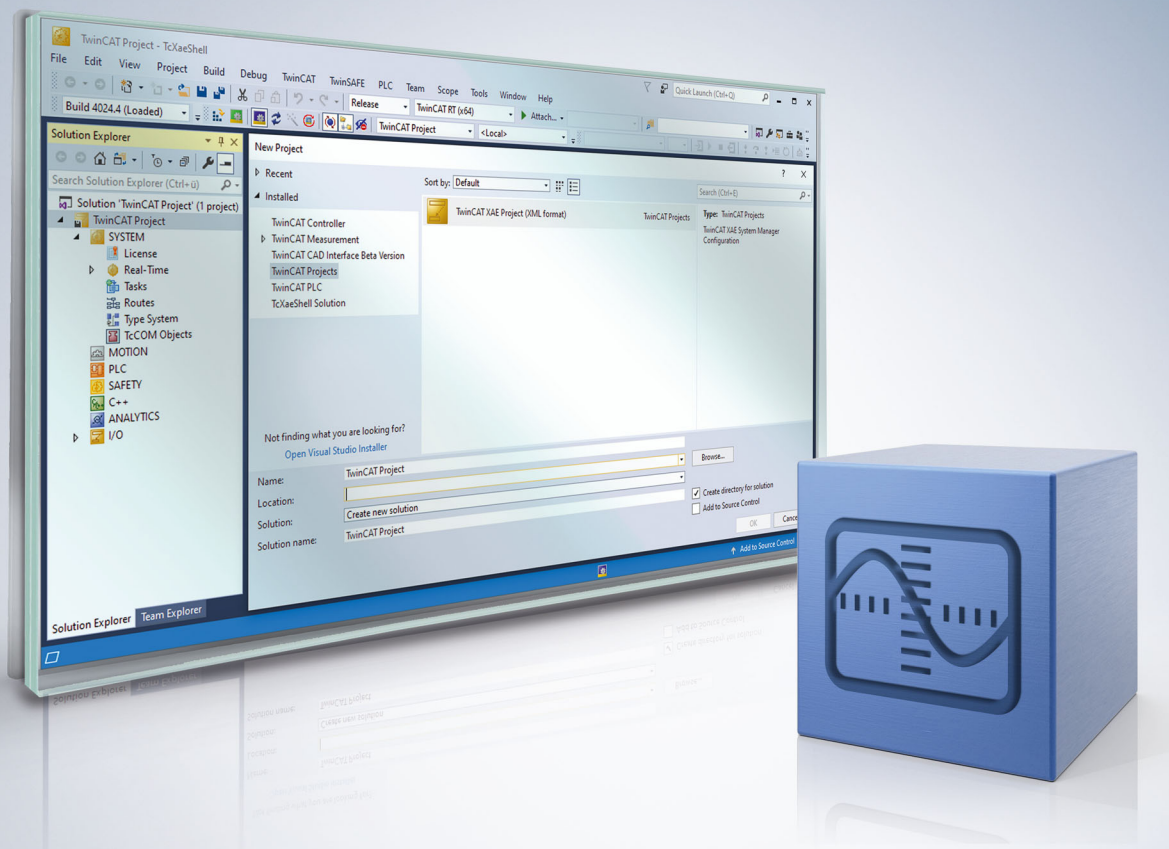
BECKHOFF New Automation Technology

Handbuch | DE

TF3710

TwinCAT 3 | Interface for LabVIEW™



15.04.2024 | Version: 1.5.2

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.

1.4 Ausgabestände der Dokumentation

Version	Änderungen
1.5.x	TwinCAT 3.1 Build 4026 • Kurz-Information zur Installation [▶ 11]
	Neu: • CoE Read und CoE Write [▶ 82] ◦ VIs zum Lesen und Schreiben von CoE-Objekten aus LabVIEW™ • Beispiel CoE Lesen oder Schreiben [▶ 112]

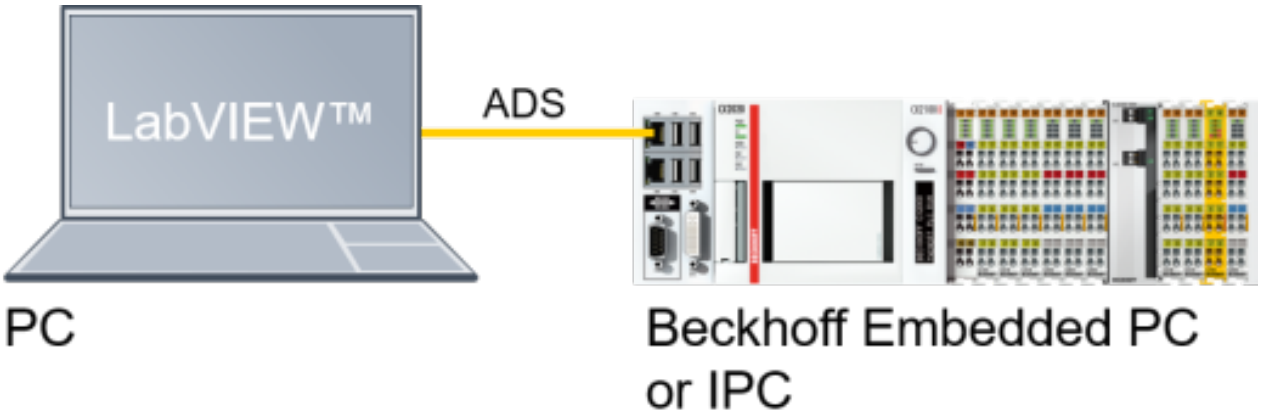
Inhaltsverzeichnis

2 Übersicht

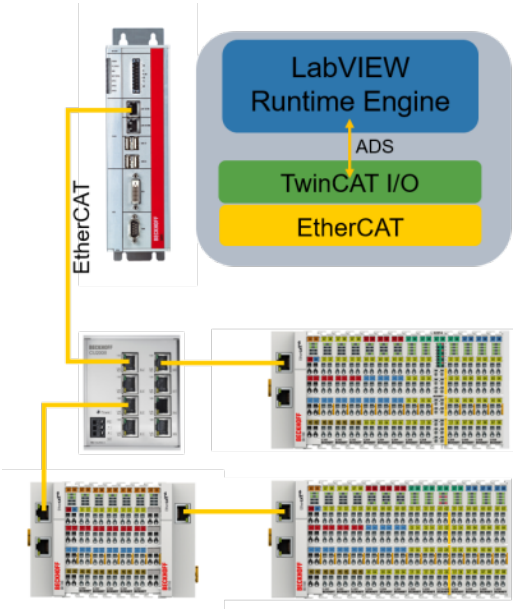
Das TwinCAT 3 Interface for LabVIEW™ ermöglicht den Datenaustausch zwischen LabVIEW™ und einer oder mehreren TwinCAT-Laufzeitumgebungen. Der Datenaustausch findet über das Protokoll *TwinCAT ADS* statt.

Produktinformationen

Mit dem Produkt TF3710 TwinCAT 3 Interface for LabVIEW™ vereinen Sie die Welten von NI™ und Beckhoff. Durch einen performanten Datenaustausch zwischen einer TwinCAT-Laufzeitumgebung und Ihrer LabVIEW™-Applikation erhalten Sie die Möglichkeit, einen Teil Ihrer Anwendung in TwinCAT und einen anderen Teil in LabVIEW™ zu realisieren. Sie entscheiden über die Verteilung. In einer Minimalkonfiguration können Sie TwinCAT auch ausschließlich als Treiber Ihrer Feldbusteilnehmer nutzen und Ihre Anwendung vollständig in LabVIEW™ realisieren. Ebenso können Sie Ihre Anwendung hauptsächlich in TwinCAT realisieren und LabVIEW™ nur als Human Machine Interface (HMI) nutzen.



Dabei unterstützt das TwinCAT 3 Interface for LabVIEW™ alle LabVIEW™-Editionen, wie *Base*, *Full*, *Professional* und auch die *Runtime Engine*, sodass Sie Ihre LabVIEW™-Applikation mit dem LabVIEW™ Applicationbuilder auch als Stand-alone-Anwendung ausliefern können. Die Windows-Betriebssysteme auf Beckhoff IPCs und Embedded PCs eignen sich dabei hervorragend als Plattform direkt auf der Maschinensteuerung.



Produktkomponenten

Das Produkt TF3710 TwinCAT 3 Interface for LabVIEW™ besteht aus den folgenden Komponenten:

- LabVIEW™ Virtuelle Instrumente (VIs) für Ihre Function-Palette,

- Produktspezifische Dynamic Link Libraries (DLLs),
- Beispiel-VIs.

Bei Installation auf einem System mit LabVIEW™-Runtime werden nur die DLLs installiert; sonst werden alle Komponenten installiert.

3 Installation

Systemvoraussetzungen

Die Mindestanforderungen zur Nutzung des Produkts TF3710 TwinCAT 3 Interface for LabVIEW™ werden im Folgenden beschrieben.

Beachten Sie bitte folgende Unterscheidung: Die Lizenz TF3710 wird auf der TwinCAT-Laufzeitumgebung benötigt. Die Komponenten des TF3710 TwinCAT 3 Interface for LabVIEW™ hingegen, werden auf dem System mit der LabVIEW™-Installation benötigt.

Engineering

Auf Ihrer Engineering-Umgebung benötigen Sie neben LabVIEW™ eine TwinCAT 3.1 ADS-Installation. Wenn Sie eine TwinCAT 3-Engineering-Umgebung oder eine TwinCAT-Laufzeit auf Ihrem Engineering-System installiert haben, muss TwinCAT 3.1 ADS nicht installiert werden.

- TwinCAT 3.1 ADS
- LabVIEW™ 2017, 2018, 2019, 2020, 2021, 2022, 2023
  - Runtime, Base, Full, Community, Professional Edition
  - 32 bit, 64 bit

Runtime

Seitens der TwinCAT-Laufzeit benötigen Sie folgende Komponenten:

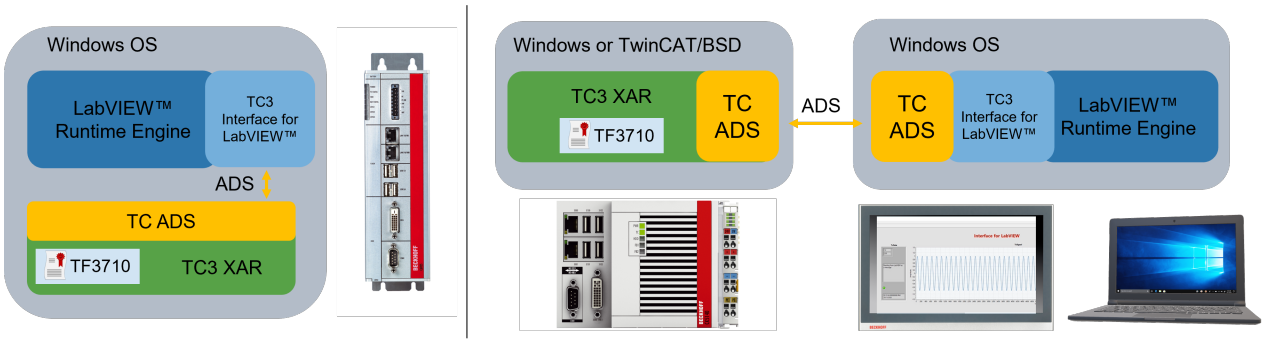
- TwinCAT 3.1 XAR build 3.1.4024.12 oder höher
  Wenn eine niedrigere Version erforderlich sein sollte, melden Sie sich bitte beim Beckhoff Support.
- Betriebssysteme: Windows Betriebssystem, TwinCAT/BSD
- Eine Lizenz für TF3710 TwinCAT 3 Interface for LabVIEW™

Für Testzwecke kann eine 7-Tage-Testversion wiederholt aktiviert werden.

Seitens der LabVIEW™-Laufzeit ist erforderlich:

- LabVIEW™ Runtime Engine 2017, 2018, 2019, 2020, 2021, 2022, 2023
- Windows NT basiertes Betriebssystem

Dabei ist zu beachten, dass die TwinCAT- und LabVIEW™-Laufzeit sowohl auf dem gleichen als auch auf separaten über Netzwerk verbundenen Systemen installiert werden können.



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:

TF3710 | TwinCAT 3 Interface for LabVIEW™

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

Nachfolgend wird die Installation der TwinCAT 3 Function TwinCAT 3 Interface for LabVIEW™ für Windows-basierte Betriebssysteme beschrieben.

HINWEIS

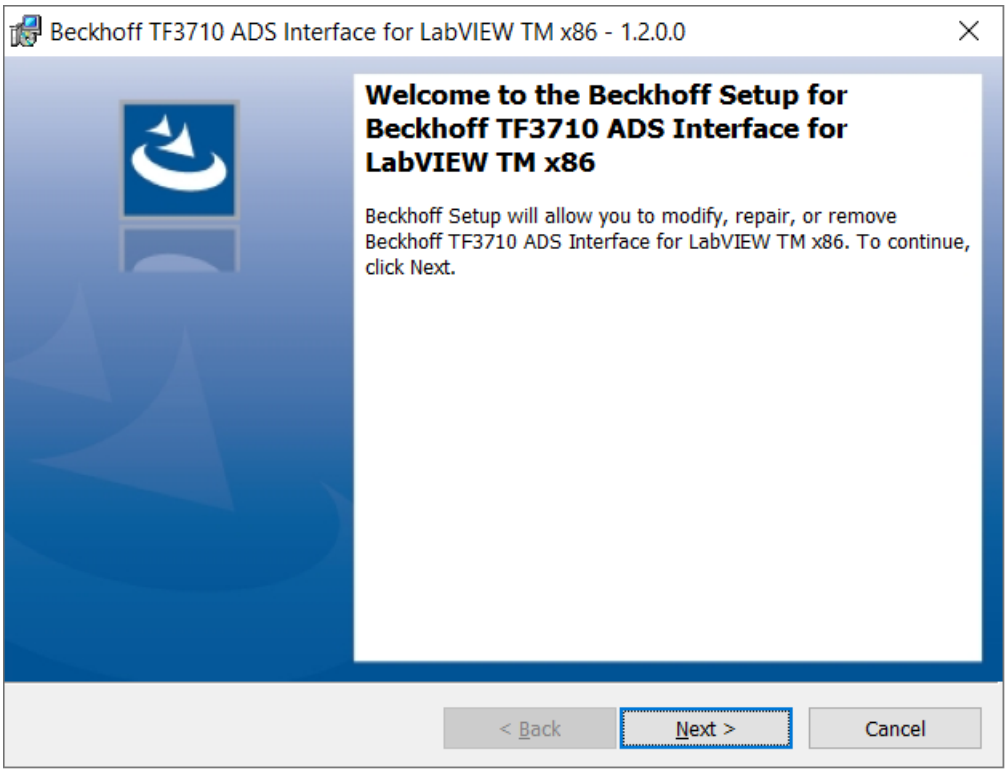
Unterschiedliche Setups für 32bit und 64bit LabVIEW™

Auf der Download-Seite der Beckhoff-Homepage finden Sie zwei Setups. Das Setup mit der Bezeichnung „x86“ integriert das Interface for LabVIEW™ in installierte 32bit LabVIEW™-Umgebungen. Das Setup mit der Bezeichnung „x64“ ist entsprechend für 64bit LabVIEW™-Umgebungen.

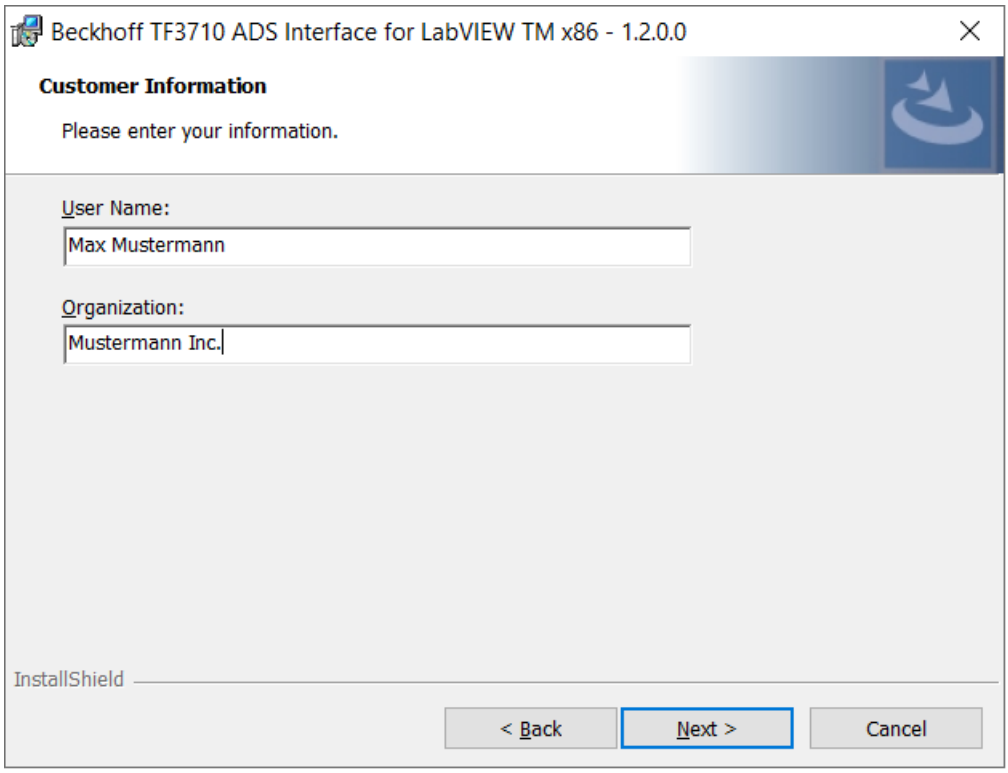
✓ Die Setup-Datei der TwinCAT 3 Function wurde von der Beckhoff-Homepage heruntergeladen.

1. Führen Sie die Setup-Datei als Administrator aus. Wählen Sie dazu im Kontextmenü der Datei den Befehl **Als Administrator ausführen**.

⇨ Der Installationsdialog öffnet sich.

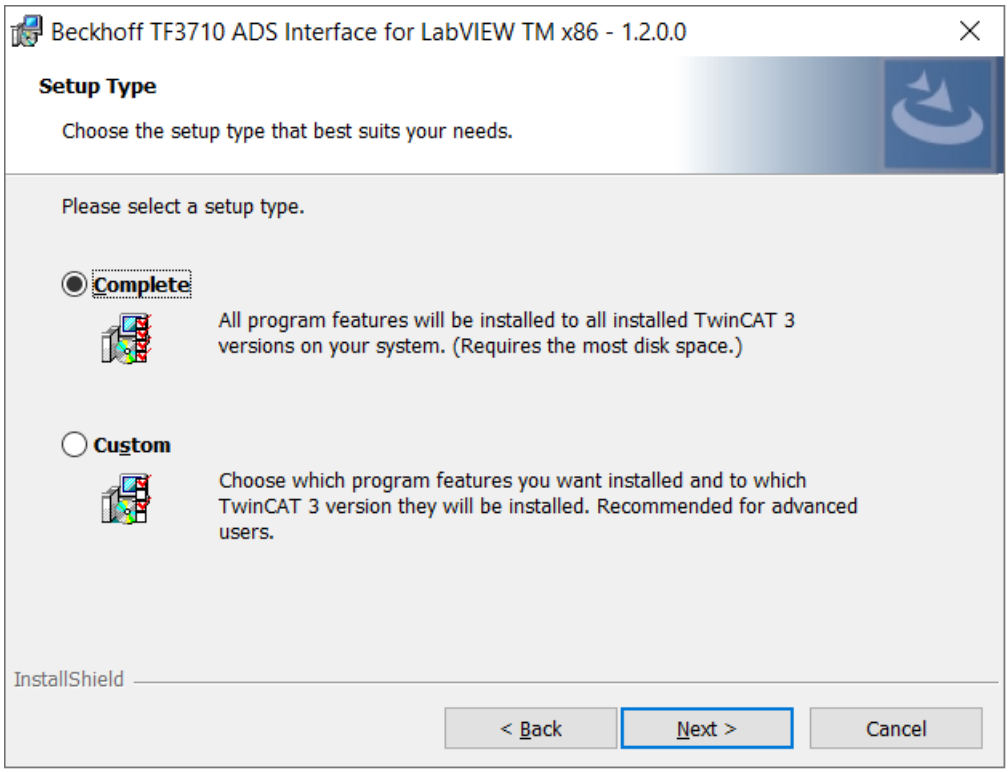


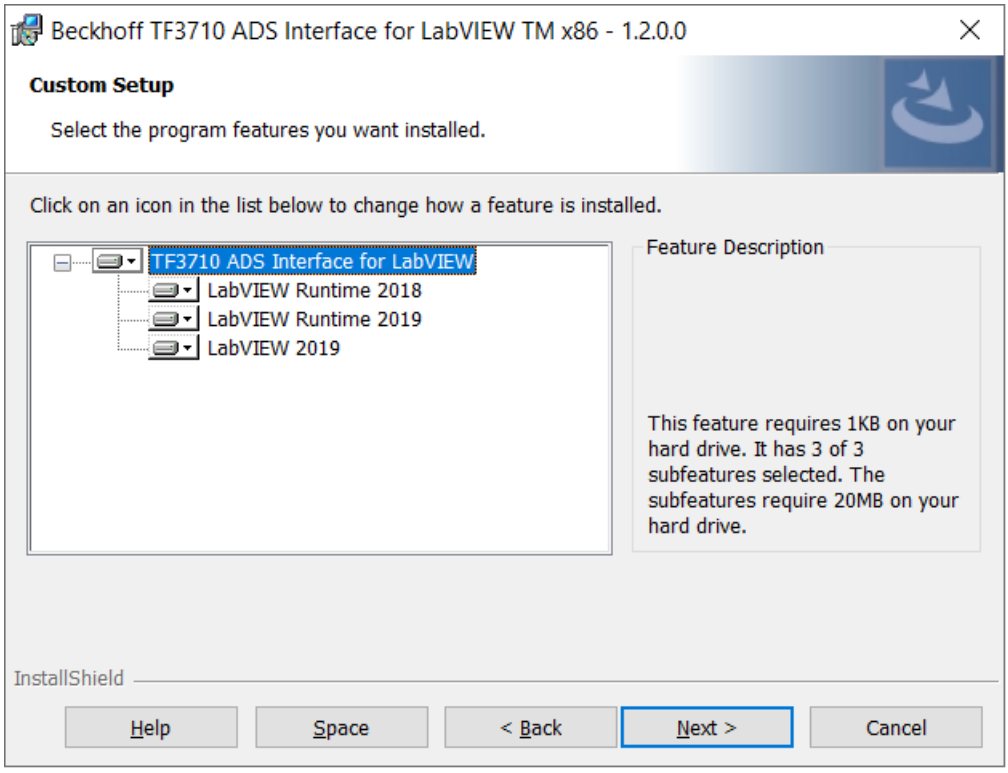
2. Akzeptieren Sie die Endbenutzerbedingungen und klicken Sie auf **Next**.
3. Geben Sie Ihre Benutzerdaten ein.



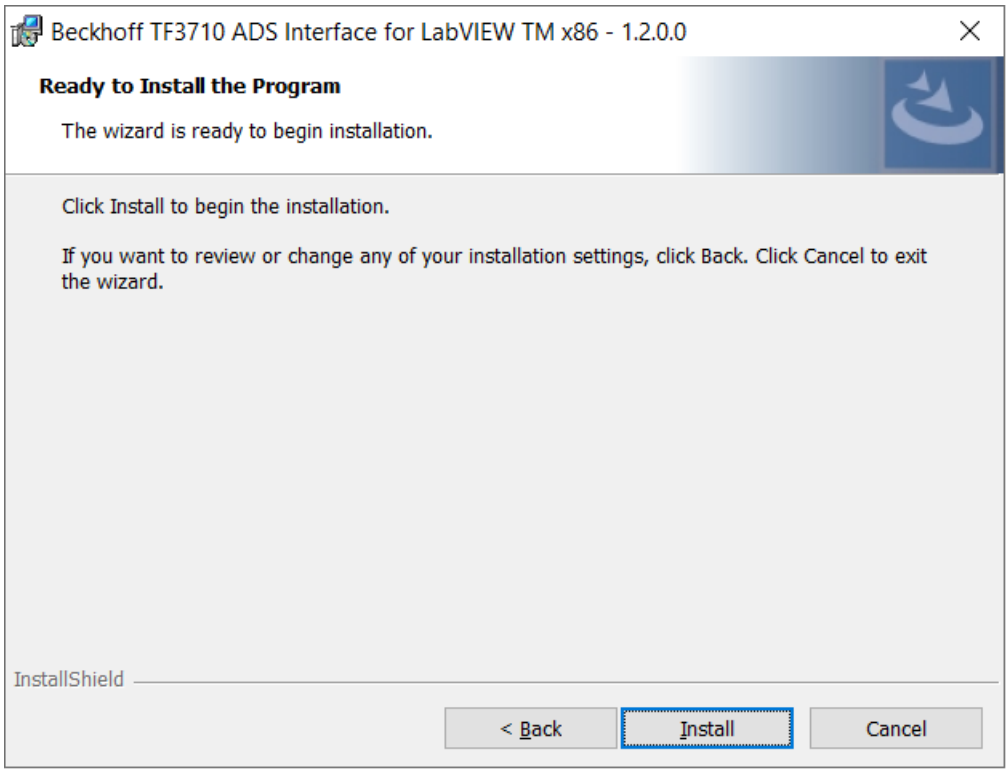
4. Wenn Sie die TwinCAT 3 Function vollständig installieren möchten, wählen Sie **Complete** als Installationstyp. Dabei wird für jedes auf dem System gefundene LabVIEW™ das Produkt installiert. Wenn Sie das Produkt nur für einzelne LabVIEW™-Umgebungen installieren möchten, wählen Sie

Custom.


Beckhoff TF3710 ADS Interface for LabVIEW TM x86 - 1.2.0.0
Setup Type
Choose the setup type that best suits your needs.
Please select a setup type.
Complete
All program features will be installed to all installed TwinCAT 3 versions on your system. (Requires the most disk space.)
Custom
Choose which program features you want installed and to which TwinCAT 3 version they will be installed. Recommended for advanced users.
InstallShield
< Back
Next >
Cancel


Beckhoff TF3710 ADS Interface for LabVIEW TM x86 - 1.2.0.0
Custom Setup
Select the program features you want installed.
Click on an icon in the list below to change how a feature is installed.
TF3710 ADS Interface for LabVIEW
LabVIEW Runtime 2018
LabVIEW Runtime 2019
LabVIEW 2019
Feature Description
This feature requires 1KB on your hard drive. It has 3 of 3 subfeatures selected. The subfeatures require 20MB on your hard drive.
InstallShield
Help
Space
< Back
Next >
Cancel

5. Wählen Sie **Next** und anschließend **Install**, um die Installation zu beginnen.



⇨ Ein Dialog weist Sie ggf. darauf hin, dass das TwinCAT-System für die weitere Installation gestoppt werden muss.

Mass Compile

Bei der Installation wird die Bibliothek des TwinCAT 3 Interfaces for LabVIEW™ zur Massenkompilierung von VIs hinzugefügt. Dazu wird ein LabVIEW™ VI gestartet.

6. Bestätigen Sie den Dialog mit **Yes.**

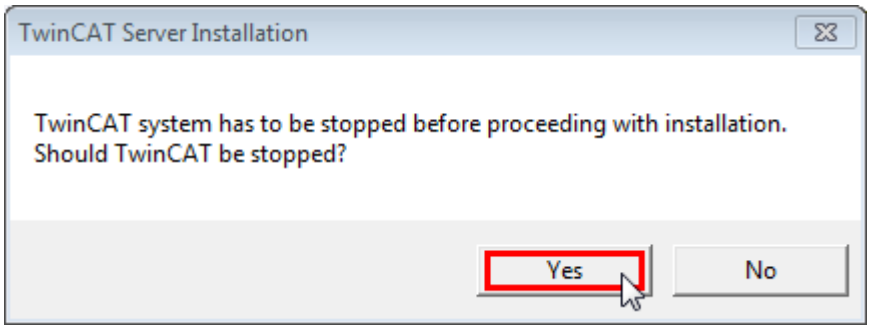


Abb. 1:

7. Wählen Sie **Finish**, um das Setup zu beenden.

⇨ Die TwinCAT 3 Function wurde erfolgreich installiert und kann lizenziert werden (siehe Lizensierung).

4 Lizenzierung

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

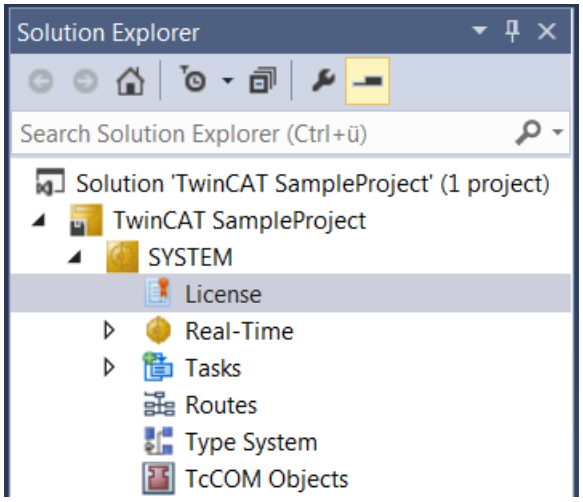
Lizenzierung der Vollversion einer TwinCAT 3 Function

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

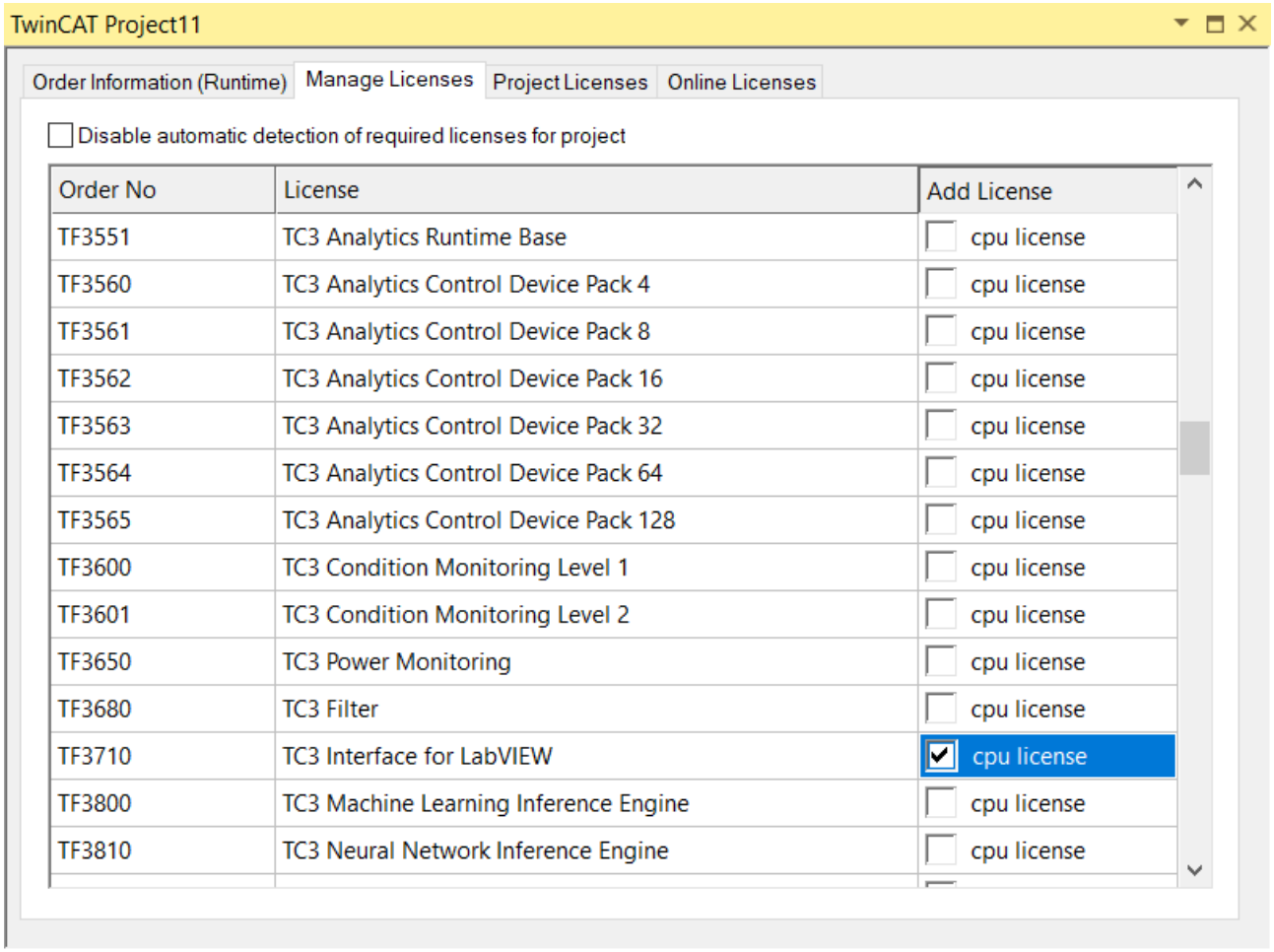
Lizenzierung der 7-Tage-Testversion einer TwinCAT 3 Function

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

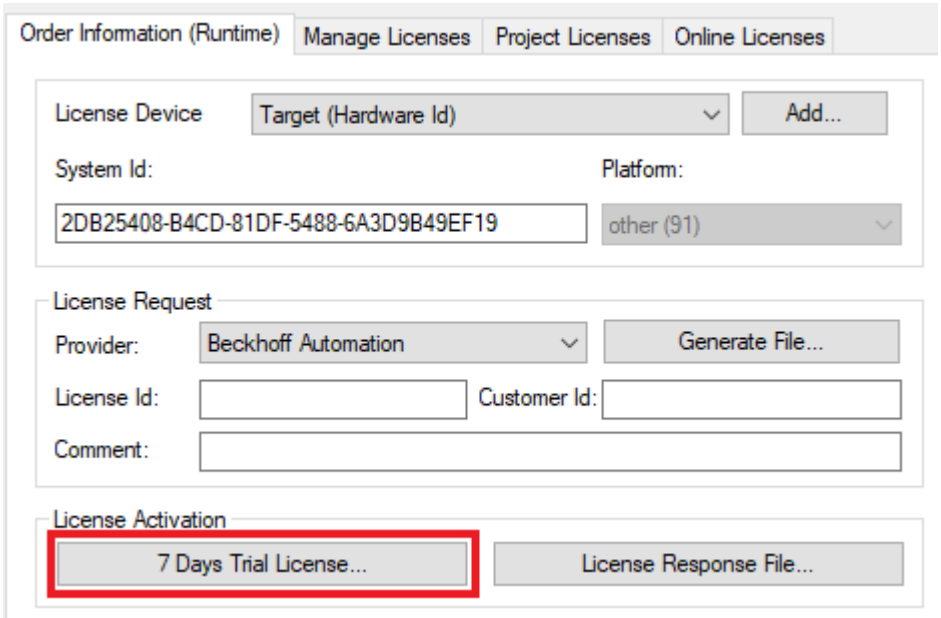
1. Starten Sie die TwinCAT-3-Entwicklungsumgebung (XAE).
2. Öffnen Sie ein bestehendes TwinCAT-3-Projekt oder legen Sie ein neues Projekt an.
3. Wenn Sie die Lizenz für ein Remote-Gerät aktivieren wollen, stellen Sie das gewünschte Zielsystem ein. Wählen Sie dazu in der Symbolleiste in der Drop-down-Liste **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.**



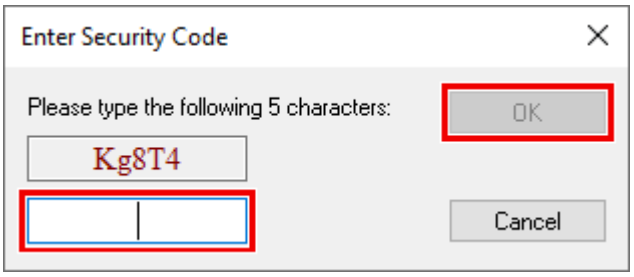
   ⇨ Der TwinCAT-3-Lizenzmanager öffnet sich.
5. Öffnen Sie die Registerkarte **Manage Licenses.** Aktivieren Sie in der Spalte **Add License** das Auswahlkästchen für die Lizenz, die Sie Ihrem Projekt hinzufügen möchten (z. B. „TF3710 TwinCAT 3 Interface for LabVIEW“).



