synchronizing multiple message operations. This can be very useful in scenarios where data comes in from different data sources/channels. The function block FB_IotFunctions_Request [▶ 37] provides different mechanisms for synchronizing message operations.

Example scenario

A cocktail mixer provides different ingredients and every cocktail consists of five ingredients. An ingredient is added to the recipe by pressing a button. The cocktail mixer starts mixing the cocktail after the fifth ingredient has been selected. When an ingredient is selected, an MQTT message is published to a (different) topic. These messages should be received by TC3 IoT Functions.

Basic setup

TC3 IoT Data Agent is configured with an MQTT gate and five different channels for TC3 IoT Functions. Every channel is configured for an “ingredient topic”.



To synchronize the read operation for five different ingredients, an instance of FB_IotFunctions_Request is created. Then an instance of FB_IotFunctions_Message is created for each of the five channels.

```
fbRequestIngredients : FB_IotFunctions_Request;

fbReadIngredient : ARRAY[0..4] OF FB_IotFunctions_Message := [(nChannelId := 1),(nChannelId := 2),
(nChannelId := 3),(nChannelId := 4),(nChannelId := 5)];
nIn : ARRAY[0..4] OF STRING;
```

The synchronized request then can be created as follows:

```
IF NOT fbRequestIngredients.bBusy THEN
  IF fbRequestIngredients.bError THEN
    ...
  ELSE
    IF fbRequestIngredients.bTimeoutOccurred THEN
      ...
    ELSE
      // request was successful
      ...
    END_IF
  END_IF

  // Prepare next read operation for ingredients
  fbRequestIngredients.Create();
  fbRequestIngredients.EnqueueRead(ADR(fbReadIngredient [0]),ADR(nIn[0]),SIZEOF(nIn[0]));
  fbRequestIngredients.EnqueueRead(ADR(fbReadIngredient [1]),ADR(nIn[1]),SIZEOF(nIn[1]));
  fbRequestIngredients.EnqueueRead(ADR(fbReadIngredient [2]),ADR(nIn[2]),SIZEOF(nIn[2]));
  fbRequestIngredients.EnqueueRead(ADR(fbReadIngredient [3]),ADR(nIn[3]),SIZEOF(nIn[3]));
  fbRequestIngredients.EnqueueRead(ADR(fbReadIngredient [4]),ADR(nIn[4]),SIZEOF(nIn[4]));
  fbRequestIngredients.Execute();
END_IF
```

Sample04 shows the full sample code (see Samples [▶ 41]).

Synchronization conditions

The function block instance fbRequest contains different synchronization conditions. These can be used to determine whether the read operations within the request were successful, threw an error or resulted in a timeout. Note how the different timeout and retry settings can help in supporting this use case (see Timeout settings [▶ 19]).

Condition	Description
bBusy	TRUE: Request is still in progress and not all operations have been completed successfully FALSE: Request has been finished. To figure out if an error or timeout occurred, further flags need to be evaluated.
bTimeoutOccurred	TRUE: RequestTimeout has been triggered for at least one operation. FALSE: No timeout occurred.
bError	TRUE: An error occurred for at least one operation. FALSE: No error occurred.
Flags of each message operation	In addition, the flags (error, success, bDataAvailable, bDataChanged) of each message operation can be analyzed in order to figure out if a request operation was successful or failed.

4.5 Timeout settings

The function blocks of TC3 IoT Functions include several timeout settings that may help the user to handle errors regarding the retry operations. The following section explains the different timeout settings more detailed.

RequestTimeout

The RequestTimeout can be either set globally on an instance of FB_IotFunctions_Connector or individually on an instance of FB_IotFunctions_Message. Individual settings always override global settings. The RequestTimeout closely works together with the MessageRetryInterval setting.

The RequestTimeout specifies when a message operation (read/write) will time out. For example, if RequestTimeout is set to 10000, a read operation will time out after 10 seconds if no data has been received. If data is received within 10 seconds, the operation will finish immediately.

MessageRetryInterval

The MessageRetryInterval can be either set globally on an instance of FB_IotFunctions_Connector or individually on an instance of FB_IotFunctions_Message. Individual settings always override global settings. The MessageRetryInterval closely works together with the RequestTimeout setting.

The MessageRetryInterval specifies the time interval in [ms] when a message operation will be retried. The upper limit of the interval is always the RequestTimeout. For example, if RequestTimeout is set to 10000 and MessageRetryInterval to 1000, a read operation will be retried ten times before the RequestTimeout is triggered and the read operation times out.

CumulativeTimeout

The CumulativeTimeout can be set on instances of FB_IotFunctions_Request when synchronizing message operations (see Synchronizing message operations [▶ 17]).

The use case is as follows (example):

- There are three read operations that should be synchronized.
- Every read operation comes from a different channel

- RequestTimeout is globally set to 10000 ms
- CumulativeTimeout is globally set to 3000 ms
- After 8000 ms the first message comes in via channel 1.
- Since only 2000 ms remain until the RequestTimeout time is reached, the CumulativeTimeout is added to the remaining RequestTimeout time, to prevent a time out of the whole request. The RequestTimeout time is then 5000 ms. This gives the request more time to gather data via the other two read operations. If there is no data on the other two operations after 5000 ms, the whole request will time out.

5 Configuration

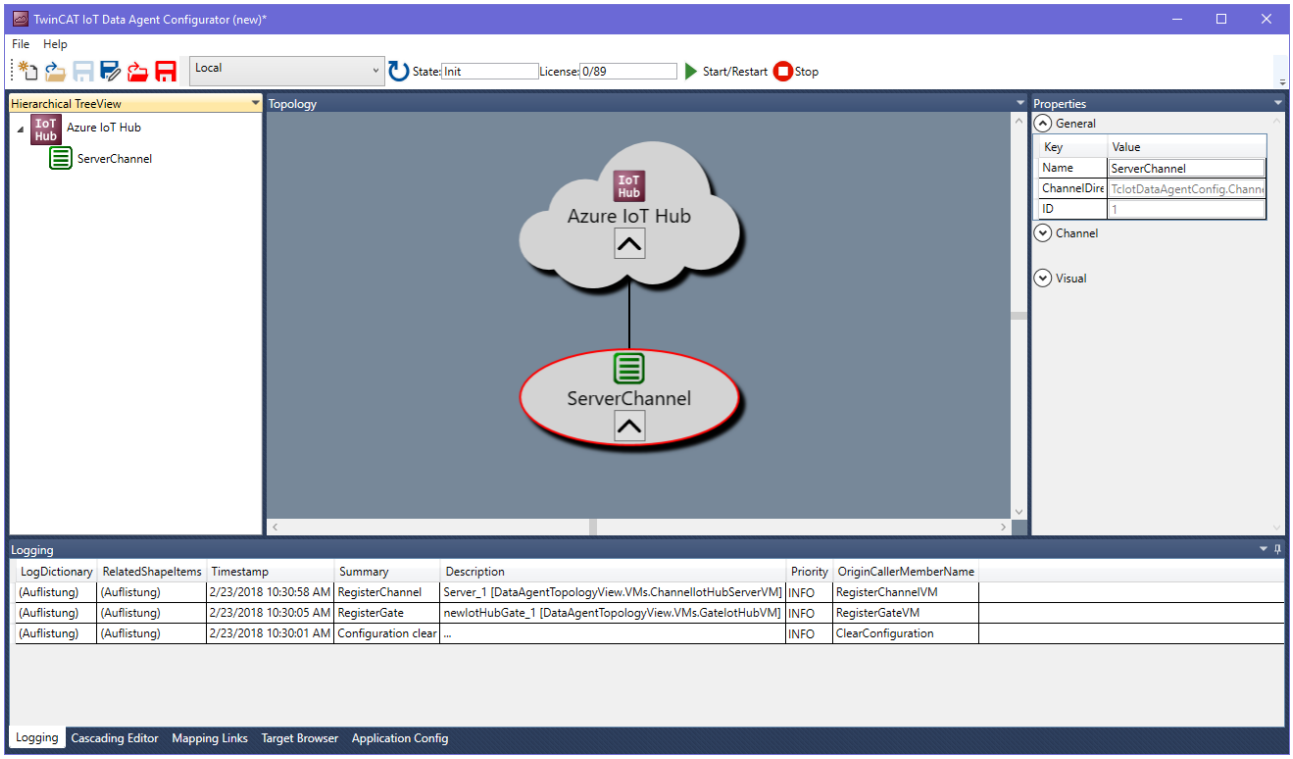
5.1 Overview

TC3 IoT Functions uses TC3 IoT Data Agent functionalities in order to connect to cloud services. To configure access to a cloud service and provide access credentials, the TC3 IoT Data Agent configurator can be used.

