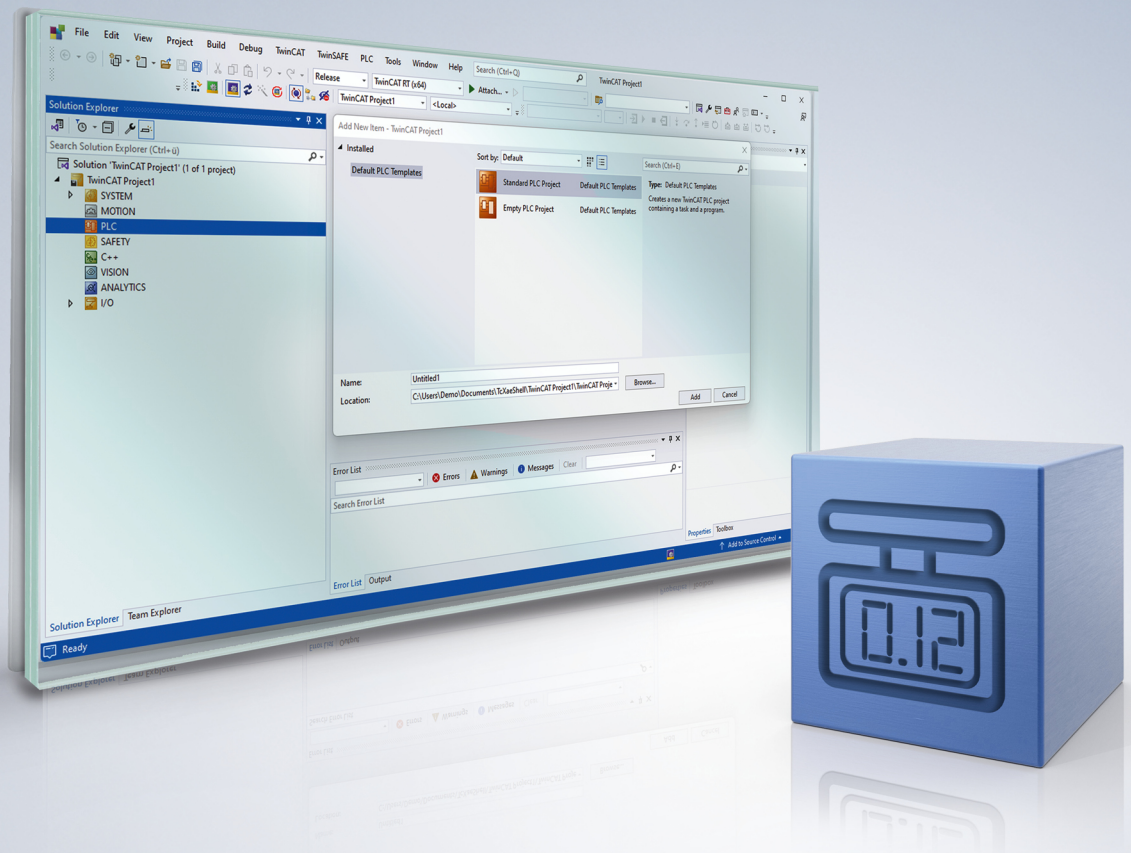
BECKHOFF New Automation Technology

Manual | EN

TF3685

TwinCAT 3 | Weighing Library



2026-01-29 | Version: 1.2.1

Table of contents

1 Foreword 5
1.1 Notes on the documentation 5
1.2 For your safety 6
1.3 Notes on information security 7
2 Overview 8
3 Installation 9
3.1 System requirements 9
3.2 Installation under TwinCAT 4026 9
3.3 Installation 9
3.4 Licensing 12
4 System requirements 15
5 Technical introduction 16
5.1 Measuring procedure 16
6 PLC API 20
6.1 Function blocks 20
6.1.1 FB_WG_ComboFilter 22
6.1.2 FB_WG_Scaling 24
6.1.3 FB_WG_Weighing 30
6.2 Data types 34
6.2.1 Configuration structures 34
6.2.2 E_WG_Calibrate 39
6.2.3 E_WG_AutoTareType 39
7 Samples 40
7.1 Dynamic weighing 40
8 Appendix 43
8.1 Return codes 43
8.2 FAQ - frequently asked questions and answers 45
9 Support and Service 46

1 Foreword

1.1 Notes on the documentation

This description is intended exclusively for trained specialists in control and automation technology who are familiar with the applicable national standards.
The documentation and the following notes and explanations must be complied with when installing and commissioning the components.
The trained specialists must always use the current valid documentation.

The trained specialists must ensure that the application and use of the products described is in line with all safety requirements, including all relevant laws, regulations, guidelines, and standards.

Disclaimer

The documentation has been compiled 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 notice.
Claims to modify products that have already been supplied may not be made on the basis of the data, diagrams, and descriptions in this documentation.

Trademarks

Beckhoff®, ATRO®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, MX-System®, Safety over EtherCAT®, TC/BSD®, TwinCAT®, TwinCAT/BSD®, TwinSAFE®, XFC®, XPlanar®, and XTS® are registered and licensed trademarks of Beckhoff Automation GmbH.
If third parties make use of the designations or trademarks contained in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.



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

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.
The distribution and reproduction of this document, as well as the use and communication of its contents without express authorization, are prohibited.
Offenders will be held liable for the payment of damages. All rights reserved in the event that a patent, utility model, or design are registered.

Third-party trademarks

Trademarks of third parties may be used in this documentation. You can find the trademark notices here: https://www.beckhoff.com/trademarks.

1.2 For your safety

Safety regulations

Read the following explanations for your safety.
Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

All the components are supplied in particular hardware and software configurations which are 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 technology who are familiar with the applicable national standards.

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings

⚠ DANGER
Hazard with high risk of death or serious injury.

⚠ WARNING
Hazard with medium risk of death or serious injury.

⚠ CAUTION
There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment

NOTICE
The environment, equipment, or data may be damaged.

Information on handling the product



This information includes, for example:
recommendations for action, assistance or further information on the product.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

2 Overview

The TwinCAT 3 Weighing PLC Library allows a scale for weight measurement to be integrated into the PC-based machine control system – particularly in conjunction with the I/Os of the ELM35xx, ED336x-0x00 and EL3356 EtherCAT Terminals. The focus is primarily on the dynamic weighing process. Signal filtering is particularly demanding here, as the weighing time has a significant influence on the overall processing time of the machine. Rapid signal filtering with the same level of precision produces a faster weight result, which ultimately makes for faster machines.

Since a load cell and a measured value acquisition via the corresponding EtherCAT Terminals do not yet constitute a scale, this is exactly where the PLC library comes in. It takes over the scaling of the measured values, while the new PLC function blocks cover functions such as zeroing and taring. In addition to manual triggering of the weight measurement, automatic measurement is also possible. The production material is detected and the measurement is taken directly. The main advantage is that, depending on the application, even external triggers such as photoelectric sensors and initiators can be omitted.

The license of the TF3680 | TwinCAT 3 filter library is also included in this product.

Components

- TwinCAT Weighing PLC library: Tc3_Weighing.compiled-library
- TwinCAT Filter PLC library: Tc3_Filter.compiled-library
- Versioned driver: TcWeighing.tmx
- Description file: TcWeighing.tmc

3 Installation

3.1 System requirements

Technical data	Description
Operating system	Windows 10, Windows 11, TwinCAT/BSD, Beckhoff RT Linux®
Target platform	PC architecture (x86, x64, Arm®)
Minimum TwinCAT version	3.1.4024.50
Required TwinCAT license	TF3685 TwinCAT 3 Weighing Library

3.2 Installation under TwinCAT 4026

TwinCAT Package Manager

If you are using TwinCAT 3.1 Build 4026 (and higher) on the Microsoft Windows operating system, you can install this function via the TwinCAT Package Manager, see Installation documentation.

Normally you install the function via the corresponding workload; however, you can also install the packages contained in the workload individually. This documentation briefly describes the installation process via the workload.

Command line program TcPkg

You can use the TcPkg **C**ommand **L**ine **I**nterface (CLI) to display the available workloads on the system:

```
tcpkg list TF3685
```

You can install the workloads of the TF3685 TwinCAT 3 Weighing function via the following command.

```
tcpkg install TF3685.WeighingLibrary.XAE
```

TwinCAT Package Manager UI

You can use the **U**ser **I**nterface (UI) to display all available workloads and install them if required.
To do this, follow the corresponding instructions in the interface.

3.3 Installation

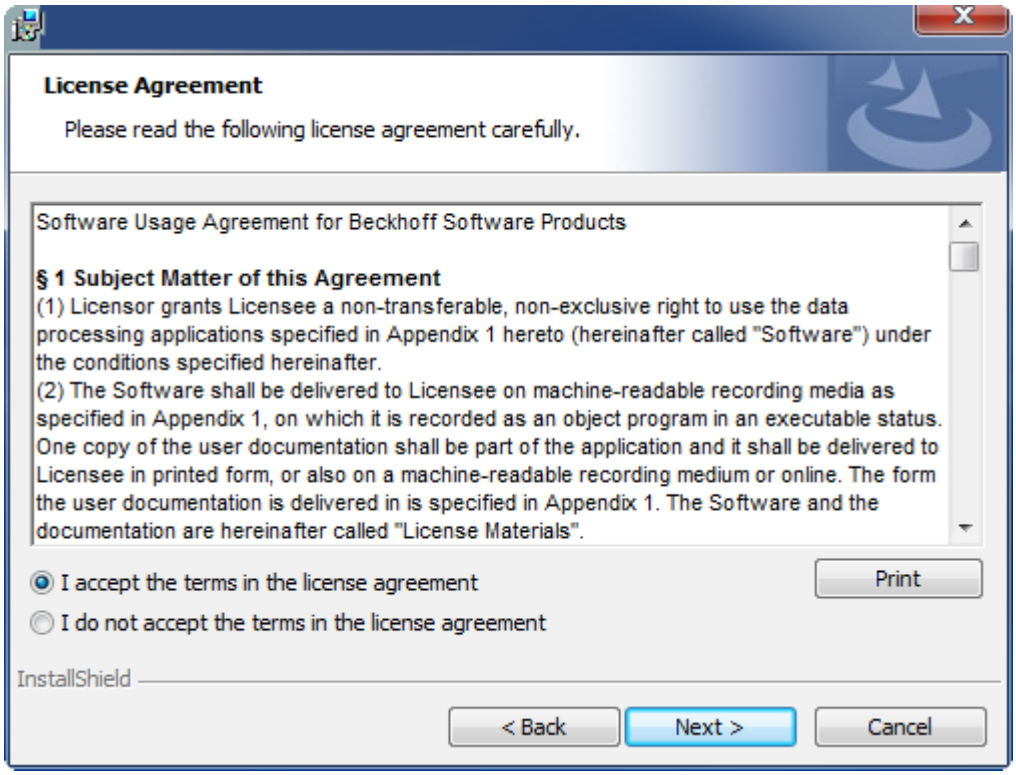
Setup installation (TwinCAT 3.1 Build 4024)

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

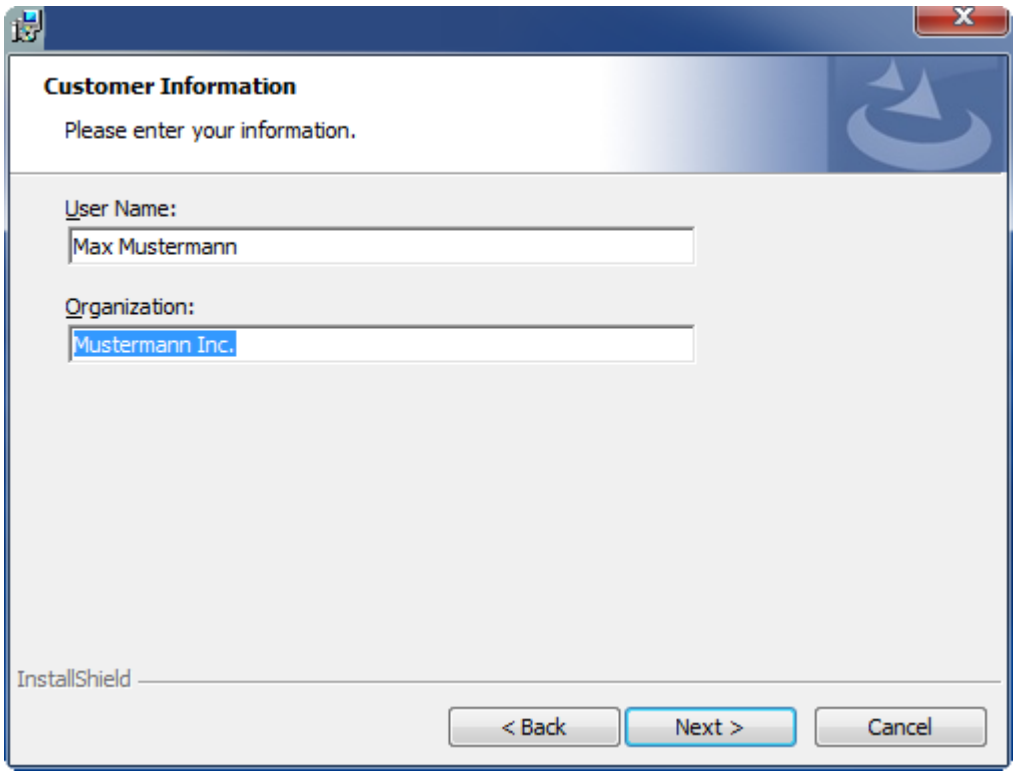
✓ The TwinCAT 3 function setup file was downloaded from the Beckhoff website.

