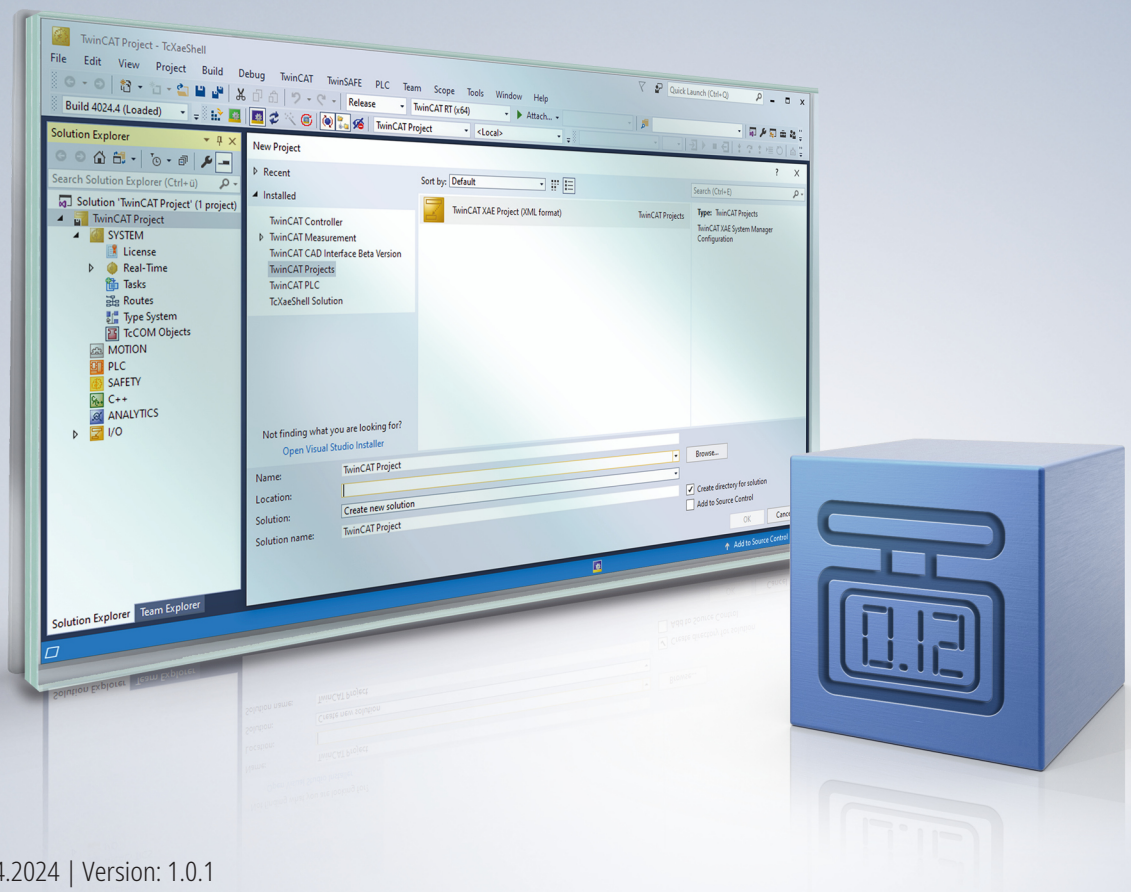
BECKHOFF New Automation Technology

Handbuch | DE

TF3685

TwinCAT 3 | Weighing Library



09.04.2024 | Version: 1.0.1

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

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

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

Disclaimer

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

Marken

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

Patente

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

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

Copyright

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

Lesen Sie die folgenden Erklärungen zu Ihrer Sicherheit.
Beachten und befolgen Sie stets produktspezifische Sicherheitshinweise, die Sie gegebenenfalls an den entsprechenden Stellen in diesem Dokument vorfinden.

Haftungsausschluss

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

Qualifikation des Personals

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

Signalwörter

Im Folgenden werden die Signalwörter eingeordnet, die in der Dokumentation verwendet werden. Um Personen- und Sachschäden zu vermeiden, lesen und befolgen Sie die Sicherheits- und Warnhinweise.

Warnungen vor Personenschäden

⚠ GEFAHR

Es besteht eine Gefährdung mit hohem Risikograd, die den Tod oder eine schwere Verletzung zur Folge hat.

⚠ WARNUNG

Es besteht eine Gefährdung mit mittlerem Risikograd, die den Tod oder eine schwere Verletzung zur Folge haben kann.

⚠ VORSICHT

Es besteht eine Gefährdung mit geringem Risikograd, die eine mittelschwere oder leichte Verletzung zur Folge haben kann.

Warnung vor Umwelt- oder Sachschäden

HINWEIS

Es besteht eine mögliche Schädigung für Umwelt, Geräte oder Daten.

Information zum Umgang mit dem Produkt

Diese Information beinhaltet z. B.:
Handlungsempfehlungen, Hilfestellungen oder weiterführende Informationen zum Produkt.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter https://www.beckhoff.de/secinfo.

2 Übersicht

Die TwinCAT 3 Weighing SPS-Bibliothek erlaubt es, insbesondere mit den I/Os der EtherCAT-Klemmen ELM35xx und EL3356-0010, eine Waage für die Gewichtsmessung in die PC-basierte Maschinensteuerung zu integrieren. Der Fokus liegt dabei hauptsächlich auf dem Prozess des dynamischen Wiegens. Dabei ist die Signalfilterung besonders anspruchsvoll, da die Wiegezeit maßgeblich die Gesamtprozesszeit der Maschine beeinflusst. Eine schnelle Signalfilterung bei gleicher Präzision bedeutet folglich ein schnelleres Gewichtsergebnis, was sich dann in schnelleren Maschinen widerspiegelt.

Da eine Wägezelle und eine Messwerterfassung über die entsprechenden EtherCAT-Klemmen noch keine Waage abbilden, setzt genau dort diese SPS-Bibliothek an. Es wird die Skalierung der Messwerte übernommen, und Funktionen wie Nullsetzen und Tarieren werden ebenfalls über die neuen SPS-Bausteine ermöglicht. Neben einer manuellen Triggerung der Gewichtsmessung ist auch eine automatische Messung möglich. Das Produktionsgut wird erkannt und die Messung direkt durchgeführt. Der wesentliche Vorteil ist dabei, dass je nach Applikation sogar externe Trigger, wie Lichtschranken und Initiatoren, entfallen können.

Die Lizenz der TF3680 | TwinCAT 3 Filter Bibliothek ist ebenfalls in diesem Produkt enthalten.

Komponenten

- TwinCAT Weighing SPS-Bibliothek: Tc3_Weighing.compiled-library
- TwinCAT Filter SPS-Bibliothek: Tc3_Filter.compiled-library
- Versionierter Treiber: TcWeighing.tmx
- Beschreibungsdatei: TcWeighing.tmc

3 Installation

Systemvoraussetzungen

Technische Daten	Beschreibung
Betriebssystem	Windows 10, TwinCAT/BSD
Zielplattform	PC-Architektur (x86,x64)
Minimale TwinCAT-Version	3.1.4024.50
Erforderliche TwinCAT-Lizenz	TF3685 TwinCAT 3 Weighing

TwinCAT Package Manager: Installation (TwinCAT 3.1 Build 4026)

Eine ausführliche Anleitung zur Installation von Produkten finden Sie im Kapitel Workloads installieren in der Installationsanleitung TwinCAT 3.1 Build 4026.

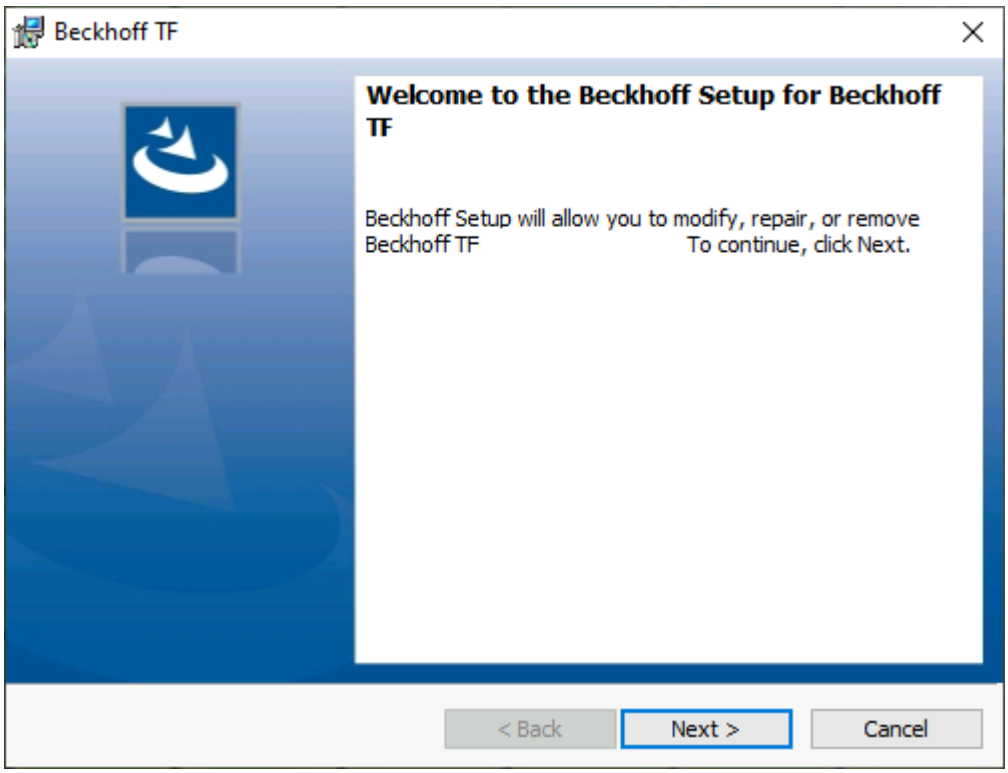
Installieren Sie den folgenden Workload, um das Produkt nutzen zu können:

- TF3685 | TwinCAT 3 Weighing

TwinCAT Setup: Installation (TwinCAT 3.1 Build 4024 und früher)

Wenn Sie TwinCAT 3.1 Build 4024 auf dem Betriebssystem Microsoft Windows verwenden, können Sie diese Function über ein Setup-Paket installieren, welches Sie auf der Beckhoff Webseite unter https://www.beckhoff.com/download herunterladen können.

Die Installation kann hierbei sowohl auf Engineering- als auch Runtime-Seite erfolgen, je nachdem, auf welchem System Sie die Function benötigen. Der folgende Screenshot zeigt die Setup-Oberfläche.



Zur Durchführung des Installationsvorgangs, folgen Sie den entsprechenden Anweisungen im Setup-Dialog.