Within the configurator, the following configuration steps are required:

1. Create a gate (e.g. Azure IoT Hub) and configure all required connection credentials.
2. Create a server channel on the new gate and note the channel ID.
3. Activate the configuration and start TC3 IoT Data Agent.

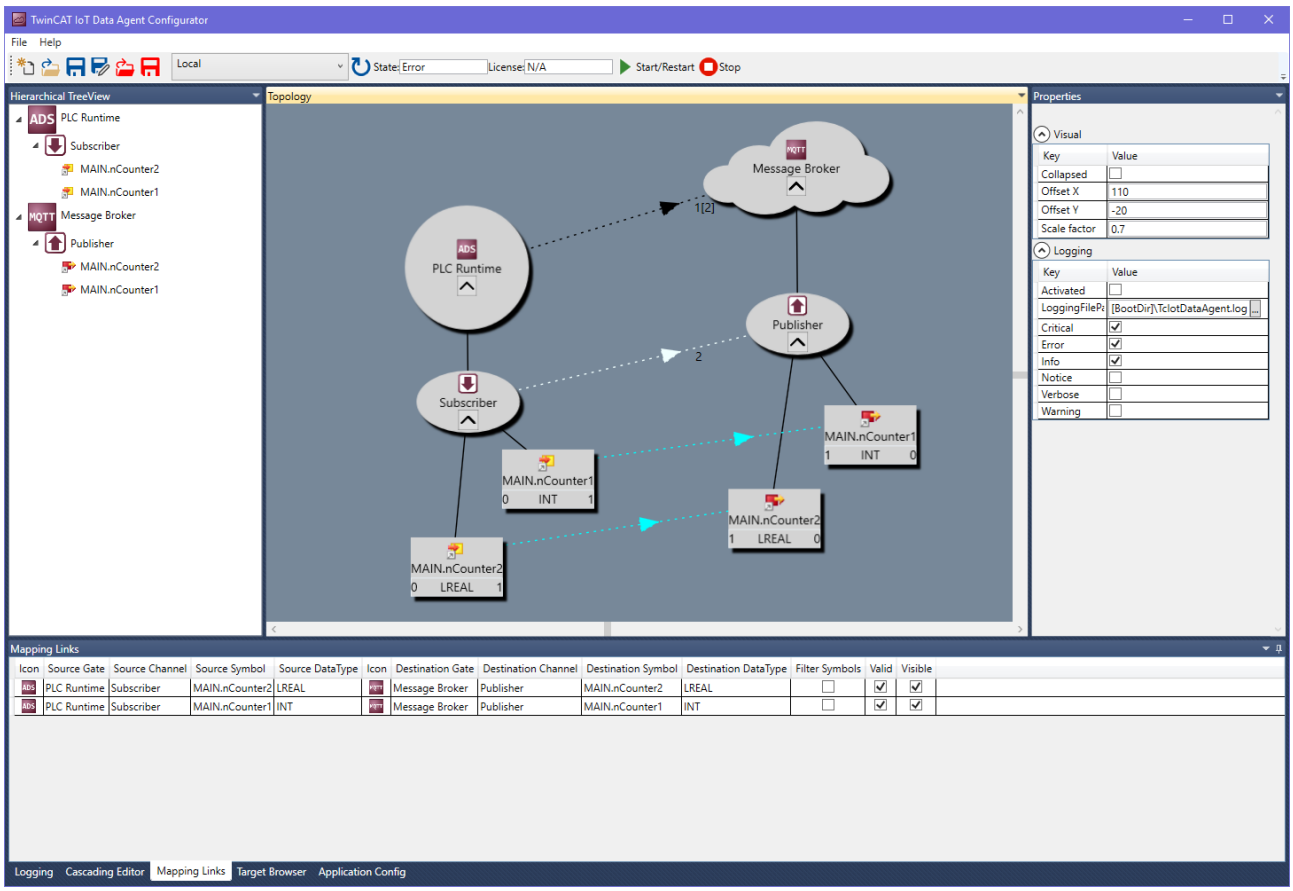
In order to use this configuration in TC3 IoT Functions, reference the channel ID in the function block FB_IotFunctions_Message.



5.2 Configurator

The TC3 IoT Data Agent configurator is an easy-to-use, graphical user interface that abstracts the XML configuration file and provides a modern interface that includes all functionalities to easily configure symbols that should be send to or received from a cloud service.

The configurator is also used to configure TC3 IoT Functions.



Standard components

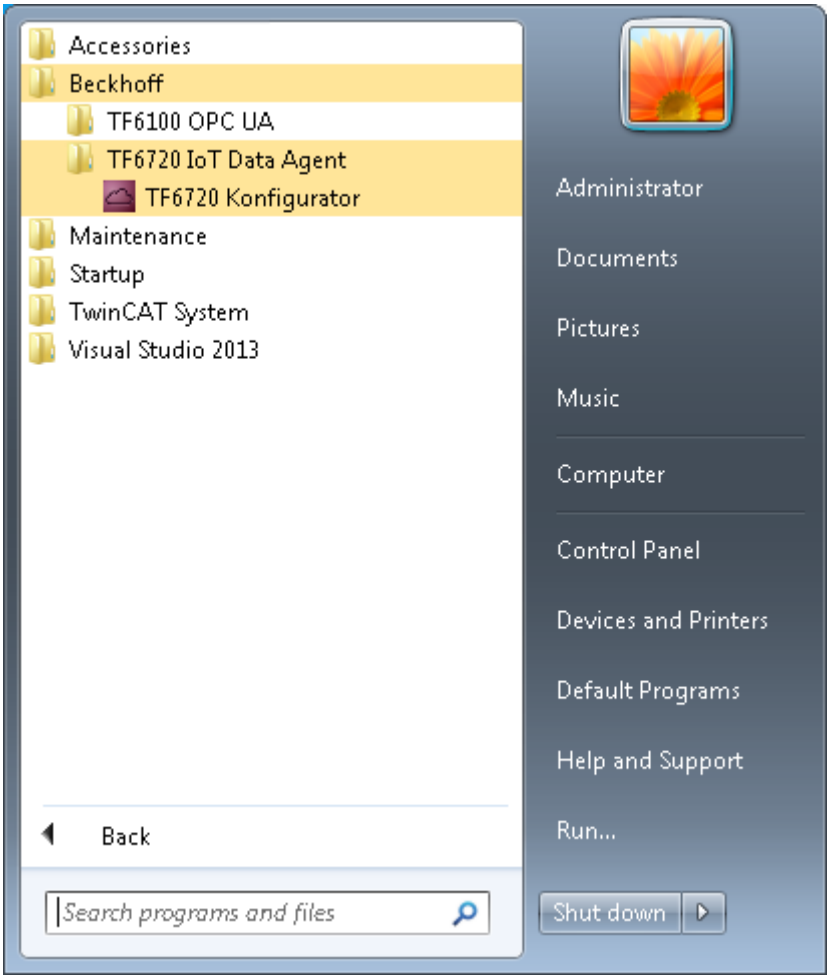
The TC3 IoT Data Agent configurator consists of the following areas:

Menu and toolbar	Provides commands for saving and opening files and for starting and stopping the application
Hierarchical TreeView	Gives a hierarchical overview of the configuration to create and edit a configuration
Topology view	Gives a graphical overview of the configuration to create and edit a configuration
Properties window	Shows the properties of an activated component in the topology or tree view.
Logging	Provides log information from the configurator.
Cascading Editor	Helps to navigate through large and complex navigations by providing filtering mechanisms for symbols
Mapping Links	Gives an overview of all links between symbols in the configuration
Target Browser	Is used to provide symbolic access for ADS and OPC UA target runtimes

Installation

The configurator is automatically installed by the setup and is available as a shortcut on the Windows start menu.