1. Run the setup file as administrator. To do this, select the **Run As Admin** command in the context menu of the file.
   ⇨ The installation dialog opens.

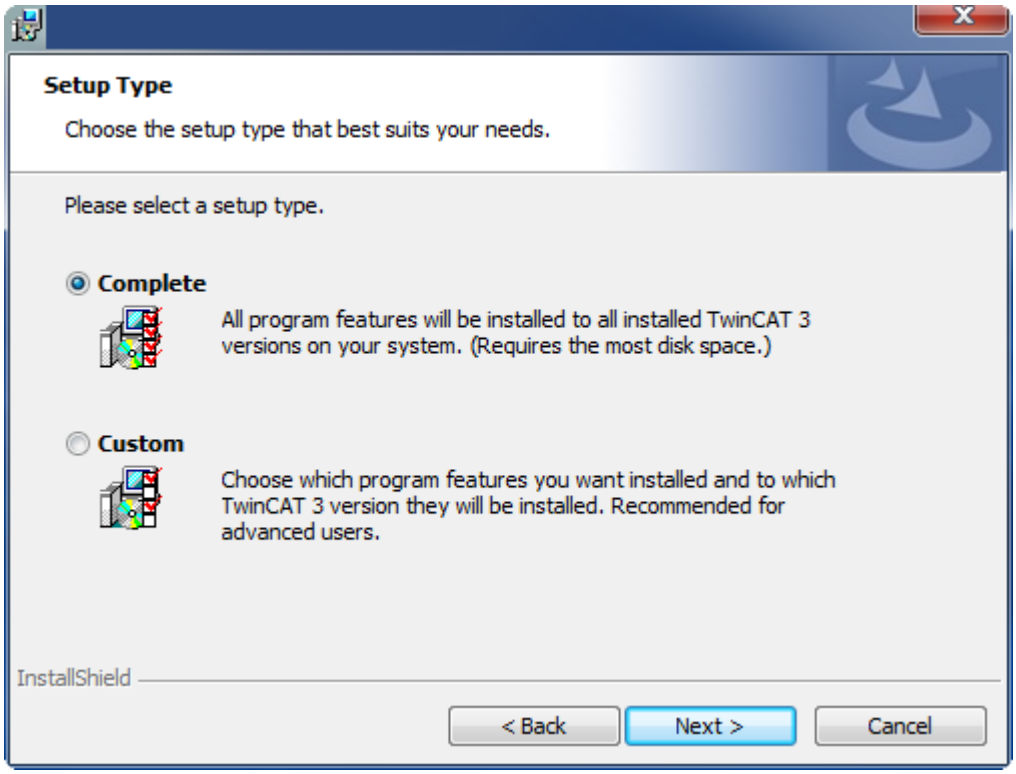
2. Accept the end user licensing agreement and click **Next**.



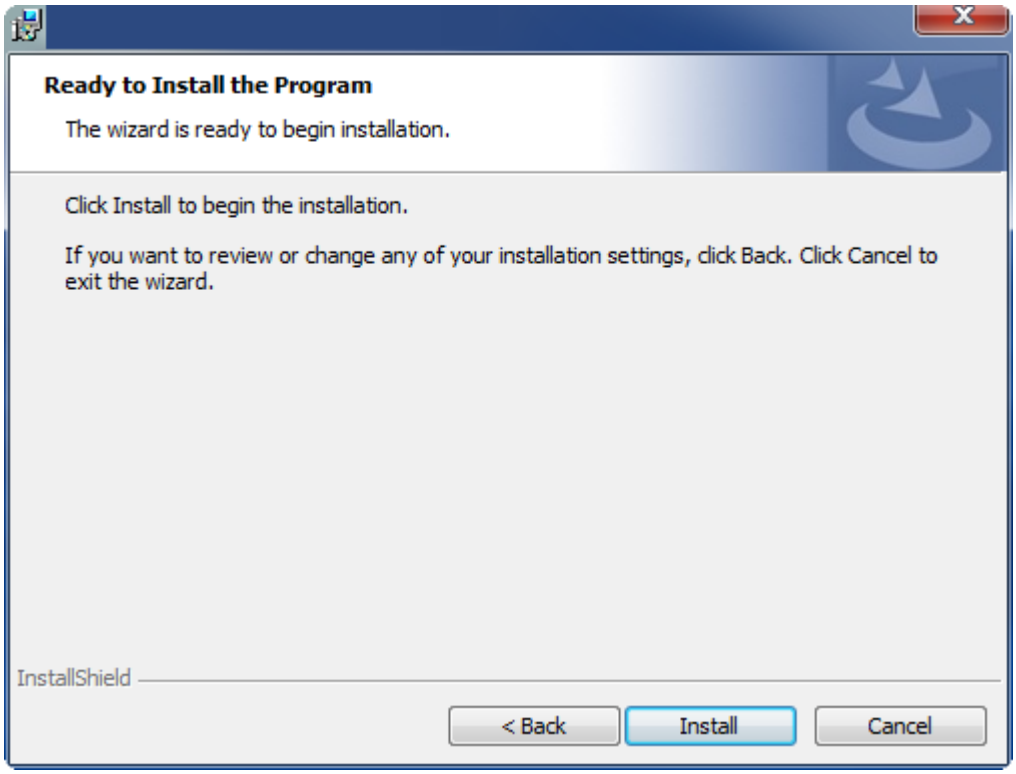
3. Enter your user data.



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

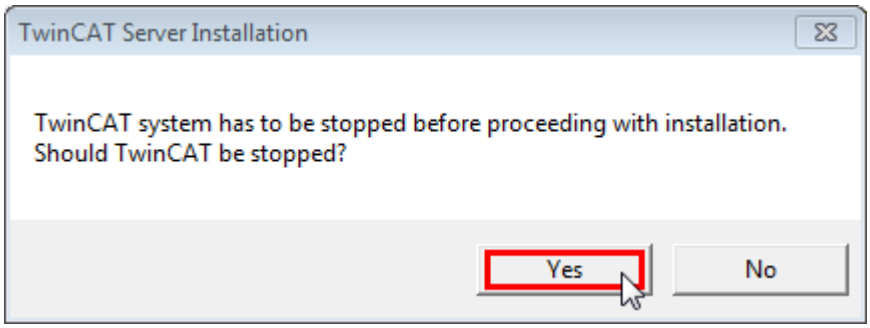


5. Click **Next**, then **Install** to start the installation.

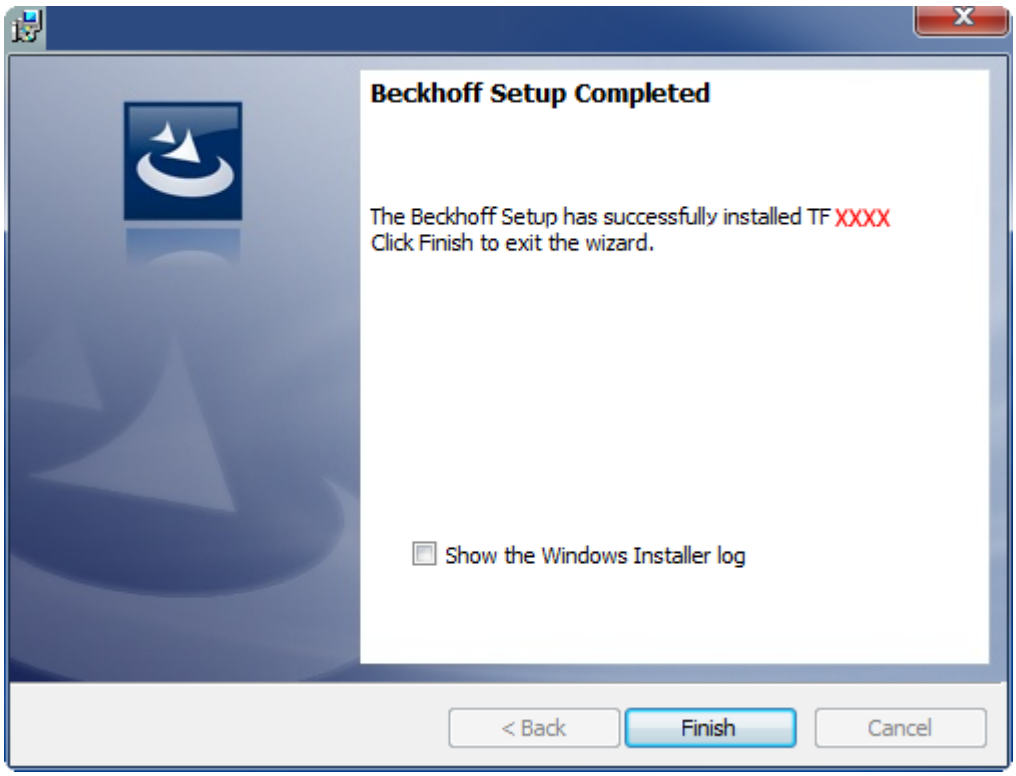


⇨ A dialog box informs you that the TwinCAT system must be stopped to proceed with the installation.

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



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



⇨ The TwinCAT 3 function has been installed successfully.

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".

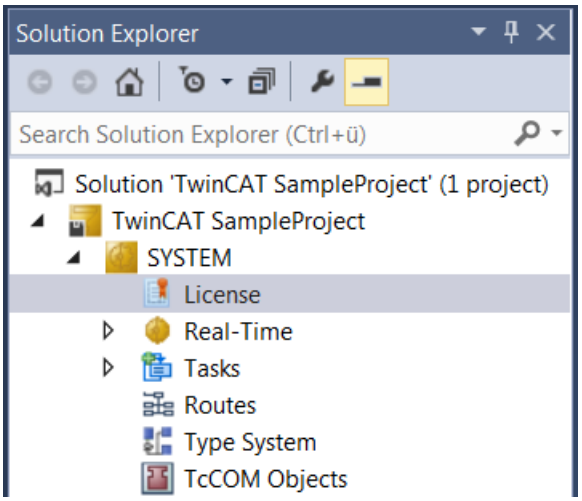
Licensing the 7-day test version of a TwinCAT 3 Function



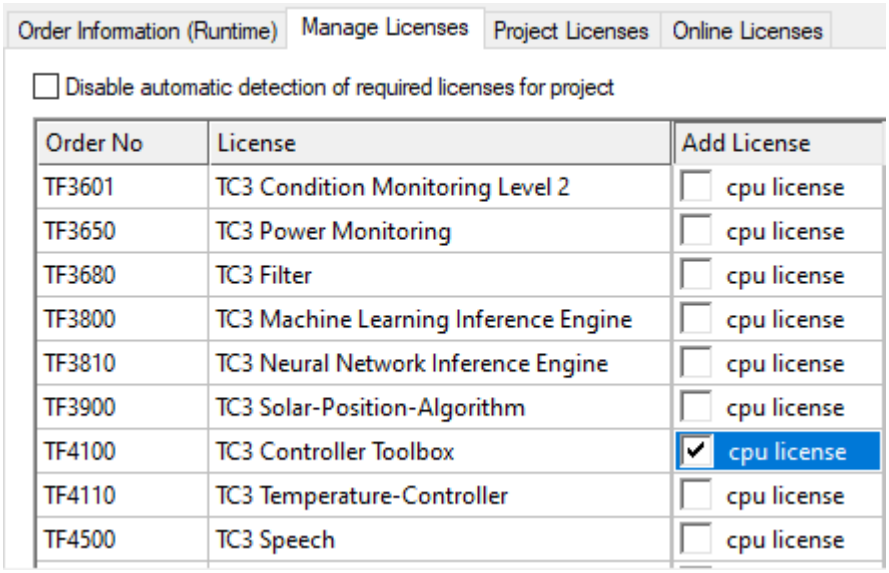
A 7-day test version cannot be enabled for a TwinCAT 3 license dongle.

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.



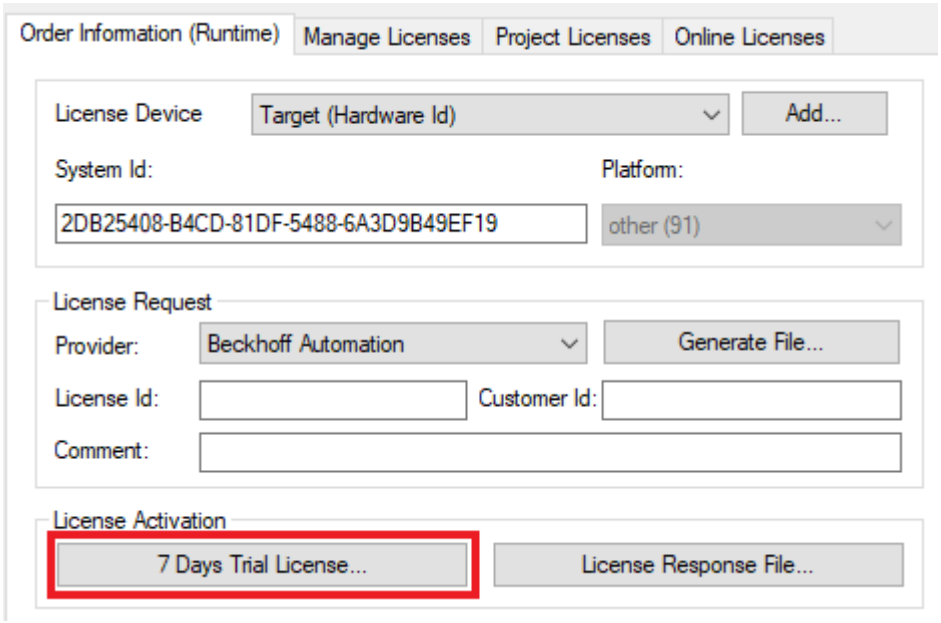
   ⇨ The TwinCAT 3 license manager opens.
5. Open the **Manage Licenses** tab. In the **Add License** column, check the check box for the license you want to add to your project (e.g. "TF4100 TC3 Controller Toolbox").