HINWEIS

Unvorbereiteter TwinCAT-Neustart kann Datenverlust erzeugen

Die Installation dieser Function hat unter Umständen einen TwinCAT-Neustart zur Folge.
Stellen Sie sicher, dass keine kritischen TwinCAT-Applikationen auf dem System laufen oder fahren Sie diese zunächst geordnet herunter.

3.1 Lizenzierung

Die TwinCAT 3 Function ist als Vollversion oder als 7-Tage-Testversion freischaltbar. Beide Lizenztypen sind über die TwinCAT-3-Entwicklungsumgebung (XAE) aktivierbar.

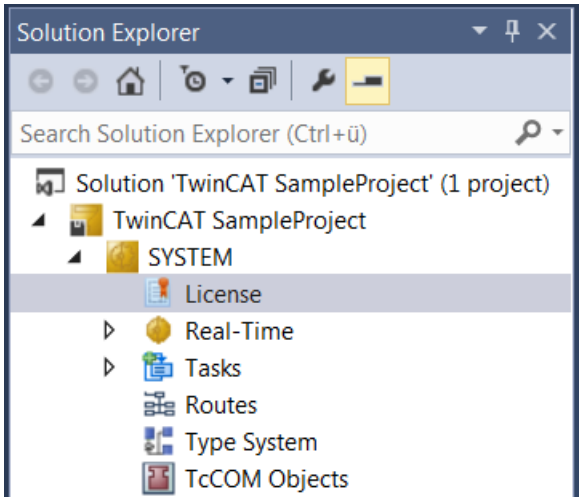
Lizenzierung der Vollversion einer TwinCAT 3 Function

Die Beschreibung der Lizenzierung einer Vollversion finden Sie im Beckhoff Information System in der Dokumentation „TwinCAT 3 Lizenzierung“.

Lizenzierung der 7-Tage-Testversion einer TwinCAT 3 Function

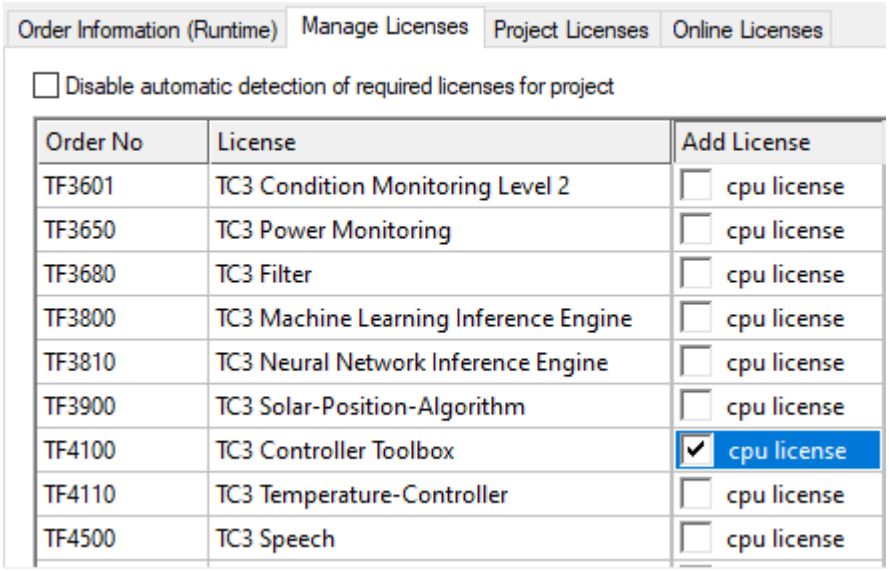
Eine 7-Tage-Testversion kann nicht für einen TwinCAT-3-Lizenz-Dongle freigeschaltet werden.

1. Starten Sie die TwinCAT-3-Entwicklungsumgebung (XAE).
2. Öffnen Sie ein bestehendes TwinCAT-3-Projekt oder legen Sie ein neues Projekt an.
3. Wenn Sie die Lizenz für ein Remote-Gerät aktivieren wollen, stellen Sie das gewünschte Zielsystem ein. Wählen Sie dazu in der Symbolleiste in der Drop-down-Liste **Choose Target System** das Zielsystem aus.
   ⇨ Die Lizenzierungseinstellungen beziehen sich immer auf das eingestellte Zielsystem. Mit der Aktivierung des Projekts auf dem Zielsystem werden automatisch auch die zugehörigen TwinCAT-3-Lizenzen auf dieses System kopiert.
4. Klicken Sie im **Solution Explorer** im Teilbaum **SYSTEM** doppelt auf **License.**



   ⇨ Der TwinCAT-3-Lizenzmanager öffnet sich.

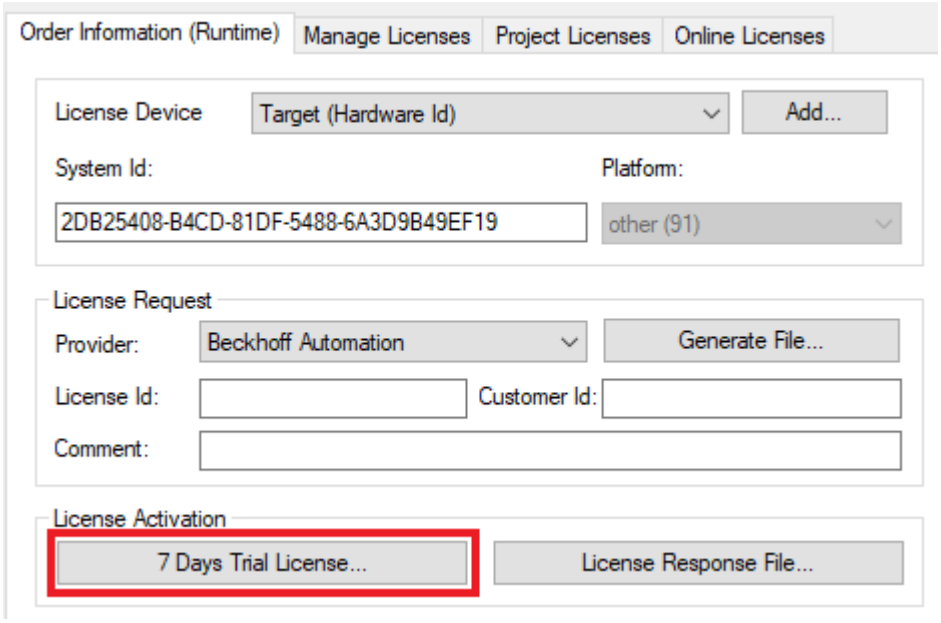
5. Öffnen Sie die Registerkarte **Manage Licenses.** Aktivieren Sie in der Spalte **Add License** das Auswahlkästchen für die Lizenz, die Sie Ihrem Projekt hinzufügen möchten (z. B. „TF4100 TC3 Controller Toolbox“).



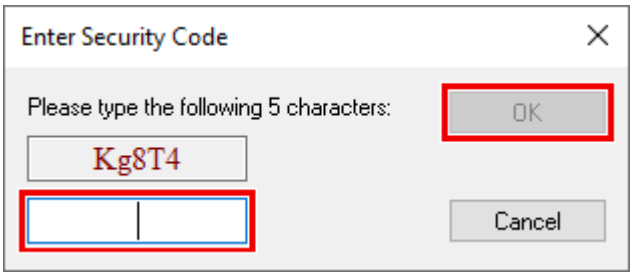
6. Öffnen Sie die Registerkarte **Order Information (Runtime).**

   ⇨ In der tabellarischen Übersicht der Lizenzen wird die zuvor ausgewählte Lizenz mit dem Status „missing“ angezeigt**.**

7. Klicken Sie auf **7 Days Trial License...**, um die 7-Tage-Testlizenz zu aktivieren.



   ⇨ Es öffnet sich ein Dialog, der Sie auffordert, den im Dialog angezeigten Sicherheitscode einzugeben.



8. Geben Sie den Code genauso ein, wie er angezeigt wird, und bestätigen Sie ihn.
9. Bestätigen Sie den nachfolgenden Dialog, der Sie auf die erfolgreiche Aktivierung hinweist.

⇨ In der tabellarischen Übersicht der Lizenzen gibt der Lizenzstatus nun das Ablaufdatum der Lizenz an.

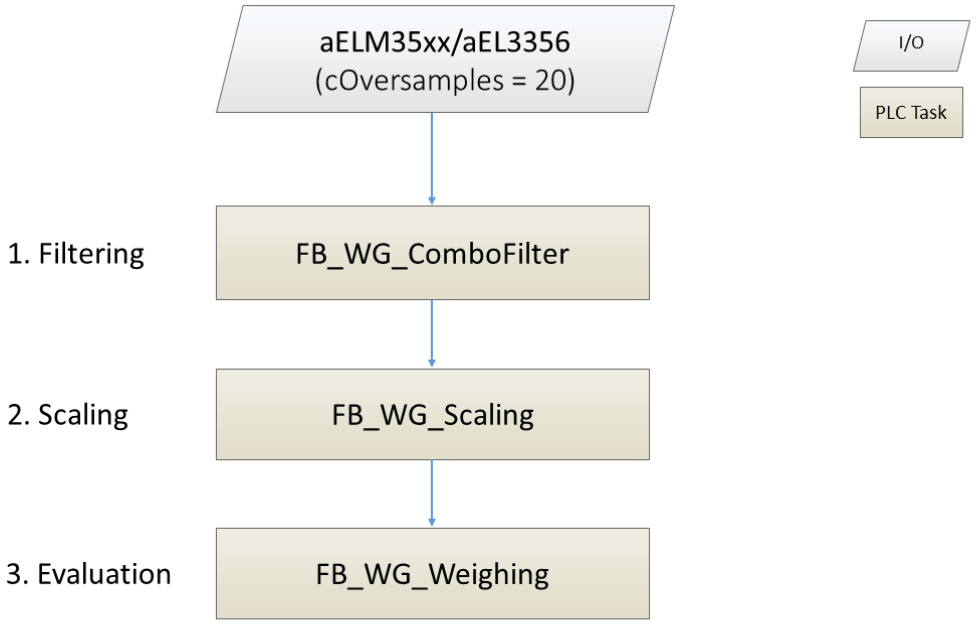
10. Starten Sie das TwinCAT-System neu.

⇨ Die 7-Tage-Testversion ist freigeschaltet.

4 Technische Einführung

4.1 Messablauf

Ein typischer Messablauf zur Bestimmung der Masse eines Körpers umfasst drei wesentliche Schritte: Filtern, Skalieren und Auswerten.