When starting the configurator the first time, it asks for the generation of an OPC UA client certificate. This certificate is used by the configurator in its integrated OPC UA target browser to connect to a server and browse its namespace. After the certificate has been generated, the configurator UI is shown.



5.2.1 Topology view

The topology view (or “canvas”) is the central graphical configuration area of the TC3 IoT Data Agent configurator. It shows the following components of a configuration:

- The configured gates, channels and symbols
- The relationship (mapping) between gates, channels and symbols
- The cardinality on each relationship

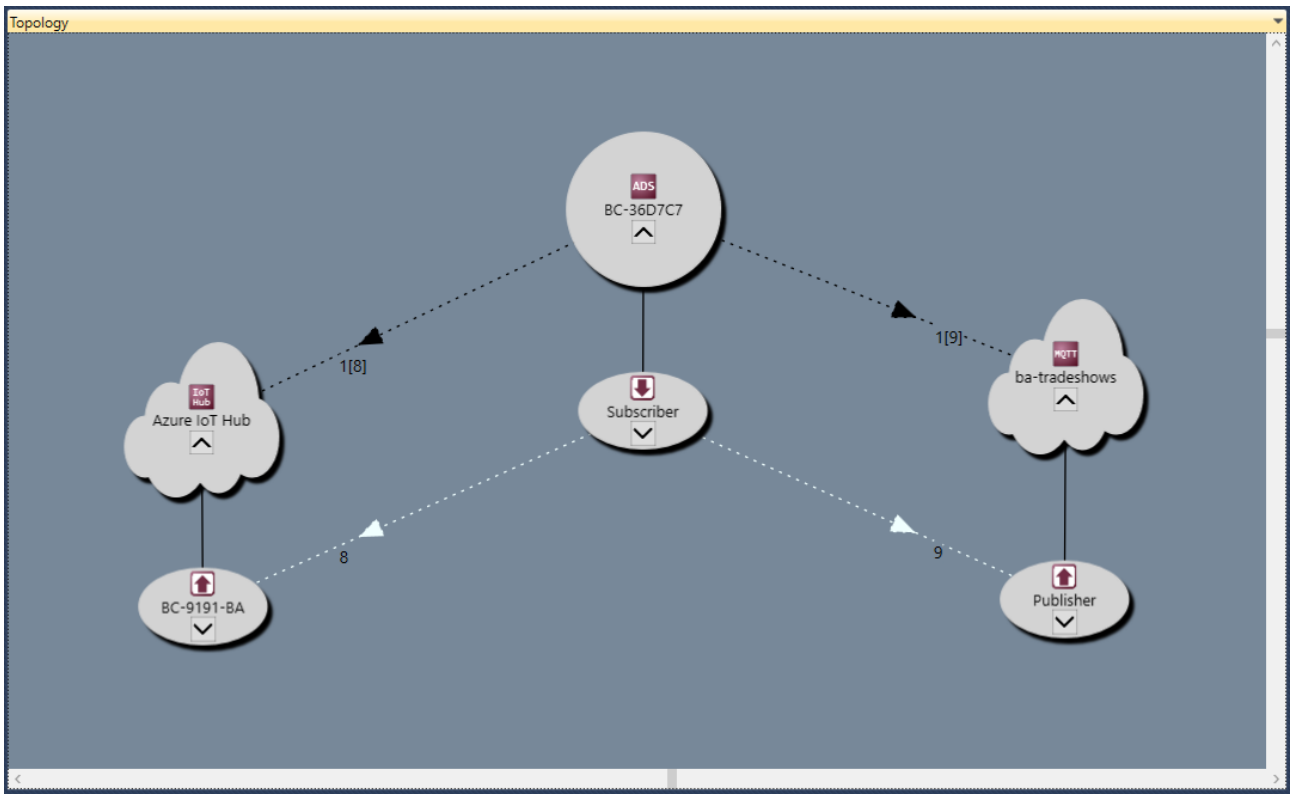
The topology view can be used to create and edit a configuration. Simply right-click the canvas to open the context menu and select from the variety of configuration options, e.g. to create a new gate, attach a new channel to a gate or remove a component from the configuration.

You can move any object in the topology view by dragging it to a new position. Each attached subcomponent is moved together with its parent. Optionally, you can also hide a subcomponent by clicking the expand button of its parent. When saving a configuration, the position of each object is saved in the configuration file.

In order to make navigational tasks a little easier, the topology view supports the following functionalities:

- Scrollbars for vertical and horizontal navigation
- Vertical navigation via mouse wheel
- Horizontal navigation via mouse wheel and SHIFT key (press and hold)
- ZoomIn/ZoomOut via mouse wheel and CTRL key (press and hold)

Instead of the topology view you can also use the tree view for configuration, but the topology view might give a better graphical overview of the currently configuration (see Tree view [▶ 24])

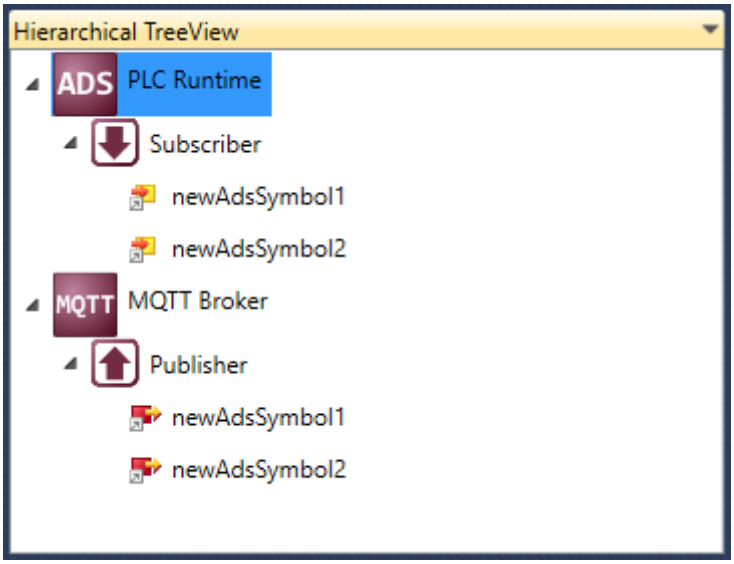


5.2.2 Tree view

The tree view provides a hierarchical view of the currently opened configuration. It shows the following components of a configuration:

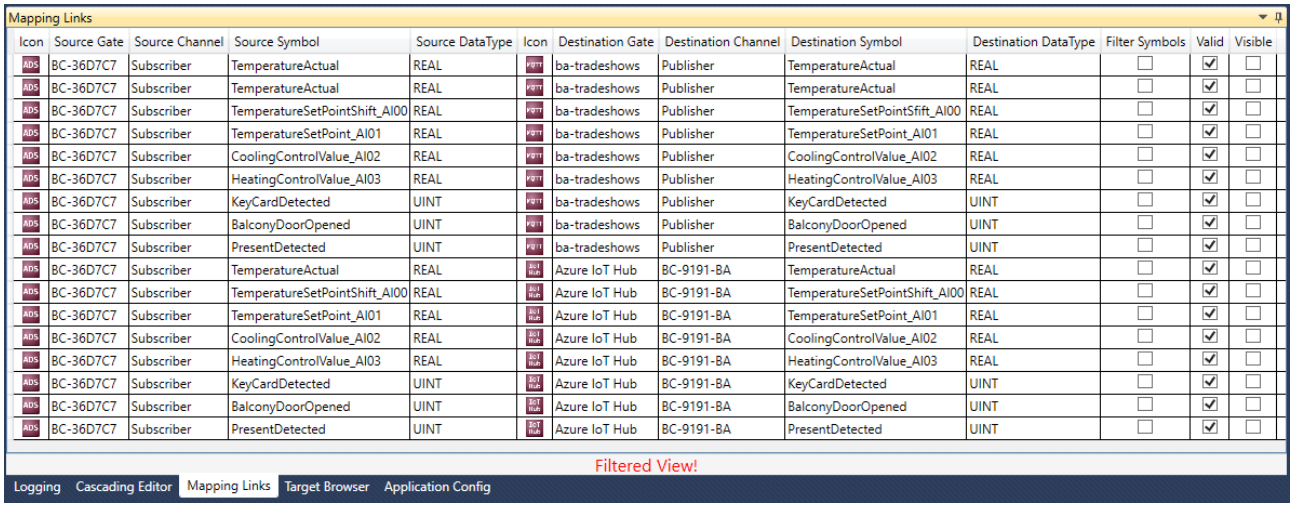
- The configured gates, channels and symbols
- The existence of a relationship (mapping) between symbols

The tree view can also be used to edit the configuration, but you might find it easier to use the topology view instead (see Topology view [▶ 23]).