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.

Quick-Guide: Lizenzabfrage schlägt fehl

Es kann zu unterschiedlichen Fehlermeldungen bei der Lizenzabfrage in LabVIEW™ kommen, z.B.: *License state issue*, *No License found, License expired, License invalid, License invalid System id*.

Überprüfen Sie Folgendes:

- Wenn eine Lizenzabfrage zuvor fehlgeschlagen ist, starten Sie LabVIEW™ neu, damit der ADS-Client im Interface for LabVIEW™ komplett entladen wird.
- Nutzen Sie das TwinCAT XAE, um den Lizenzstatus auf dem Zielsystem zu prüfen und ggf. anzupassen.

5 Quickstart

Im Folgenden wird anhand einer exemplarischen Messaufgabe beschrieben, wie Sie die Verbindung zwischen LabVIEW™ und TwinCAT einrichten. Dabei werden in TwinCAT Daten erzeugt, die von LabVIEW™ gelesen werden.

TwinCAT-Projekt aktivieren

✓ Laden Sie dieses https://infosys.beckhoff.com/content/1031/TF3710_TC3_Interface_for_LabVIEW/Resources/10083995531.zip herunter. Im ZIP-Archiv befindet sich eine tnzip-Datei.

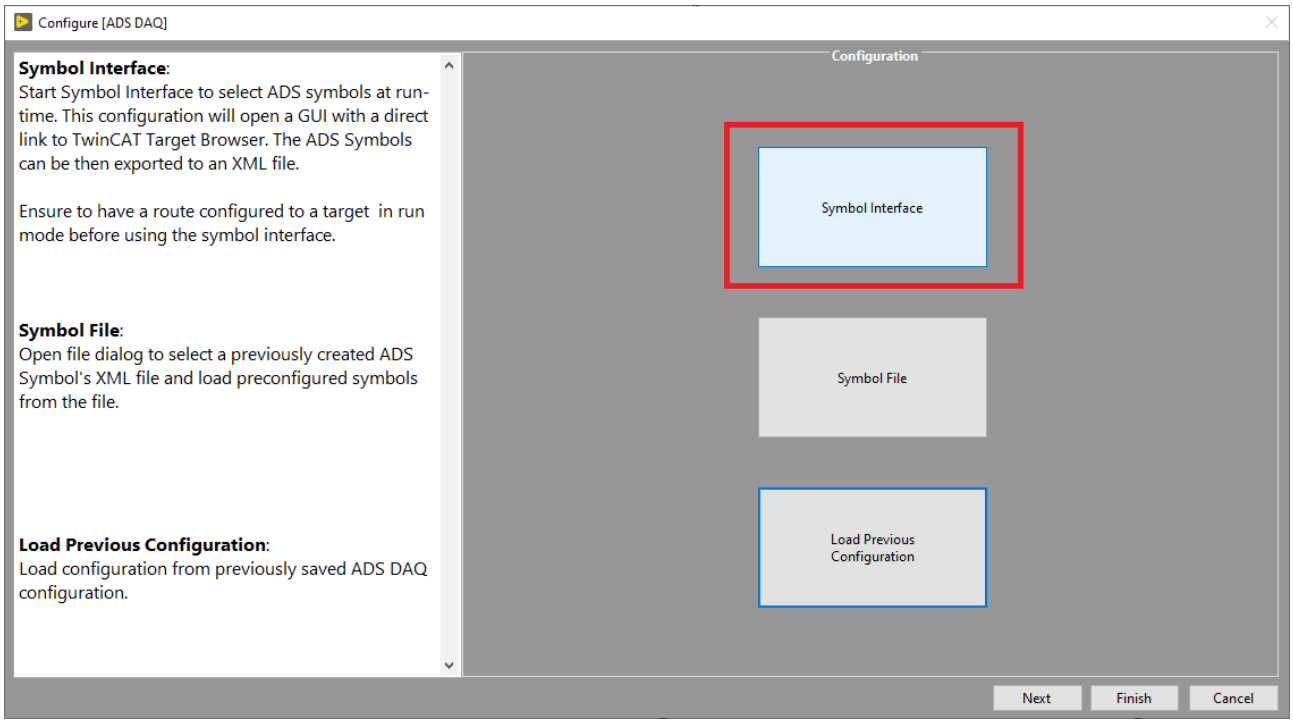
1. Öffnen Sie TwinCAT XAE und wählen Sie **File > Open > Open Solution From Archive…** um das tnzip zu laden.
2. Falls Sie auf dem Zielsystem noch nicht über eine gültige TF3710-Lizenz verfügen, aktivieren Sie unter **System > License > Manage Licenses** die Checkbox für die TF3710-Lizenz.
3. Aktivieren Sie das Projekt, z. B. auf Ihrem lokalen PC oder auch einem remote Zielsystem und starten Sie die SPS.

⇨ Das TwinCAT-Projekt läuft nun auf Ihrem Zielsystem.

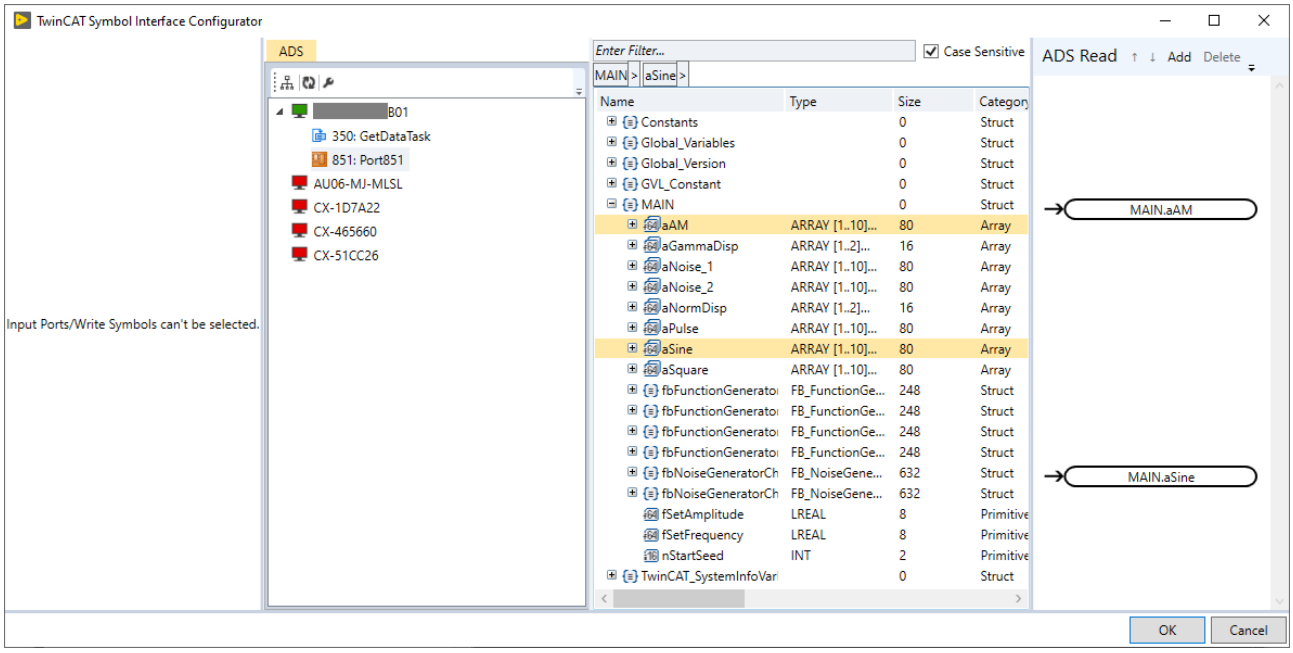
LabVIEW™-Projekt erstellen

✓ LabVIEW™ ist geöffnet.

1. Erstellen Sie ein neues Projekt und öffnen Sie ein leeres VI.
2. Speichern Sie das VI.
3. Platzieren Sie auf dem Blockdiagramm eine Instanz des ADS DAQ VI. Navigieren Sie dazu in der **Functions-Palette** zu **User Libraries > Beckhoff-LabVIEW-Interface > ADS DAQ**.
   ⇨ Es öffnet sich automatisch ein User Interface zur Konfiguration der Verbindung zu TwinCAT.
4. Wählen Sie **Symbol Interface** aus, um eine neue Konfiguration zu erstellen.

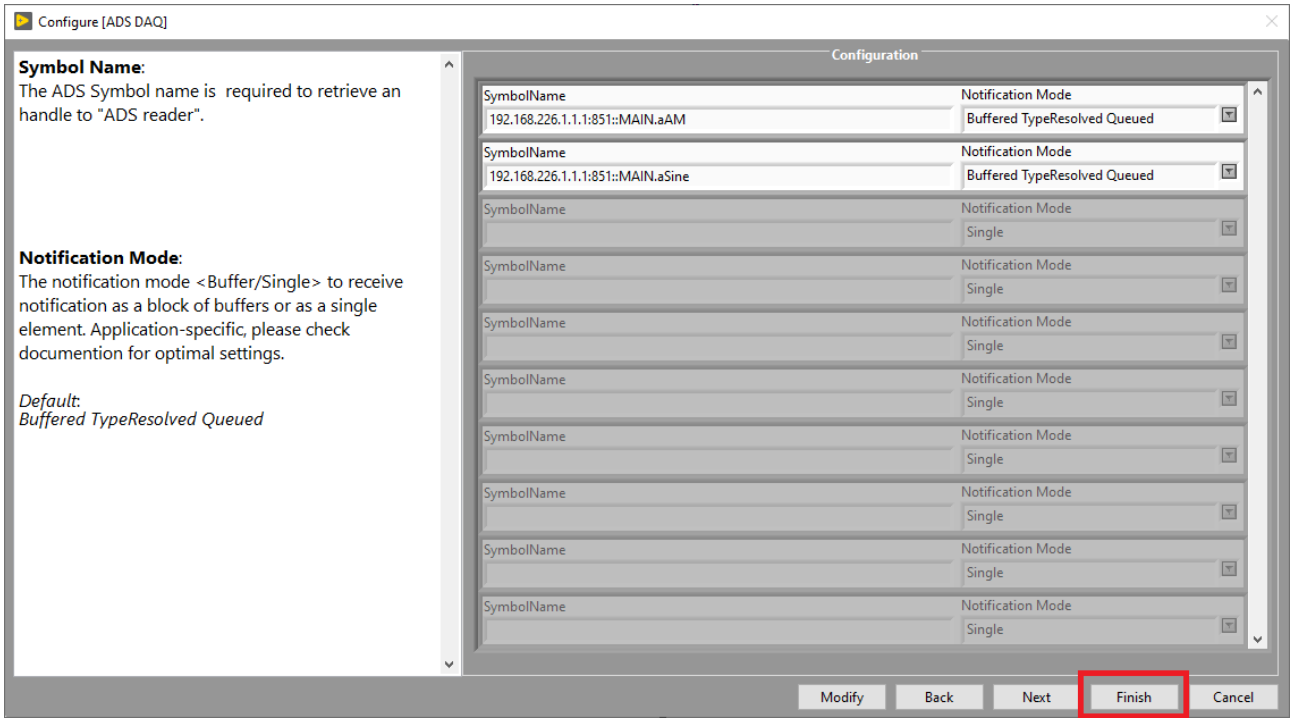


5. Browsen Sie mit dem Target Browser im mittleren Feld in Ihr Target auf dem Sie das TwinCAT-Projekt aktiviert haben.
6. Wählen Sie dann die zu lesenden ADS-Symbole. Navigieren Sie zu **851: Port851 > MAIN** und selektieren Sie die ADS-Symbole aAM und aSine. Ziehen Sie beide Symbole per Drag-and-drop auf die rechte Fläche (**ADS Read**-Fläche).

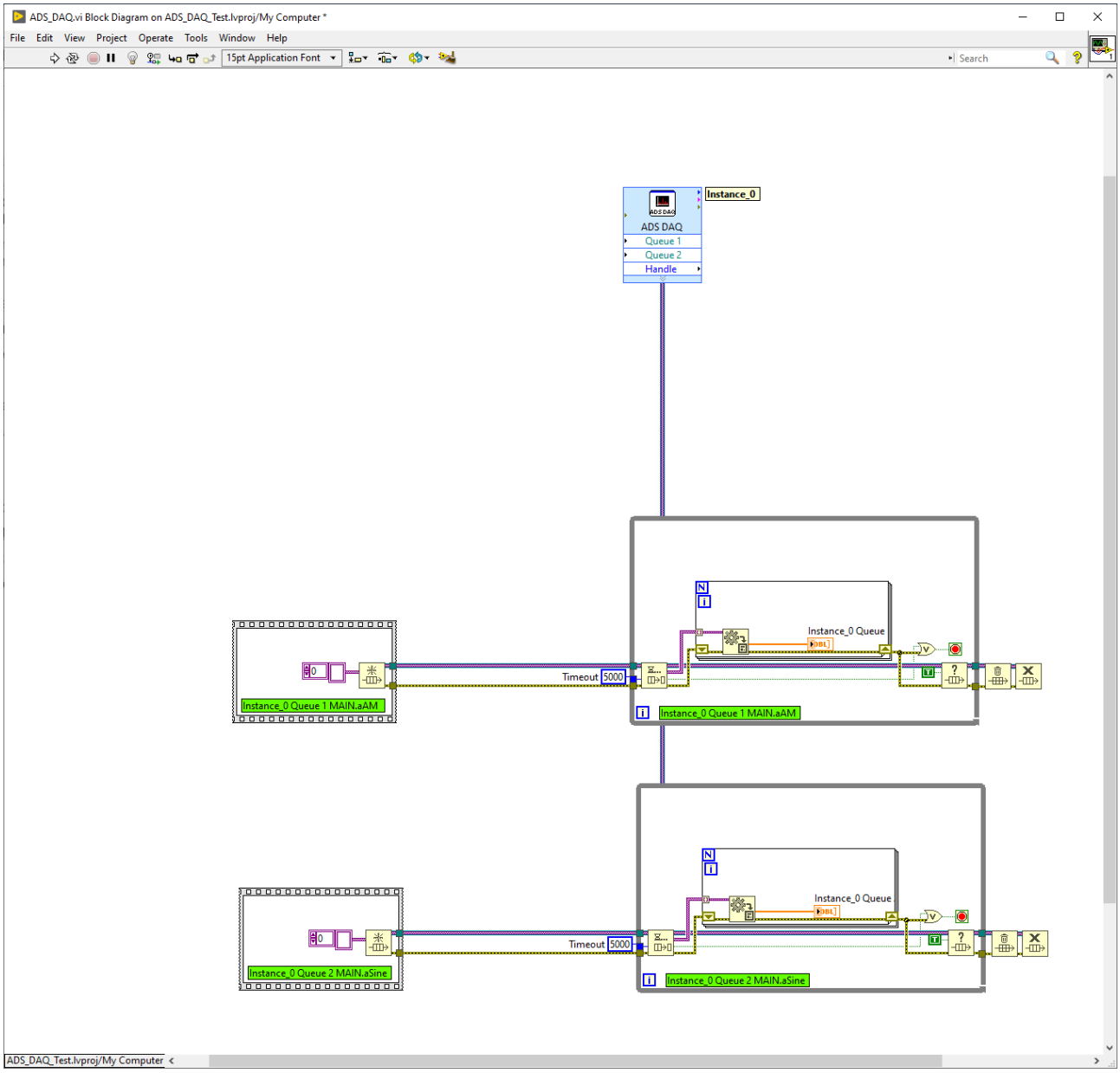


7. Wählen Sie **OK**.

   ⇨ Der Symbol Interface Configurator schließt sich und Sie bekommen weitere Einstellmöglichkeiten angezeigt. Belassen Sie es zunächst bei den Default-Einstellungen und wählen Sie **Finish**.



   ⇨ Das User Interface wird geschlossen und es wird entsprechend der Konfiguration automatisch Code auf dem Blockdiagramm generiert. Diesen können Sie individuell verändern und erweitern.

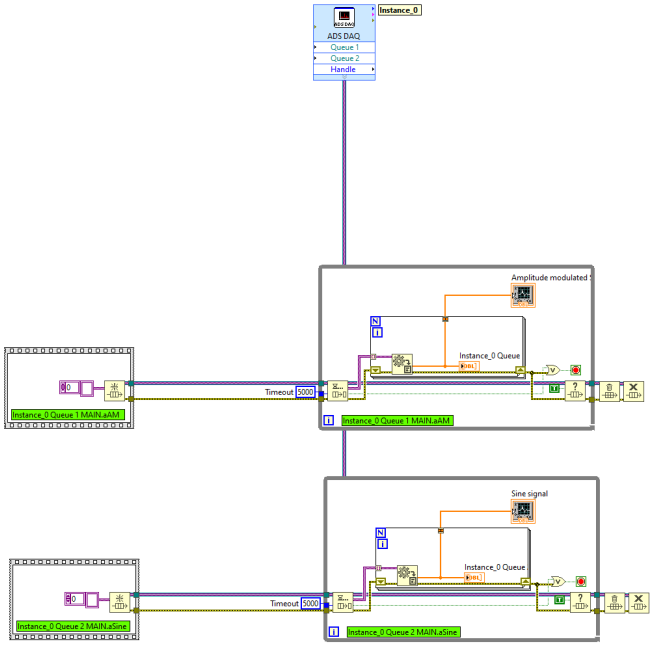


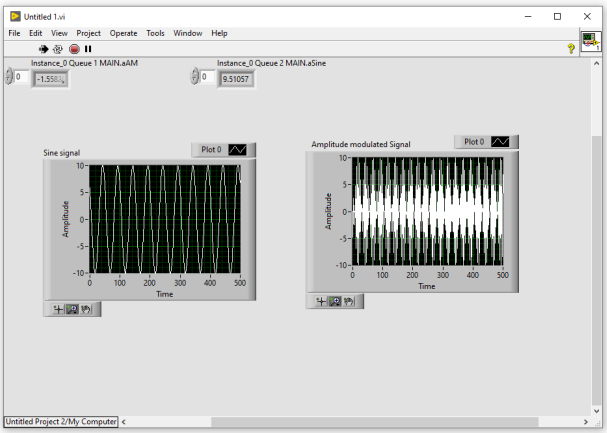
Blockdiagramm erweitern

✓ Code wurde auf dem Blockdiagramm generiert (siehe oben).

8. Erweitern Sie das Blockdiagramm, z. B. mit einem Waveform Graph.
9. Setzen Sie Ihr VI in den Run-Modus.

⇨ Betrachten Sie -in diesem Fall- die beiden Zeitsignale auf dem Front Panel.


Instance_0
ADS DAQ
Queue 1
Queue 2
Handle
Amplitude modulated
Instance_0 Queue
Timeout
5000
Instance_0 Queue 1 MAIN.aAM
Sine signal
Instance_0 Queue 2 MAIN.aSine


Untitled 1.vi
File Edit View Project Operate Tools Window Help
Instance_0 Queue 1 MAIN.aAM
Instance_0 Queue 2 MAIN.aSine
Sine signal
Plot 0
Amplitude
Time
Amplitude modulated Signal
Untitled Project 2/My Computer

6 Technische Einführung

Der Datenaustausch zwischen LabVIEW™ und TwinCAT 3 erfolgt anhand des Server-Client-Protokolls Automation Device Specification (ADS). Mit dem Interface for LabVIEW™ wird in LabVIEW™ ein ADS-Client realisiert, welcher einen performanten Datenaustausch mit TwinCAT-Laufzeiten ermöglicht. Die TwinCAT-Laufzeiten realisieren entsprechend den ADS-Server.

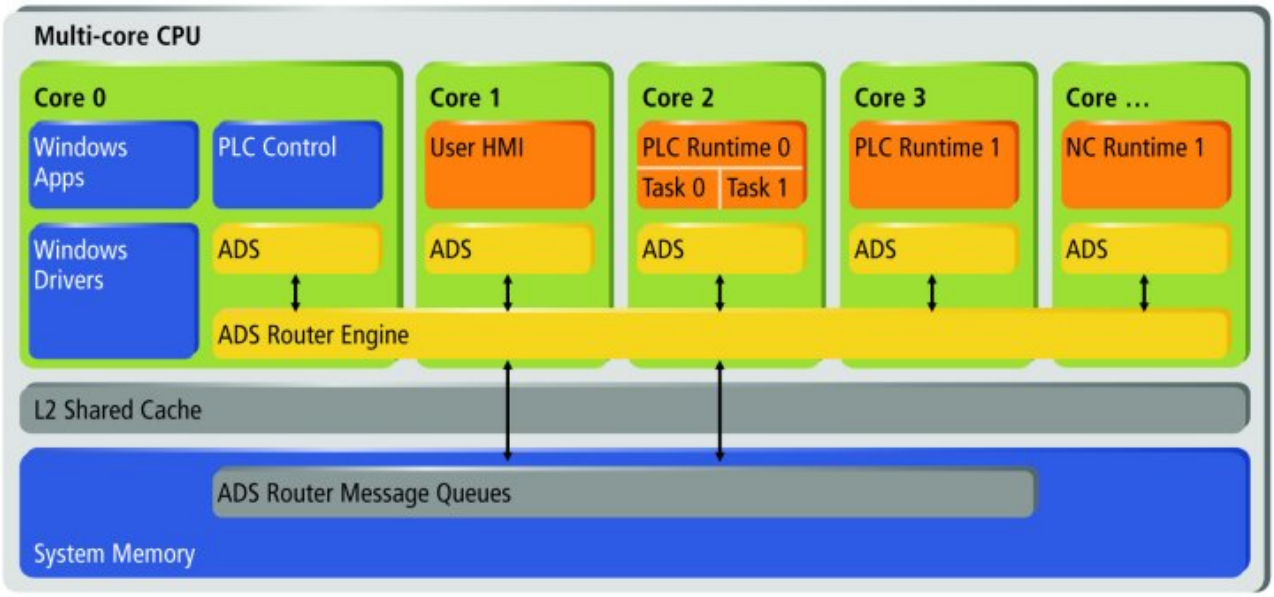
6.1 TwinCAT ADS

Grundstruktur von ADS-Geräten und ADS-Symbolen

Automation Device Specification (ADS) bildet die Grundlage für das Interface for LabVIEW™. ADS beschreibt eine geräte- und feldbusunabhängige Schnittstelle und ermöglicht die Kommunikation zwischen ADS-Geräten.

Nachfolgend wird das ADS-Gerätekonzept sowie die Identifikation eines ADS-Geräts erläutert.

Die modulare Systemarchitektur von TwinCAT erlaubt es, die einzelnen Teile der Software (z. B. TwinCAT PLC, TwinCAT NC ...) als eigenständige Geräte zu betrachten: Für jede einzelne Aufgabe gibt es ein Softwaremodul. Die Server im System sind die ausführenden Geräte, welche bestimmte Dienste anbieten. Die Clients sind Programme, welche die Dienste der Server anfordern. Ein Client baut entsprechend initial eine Verbindung zum Server auf und fragt einen Dienst an: Zum Beispiel fragt er lesend den Wert einer Variablen an oder er fragt das Schreiben einer Variablen an.



Das Interface for LabVIEW™ stellt eine ADS-Client-Schnittstelle zur Verfügung, mit welcher ein Datenaustausch (schreibend und lesend) mit TwinCAT-Laufzeiten ermöglicht wird. Die TwinCAT-Laufzeit, bzw. ihre ADS-Geräte, stellen damit als ADS-Server ihre Dienste zur Verfügung und können aus LabVIEW™ heraus genutzt werden.

Der ADS-Datenaustausch zwischen ADS-Geräten findet über den ADS-Router statt. Wie im obigen Schaubild aufgezeigt, erfolgt der Datenaustausch von ADS-Geräten, welche auf demselben System realisiert sind, über den Systemspeicher. Sind zwei ADS-Geräte, z. B. LabVIEW™ und die TwinCAT-Laufzeit, auf unterschiedlichen Systemen vorhanden, kann eine Route zwischen zwei ADS-Routern erzeugt werden. Beim Erzeugen der ADS-Route kann der Transporttyp (i.d.R. TCP/IP) für die Kommunikation zwischen den beiden ADS-Routern definiert werden. Entsprechend identifiziert sich ein ADS-Gerät durch die AMS-NetId des ADS-Routers und durch eine Portnummer, welche dann das ADS-Gerät auf dem System spezifiziert. Beispielsweise ist der Port 851 der Default-Port für die erste PLC-Instanz in der TwinCAT-Laufzeit. ADS-Dienste eines ADS-Geräts werden dann durch zwei Parameter, den Index Group und den Index Offset, spezifiziert. Zum Beispiel ist eine SPS-Variable zum Lesen oder Schreiben unter einem bestimmten Index Group und Index Offset erreichbar.

Soll eine Anwendung erstellt werden, die auf mehreren Zielsystemen eingesetzt werden soll und bei der LabVIEW™ und TwinCAT auf demselben System laufen, kann die relative AMS NetId verwendet werden. Die relative AMS NetId deutet immer auf das lokale System. Hierfür wird 0.0.0.0.0.0 für die Adresse eingesetzt.

Zusammenfassung

- AMS NetId: Identifiziert den ADS-Router, also das System.
- ADS-Route: Spezifiziert die Verbindung zwischen zwei ADS-Routern.
- Port: Identifiziert ein ADS-Gerät.
- Index Group/Offset: Spezifiziert den ADS-Systemdienst, z. B. eine Variable zum Lesen und Schreiben.

Um die Adressierung von Variablen in einer TwinCAT-Laufzeit für den Anwender komfortabler zu gestalten, erstellt TwinCAT ADS-Symbole, welche z. B. mit dem Target Browser durchsucht werden können. Der Target Browser ist auch im Interface for LabVIEW™ im VI Symbol Interface [▶ 65] integriert, sodass ADS-Symbole einfach und schnell selektiert werden können. Ein ADS-Symbol für eine Variable in TwinCAT enthält dann die oben genannten Informationen: AMS NetId, Port, Index Group und Index Offset und darüber hinaus die Bit Size sowie einen Symbolnamen und den Datentyp der Variablen.

Weitere Informationen zu ADS finden Sie unter folgenden Links:

- AMS NetId und Port: ADS device identification
- Index Group und Index Offset: Specification for ADS devices
- ADS-Routen
- Connecting Devices with same AMS NetId: AmsNAT
- Using MQTT and Message Broker with ADS: ADS-over-MQTT
- TLS encrypted ADS: Secure ADS
- Target Browser
- ADS Kommunikation aufzeichnen: ADS Monitor

Grundlegende ADS-Datenkommunikation

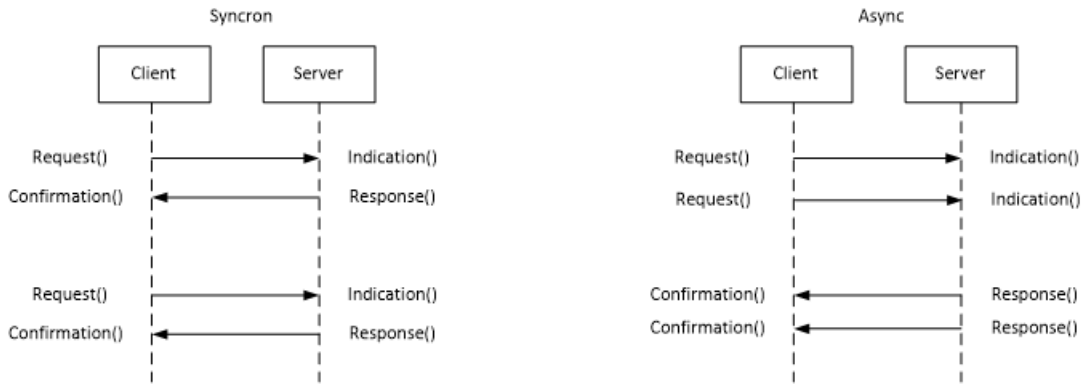
Grundlegend gibt es bei ADS drei unterschiedliche Möglichkeiten/Modi zur Datenkommunikation.

Synchrones Lesen oder Schreiben

- Der ADS-Client in LabVIEW™ wartet auf eine Antwort des ADS-Servers, bevor mit der Ausführung des Codes fortgefahren wird.

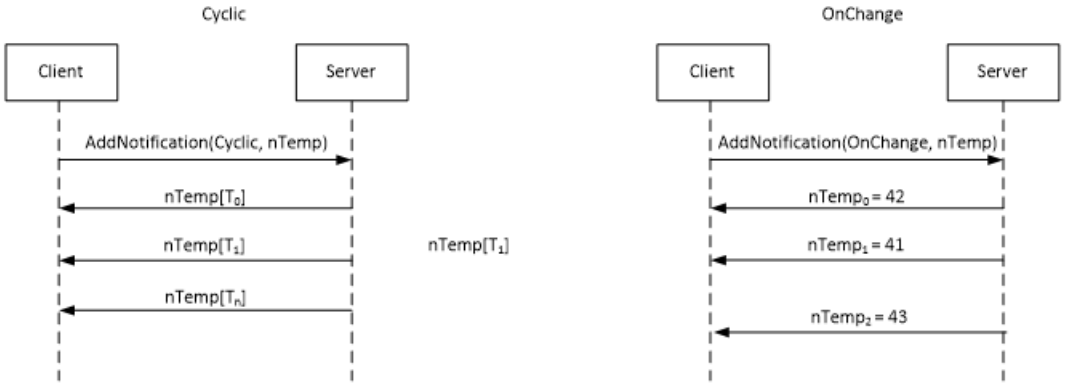
Asychrones Lesen oder Schreiben

- Der ADS-Client in LabVIEW™ sendet eine Lese- oder Schreibanfrage an den ADS Server, wartet aber nicht auf eine Antwort, sondern führt weitere Programmteile (z. B. Anfragen an andere Variablen) weiter aus.



Eventbasierte Kommunikation (nur Lesen)

- Es wird nur einmalig eine sogenannte ADS-Notification am Server angemeldet. Die Notification kann „on change“ oder „cyclic“ angemeldet werden. Nach der Anmeldung einer Notification am Server sendet dieser angeforderte Daten ohne weitere Anfragen des Clients an den Server.



Die VIs ADS Read [▶ 67] und ADS Write [▶ 73] sind polymorph. Der gewünschte Kommunikationsmodus, sync, async oder notification, kann dort selektiert werden. Die VIs bieten darüber hinaus noch Erweiterungen dieser Grundmodi an, die das LabVIEW™-seitige buffering von Daten und deren Weitergabe an den LabVIEW™-Prozess betreffen.