1. Filtern

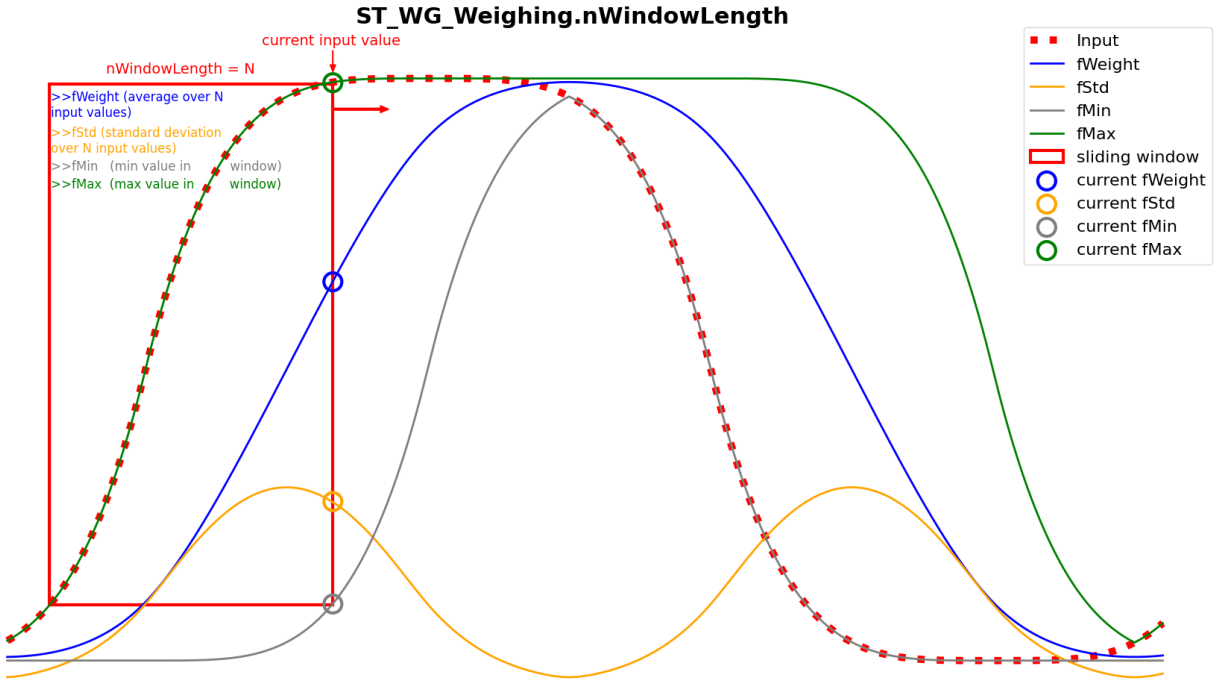
Das Signal der EtherCAT-Klemmen der Baureihen ELM35xx und EL3356 weist typischerweise ein Maß an Rauschen auf, das eine Filterung erforderlich macht, um valide Messergebnisse zu gewährleisten. Der Funktionsbaustein FB_WG_ComboFilter [▶ 19] bietet hierfür eine effektive Lösung, indem er eine Kombination aus PTn-, Moving-Average- und Notch-Filter in Serie schaltet. Sollten diese Filteroptionen nicht ausreichend sein, steht eine Auswahl weiterer Filter aus der TwinCAT 3 Filter Bibliothek zur Verfügung.

2. Skalieren

Das gefilterte Eingangssignal muss anschließend skaliert werden, um das Gewicht in der gewünschten Einheit (z. B. Gramm [g]) anzugeben. Die Skalierung erfolgt durch den Funktionsbaustein FB_WG_Scaling [▶ 22]. Für eine akkurate Messung ist zudem eine Kalibrierung des SPS-Bausteins erforderlich, die beispielsweise mittels einer Zweipunktkalibrierung durchgeführt werden kann.

3. Auswerten

Im abschließenden Schritt wird das skalierte Signal mit Hilfe des Funktionsbausteins FB_WG_Weighing [▶ 28] analysiert. Dabei ist es erforderlich, die Konfigurationsstruktur ST_WG_Weighing [▶ 33] mit den passenden Parametern zu versehen. Ein Schlüsselparameter in dieser Struktur ist `ST_WG_Weighing.nWindowLength`, der die Anzahl der Samples definiert, die für die Berechnung des gleitenden Mittelwerts herangezogen werden – eine Größe, die auch als Fenstergröße bekannt ist. Dieser Parameter bestimmt, auf Basis wie vieler vergangener Werte die Ausgaben `fWeight`, `fStd`, `fMin` und `fMax` des Bausteins FB_WG_Weighing [▶ 28] kalkuliert werden. Hierbei repräsentiert `fWeight` den Mittelwert, `fStd` die Standardabweichung und `fMin`/`fMax` den minimalen bzw. maximalen Wert der letzten `nWindowLength` Eingangswerte. Eine ergänzende Abbildung kann diese Zusammenhänge veranschaulichen.



Um zusätzliche Ergebnisse des Funktionsbausteins wie `bValidMeasurement`, `bNewResult`, `tLastResult`, `fLastWeight` und `fLastStd` zu erhalten, ist es erforderlich, die Unterstruktur ST_WG_Weighing_Validation [▶ 34] entsprechend zu konfigurieren.

Innerhalb von ST_WG_Weighing_Validation [▶ 34] definieren die Parameter `fThresholdWeight`, `fMaxWeightDeviation` und `fMaxStd` die Kriterien für eine gültige Messung. Damit eine Messung als gültig gilt, müssen folgende Bedingungen erfüllt sein:

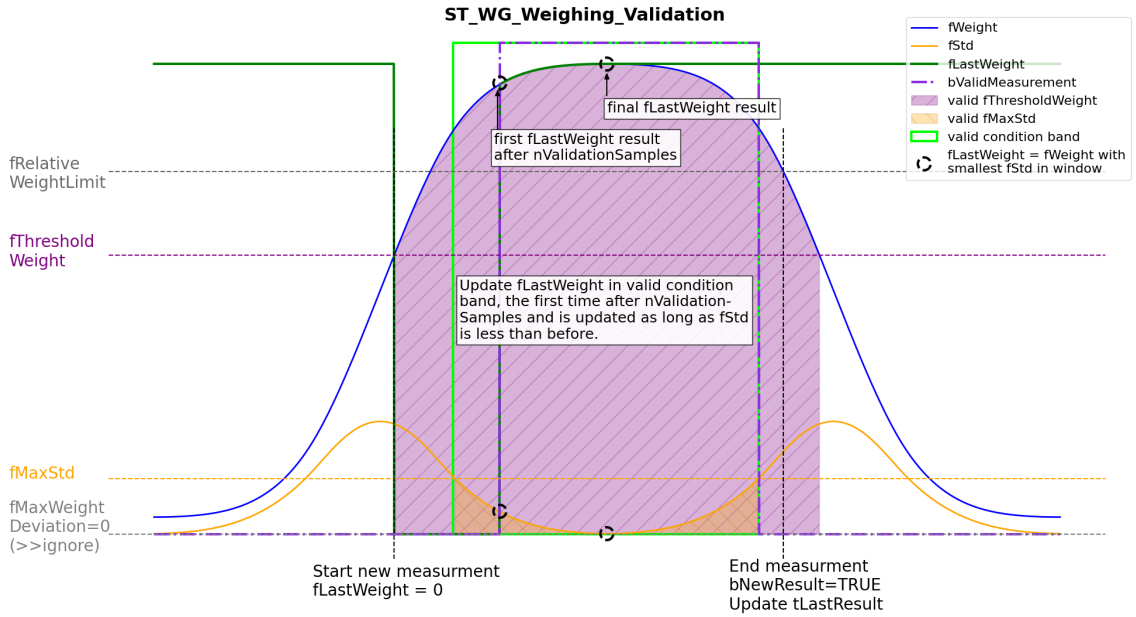
- `FB_WG_Weighing.fWeight` muss größer oder gleich `fThresholdWeight` sein.
- Die Differenz `FB_WG_Weighing.fMax - FB_WG_Weighing.fMin` darf `fMaxWeightDeviation` nicht überschreiten.
- `FB_WG_Weighing.fStd` muss kleiner oder gleich `fMaxStd` sein.

Diese Bedingungen müssen über die Anzahl der in `nValidationSamples` definierten aufeinanderfolgenden Samples hinweg erfüllt werden, um `FB_WG_Weighing.bValidMeasurement` auf `TRUE` zu setzen.

Die Messung wird eingeleitet, sobald `FB_WG_Weighing.fWeight` erstmalig den Wert von `ST_WG_Weighing_Validation.fThresholdWeight` übersteigt. Solange `FB_WG_Weighing.bValidMeasurement` den Wert `TRUE` hat, wird das Gewicht (`fWeight`) mit der kleinsten Standardabweichung (`fStd`) gesucht und kontinuierlich in `FB_WG_Weighing.fLastWeight` und in `FB_WG_Weighing.fLastStd` aktualisiert. Wenn der Parameter `ST_WG_Weighing_Validation.fRelativeWeightLimit` definiert ist, endet die Messung, sobald `FB_WG_Weighing.fWeight` unter den Wert von `fThresholdWeight * fRelativeWeightLimit` fällt. Ist dieser Parameter nicht gesetzt, endet die Messung, wenn `fWeight` unter `fThresholdWeight` fällt.

Am Ende der Messung wird der Zeitstempel in `FB_WG_Weighing.tLastResult` festgehalten und `FB_WG_Weighing.bNewResult` für genau einen Zyklus auf `TRUE` gesetzt. Wenn `FB_WG_Weighing.fWeight` erneut den Wert `fThresholdWeight` übersteigt, wird `FB_WG_Weighing.fLastWeight` zurückgesetzt und eine neue Messung beginnt.

In der nachfolgenden Abbildung wird der beschriebene Prozess noch einmal visuell verdeutlicht und zeigt den Zusammenhang zwischen den Parametern und den Bedingungen für eine gültige Messung:



ST_WG_Weighing_AutoTare [▶ 35] lässt sich analog zu ST_WG_Weighing_Validation [▶ 34] konfigurieren, um Ergebnisse wie `fAutoTareOffset` und `bNewAutoTareResult` vom Funktionsbaustein zu erhalten. Diese sind essenziell, um den Funktionsbaustein FB_WG_Scaling [▶ 22] automatisch zu tarieren, etwa durch den Aufruf von AutoTare [▶ 31](fbScaling,E_WG_AutoTareType [▶ 36].`eEnd`).