5.2.3 Mappings

The mappings window gives an overview of all links between symbols in the currently opened configuration. When a link is selected, it is automatically highlighted in the topology view.



5.2.4 Target Browser

The Target Browser is used to provide symbolic access for ADS and OPC UA target runtimes. It can be used to configure symbols for a target runtime by drag-and-drop.

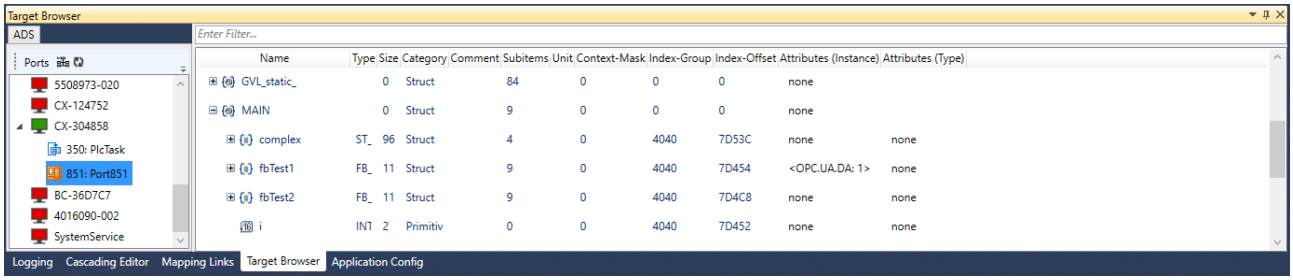
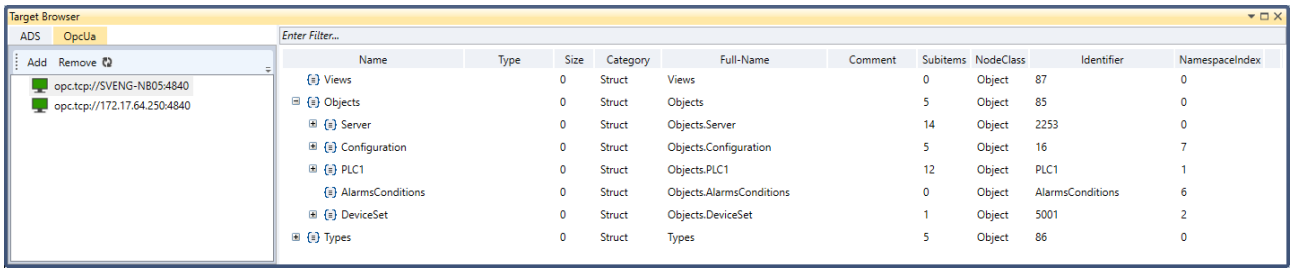
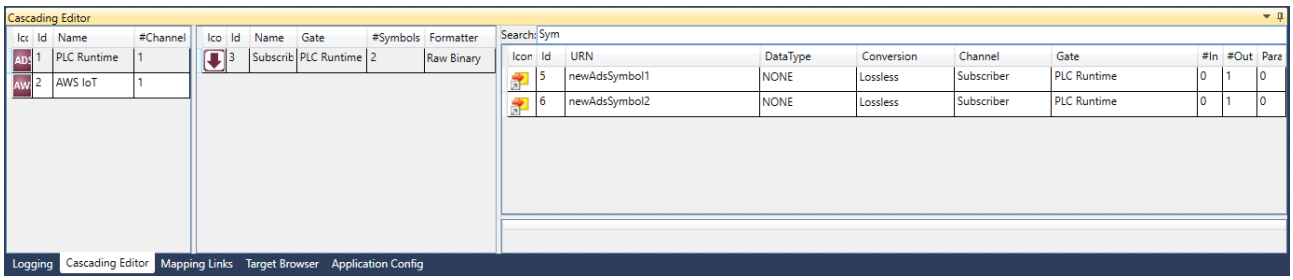


Fig. 1:



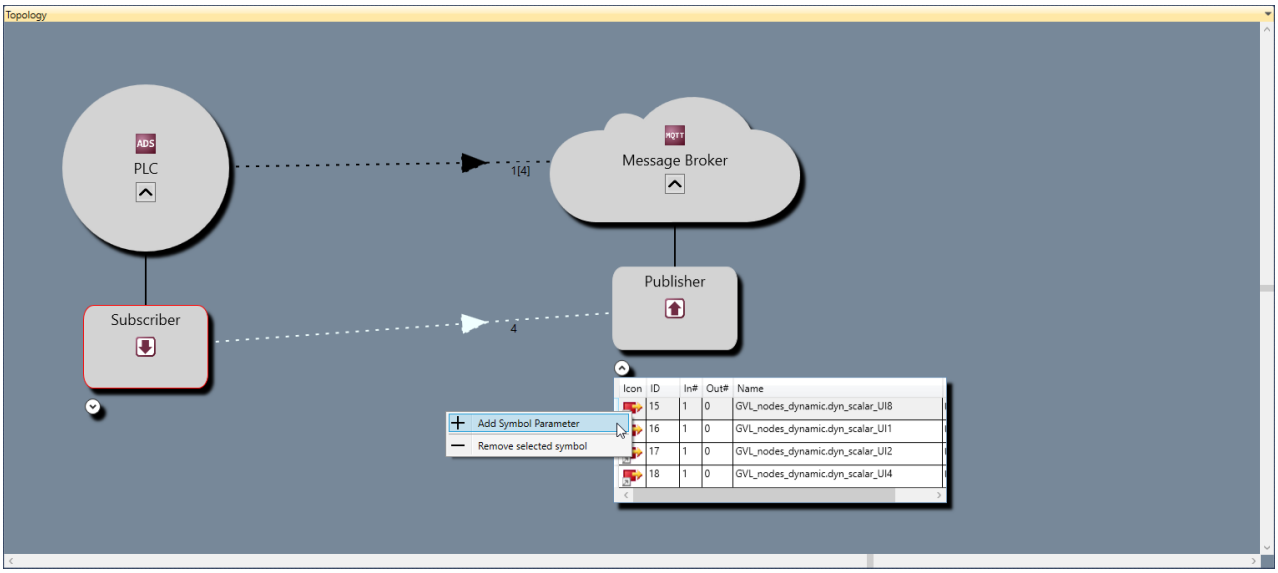
5.2.5 Cascading Editor

The Cascading Editor helps to navigate through large and complex configurations by providing filtering mechanisms for symbols. Starting from left to right you can select gates and channels in order to display the corresponding symbols. In addition, you can also search for symbol names using free text. When a component is selected, it is automatically highlighted in the topology view.