6.2 Kommunikations-Modi

Erste Einordnung

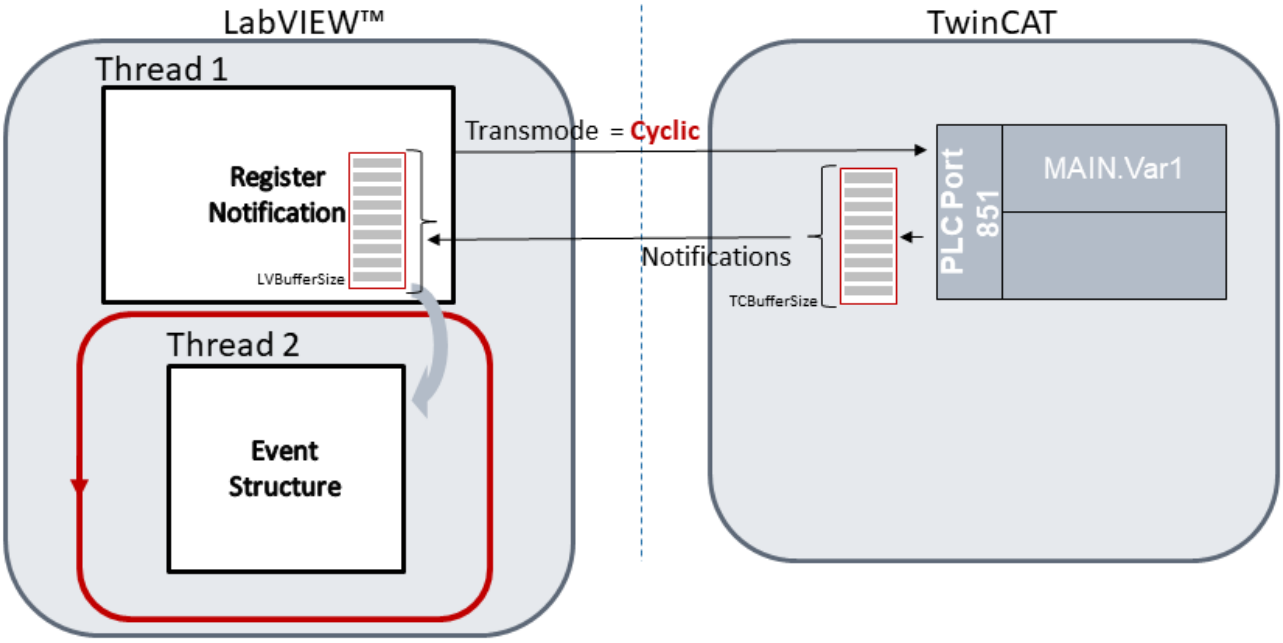
Die Erweiterungen der Grundfunktionalitäten, sync, async und notification, werden im Folgenden erläutert. Ebenso werden praktische Empfehlungen gegeben, welcher Modus für exemplarische Anwendungen die bevorzugte Wahl darstellt.

Folgende Tabelle zeigt alle bereitgestellten Modi mit einer ersten Einordnung.

Modus	Read	Write	Kurzbeschreibung
Sync Single [▶ 29]	X	X	Lesen/Schreiben einer Variablen in einer abgeschlossenen einmaligen Aktion. Der Client wartet während der Anfrage an den Server.
Async Single [▶ 29]	X	X	Lesen/Schreiben einer Variablen in einer abgeschlossenen einmaligen Aktion. Der Client wartet während der Anfrage an den Server nicht.
Noti. Single [▶ 29]	X		Warten auf ein „On Change“-Event in TwinCAT als einmalige Aktion. Die Notification wird nach dem Event abgemeldet.
Noti. Buffered [▶ 30]	X		Einmaliges Lesen einer Zeitreihe mit definierter Länge. Nach Erhalt der Zeitreihe wird die Notification abgemeldet.
E-Noti. Single [▶ 34]	X		Kontinuierliches Lesen (zyklisch oder „on change“). Die Notification wird nicht automatisch abgemeldet, sondern läuft bis sie explizit abgemeldet wird. Die empfangenen Notifications werden in LabVIEW™ an eine Ereignisstruktur weitergeleitet.
E-Noti. Buffered [▶ 35]	X		Gepuffertes kontinuierliches Lesen (zyklisch oder „on change“). Die Notification wird nicht automatisch abgemeldet, sondern läuft, bis sie explizit abgemeldet wird. Die empfangenen Notifications werden in LabVIEW™ an eine Ereignisstruktur weitergeleitet. Der Puffer ermöglicht einen höheren Datendurchsatz, verursacht jedoch Latenz.

Modus	Read	Write	Kurzbeschreibung
LVB-Noti. Single [▶ 32]	X		Kontinuierliches Lesen (zyklisch oder „on change“). Die Notification wird nicht automatisch abgemeldet, sondern läuft bis sie explizit abgemeldet wird. LabVIEW-seitig werden die Daten in einen Puffer geschrieben, auf den der Nutzer direkten Zugriff hat.

In der obigen Tabelle wird einigen Modi die Eigenschaft „Buffered“ zugesprochen. Das untere Schaubild verdeutlicht den Systemaufbau. Bei „Noti. Buffered“ und „E-Noti. Buffered“ bezieht sich der angesprochene Buffer auf einen LabVIEW™-seitigen Buffer der Größe `LVBufferSize`. Der Buffer wird mit Daten, welche aus TwinCAT gelesen werden, gefüllt, und beim Erreichen der Buffer-Größe an LabVIEW™ übergeben. Darüber hinaus gibt es aus dem Standard-ADS-Interface einen TwinCAT-seitigen Buffer, welcher allerdings nur bei ADS-Notifications vom Typ *Cyclic* verwendet werden kann. Standardmäßig ist dieser Buffer `TcBufferSize` = 10 Samples, kann aber parametriert werden.



Betrachtet wird im Folgenden exemplarisch obiges Beispiel mit den Parametern:

TcBufferSize = 10

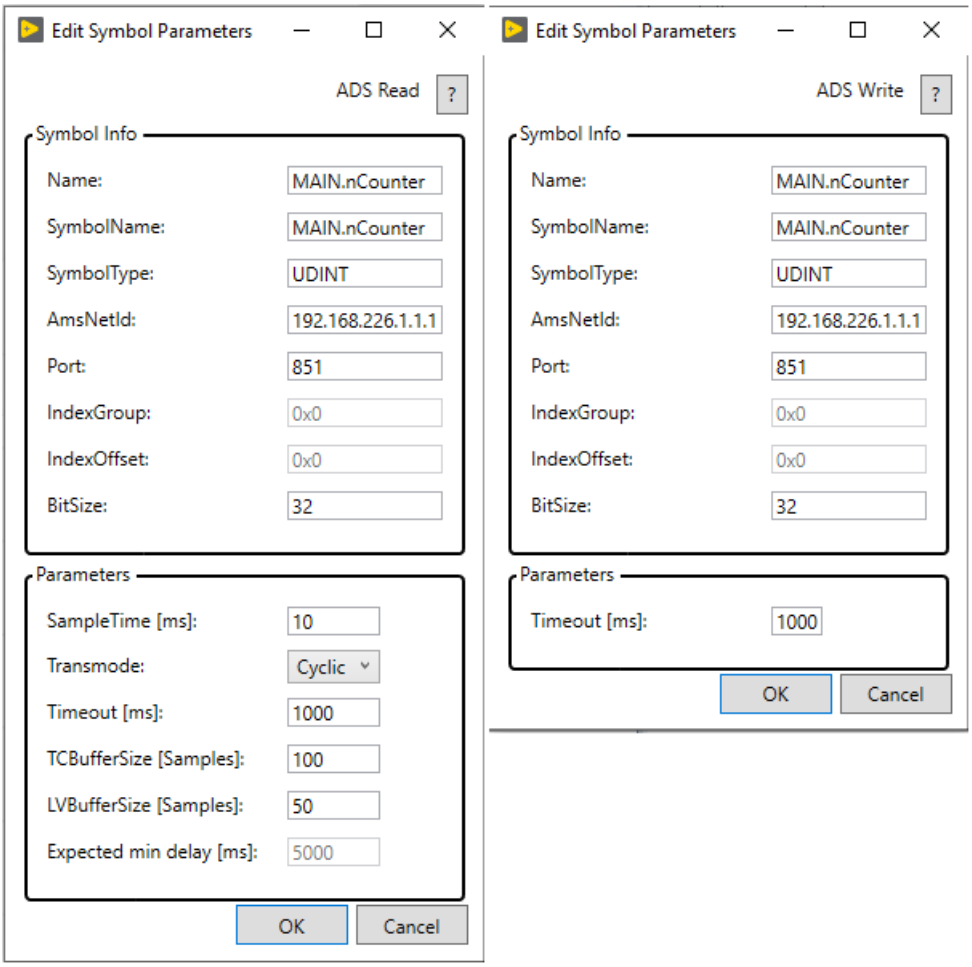
LVBufferSize = 500

ADS-Notification vom Typ Cyclic mit 1 ms Sample Time (entspricht hier der Zykluszeit der SPS).

Alle 1 ms wird nun TwinCAT-seitig ein Sample in den TwinCAT-seitigen Buffer geschrieben. Nach 10 ms ist der Buffer gefüllt und wird an den LabVIEW™-seitigen Buffer übertragen. Nach dem Empfang von 50 Nachrichten mit jeweils 10 Samples, also nach 500 ms (plus Kommunikationszeit), ist der LabVIEW™-seitige Buffer gefüllt und übergibt dann das gesamte Datenpaket von 500 Samples an LabVIEW™. Entsprechend existiert eine *Delay-Time* von mindestens 500 ms bis zum Erhalt des Datenpakets in LabVIEW™. Die minimal zu erwartende Verzögerungszeit wird auch im Konfigurationsdialog (Edit Symbol Parameters) im unteren Teil als „Expected min delay“ angezeigt.

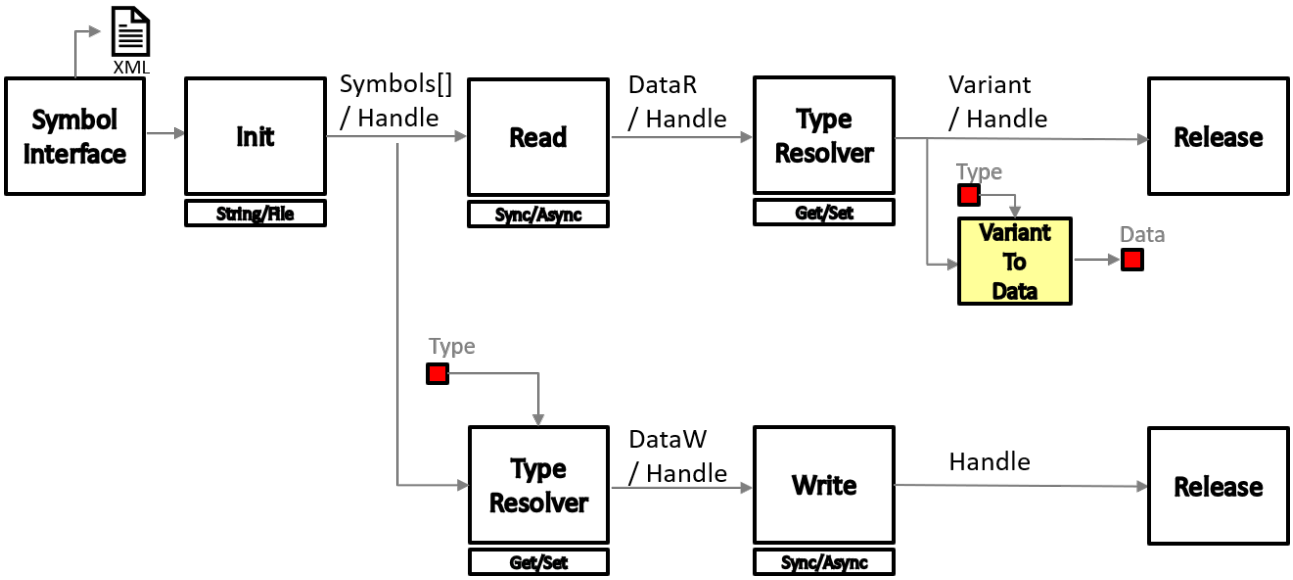
Parametrieren und Realisieren der Kommunikations-Modi

Die im obigen Abschnitt genannten Parameter können im User Interface (UI) des Symbol Interface [▶ 65] eingestellt werden.



Neben der Parametrierung der ADS-Symbole ist die Grundstruktur des LabVIEW™-Programms zu beachten, welche nachfolgend skizziert ist.

Das Symbol Interface [▶ 65] öffnet einen Dialog mit dem Target Browser und der Möglichkeit, jedes ADS-Symbol nach obiger Abbildung zu parametrieren. Der Ausgang des Symbol Interface ist eine Zeichenkette in XML-Format, welche alle selektierten ADS-Symbole und deren Parametrierung beschreibt. Diese Zeichenkette wird mit dem Init VI [▶ 67] verknüpft. Alternativ kann das Symbol Interface die XML exportieren. Diese XML-Datei kann dann direkt an das Init VI übergeben werden. Das Init VI übergibt dann Handles der ADS-Symbole an ein Read VI [▶ 67], welches als polymorphic VI zu den oben beschriebenen Optionen eingestellt werden kann. Am Ausgang liegen Data-Handles, welche mit einem Type Resolver VI [▶ 74] verknüpft werden. Der Ausgang des Type Resolver VI ist ein Variant Type, welcher in das korrekte Format gecastet werden kann. Beim Schreiben ist die Reihenfolge von Type Resolver und Write VI [▶ 73] entsprechend vertauscht.



Die oben genannte Struktur zeigt schematisch den Aufbau mit Low Level VIs. Zur Vereinfachung in der Programmierung werden, je nach Kommunikations-Modus, mehrere Low Level VIs kombiniert. Siehe dazu z. B. Synchron Lesen [▶ 29] oder Asynchron Lesen [▶ 29].

Praktische Erwägungen

Es existieren diverse Möglichkeiten zum Lesen von Daten. Um die Auswahl in der Praxis einfacher zu gestalten, ist nachfolgend eine Tabelle aufgeführt, welche exemplarische Anwendungsszenarien beschreibt und einen Kommunikations-Modus vorschlägt.

Signal-Typ in TwinCAT	Beispiel	Kommunikations-Modus
Quasi-statische Parameter	Lese den aktuellen Parametersatz eines PID-Reglers oder die ID des aktuellen Produkts in der Maschine. Lesen von Einstellungen der Maschine.	Sync Read [▶ 29], Async Read [▶ 29]
Kontinuierliches Signal	Stream eines oder mehrerer Sensorsignale (Temperatur, Dehnung, Kraft, …). Es existiert kein definiertes Ende des Datenstroms. Allgemeine DAQ Applikation	Schnelle Reaktion notwendig: E-Noti. Single [▶ 34] (Transmode „cyclic“)
		Hoher Datendurchsatz: E-Noti. Buffered [▶ 30] oder LVB-Noti. Single [▶ 32]
Event	Warten auf TwinCAT Event, z. B. Start eines Prozesses: bStart wechselt von FALSE auf TRUE. Das Event taucht einmalig auf und wird dann nicht mehr benötigt und kann abgemeldet werden.	Noti. Single [▶ 29] (Transmode „on change”)
Event	Reagieren auf Veränderungen von Zuständen, z. B. Veränderung der zu lesenden Signale je nach State der Maschine: Betrachten der nState-Variablen. Das Event tritt häufig auf und dieses Event soll durchgängig beobachtet werden.	E-Noti. Single [▶ 34] (Transmode „on change”)
Signal definierter Länge	Lesen von genau 1000 Samples eines bestimmten Sensorsignals.	Noti. Buffered [▶ 30]

6.2.1 Einmaliges Lesen

Dieser Kommunikations-Modus ist ideal, um z. B. eine TwinCAT-Konfiguration oder einen vollständig gefüllten Datenpuffer in der SPS einmalig zu lesen. Das einmalige Lesen benötigt, im Gegensatz zum kontinuierlichen Lesen [▶ 31], keine zusätzliche Programmierung in LabVIEW™.

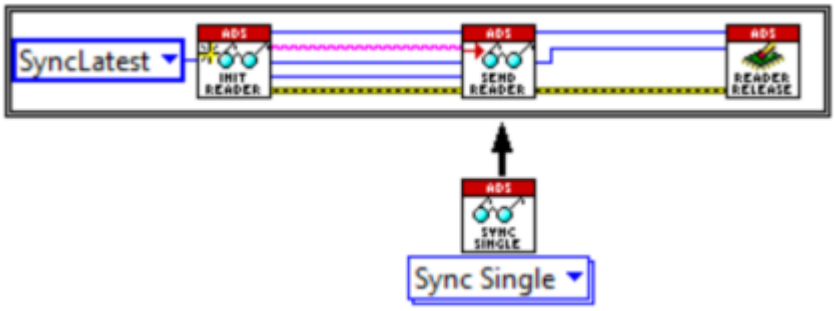
Das Produkt TF3710 TwinCAT 3 Interface for LabVIEW™ kategorisiert das einmalige Lesen in vier Fälle:

1. Synchron Lesen
2. Asynchron Lesen
3. Notification Single
4. Notification Buffered

Synchron Lesen

Beim synchronen Lesen wird, nach Absenden einer Anfrage vom Client, auf eine Antwort vom ADS-Server gewartet, bevor der Programmcode weiter ausgeführt wird. Der Reader wird sofort nach einer erfolgreichen Rückmeldung vom Server freigegeben. Somit eignet sich diese Art des Lesens, um direkt nach Erhalt der Daten aus TwinCAT diese zu verrechnen oder anzuzeigen.

Der polymorphic Block „**Sync Single**“ ist zusammengesetzt aus mehreren Low-Level VIs [▶ 85] und verbindet somit das Erstellen eines Handles, das Lesen und das Freigeben.

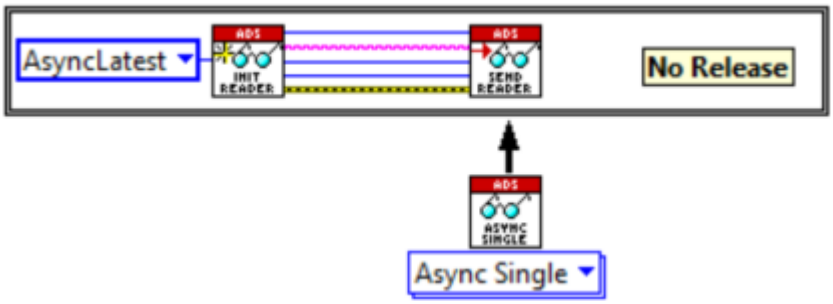


Beispiele in LabVIEW™: Grundlegende Beispiele [▶ 104]

Asynchron Lesen

Beim asynchronen Lesen wird Client-seitig nicht auf eine Antwort vom ADS-Server gewartet. Bei dieser Betriebsart kann nicht gewährleistet werden, dass der Reader das Datenpaket schon empfangen hat oder nicht. Als Folge kann der Reader auch nicht freigegeben werden.

Das polymorphic VI „**Async Single**“ initialisiert daher den Reader und schickt die Anfrage an TwinCAT. Das Freigeben (Release) ist nicht Teil des VI.



Beispiele in LabVIEW™: Grundlegende Beispiele [▶ 105]

Notification Single

Notification Single wird nur für den *Transmode* **„on change“** unterstützt.

Der Block **„Noti. Single“** benötigt, im Gegensatz zu Notification E-Single [▶ 34], keine durch den Nutzer programmierte Ereignisstruktur. Dafür ist die Notification auch nur für einen einmalige Nutzen, d. h. Einfangen eines einmaligen Events gedacht. Die Notification wird im Hintergrund registriert, empfangen und dann entfernt.

Dieser Block liefert als Rückgabewert ein Array von zwei Werten, den letzten Zustand vor der Wertänderung und den neuen Zustand nach der Wertänderung, also z. B. [FALSE, TRUE], wenn eine Variable den Wert von FALSE auf TRUE verändert hat.

Beispiele in LabVIEW™: Grundlegende Beispiele [▶ 104]

Notification Buffered

Der Block **„Noti. Buffered“** benötigt, im Gegensatz zu Ereignisgesteuertes Lesen [▶ 35], keine durch den Nutzer programmierte Ereignisstruktur. Der Block nutzt einen Puffer der Größe `LVBufferSize`, um die von TwinCAT empfangenen Daten in einer Mittelschicht zu speichern. Erst wenn der Puffer voll ist, wird die Notification entfernt und danach die gespeicherten Daten an LabVIEW™ weitergegeben.

Entsprechend ist mit diesem Kommunikations-Modus der Empfang einer Zeitreihe mit vordefinierter Länge (LVBufferSize Samples) einfach zu realisieren.

HINWEIS

LVBufferSize

Die Puffergröße wird durch den Parameter `LVBufferSize` beim Erzeugen vom ADS-Symbol bestimmt, siehe Symbol Interface [▶ 65] VI.

Beispiele in LabVIEW™: Grundlegende Beispiele [▶ 104]

6.2.2 Einmaliges Schreiben

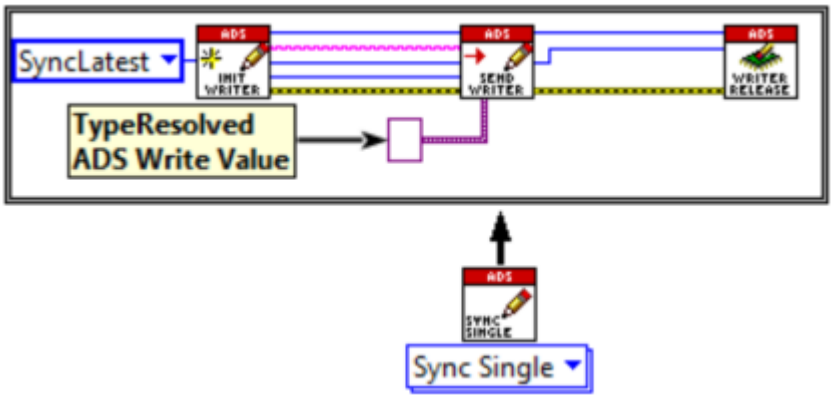
Das TwinCAT 3 Interface for LabVIEW™ kategorisiert das einmalige Schreiben wie folgt:

1. Synchron Schreiben
2. Asynchron Schreiben

Synchron Schreiben

Beim synchronen Schreiben wird auf eine Antwort des Servers (TwinCAT) gewartet. Der Writer wird nach einem erfolgreichen Versand des Datenpakets an TwinCAT freigegeben.

Dieser Prozess wird von dem polymorphic VI „**Sync Single**“ im Hintergrund durchgeführt.

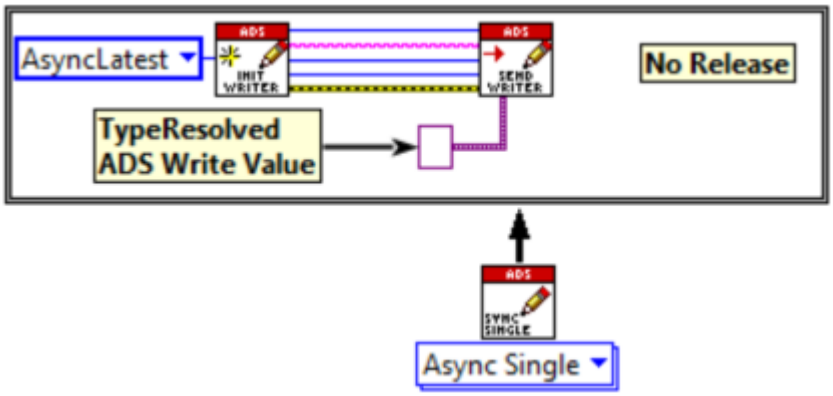


Beispiele in LabVIEW™: Grundlegende Beispiele [▶ 106]

Asynchron Schreiben

Beim asynchronen Schrieben wird nicht auf eine positive Quittierung des Servers gewartet. Bei dieser Betriebsart kann nicht gewährleistet werden, dass der Writer das Datenpaket schon versendet hat oder nicht. Als Folge kann der Writer auch nicht freigegeben werden.

Das polymorphic VI „**Async Single**“ initialisiert daher den Writer und schickt nur die Anfrage an den Server. Eine Freigabe (Release) wird nicht im VI durchgeführt.



Beispiele in LabVIEW™: Grundlegende Beispiele [▶ 106]

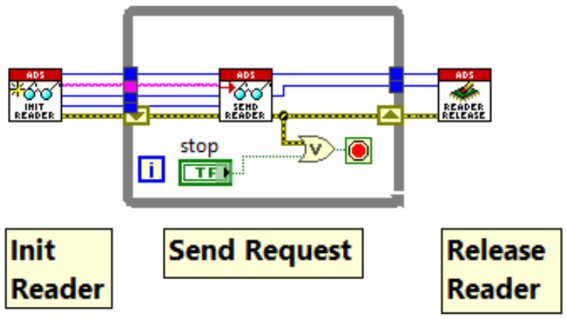
6.2.3 Daten kontinuierlich lesen

Beim kontinuierlichen Lesen einer Zeitreihe aus TwinCAT bekommt der Client in LabVIEW™ kontinuierlich Datenpakete vom ADS-Server. Die Datenpakete können einerseits im Polling-Verfahren zyklisch und andererseits als ADS-Notification event-basiert angefragt werden. Beide Varianten werden im Folgenden erläutert.

Lesen mit Polling-Zyklus

Einfaches Lesen

Beim Polling sendet der Client in einem definierten Zeitabstand Anfragen an den Server. Dies kann beispielsweise mit Hilfe der Low-Level [▶ 85] VIs einfach aufgebaut werden. Der ADS-Reader wird nur einmal initialisiert und fragt mit jedem Zyklus der Schleife ein neues Datenpaket aus TwinCAT an. Mit jedem Zyklus wird eine neue Anfrage an den Server gesendet und auf eine entsprechende Antwort gewartet. Nach Erreichen der Abbruchbedingung der Schleife wird der ADS-Reader freigegeben. Das Bild unten zeigt die vollständige Vorgehensweise in LabVIEW™.

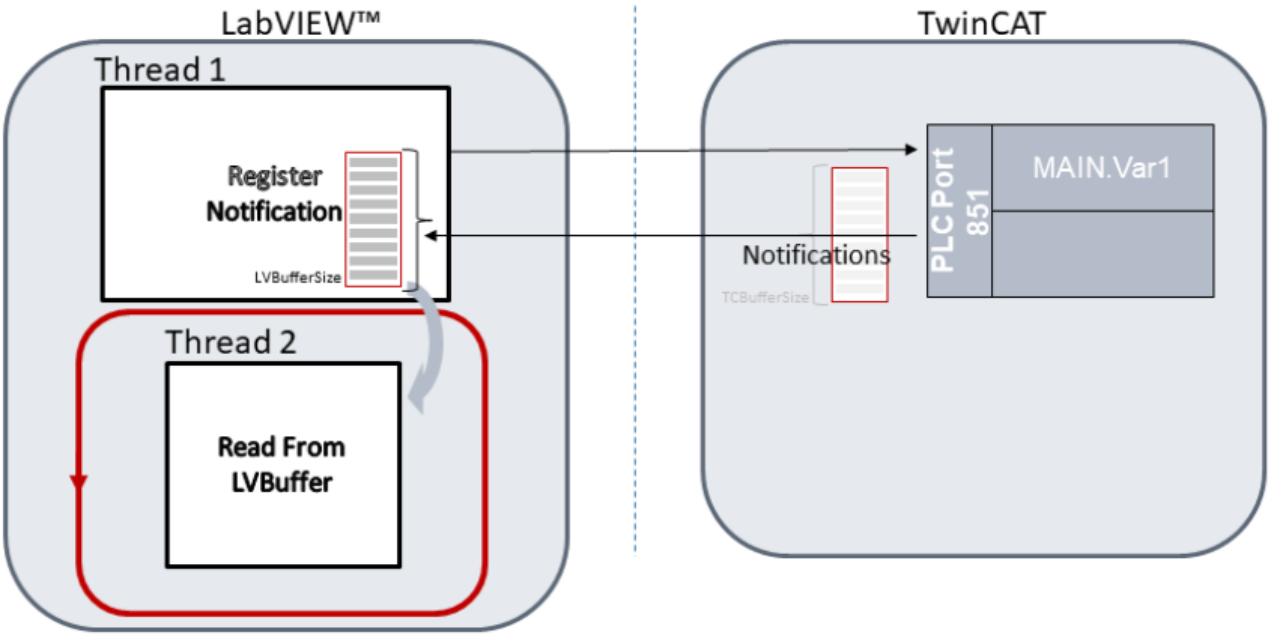


Beispiel in LabVIEW™: Grundlegende Beispiele [▶ 107]

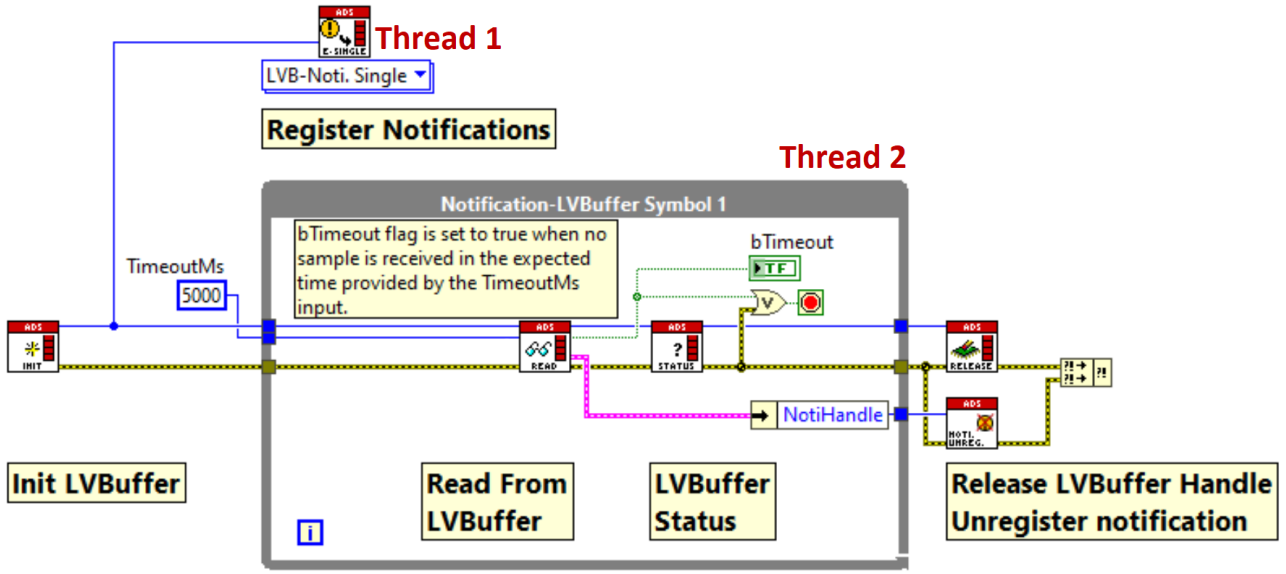
Das Verfahren ist fehleranfällig in der Hinsicht, dass nicht garantiert werden kann, dass ein sich zyklisch ändernder Wert in der SPS lückenfrei gesampled wird. Für diesen Use Case wird empfohlen, mit E-Notifications zu arbeiten, welche im Folgenden beschrieben sind.

Lesen mit LVB-Notification

Im Vergleich zu Ereignisgesteuertes Lesen [▶ 33], können die ADS-Notifications auch mit einem bestimmten Polling-Zyklus kontinuierlich gelesen werden. Das Lesen geschieht in diesem Fall asynchron, wie beschrieben in der Grafik unten. Hier werden die ADS-Notifications von dem Thread1 registriert und in den LVBuffer geschrieben. Später wird der LVBuffer mit dem Thread2 gelesen.