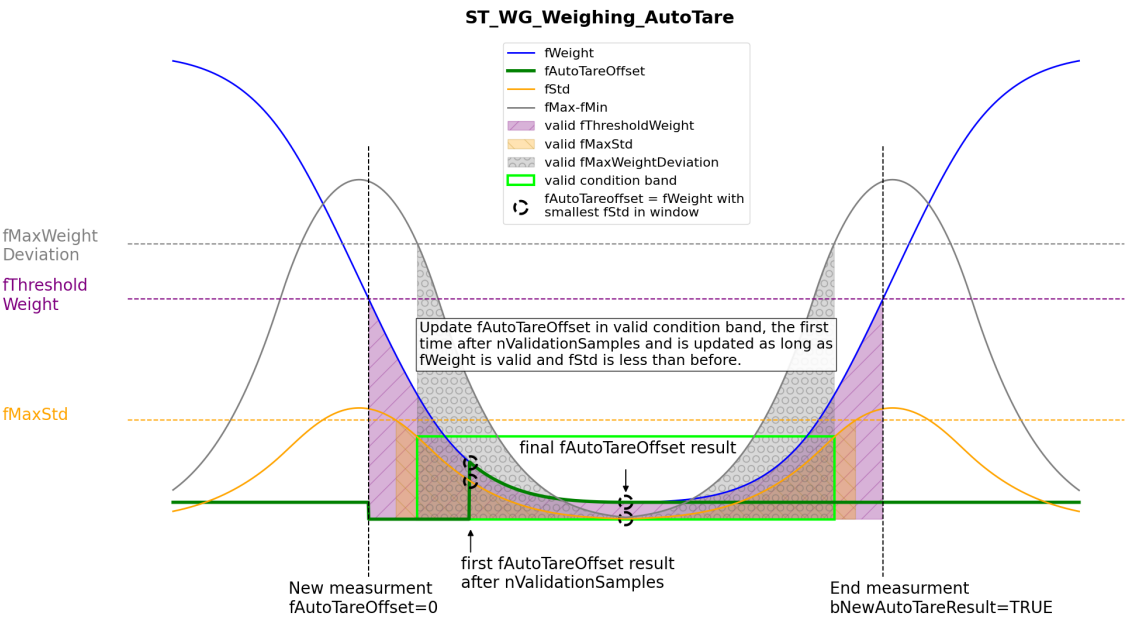
Dabei legen die Parameter `fThresholdWeight, fMaxWeightDeviation` und `fMaxStd` in ST_WG_Weighing_AutoTare [▶ 35] die Kriterien für die Validierung einer Messung fest. Eine Messung gilt als gültig, wenn:

- `FB_WG_Weighing.fWeight` den Wert `fThresholdWeight` nicht übersteigt.
- Die Differenz `FB_WG_Weighing.fMax - FB_WG_Weighing.fMin` die festgelegte `fMaxWeightDeviation` nicht überschreitet.
- `FB_WG_Weighing.fStd` kleiner oder gleich dem definierten `fMaxStd` ist.

Die Aktualisierung von `fAutoTareOffset` beginnt, sobald `fWeight` erstmals unter `fThresholdWeight` fällt und die genannten Bedingungen über eine Reihe von in `nValidationSamples` definierten aufeinanderfolgenden Samples hinweg erfüllt sind. Das System sucht das Gewicht mit der niedrigsten Standardabweichung und aktualisiert kontinuierlich `fAutoTareOffset`.

Die Messung endet, sobald `FB_WG_Weighing.fWeight` den `fThresholdWeight` übersteigt. Daraufhin wird `FB_WG_Weighing.bNewAutoTareResult` einmalig auf `TRUE` gesetzt, was das Ende der Messung signalisiert. Fällt `FB_WG_Weighing.fWeight` unter den Schwellenwert `fThresholdWeight`, wird `FB_WG_Weighing.fAutoTareOffset` zurückgesetzt und eine neue Messung eingeleitet.

Die nachstehende Abbildung veranschaulicht den Ablauf und verdeutlicht die Beziehung zwischen den Parametern und den Kriterien für eine gültige Messung.


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

5 SPS-API

5.1 Funktionsbausteine

Grundstruktur der Funktionsbausteine

Alle Funktionsbausteine der Bibliothek TwinCAT Weighing SPS-Bibliothek basieren auf derselben Grundstruktur. Dies erleichtert das Engineering, wenn von einem Weighing-Typ zu einem anderen gewechselt wird.

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
```

Eingänge

Zur Konfiguration des Weighing-Bausteins wird den Funktionsbausteinen bei der Instanziierung eine Konfigurationsstruktur vom Typ ST_WG_<type> übergeben. Die Konfigurationsstruktur kann in der Deklaration oder über die Methode `Configure()` zur Laufzeit zugewiesen werden.

Siehe auch: Datentypen [▶ 32] > Konfigurationsstrukturen [▶ 32]

Beispiel zur Konfiguration in der Deklaration:

```
(* 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);
```

Ausgänge

Alle Funktionsbausteine haben als Ausgangsparameter ein Error-Flag `bError` und ein Flag `bConfigured` vom Typ `BOOL`. Diese zeigen an, ob ein Fehler vorliegt und ob die zugehörige Funktionsbausteininstanz erfolgreich konfiguriert wurde. Der Ausgang `ipResultMessage` vom Typ `I_TcMessage` bietet verschiedene Eigenschaften zur Erläuterung einer Event-Ursache sowie Methoden zur Verarbeitung der Message (Event-Liste).

Siehe auch: I_TcEventBase und I_TcMessage

Methoden

Alle Funktionsbausteine der Bibliothek Tc3_Weighing verfügen über drei Methoden. Diese liefern einen positiven Rückgabewert, wenn sie fehlerfrei ausgeführt wurden.

Configure()

Die Methode kann zur Laufzeit verwendet werden, um die Instanz eines Weighing-Bausteins initial zu konfigurieren (wenn nicht schon in der Deklaration geschehen) oder zu rekonfigurieren.

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

Call()

Die Methode berechnet aus einem Eingangssignal, das in Form eines Pointers übergeben wird, ein manipuliertes Ausgangssignal.

```
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()

Die Methode setzt den internen Status eines Weighing-Bausteins zurück. Der Einfluss der vergangenen Werte auf den aktuellen Ausgangswert wird eliminiert.

```
METHOD Reset : BOOL
```

Eigenschaften

Die Bibliothek Tc3_Weighing referenziert auf den TwinCAT 3 EventLogger und stellt somit sicher, dass Informationen (Ereignisse) über die standardisierte Schnittstelle I_TcMessage bereitgestellt werden.

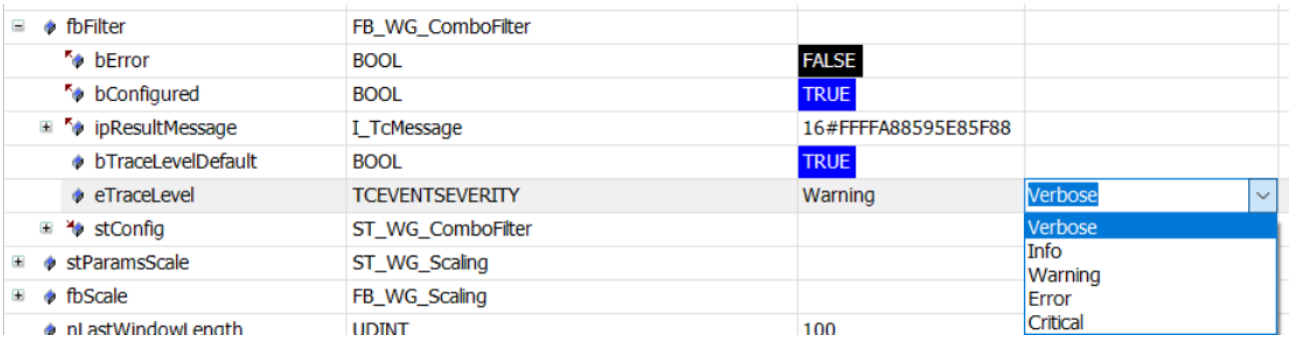
Jeder Funktionsbaustein verfügt über die Eigenschaften `eTraceLevel` vom Typ `TcEventSeverity` und `eTraceLevelDefault` vom Typ `BOOL`.

Das Trace-Level bestimmt die Severity eines Events (Verbose, Info, Warning, Error, Critical) und wird über die Eigenschaft `eTraceLevel` gesetzt.

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

Über die Eigenschaft `eTraceLevelDefault` kann das Trace-Level wieder auf den Standardwert (`TcEventSeverity.Warning`) gesetzt werden. Auf die Eigenschaft kann lesend und schreibend zugegriffen werden, d. h. über die Eigenschaft `eTraceLevelDefault` kann abgefragt werden, ob der Standardwert gesetzt ist.

Die Eigenschaften können auch im Online View gesetzt werden.



Umgang mit Oversampling

Alle Funktionsbausteine sind Oversampling-fähig, wobei unterschiedliche Arten der Nutzung möglich sind. Die Deklaration der Weighing-Funktionsbausteininstanz `fbFilter` ist hier immer gleich.

Einkanalige Anwendung mit Oversamples

Die Ein- und Ausgangsarrays können als eindimensionale Größen deklariert werden.

```
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));
```

Einkanalige Anwendung ohne Oversamples

Wenn kein Oversampling angewendet wird, können die Ein- und Ausgangsgrößen auch als `LREAL` deklariert werden.

```
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));
```

5.1.1 FB_WG_ComboFilter

Der Funktionsbaustein FB_WG_ComboFilter realisiert einen in Serie geschalteter PTn-, Moving-Average- und Notch-Filter.

Die Filterspezifikation wird mit der Struktur ST_WG_ComboFilter [▶ 32] übergeben.

Syntax

Deklaration:

```
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
```

Eingänge

Name	Typ	Beschreibung
stConfig	ST_WG_ComboFilter [▶ 32]	Struktur zur Konfiguration des Filterverhaltens

Ausgänge

Name	Typ	Beschreibung
bError	BOOL	TRUE, wenn ein Fehler auftritt.
bConfigured	BOOL	TRUE, wenn die Konfiguration erfolgreich war.
ipResultMessage	I_TCMessage	Schnittstelle, die Eigenschaften und Methoden zum Messagehandling bereitstellt.

Methoden

Name	Definitionsort	Beschreibung
Configure()	Lokal	Lädt eine neue (oder initiale) Konfigurationsstruktur.
Call()	Lokal	Berechnet das Ausgangssignal bei gegebenem Eingangssignal und Konfiguration des Funktionsbausteins.
Reset()	Lokal	Setzt interne Zustände zurück.

Eigenschaften

Name	Typ	Zugriff	Definitionsort	Initialwert	Beschreibung
bTraceLevelDefault	BOOL	Get, Set	Lokal	TRUE	TRUE, wenn eTraceLevel = Warning.
eTraceLevel	TcEventSeverity	Get, Set	Lokal	Critical	Severity eines Events

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4024.50	PC oder CX (x64, x86)	Tc3_Weighing

5.1.1.1 Configure

Die Methode kann zur Laufzeit verwendet werden, um die Instanz eines Filters initial zu konfigurieren (wenn nicht schon in der Deklaration geschehen) oder zu rekonfigurieren.