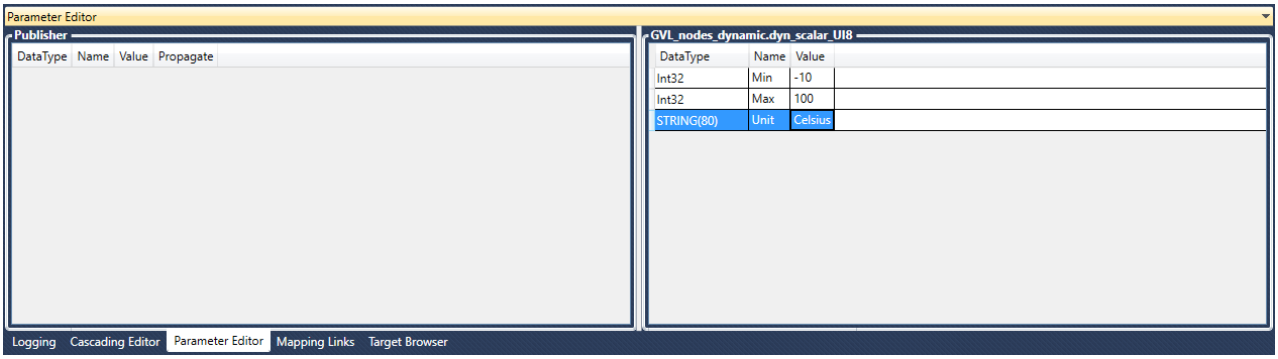


5.2.6 Parameter Editor

The Parameter Editor allows to configure symbol meta data for symbols in a publisher channel. As a prerequisite, the publisher channel has to be configured to use the "TwinCAT JSON" data format. You can then add new symbol parameters by right clicking a symbol and select **Add Symbol Parameter**.



This will add a new symbol parameter entry to the Parameter Editor. You can then specify the data type, name and value of the new parameter.

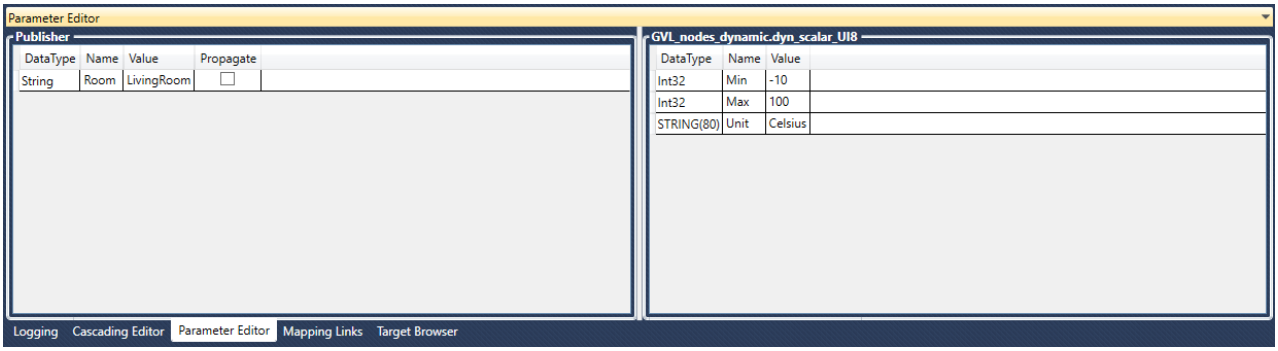


A result of the above configuration may look as follows when using the TwinCAT JSON data format. In this configuration, the variable "GVL_nodes_dynamic.dyn_scalar_UI8" has been configured with the symbol parameters Min, Max and Unit. These parameters will be added to the "MetaData" section of the TwinCAT JSON formatted message.

```
{
  "Timestamp": "2020-09-28T12:32:49.0370000+02:00",
  "GroupName": "Publisher",
  "Values": {
    "GVL_nodes_dynamic.dyn_scalar_UI8": 259096147,
    "GVL_nodes_dynamic.dyn_scalar_UI1": 83,
    "GVL_nodes_dynamic.dyn_scalar_UI2": 32339,
    "GVL_nodes_dynamic.dyn_scalar_UI4": 259096147
```

```
  },
  "MetaData": {
    "GVL_nodes_dynamic.dyn_scalar_UI8": {
      "Timestamp": "2020-09-28T12:32:49.0350000+02:00",
      "Min": -10,
      "Max": 100,
      "Unit": "Celsius"
    },
    "GVL_nodes_dynamic.dyn_scalar_UI1": {
      "Timestamp": "2020-09-28T12:32:49.0350000+02:00"
    },
    "GVL_nodes_dynamic.dyn_scalar_UI2": {
      "Timestamp": "2020-09-28T12:32:49.0350000+02:00"
    },
    "GVL_nodes_dynamic.dyn_scalar_UI4": {
      "Timestamp": "2020-09-28T12:32:49.0350000+02:00"
    }
  }
}
```


In addition, you can also add parameters to a channel, which will add meta data properties to the root message. Example:



As a result, the root message now contains a static meta data property called "Room", which has the value "LivingRoom":


```
{
  "Timestamp": "2020-09-30T09:55:58.1160000+02:00",
  "GroupName": "Publisher",
  "Room": "LivingRoom",
  "Values": {
    "GVL_nodes_dynamic.dyn_scalar_UI8": 267266470,
    "GVL_nodes_dynamic.dyn_scalar_UI1": 166,
    "GVL_nodes_dynamic.dyn_scalar_UI2": 10662,
    "GVL_nodes_dynamic.dyn_scalar_UI4": 267266470
    },
  "MetaData": {
    "GVL_nodes_dynamic.dyn_scalar_UI8": {
      "Timestamp": "2020-09-30T09:55:58.1140000+02:00",
      "Min": -10,
      "Max": 100,
      "Unit": "Celsius"
    },
    "GVL_nodes_dynamic.dyn_scalar_UI1": {
      "Timestamp": "2020-09-30T09:55:58.1140000+02:00"
    },
    "GVL_nodes_dynamic.dyn_scalar_UI2": {
      "Timestamp": "2020-09-30T09:55:58.1140000+02:00"
    },
    "GVL_nodes_dynamic.dyn_scalar_UI4": {
      "Timestamp": "2020-09-30T09:55:58.1140000+02:00"
    }
  }
}
```


5.2.7 Settings

This section provides some detailed information about the different configuration parameters that can be set on gates, channels and symbols.

5.2.7.1 Gates

A Gate represents a communication protocol or specific connectivity service, e.g. ADS, OPC UA, MQTT, AWS IoT or Microsoft Azure IoT Hub. Each Gate is configured with parameters that are specific for the corresponding Gate type.

ADS

A Beckhoff ADS device represents either a TwinCAT 2/3 or a Beckhoff BC device. ADS is the common Beckhoff communication protocol and can be used to access many parts of the TwinCAT system. The most common application scenarios for TC3 IoT Data Agent involve access to a TwinCAT 2/3 PLC, C++, TcCOM modules or the I/O process image, which is done via ADS.

When configuring an ADS gate, the following settings are required by the TC3 IoT Data Agent to access the underlying ADS device:

Setting	Description
AmsNetId	AmsNetId of the target device, e.g. 127.0.0.1.1.1 for the local device.
AdsPort	AdsPort of the target device, e.g. 801 (TwinCAT 2 PLC) or 851 (TwinCAT 3 PLC)
IoMode	Specifies how the TC3 IoT Data Agent should communicate with the ADS device. The following options can be set: • Direct: accesses every symbol with a separate ADS command. Mandatory for BC controllers but increases ADS traffic • Batched: accesses symbols batched into an ADS sum command, which optimizes ADS traffic and can be used for TwinCAT 2 and 3 PLC or C++ runtimes

OPC UA

OPC UA is a standardized, industrial, client/server communication protocol and adopted by many vendors for different use cases. The TC3 IoT Data Agent can access OPC UA server devices to connect variables (so-called “nodes”) on those devices with IoT services.

When configuring an OPC UA gate, the following settings are required by the TC3 IoT Data Agent to access the underlying OPC UA device:

Setting	Description
Server URL	The OPC UA Server URL, e.g. opc.tcp://localhost:4840
Security policy	The OPC UA security policy that the TC3 IoT Data Agent should use during connection establishment with the OPC UA server
Security mode	The OPC UA message security mode that the TC3 IoT Data Agent should use during connection establishment with the OPC UA server
Authentication mode	The OPC UA authentication mode that the TC3 IoT Data Agent should use during connection establishment with the OPC UA server

MQTT

MQTT can be used for connecting to a generic message broker, e.g. Mosquitto, HiveMQ or similar broker types.