Im Gegenteil zu Notification E-Buffered [▶ 35] werden in diesem Fall keine LabVIEW™-Events eingesetzt. Der LabVIEW™-Nutzer hat dementsprechend einen direkten Zugriff auf den **LVBuffer**. In LabVIEW™ sieht das Blockdiagramm wie folgt aus:



Beispiele in LabVIEW™: Read Notification-LVBuffer Multiple [▶ 108]

Das Blockdiagramm nutzt die LVBuffer Blöcke (siehe LVBuffer [▶ 81]):

- Init LVBuffer Handle
- Read From LVBuffer
- LVBuffer Status
- Release LVBuffer Handle

Das Blockdiagramm nutzt die ADS Notification Blöcke:

- ADS-Read [▶ 72]
- Notification [▶ 80]

HINWEIS

Polling-Zyklus

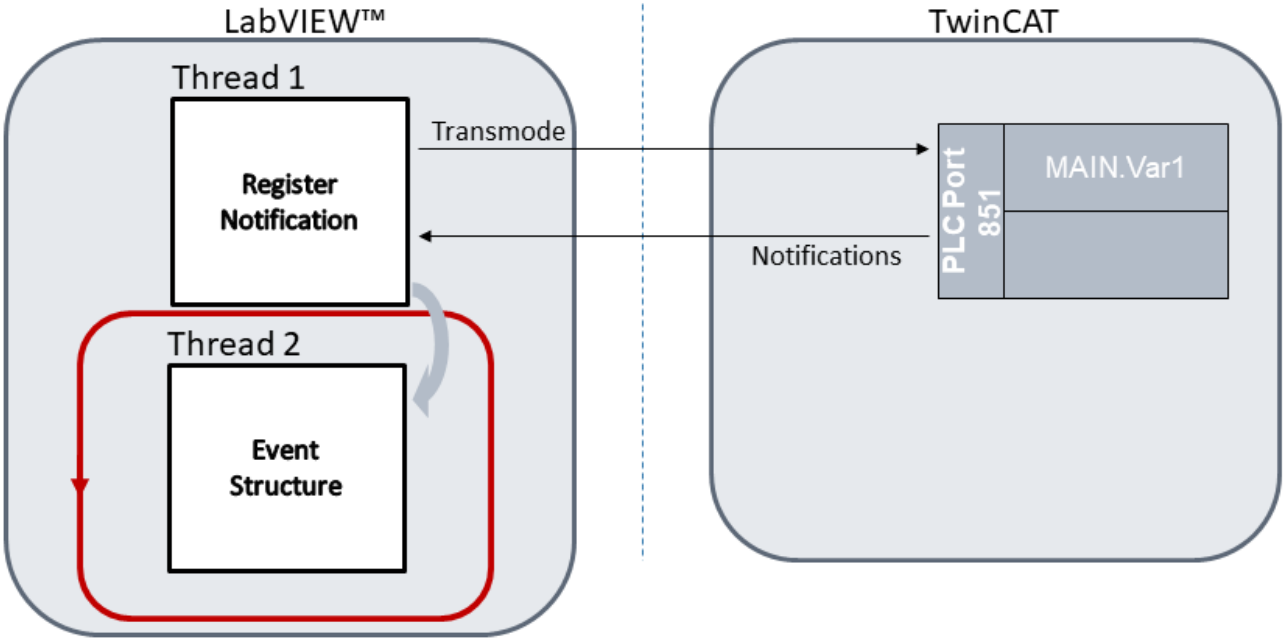
Wenn der Polling-Zyklus langsamer als die Notification-Cycle-Time ist, kann der LVBuffer volllaufen und dadurch Samples verloren gehen.

6.2.3.1 Ereignisgesteuertes Lesen

Beim ereignisgesteuerten Lesen werden LabVIEW™-Ereignisse und ADS-Notifications zusammen verwendet. Eine ADS-Notification muss nur einmalig am Server registriert werden. Danach werden die Daten des Servers zyklisch oder on change (siehe „Transmode“ im Kapitel Kommunikations-Modi [▶ 25]) vom Client empfangen. Der Client muss entsprechend nur eine Anfrage stellen und der Server steuert dann, ob eine neue Nachricht an den Client versendet werden muss.

Der ADS-Client in LabVIEW™ leitet die empfangenen Notifications (Datenpakete) als LabVIEW™-Event an eine Ereignisstruktur (*event structure*) weiter. Siehe dazu auch die LabVIEW™-Dokumentation zu User events. Die Ereignisstruktur fügt das empfangene LabVIEW™- Event in eine interne Warteschlange ein. Die Ereignisse in der Warteschlange werden von LabVIEW™ im FIFO-Prinzip abgearbeitet. Wenn die Notifications schneller empfangen werden als eine Abarbeitung der Ereignisstruktur in LabVIEW™ möglich ist, wird die Warteschlange größer. So wird sichergestellt, dass keine Notification verloren geht. Es muss beachtet werden, dass dadurch der in Anspruch genommene Speicherbedarf wächst. Dementsprechend muss berücksichtigt werden, dass ein Zeitpunkt erreicht werden muss, an dem der Speicherbedarf (die ausstehenden Events) abgearbeitet werden kann.

Es wird empfohlen, das *Event Inspector Window* von LabVIEW™ (View > Event Inspector Window) zu nutzen, um die Abarbeitung der LabVIEW™-Events zu beobachten.



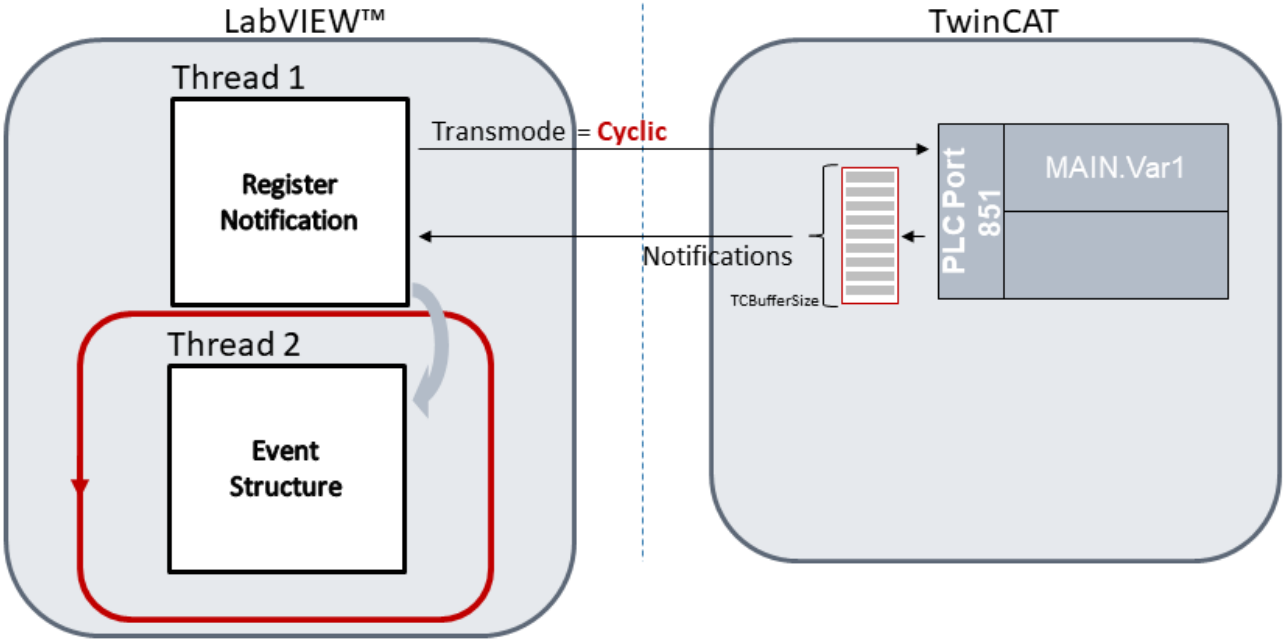
Das Interface for LabVIEW™ bietet zwei verschiedene Betriebsarten, um ADS-Notifications kontinuierlich als LabVIEW™-Events anzumelden:

1. E-Notification Single
2. E-Notification Buffered

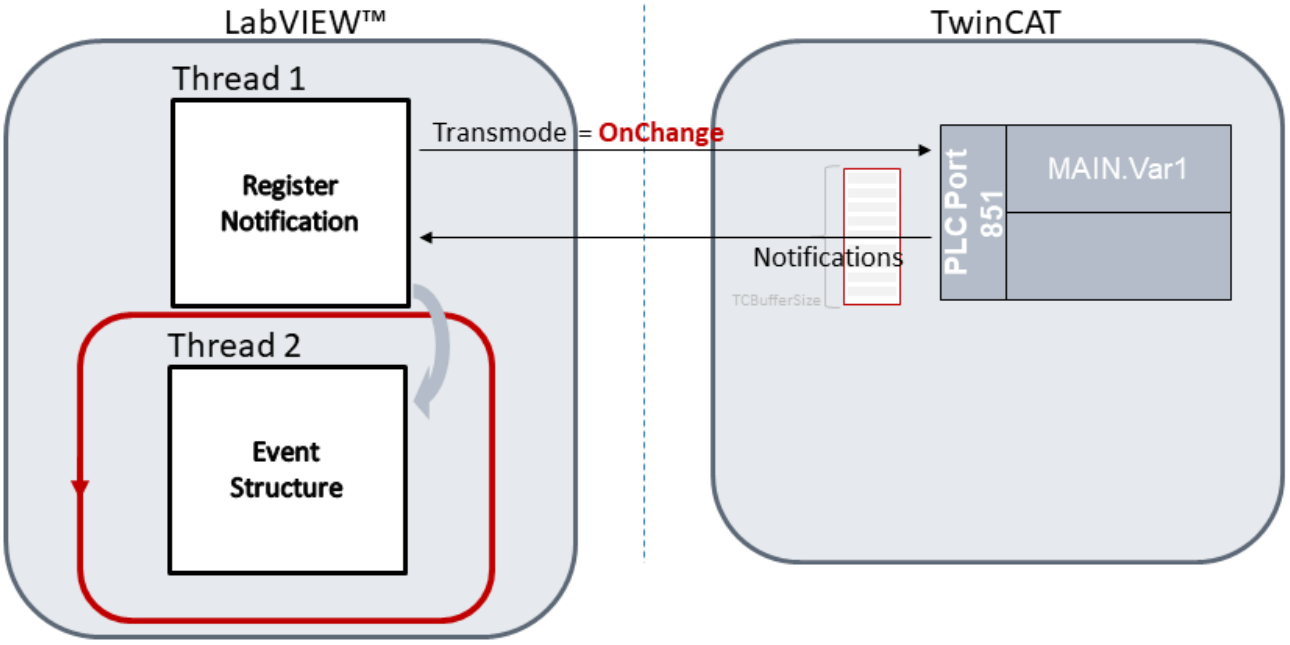
E-Notification Single

Die Betriebsart *E-Notification Single* leitet das empfangene ADS-Datenpaket sofort an die LabVIEW™-Ereignisstruktur weiter, erzeugt also direkt ein LabVIEW™-Event. In einem Datenpaket können mehrere Samples enthalten sein. Dafür sind der *Transmode* und die *TCBufferSize* entscheidende Parameter:

Bei *Transmode* „**Cyclic**“ wird der TwinCAT-seitige Puffer (der Größe TCBufferSize) genutzt. Hier werden die Notifications erst in den TCBuffer geschrieben und dann gebündelt an den LabVIEW™-Client versendet, wenn der Puffer gefüllt ist. So erreicht man einen höheren Datendurchsatz und belastet das Netzwerk nicht so stark mit vielen Nachrichten geringen Nutzdatenvolumens.



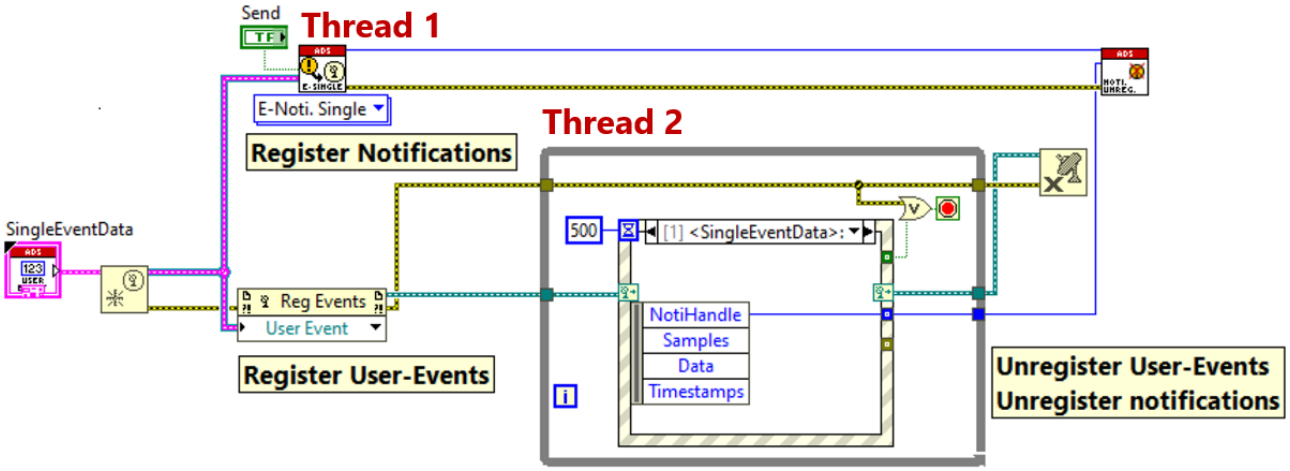
Beim *Transmode* „**OnChange**“ wird der TCBuffer **nicht** genutzt. Hier werden die einzelnen Notifications **ohne** Pufferung direkt an Client weitergeschickt. So empfängt der Client jede Änderung des Zustands mit minimaler Latenz und kann direkt darauf reagieren.



In LabVIEW™ entspricht das Blockdiagramm vom Prinzip dem obigen Bild. Hier laufen der *Register Notification Block* und die *Event Structure* in zwei unterschiedlichen Threads. Das Event wird nur einmal registriert und danach wird in der Ereignisstruktur auf die Notifications gewartet. Jede empfangene Notification enthält:

- Notification Handle
- Anzahl von Samples
- Array von Notification Data
- Array von ADS-Zeitstempeln

Wenn keine Notification empfangen wird, geht die Ereignisstruktur in einen Timeout.



Beispiele in LabVIEW™: Read Notification-Event Single [▶ 107], Read Notification-Event Multiple [▶ 108]

E-Notification Buffered

Die Betriebsart *E-Notification Buffered* nutzt zusätzlich zur obigen Struktur einen Datenpuffer auf LabVIEW™-Seite. Die Puffergröße wird vom Parameter `LVBufferSize` beim Erzeugen des ADS-Symbols festgelegt. Hier werden die empfangenen ADS-Datenpakete zunächst in einer Zwischenschicht in den LabVIEW™-seitigen Puffer geschrieben und erst dann als LabVIEW™-Event an die Eventstruktur weitergeleitet, wenn der Puffer gefüllt wurde.

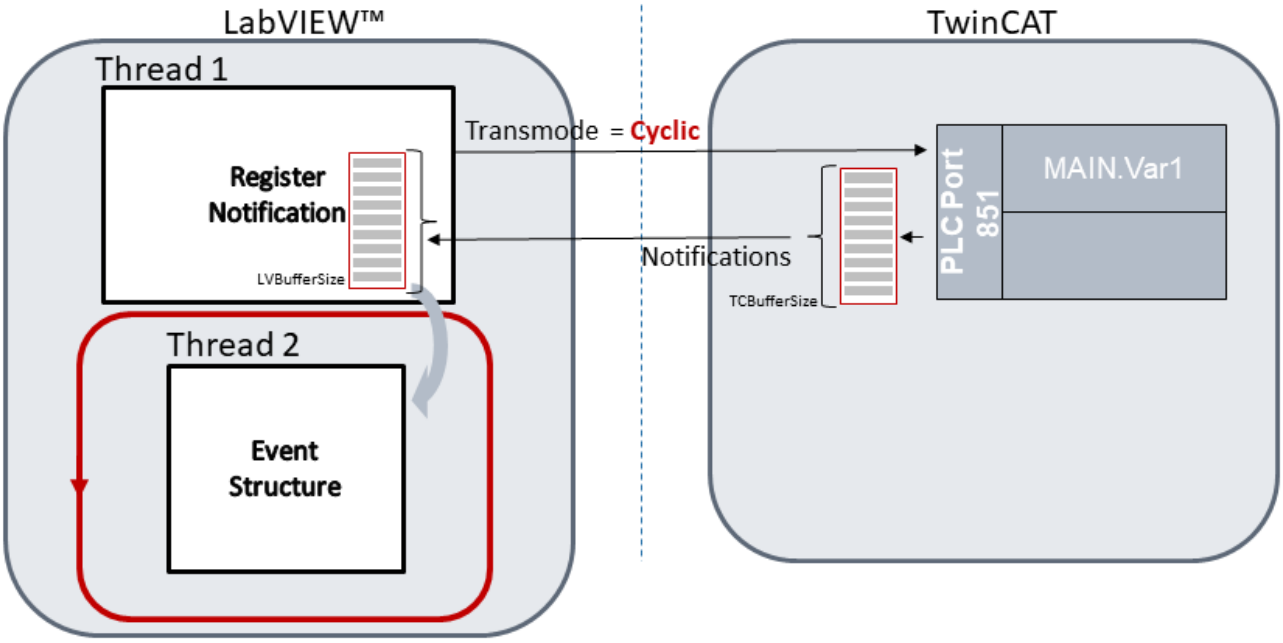
Durch dieses Vorgehen werden seltener LabVIEW™-Events erzeugt als in der E-Notification Single Variante. Durch das Hineinreicheichen von größeren Datenpaketen nach LabVIEW™ können z. B. Verarbeitungsprozesse effizienter gestaltet werden, sodass insgesamt ein höherer Datendurchsatz erreicht wird.

Puffergröße

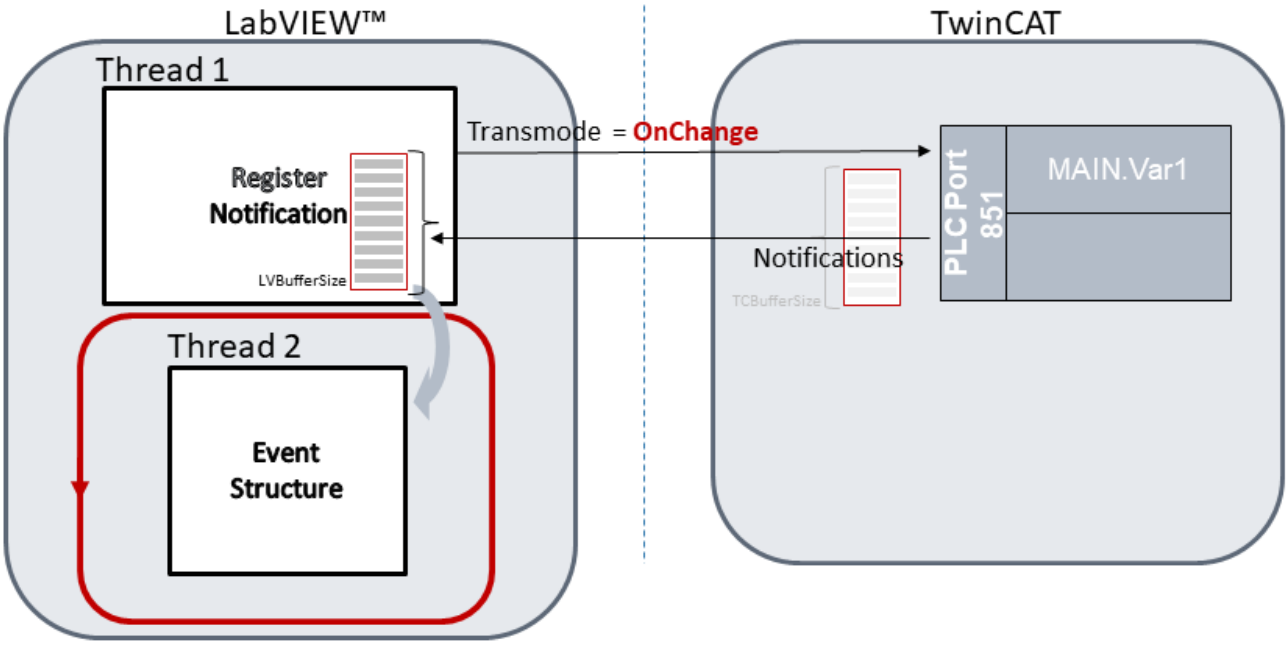
Das TwinCAT 3 Interface für LabVIEW™ reserviert beim Erzeugen von LVBuffer noch zusätzliche 10% Reserve-Pufferspeicher.

Auch für diese Betriebsart sind der*Transmode* und die *TCBufferSize* die entscheidenden Parameter:

Beim *Transmode* „**Cyclic**“ wird der TCBuffer und der LVBuffer genutzt.



Beim *Transmode* „**OnChange**“ wird der TCBuffer **nicht** genutzt. Hier werden die einzelnen Notifications ohne Pufferung direkt an LVBuffer übertragen.

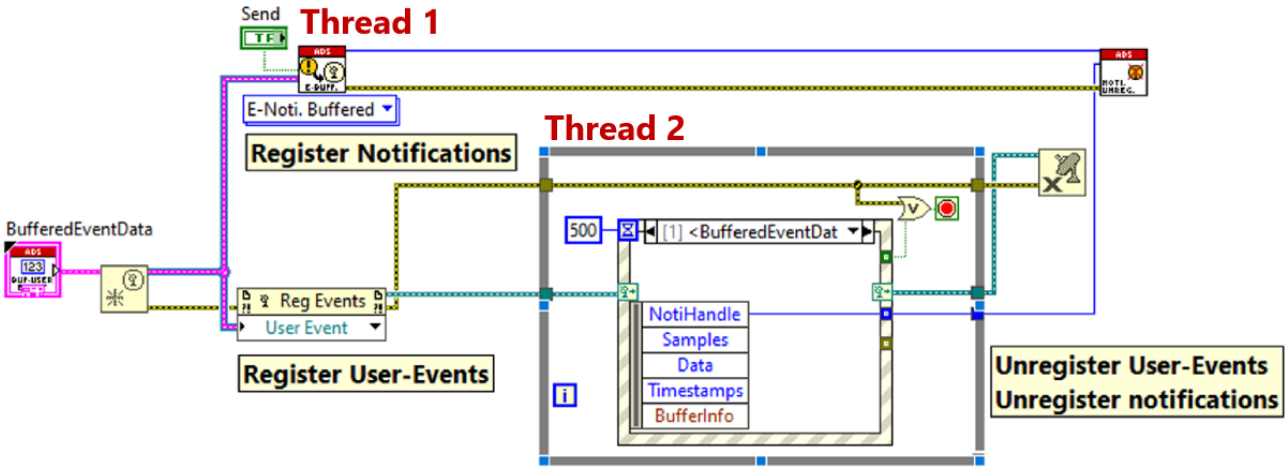


In LabVIEW™ gleicht das Blockdiagramm der Betriebsart E-Notification Single. Hier besitzt jede empfangene Notification ein zusätzliches Feld, die *BufferInfo*. Die BufferInfo bietet zusätzliche Informationen bezüglich des LVBuffer:

- Previous Overflow Samples: Beschreibt einen Counter, der angibt, wie oft der angelegte Puffer nicht ausgereicht hat. Beispiel: Der Puffer ist vom Nutzer mit 50 Samples angelegt worden. Wie oben beschrieben, wird intern 10% mehr Speicherplatz angelegt, also 55 Samples. Pro Notification können durch die TwinCAT-seitige Pufferung mehrere Samples übertragen werden. Ist der Puffer in LabVIEW™ bereits mit 45 Samples belegt, und kommt ein weiteres Paket mit 10 Samples, wird der

maximale Puffer nicht überschritten und der Counter inkrementiert nicht. Würde das ADS-Paket hingegen mehr als 10 Samples beinhalten, würden Daten verloren gehen, da der Puffer nicht ausreicht. Der Counter inkrementiert entsprechend um Eins.

- Missing Samples: Beschreibt die Differenz zwischen empfangenen und erwarteten Samples. Beispiel: Der LabVIEW™-seitige Puffer hat 50 Samples, also werden 50 Samples erwartet. Werden aber nur 48 Samples empfangen, ist Missing Samples entsprechend Zwei.
- Buffer Usage: Puffer Verbrauch in Prozent %



Beispiel in LabVIEW™: Read Notification-Event Buffered [▶ 108], Read Notification-Event Multiple [▶ 108]

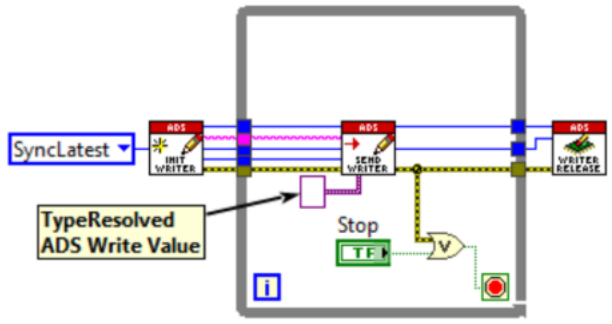
6.2.4 Daten kontinuierlich Schreiben

Beim kontinuierlichen Schreiben sendet LabVIEW™ Schreib-Anfragen an TwinCAT. Das Senden von Anfragen erfolgt mit einem Polling-Zyklus.

Schreiben mit Polling-Zyklus

Das Schreiben mit dem ADS-Synchron-Block nutzt sogenanntes kontinuierliches Polling. Hier werden die Low-Level [▶ 85] Writer Blocks genutzt. Der ADS-Writer wird nur einmal initialisiert und sendet mit jedem Zyklus (Schleifendurchlauf) ein neues *TypeResolved*-Datenpaket an den ADS-Server. Mit jedem Zyklus wird eine neue Anfrage an den Server geschickt und auf eine Quittierung gewartet. Nach Erreichen der Bedingung zum Verlassen der Schleife wird der ADS-Writer freigegeben. Das Bild unten zeigt die vollständige Vorgehensweise in LabVIEW™.

Diese Art des Schreibens von LabVIEW™ nach TwinCAT kann z. B. genutzt werden, um in kurzen Zykluszeiten Werte direkt auf eine Ausgangsklemme zu schreiben.



Beispiel in LabVIEW™: Grundlegende Beispiele [▶ 109]

6.3 Type Resolving

Das TwinCAT 3 Interface for LabVIEW ™ bietet **TypeResolver,** um den ADS-Datenstrom aus einem LabVIEW™-Datentyp in einen TC3-Datentyp oder umgekehrt zu parsen. Es wird zuerst ein Vergleich zwischen dem LabVIEW™-Datentyp und dem TC3-Datentyp gemacht. Wenn die Datentypen gleich sind, wird der ADS-Datenstrom passend geparst und kopiert. Der TypeResolver unterstützt die folgenden Betriebsarten:

1. ADS-Datenstrom aus TC3 auflösen.
2. ADS-Datenstrom für TC3 auflösen.

ADS-Datenstrom aus TC3 auflösen

Der Block **From TC** initialisiert den TypeResolver, parst und konvertiert die ADS-Daten, die mit ADS Read von TwinCAT gelesen worden sind, in den LabVIEW™-Datentyp *Variant*. Der Block Type Release gibt den TypeResolver aus dem Speicher wieder frei. Das Konvertieren von *Variant* in den passenden LabVIEW™-Datentyp erfolgt durch das Variant-to-Data VI.

Die folgende Grafik zeigt, wie der TypeResolver aus den Low Level VIs [▶ 91] zusammengesetzt ist. In diesem Beispiel gehen wir von einer Struktur in TwinCAT aus, welche aus drei Elementen besteht.

Definition der Struktur:

```
TYPE ST_toLabVIEW :
STRUCT
    TCSignedWord  : INT;
    TCUnsignedInt : UINT;
    TCStringArray : ARRAY [1..5] OF STRING;
END_STRUCT
END_TYPE
```

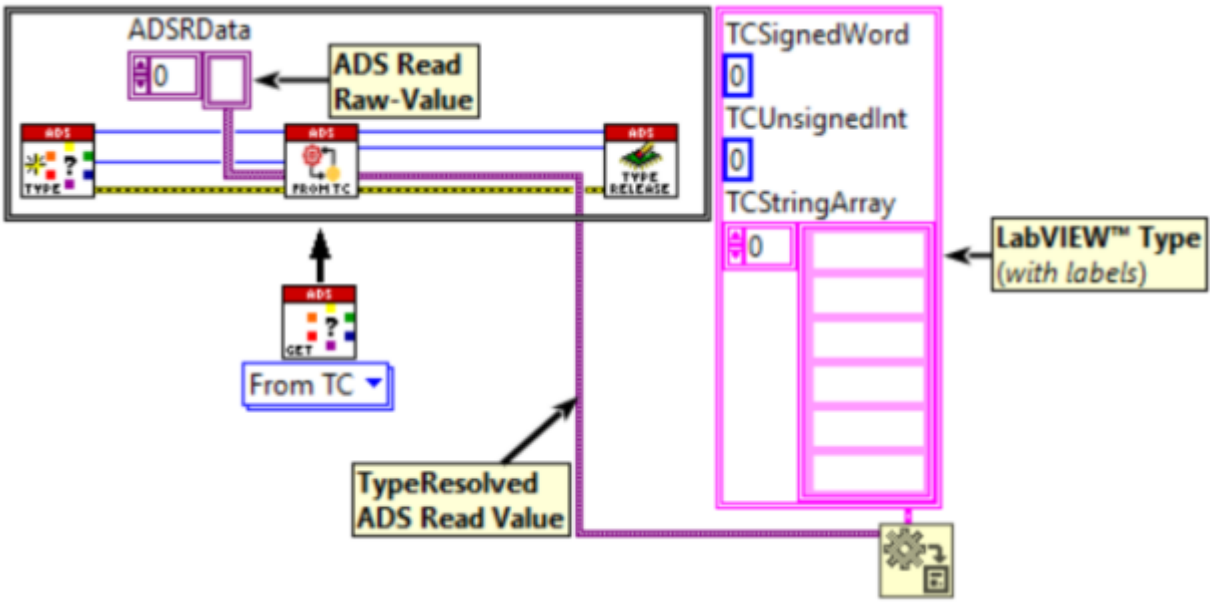
Der TC3-Datentyp und der LabVIEW™-Datentyp müssen für den Umsetzungsvorgang übereinstimmen. Im Anhang finden Sie dazu eine Übersetzungstabelle [▶ 118] als Übersicht. Auf der rechten Seite im Blockdiagramm ist ein LabVIEW™-Container abgebildet, welcher die oben genannte Struktur widerspiegelt.

Aufbau des LabVIEW™-Containers (nicht initialisiert, nur definiert):

- I16
- U16
- TCStringArray ist 1..n Elemente und vom Typ String (wird in LabVIEW™ nicht näher spezifiziert)

Das Kopieren von Daten kann aus folgenden Gründen fehlschlagen:

- Der LabVIEW™-Container ist nicht in der korrekten Reihenfolge in seinen Elementen aufgebaut.
- Die Einträge der Struktur und des Containers haben nicht denselben Datentyp.



ADS-Datenstrom für TC3 auflösen

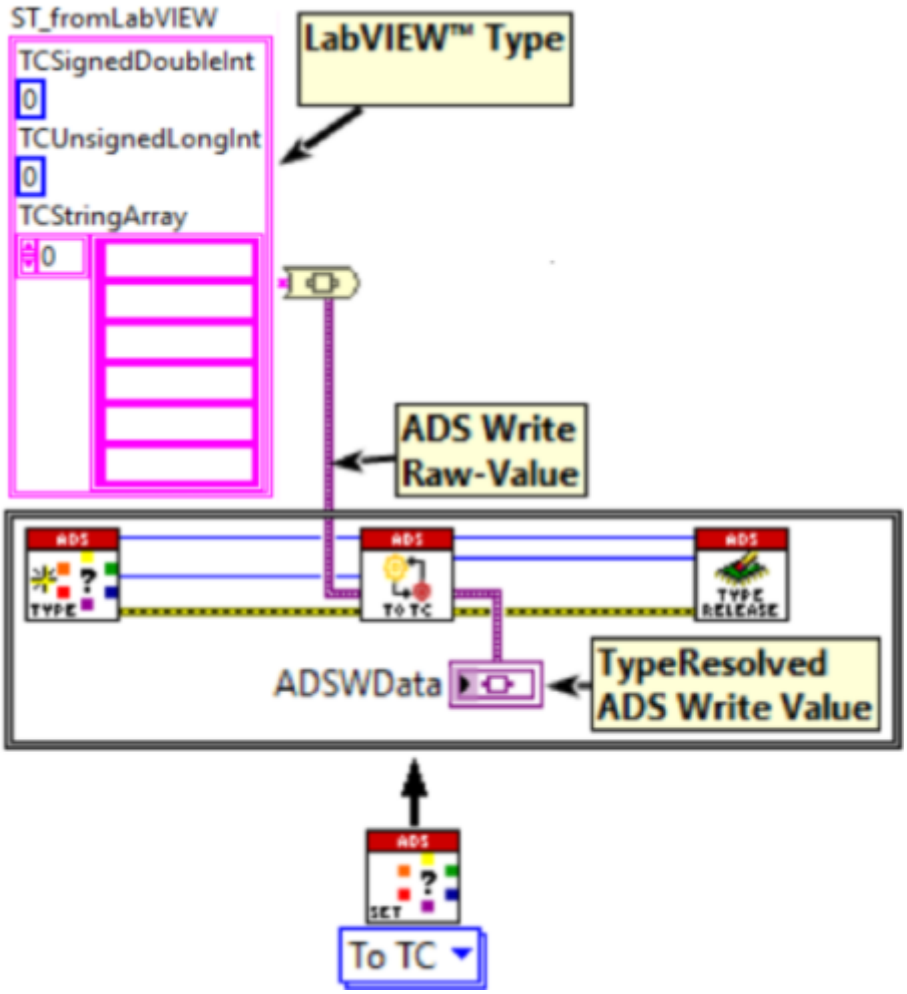
Der Block *To TC* initialisiert den TypeResolver, vergleicht den LabVIEW™-Datentyp mit dem TC3-Datentyp und konvertiert die LabVIEW™-Daten in ADS-Rohdaten. Am Ende gibt der Block den TypeResolver aus dem Speicher frei. Die konvertierten ADS-Daten können danach direkt an einen ADS Write [▶ 73]-Block weitergegeben werden.

Die unten stehende Grafik zeigt ein ähnliches Szenario wie zuvor die Beschreibung *From TC*. Hier werden die Daten von einem LabVIEW™-Datentyp zu einem TC3-Datentyp konvertiert. Die Konvertierung kann aus folgenden Gründen fehlschlagen:

- Der LabVIEW™-Container ist nicht in der korrekten Reihenfolge in seinen Elementen aufgebaut.
- Die Einträge der Struktur und des Containers haben nicht denselben Datentyp.
- Die Elemente in der Struktur sind nicht initialisiert.