Wenn eine Filterinstanz nicht konfiguriert ist, können die Methoden `Call()` und `Reset()` nicht verwendet werden.

Syntax

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

Eingänge

Name	Typ	Beschreibung
stConfig	ST_WG_ComboFilter [▶ 32]	Struktur zur Konfiguration des Filterverhaltens

Beispiel

```
(*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
```

Rückgabewert

Name	Typ	Beschreibung
Configure	BOOL	TRUE, wenn die Weighing-Instanz erfolgreich konfiguriert wurde.

5.1.1.2 Call

Die Methode berechnet aus einem Eingangssignal, das in Form eines Pointers übergeben wird, ein manipuliertes Ausgangssignal.

Syntax

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

Eingänge

Name	Typ	Beschreibung
pIn	POINTER TO LREAL	Adresse des Eingangsarrays
nSizeIn	UDINT	Größe des Eingangsarrays
pOut	POINTER TO LREAL	Adresse des Ausgangsarrays
nSizeOut	UDINT	Größe des Ausgangsarrays

Rückgabewert

Name	Typ	Beschreibung
Call	BOOL	Liefert TRUE, wenn ein manipuliertes Ausgangssignal berechnet wurde.

Beispiel

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

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

5.1.1.3 Reset

Die Methode setzt den internen Status der Weighing-Instanz zurück. Durch das Reset des Funktionsbausteins wird die Weighing-Instanz in seinen Ursprungszustand, d. h. ohne Vergangenheitseinfluss, versetzt. Die Weighing-Instanz wird also wieder auf den letzten Konfigurationszustand zurückgesetzt.

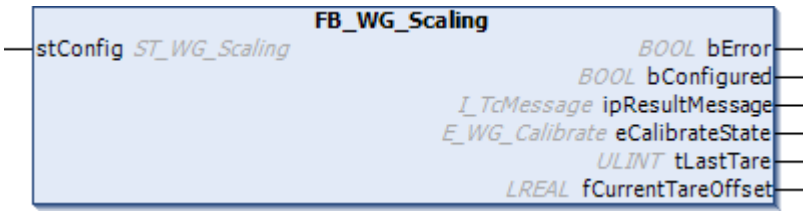
Syntax

```
METHOD Reset : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
Reset	BOOL	Liefert TRUE, wenn der interne Status der Weighing-Instanz erfolgreich zurückgesetzt wurde.

5.1.2 FB_WG_Scaling



Der Funktionsbaustein FB_WG_Scaling dient der Skalierung von Rohwerten. Die Rohwerte können sowohl einzeln als auch als Array, beispielsweise als Oversampling-Werte, skaliert werden.

Die Konfigurationsstruktur wird mit ST_WG_Scaling [▶ 32] übergeben.

Syntax

Deklaration:

```
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
```

Eingänge

Name	Typ	Beschreibung
stConfig	ST_WG_Scaling [▶ 32]	Bausteinspezifische Konfigurationsstruktur

Ausgänge

Name	Typ	Beschreibung
bError	BOOL	TRUE, wenn ein Fehler auftritt.
bConfigured	BOOL	TRUE, wenn die Konfiguration erfolgreich war.
ipResultMessage	I_TCMessage	Schnittstelle, die Eigenschaften und Methoden zum Messagehandling bereitstellt.
eCalibrateState	E_WG_Calibrate [▶ 36]	Aktueller calibrate/tare Zustand.
tLastTare	ULINT	Zeitstempel des letzten Tare()/ [▶ 26]UpdateTareOffset()- [▶ 27]Methodenaufruf.
fCurrentTareOffset	LREAL	Aktualisiert sich bei jedem Tare()/ [▶ 26]UpdateTareOffset()- [▶ 27]Methodenaufruf.

Methoden

Name	Definitionsort	Beschreibung
Configure()	Lokal	Lädt eine neue (oder initiale) Konfigurationsstruktur.
Call()	Lokal	Berechnet das Ausgangssignal bei gegebenem Eingangssignal und Konfiguration des Funktionsbausteins.
Reset()	Lokal	Setzt interne Zustände zurück.
ApplyCalibration()	Lokal	Beendet den Kalibrierungsprozess.
CalibrateRefHigh()	Lokal	Triggert die fReferenceHigh Kalibrierung an.
CalibrateRefLow()	Lokal	Triggert die fReferenceLow Kalibrierung an.
Tare()	Lokal	Triggert die Tara Kalibrierung an.
UpdateTareOffset()	Lokal	Setzt den Tara Offset Wert manuell und aktualisiert den tLastTare/fCurrentTareOffset Ausgang.

Eigenschaften

Name	Typ	Zugriff	Definitionsort	Initialwert	Beschreibung
bTraceLevelDefault	BOOL	Get, Set	Lokal	TRUE	TRUE, wenn eTraceLevel = Warning
eTraceLevel	TcEventSeverity	Get, Set	Lokal	Warning	Severity eines Events
nTimeStamp	ULINT	Get, Set	Lokal	0	Zeitstempel des ältesten Eingangswert des nächsten Call()-Aufrufs.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4024.50	PC oder CX (x64, x86)	Tc3_Weighing

5.1.2.1 Configure

Die Methode kann zur Laufzeit verwendet werden, um die Instanz eines Filters initial zu konfigurieren (wenn nicht schon in der Deklaration geschehen) oder zu rekonfigurieren.

Wenn eine Weighing-Instanz nicht konfiguriert ist, können die Methoden `Call()` und `Reset()` nicht verwendet werden.

Syntax

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

Eingänge

Name	Typ	Beschreibung
stConfig	ST_WG_Scaling [▶ 32]	Struktur zur Konfiguration des Filterverhaltens

Beispiel

```
(*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
```

Rückgabewert

Name	Typ	Beschreibung
Configure	BOOL	TRUE, wenn die Weighing-Instanz erfolgreich konfiguriert wurde.

5.1.2.2 Call

Die Methode berechnet aus einem Eingangssignal, das in Form eines Pointers übergeben wird, ein manipuliertes Ausgangssignal.

Syntax

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

Eingänge

Name	Typ	Beschreibung
pIn	POINTER TO LREAL	Adresse des Eingangsarrays
nSizeIn	UDINT	Größe des Eingangsarrays
pOut	POINTER TO LREAL	Adresse des Ausgangsarrays
nSizeOut	UDINT	Größe des Ausgangsarrays

Rückgabewert

Name	Typ	Beschreibung
Call	BOOL	Liefert TRUE, wenn ein manipuliertes Ausgangssignal berechnet wurde.

Beispiel

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

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

5.1.2.3 Reset

Die Methode setzt den internen Status der Weighing-Instanz zurück. Durch das Reset des Funktionsbausteins wird die Weighing-Instanz in seinen Ursprungszustand, d. h. ohne Vergangenheitseinfluss, versetzt. Die Weighing-Instanz wird also wieder auf den letzten Konfigurationszustand zurückgesetzt.

Syntax

```
METHOD Reset : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
Reset	BOOL	Liefert TRUE, wenn der interne Status der Weighing-Instanz erfolgreich zurückgesetzt wurde.

5.1.2.4 ApplyCalibration

Die Methode kann zur Laufzeit verwendet werden, um den getriggerten Kalibrierungsprozess (CalibrateRefHigh() [▶ 25]/CalibrateRefLow() [▶ 26]) abzuschließen oder abzubrechen.

Syntax

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

Eingänge

Name	Typ	Beschreibung
bAccept	BOOL	Wenn TRUE, dann werden die CalibrateRefHigh()/ CalibrateRefLow() Ergebnisse übernommen. Ansonsten verworfen.

Beispiel

```
(*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
```

Rückgabewert

Name	Typ	Beschreibung
ApplyCalibration	BOOL	TRUE, wenn die Methode erfolgreich ausgeführt wurde.

5.1.2.5 CalibrateRefHigh

Die Methode kann zur Laufzeit verwendet werden, um die fReferenceHigh Kalibrierung zu triggern.

Syntax

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

Eingänge

Name	Typ	Beschreibung
nDurationInSamples	UDINT	Anzahl der Samples über die gemittelt werden soll.
fRefHigh	LREAL	Neuer fReferenceHigh Wert.

Beispiel

```
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
```

Rückgabewert

Name	Typ	Beschreibung
CalibrateRefHigh	BOOL	TRUE, wenn die Methode erfolgreich ausgeführt wurde.

5.1.2.6 CalibrateRefLow

Die Methode kann zur Laufzeit verwendet werden, um die fReferenceLow Kalibrierung zu triggern.

Syntax

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

Eingänge

Name	Typ	Beschreibung
nDurationInSamples	UDINT	Anzahl der Samples über die gemittelt werden soll.
fRefLow	LREAL	Neuer fReferenceLow Wert.

Beispiel

```
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
```

Rückgabewert

Name	Typ	Beschreibung
CalibrateRefLow	BOOL	TRUE, wenn die Methode erfolgreich ausgeführt wurde.

5.1.2.7 Tare

Die Methode kann zur Laufzeit verwendet werden, um den Funktionsbaustein zu tarieren. Es wird über nDurationInSamples – Ausgangswerten der Mittelwert berechnet. Abschließend wird das Ergebnis der Methode UpdateTareOffset [▶ 27]() übergeben.

Syntax

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

Eingänge

Name	Typ	Beschreibung
nDurationInSamples	UDINT	Anzahl der Samples über die gemittelt werden soll.

Beispiel

```
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
```

Rückgabewert

Name	Typ	Beschreibung
Tare	BOOL	TRUE, wenn die Methode erfolgreich ausgeführt wurde.

5.1.2.8 UpdateTareOffset

Die Methode kann zur Laufzeit verwendet werden, um eine Tarierung manuell durchzuführen. Es bedeutet, dass der fOffset-Wert (Gewicht) aus den berechneten Ausgangswerten herausgerechnet wird. Außerdem werden die Funktionsbaustein – Ausgänge tLastTare und fCurrentTareOffset (= fCurrentTareOffset- fOffset) aktualisiert.

Syntax

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

Eingänge

Name	Typ	Beschreibung
fOffset	LREAL	Das neue Tara Gewicht.