Setting	Description
Broker address	The IP address or hostname of the MQTT message broker
Port	MQTT specifies port 1883 for unencrypted communication and 8883 for encrypted communication
ClientId	A numeric or string-based value that identifies the client. Depending on the message broker type, this ID should be unique
Authentication mode	Specifies if the TC3 IoT Data Agent should authenticate to the broker by using a username/password combination
TLS mode	Specifies if TLS should be used to secure the communication channel to the message broker. Note that the message broker also needs to support TLS in order for this to work. Different options are available for TLS: • CA certificate: Only uses the CA certificate for server authentication • Client certificate: Uses a client certificate for mutual client/server authentication • PSK: Used a common PSK-Identity and PSK-Key that is known to the message broker and the client

Microsoft Azure IoT Hub

With Azure IoT Hub the Microsoft Azure cloud platform offers a connectivity service in the cloud that provides bi-directional communication, device security and automatic scalability. In the TC3 IoT Data Agent, the IoT Hub can be configured as a special gate type.

Setting	Description
HostName	URL of the Azure IoT Hub instance
DeviceId	DeviceId of the device that has been created on the IoT Hub configuration website
SharedAccessKey	Either the primary or secondary device key that has been generated together with the device on the IoT Hub configuration website
CA file	CA file that is used for server authentication. At the time writing this article, three certificates are in play during the server authentication, which form part of the certificate chain and are linked together. • Root CA: Baltimore CyberTrust Root • Intermediate CA: Baltimore CyberTrust • Wilcard certificate During the initial TLS handshake, only the first two are sent by the server to the client. The client will normally only validate the Root CA of the chain and will determine if it is trusted. In order to acquire the CA file for the Root CA, you can open the Windows certificate store (certmgr.msc), browse to the trusted root authorities and export the Baltimore CyberTrust Root certificate.

AWS IoT

With AWS IoT the Amazon Web Services cloud platform offers a message broker service in the cloud that provides bi-directional communication, device security and automatic scalability. In the TC3 IoT Data Agent, AWS IoT is configured as a regular MQTT gate. However, some special MQTT settings have to be configured to successfully connect to an AW IoT instance.

Setting	Description
Broker address	The URL of the AWS IoT instance
Port	Port 8883 for encrypted TLS communication is mandatory
ClientId	Can be set to anything but needs to be unique. Typically this could be the AWS IoT thing name.
Authentication mode	Set to “No authentication”. Authentication on AWS IoT is done via the TLS client certificate.
TLS mode	Automatically set to “Client certificate” by the configurator. Select path to CA file, client certificate file and client key file. These are the files that can be generated and downloaded on the AWS IoT configuration website.

5.2.7.2 Channels

Channels are configured on a Gate to send (“publisher”) or receive (“subscriber”) data to/from a Gate.

- Source: This Gate is the source of data, meaning the TC3 IoT Data Agent connects to the Gate and retrieves data from it in order to send this data somewhere else (to a “Destination Gate”). Technically, this is also referred to as the “Subscriber Channel”.
- Destination: This Gate is the target of data, meaning the TC3 IoT Data Agent connects to the Gate and sends data to it, which has been acquired from another Gate (from a “Source Gate”). Technically, this is also referred to as the “Publisher Channel”.

Every channel has different settings that either describe the data format that should be used for this channel or the sampling settings that the TC3 IoT Data Agent should use for gathering data. These settings can also depend on the gate type that the channel has been configured for.

The following tables list all available settings.

Setting	Description	Applicable to Gate type
Direction	Sets whether the channel should either be a publisher (sender) or subscriber (receiver) channel. Depending on the selection and gate type, other settings are required or pre-selected.	All Gates

Setting	Description	Applicable to Gate type
SamplingMode	Selects if the channel should use either cyclic or event-based sampling mechanisms when gathering data from a source. Note that, depending on the direction, not all Gates might support both types. An MQTT Gate, for example, uses when receiving data always SamplingMode “event” (because of the Pub/Sub principle it is always event-based).	All Gates
CycleTime	Only applicable for SamplingMode “cyclic”. Sets the sampling rate in [ms].	All Gates
Timeout	The timeout for a communication with the Gate in [ms].	All Gates
PartialUpdate	Enable/Disable partial updates on this channel. When enabled (default), a publish includes only the updated symbol. When disabled, a publish includes all symbols of a channel with their last known value. Only applicable to publisher channels.	All Gates
BufferSize	Sets the size (amound of messages) of the ringbuffer on connection loss.	MQTT, AWS IoT, Azure IoT Hub

Setting	Description	Applicable to Gate type
Formatter	Sets the data format that should be used for this channel, e.g. binary or JSON. Note that some Gates require their Channels to use a fixed data format, e.g. ADS or OPC UA Gates, because the communication with those devices requires a specific format. In this case the Formatter is fixed and not changeable via the configurator.	MQTT, AWS IoT, Azure IoT Hub
FormatterType	Sets the formatter type on this channel. In most cases the formatter type is an InOut type. Consult our documentation article about writing custom plugins via the formatter interface for more information about this setting.	MQTT, AWS IoT, Azure IoT Hub

Setting	Description	Applicable to Gate type
Topic	Specifies the MQTT topic that should be used to publish or subscribe to.	MQTT
QoS	Specifies the QoS (Quality-of-Service) level that should be used when publishing or subscribing.	MQTT, AWS IoT
Retain	Specifies if a message should be send as retain. (only relevant for publisher channel)	MQTT
SendStateInfo	When activated, the TC3 IoT Data Agent publishes its “OnlineState” to the sub topic /Desc/ as well as uses this sub topic in its LastWill. When the TC3 IoT Data Agent connects to the Message Broker, a JSON message will be published to this topic, which contains { “OnlineState” : true } When the TC3 IoT Data Agent disconnects orderly from the Message Broker, the following message is sent to this topic: { “OnlineState” : false } When the Message Broker detects, that the TC3 IoT Data Agent lost connection, the following message is sent to this topic (LastWill): { “OnlineState” : false }	MQTT, AWS IoT

Sampling Modes

The TC3 IoT Data Agent includes different sampling modes that influence the way data is acquired from a source or written to a destination. The sampling mode can be set on a channel. The following sampling modes are currently available:

- Cyclic
- OnChange
- TriggerSymbol

Cyclic

Cyclic sampling means that the TC3 IoT Data Agent is cyclically sampling the gate for data (subscriber channel) or cyclically writing data to a gate (publisher channel). On a subscriber channel, this results in cyclic read commands whereas on a publisher channel this results in cyclic write commands, e.g. on an ADS or OPC UA gate. On gate types that are based on publisher/subscriber concepts, e.g. MQTT, AWS IoT and Azure IoT Hub gates, cyclic requests on a subscriber channel are automatically replaced by subscriptions whereas on a publisher channel this results in cyclic publish commands.

OnChange

OnChange sampling means that the TC3 IoT Data Agent only communicates data with a gate if the value of a variable has changed.

Trigger symbols

Trigger symbols enable “on demand” sampling, e.g. if a certain condition for a specified symbol (the so-called “trigger symbol”) is fullfilled. Different condition types can be set. They are configured as part of a channel and allow to specify the following condition types.

Condition type	Description
EQ	Trigger symbol value equals a given value
NE	Trigger symbol value differs from a given value
LE	Trigger symbol value is less than or equals a given value
GE	Trigger symbol value is greater than or equals a given value
LT	Trigger symbol value is less than given value
GT	Trigger symbol value is greater than given value

If the condition is fullfilled, all symbols in that channel are published to the corresponding gate. In addition you can specify how often the symbol values should be send.

Send behavior	Description
risingEdge	Symbols are only send once when the condition is fullfilled
continuous	Symbols are send as long as the condition is fullfilled

Usage of trigger symbols

Trigger symbols can only be configured for ADS and OPC UA subscriber symbols.

- Add the symbol that should act as trigger symbol to the ADS or OPC UA subscriber channel.
- Configure the linked publisher channel with SamplingMode “OnTrigger” to specify the previously added symbol as trigger symbol.