Initialisierung des LabVIEW™-Datentyps

Beim Konvertieren vom LabVIEW™- in den TC3-Datentyp muss der LabVIEW™-Datentyp vorinitialisiert sein. Beim komplexen Datentyp, wie einer Struktur, muss der komplette Datentyp initialisiert sein. Beim Array müssen die einzelnen Elemente initialisiert sein.



Für obige Grafik ist zum Beispiel folgende TwinCAT-Struktur in der PLC als Ziel zum Schreiben denkbar:

```
TYPE ST_fromLabVIEW :
STRUCT
    TCSignedDoubleInt  : DINT;
    TCUnsignedLongInt : ULINT;
    TCStringArray : ARRAY [1..6] OF STRING(80);
END_STRUCT
END_TYPE
```

Der passende initialisierte LabVIEW™-Container ist dann folgendermaßen aufgebaut:

- I32

- U64
- String-Array mit 6 Elementen (es dürfen hier jeweils 80 Character pro Array-Element genutzt werden)

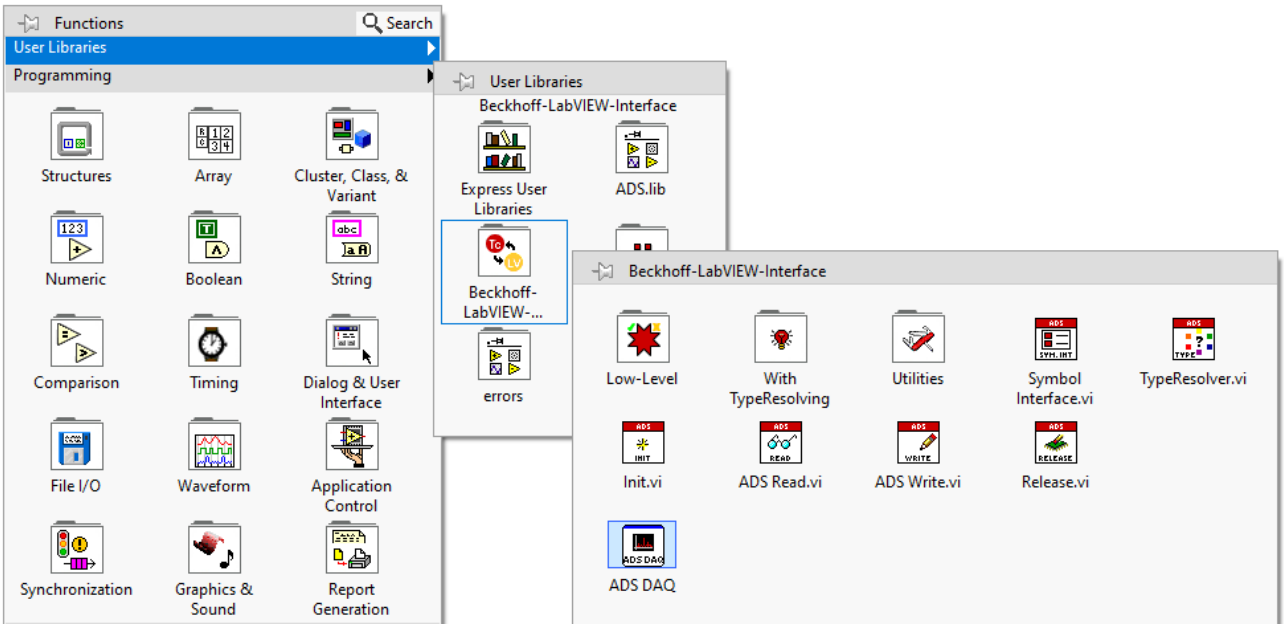
Automatische Typ-Generierung

Die manuelle Erstellung von passenden Datentypen in LabVIEW™ ist unter Umständen zeitaufwendig und fehleranfällig. Das TwinCAT Interface for LabVIEW™ bietet die Möglichkeit der automatischen Typ-Generierung. Nutzen Sie dazu den TypeResolver [▶ 91] und den Utilities [▶ 77] Wrapper Block. Im Kapitel Beispiele [▶ 104] finden Sie verschiedene Varianten, wie Sie diese VIs effektiv einsetzen können: Beispiel Type-Resolver [▶ 77].

7 LabVIEW™-VIs

Das TwinCAT 3 Interface for LabVIEW™ bietet Bedienelemente und VIs für die Nutzung in LabVIEW™.

Die VIs befinden sich im Blockdiagramm in der *functions palette*: **Functions > User Libraries > Beckhoff-LabVIEW-Interface**.



Der Hauptordner beinhaltet die Grund-VIs, die zum Aufbauen eines Programms zum Lesen über ADS, zum Schreiben über ADS, zum TypeResolving und zum Freigeben des ADS-Clients genutzt werden können. Zusätzlich beinhaltet der Hauptordner noch die Unterordner: Low-Level, With TypeResolving und Utilities.

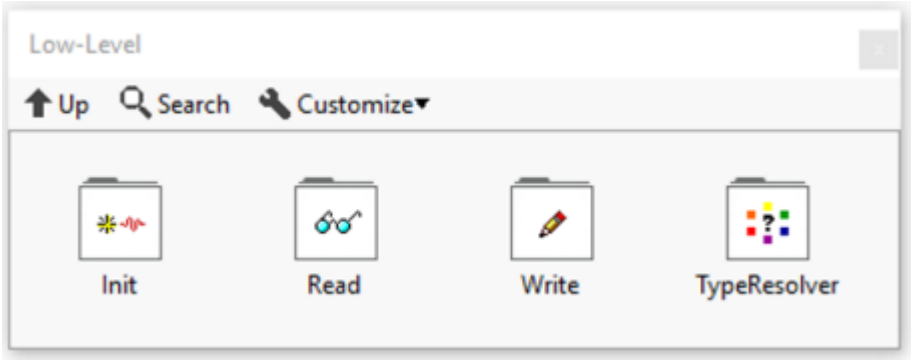
Low-Level

Der *Low-Level*-Unterordner enthält Low-Level-VIs. Die Low-Level-VIs funktionieren nach dem gleichen Prinzip wie die Grund-VI. Die Low-Level-VIs bringen etwas mehr Programmierungsaufwand mit sich, sind jedoch, im Vergleich zu den Grund-VIs, performanter (im Sinne des Datendurchsatzes) und bieten mehr Freiheit in der Realisierung komplexer Programme. Daten kontinuierlich lesen [▶ 31] ist ein Beispiel, welches die Low-Level-VIs für schnelles Lesen über ADS nutzt. Daten kontinuierlich Schreiben [▶ 37] ist ein ähnliches Beispiel. Nicht nur das Lesen und Schreiben können in dieser Art und Weise beschleunigt werden, sondern auch das TypeResolving, siehe bspw. Continous Read [▶ 107].

Die Tabelle beschreibt die Unterordner und deren Inhalt und Funktion:

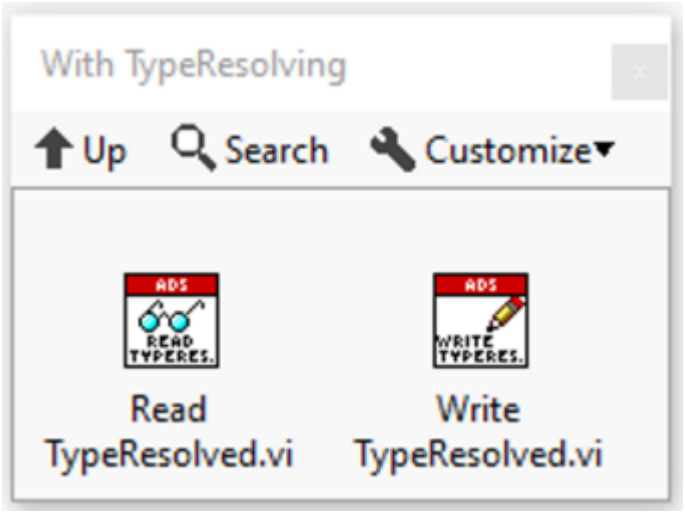
Unterordner	VIs	Funktion
Init [▶ 86]	Base Init	Initialisiert den ADS-Client.
	Get List of ReadWrite Symbols	Erstellt eine Liste von ADS-Lese- und Schreibe-Symbolen.
	Get List of Registered Targets	Erstellt eine Liste von eingetragenen ADS-Zielsystemen.
Read [▶ 87]	Init Reader	Initialisiert den ADS-Reader.
	Send Reader-Request	Sendet eine Anfrage an den ADS-Server.
	Register Notification	Meldet die Notification am ADS-Server an.
	TryReadData	Überprüft die Antwort vom Server und liest den Datenstrom.
	Release Reader	Gibt den Reader aus dem Speicher frei.
Write [▶ 89]	Init Writer	Initialisiert den ADS-Writer.
	Send Writer-Request	Sendet eine Anfrage an den ADS-Server.
	CheckWriteStatus	Überprüft für die Antwort vom ADS-Server, ob das Datenpaket empfangen wurde.

Unterordner	VIs	Funktion
	Release Writer	Gibt den Writer aus dem Speicher frei.
TypeResolver [▶ 91]	Init Type	Initialisiert den TypeResolver.
	Resolve From TC Type	Konvertiert den TC3-Datentyp in einen LabVIEW™-Datentyp Variant.
	Resolve To TC Type	Konvertiert den LabVIEW™-Datentyp Variant in TC3-Datentyp.
	Release Type	Gibt den TypeResolver aus dem Speicher frei.



With TypeResolving

Der *With TypeResolving*-Unterordner enthält zwei VIs für das Lesen und Schreiben über ADS mit integriertem TypeResolver Block.

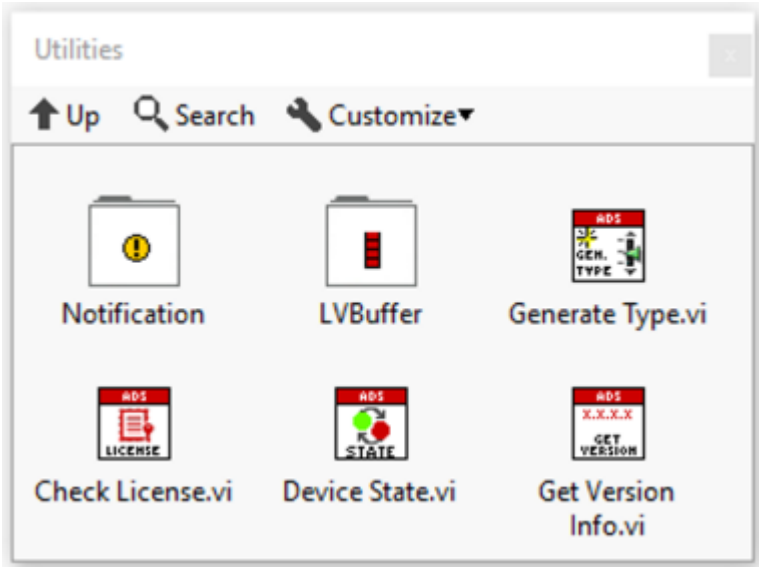


Utilities

Der *Utilities*-Unterordner enthält zusätzliche VIs für folgende Zwecke:

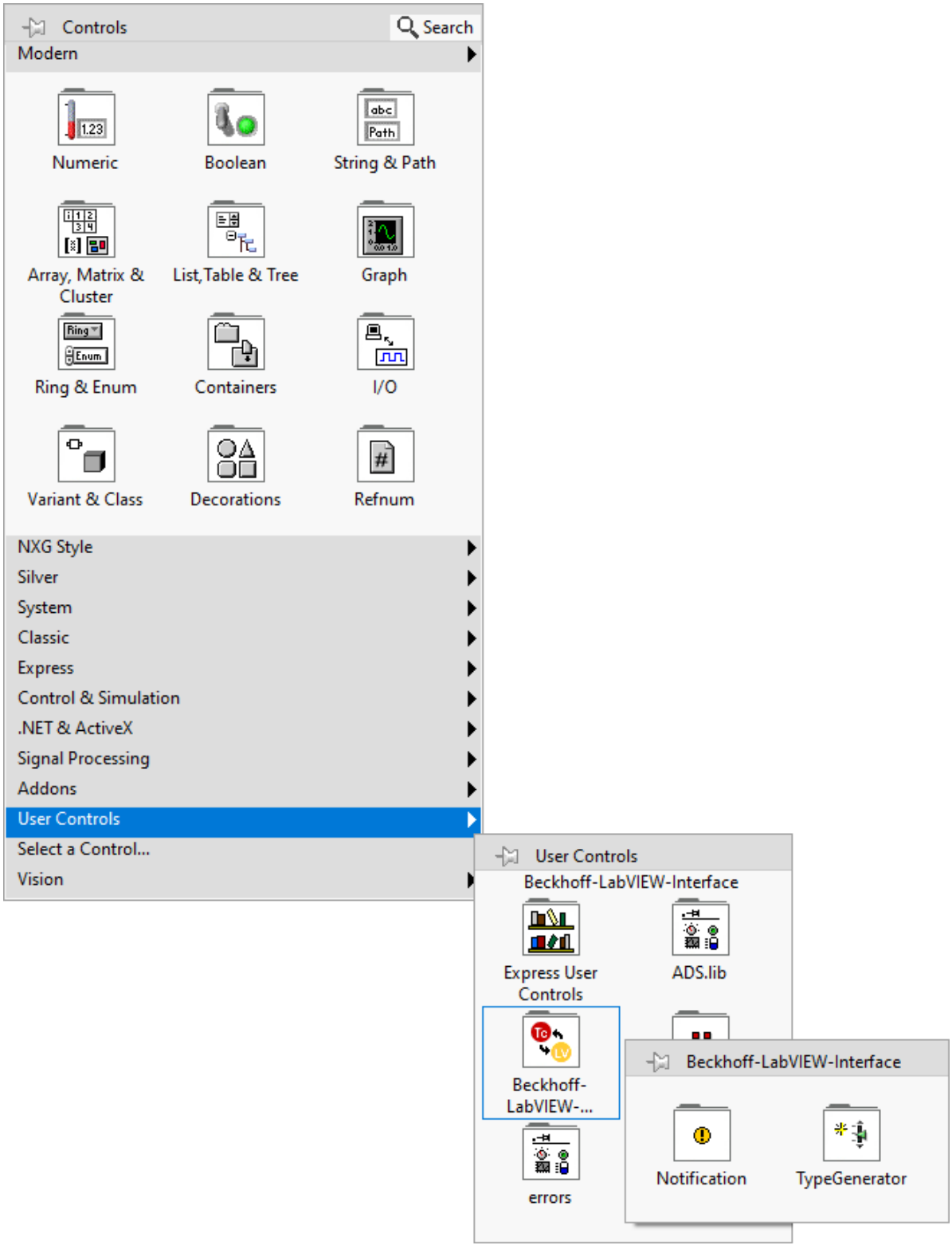
Unterordner	VIs	Funktion
Notification [▶ 79]	ADS To LabVIEW Timestamp	Konvertiert ADS-Zeitstempel in LabVIEW™-Zeitstempel.
	Notification Data To Variant Array	Baut ein Array von LabVIEW™ *Variant* aus dem Notification Datenstrom.
	Stop Notification	Stoppt die ADS-Notifications.
	Start Notification	Startet die ADS-Notifications.
	Unregister Notification	Meldet die Notification am Server ab.
	Check License	Überprüft den Lizenzzustand auf einem vorgegebenen Zielsystem.

Unterordner	VIs	Funktion
	Set Device State, Get Device State	Liest oder ändert den Zustand eines ADS-Device.
	Get Version Info	Gibt Informationen bezüglich der Produktversion.



Bedienelemente

Die Bedienelementen befinden sich im Frontpanel in der *controls palette* unter: **User Controls > Beckhoff-LabVIEW-Interface**.



Der Unterordner „Notification“ enthält Bedienelemente, die beim Initialisieren des LabVIEW™-Ereignis für ADS-Notifications benötigt werden.

Des Weiteren beinhaltet der Unterordner „TypeGenerator“ TypeGenerator [▶ 93]-Klassen-Objekte, um TwinCAT-Typen in LabVIEW™-Typen zu konvertieren (siehe dazu Grundlegende Beispiele [▶ 110]).

Folgende Tabelle beschreibt die Funktion der Bedienelemente:

Unterordner	Controls	Funktion
Notification	Single User-Event Data	Bedienelement für das Initialisieren eines LabVIEW™-Events für **Single** ADS-Notifications.

Unterordner	Controls	Funktion
	Buffered User-Event Data	Bedienelement für das Initialisieren eines LabVIEW™-Events für **Buffered** ADS-Notifications.
TypeGenerator	CBase	Klassen-Objekt; Basis Klasse des TypeGenerator
	CBool	Klassen-Objekt; Boolesche Klasse des TypeGenerator.
	CNumeric	Klassen-Objekt; Numerische Klasse des TypeGenerator.
	CString	Klassen-Objekt; LabVIEW™ Zeichenkette Klasse des TypeGenerator.
	CArray	Klassen-Objekt; Array Klasse des TypeGenerator.
	CTimestamp	Klassen-Objekt; LabVIEW™ Zeitstempel Klasse des TypeGenerator.
	CCluster	Klassen-Objekt; LabVIEW™ Cluster Klasse des TypeGenerator.

7.1 ADS DAQ

Das ADS DAQ (**D**ata **Acq**uisition) VI ist ein LabVIEW™ Express VI zum einfachen Konfigurieren von Messaufgaben mit TwinCAT 3, d. h. mit diesem VI können Sie lesend auf TwinCAT-Laufzeiten zugreifen.

Das User-Interface des ADS DAQ VIs leitet Sie Schritt für Schritt durch die Konfiguration Ihrer Messaufgabe:

- Auswahl der zu lesenden Datenpunkte (ADS-Symbole)
- Konfiguration des Lese-Modus (ADS-Notification)
- Konfiguration des Typ-Generators
- Konfiguration von Start- und Endbedingung der Messaufgabe

Nach dem Platzieren der ADS DAQ Instanz im LabVIEW™-Blockdiagramm oder per Doppelklick, öffnet sich das Konfigurationsfenster. Mit Hilfe der nachfolgend beschriebenen Auswahlfenster können die Konfigurationen vorgenommen werden. Nach Fertigstellung der Konfiguration erstellt die Instanz alle notwendigen Ressourcen für das Lesen der Daten.

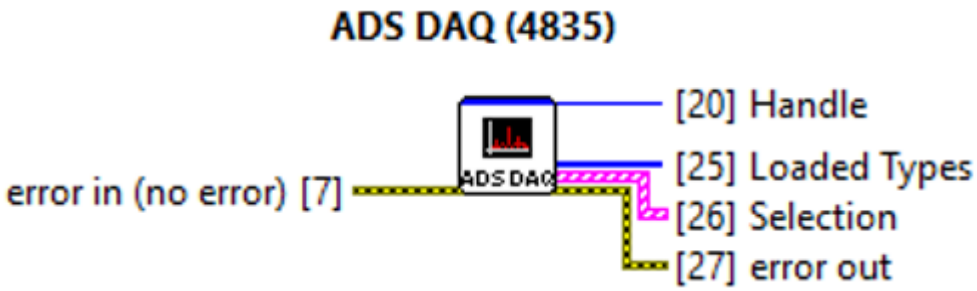
VI vor Nutzung des ADS DAQ VI speichern

Das ADS DAQ VI speichert die Konfiguration einer Instanz im Pfad des aktuellen Projekts/VI. Daher ist es erforderlich, dass das Projekt/VI zuvor gespeichert wird.

ADS DAQ VI beschleunigt öffnen

✓ Die Bibliothek muss vorkompiliert sein.

1. Öffnen Sie in den LabVIEW™ Einstellungen unter **Tools > Advanced** die Einstellungen für „Mass Compile“.
2. Wählen Sie den Ordner der Bibliothek des TwinCAT 3 Interfaces for LabVIEW™ aus, z. B. *C:\Program Files\ National Instruments\LabVIEW 2023\user.lib\Beckhoff-LabVIEW-Interface*.
3. Starten Sie „Mass Compile“.



Ausgang	Bedeutung
[20] Handle	Handle auf den ADS-Client
[25] Loaded Types	Ein Array von LabVIEW™ Enums: • Beschreibt, welche Datentypen generiert wurden.

Ausgang	Bedeutung
[26] Selection	LabVIEW™-Cluster bestehend aus zwei Elementen: • SymbolName: Der Name des ADS-Symbols • Notification Mode: LabVIEW™ Enum ◦ Single TypeResolved Queue: Liest nur ein Sample als Notification und fügt das Sample der LabVIEW™ Queue hinzu (nur im LabVIEW™ 32-Bit). ◦ Single TypeResolved Control: Liest nur ein Sample als Notification und schreibt das Sample in das LabVIEW™ Anzeigeelement (nur im LabVIEW™ 32-Bit). ◦ Buffered TypeResolved Queue: Liest eine Anzahl von Samples, wie beschrieben von **LVBufferSize**, und fügt die Samples der LabVIEW™ Queue hinzu. ◦ Buffered TypeResolved Control: Liest eine Anzahl von Samples, wie beschrieben von **LVBufferSize**, und schreibt die Samples in das LabVIEW™ Anzeigeelement.

Anzahl von Symbolen

Aktuell unterstützt der ADS DAQ Block das Lesen von maximal 10 ADS-Symbole pro Instanz. Nutzen Sie mehrere Instanzen oder das Upgrade ADS FlexDAQ [▶ 54], wenn Sie mehr als 10 ADS-Symbole lesen möchten.

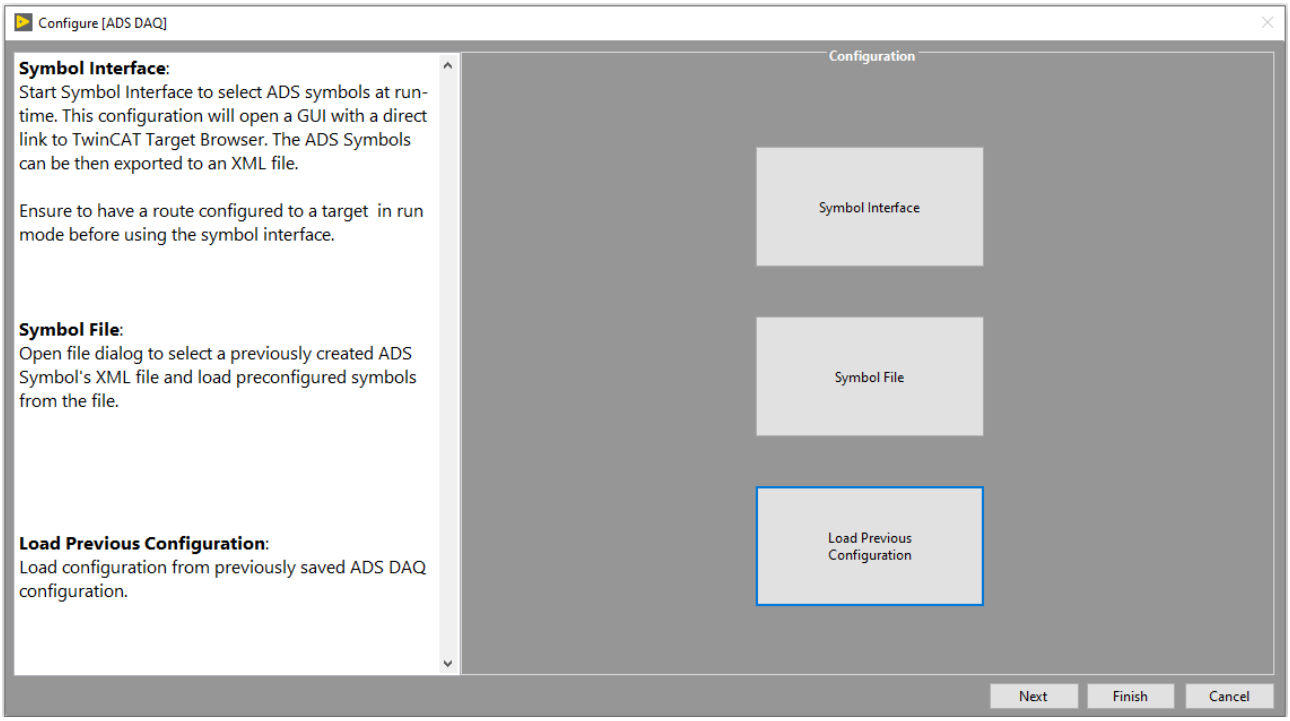
Generierung von TwinCAT-3-Datentypen

Um alle Notification Modi zu unterstützen, werden alle generierten Typen in Arrays umgewandelt. So werden Notifications Buffered ebenso wie Notifications Single in gleicher Art und Weise unterstützt.

Symbol-Auswahl-Fenster

In diesem Fenster werden die ADS-Symbole selektiert, die mit der ADS DAQ Instanz gelesen werden sollen. Das Fenster bietet drei verschiedene Wege, um die ADS-Symbole auszuwählen:

1. Symbol Interface [▶ 65]**:** Öffnet ein zusätzlich graphisches User-Interface zum Browsen von ADS-Symbolen.
   - Browsen Sie in die verbundenen Targets.
   - Selektieren Sie die gewünschten ADS-Symbole und ziehen Sie diese per Drag-and-drop in das rechte Feld.
   - Optional können Sie im rechten Feld die ausgewählten ADS-Symbole als Liste exportieren und als XML-Datei speichern.
2. **Symbol File:** Öffnet ein LabVIEW™-Datei-Dialogfeld, um die ADS-Symbole anhand einer aus dem Symbol Interface exportierten XML-Datei einzulesen.
   - Liest nur die Symbole ein, beinhaltet jedoch nicht alle weiteren Einstellungen des ADS DAQ VI aus den weiteren Konfigurationsansichten.
3. **Load Previous Configuration:** Lädt die letzte Konfiguration (falls existent), mit der die ADS DAQ Instanz bereits konfiguriert wurde.
   - Die Konfiguration öffnet sich immer als leere Konfiguration. Wenn Sie die existierende Konfiguration anpassen möchten, wählen Sie **Load Previous Configuration**.



Nach Auswahl der ADS-Symbole, wählen Sie **Next**.

Wählen Sie zu jeder Zeit im Dialog **Finish**, um die Konfiguration zu speichern und zu schließen.

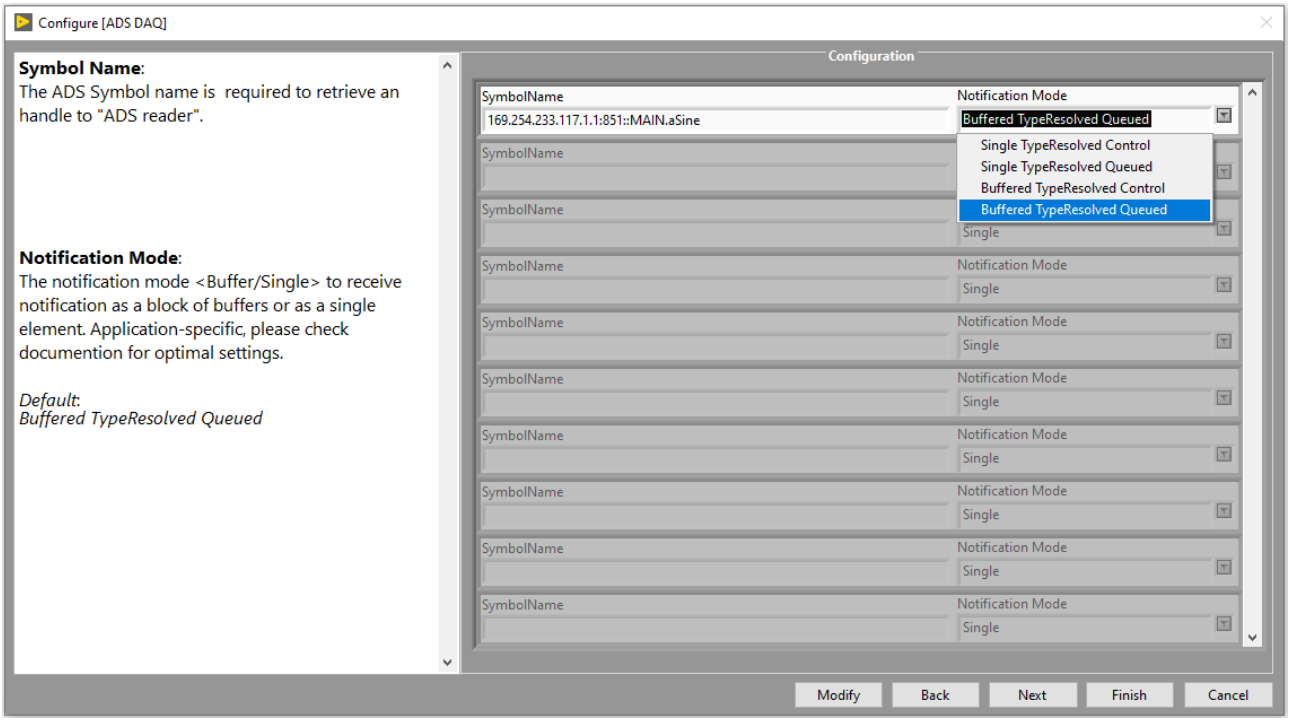
Notification-Auswahl-Fenster

In diesem Fenster wird der Notification Mode bei einzelnen ausgewählten ADS-Symbolen spezifiziert. Der Notification Mode beschreibt auf einer Seite die Art und Weise des Lesens (*single/buffered*) und auf der anderen Seite das Darstellen der gelesenen Daten in LabVIEW™ (*control/queue*). Default-Einstellung ist *Buffered TypeResolved Queued*.

Folgende Tabelle beschreibt die unterschiedlichen Eigenschaften:

	Single TypeResolved	Buffered TypeResolved
Control (Nur empfehlenswert, wenn die Daten nicht verarbeitet oder gespeichert werden müssen.)	Die einzelnen Notifications werden direkt in einem LabVIEW™ Control dargestellt.	Die Notifications werden zuerst in eine Puffer-Zwischenschicht geschrieben und in den LabVIEW™-Prozess übergeben, wenn der Puffer gefüllt ist. Die Daten werden in einem LabVIEW™ Control dargestellt.
Queued (Empfehlenswert, wenn Daten verarbeitet, gespeichert, … werden sollen.)	Die einzelnen Notifications werden direkt in eine LabVIEW™ Queue eingefügt.	Die Notifications werden zuerst in eine Puffer-Zwischenschicht geschrieben und in den LabVIEW™-Prozess übergeben, wenn der Puffer gefüllt ist. Die Daten werden in eine LabVIEW™ Queue eingefügt.

Weitere Informationen zu Single- und Buffered-Mode, siehe Kommunikations-Modi [▶ 25] und Ereignisgesteuertes Lesen [▶ 33].

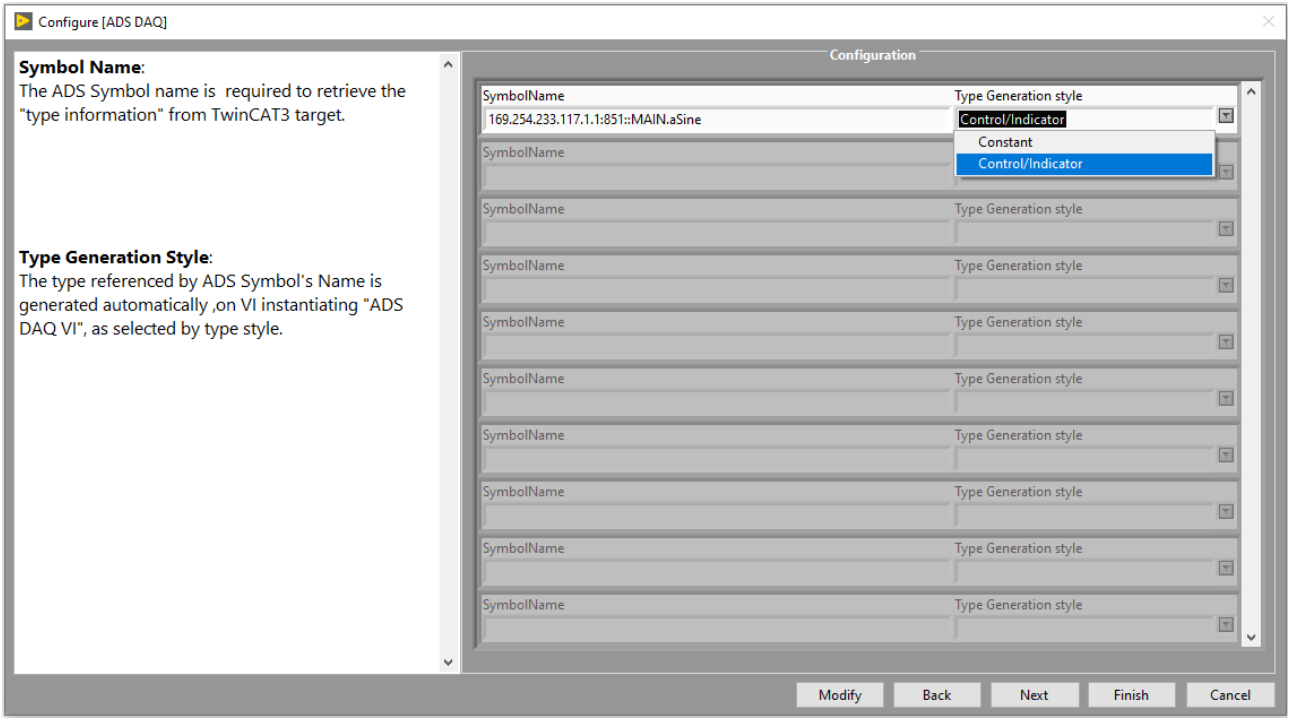


Wählen Sie **Modify,** um das Symbol Interface zu öffnen und Änderungen an den zu lesenden ADS-Symbolen vorzunehmen.