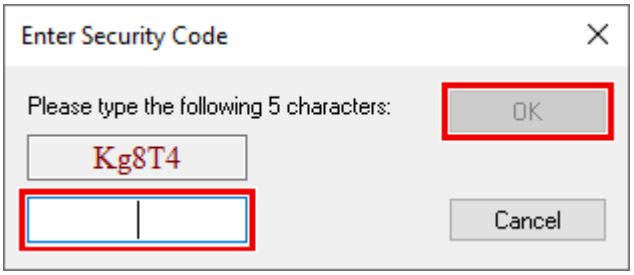
Order No	License	Add License
TF3601	TC3 Condition Monitoring Level 2	cpu license
TF3650	TC3 Power Monitoring	cpu license
TF3680	TC3 Filter	cpu license
TF3800	TC3 Machine Learning Inference Engine	cpu license
TF3810	TC3 Neural Network Inference Engine	cpu license
TF3900	TC3 Solar-Position-Algorithm	cpu 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.

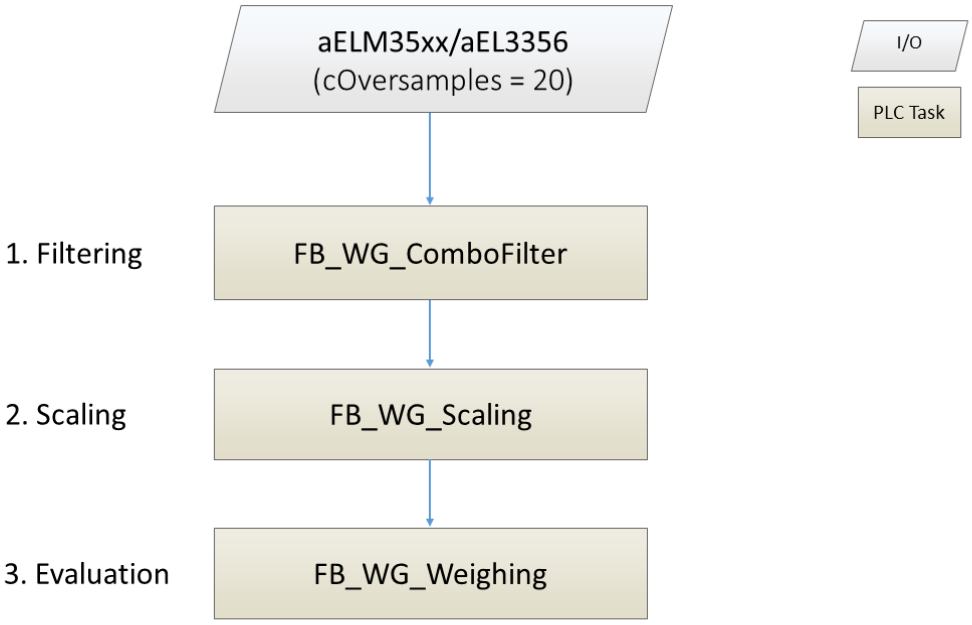
4 System requirements

Technical data	Description
Operating system	Windows 10, Windows 11, TwinCAT/BSD, Beckhoff RT Linux®
Target platform	PC architecture (x86, x64, Arm®)
Minimum TwinCAT version	3.1.4024.50
Required TwinCAT license	TF3685 TwinCAT 3 Weighing Library

5 Technical introduction

5.1 Measuring procedure

A typical measuring procedure for determining the mass of a body consists of three main steps: filtering, scaling and evaluation.



1. Filtering

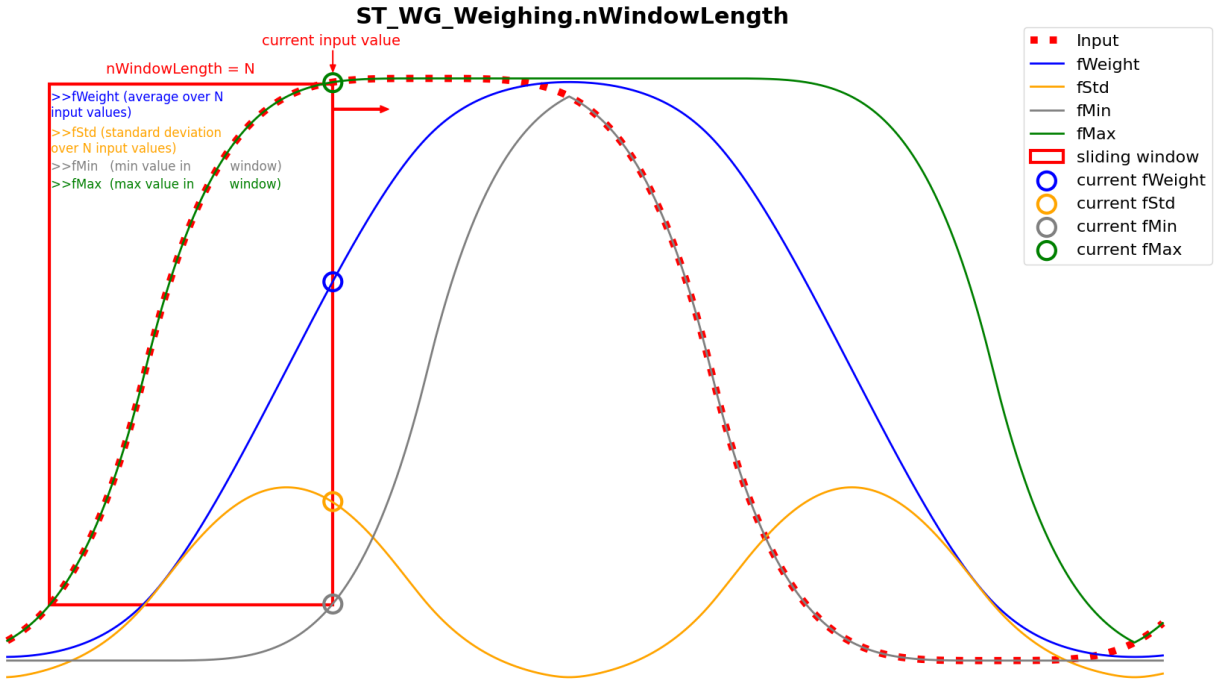
The signal from the EtherCAT Terminals of the ELM35xx and EL3356 series typically has a level of noise that requires filtering to ensure valid measurement results. The function block FB_WG_ComboFilter [▶ 22] offers an effective solution for this by switching a combination of PTn, Moving Average and Notch filters in series. If these filter options are not sufficient, a selection of further filters from the TwinCAT 3 filter library is available.

2. Scaling

The filtered input signal must then be scaled in order to specify the weight in the desired unit (e.g. grams [g]). Scaling is performed by the function block FB_WG_Scaling [▶ 24]. Accurate measurement also requires calibration of the PLC function block, which can be carried out using a two-point calibration, for example.

3. Evaluation

In the final step, the scaled signal is analyzed using the function block FB_WG_Weighing [▶ 30]. It is necessary to provide the configuration structure ST_WG_Weighing [▶ 35] with the appropriate parameters. A key parameter in this structure is `ST_WG_Weighing.nWindowLength`, which defines the number of samples used to calculate the moving average - a variable also known as the window size. This parameter determines how many past values are used to calculate the outputs `fWeight`, `fStd`, `fMin` and `fMax` of the function block FB_WG_Weighing [▶ 30]. Here, `fWeight` represents the average, `fStd` the standard deviation and `fMin/fMax` the minimum or maximum value of the last `nWindowLength` input values. A supplementary figure can illustrate these relationships.



To obtain additional results from the function block, such as `bValidMeasurement`, `bNewResult`, `tLastResult`, `fLastWeight` and `fLastStd`, it is necessary to configure the substructure ST_WG_Weighing_Validation [▶ 36] accordingly.

Within ST_WG_Weighing_Validation [▶ 36], the parameters `fThresholdWeight`, `fMaxWeightDeviation` and `fMaxStd` define the criteria for a valid measurement. For a measurement to be considered valid, the following conditions must be met:

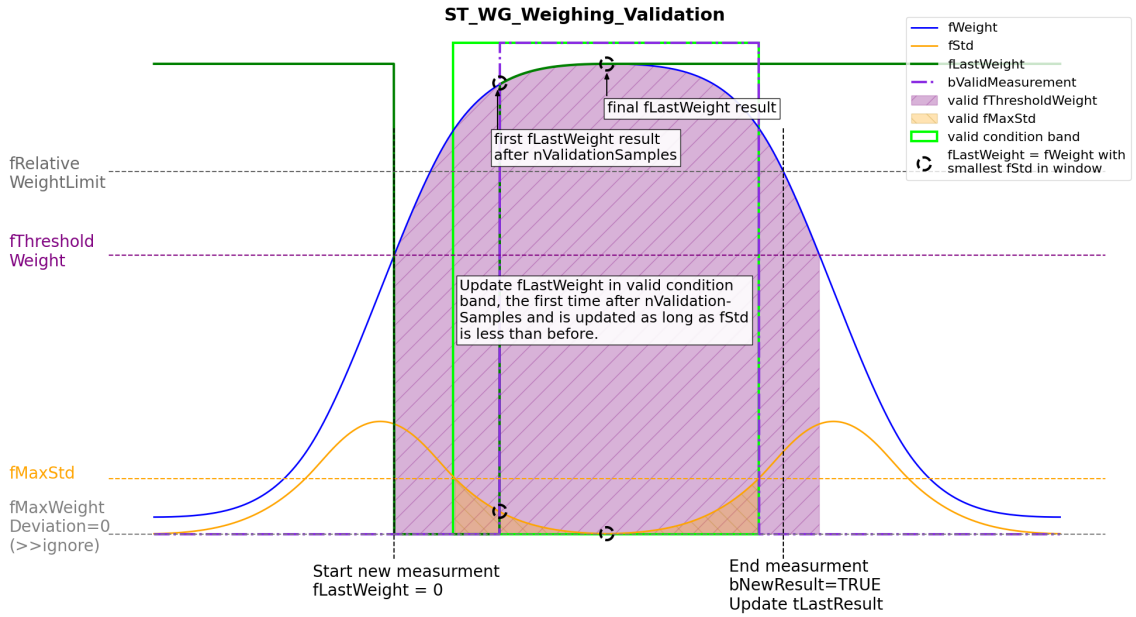
- `FB_WG_Weighing.fWeight` must be greater than or equal to `fThresholdWeight`.
- The difference `FB_WG_Weighing.fMax - FB_WG_Weighing.fMin` must not exceed `fMaxWeightDeviation`.
- `FB_WG_Weighing.fStd` must be less than or equal to `fMaxStd`.

These conditions must be met over the number of consecutive samples defined in `nValidationSamples` in order to set `FB_WG_Weighing.bValidMeasurement` to `TRUE` .

The measurement is initiated as soon as `FB_WG_Weighing.fWeight` exceeds the value of `ST_WG_Weighing_Validation.fThresholdWeight` for the first time. As long as `FB_WG_Weighing.bValidMeasurement` has the value `TRUE` , the weight (`fWeight`) with the smallest standard deviation (`fStd`) is searched for and continuously updated in `FB_WG_Weighing.fLastWeight` and in `FB_WG_Weighing.fLastStd`. If the parameter `ST_WG_Weighing_Validation.fRelativeWeightLimit` is defined, the measurement ends as soon as `FB_WG_Weighing.fWeight` falls below the value of `fThresholdWeight * fRelativeWeightLimit` . If this parameter is not set, the measurement ends when `fWeight` falls below `fThresholdWeight` .

At the end of the measurement, the timestamp is saved in `FB_WG_Weighing.tLastResult` and `FB_WG_Weighing.bNewResult` is set to `TRUE` for exactly one cycle. If `FB_WG_Weighing.fWeight` exceeds the value `fThresholdWeight` again, `FB_WG_Weighing.fLastWeight` is reset and a new measurement begins.

The following figure illustrates the process described above and shows the relationship between the parameters and the conditions for a valid measurement:



ST_WG_Weighing_AutoTare [▶ 38] can be configured in the same way as ST_WG_Weighing_Validation [▶ 36] to receive results such as `fAutoTareOffset` and `bNewAutoTareResult` from the function block. These are essential for automatically taring the function block FB_WG_Scaling [▶ 24], for example by calling AutoTare [▶ 33](fbScaling,E_WG_AutoTareType [▶ 39].`eEnd`).