5.2.7.3 Symbols

Symbols represent variables from a gate, e.g. a TwinCAT PLC variable. The symbol configuration includes the address information that the Data Agent requires in order to read a symbol’s value or write to it. This address information is therefore depending on the gate type.

On gates that support a target browser [▶ 25], the browser will automatically detect the correct address information for a symbol. For all other gates, this information needs to be entered manually.

Gate type	Setting	Description
ADS	URN	Symbol name address information of the ADS variable. Might not work with all ADS devices, e.g. BC devices do not support symbol names.
ADS	IndexGroup IndexOffset	IndexGroup/IndexOffset combination can be used to access data of an ADS device that does not support symbolic address information. In case of the TwinCAT PLC, the IndexGroup/IndexOffset combination directly represents a memory address, e.g. from a PLC variable, which may change after a re-compilation of the TwinCAT project. It is therefore common practice to use the TwinCAT PLC symbol server instead, which provides symbolic information for its PLC variables, which means that a variable can be accessed via its symbol name instead, which stays valid even after a re-compilation or online change (if the symbol still exists). However, some ADS services do not include such a symbol server, e.g. small Beckhoff BC devices. In those cases the IndexGroup/IndexOffset combination needs to be used.
ADS	DataType	Data type of the symbol
ADS	Conversion	Specifies conversion mode for this symbol.
OPC UA	Name	Descriptive name of the OPC UA node. Only used in Configurator, does not represent any online address information.
OPC UA	Identifier	Identifier of OPC UA symbol on the server, e.g.: • s = MAIN.nCounter (if the IdentifierType is “String”) • n = 42 (if the IdentifierType is “Numeric”)
OPC UA	NsName	Namespace name in which the symbol is located. This corresponds to the namespace index, which is part of an OPC UA NodeId. The translation can be done via the NamespaceArray.
OPC UA	AttributeId	The AttributeId defines the OPC UA attribute that should be used by the Data Agent when reading a symbol. In most scenarios, this is the “Value” of a symbol.
OPC UA	Conversion	Specifies conversion mode for this symbol.
MQTT	URN	Name of the MQTT symbol. This represents the name that is being used in the JSON format as a key, also when receiving data from the MQTT Gate.
MQTT	DataType	Data type of the symbol
MQTT	Conversion	Specifies conversion mode for this symbol.
IoT Hub	URN	Name of the IoT Hub symbol. This represents the name that is being used in the JSON format as a key, also when receiving data from IoT Hub.
IoT Hub	DataType	Data type of the symbol
IoT Hub	Conversion	Specifies conversion mode for this symbol.

When configuring a Channel, symbols can be added via a target browser or manually by providing the correct address information. Note that not all gate types include target browser functionalities. In this case symbols need to be configured manually.

Manual symbol configuration

If the target device is not online or does not provide any symbolic address information (e.g. the BC9191), symbols may also be added manually by entering the symbol address. The tables at the beginning of this document show which information is required in this case.

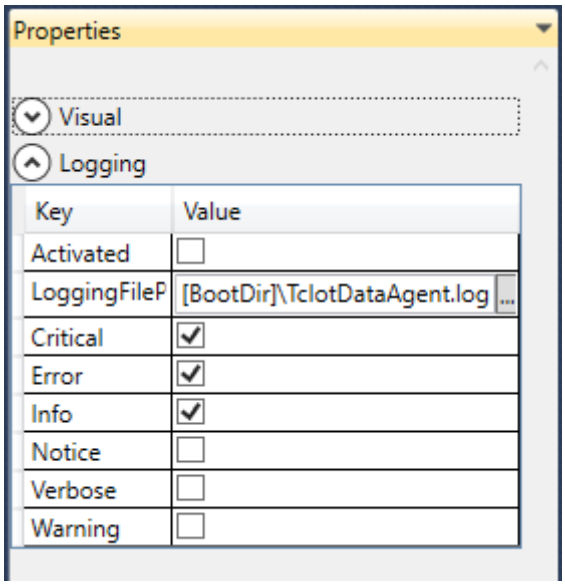
Type conversion

The TC3 IoT Data Agent supports data type conversion before the data is published to a Gate. Type conversion occurs on symbol level, which means that different Symbols can use different conversion modes. The following table shows the different conversion modes available.

Conversion mode	Description
Losless	Default setting. “Smaller” types can be converted into “larger” types. For example, a subscriber symbol of data type INT can be published to a symbol of data type Int32 (2 byte to 4 byte).
Lossy	If required, “larger” symbols can also be converted into “smaller” symbols. For example, a subscriber symbol of data type DINT can be published to a symbol of data type Int16 (4 byte to 2 byte). Depending on the value of the subscriber symbol, this can of course result in cut-off values.
Strict	If required, symbols can also be configured to use “strict mode”. In this conversion mode, the data type size of a subscriber symbol needs to match exactly the data type of the mapped publisher symbol. For example, a subscriber symbol of data type INT can only be published to a symbol of data type Int16 (2 byte to 2 byte).

5.2.8 Error logging

For troubleshooting purposes, the TC3 IoT Data Agent can generate a logfile, which can be populated based on different logging levels. All necessary settings can be configured via the properties window of the configurator. Simply click on an empty spot on the topology view and configure the logging settings in the properties window.



Note that all settings are tied to the currently opened configuration and have to be set individually for every configuration. The default directory in which the logfile is created, is the TwinCAT boot directory. The placeholder [BootDir] automatically selects the TwinCAT boot directory.

Logging window

The logging window inside of the configurator is only showing log events that are related to the configurator. In order to create a log for the TC3 IoT Data Agent background service, the settings above are required.

6 PLC API

6.1 Function blocks

6.1.1 FB_IotFunctions_Connector

The function block enables communication with a local/remote TC3 IoT Data Agent installation. The Execute method of the function block must be called cyclically in order to ensure the background communication with the TC3 IoT Data Agent to facilitate the reception of messages. All connection parameters as well as global settings exist as input parameters.

Syntax

```
FUNCTION_BLOCK FB_IotFunctions_Connector
VAR_INPUT
    sAmsNetId                       : STRING;
    nDefaultRequestTimeout          : UDINT;
    eDefaultMessageAckOption        : EIotFunctionsAckOption;
    nDefaultMessageRetryInterval    : UDINT;
    nDefaultMessageCumulativeTimeout : UDINT;
END_VAR
VAR_OUTPUT
    hrInitializationErrorCode : HRESULT;
    hrLastMessageErrorCode    : HRESULT;
    driverOTCID               : OTCID;
    fbTcResultEvent           : FB_TcIotFunctionsResultEvent;
    stIotFunctionsEvent       : ST_IotFunctionsEvent;
    pStIotFunctionsRequests   : POINTER TO ST_IotFunctionsRequestContainer;
END_VAR
```

Inputs

Name	Type	Description
sAmsNetId	STRING	Target Data Agent Instance AmsNetId [127.0.0.1.1.1]
nDefaultRequestTimeout	UDINT	Default timeout value in milliseconds [10000]
nDefaultMessageRetryInterval	UDINT	Default message retry interval in milliseconds (0 disables retries) [0]
nDefaultMessageCumulativeTimeout	UDINT	Default cumulative timeout in milliseconds (successfully receiving message data will prolong the containing request's timeout by this value) [0]

Outputs

Name	Type	Description
hrInitializationErrorCode	HRESULT	HRESULT of driver instantiation and configuration
hrLastMessageErrorCode	HRESULT	HRESULT of latest reported error occurred in any message during Execute invocation.
driverOTCID	OTCID	OTCID of the instantiated driver (passable to request and message FB input variables in multi data agent environments)
pStIotFunctionsRequests	POINTER TO ST_IotFunctionsRequestContainer [▶ 39]	Description of latest error code.

Methods

Name	Description
Execute	Enables background communication with TC3 IoT Data Agent. The method must be called cyclically.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.14	IPC or CX (c86, x64, ARM)	Tc3_IotFunctions

6.1.2 FB_IotFunctions_Message

The function block provides read/write operations for messages. All message parameters as well as global settings exist as input parameters.