Wählen Sie **Next**, um zur nächsten Konfigurationsseite zu gelangen.

Typ-Generation-Auswahl-Fenster

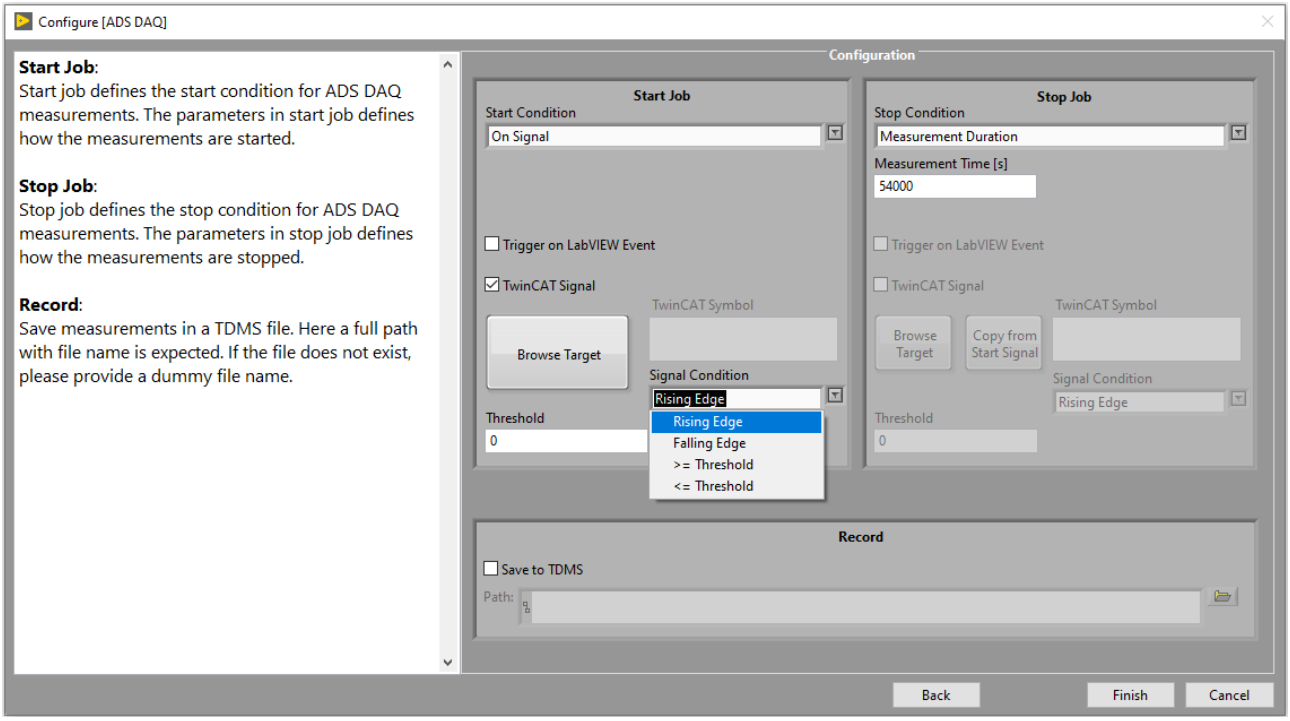
In diesem Fenster kann ausgewählt werden, ob der jeweilige Datentyp des zu lesenden ADS-Symbols als LabVIEW™-Konstante oder als Bedien-/Anzeigeelement generiert werden soll. Sowohl Konstanten als auch Bedien-/Anzeigeelemente werden im Blockdiagramm des VI automatisch generiert, wo ebenfalls die ADS DAQ Instanz liegt. Default-Einstellung ist *Control/Indicator*.



Mess-Job-Konfigurator

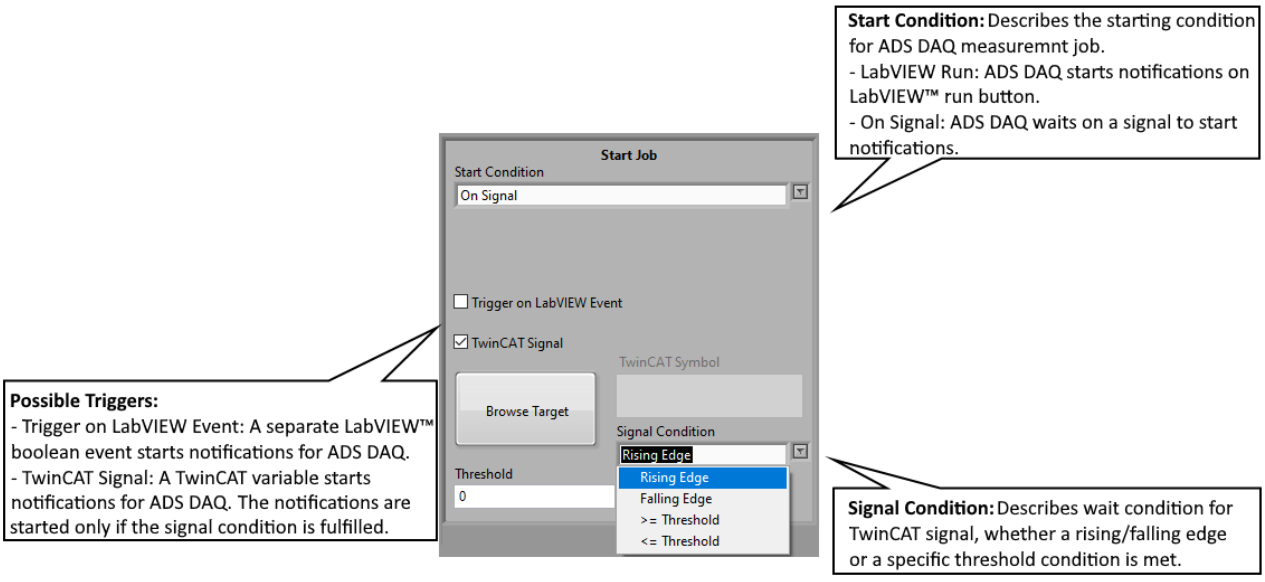
In diesem Fenster kann die ADS DAQ Instanz mit zusätzlichen Start-/Stop-/Aufnahme- Bedingungen konfiguriert werden.

Default-Einstellung ist bei *Start: LabVIEW run* und bei *Stop: LabVIEW Abort, kein Record*.



Start Job

Das Bedienelement *Start Job* konfiguriert das Starten des Lesevorgangs. Der Parameter *Start Condition* beschreibt die Art und Weise des Startens.



- **LabVIEW™ Run:** Die ADS-Notifications werden nach Starten des VI sowohl automatisch registriert als auch gestartet.

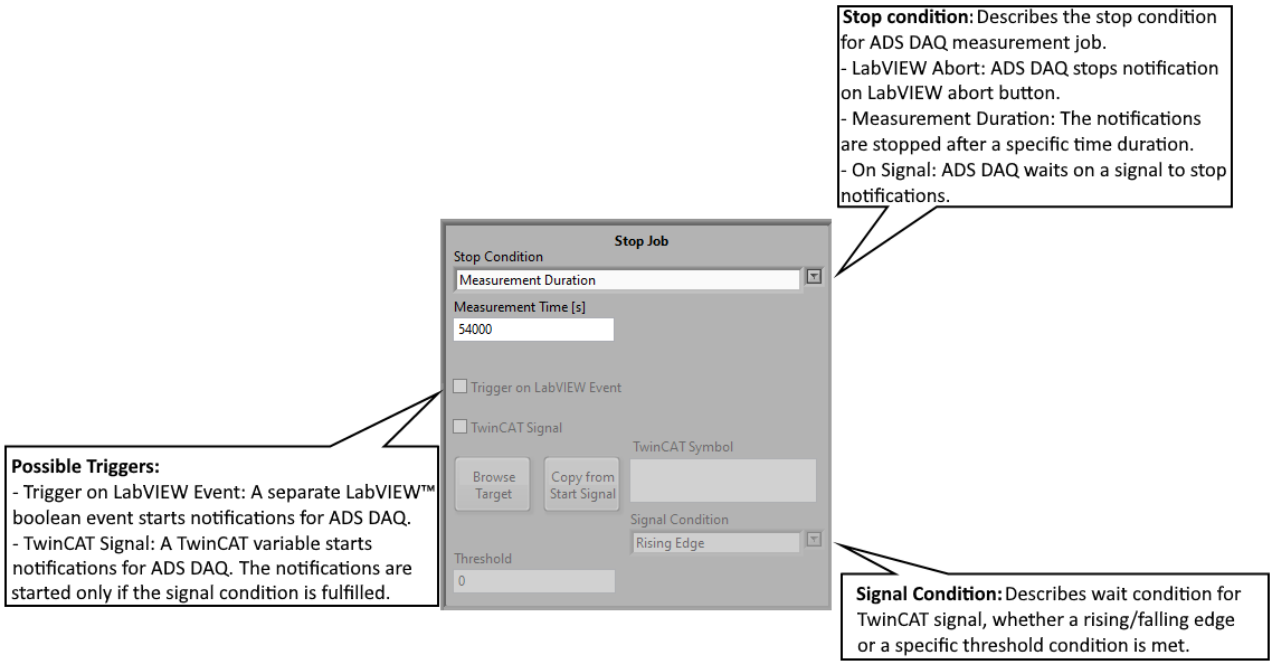
- **On Signal:** Die ADS-Notifications werden in diesem Fall nicht automatisch gestartet. Die ADS DAQ Instanz wartet auf einen bestimmten Trigger aus LabVIEW™ oder TwinCAT.
  - **Trigger on LabVIEW™ Event:** Mit dieser Auswahl generiert die ADS DAQ Instanz eine separate Event-Logik, um die ADS-Notifications von LabVIEW™ ausgehend zu starten.
  - **TwinCAT-Signal:** Die DAQ Instanz wird durch ein (zusätzliches) TwinCAT-Signal getriggert und startet daher die ADS-Notifications nicht automatisch. Die Schaltfläche *Browse Target* öffnet das Symbol Interface. Ziehen Sie ein ADS-Symbol nach rechts auf die „Lesen“-Fläche. Dabei sind alle Primärdatentypen erlaubt. Des Weiteren ist eine Signalbedingung zu definieren, auf die das ADS DAQ VI getriggert werden soll.

Die untenstehende Tabelle beschreibt den Übergang (*Letzter Wert* ➔ *Neuer Wert*) von verschiedenen TwinCAT-Typen mit der hier verwendeten Definition von Rising und Falling Edge.

TwinCAT-Typ	Rising Edge		Falling Edge	
	Letzter Wert	Neuer Wert	Letzter Wert	Neuer Wert
Boolescher Typ	0	1	1	0
Numerischer ganzzahliger Typ (i8, i16, i32, i64, u8, u16, u32, u64)	< Threshold	= Threshold	> Threshold	= Threshold
Numerischer rationaler Typ (float32, float64) Epsilon: 1.0 e^{-7}	< Threshold	= Threshold	> Threshold	= Threshold

Stop Job

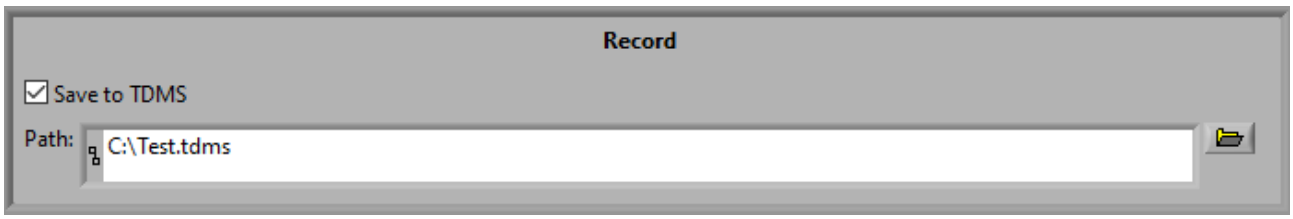
Das Bedienelement *Stop Job* konfiguriert das Stoppen von ADS-Notifications, also den Stopp des Lesevorgangs. Der Parameter *Stop Condition* beschreibt die Art und Weise des Stoppens.



- **LabVIEW™ Abort:** Die ADS-Notifications werden nach Stoppen des VIs automatisch unregistriert als auch gestoppt.
- **Measurement Duration:** Die ADS-Notifications stoppen automatisch nach Ablauf der Mess-Zeit.
- **On Signal:** Diese Option verhält sich identisch wie die Auswahl *On Signal* in **Start Job** (s.o.).
  - Wählen Sie **Copy from Start Signal**, um die Start-Bedingung von Start- zu Stop Job zu kopieren (gilt nur für das TwinCAT-Signal).

Record Job

Die Aufnahme von ADS DAQ Messdaten kann mit dem Bedienelement *Record Job* konfiguriert werden. Nach der Konfiguration und dem Klicken von **Finish**, generiert die ADS DAQ Instanz einen zusätzlichen Block To TDMS, um die empfangenen Daten unter dem angegebenen Dateinamen und Pfad als LabVIEW™-TDMS-Datei zu speichern.

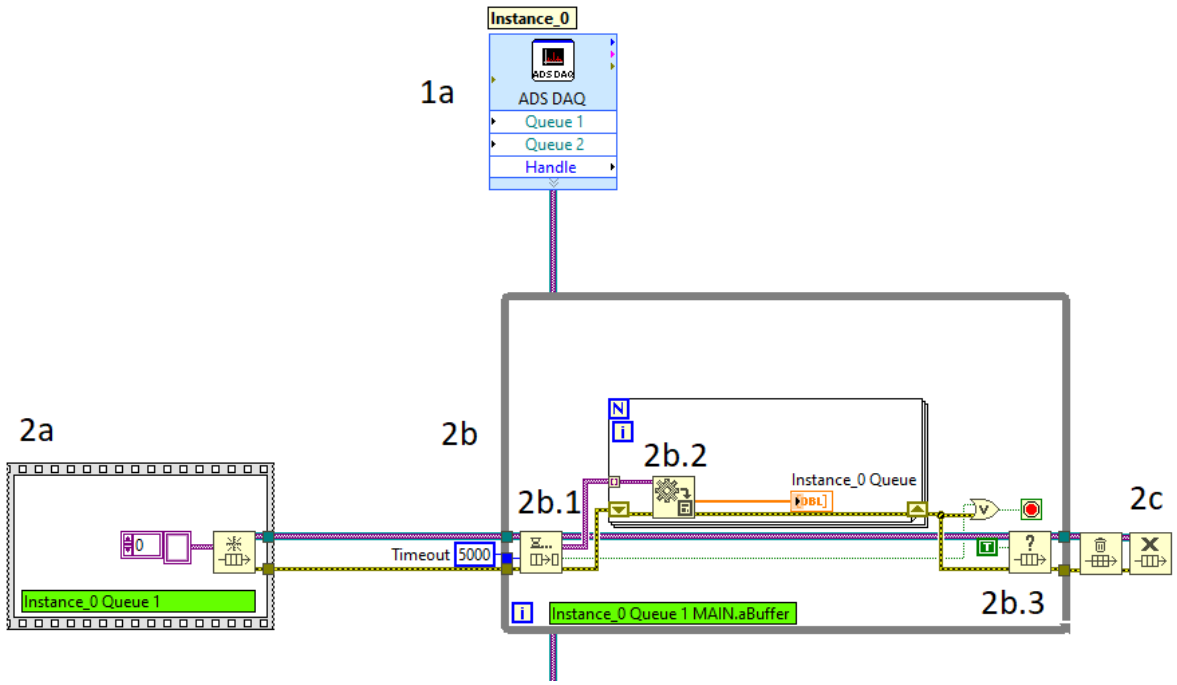


Wählen Sie **Finish**, um die Einstellungen zu speichern. Die automatische Code-Generierung für Ihre Konfiguration startet.

Automatisch generierter Code im Blockdiagramm

Nachfolgend werden exemplarisch zwei Varianten des automatisch generierten Codes erläutert.

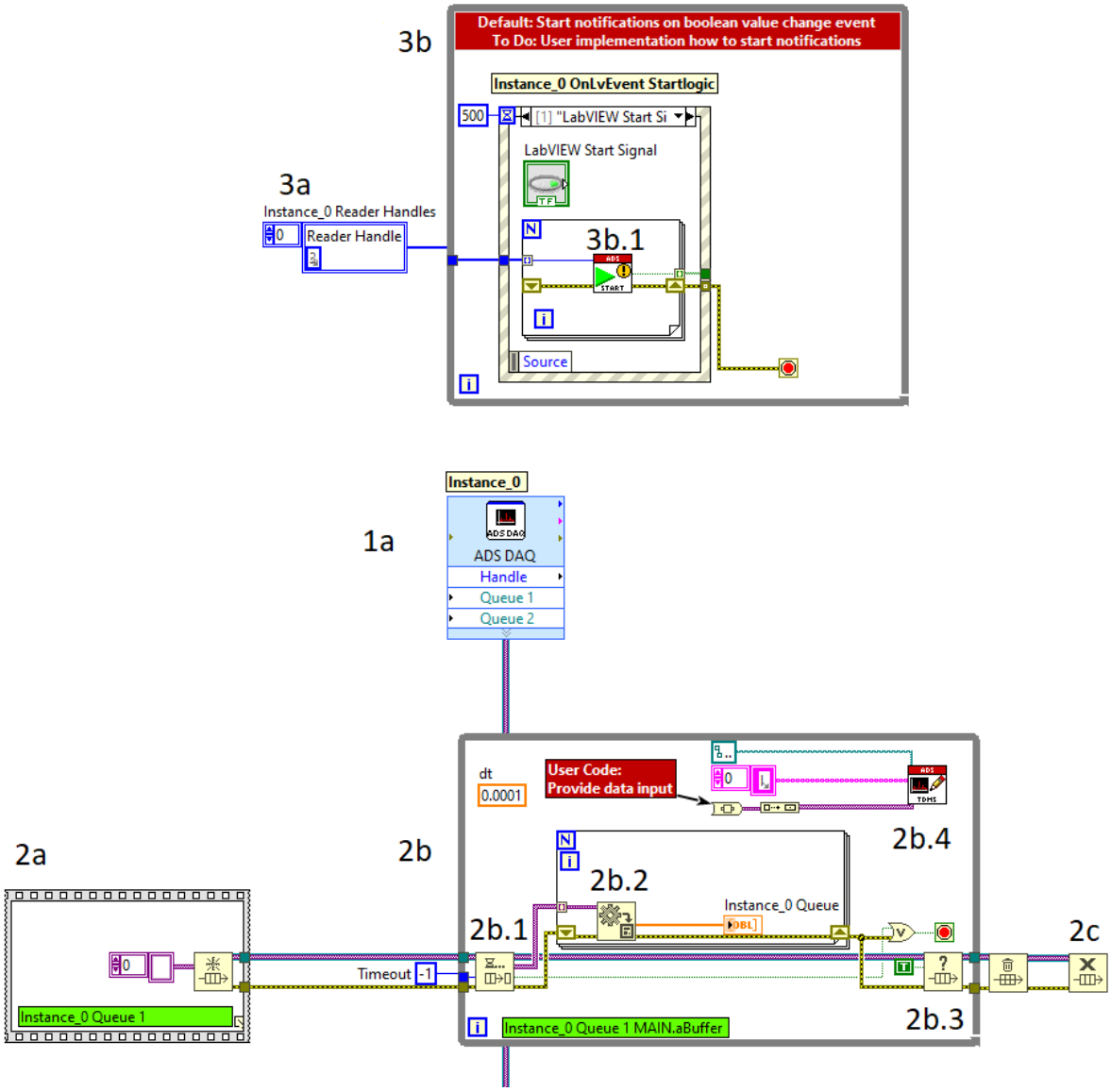
Im ersten Beispiel wird das ADS DAQ VI mit **Default-Einstellungen** generiert, d. h. Buffered Type Resolved Queue, Control/Indicator, Start mit LabVIEW™ Run, Stop mit LabVIEW™ Stop. Gelesen wird ein ADS-Symbol MAIN.aBuffer.



- **1a:** Von der Instanz des ADS DAQ VI geht ein Handle von Queue 1 auf die darunterliegenden Queue-Blöcke. Die Queue des ADS DAQ VI beinhaltet bereits Type-Resolved-Datenpakete. Jedes Datenpaket ist von der Größe *LvBufferSize* (vgl. Einstellungen in den Symbol Properties [▶ 25]), da **Buffered Type Resolved Queue** konfiguriert wurde.
- **2a:** Initialisierung der Queue mit einem LabVIEW™ Variant Array als Datentyp.
- **2b:** While-Schleife mit Startbedingung „LabVIEW™ Run“ und Endbedingung „Timeout oder Fehler aufgetreten“.
- **2b.1:** Dequeue-Element: Wartet auf empfangene Datenpakete mit spezifiziertem Timeout von 5 Sekunden (anpassbar).
- **2b.2:** For-Schleife: Entnimmt die einzelnen Elemente des Variant Arrays und konvertiert den Datentyp in den entsprechenden LabVIEW™-Datentyp. Hier kann der Nutzer beispielsweise direkt auf den umgewandelten Typ zugreifen und damit weiterarbeiten.

- **2b.3:** Überprüft den aktuellen Zustand der Queue, z. B., ob die Queue stetig wächst. Wenn dies der Fall ist, kommen die Daten schneller von ADS DAQ VI an als sie in den LabVIEW™-Datentyp konvertiert werden können. Optional kann der Nutzer hier ein Anzeigeelement oder Ähnliches einfügen, um die Queue zu überwachen.
- **2c:** Gibt den Queue-Speicher frei. Danach wird die Queue aus dem LV-Speicher freigegeben. Optional kann der Nutzer nach diesen Schritten das ADS-Client-Handle aus dem Speicher freigeben.

Im zweiten Beispiel wird die Startbedingung von LabVIEW™-Run auf Trigger on LabVIEW™-Event geändert und das *recording* in eine TDMS-Datei aktiviert.



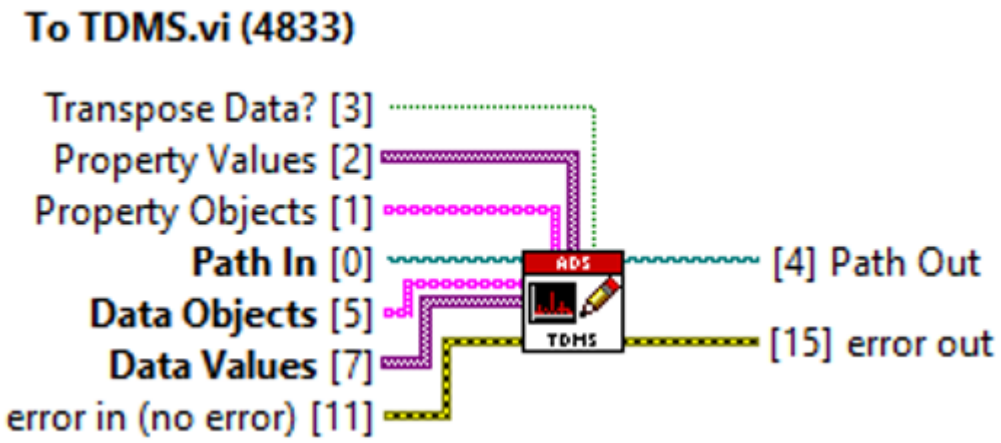
- Die bereits im ersten Beispiel beschriebenen Blöcke bleiben in ihrer Funktion identisch. Nur der Timeout wurde in diesem Fall zu -1 gesetzt (unendlich lange warten), da der Start der Messung durch LabVIEW™ getriggert wird.
- **2b.4:** Der TDMS-Block wird automatisch generiert aufgrund der Einstellung, dass ein *recording* stattfinden soll. Der Nutzer muss die zu speichernden Daten manuell mit dem To Variant-Block verknüpfen. Sollten die empfangenen Daten einem LabVIEW™-Signal entsprechen, können diese vor dem To Variant in eine LabVIEW™-Waveform umgewandelt werden. Der Nutzer kann dazu die automatisch generierte Konstante *dt* nutzen, welche dem zeitlichen Abstand zweier Datenpunkte entspricht (Abtastperiodendauer).
- **3a:** Die ADS Reader-Handle werden bei jeder neuen Konfiguration des ADS DAQ VI neu generiert. Die obsoleten Reader-Handle werden automatisch aus dem Speicher freigegeben.
- **3b:** Es wird eine Logik erstellt, die bei einer boolschen Werteänderung ein Trigger-Event generiert. Dieses wird genutzt, um die ADS-Notifications zu starten.

- **3b.1:** Startet die ADS-Notification für die angemeldeten ADS-Symbole.

Hinweise zum „To TDMS“ im Blockdiagramm

Der Block *To TDMS* ermöglicht das Speichern von ADS DAQ Instanz Daten in einer LabVIEW™-TDMS-Datei (Das NI-TDMS Dateiformat). Dieser Block wird von der ADS DAQ Instanz automatisch generiert, allerdings nur dann, wenn als Notification-Mode *Queued* gewählt wurde, vgl. ADS DAQ [▶ 47].

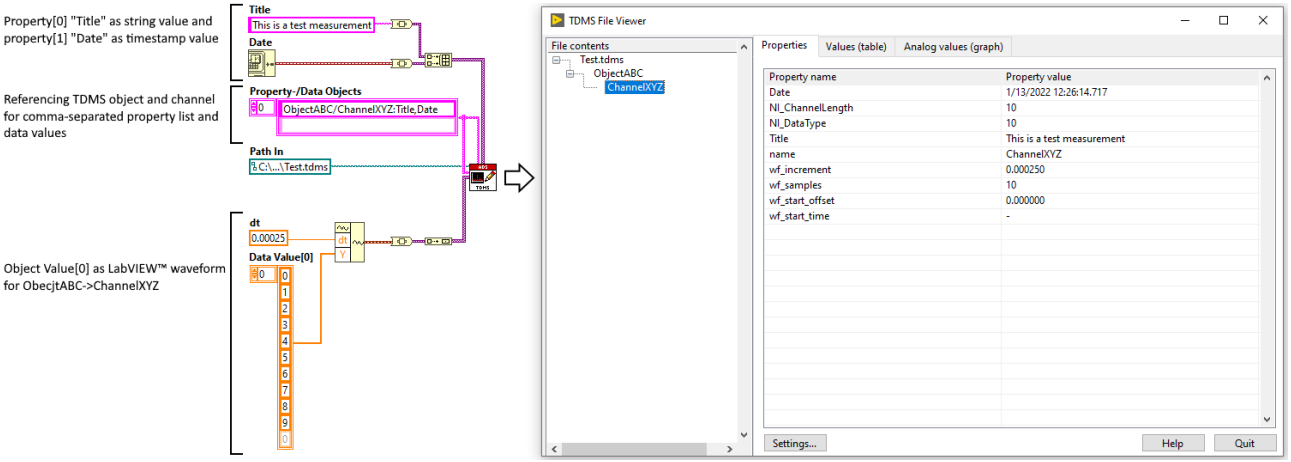
Das TDMS-Dateiformat bietet TDMS-Objekte und TDMS-Kanäle, um Daten bzw. Eigenschaften (Name, Datum, …) unter TDMS-Objekten hierarchisch zu ordnen. Mit den Eingängen **Property-/Data Objects** können sowohl Eigenschaften als auch Daten referenziert bzw. eingeordnet werden. Die Eingänge **Property-/Data Values** helfen dabei, Messwerte unter TDMS-Objekten und -Kanälen einzugeben.



Eingang/Ausgang	Bedeutung
[0] Path In	Der Pfad zur TDMS-Datei
[1] Property Objects	Ein Array einer LabVIEW™-Zeichenkette: • Referenziert ein bestimmtes Objekt (Channel) für Eigenschaften mit dem im Folgenden beschriebenen Format: 1. Objekt und Channels werden mit Schrägstrich „/“ getrennt. 2. Die Liste von Eigenschaften wird durch eine mit Komma getrennte Zeichenkette („,“) eingegeben. 3. Die Punkte 1 und 2 werden mit einem Doppelpunkt („:“) getrennt. **Beispiel:** Property Object[0]=ObjektABC/ChannelXYZ:Name,Author,Datum TDMS Objekt=ObjektABC TDMS Channel=ChannelXYZ Eingenschaft[0]=Name (Zeichenkette) Eingenschaft[1]=Author (Zeichenkette) Eingenschaft[2]=Datum (Zeitstempel)
[2] Property Values	Ein Array von LabVIEW™ Variants: • Beschreibt die Werte von Eigenschaften.
[3] Transpose Data?	*Flag* zum Transponieren der eingegebenen 2D- der 3D-Arrays.
[5] Data Objects	Ein Array einer LabVIEW™ Zeichenkette: • Referenziert ein bestimmtes Objekt (Channel) für Daten. Für das Referenzieren wird das folgende Format genutzt: 1. Objekt und Channels werden mit Schrägstrich („/“) getrennt. **Beispiel:** Data Object[0]=ObjektABC/ChannelXYZ
[7] Data Values	Ein Array von LabVIEW™ Variants mit folgenden unterstützten LabVIEW™-Typen:

Eingang/Ausgang	Bedeutung
	• LabVIEW™-Waveform • 1D, 2D, 3D Array (2D, 3D Array werden intern in 1D umgeformt): ◦ Boolescher Typ ◦ Ganzzahlige nummerische Typen (i8 … i64,u8 … u64) ◦ Nicht ganzzahlige nummerische Typen (float32, float64) ◦ LabVIEW™-Zeichenkette
[4] Path Out	Der Pfad der TDMS-Datei

Folgende Grafik zeigt exemplarisch den *To TDMS* -Block im Einsatz.



7.2 ADS FlexDAQ

Das ADS FlexDAQ (**Flex**ible **D**ata **Acq**uisition) VI ist eine Weiterentwicklung des ADS DAQ [▶ 45] VI. Ebenso wie beim ADS DAQ, handelt es sich um ein LabVIEW™ Express VI, welches das Konfigurieren von Messaufgaben mit TwinCAT 3 vereinfacht und lesend auf die TwinCAT-Laufzeit zugreift.

Das ADS FlexDAQ VI bietet darüber hinaus die folgenden weiteren Optionen:

- Es können beliebig viele ADS-Symbole pro Instanz gelesen werden.
- Den Symbolen kann eine eindeutige Loop-ID zugewiesen werden. Symbole mit derselben Loop-ID werden in derselben while-Schleife gelesen.
- Jedem Symbol kann ein entsprechender TDMS Flag zugewiesen werden, um die zu lesenden Daten in einer TDMS-Datei zu speichern.

Das User-Interface leitet Sie Schritt für Schritt durch die Konfiguration Ihrer Messaufgabe:

- Auswahl der zu lesenden Datenpunkte (ADS-Symbole)
- Konfiguration des Lese-Modus (ADS-Notification)
- Konfiguration der Loop-IDs
- Konfiguration der Speicherung von Messdaten in einer TDMS-Datei
- Konfiguration von Start- und Endbedingung der Messaufgabe

Nach dem Platzieren der ADS FlexDAQ Instanz im LabVIEW™-Blockdiagramm oder per Doppelklick, öffnet sich das Konfigurationsfenster. Mit Hilfe der nachfolgend beschriebenen Auswahlfenster können die Konfigurationen vorgenommen werden. Nach Fertigstellung der Konfiguration erstellt die Instanz alle notwendigen Ressourcen für das Lesen der Daten.

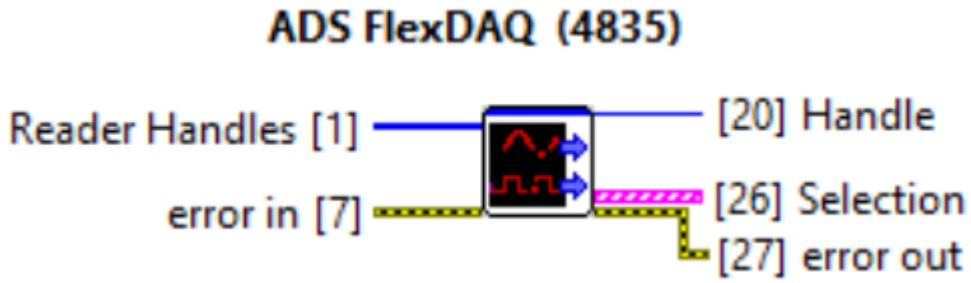
VI vor Nutzung des ADS FlexDAQ VI speichern

Das ADS FlexDAQ VI speichert die Konfiguration einer Instanz im Pfad des aktuellen Projekts. Daher ist es erforderlich, dass das Projekt zuvor gespeichert wird.

ADS FlexDAQ VI beschleunigt öffnen

✓ Die Bibliothek muss vorkompiliert sein.

1. Öffnen Sie in den LabVIEW™ Einstellungen unter **Tools > Advanced** die Einstellungen für „Mass Compile“.
2. Wählen Sie den Ordner der Bibliothek des TwinCAT 3 Interfaces for LabVIEW™ aus, z. B. *C:\Program Files\ National Instruments\LabVIEW 2023\user.lib\Beckhoff-LabVIEW-Interface* .
3. Starten Sie „Mass Compile“.