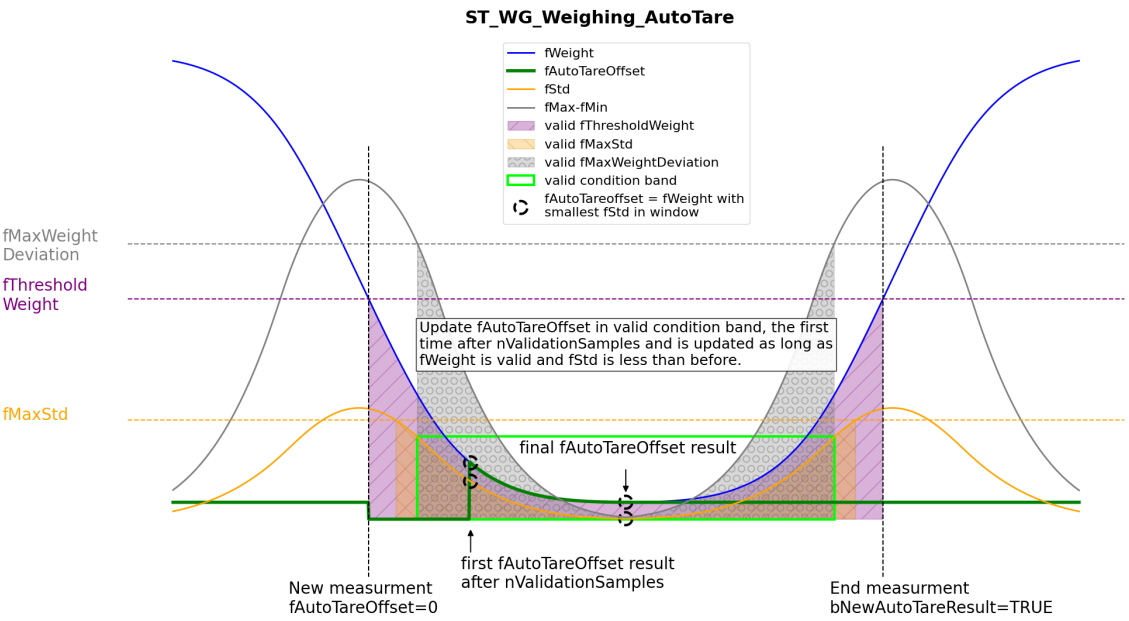
The parameters `fThresholdWeight`, `fMaxWeightDeviation` and `fMaxStd` in ST_WG_Weighing_AutoTare [▶ 38] define the criteria for the validation of a measurement. A measurement is considered valid if:

- `FB_WG_Weighing.fWeight` does not exceed the value `fThresholdWeight`.
- The difference `FB_WG_Weighing.fMax - FB_WG_Weighing.fMin` does not exceed the specified `fMaxWeightDeviation`.
- `FB_WG_Weighing.fStd` is less than or equal to the defined `fMaxStd`.

The update of `fAutoTareOffset` begins as soon as `fWeight` falls below `fThresholdWeight` for the first time and the stated conditions are met over a series of consecutive samples defined in `nValidationSamples` . The system searches for the weight with the lowest standard deviation and updates continuously `fAutoTareOffset`.

The measurement ends as soon as `FB_WG_Weighing.fWeight` exceeds `fThresholdWeight`. `FB_WG_Weighing.bNewAutoTareResult` is then set once to `TRUE` , which signals the end of the measurement. If `FB_WG_Weighing.fWeight` falls below the threshold value `fThresholdWeight`,, `FB_WG_Weighing.fAutoTareOffset` is reset and a new measurement is initiated.

The figure below illustrates the process and clarifies the relationship between the parameters and the criteria for a valid measurement.


ST_WG_Weighing_AutoTare
fWeight
fAutoTareOffset
fStd
fMax-fMin
valid fThresholdWeight
valid fMaxStd
valid fMaxWeightDeviation
valid condition band
fAutoTareoffset = fWeight with smallest fStd in window
fMaxWeight Deviation
fThreshold Weight
fMaxStd
Update fAutoTareOffset in valid condition band, the first time after nValidationSamples and is updated as long as fWeight is valid and fStd is less than before.
final fAutoTareOffset result
first fAutoTareOffset result after nValidationSamples
New measurment fAutoTareOffset=0
End measurment bNewAutoTareResult=TRUE

6 PLC API

6.1 Function blocks

Basic structure of the function blocks

All function blocks in the TwinCAT Weighing PLC Library are based on the same basic structure. This simplifies the engineering process when changing from one Weighing type to another.

Syntax

```
FUNCTION_BLOCK FB_WG_<type>
VAR_INPUT
    stConfig        : ST_WG_<type>;
END_VAR
VAR_OUTPUT
    bError          : BOOL;
    bConfigured     : BOOL;
    ipResultMessage : I_TCMessage;
END_VAR
```

Inputs

To configure the Weighing function block, a configuration structure of type ST_WG_<type> is transferred to the function blocks during instantiation. The configuration structure can be assigned in the declaration or via the method `Configure()` at runtime.

See also: Data types [▶ 34] > Configuration structures [▶ 34]

Sample of configuration in the declaration:

```
(* define configure structure - exemplary for ComboFilter *)
stParams : ST_WG_ComboFilter := (
    nOrder := nOrder,
    fCutoff := fCutoff,
    fSamplingRate := fSampleRate,
    nSamplesToFilter := nSamplesToFilter);
```

```
(* create filter instance with configure structure *)
fbFilter : FB_WG_ComboFilter := (stConfig := stParams);
```

Outputs

All function blocks have an error flag `bError` and a flag `bConfigured` of type `BOOL` as output parameters. These indicate whether an error has occurred and whether the corresponding function block instance has been successfully configured. The output `ipResultMessage` of type `I_TcMessage` provides various properties for explaining the cause of an event and methods for processing the message (event list).

See also: I_TcEventBase und I_TcMessage

Methods

All function blocks in the Tc3_Weighing library have three methods. They return a positive value if they were executed without errors.

Configure()

The method can be used at runtime to initially configure the instance of a Weighing function block (if not already done in the declaration) or to reconfigure it.

```
METHOD Configure : BOOL
VAR_INPUT
    stConfig : ST_WG_<type>;
END_VAR
```

Call()

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

```
METHOD Call : BOOL
VAR_INPUT
     pIn        : POINTER TO LREAL; (*address of input array*)
     nSizeIn    : UDINT;            (*size of input array*)
     pOut       : POINTER TO LREAL; (*address of output array*)
     nSizeOut   : UDINT;            (*size of output array*)
END_VAR
```

Reset()

The method resets the internal status of a Weighing function block. The influence of the past values on the current output value is eliminated.

```
METHOD Reset : BOOL
```

Properties

The Tc3_Weighing library references the TwinCAT 3 EventLogger and thus ensures that information (events) is provided via the standardized interface I_TcMessage.

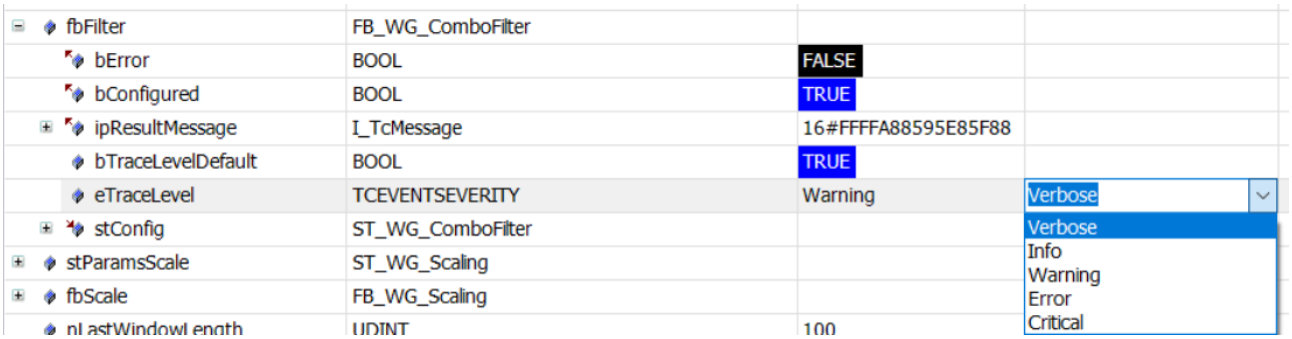
Each function block has the properties `eTraceLevel` of type `TcEventSeverity` and `eTraceLevelDefault` of type `BOOL`.

The trace level determines the severity of an event (Verbose, Info, Warning, Error, Critical) and is set via the property `eTraceLevel`.

```
(* Sample of setting fbFilter to trace level info *)
fbFilter.eTraceLevel := TcEventSeverity.Info;
```

The property `eTraceLevelDefault` can be used to set the trace level back to the default value (`TcEventSeverity.Warning`). The property can be read and written, i.e. the property `eTraceLevelDefault` can be used to query whether the default value is set.

The properties can also be set in Online View.



Dealing with oversampling

All function blocks are oversampling-capable, whereby different types of use are possible. The declaration of the Weighing function block instance `fbFilter` is always the same here.

1-channel application with oversamples

The input and output arrays can be declared as one-dimensional variables.

```
VAR CONSTANT
    cOversamples : UINT := 10;
END_VAR
VAR
    aInput  : ARRAY [1..cOversamples] OF LREAL;
    aOutput : ARRAY [1..cOversamples] OF LREAL;
END_VAR
```

```
bSucceed := fbFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

1-channel application without oversamples

If no oversampling is used, the input and output variables can also be declared as LREAL.

```
VAR CONSTANT
    cOversamples : UINT := 1;
END_VAR
VAR
    fInput  : LREAL;
    fOutput : LREAL;
END_VAR
```

```
bSucceed := fbFilter.Call(ADR(fInput), SIZEOF(fInput), ADR(fOutput), SIZEOF(fOutput));
```

6.1.1 FB_WG_ComboFilter



The function block FB_WG_ComboFilter implements a PTn, Moving Average and Notch filter connected in series.

The filter specification is transferred with the structure ST_WG_ComboFilter [▶ 34].

Syntax

Declaration:

```
fbFilter : FB_WG_ComboFilter(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_WG_ComboFilter
VAR_INPUT
    stConfig        : ST_WG_ComboFilter;
END_VAR
VAR_OUTPUT
    bError          : BOOL;
    bConfigured     : BOOL;
    ipResultMessage : I_TCMessage;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_WG_ComboFilter [▶ 34]	Structure for configuring the filter behavior

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling.

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and configuration of the function block.
Reset()	Local	Resets internal states.

Properties

Name	Type	Access	Definition location	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE, if eTraceLevel = Warning.
eTraceLevel	TcEventSeverity	Get, Set	Local	Critical	Severity of an event

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.50	PC or CX (x64, x86)	Tc3_Weighing

6.1.1.1 Configure

This method can be used at runtime to initially configure the instance of a filter (if it was not already configured in the declaration) or to reconfigure it.

If a filter instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure : BOOL
VAR_INPUT
    stConfig : ST_WG_ComboFilter;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_WG_ComboFilter [▶ 34]	Structure for configuring the filter behavior

Sample

```
(*Declaration without configuration*)
fbFilter : FB_WG_ComboFilter();
```

```
(* initial configuration of fbFilter *)
IF bInit THEN
    bSucceed := fbFilter.Configure(stConfig := stParams);
    bInit    := FALSE;
END_IF

(* reconfigure fbFilter on bReconfigure = TRUE *)
IF bReconfigure THEN
    stParams.nSamplesToFilter := 11; (*change filter order*)
    bSucceed         := fbFilter.Configure(stConfig := stParams);
    bReconfigure     := FALSE;
END_IF
```

Return value

Name	Type	Description
Configure	BOOL	TRUE if the weighing instance was configured successfully.

6.1.1.2 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

Syntax

```
METHOD Call : BOOL
VAR_INPUT
     pIn         : POINTER TO LREAL;
     nSizeIn     : UDINT;
     pOut        : POINTER TO LREAL;
     nSizeOut    : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array
pOut	POINTER TO LREAL	Address of the output array
nSizeOut	UDINT	Size of the output array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput := ARRAY [1..cOversamples] OF LREAL;
aOutput := ARRAY [1..cOversamples] OF LREAL;
```

```
bSucceed := fbWeighing.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

6.1.1.3 Reset

The method resets the internal status of the weighing instance. By resetting the function block, the weighing instance is reset to its original state, i.e. without any influence from the past. The weighing instance is therefore reset to the last configuration status.

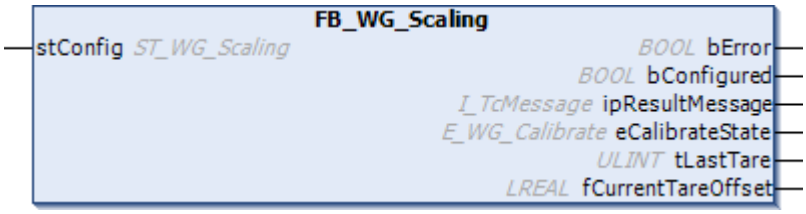
Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the weighing instance has been successfully reset.

6.1.2 FB_WG_Scaling



The function block FB_WG_Scaling is used for scaling raw values. The raw values can be scaled individually or as an array, for example as oversampling values.

The configuration structure is transferred with ST_WG_Scaling [▶ 35].

Syntax

Declaration:

```
fbScaling : FB_WG_Scaling(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_WG_Scaling
VAR_INPUT
    stConfig        : ST_WG_Scaling;
END_VAR
VAR_OUTPUT
    bError              : BOOL;
    bConfigured         : BOOL;
    ipResultMessage     : I_TCMessage;
    eCalibrateState     : ULINT;
    tLastTare           : ULINT;
    fCurrentTareOffset  : LREAL;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_WG_Scaling [▶ 35]	Function block-specific configuration structure

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling.
eCalibrateState	E_WG_Calibrate [▶ 39]	Current calibrate/tare state.
tLastTare	ULINT	Timestamp of the last Tare()/ [▶ 29]UpdateTareOffset() [▶ 30] method call.
fCurrentTareOffset	LREAL	Updates itself with every Tare()/ [▶ 29]UpdateTareOffset() [▶ 30] method call.