Syntax

```
FUNCTION_BLOCK FB_IotFunctions_Message
VAR_INPUT
  nChannelId                : UINT;
  nRequestTimeout           : UDINT;
  eMessageAckOption         : EIotFunctionsAckOption;
  nMessageRetryInterval     : UDINT;
  nMessageCumulativeTimeout : UDINT;
  eSumCommandMode           : EIotFunctionsSumCommandMode;
  iotFunctionsDriverOTCID   : OTCID;
END_VAR
VAR_OUTPUT
  hResult             : HRESULT;
  pStMessageDetails   : POINTER TO ST_IotFunctionsMessage;
  pStRequestDetails   : POINTER TO ST_IotFunctionsRequest;
  bInitialized        : BOOL;
  fbTcResultEvent     : FB_TcIotFunctionsResultEvent;
  stIotFunctionsEvent : ST_IotFunctionsEvent;
END_VAR
```

Inputs

Name	Type	Description
nChannelId	UINT	Channel Id of corresponding IotFunctions channel in target data agent instance's configuration.
nRequestTimeout	UDINT	Timeout value in milliseconds.
nMessageRetryInterval	UDINT	Message interval in milliseconds (0 disables retries).
nMessageCumulativeTimeout	UDINT	Cumulative timeout (successfully receiving message data will prolong the containing request's timeout by this value).
iotFunctionsDriverOTCID	OTCID	Target IotFunctions Driver Instance's OTCID (see driverOTCID of FB_IotFunctions_Connector)

Outputs

Name	Type	Description
hResult	HRESULT	Contains the last hresult (is updated when bBusy or bError are accessed or Read/Write is invoked)
pStMessageDetails	POINTER TO ST_IotFunctionsMessage [▶ 38]	Pointer to structure containing detailed information about the underlying message
pStRequestDetails	POINTER TO ST_IotFunctionsRequest [▶ 39]	Pointer to structure containing detailed information about the underlying request in which this message is executed
bInitialized	BOOL	Indicates whether interface lookup and initialization was successful

Methods

Name	Description
Read	Reads a message from the channel
Acknowledge	Acknowledges error/success status and frees associated message objects. Call this method after status acknowledgment if no new read/write operation will be started this cycle to prevent multiple evaluation of latest status.
Write	Writes a message to the channel

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.14	IPC or CX (c86, x64, ARM)	Tc3_IotFunctions

6.1.3 FB_IotFunctions_Request

The function block enables the synchronization of multiple messages (see Synchronizing message operations [▶ 17]).

Syntax

```
FUNCTION_BLOCK FB_IotFunctions_Request
VAR_INPUT
    iotFunctionsDriverOTCID : OTCID;
    nRequestTimeout         : UDINT;
    eSumCommandMode         : EIotFunctionsSumCommandMode;
END_VAR
VAR_OUTPUT
    hResult             : HRESULT;
    pStRequestDetails   : POINTER TO ST_IotFunctionsRequest;
    bInitialized        : BOOL;
    fbTcResultEvent     : FB_TcIotFunctionsResultEvent;
    stIotFunctionsEvent : ST_IotFunctionsEvent;
END_VAR
```

Inputs

Name	Type	Description
iotFunctionsDriverOTCID	OTCID	Target IotFunctions Driver Instance's OTCID [defaults to first created FB_IotFunctions_Connector's created object]
nRequestTimeout	UDINT	Timeout value in milliseconds

Outputs

Name	Type	Description
hResult	HRESULT	Contains the last message's hresult (is updated when bBusy or bError are accessed or Create/Execute/ EnqueueRead/EnqueueWrite is invoked)
pStRequestDetails	POINTER TO ST_IotFunctionsRequest [▶ 39]	Pointer to structure containing details information about the underlying request
bInitialized	BOOL	Indicates whether interface lookup and initialization was successful

Methods

Name	Description
Create	Creates a new request
EnqueueRead	Adds an instance of FB_IotFunctions_Message [▶ 36] for read operation
EnqueueWrite	Adds an instance of FB_IotFunctions_Message [▶ 36] for write operation
Execute	Executes the request
Acknowledge	Acknowledges error/success status and frees associated message objects. Call this method after status acknowledgment if no new read/write operation will be started this cycle to prevent multiple evaluation of latest status.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.14	IPC or CX (c86, x64, ARM)	Tc3_IotFunctions

6.2 Data types

6.2.1 ST_IotFunctionsEvent

Contains detailed information about the cause of the latest error code result.

```
(* guid of underlying event class *)
uuidEventClass : GUID;

(* event id of underlying event *)
nEventId : UDINT;

(* severity level of underlying event *)
nSeverity : UDINT;

(* verbose message of underlying event *)
sEventMsg : STRING(255);

(* verbose event class name of underlying event *)
sEventClass : STRING(255);

(* verbose origin of underlying event *)
sSourcePath : STRING(255);

(* cycle in which underlying event occurred *)
nCycle : ULINT;
```

6.2.2 ST_IotFunctionsMessage

Contains detailed information about the underlying message object.

```
(* request Id of governing request containing this message *)
nRequestId : ULINT;

(* cycle in which this message was created *)
nCycleCreated : ULINT;

(* time passed since creation (in milliseconds) *)
nAge : ULINT;

(* corresponding IoTDataAgent channel *)
nChannelId : UDINT;

(* hrResult of latest action *)
hrResultCode : HRESULT;

(* message Id *)
nMessageId : UINT;

(* amount of initiated Ads requests during message lifetime *)
nAdsRequestCount : UINT;

(* amount of received Ads confirmations during message lifetime *)
nAdsConfirmationCount : UINT;

(* current (internal) state of message object. *)
eMessageState : EIotFunctionsMessageState;

(* message direction [read/write] *)
eMessageDirection : EIotFunctionsMessageDirection;

(* indicates if the accepted data differes from the previously contained data (relevant for reading
data) *)
bBufferChanged : BOOL;

(* name of the corresponding symbol *)
sSymbolName : STRING(255);
```

6.2.3 ST_IotFunctionsRequest

Contains detailed information about the underlying request object.

```
(* request Id *)
nRequestId : ULINT;

(* cycle in which this request was created *)
nCycleCreated : ULINT;

(* time passed since creation (in milliseconds) *)
nAge : ULINT;

(* time until request expires (timeout) (in milliseconds) *)
nTimeToLive : ULINT;

(* count of currently pending messages in this request *)
nPendingCount : UDINT;

(* count of currently contained messages in this request *)
nTotalCount : UINT;

(* indicates if the corresponding internal object has been removed *)
bIsRemoved : BOOL;

(* indicates if any contained message has timed out *)
bIsTimedOut : BOOL;

(* indicates if all contained messages have been processed (regardless of success, error or timeout
states) *)
bIsCompleted : BOOL;
```

6.2.4 ST_IotFunctionsRequestContainer

Contains an array of ST_IotFunctionsRequests that represents the requests handled by the containing FB_IotFunctions_Connector instance.

```
(* array of ST_IotFunctionsRequests *)
apIotRequests : ARRAY [0..99] OF POINTER TO ST_IotFunctionsRequest;
```

7 Samples

Samples for TC3 IoT Functions can be downloaded as a single container solution: https://infosys.beckhoff.com/content/1033/tf6710_tc3_iot_functions/Resources/zip/5247017867.zip

8 Appendix

8.1 Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

Beckhoff's branch offices and representatives

Please contact your Beckhoff branch office or representative for local support and service on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on her internet pages: https://www.beckhoff.com

You will also find further documentation for Beckhoff components there.

Beckhoff Support

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

- support
- design, programming and commissioning of complex automation systems
- and extensive training program for Beckhoff system components

Hotline:	+49 5246 963 157
Fax:	+49 5246 963 9157
e-mail:	support@beckhoff.com

Beckhoff Service

The Beckhoff Service Center supports you in all matters of after-sales service:

- on-site service
- repair service
- spare parts service
- hotline service

Hotline:	+49 5246 963 460
Fax:	+49 5246 963 479
e-mail:	service@beckhoff.com

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

Phone:	+49 5246 963 0
Fax:	+49 5246 963 198
e-mail:	info@beckhoff.com
web:	https://www.beckhoff.com

More Information:
www.beckhoff.com/tf6710

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