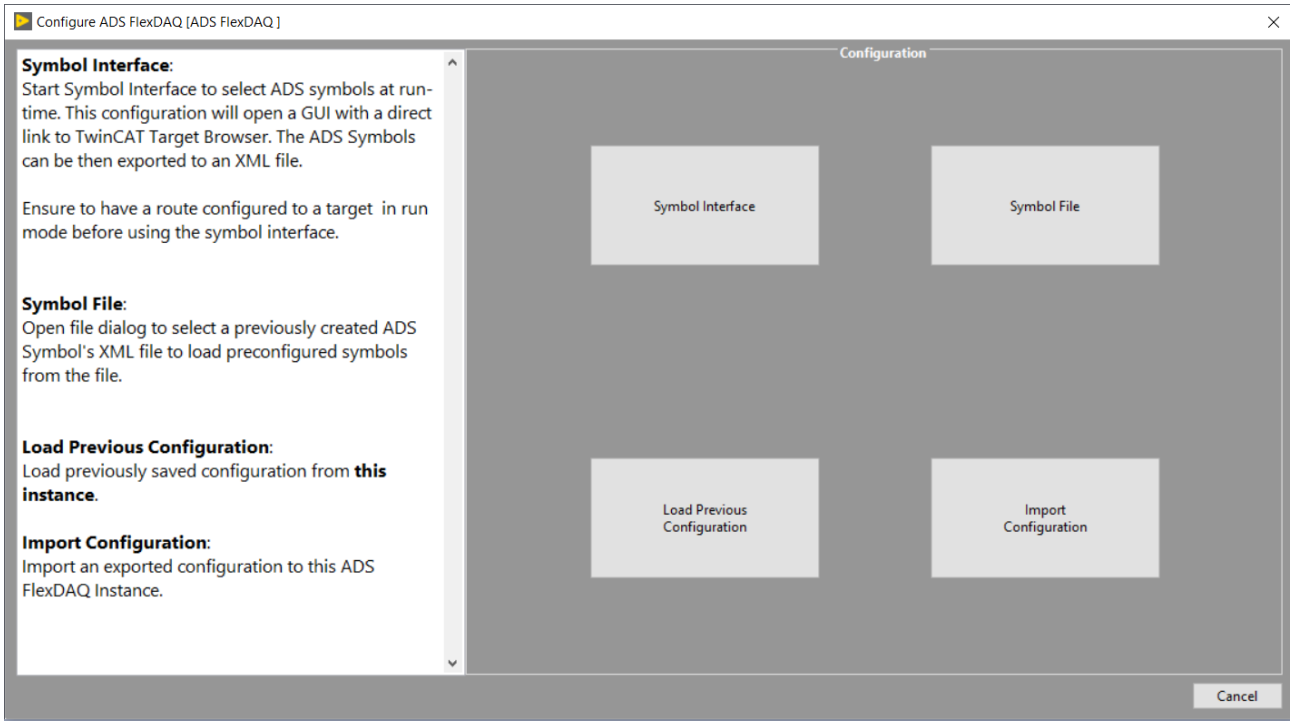
Eingang/Ausgang	Bedeutung
[1] Reader Handles	Handles auf die lesenden Symbole
[20] Handle	Handle auf den ADS-Client
[26] Selection	LabVIEW™-Cluster bestehend aus zwei Elementen: • SymbolName: Der Name des ADS-Symbols • Notification Mode: LabVIEW™ Enum ◦ Single TypeResolved Queue: Liest nur ein Sample als Notification und fügt das Sample der LabVIEW™-Queue hinzu (nur im LabVIEW™ 32-Bit). ◦ Single TypeResolved Control: Liest nur ein Sample als Notification und schreibt das Sample in das LabVIEW™-Anzeigeelement (nur im LabVIEW™ 32-Bit). ◦ Buffered TypeResolved Queue: Liest eine Anzahl von Samples, wie beschrieben von **LVBufferSize**, und fügt die Samples der LabVIEW™-Queue hinzu. ◦ Buffered TypeResolved Control: Liest eine Anzahl von Samples, wie beschrieben von **LVBufferSize**, und schreibt die Samples in das LabVIEW™-Anzeigeelement.

Generierung von TwinCAT-3-Datentyp

Um alle Notification Modi zu unterstützen, werden alle generierten Typen in Arrays umgewandelt. So werden Notifications Buffered ebenso wie Notifications Single in gleicher Art und Weise unterstützt.

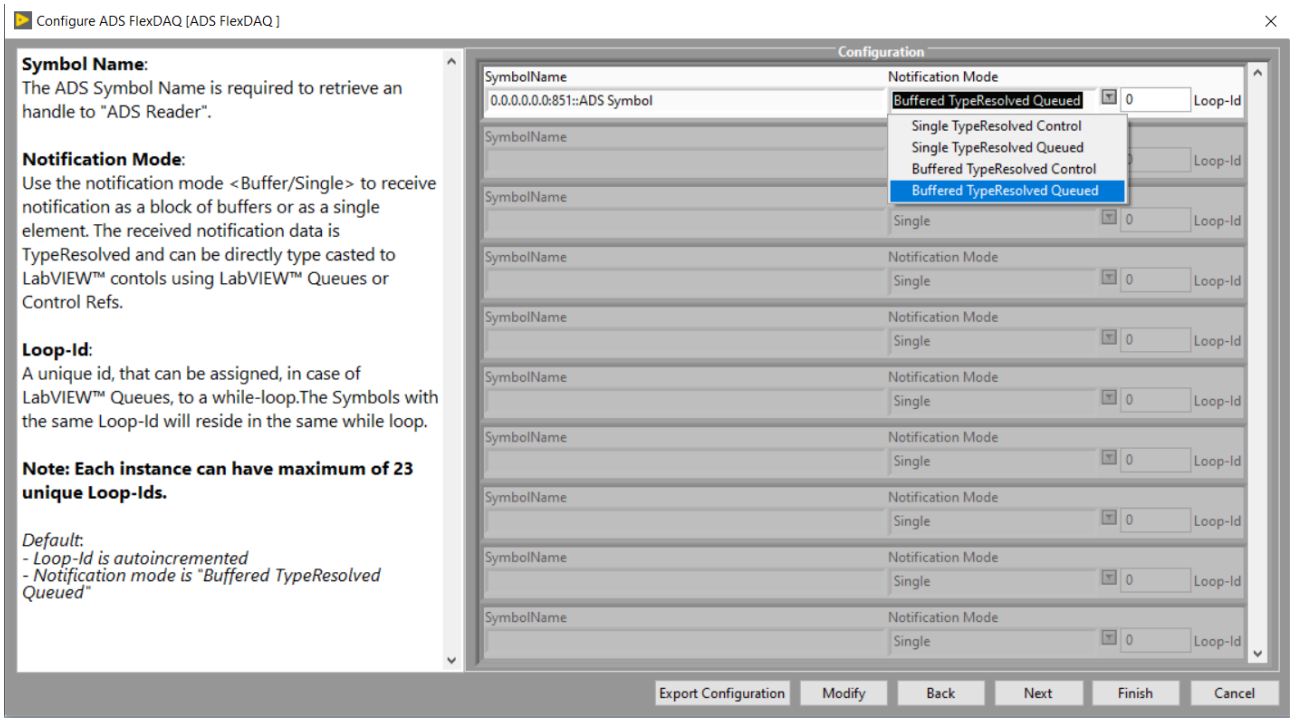
Symbol-Auswahl-Fenster

Genauso wie beim ADS DAQ [▶ 45], werden mit Hilfe des Symbol-Auswahl-Fensters die ADS-Symbole ausgewählt, die mit der ADS Flex DAQ Instanz gelesen werden sollen. Zusätzlich gibt es die Option **Import Configuration,** mit Hilfe derer eine gespeicherte Konfiguration importiert werden kann**.** Dazu öffnet sich ein LabVIEW™-Datei-Dialogfeld, in dem ein Pfad zu einer exportierten Konfiguration ausgewählt werden kann.



Notification und Loop-ID-Auswahl-Fenster

In diesem Fenster werden der Notification Mode und die Loop-ID ausgewählt. Die möglichen Notification Modi können im Abschnitt ADS DAQ [▶ 47] nachgelesen werden. Mit der Loop-ID wird das Lesen der Symbole auf while-Schleifen verteilt. Beim ADS Flex DAQ besteht die Möglichkeit, mehreren Symbolen die gleiche Loop-ID zuzuweisen. Diese werden dann in der gleichen while-Schleife verarbeitet. Somit wird die Anzahl der Schleifen reduziert.

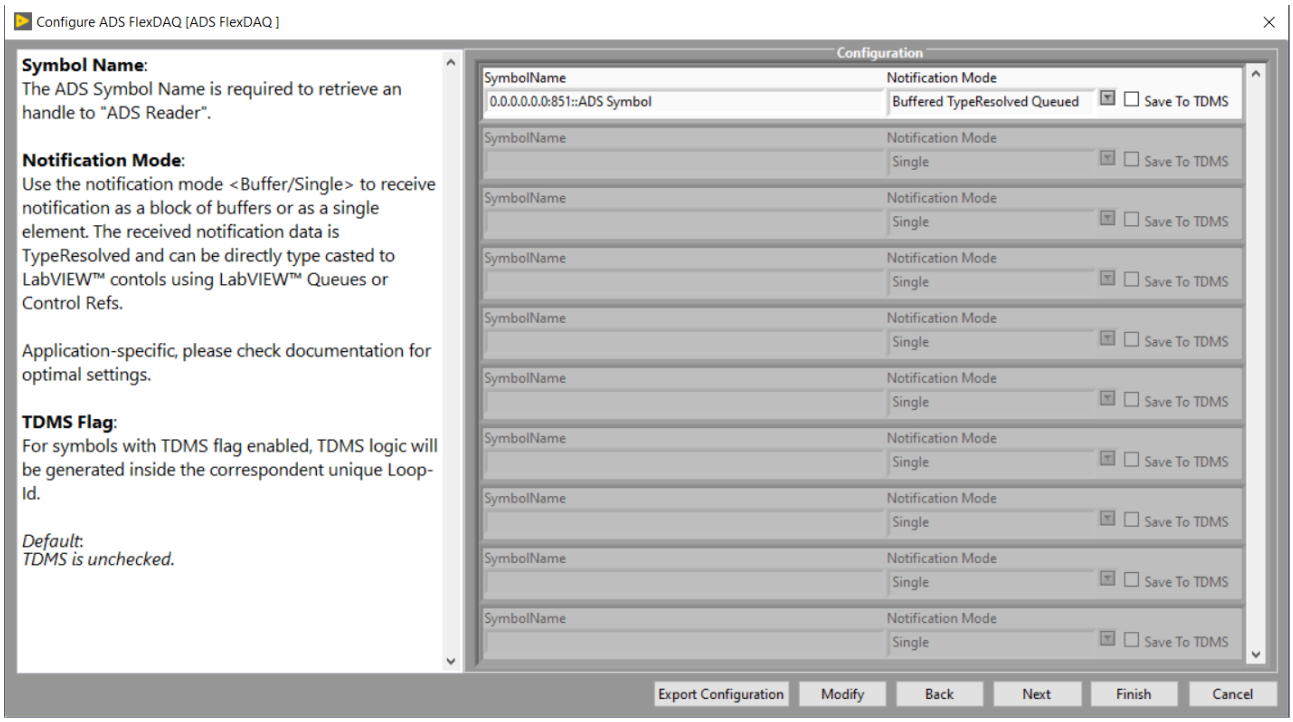


Maximale Loop-IDs

Die Anzahl der Eingänge ist bei der ADS Flex DAQ Instanz auf 23 begrenzt. Folglich können auch maximal 23 verschiedene Loop-IDs zugewiesen werden.

Speichern von Messdaten

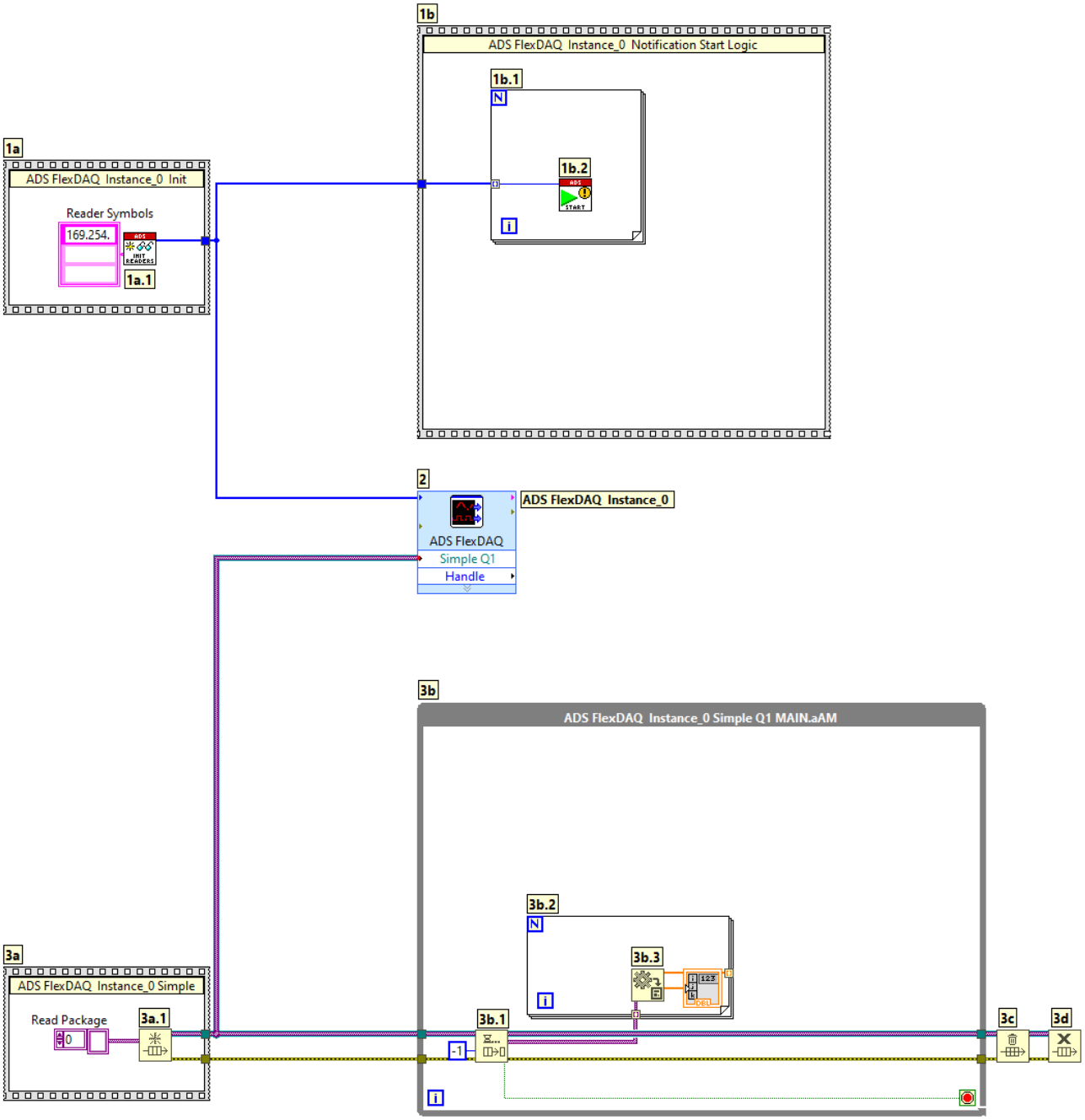
Für jedes Symbol gibt es die Option „Save To TDMS“, mit der das jeweilige Symbol beim Lesen aus TwinCAT in einer TDMS-Datei gespeichert wird. Hierzu wird der Block ADS DAQ [▶ 53] genutzt.



Automatisch generierter Code im Blockdiagramm

Nachfolgend werden exemplarisch zwei Varianten des automatisch generierten Codes erläutert.

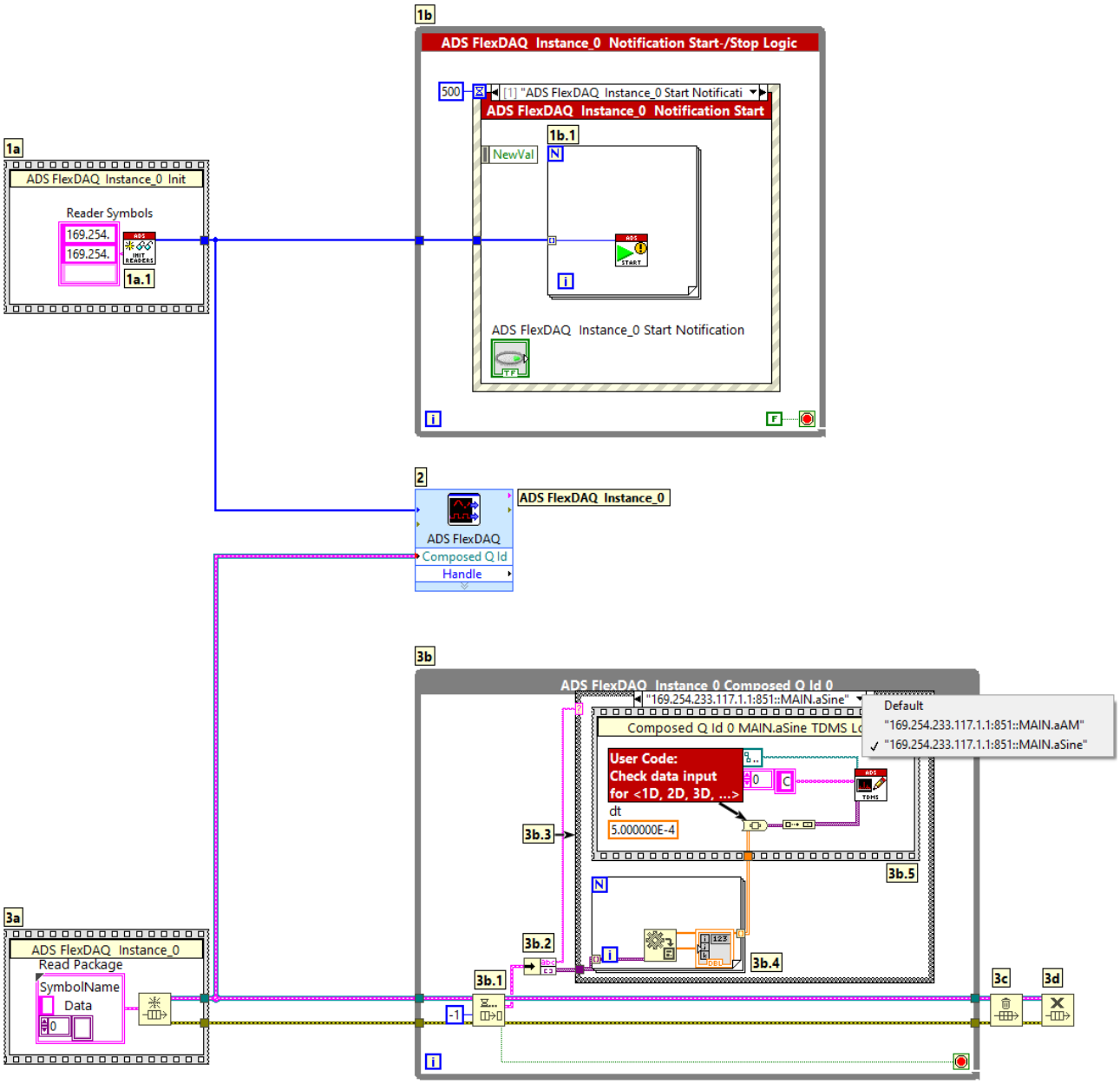
Im ersten Beispiel wird das ADS FlexDAQ VI mit **Default-Einstellungen** generiert, d. h. Buffered Type Resolved Queue, keine Speicherung von Messdaten, Start mit LabVIEW™ Run, Stop mit LabVIEW™ Stop. Gelesen wird ein ADS-Symbol MAIN.aAM.



- **1a:** Für die ausgewählten ADS-Symbole werden ADS Reader-Handles initialisiert. Die ADS Reader-Handles werden bei jeder neuen Konfiguration des ADS FlexDAQ VI neu generiert. Die obsoleten Reader-Handles werden automatisch aus dem Speicher freigegeben.
- **1b:** Die ADS-Notification wird mit Hilfe des Handles gestartet. Hierzu wird auf jedes einzelne Handle iteriert und die ADS-Notification gestartet.
- **2:** Von der Instanz des ADS FlexDAQ geht ein Handle von Simpe Queue1 auf die darunterliegenden Queue-Blöcke. Die Queue des ADS FlexDAQ VI beinhaltet bereits Type-Resolved-Datenpakete. Jedes Datenpaket ist von der Größe *LvBufferSize* (vgl. Einstellungen in den Kommunikations-Modi [▶ 25]), da **Buffered Type Resolved Queue** konfiguriert wurde.
- **3a:** Die Queue wird mit einem LabVIEW™ Variant Array als Datentyp initialisiert.
- **3b:** While-Schleife mit Startbedingung „LabVIEW™ Run“ und ohne Endbedingung.
- **3b.1:** Dequeue-Element: Wartet auf empfangene Datenpakete für eine unendliche Zeit (anpassbar).
- **3b.2:** For-Schleife: Entnimmt die einzelnen Elemente des Variant Arrays und konvertiert den Datentyp in den entsprechenden LabVIEW™-Datentyp. Hier kann der Nutzer beispielsweise direkt auf den umgewandelten Typ zugreifen und damit weiterarbeiten.

- **3c und 3d:** Der Queue-Speicher wird freigegeben. Danach wird die Queue aus dem LabVIEW™-Speicher freigegeben. Optional kann der Nutzer nach diesen Schritten das ADS-Client-Handle aus dem Speicher freigeben.

Im zweiten Beispiel wird die Startbedingung von LabVIEW™-Run auf „Trigger on LabVIEW™ Event“ geändert und das Speichern der Daten in eine TDMS-Datei aktiviert. Es werden zwei Symbole gelesen: MAIN.aAM und MAIN.aSine. Diesen wird die gleiche Loop-ID zugewiesen. Somit werden beide Symbole in der gleichen while-Schleife gelesen.



- Die bereits im ersten Beispiel beschrieben Blöcke **1a, 2, 3b, 3b.2, 3b.4, 3c** und **3d** bleiben in ihrer Funktion identisch.
- **1b:** Es wird eine Logik erstellt, die bei einer boolschen Wertänderung ein Trigger-Event generiert. Dieses wird genutzt, um die ADS-Notifications zu starten.
- **3a:** Die Queue wird mit einem LabVIEW™ Cluster als Datentyp initialisiert mit den Member Variablen:
  - SymbolName: Identifiziert das gelesene Datenpaket
  - Data: TypeResolved ADS-Notification-Datenpaket
- **3b.2:** Das gelesene Datenpaket wird entpackt und an die entsprechende Case-Struktur verteilt.
- **3b.3:** Case-Struktur, die die ADS-Daten für die entpackten Symbol-Daten an LabVIEW™ weiter gibt. Im Beispiel verwaltet die Case-Struktur zwei ADS Symbol MAIN.aAM und MAIN.aSine.

- **3b.5:** Der TDMS-Block wird automatisch generiert aufgrund der Einstellung, dass die Daten gespeichert werden sollen. Die zu speichernden Daten sind automatisch mit dem To Variant-Block verknüpft. Sollten die empfangenen Daten einem LabVIEW™-Signal entsprechen, können diese vor dem To Variant in eine LabVIEW™-Waveform umgewandelt werden. Der Nutzer kann dazu die automatisch generierte Konstante *dt* nutzen, welche dem zeitlichen Abstand zweier Datenpunkte entspricht (Abtastperiodendauer).

7.3 ADS Write Assistant

Das ADS Write Assistant VI ist ebenso wie das ADS FlexDAQ [▶ 54] ein LabVIEW™ Express VI, welches die Konfiguration von Übertragungsaufgaben vereinfacht. Mit dem ADS Write Assistant VI können Daten aus LabVIEW™ nach TwinCAT 3 geschrieben werden.

Das User-Interface des ADS Write Assistant VIs leitet Sie Schritt für Schritt durch die Konfiguration Ihrer Übertragungsaufgabe:

- Auswahl der zu schreibenden Datenpunkte (ADS-Symbole)
- Konfiguration der Loop-IDs
- Konfiguration von Start- und Endbedingung des Schreibvorgangs mit Hilfe des Übertragungsjob-Auswahl-Fensters

Nach dem Platzieren der ADS Write Assistant Instanz im LabVIEW™-Blockdiagramm oder per Doppelklick, öffnet sich das Konfigurationsfenster. Mit Hilfe der nachfolgend beschriebenen Auswahlfenster können die Konfigurationen vorgenommen werden. Nach Fertigstellung der Konfiguration erstellt die Instanz alle notwendigen Ressourcen für das Schreiben der Daten.

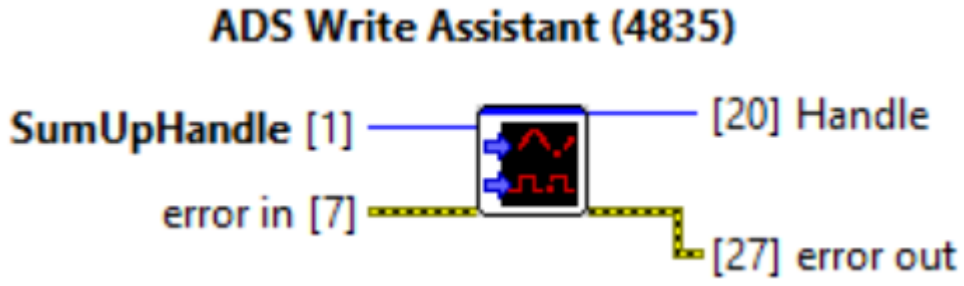
VI vor Nutzung des ADS Write Assistant VI speichern

Das ADS Write Assistant VI speichert die Instanz-Konfiguration im Pfad des aktuellen Projekts. Daher ist es erforderlich, dass das Projekt zuvor gespeichert wurde.

ADS Write Assistant VI beschleunigt öffnen

✓ Die Bibliothek muss vorkompiliert sein.

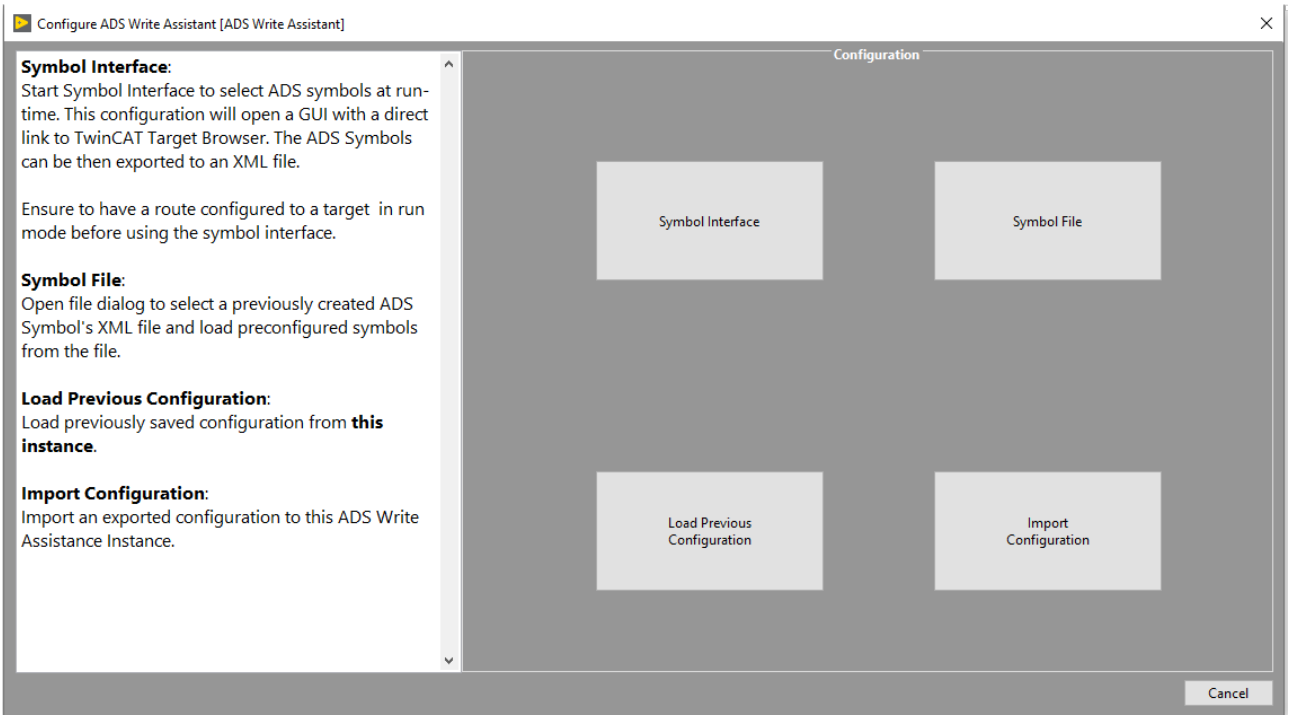
1. Öffnen Sie in den LabVIEW™ Einstellungen unter **Tools > Advanced** die Einstellungen für „Mass Compile“.
2. Wählen Sie den Ordner der Bibliothek des TwinCAT 3 Interfaces for LabVIEW™ aus, z. B. *C:\Program Files\ National Instruments\LabVIEW 2023\user.lib\Beckhoff-LabVIEW-Interface*.
3. Starten Sie „Mass Compile“.



Eingang/Ausgang	Bedeutung
[1] SumUp Handle	Das Handle auf den SumUp Writer
[20] Handle	Das Handle auf den ADS-Client

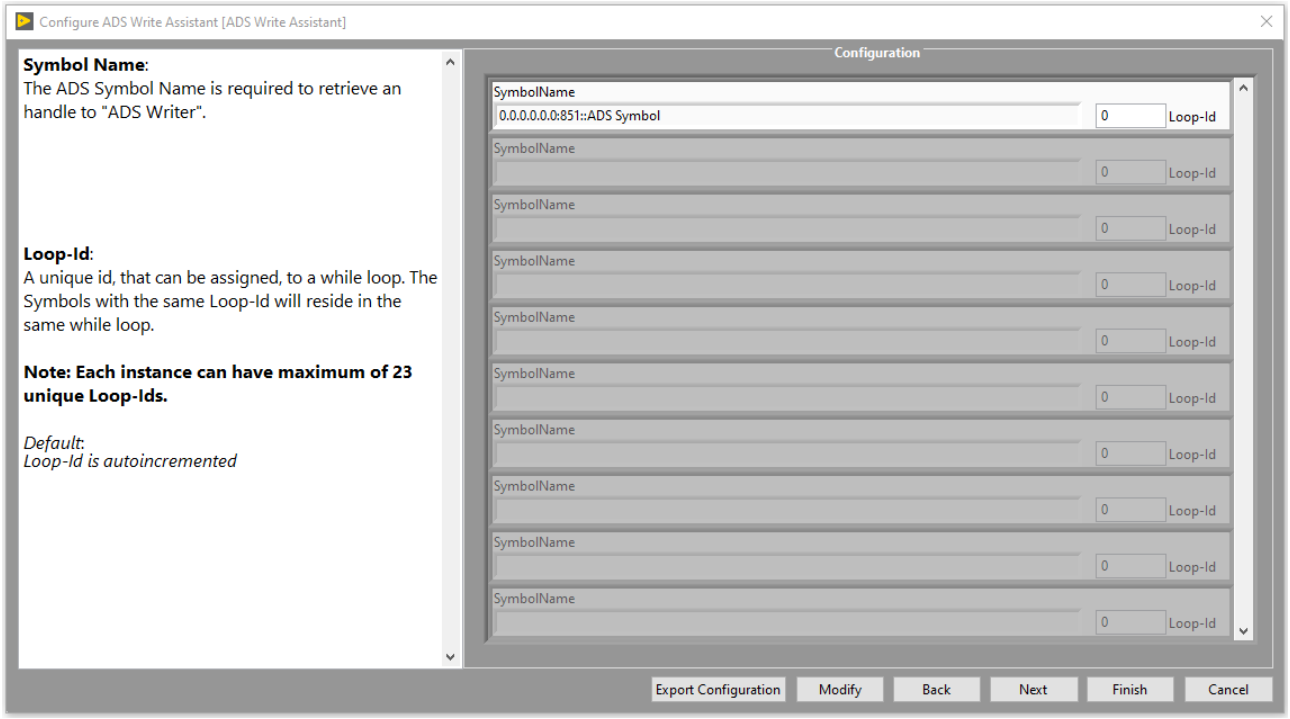
Symbol-Auswahl-Fenster

Das Symbol-Auswahl-Fenster beim ADS Write Assistant bietet die gleiche Funktionalität wie beim ADS FlexDAQ [▶ 55]. Hier kann eine der Optionen ausgewählt werden, um eine neue Konfiguration zu starten oder die Einstellungen einer bestehenden Konfiguration zu übernehmen. Wenn Sie über Symbol Interface eine neue Konfiguration starten, gelangen Sie automatisch weiter zum nächsten Fenster.