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and configuration of the function block.
Reset()	Local	Resets internal states.
ApplyCalibration()	Local	Ends the calibration process.
CalibrateRefHigh()	Local	Triggers the fReferenceHigh calibration.
CalibrateRefLow()	Local	Triggers the fReferenceLow calibration.
Tare()	Local	Triggers the tare calibration.
UpdateTareOffset()	Local	Sets the tare offset value manually and updates the tLastTare/fCurrentTareOffset output.

Properties

Name	Type	Access	Definition location	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE, if eTraceLevel = Warning
eTraceLevel	TcEventSeverity	Get, Set	Local	Warning	Severity of an event
nTimeStamp	ULINT	Get, Set	Local	0	Timestamp of the oldest input value of the next Call().

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.50	PC or CX (x64, x86)	Tc3_Weighing

6.1.2.1 Configure

This method can be used at runtime to initially configure the instance of a filter (if it was not already configured in the declaration) or to reconfigure it.

If a weighing instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure : BOOL
VAR_INPUT
    stConfig : ST_WG_Scaling;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_WG_Scaling [▶ 35]	Structure for configuring the filter behavior

Sample

```
(*Declaration without configuration*)
fbScaling : FB_WG_Scaling();
```

```
(* initial configuration of fbScaling *)
IF bInit THEN
    bSucceed := fbScaling.Configure(stConfig := stParams);
    bInit    := FALSE;
END_IF

(* reconfigure fbScaling on bReconfigure = TRUE *)
IF bReconfigure THEN
    stParams.fRawHigh:= 10; (*change fRawHigh*)
    bSucceed         := fbScaling.Configure(stConfig := stParams);
    bReconfigure     := FALSE;
END_IF
```

Return value

Name	Type	Description
Configure	BOOL	TRUE if the weighing instance was configured successfully.

6.1.2.2 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer.

Syntax

```
METHOD Call : BOOL
VAR_INPUT
     pIn         : POINTER TO LREAL;
     nSizeIn     : UDINT;
     pOut        : POINTER TO LREAL;
     nSizeOut    : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array
pOut	POINTER TO LREAL	Address of the output array
nSizeOut	UDINT	Size of the output array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput := ARRAY [1..cOversamples] OF LREAL;
aOutput := ARRAY [1..cOversamples] OF LREAL;
```

```
bSucceed := fbWeighing.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutput), SIZEOF(aOutput));
```

6.1.2.3 Reset

The method resets the internal status of the weighing instance. By resetting the function block, the weighing instance is reset to its original state, i.e. without any influence from the past. The weighing instance is therefore reset to the last configuration status.

Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the weighing instance has been successfully reset.

6.1.2.4 ApplyCalibration

The method can be used at runtime to complete or cancel the triggered calibration process (CalibrateRefHigh() [▶ 28]/CalibrateRefLow() [▶ 28]).

Syntax

```
METHOD ApplyCalibration : BOOL
VAR_INPUT
     bAccept : BOOL;
END_VAR
```

Inputs

Name	Type	Description
bAccept	BOOL	If TRUE, the CalibrateRefHigh()/CalibrateRefLow() results are accepted. Otherwise discarded.

Sample

```
(*Declaration without configuration*)
fbScaling : FB_WG_Scaling();
```

```
(* accept calibration *)
IF bAcceptCalibration THEN
    fbScaling.ApplyCalibration(bAccept := TRUE);
    bAcceptCalibration := FALSE;
END_IF

(* discard calibration *)
IF bDiscardCalibration THEN
    fbScaling.ApplyCalibration(bAccept := FALSE);
    bDiscardCalibration:= FALSE;
END_IF
```

Return value

Name	Type	Description
ApplyCalibration	BOOL	TRUE if the method was executed successfully.

6.1.2.5 CalibrateRefHigh

The method can be used at runtime to trigger the fReferenceHigh calibration.

Syntax

```
METHOD CalibrateRefHigh : BOOL
VAR_INPUT
    nDurationInSamples  : UDINT;
    fRefHigh            : LREAL;
END_VAR
```

Inputs

Name	Type	Description
nDurationInSamples	UDINT	Number of samples to be averaged.
fRefHigh	LREAL	New fReferenceHigh value.

Sample

```
stParamsScale:  T_WG_Scaling := (fRawLow := 0, fRawHigh := 1, fReferenceHigh := 1,
fReferenceLow := 0);
fbScaling :FB_WG_Scaling:=(stConfig:=stParamsScale);
IF bCalibrateReferenceHigh THEN
    fbScaling.CalibrateRefHigh(nDurationInSamples := 10, fRefHigh := 1.1);
    bCalibrateReferenceHigh := FALSE;
END_IF
```

Return value

Name	Type	Description
CalibrateRefHigh	BOOL	TRUE if the method was executed successfully.

6.1.2.6 CalibrateRefLow

The method can be used at runtime to trigger the fReferenceLow calibration.

Syntax

```
METHOD CalibrateRefLow : BOOL
VAR_INPUT
    nDurationInSamples : UDINT;
    fRefLow            : LREAL;
END_VAR
```

Inputs

Name	Type	Description
nDurationInSamples	UDINT	Number of samples to be averaged.
fRefLow	LREAL	New fReferenceLow value.

Sample

```
stParamsScale: ST_WG_Scaling :=  (fRawLow := 0, fRawHigh := 1, fReferenceHigh := 1,
fReferenceLow := 0);
fbScaling :FB_WG_Scaling:=(stConfig:=stParamsScale);
IF bCalibrateReferenceLow THEN
    fbScaling.CalibrateRefLow(nDurationInSamples := 10, fRefLow := 0.1);
    bCalibrateReferenceLow := FALSE;
END_IF
```

Return value

Name	Type	Description
CalibrateRefLow	BOOL	TRUE if the method was executed successfully.

6.1.2.7 Tare

The method can be used at runtime to tare the function block. The average is calculated via nDurationInSamples - output values. Finally, the result is passed to the method UpdateTareOffset [▶ 30]().

Syntax

```
METHOD Tare : BOOL
VAR_INPUT
    nDurationInSamples : UDINT;
END_VAR
```

Inputs

Name	Type	Description
nDurationInSamples	UDINT	Number of samples to be averaged.

Sample

```
stParamsScale: ST_WG_Scaling := (fRawLow := 0, fRawHigh := 1, fReferenceHigh := 1, fReferenceLow :=
0);
fbScaling :FB_WG_Scaling:=(stConfig:=stParamsScale);
IF bTare THEN
    fbScaling.Tare(nDurationInSamples := 10);
    bTare := FALSE;
END_IF
```

Return value

Name	Type	Description
Tare	BOOL	TRUE if the method was executed successfully.

6.1.2.8 UpdateTareOffset

The method can be used at runtime to perform manual taring. This means that the fOffset value (weight) is subtracted from the calculated output values. In addition, the function block outputs tLastTare and fCurrentTareOffset (= fCurrentTareOffset- fOffset) are updated.

Syntax

```
METHOD UpdateTareOffset := BOOL
VAR_INPUT
    fOffset := LREAL;    (* It corresponds to tare weight.*)
END_VAR
```

Inputs

Name	Type	Description
fOffset	LREAL	The new tare weight.

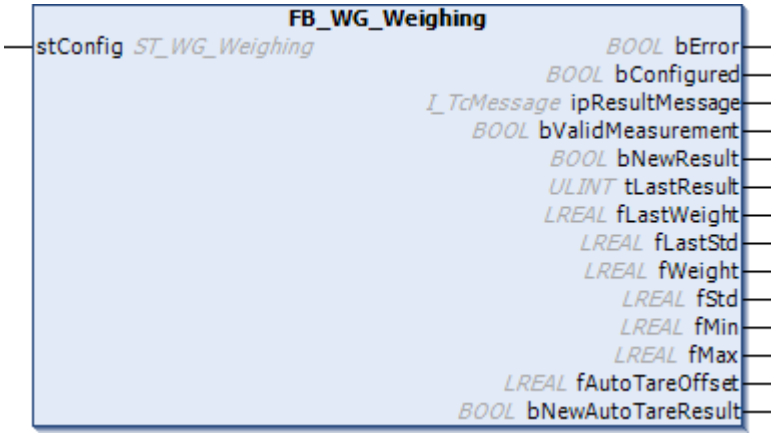
Sample

```
stParamsScale: ST_WG_Scaling :=  (fRawLow := 0, fRawHigh := 1, fReferenceHigh := 1,
fReferenceLow := 0);
fbScaling :FB_WG_Scaling:=(stConfig:=stParamsScale);
IF bUpdateTareOffset THEN
    fbScaling.UpdateTareOffset (fOffset := 5.0);
    bUpdateTareOffset := FALSE;
END_IF
```

Return value

Name	Type	Description
UpdateTareOffset	BOOL	TRUE if the method was executed successfully.

6.1.3 FB_WG_Weighing



The function block FB_WG_Weighing is used to determine a measured weight.

The configuration structure is transferred with ST_WG_Weighing [▶ 35].

Syntax

Declaration:

```
fbWeighing := FB_WG_Weighing(stConfig := ...)
```

Definition:

```
FUNCTION_BLOCK FB_WG_Weighing
VAR_INPUT
    stConfig := ST_WG_Weighing; (*The input parameters of this function block represent
initialization parameters and must already be assigned in the declaration of the FB instance!
```

```
(Alternative: Configure() method)*)
END_VAR
VAR_OUTPUT
    bValidMeasurement  : BOOL := FALSE;    // TRUE if ST_WG_Weighing_Validation-conditions are valid
(only if nWindowLength is full).
    bNewResult         : BOOL := FALSE;    // TRUE if a new result has been occured (at the end of
the Validation measurement).
    tLastResult        : ULINT := 0;       // Timestamp of new occured result.
    fLastWeight        : LREAL := 0.0;     // Last weighing result.
    fLastStd           : LREAL := 0.0;     // Last standard deviation result.
    fWeight            : LREAL := 0.0;     // Moving average of nWindowLength input values.
    fStd               : LREAL := 0.0;     // Moving standard deviation of nWindowLength input
values.
    fMin               : LREAL := 0.0;     // Minimum value of moving nWindowLength input values.
    fMax               : LREAL := 0.0;     // Maximum value of moving nWindowLength input values.
    fAutoTareOffset    : LREAL := 0.0;     // Last auto tare offset result.
    bNewAutoTareResult : BOOL := FALSE;    // TRUE if a new result has been occured (at the end of
the AutoTare measurement).
END_VAR
VAR
```

Inputs

Name	Type	Description
stConfig	ST_WG_Weighing [▶ 35]	Function block-specific configuration structure

Outputs

Name	Type	Description
bError	BOOL	TRUE, if an error occurs.
bConfigured	BOOL	TRUE if the configuration was successful.
ipResultMessage	I_TCMessage	Interface that provides properties and methods for message handling
bValidMeasurement	BOOL	TRUE if the ST_WG_Weighing_Validation [▶ 36] conditions are met.
bNewResult	BOOL	TRUE, if a new result was calculated.
tLastResult	ULINT	The timestamp of the last calculated result.
fLastWeight	LREAL	The weight (moving average) of the last result.
fLastHour	LREAL	The standard deviation of the last result.
fWeight	LREAL	The current weight (moving average) of the last result.
fStd	LREAL	The current moving standard deviation.
fMin	LREAL	Minimum fWeight weight in the sliding window.
fMax	LREAL	Maximum fWeight weight in the sliding window.
fAutoTareOffset	LREAL	The tare weight of the last result.
bNewAutoTareResult	BOOL	TRUE if a new tare weight has been calculated.

Methods

Name	Definition location	Description
Configure()	Local	Loads a new (or initial) configuration structure.
Call()	Local	Calculates the output signal for a given input signal and configuration of the function block.
Reset()	Local	Resets internal states.

Properties

Name	Type	Access	Definition location	Initial value	Description
bTraceLevelDefault	BOOL	Get, Set	Local	TRUE	TRUE, if eTraceLevel = Warning.
eTraceLevel	TcEventSeverity	Get, Set	Local	Warning	Severity of an event
nTimeStamp	ULINT	Get, Set	Local	0	Timestamp of the oldest input value of the next Call().

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.50	PC or CX (x64, x86)	Tc3_Weighing

6.1.3.1 Configure

The method can be used at runtime to initially configure the weighing instance (if not already done in the declaration) or to reconfigure it.

If a weighing instance is not configured, the methods `Call()` and `Reset()` cannot be used.

Syntax

```
METHOD Configure := BOOL
VAR_INPUT
     stConfig := ST_WG_Weighing;
END_VAR
```

Inputs

Name	Type	Description
stConfig	ST_WG_Weighing [▶ 35]	Configuration structure

Sample

```
(*Declaration without configuration*)
fbWeighing : FB_WG_Weighing();
```

```
(* initial configuration of fbWeighing *)
IF bInit THEN
    bSucceed := bWeighing.Configure(stConfig := stParams);
    bInit    := FALSE;
END_IF

(* reconfigure fbWeighing on bReconfigure := TRUE *)
IF bReconfigure THEN
    stParams.nWindowLength := 50; (*change window length*)
    bSucceed         := fbWeighing.Configure(stConfig := stParams);
    bReconfigure     := FALSE;
END_IF
```

Return value

Name	Type	Description
Configure	BOOL	TRUE if the weighing instance was configured successfully.

6.1.3.2 Call

The method calculates a manipulated output signal from an input signal that is transferred in the form of a pointer. If oversampling is used, not all information is displayed. The results of the function block outputs refer to the oldest input value of Call().