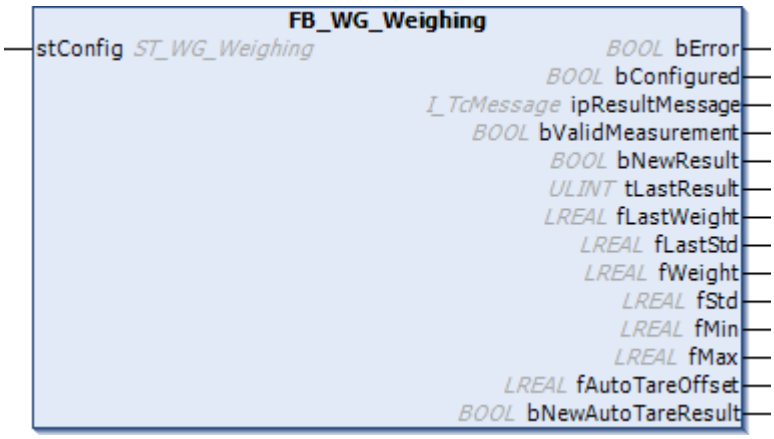
Beispiel

```
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
```

Rückgabewert

Name	Typ	Beschreibung
UpdateTareOffset	BOOL	TRUE, wenn die Methode erfolgreich ausgeführt wurde.

5.1.3 FB_WG_Weighing



Der Funktionsbaustein FB_WG_Weighing dient zur Bestimmung eines Messgewichtes.

Die Konfigurationsstruktur wird mit ST_WG_Weighing [▶ 33] übergeben.

Syntax

Deklaration:

```
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
```

Eingänge

Name	Typ	Beschreibung
stConfig	ST_WG_Weighing [▶ 33]	Bausteinspezifische Konfigurationsstruktur

Ausgänge

Name	Typ	Beschreibung
bError	BOOL	TRUE, wenn ein Fehler auftritt.
bConfigured	BOOL	TRUE, wenn die Konfiguration erfolgreich war.
ipResultMessage	I_TCMessage	Schnittstelle, die Eigenschaften und Methoden zum Messagehandling bereitstellt
bValidMeasurement	BOOL	TRUE, wenn die ST_WG_Weighing_Validation [▶ 34] Bedingungen erfüllt sind.
bNewResult	BOOL	TRUE, wenn ein neues Ergebnis berechnet wurde.
tLastResult	ULINT	Der Zeitstempel des zuletzt berechneten Ergebnisses.
fLastWeight	LREAL	Das Gewicht (gleitender Mittelwert) des letzten Ergebnisses.
fLastStd	LREAL	Die Standardabweichung des letzten Ergebnisses.
fWeight	LREAL	Das aktuelle Gewicht (gleitender Mittelwert) des letzten Ergebnisses.
fStd	LREAL	Die aktuelle gleitende Standardabweichung.
fMin	LREAL	Minimales fWeight-Gewicht im gleitenden Fenster.
fMax	LREAL	Maximales fWeight-Gewicht im gleitenden Fenster.
fAutoTareOffset	LREAL	Das Tara Gewicht des letzten Ergebnisses.
bNewAutoTareResult	BOOL	TRUE, wenn ein neues Tara Gewicht berechnet wurde.

Methoden

Name	Definitionsort	Beschreibung
Configure()	Lokal	Lädt eine neue (oder initiale) Konfigurationsstruktur.
Call()	Lokal	Berechnet das Ausgangssignal bei gegebenem Eingangssignal und Konfiguration des Funktionsbausteins.
Reset()	Lokal	Setzt interne Zustände zurück.

Eigenschaften

Name	Typ	Zugriff	Definitionsort	Initialwert	Beschreibung
bTraceLevelDefault	BOOL	Get, Set	Lokal	TRUE	TRUE, wenn eTraceLevel = Warning.
eTraceLevel	TcEventSeverity	Get, Set	Lokal	Warning	Severity eines Events
nTimeStamp	ULINT	Get, Set	Lokal	0	Zeitstempel des ältesten Eingangswert des nächsten Call()-Aufrufs.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4024.50	PC oder CX (x64, x86)	Tc3_Weighing

Sehen Sie dazu auch

- ST_WG_Weighing_AutoTare [▶ 35]

5.1.3.1 Configure

Die Methode kann zur Laufzeit verwendet werden, um die Weighing-Instanz initial zu konfigurieren (wenn nicht schon in der Deklaration geschehen) oder zu rekonfigurieren.

Wenn eine Weighing-Instanz nicht konfiguriert ist, können die Methoden `Call()` und `Reset()` nicht verwendet werden.

Syntax

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

Eingänge

Name	Typ	Beschreibung
stConfig	ST_WG_Weighing [▶ 33]	Konfigurationsstruktur

Beispiel

```
(*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
```

Rückgabewert

Name	Typ	Beschreibung
Configure	BOOL	TRUE, wenn die Weighing-Instanz erfolgreich konfiguriert wurde.

5.1.3.2 Call

Die Methode berechnet aus einem Eingangssignal, das in Form eines Pointers übergeben wird, ein manipuliertes Ausgangssignal. Falls Oversampling benutzt wird, so werden nicht alle Informationen dargestellt. Die Ergebnisse der Funktionsbaustein-Ausgänge beziehen sich auf den ältesten Eingangswert des Call()-Aufrufs.

Syntax

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

Eingänge

Name	Typ	Beschreibung
pIn	POINTER TO LREAL	Adresse des Eingangsarrays
nSizeIn	UDINT	Größe des Eingangsarrays

Rückgabewert

Name	Typ	Beschreibung
Call	BOOL	Liefert TRUE, wenn ein manipuliertes Ausgangssignal berechnet wurde.

Beispiel

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

5.1.3.3 Reset

Die Methode setzt den internen Status der Weighing-Instanz zurück. Durch das Reset des Funktionsbausteins wird die Weighing-Instanz in seinen Ursprungszustand, d. h. ohne Vergangenheitseinfluss, versetzt. Die Weighing-Instanz wird also wieder auf den letzten Konfigurationszustand zurückgesetzt.

Syntax

```
METHOD Reset : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
Reset	BOOL	Liefert TRUE, wenn der interne Status der Weighing-Instanz erfolgreich zurückgesetzt wurde.

5.1.3.4 AutoTare

Die Methode kann zur Laufzeit verwendet werden, um eine FB_WG_Scaling [▶ 22] -Instanz automatisch zu tarieren.

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
```

Eingänge

Name	Typ	Beschreibung
IScaling	I_WG_Scaling	Funktionsblock mit dem Interface I_WG_Scaling
eAutoTareType	E_WG_AutoTareType [▶ 36]	Verhalten zum automatischen Tarieren

Beispiel

```
// 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)
```

Rückgabewert

Name	Typ	Beschreibung
AutoTare	BOOL	TRUE, wenn der Methodenaufruf erfolgreich durchgeführt wurde.

5.2 Datentypen

5.2.1 Konfigurationsstrukturen

Allgemeine Beschreibung

Für jeden Funktionsbaustein FB_WG_<type> existiert eine individuelle Konfigurationsstruktur ST_WG_<type>. In der Konfigurationsstruktur werden alle Parameter definiert, die zur Berechnung der Übertragungsfunktion, der Ein- und Ausgangsgrößen (Größe und Form der Arrays) sowie der internen Zustände benötigt werden.

5.2.1.1 ST_WG_ComboFilter

Konfigurationsstruktur für den Funktionsbaustein FB_WG_ComboFilter [▶ 19].

```
(* 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` ist die Filterordung (1-10).
- `fCutoff` ist die Grenzfrequenz in Hz (größer 0 und kleiner `fSamplingRate` /2)
- `fSamplingRate` ist die Abtastrate f_s in Hz.
- `nSamplesToFilter` ist die Anzahl der Samples (größer 0) zur Bildung des gleitenden Mittelwerts (oft als Fenstergröße bezeichnet).
- fNotchFrequency ist die Notchfrequenz in Hz (größer 0 und kleiner `fSamplingRate` /2)
- `bReset` ist ein boolescher Parameter, der angibt, ob die internen Vergangenheitswerte bei Neukonfiguration zurückgesetzt werden sollen.

5.2.1.2 ST_WG_Scaling

Konfigurationsstruktur für den Funktionsbaustein FB_WG_Scaling [▶ 22].

```
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
```

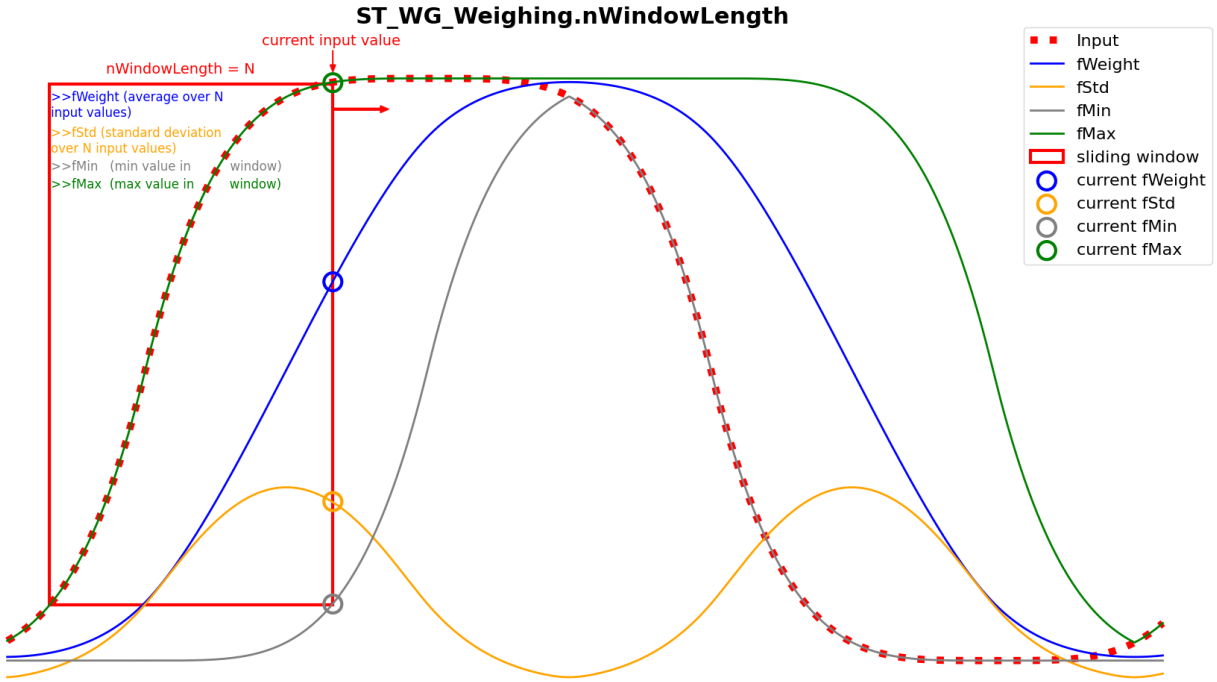
- `fRawLow` muss kleiner als `fRawHigh` sein.
- `fRawHigh` muss größer als `fRawLow` sein.
- `fReferenceLow` muss kleiner als `fReferenceHigh` sein.
- `fReferenceHigh` muss größer als `fReferenceLow` sein.

5.2.1.3 ST_WG_Weighing

Konfigurationsstruktur für den Funktionsbaustein FB_WG_Weighing [▶ 28].