Syntax

```
METHOD Call : BOOL
VAR_INPUT
    pIn         : POINTER TO LREAL;
    nSizeIn     : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pIn	POINTER TO LREAL	Address of the input array
nSizeIn	UDINT	Size of the input array

Return value

Name	Type	Description
Call	BOOL	Returns TRUE if a manipulated output signal has been calculated.

Sample

```
aInput := ARRAY [1..cOversamples] OF LREAL;
bSucceed := fbWeighing.Call(ADR(aInput), SIZEOF(aInput));
```

6.1.3.3 Reset

The method resets the internal status of the weighing instance. By resetting the function block, the weighing instance is reset to its original state, i.e. without any influence from the past. The weighing instance is therefore reset to the last configuration status.

Syntax

```
METHOD Reset : BOOL
```

Return value

Name	Type	Description
Reset	BOOL	Returns TRUE if the internal status of the weighing instance has been successfully reset.

6.1.3.4 AutoTare

The method can be used at runtime to automatically tare a FB_WG_Scaling [▶ 24] instance.

Syntax

```
//This method tares the function block with the interface I_WG_Scaling with the current
fAutoTareOffset-value automaticly if fAutoTareOffset is not zero.
//It calls I_WG_Scaling.UpdateTareOffset(fOffset := fAutoTareOffset) and FB_WG_Weighing.Reset().
//A new FB_WG_Weighing.fAutoTareOffset value will be updated after ST_WG_Weighing.nWindowLength +
ST_WG_Weighing_AutoTare.nValidationSamples at the earliest.
METHOD AutoTare := BOOL
VAR_INPUT
    IScaling         : I_WG_Scaling;      //function block with the interface I_WG_Scaling
    eAutoTareType    : E_WG_AutoTareType; // AutoTare behaviour (end or continously)
END_VAR
```

Inputs

Name	Type	Description
IScaling	I_WG_Scaling	Function block with the interface I_WG_Scaling
eAutoTareType	E_WG_AutoTareType [▶ 39]	Behavior for automatic taring

Sample

```
// Scaling
stParamsScaling : ST_WG_Scaling :=    (
        fRawLow:=0.0,
        fReferenceLow:=0.0,
        fRawHigh:=1000.0,
        fReferenceHigh:=100.0);

fbScaling : FB_WG_Scaling:=(stConfig:=stParamsScale);

// Weighing
stParamsWeighing : ST_WG_Weighing :=    (
        nWindowLength:=100,
        Validation:=(nValidationSamples:=100, fThresholdWeight:=20.0, fMaxStd:=5.0, fMaxWeightDeviat
ion:=0.0),
        AutoTare:=(nValidationSamples:=100, fThresholdWeight:=10.0, fMaxStd:=1.0, fMaxWeightDeviatio
n:=0.0)
    );
fbWeighing : FB_WG_Weighing:=(stConfig:=stParamsWeighing);
eAutoTareType : E_WG_AutoTareType : E_WG_AutoTareType.eContinously;
fbWeighing.AutoTare(fbScaling, eAutoTareType);
```

Return value

Name	Type	Description
AutoTare	BOOL	TRUE if the method call was executed successfully.

6.2 Data types

6.2.1 Configuration structures

General description

There is an individual configuration structure ST_WG_<type> for each function block FB_WG_<type>. In the configuration structure all parameters are defined that are required for the calculation of the transfer function, the input and output variables (size and form of the arrays) as well as the internal states.

6.2.1.1 ST_WG_ComboFilter

Configuration structure for the function block FB_WG_ComboFilter [▶ 22].

```
(* Optional parameters are ignored if they are zero.*)
TYPE ST_WG_ComboFilter :
STRUCT
   nOrder            : UDINT   := 6;      (* Order has to be between one and ten. *)
   fCutoff           : LREAL   := 10.0;   (* Cutoff frequency [Hz] has to be greater than zero and
smaller or equal than fSamplingrate/2. *)
   fSamplingRate     : LREAL   := 1000.0; (* Sampling rate [Hz] has to be greater than zero. *)
   nSamplesToFilter  : UDINT   := 200;    (* Number of samples must be greater than zero. It
corresponds to the window size of the moving average filter (optional). *)
   fNotchFrequency   : LREAL   := 0.0;    (* Notch frequency [Hz] has to be greater than zero and
smaller or equal than fSamplingrate/2. The quality factor Q has a default value of 30.0 (optional).
*)
   bReset            : BOOL := TRUE;      (* Reset memory, if bReset = TRUE *)
END_STRUCT
END_TYPE
```

- `nOrder` is the filter order (1-10).

- `fCutoff` is the cut-off frequency in Hz (greater than 0 and less than `fSamplingRate` /2)
- `fSamplingRate` is the sampling rate f_s in Hz.
- `nSamplesToFilter` is the number of samples (greater than 0) to form the moving average (often referred to as the window size).
- fNotchFrequency is the notch frequency in Hz (greater than 0 and less than `fSamplingRate` /2)
- `bReset` is a Boolean parameter that specifies whether the internal past values should be reset on reconfiguration.

6.2.1.2 ST_WG_Scaling

Configuration structure for the function block FB_WG_Scaling [▶ 24].

```
TYPE ST_WG_Scaling :
STRUCT
   fRawLow          : LREAL   := 0.0;     (* fRawLow must be smaller than fRawHigh. *)
   fRawHigh         : LREAL   := 1000.0; (* fRawHigh must be greater than fRawLow. *)
   fReferenceLow    : LREAL   := 0;       (* fReferenceLow must be smaller than fReferenceHigh. *)
   fReferenceHigh   : LREAL   := 100.0;   (* fReferenceHigh must be greater than fReferenceLow. *)
END_STRUCT
END_TYPE
```

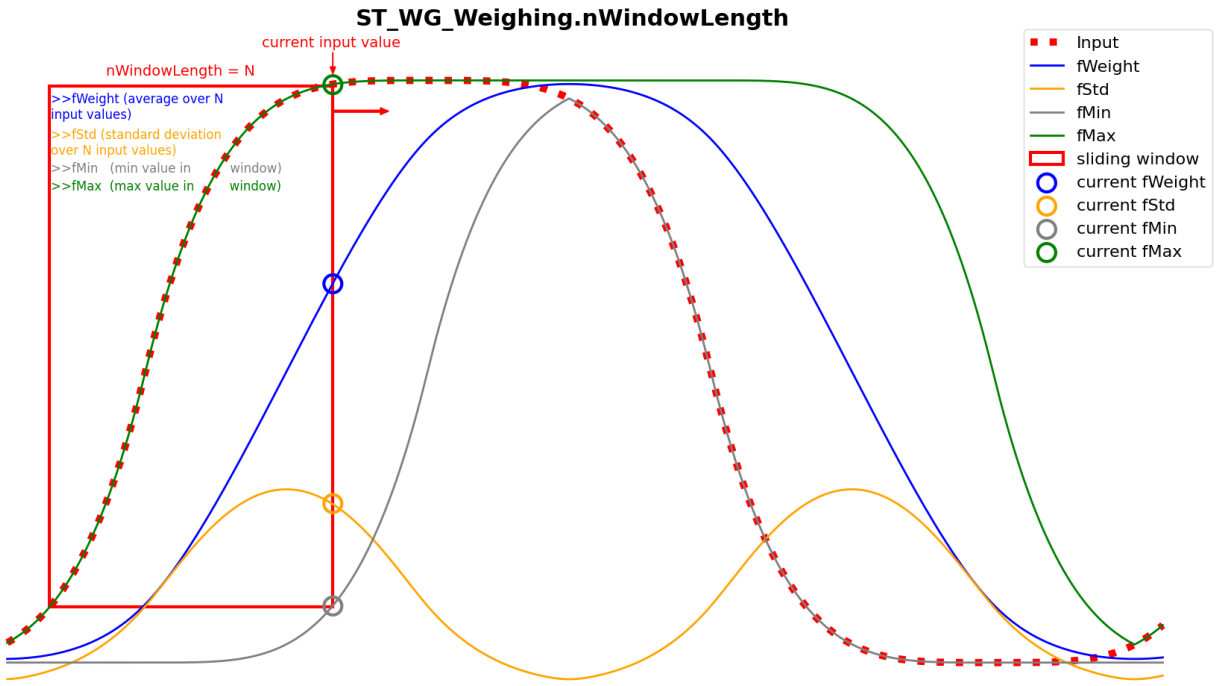
- `fRawLow` must be smaller than `fRawHigh`.
- `fRawHigh` must be greater than `fRawLow`.
- `fReferenceLow` must be smaller than `fReferenceHigh`.
- `fReferenceHigh` must be greater than `fReferenceLow`.

6.2.1.3 ST_WG_Weighing

Configuration structure for the function block FB_WG_Weighing [▶ 30].

```
TYPE ST_WG_Weighing :
STRUCT
   nWindowLength : UDINT := 100;  (* Size in samples of a sliding window and must be greater than
zero. It specifies over how many values the function block outputs fWeight, fStd, fMin and fMax
should be calculated. If the amount of existing input values is smaller than nWindowLength the
calculation will be done with the already existing values. *)
   Validation    : ST_WG_Weighing_Validation;
   AutoTare      : ST_WG_Weighing_AutoTare;
END_STRUCT
END_TYPE
```

- `nWindowLength` is the number of samples used to form the moving average (often referred to as the window size). The parameter specifies how many values are to be used to calculate the function block outputs `fWeight`, `fStd`, `fMin` and `fMax`. If the number of existing input values is less than `nWindowLength`, the calculation is carried out using the existing input values.
- `Validation` is an optional substructure that influences the function block outputs `bValidMeasurment`, `bNewResult`, `tLastResult` and `fLastWeight`.
- `AutoTare` is an optional substructure that influences the function block outputs `fAutoTareOffset` and `bNewAutoTareResult`.



6.2.1.3.1 ST_WG_Weighing_Validation

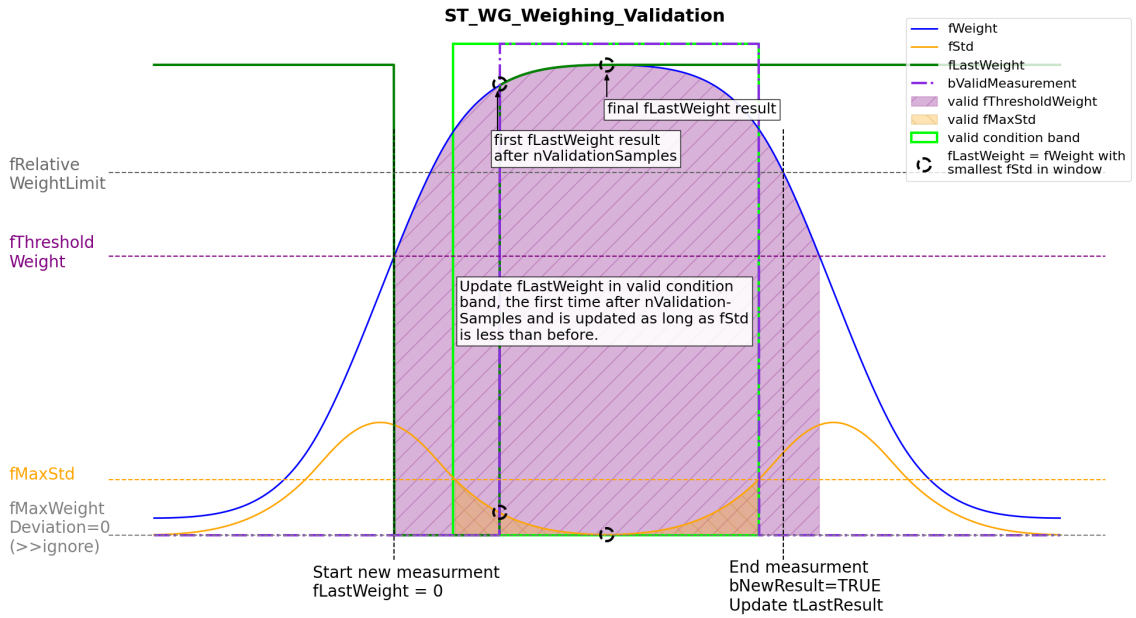
Substructure for the configuration structure ST_WG_Weighing [▶ 35]. Set parameters affect the function block outputs bValidMeasurement, bNewResult, tLastResult, fLastWeight and fLastStd [▶ 31].

```
(*This configure struct helps to find the actual weight during one measuring cycle.
A measuring cycle starts when FB_WG_Weighing.fWeight exceeds fThresholdWeight and ends when it falls
below fRelativeWeightLimit*FB_WG_Weighing.fLastWeight or fThresholdWeight (if fRelativeWeightLimit
is not set).
The actual weight will be displayed in FB_WG_Weighing.fLastWeight (FB_WG_Weighing.fWeight with
smallest FB_WG_Weighing.fStd).