```
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` ist die Anzahl der Samples zur Bildung des gleitenden Mittelwerts (oft als Fenstergröße bezeichnet). Der Parameter gibt an, über wie viele Werte die Funktionsbaustein Ausgänge `fWeight`, `fStd`, `fMin` und `fMax` berechnet werden sollen. Wenn die Anzahl der existierenden Eingangswerte kleiner als `nWindowLength` ist, dann wird die Berechnung über die existierenden Eingangswerte durchgeführt.
- `Validation` ist eine optionale Unterstruktur, welche die Funktionsblock Ausgänge `bValidMeasurment`, `bNewResult`, `tLastResult` und `fLastWeight` beeinflusst.
- `AutoTare` ist eine optionale Unterstruktur, welche die Funktionsblock Ausgänge `fAutoTareOffset` und `bNewAutoTareResult` beeinflusst.

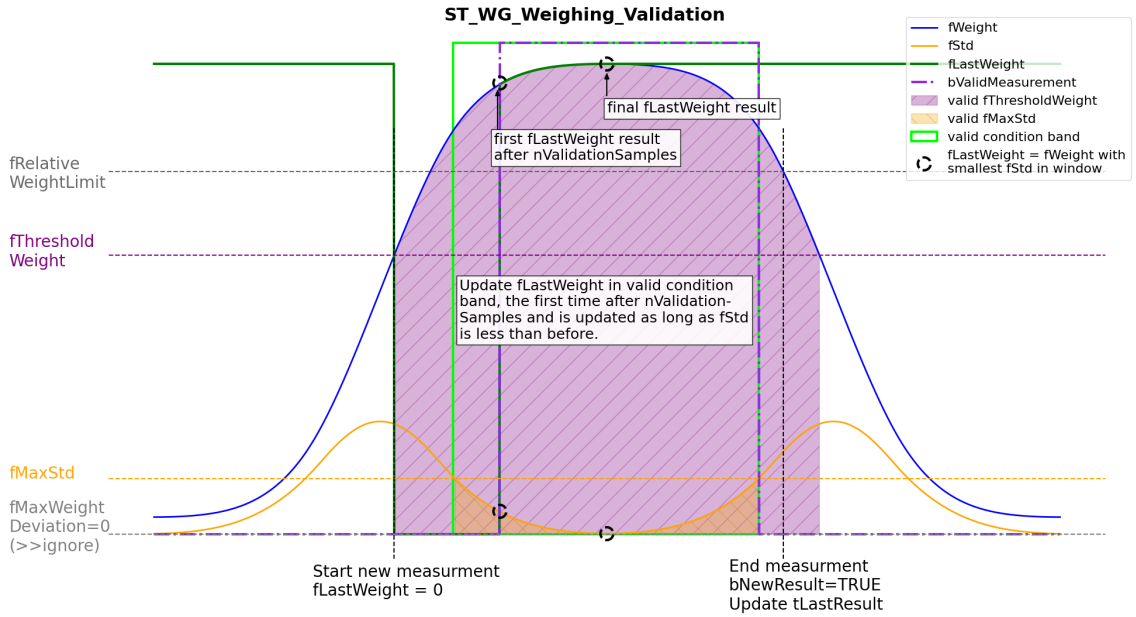


5.2.1.3.1 ST_WG_Weighing_Validation

Unterstruktur für die Konfigurationsstruktur ST_WG_Weighing [▶ 33]. Gesetzte Parameter wirken sich bei den Funktionsbaustein Ausgängen bValidMeasurement, bNewResult, tLastResult, fLastWeight und fLastStd [▶ 29] aus.

```
(*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` ist der Mindestwert für das gemessene Gewicht. Diese Bedingung ist erfüllt, wenn `FB_WG_Weighing.fWeight` größer oder gleich `fThresholdWeight` ist.
- `nValidationSamples` ist die Anzahl der Eingangswerte, für die die anderen `ST_WG_Weighing_Validation` – Parameterbedingungen (`fThresholdWeight, fMaxStd, fMaxWeightDeviation`) erfüllt sein müssen, damit `FB_WG_Weighing.bValidMeasurement=TRUE` ist (optional, empfohlen).
- `fMaxStd` ist die obere Grenze für die Standardabweichung. Diese Bedingung ist erfüllt, wenn `FB_WG_Weighing.fStd` kleiner oder gleich `fMaxStd` ist (optional, empfohlen).
- `fMaxWeightDeviation` ist die obere Grenze für die maximale Gewichtsabweichung. Diese Bedingung ist erfüllt, wenn `FB_WG_Weighing.fMax - FB_WG_Weighing.fMin` kleiner oder gleich `fMaxWeightDeviation` ist (optional, empfohlen).
- `fRelativeWeightLimit` (> 0 und < 1) gibt an, dass `FB_WG_Weighing.bNewResult` und `FB_WG_Weighing.tLastResult` aktualisiert werden, wenn `FB_WG_Weighing.fWeight` unter dem Grenzwert `fRelativeWeightLimit·FB_WG_Weighing.fLastWeight` fällt (optional).



Steigt `FB_WG_Weighing.fWeight` über `fThresholdWeight`, wird so lange nach dem Gewicht mit dem kleinsten `FB_WG_Weighing.fStd` gesucht, bis `FB_WG_Weighing.fWeight` die `fThresholdWeight` Grenze wieder verlässt. Ist `fRelativeWeightLimit` gesetzt, dann endet die Messung, wenn `FB_WG_Weighing.fWeight` unter `fThresholdWeight*fRelativeWeightLimit` fällt. Das ermittelte Gewicht wird das erste Mal bei einer steigenden Flanke von `FB_WG_Weighing.bValidMeasurement` in `FB_WG_Weighing.fLastWeight` angezeigt und so lange aktualisiert, bis die Messung beendet ist. Beim Ende der Messung wird der Zeitstempel in `FB_WG_Weighing.tLastResult` gesetzt und für einen Zyklus `FB_WG_Weighing.bNewResult` auf TRUE gesetzt. Steigt `FB_WG_Weighing.fWeight` wieder über `fThresholdWeight`, dann wird `FB_WG_Weighing.fLastWeight` auf null gesetzt und es beginnt eine neue Messung.

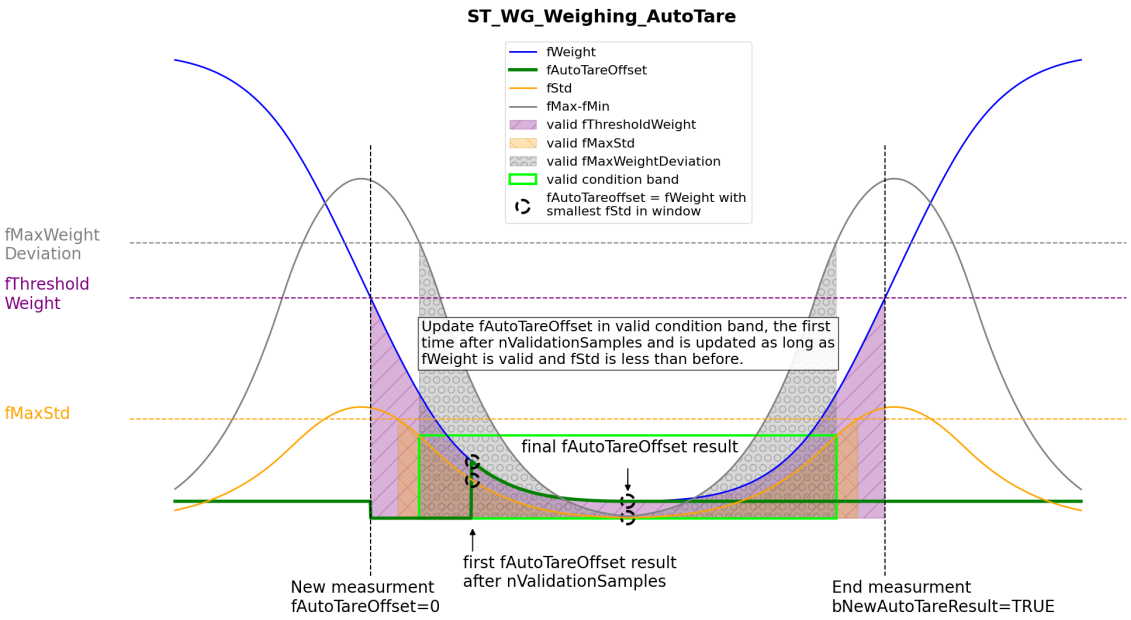
5.2.1.3.2 ST_WG_Weighing_AutoTare

Unterstruktur für die Konfigurationsstruktur ST_WG_Weighing [▶ 33]. Wenn Parameter gesetzt werden, wirkt sich das ausschließlich in FB_WG_Weighing.fAutoTareOffset und FB_WG_Weighing.bNewAutoTareResult aus. Eine Instanz fbScaling von FB_WG_Scaling [▶ 22] lässt sich automatisch tarieren, indem z. B. AutoTare [▶ 31](fbScaling, E_WG_AutoTareType [▶ 36].eEnd) aufgerufen wird.

```
(* 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` ist der Maximalwert für das gemessene Gewicht. Diese Bedingung ist erfüllt, wenn `FB_WG_Weighing.fWeight` kleiner oder gleich `fThresholdWeight` ist .

- `nValidationSamples` ist die Anzahl der Eingangswerte, für die die anderen `ST_WG_Weighing_AutoTare` – Parameterbedingungen (`fThresholdWeight, fMaxStd, fMaxWeightDeviation`) erfüllt sein müssen, damit `FB_WG_Weighing.fAutoTareOffset` gesetzt wird (optional, empfohlen).
- `fMaxStd` ist die obere Grenze für die Standardabweichung. Diese Bedingung ist erfüllt, wenn `FB_WG_Weighing.fStd` kleiner oder gleich `fMaxStd` ist (optional, empfohlen).
- `fMaxWeightDeviation` ist die obere Grenze für die maximale Gewichtsabweichung. Diese Bedingung ist erfüllt, wenn `FB_WG_Weighing.fMax - FB_WG_Weighing.fMin` kleiner oder gleich `fMaxWeightDeviation` ist (optional, empfohlen).



Fällt `FB_WG_Weighing.fWeight` unter `fThresholdWeight`, wird so lange nach dem Tara Gewicht mit dem kleinsten `FB_WG_Weighing.fStd` gesucht, bis `FB_WG_Weighing.fWeight` die `fThresholdWeight` Grenze wieder verlässt. Das Tara Gewicht wird frühestens nach `nValidationSamples` – Werten in `FB_WG_Weighing.fWeight` dargestellt.

5.2.2 E_WG_Calibrate

ENUM für den Kalibrierungsausgang von FB_WG_Scaling [▶ 22] .

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
```

5.2.3 E_WG_AutoTareType

ENUM für den Methodeneingang von FB_WG_Weighing.AutoTare() [▶ 31].

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
```

6 Beispiele

6.1 Dynamisches Wiegen

In diesem Beispiel wird gezeigt, wie der Prozess des dynamischen Wiegens mit der Weighing SPS-Bibliothek funktioniert.

Download: https://infosys.beckhoff.com/content/1031/TF3685_TC3_Weighing_Library/Resources/16127833867.zip (*.tnzip)

Beschreibung

Das Eingangssignal, ein verrauschtes Trapezsignal, wird im MAIN-Programm über die Methode `GenerateInputs()` mit einem Signalgenerator erzeugt. Das simulierte Signal wird dem Funktionsbaustein FB_DynamicWeighing (`fbDynamicWeighing`) übergeben, der es intern an weitere Funktionsbausteine weiterleitet. Dazu zählen die Filterung mit dem Baustein FB_WG_ComboFilter [▶ 19] (`fbComboFilter`), die Skalierung mit dem Baustein FB_WG_Scaling [▶ 22] (`fbScale`) sowie die Auswertung mit dem Baustein FB_WG_Weighing [▶ 28] (`fbWeighing`).

Programmparameter

Die Tabelle unten zeigt eine Liste mit wichtigen Parametern für die Konfiguration der verwendeten Funktionsbausteine.

Variable	Beschreibung	Standardwert
fRawAmplitudeSignal	Amplitude des Trapezsignals	1000.0
fAbsoluteNoise	Absoluter Rauschbetrag	200.0
fFrequency	Grundfrequenz des Trapezsignals	1.0 Hz
bActivateSlope	Ein driftendes Testsignal wird aktiviert/deaktiviert	FALSE
bAddWeight	Addiert einmalig ein Offset-Wert zum Testsignal	FALSE
eAutoTareType	Auswahl für das automatische Tarieren	E_WG_AutoTareType.eContinously

Folgende globale Konstanten werden definiert.

Variable	Beschreibung	Standardwert
cOversamples	Anzahl von Oversamples des Eingangskanals	10
cSamplingRate	Samplerate des Eingangskanals in Hz	1000

Implementierung:

Zunächst werden die entsprechenden Strukturen und Funktionsbausteine deklariert und initialisiert:

```
// 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);
```

Im Implementierungsteil werden die Funktionsbausteininstanzen über die entsprechenden `Call()` Methoden ausgeführt.

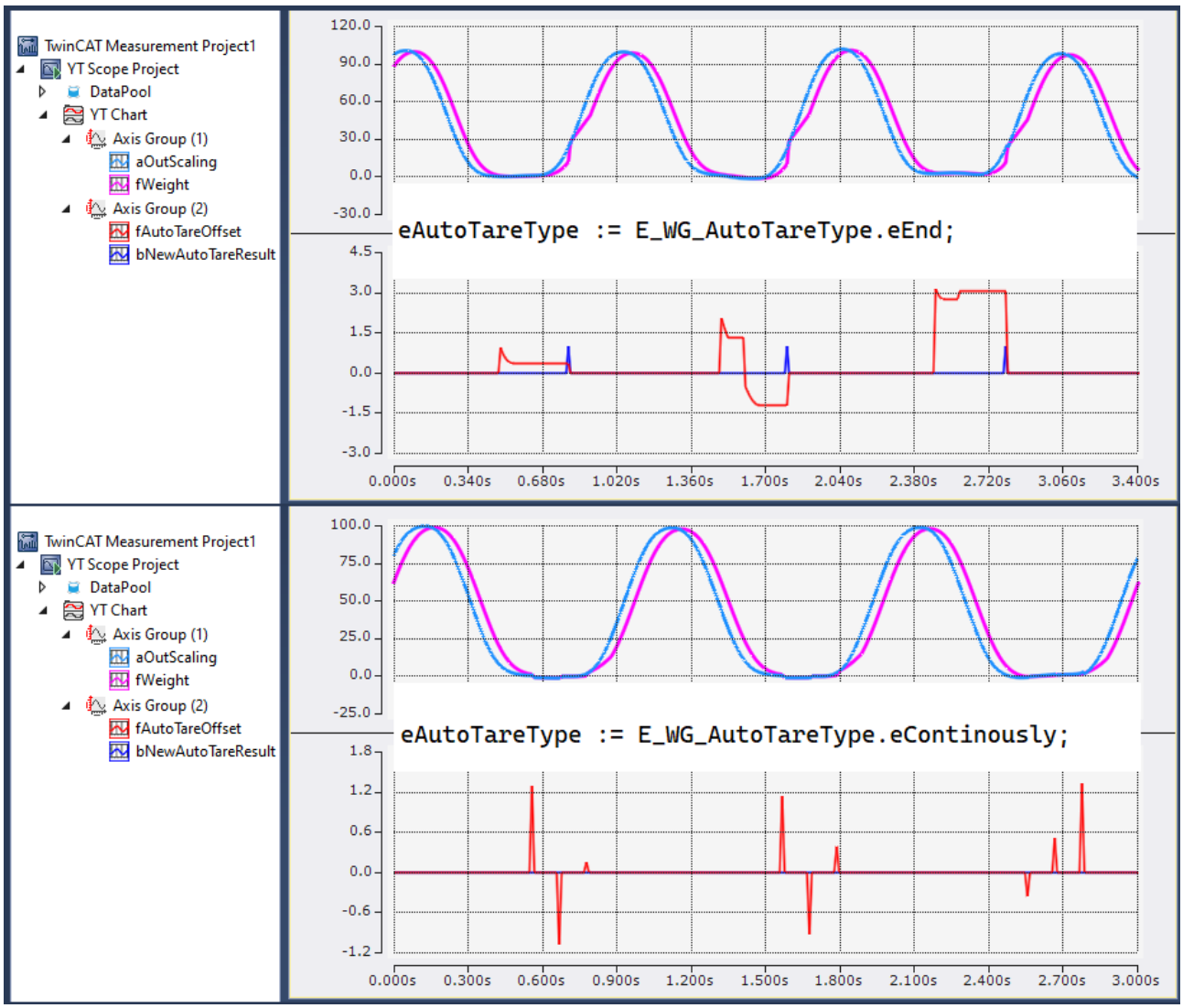
```
// Execute weighing
IF NOT fbComboFilter.Call(ADR(aInput), SIZEOF(aInput), ADR(aOutFilter), SIZEOF(aOutFilter)) THEN
    SetError(fbComboFilter);
END_IF

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
```

Automatisches Tarieren:

Die Instanz `fbScale` lässt sich über die `fbWeighing`-Instanz wie folgt automatisch tarieren:

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

In Abhängigkeit vom eAutoTareType-Initialisierungswert gibt es folgende Fallunterscheidung:



`E_WG_AutoTareType.eEnd` (oben): `fbScale` wird mit dem Wert `fAutoTareOffset` tariert, wenn `bNewAutoTareResult` gleich `TRUE` ist. `E_WG_AutoTareType.eContinously` (unten): `fbScale` wird mit dem Wert `fAutoTareOffset` tariert, wenn `fAutoTareOffset` ungleich 0 ist. Nach dem Tarieren wird jedes Mal automatisch ein `fbWeighing.Reset()` ausgeführt.

7 Anhang

7.1 Rückgabecodes

Rückgabecodes des 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.'

Definierte 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	Samplingrate 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	Samplingrate 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.

7.2 FAQ - Häufig gestellte Fragen und Antworten

In diesem Bereich werden häufig gestellte Fragen beantwortet, um Ihnen die Arbeit mit der TwinCAT 3 Weighing Bibliothek zu erleichtern.
Wenn Sie weitere Fragen haben, kontaktieren Sie unseren Support (-157).

1. Kann ich für komplexere Applikationen die Filter des FB_WG_ComboFilter erweitern? [▶ 43]

Kann ich für komplexere Applikationen die Filter des FB_WG_ComboFilter erweitern?

Der FB_WG_ComboFilter enthält bereits drei unterschiedliche Filtertypen, die Sie je nach Anwendung hinzuschalten können. Sollte das immer noch nicht ausreichen, ist es möglich weitere Filter durch die TF3680 TwinCAT 3 Filter Bibliothek hinzuzuschalten. Für diesen Anwendungsfall ist die Lizenz der TF3680 bereits in der Lizenz für die TF3685 TwinCAT 3 Weighing enthalten. Eine zusätzliche Lizenz ist somit nicht erforderlich.

7.3 Support und Service

Beckhoff und seine weltweiten Partnerfirmen bieten einen umfassenden Support und Service, der eine schnelle und kompetente Unterstützung bei allen Fragen zu Beckhoff Produkten und Systemlösungen zur Verfügung stellt.

Downloadfinder

Unser Downloadfinder beinhaltet alle Dateien, die wir Ihnen zum Herunterladen anbieten. Sie finden dort Applikationsberichte, technische Dokumentationen, technische Zeichnungen, Konfigurationsdateien und vieles mehr.

Die Downloads sind in verschiedenen Formaten erhältlich.

Beckhoff Niederlassungen und Vertretungen

Wenden Sie sich bitte an Ihre Beckhoff Niederlassung oder Ihre Vertretung für den lokalen Support und Service zu Beckhoff Produkten!

Die Adressen der weltweiten Beckhoff Niederlassungen und Vertretungen entnehmen Sie bitte unserer Internetseite: www.beckhoff.com

Dort finden Sie auch weitere Dokumentationen zu Beckhoff Komponenten.

Beckhoff Support

Der Support bietet Ihnen einen umfangreichen technischen Support, der Sie nicht nur bei dem Einsatz einzelner Beckhoff Produkte, sondern auch bei weiteren umfassenden Dienstleistungen unterstützt:

- Support
- Planung, Programmierung und Inbetriebnahme komplexer Automatisierungssysteme
- umfangreiches Schulungsprogramm für Beckhoff Systemkomponenten

Hotline:	+49 5246 963-157
E-Mail:	support@beckhoff.com

Beckhoff Service

Das Beckhoff Service-Center unterstützt Sie rund um den After-Sales-Service:

- Vor-Ort-Service
- Reparaturservice
- Ersatzteilservice
- Hotline-Service

Hotline:	+49 5246 963-460
E-Mail:	service@beckhoff.com

Beckhoff Unternehmenszentrale

Beckhoff Automation GmbH & Co. KG

Hülshorstweg 20
33415 Verl
Deutschland

Telefon:	+49 5246 963-0
E-Mail:	info@beckhoff.com
Internet:	www.beckhoff.com

Mehr Informationen:
www.beckhoff.com/tf3685

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