Optional parameters are ignored if they are zero. None of the parameters can be less than zero.*)
TYPE ST_WG_Weighing_Validation :
STRUCT
    fThresholdWeight       : LREAL    := 50.0;    (* Minimum value for the measured weight. This
condition is fulfilled if FB_WG_Weighing.fWeight is greater than or equal to fThresholdWeight. *)
    nValidationSamples     : UDINT    := 10;      (* Number of input values for which the other
ST_WG_Weighing_Validation parameter conditions (fThresholdWeight, fMaxStd, fMaxWeightDeviation) must
be fulfilled so that FB_WG_Weighing.bValidMeasurement=TRUE (optional, recommended). *)
    fMaxStd                : LREAL    := 5.0;     (* Upper limit for the standard deviation. This
condition is fulfilled if FB_WG_Weighing.fStd is less than or equal to fMaxStd (optional,
recommended). *)
    fMaxWeightDeviation    : LREAL    := 0.0;     (* Upper limit for the maximum weight deviation.
This condition is fulfilled if FB_WG_Weighing.fMax - FB_WG_Weighing.fMin is less than or equal to
fMaxWeightDeviation (optional, recommended). *)
    fRelativeWeightLimit   : LREAL    := 0.0;     (* fRelativeWeightLimit (> 0 and < 1) specifies
that FB_WG_Weighing.bNewResult and FB_WG_Weighing.tLastResult are updated if FB_WG_Weighing.fWeight
falls below the fRelativeWeightLimit * FB_WG_Weighing.fLastWeight limit value (optional). *)
END_STRUCT
END_TYPE
```

- `fThresholdWeight` is the minimum value for the measured weight. This condition is met if `FB_WG_Weighing.fWeight` is greater than or equal to `fThresholdWeight`.
- `nValidationSamples` is the number of input values for which the other `ST_WG_Weighing_Validation` parameter conditions (`fThresholdWeight`, `fMaxStd`, `fMaxWeightDeviation`) must be fulfilled so that `FB_WG_Weighing.bValidMeasurement=TRUE` (optional, recommended).
- `fMaxStd` is the upper limit for the standard deviation. This condition is met if `FB_WG_Weighing.fStd` is less than or equal to `fMaxStd` (optional, recommended).
- `fMaxWeightDeviation` is the upper limit for the maximum weight deviation. This condition is met if `FB_WG_Weighing.fMax - FB_WG_Weighing.fMin` is less than or equal to `fMaxWeightDeviation` (optional, recommended).

- `fRelativeWeightLimit` (> 0 and < 1) specifies that `FB_WG_Weighing.bNewResult` and `FB_WG_Weighing.tLastResult` are updated if `FB_WG_Weighing.fWeight` falls below the limit value `fRelativeWeightLimit`-`FB_WG_Weighing.fLastWeight` (optional).



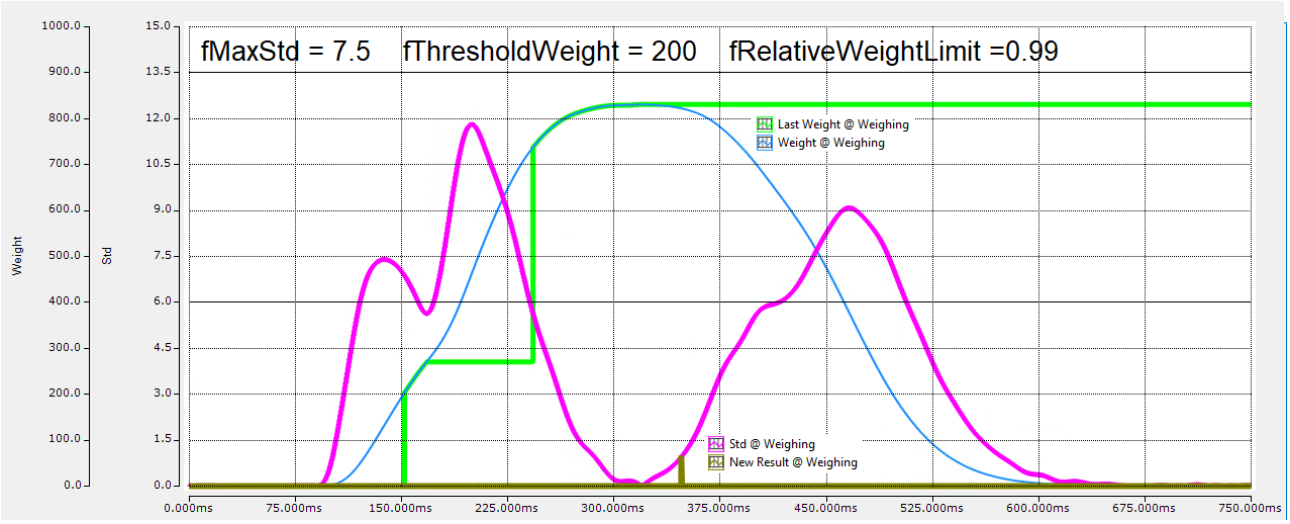
If `FB_WG_Weighing.fWeight` rises above `fThresholdWeight`, the weight with the lowest `FB_WG_Weighing.fStd` is searched for until `FB_WG_Weighing.fWeight` leaves the `fThresholdWeight` limit again. If `fRelativeWeightLimit` is set, the measurement ends when `FB_WG_Weighing.fWeight` falls below `fThresholdWeight`*`fRelativeWeightLimit`. The weight determined is displayed for the first time on a rising edge of `FB_WG_Weighing.bValidMeasurement` in `FB_WG_Weighing.fLastWeight` and is updated until the measurement is complete. At the end of the measurement, the timestamp is set in `FB_WG_Weighing.tLastResult` and `FB_WG_Weighing.bNewResult` is set to TRUE for one cycle. If `FB_WG_Weighing.fWeight` rises above `fThresholdWeight` again, `FB_WG_Weighing.fLastWeight` is set to zero and a new measurement begins.

Measurement cycle example with real data

The following figure shows that `FB_WG_Weighing.fLastWeight` remains constant after reaching a minimum of `FB_WG_Weighing.fStd`. Shortly before this, `FB_WG_Weighing.fLastWeight` adjusts continuously as long as `fStd` continues to decline.

There is therefore a stepwise increase in `Weighing.fLastWeight` between the two minima. At the global minimum of `FB_WG_Weighing.fStd` (here at 320 ms), `fLastWeight` finally assumes the final value.

As `fRelativeWeightLimit` is set to 0.99 in this example, `FB_WG_Weighing.bNewResult` is triggered shortly afterwards (at 347 ms). If `fRelativeWeightLimit` = 0, the new result would only be reported at 538 ms.



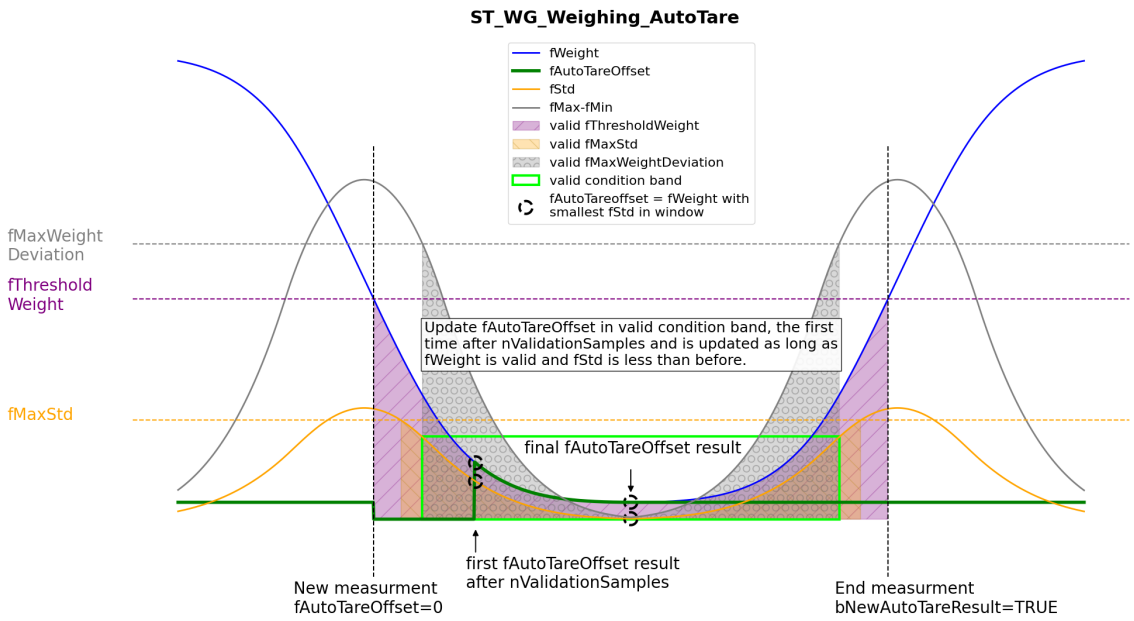
6.2.1.3.2 ST_WG_Weighing_AutoTare

Substructure for the configuration structure ST_WG_Weighing [▶ 35]. If parameters are set, this only affects FB_WG_Weighing.fAutoTareOffset and FB_WG_Weighing.bNewAutoTareResult. An instance of fbScaling from FB_WG_Scaling [▶ 24] can be automatically tared, e.g. by calling AutoTare [▶ 33](fbScaling, E_WG_AutoTareType [▶ 39].eEnd).

```
(* This configure struct helps to find the tare weight during one measuring cycle. A measuring cycle
starts when the FB_WG_Weighing.fWeight falls below fThresholdWeight and ends when it exceeds
fThresholdWeight.
The tare weight will be updated (FB_WG_Weighing.fWeight with smallest FB_WG_Weighing.fStd) in
FB_WG_Weighing.fAutoTareOffset until the measuring cycle ends. FB_WG_Scaling can be automatically
tared by calling fbWeighing.AutoTare(fbScale, E_WG_AutoTareType.eEnd).

Optional parameters are ignored if they are zero. None of the parameters can be less than zero.*)
TYPE ST_WG_Weighing_AutoTare :
STRUCT
   fThresholdWeight     : LREAL   := 20.0; (* Maximum value for the measured weight. This condition
is fulfilled if FB_WG_Weighing.fWeight is smaller than or equal to fThresholdWeight. *)
   nValidationSamples   : UDINT   := 50;   (* Number of input values for which the other
ST_WG_Weighing_AutoTare parameter conditions (fThresholdWeight, fMaxStd, fMaxWeightDeviation) must
be fulfilled in order for FB_WG_Weighing.fAutoTareOffset to be updated. (optional, recommended). *)
   fMaxStd              : LREAL   := 0.0;   (* Upper limit for the standard deviation. This condition
is fulfilled if FB_WG_Weighing.fStd is less than or equal to fMaxStd (optional, recommended). *)
   fMaxWeightDeviation  : LREAL   := 0.0;    (* Upper limit for the maximum weight deviation. This
condition is fulfilled if FB_WG_Weighing.fMax - FB_WG_Weighing.fMin is less than or equal to
fMaxWeightDeviation (optional, recommended). *)
END_STRUCT
END_TYPE
```

- `fThresholdWeight` is the maximum value for the measured weight. This condition is met if `FB_WG_Weighing.fWeight` is less than or equal to `fThresholdWeight`.
- `nValidationSamples` is the number of input values for which the other `ST_WG_Weighing_AutoTare` parameter conditions (`fThresholdWeight, fMaxStd, fMaxWeightDeviation`) must be fulfilled so that `FB_WG_Weighing.fAutoTareOffset` (optional, recommended).
- `fMaxStd` is the upper limit for the standard deviation. This condition is met if `FB_WG_Weighing.fStd` is less than or equal to `fMaxStd` (optional, recommended).
- `fMaxWeightDeviation` is the upper limit for the maximum weight deviation. This condition is met if `FB_WG_Weighing.fMax - FB_WG_Weighing.fMin` is less than or equal to `fMaxWeightDeviation` (optional, recommended).



If `FB_WG_Weighing.fWeight` falls below `fThresholdWeight`, the tare weight with the lowest `FB_WG_Weighing.fStd` is searched for until `FB_WG_Weighing.fWeight` leaves the `fThresholdWeight` limit again. The tare weight is displayed at the earliest after `nValidationSamples` values in `FB_WG_Weighing.fWeight`.

6.2.2 E_WG_Calibrate

ENUM for the calibration output of FB_WG_Scaling [▶ 24] .

Syntax

Definition:

```
TYPE E_WG_Calibrate : (
   eIdle          :=1,    (* eIdle represents no calibration. *)
   eCalibrateLow  :=2,    (* eCalibrateLow represents that the CalibrateRefLow() process is still
running. *)
   eCalibrateHigh :=3,    (* eCalibrateHigh that the CalibrateRefHigh() process is still running. *)
   eCalibrateIdle :=4,    (* eCalibrateIdle represents the temporary completed CalibrateRefLow()/
CalibrateRefHigh() process. ApplyCalibration() completes or discards the process. *)
   eTare          :=5     (* eTare represents the current Tare() process. *)
   ) UDINT
END_TYPE
```

6.2.3 E_WG_AutoTareType

ENUM for the method input of FB_WG_Weighing.AutoTare() [▶ 33].

Syntax

Definition:

```
TYPE E_WG_AutoTareType : (
    eEnd := 0,     // Tares at the end of the AutoTare measurement (if
FB_WG_Weighing.bNewAutoTareResult = TRUE).
    eContinously, // Tares continiously, if FB_WG_Weighing.fAutoTareOffset is not 0.
    eIdle          // Do nothing
   )
END_TYPE
```

7 Samples

7.1 Dynamic weighing

This sample shows how the dynamic weighing process works with the Weighing PLC library.

Download: https://infosys.beckhoff.com/content/1033/TF3685_TC3_Weighing_Library/Resources/16127833867.zip (*.tnzip)

Description

The input signal, a noisy trapezoidal signal, is generated in the MAIN program using the method `GenerateInputs()` with a signal generator. The simulated signal is transferred to the function block FB_DynamicWeighing (`fbDynamicWeighing`), which forwards it internally to other function blocks. This includes filtering with the function block FB_WG_ComboFilter [▶ 22] (`fbComboFilter`), scaling with the function block FB_WG_Scaling [▶ 24] (`fbScale`) and evaluation with the function block FB_WG_Weighing [▶ 30] (`fbWeighing`).

Program parameters

The table below shows a list of important parameters for configuring the function blocks used.

Variable	Description	Default value
fRawAmplitudeSignal	Amplitude of the trapezoidal signal	1000.0
fAbsoluteNoise	Absolute noise amount	200.0
fFrequency	Base frequency of the trapezoidal signal	1.0 Hz
bActivateSlope	A drifting test signal is activated/deactivated	FALSE
bAddWeight	Adds an offset value to the test signal once	FALSE
eAutoTareType	Selection for automatic taring	E_WG_AutoTareType.eContinously

The following global constants are defined.

Variable	Description	Default value
cOversamples	Number of oversamples of the input channel	10
cSamplingRate	Sample rate of the input channel in Hz	1000

Implementation:

First, the corresponding structures and function blocks are declared and initialized:

```
// Filter
    stParamsComboFilter : ST_WG_ComboFilter  :=
    (
        nOrder : =6,
        fCutoff : =10.0,
        fSamplingRate : =TO_LREAL(cSamplingRate),
        nSamplesToFilter : =200,
        bReset := FALSE
    );

    fbComboFilter := FB_WG_ComboFilter:=(stConfig := stParamsComboFilter);
```

```
// Scaling
    stParamsScale : ST_WG_Scaling : =
    (
        fRawLow : =0.0,
        fReferenceLow : =0.0,
        fRawHigh : =1000.0,
        fReferenceHigh : =100.0
    );

    fbScale : FB_WG_Scaling : =(stConfig : =stParamsScale, eTraceLevel : =TcEventSeverity.Info);
```

```
// Weighing
    stParamsWeighing : ST_WG_Weighing : =
    (
        nWindowLength : =100,
        Validation:=(nValidationSamples : =100, fThresholdWeight : =20.0, fMaxStd : =5.0,
fMaxWeightDeviation : =0.0),
        AutoTare : =(nValidationSamples : =100, fThresholdWeight : =10.0, fMaxStd:=1.0,
fMaxWeightDeviation : =0.0)
    );

    fbWeighing : FB_WG_Weighing : =(stConfig : =stParamsWeighing);
```

In the implementation part, the function block instances are executed using the corresponding `Call()` methods.

```
// Execute weighing
IF NOT fbComboFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutFilter), SIZEOF(aOutFilter)) THEN
    SetError(fbComboFilter);
END_IF

IF NOT fbScale.Call(ADR(aOutFilter), SIZEOF(aOutFilter), ADR(aOutScaling), SIZEOF(aOutScaling)) THEN
    SetError(fbScale);
END_IF

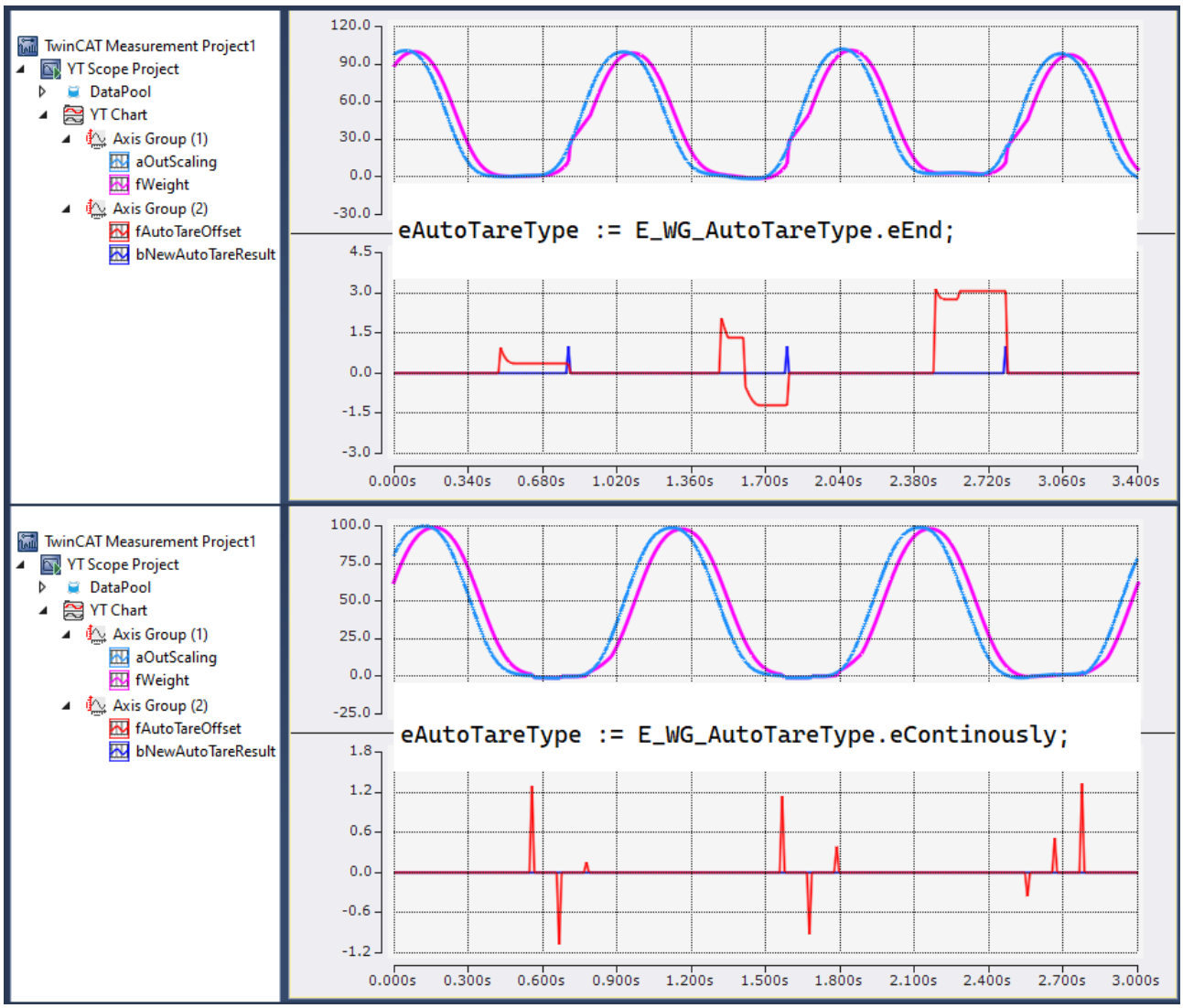
IF NOT fbWeighing.Call(ADR(aOutScaling), SIZEOF(aOutScaling)) THEN
    SetError(fbWeighing);
END_IF
```

Automatic taring:

The instance `fbScale` can be automatically tared via the `fbWeighing` instance as follows:

```
// Execute AutoTare
IF NOT fbWeighing.AutoTare(fbScale, eAutoTareType)THEN
    SetError(fbWeighing);
END_IF
```

Depending on the eAutoTareType initialization value, there is the following case distinction:



`E_WG_AutoTareType.eEnd` (above): `fbScale` is tared with the value `fAutoTareOffset` if `bNewAutoTareResult` is equal to `TRUE`. `E_WG_AutoTareType.eContinously` (below): `fbScale` is tared with the value `fAutoTareOffset` if `fAutoTareOffset` is not equal to 0. After taring, a `fbWeighing.Reset()` is automatically executed each time.

8 Appendix

8.1 Return codes

Return codes of the ipResultMessage.

Online Watch:

fbWeighing	FB_WG_Weighing	
bError	BOOL	TRUE
bConfigured	BOOL	FALSE
ipResultMessage	I_TcMessage	16#FFFFCA01F9E86638
eSeverity	TCEVENTSEVERITY	Error
ipSourceInfo	I_TcSourceInfo	16#FFFFCA01F9E86480
nEventId	UDINT	12297
sEventClassName	STRING(255)	'TcWeighingEventClass'
sEventText	STRING(255)	'WindowLength must be greater than zero.'

Defined events:

nEventId (hex)	Name	sEventText
16#0001	DbgMessage	Dbg: {0}.
16#1001	TcCom_Transition_PS_Failed	Error in Transition PREOP->SAFEOP.
16#1002	TcCom_Transition_SO_Failed	Error in Transition SAFEOP->OP.
16#1003	TcCom_Transition_SO_Failed_NoTask	Error in Transition SAFEOP->OP: No Task assigned. Module will not be executed cyclically.
16#1004	TcCom_Transition_OS_Failed	Error in Transition OP->SAFEOP.
16#1005	TcCom_Transition_SP_Failed	Error in Transition SAFEOP->PREOP.
16#1006	TcCom_CyclicCallerAssigned	Cyclic caller is assigned. Methods can not be called.
16#1007	TcCom_InvalidObjectState	Invalid object state.
16#1008	TcCom_InvalidSymbolSize	Invalid symbol size.
16#1009	TcCom_InvalidDataAreaNo	Invalid data area number.
16#2001	Init_NoRouterMemory	Memory could not be allocated dynamically. Check size of rotuer memory.
16#3001	Config_InvalidPointer	Null pointer was allocated.
16#3002	Config_NoRouterMemory	Memory could not be allocated dynamically. Check size of rotuer memory.
16#3003	Config_InvalidCutOff	Cutoff must be greater than zero and smaller than Samplingrate/2.
16#3004	Config_InvalidSamplingRate	Sampling rate must be greater than zero.
16#3005	Config_InvalidSamplesToFilter	SamplesToFilter must be greater than zero.
16#3008	Config_InvalidOrder	Order must be greater than zero and smaller than eleven.
16#3009	Config_InvalidWindowLength	WindowLength must be greater than zero.
16#3010	Config_InvalidThreshold	Threshold must be equal or greater than zero.
16#3011	Config_InvalidMaxStd	MaxStd must be equal or greater than zero.
16#3012	Config_InvalidMaxWeightDeviation	MaxWeightDeviation must be equal or greater than zero.
16#3013	Config_InvalidReferenceValue	ReferenceHigh must be greater than ReferenceLow.
16#3014	Config_InvalidRawValue	RawHigh must be greater than RawLow.
16#3015	Config_InvalidCalibrationValues	RawHigh must be greater than RawLow. ReferenceHigh must be greater than ReferenceLow.
16#3016	Config_InvalidRelativeWeightLimit	RelativeWeightLimit must be greater or equal than zero and smaller than one.
16#3017	Config_InvalidNotchFrequency	NotchFrequency must be greater than zero and smaller than fSamplingrate/2.
16#4001	Run_MissingConfiguration	Missing configuration.
16#4002	Run_InvalidPointer	Null pointer was allocated.
16#4003	Run_InvalidInputSize	Invalid input size.
16#4004	Run_InvalidOutputSize	Output size array cant be smaller than input array size.
16#6001	Warning_CalibrationProcessFailed	Calibration has not been processed successfully. RawHigh must be greater than RawLow. ReferenceHigh must be greater than ReferenceLow.

nEventId (hex)	Name	sEventText
16#6002	Warning_CouldNotTriggerTare	The tare process could not be triggered because the calibration process had not yet been completed.
16#6003	Warning_InvalidState	Invalid state.
16#7003	Warning_InvalidCutOff	Cutoff must be greater than zero and smaller than Samplingrate/2.
16#7004	Warning_InvalidSamplingRate	Sampling rate must be greater than zero.
16#7005	Warning_InvalidSamplesToFilter	SamplesToFilter must be greater than zero.
16#7008	Warning_InvalidOrder	Order must be greater than zero and smaller than eleven.
16#7009	Warning_InvalidWindowLength	WindowLength must be greater than zero.
16#7010	Warning_InvalidThreshold	Threshold must be equal or greater than zero.
16#7011	Warning_InvalidMaxStd	MaxStd must be equal or greater than zero.
16#7012	Warning_InvalidMaxWeightDeviation	MaxWeightDeviation must be equal or greater than zero.
16#7013	Warning_InvalidReferenceValue	ReferenceHigh must be greater than ReferenceLow.
16#7014	Warning_InvalidRawValue	RawHigh must be greater than RawLow.
16#7015	Warning_InvalidCalibrationValues	RawHigh must be greater than RawLow. ReferenceHigh must be greater than ReferenceLow.
16#7016	Warning_InvalidNotchFrequency	NotchFrequency must be greater than zero and smaller than fSamplingrate/2.

8.2 FAQ - frequently asked questions and answers

Frequently asked questions are answered in this section to make it easier for you to work with the TwinCAT 3 Weighing Library.
If you have further questions, please contact our support (-157).

1. Can I extend the filters of FB_WG_ComboFilter for more sophisticated applications? [▶ 45]

Can I extend the filters of FB_WG_ComboFilter for more sophisticated applications?

The FB_WG_ComboFilter already contains three different filter types that you can add depending on the application. If this is not enough, additional filters can be added using the TF3680 TwinCAT 3 filter library. For this application, the license for the TF3680 is already included in the license for the TF3685 TwinCAT 3 Weighing. An additional license is therefore not required.

9 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.

Download finder

Our download finder contains all the files that we offer you for downloading. You will find application reports, technical documentation, technical drawings, configuration files and much more.

The downloads are available in various formats.

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 our internet page: 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
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
e-mail:	service@beckhoff.com

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

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

Trademark statements

Beckhoff®, ATRO®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, MX-System®, Safety over EtherCAT®, TC/BSD®, TwinCAT®, TwinCAT/BSD®, TwinSAFE®, XFC®, XPlanar® and XTS® are registered and licensed trademarks of Beckhoff Automation GmbH.

Third-party trademark statements

Arm, Arm9 and Cortex are trademarks or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere.

Excel, IntelliSense, Microsoft, Microsoft Azure, Microsoft Edge, PowerShell, Visual Studio, Windows and Xbox are trademarks of the Microsoft group of companies.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

More Information:
www.beckhoff.com/tf3685

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