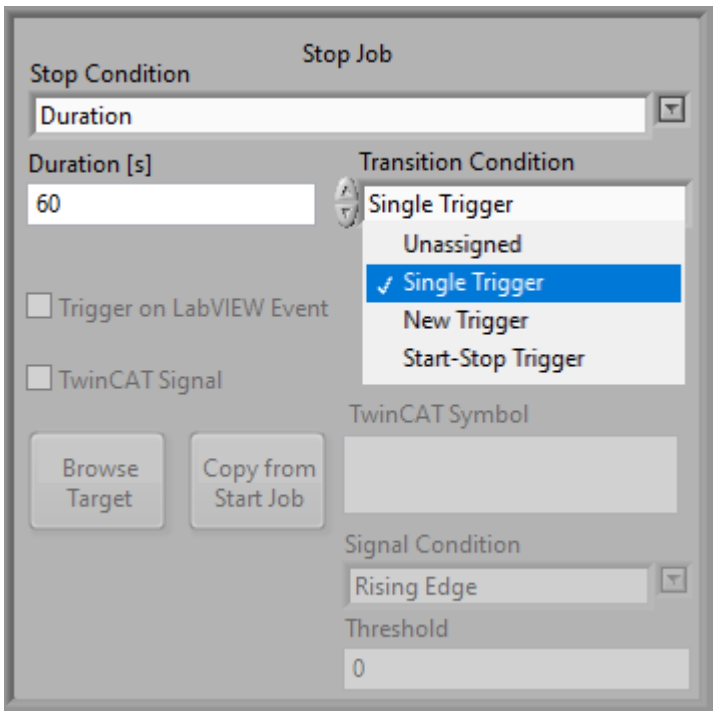
Loop-ID-Fenster

In diesem Fenster wird den Symbolen eine eindeutige Loop-ID zugewiesen. Damit wird das Schreiben der Symbole auf verschiedene while-Schleifen verteilt. Wenn zwei Symbole die gleiche Loop-ID haben, haben beide Symbole die gleiche Schleife benutzt. Die Anzahl der Loop-IDs bestimmt die Anzahl der generierten while-Schleifen im LabVIEW™-Blockdiagramm.



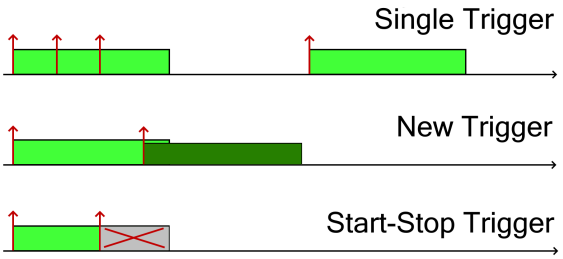
Schreibvorgang-Konfigurator (Auswahl-Fenster)

In diesem Fenster kann die Start/Stopp-Bedingung für den Datentransport von LabVIEW™ nach TwinCAT 3 konfiguriert werden. Der Konfigurator verhält sich identisch zum ADS DAQ [▶ 49] mit Ausnahme der Stopp-Bedingung.



Mit dem Bedienelement **Stop Job** konfigurieren Sie das Stoppen des Schreibvorgangs. Der Parameter **Stop Condition** beschreibt die Stopp-Bedingung.

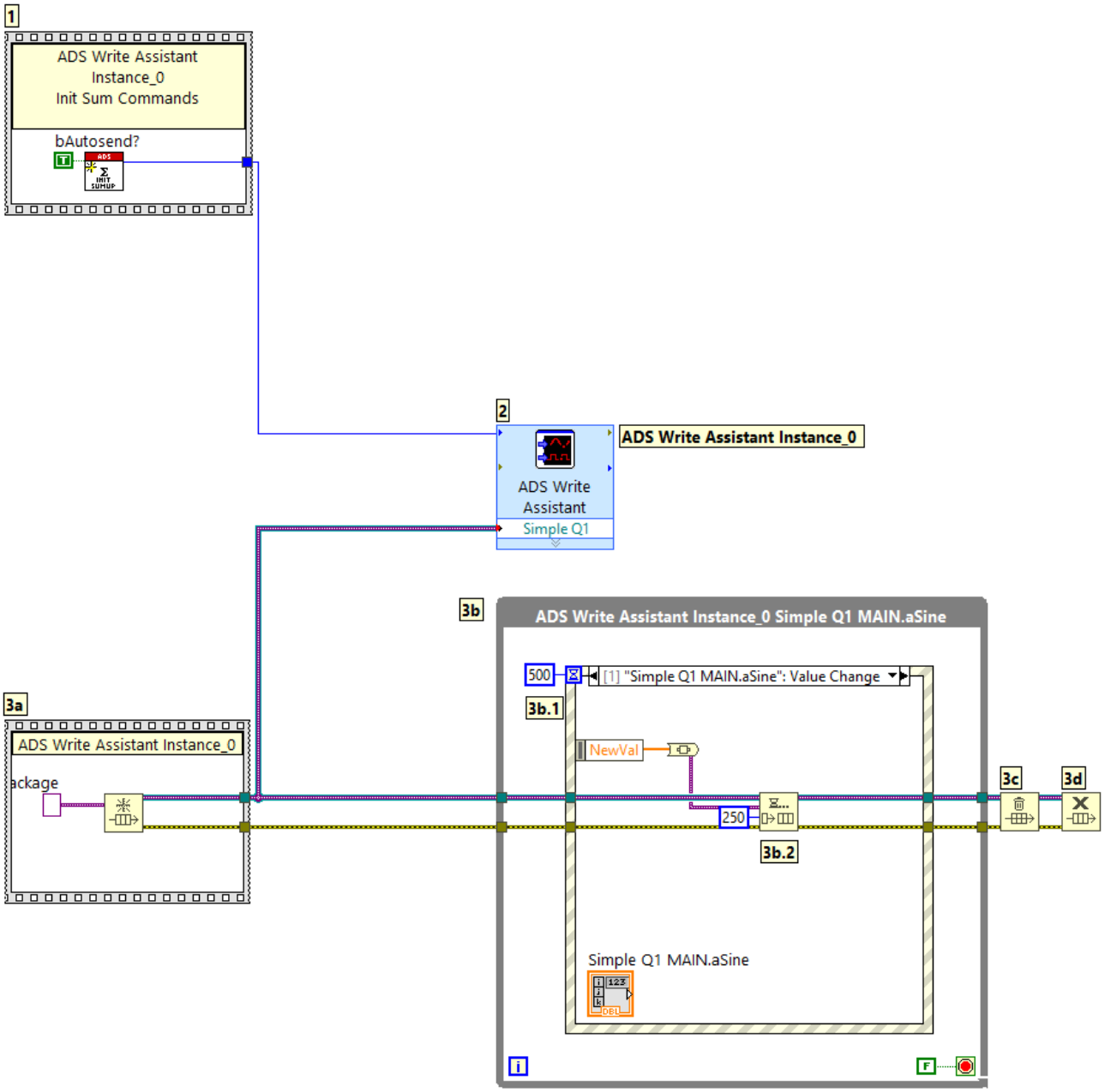
- Die Optionen **LabVIEW™ Abort** und **On Signal** werden im ADS DAQ [▶ 49] beschrieben.
- **Duration**: Wird Duration als Stopp-Bedingung gewählt, dann wird der Schreibvorgang nach einer Zeit bestimmt. Bei dieser Bedingung ist zusätzlich die Option „Transition Condition“ verfügbar. Damit kann die Schreibdauer durch einen Trigger beeinflusst werden. Das Bild unten zeigt die verschiedenen „Transition Condition“-Trigger.
  - Single Trigger: Startet eine neue Schreibdauer nur, wenn kein anderer Schreibvorgang vorhanden ist.
  - New Trigger: Startet immer einen neuen Schreibvorgang.
  - Start-Stop Trigger: Startet einen Schreibvorgang, wenn kein anderer vorhanden ist, oder im Gegenteil, stoppt einen vorhandenen Schreibvorgang.



Automatisch generierter Code im Blockdiagramm

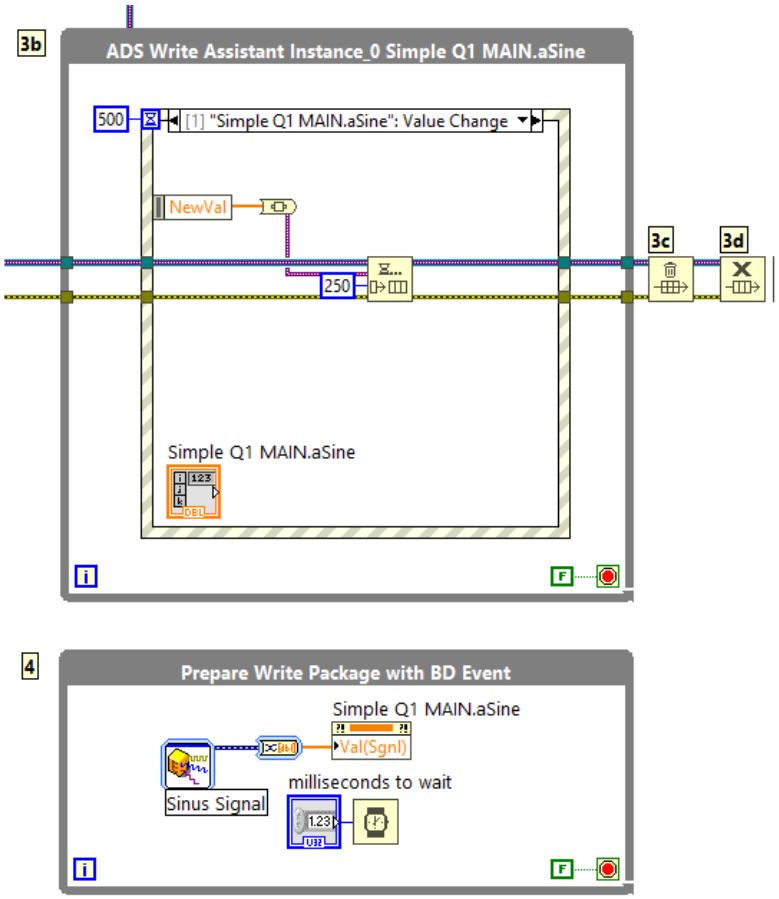
Nachfolgend werden exemplarisch zwei Varianten des automatisch generierten Codes erläutert.

Im ersten Beispiel wird das ADS Write Assistant VI mit **Default-Einstellungen** generiert, d. h. Start mit LabVIEW™ Run, Stopp mit LabVIEW™ Stop. Geschrieben wird ein ADS-Symbol MAIN.aSine. Ein Sinus-Signal, das von LabVIEW™ generiert wird und in die TwinCAT 3 Runtime übertragen wird.

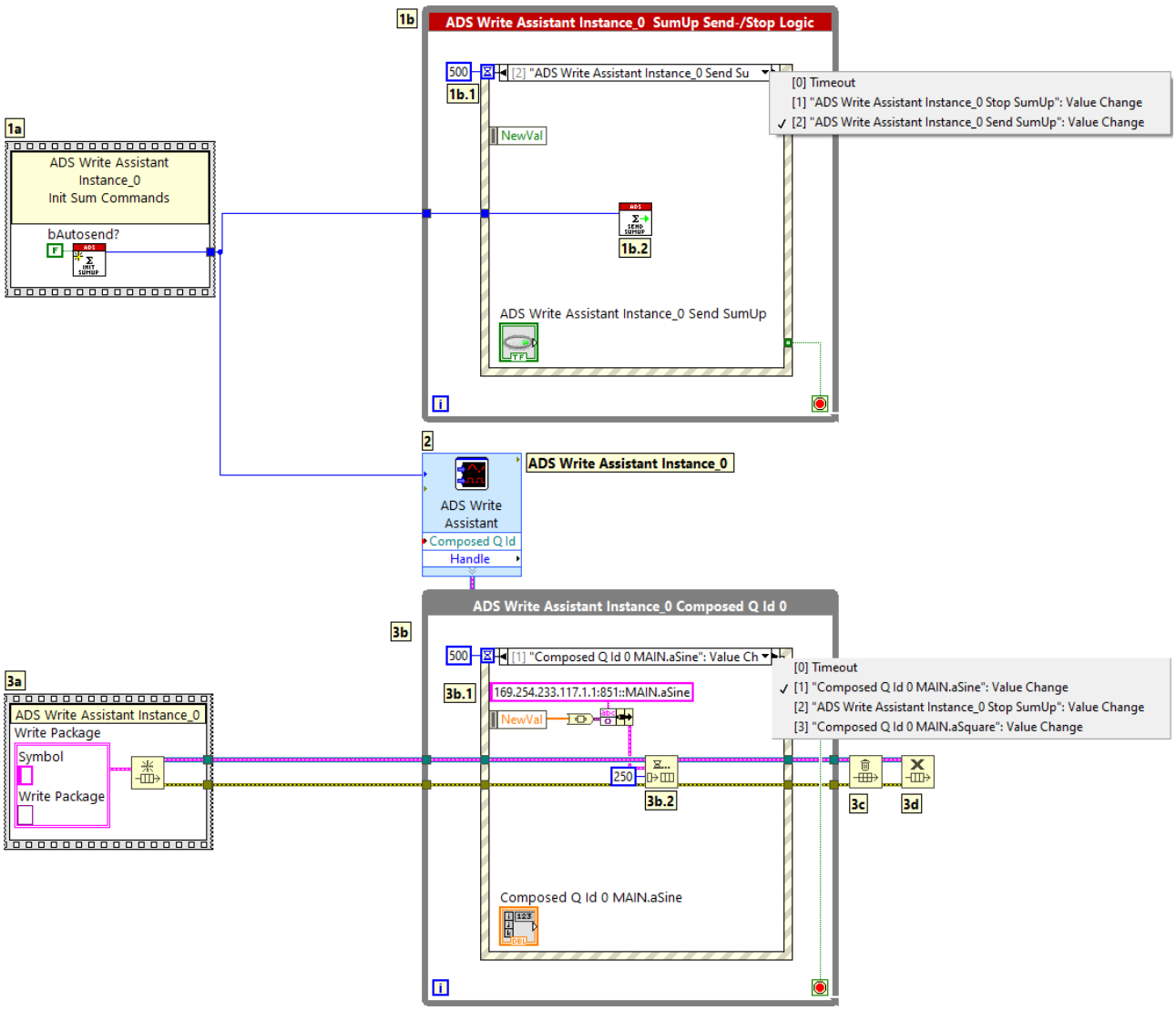


- **1:** Das ADS Writer SumUp Handle wird initialisiert und das Flag *bAutosend?* auf **True** gesetzt. Damit wird jedes neue Paket automatisch von LabVIEW™ zur TwinCAT 3 Runtime transportiert.
- **2:** Vom ADS Write Assistant geht ein Handle auf die darunterliegenden Queue-Blöcke. Die Queue leitet jedes neue Datenpaket an den ADS Write Assistant weiter. Bis dahin beinhaltet das Datenpaket nur Rohdaten, die danach vom ADS Write Assistant automatisch mit dem TypeResolver in einen TwinCAT-3-Datentyp umgewandelt werden und dann mit dem ADS SumUp nach TwinCAT übertragen werden.
- **3a:** Die Queue wird mit einem LabVIEW™ Variant als Datentyp initialisiert.
- **3b:** While-Schleife mit Startbedingung „LabVIEW™ Run“ ohne Endbedingung.
- **3b.1**: Der ADS Write Assistant generiert für jedes selektierte Symbol einen Ereignis-Case mit der Eigenschaft „Wertänderung“. Nur bei Wertänderungen werden diese dem SumUp Handle zur Verfügung gestellt. Das Bild unten zeigt ein Beispiel, wie die neuen Daten generiert werden können. Hier geht es um die sogenannten LabVIEW™-Block-Diagramm-Ereignisse.
- **4**: Das ADS Symbol MAIN.aSine ist in diesem Fall einen LREAL Array von 20 Elementen. Das generierte Sinus-Signal beinhaltet daher auch nur 20 Datenpunkte.
- **3b.2**: Enqueue-Element: Wartet auf neue Datenpakete und fügt jedes neues Datenpaket in die Warteschleife ein.

- **3c und 3d**: Gibt den Queue-Speicher frei. Danach wird die Queue aus dem LabVIEW™-Speicher freigegeben. Optional kann der Nutzer nach diesen Schritten den ADS-Client-Handle aus dem Speicher freigeben.



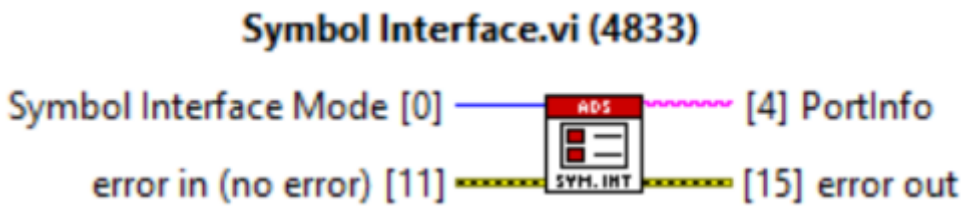
Im zweiten Beispiel werden die Start-/Stopp-Bedingungen auf Trigger on LabVIEW™ Event gesetzt. Zusätzlich werden hier zwei Symbole mit dem SumUp Handle geschrieben: MAIN.aSquare und MAIN.aSine. Den beiden Symbolen ist die gleiche Loop-ID zugewiesen. Somit nutzen beide Symbole eine while-Schleife und dieselbe Ereignis-Struktur.



- Die bereits im ersten Beispiel beschrieben Blöcke **2**, **3b**, **3b.1**, **3b.2**, **3c** und **3d** bleiben in ihrer Funktion identisch.
- **1a:** Der ADS Writer SumUp Handle wird initialisiert und das Flag *bAutosend?* auf **False** gesetzt. Das neue Paket muss explizit an TwinCAT 3 geschickt werden. Im Bespiel wird dies von dem separaten Event „Send SumUp“ gemacht.
- **1b:** Es wird eine Logik erstellt, die bei einer boolschen Wertänderung ein Trigger-Event generiert. Die Wertänderung startet den Schreibvorgang.
- **1b.2:** Ein Schreibvorgang wird mit Hilfe eines SumUp Handles gestartet.
- **3a:** Die Queue wird mit einem LabVIEW™ Cluster als Datentyp mit den folgenden Member Variablen initialisiert:
  - SymbolName: Identifiziert das zu schreibende Datenpaket.
  - Write Package: Enthält das zu übertragende Datenpaket.

7.4 Symbol Interface

Der Block *Symbol Interface* bietet ein graphisches User-Interface (UI) zum Browsen von ADS-Symbolen. Er vereinfacht es, verschiedene ADS-Symbole als Lese- oder Schreib-Symbole auszuwählen, sowie die Symbole zu parametrieren. Der Block erstellt, auf Basis der vom Nutzer erstellten Konfiguration, eine LabVIEW™-Zeichenkette, welche an das Init VI übergeben werden kann. Ebenso kann aus dem UI eine XML-Datei der erstellten Konfiguration gespeichert werden, sodass die Konfiguration geladen und ohne wiederholtes manuelles Konfigurieren an das Init VI übergeben werden kann.

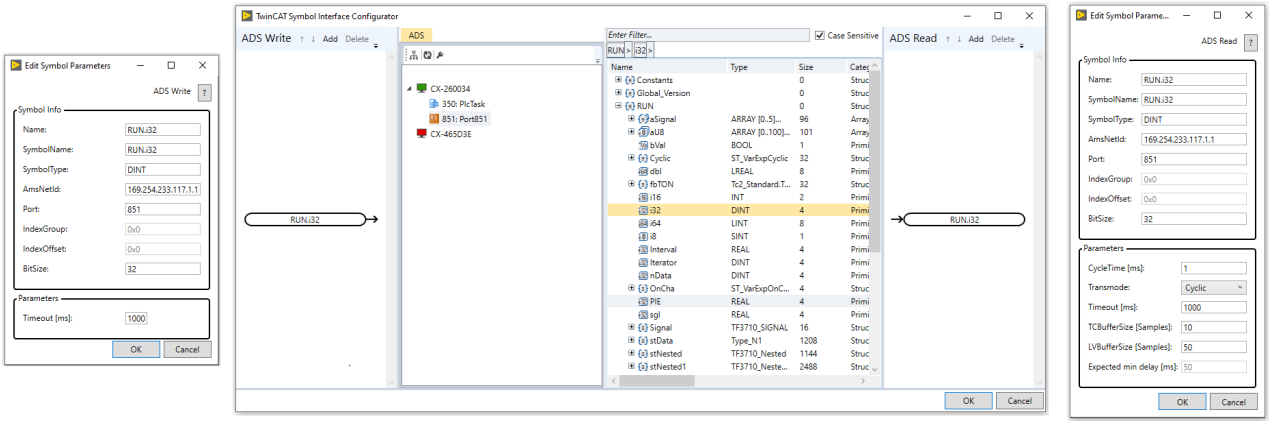


Eingang/Ausgang	Bedeutung
[0] Symbol Interface Mode	LabVIEW™-Enum bestehend aus drei Modi: • ADS-Symbole nur für den Lese-Zugriff • ADS-Symbole nur für den Schreib-Zugriff • Read & Write: Lesen und Schreiben sind erlaubt.
[4] PortInfo	LabVIEW™-Zeichenkette in XML mit ADS-Lese- und/oder Schreibe-Symbolen.

Microsoft Windows only

Der Block *Symbol Interface* lässt sich nur mit einem Windows-Betriebssystem aufrufen.

Das Ausführen des *Symbol Interface* öffnet das unten abgebildete graphische User-Interface. Im mittleren Teil ist der TwinCAT Target Browser integriert. Dieser dient dem Browsen von Zielsystemen bzw. dessen ADS-Symbolen. Per Drag-and-drop kann ein ADS-Symbol (oder durch Mehrfachauswahl mehrere ADS-Symbole gleichzeitig) in den rechten Bereich für Lesezugriffe oder in den linken Bereich für Schreibzugriffe gezogen werden. Für jedes entsprechend selektierte ADS-Symbol erscheint ein graphisches Element im Write- oder Read-Bereich. Durch einen Doppelklick auf diese graphischen Elemente erscheint (auch hier ist eine Mehrfachauswahl möglich) ein weiteres Fenster. Das Fenster beschreibt zum einen die Informationen des ADS-Symbols und gibt zum anderen die Möglichkeit, dem ADS-Symbol Parameter anzuhängen, welche bei bestimmten Lese- oder Schreibkommandos genutzt werden.



Auch manuelles Adressieren über AmsNetId, Port, Index Group und Index Offset ist möglich. Dazu legen Sie unter ADS-Read oder ADS-Write mit **New** ein neues Element an und können den Bereich „Symbol Info“ manuell ausfüllen.

Paramater CycleTime

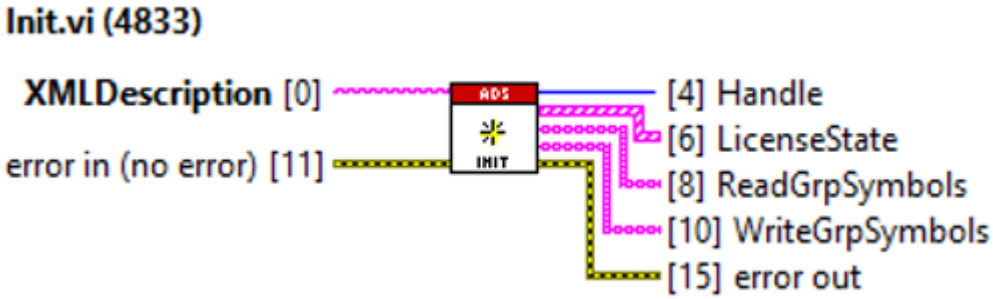
Als Default-Wert für die CycleTime wird immer die Zykluszeit angeboten, in welcher das gewählte ADS-Symbol in der TwinCAT-Laufzeit ausgeführt wird. Es sind nur Vielfache dieser Zykluszeit zulässig, da nach jedem vollendeten Zyklus eine ADS-Notification mit dem neuen Wert gesendet wird. Wird ein anderer Wert als CycleTime eingetragen, wird immer auf den nächsten zulässigen Wert aufgerundet.

7.5 Init

Die Hauptaufgabe des Blocks *Init* sind das Initialisieren und das Verbinden des ADS-Clients mit dem ADS-Router. Nach erfolgreicher Initialisierung bekommt LabVIEW™ ein Handle auf den ADS-Client zurück. Dafür nutzt der Block *Init* die Low-Level Init [▶ 85]-Blöcke im Hintergrund.

Neben seiner Hauptaufgabe führt der Block *Init* die folgenden weiteren Aufgaben aus:

- Erstellt eine Liste von ADS-Zielsystemen basierend auf dem Eingang XMLDescription.
- Überprüft jedes einzelne Zielsystem auf eine gültige TF3710 TwinCAT 3 Interface for LabVIEW™-Lizenz.
- Erstellt eine Liste, auf welchen Zielsystemen eine gültige TF3710 TwinCAT 3 Interface for LabVIEW™ abgerufen werden konnte. Lesen und Schreiben ist nur auf Zielsystemen mit gültiger Lizenz möglich.
- Erstellt eine sortierte Liste von ADS-Lese-Symbolen basierend auf dem Eingang *XMLDescription*.
- Erstellt eine sortierte Liste von ADS-Schreibe-Symbolen basierend auf dem Eingang *XMLDescription*.



Eingang/Ausgang	Bedeutung
[0] XMLDescription	LabVIEW™-XML-Zeichenkette mit ADS-Lese- und Schreib-Symbolen **oder** der Pfad als Zeichenkette zu einer vorhandenen, bereits erstellten (exportierten) XML-Datei.
[4] Handle	Handle auf den ADS-Client
[6] LicenseState	Liste der Lizenzzustände der TwinCAT-Zielsysteme
[8] ReadGrpSymbols	Liste von ADS-Lese-Symbolen
[10] WriteGrpSymbols	Liste von ADS-Schreib-Symbolen

HINWEIS

Client Handle

Das Handle auf den ADS-Client in LabVIEW™ wird nur dann freigegeben, wenn eine gültige TF3710 TwinCAT 3 Interface for LabVIEW™-Lizenz auf mindestens einem der angewählten Zielsysteme gefunden wurde. Konnte keine Lizenz gefunden werden, liefert der Init-Block einen Fehler über error out [15].

7.6 ADS-Read

Der *ADS-Read*- Block ist ein polymorphes VI und unterstützt folgende ADS-Kommunikations-Modi [▶ 25], um Daten aus TwinCAT auszulesen:

- Sync Single [▶ 29]
- Async Single [▶ 29]
- Noti. Single [▶ 29]
- Noti. Buffered [▶ 30]
- E-Noti. Sinlge [▶ 34]
- E-Noti Buffered [▶ 35]
- E-Noti. Multiple Symbols [▶ 71]
- LVB-Noti. Single Symbol [▶ 72]

- LVB-Noti. Multiple Symbols [▶ 72]

Alle Modi nutzen im Hintergrund die Low-Level Read [▶ 87]-Blöcke, um:

1. Den ADS Reader zu initialisieren.
2. Die ADS Anfrage zu schicken.
3. Auf Antwort zu warten und
4. abschließend den Reader aus dem Speicher freizugeben.

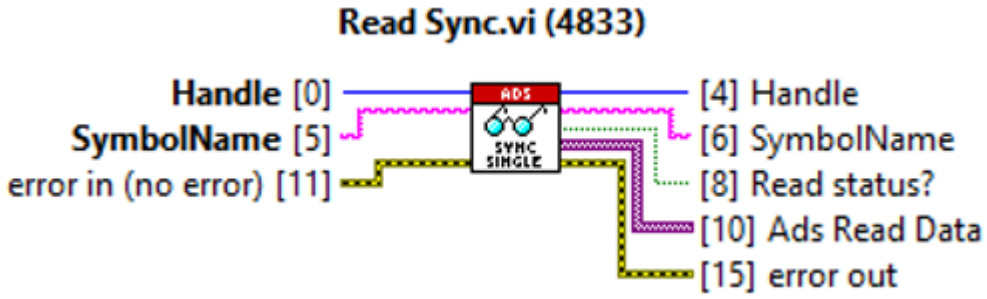
Der Abschnitt Kommunikations-Modi [▶ 25] bietet weiterführende Informationen sowie praktische Anwendungsempfehlungen. Im Kapitel Beispiele [▶ 104] finden Sie exemplarische Umsetzungen in LabVIEW™.

Je nach ausgewähltem Kommunikations-Modus werden die im Symbol Interface [▶ 65] konfigurierten Parameter genutzt oder bleiben ungenutzt. Die jeweils relevanten Parameter werden im Folgenden benannt.

Sync Single [▶ 29]

In dieser Betriebsart sendet der ADS-Client (LabVIEW™) eine Anfrage an den ADS-Server (TwinCAT) und wartet im Programmablauf auf eine Antwort des Servers.

Relevanter Parameter: Timeout

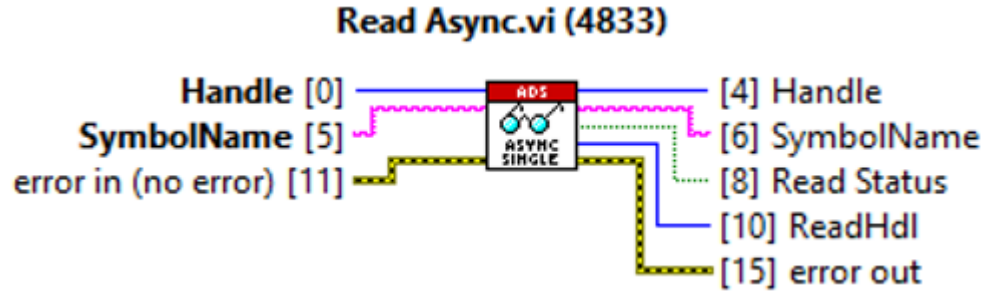


Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[4] Handle	Handle auf den ADS-Client
[6] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[8] Read status?	Lese-Status
[10] Ads Read Data	ADS-Rohdaten

Async Single [▶ 29]

In dieser Betriebsart sendet der ADS-Client (LabVIEW™) eine Anfrage an den ADS-Server (TwinCAT) und wartet im Programmablauf nicht auf eine Antwort des Servers.

Relevanter Parameter: Timeout



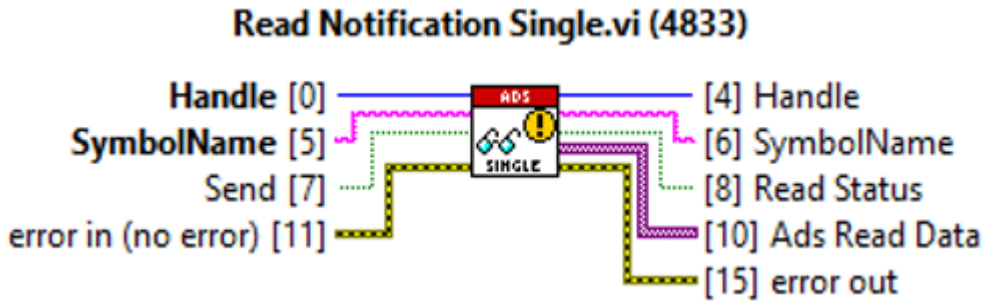
Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client

Eingang/Ausgang	Bedeutung
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[4] Handle	Handle auf den ADS-Client
[6] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[8] Read Status	Lese-Status
[10] ReadHdl	Handle auf den ADS-Reader

Noti. Single [▶ 29]

In dieser Betriebsart werden ADS-Notifications auf die betreffenden ADS-Symbole in TwinCAT registriert. Es ist nur eine Registrierung „on change“ möglich. Sobald die erste „on change“-Benachrichtigung in LabVIEW™ eingeht, wird die ADS-Notification am ADS-Server wieder abgemeldet.

Relevante Parameter: Timeout, Transmode (= on change)



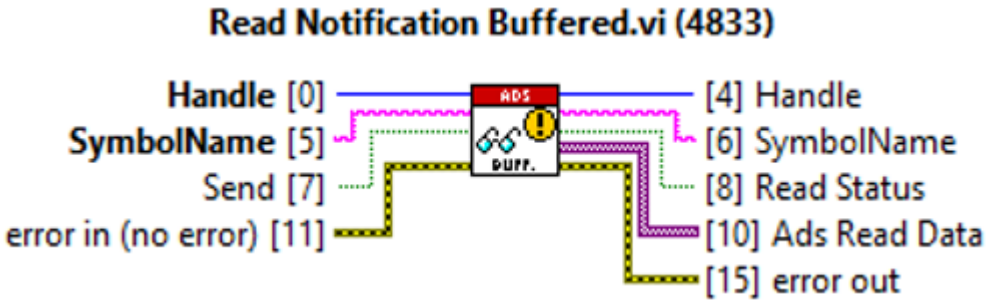
Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[7] Send	Send Flag, TRUE registriert die ADS-Notification
[4] Handle	Handle auf den ADS-Client
[6] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[8] Read Status	Lese-Status
[10] Ads Read Data	ADS-Rohdaten (Array aus zwei Einträgen, vorheriger Wert und aktueller Wert)

Noti. Buffered [▶ 30]

In diesem Kommunikations-Modus werden ADS-Notifications auf den betreffenden ADS-Symbolen in TwinCAT registriert. Der ADS-Client in LabVIEW™ nutzt zudem einen Pufferspeicher, welcher erst vollständig mit Rückmeldungen aus TwinCAT gefüllt wird, bevor der gesamte Pufferspeicher an LabVIEW™ übergeben wird. Nach Übergabe der gepufferten Daten an LabVIEW™ wird die ADS-Notification vom Server wieder abgemeldet. Es werden sowohl ADS-Notifications “on change” sowie “cyclic” unterstützt.

Relevante Parameter: Timeout, Transmode, SampleTime, LVBufferSize, TCBufferSize

Bei Transmode = “on change” ist die SampleTime sowie die TCBufferSize nicht relevant.



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client

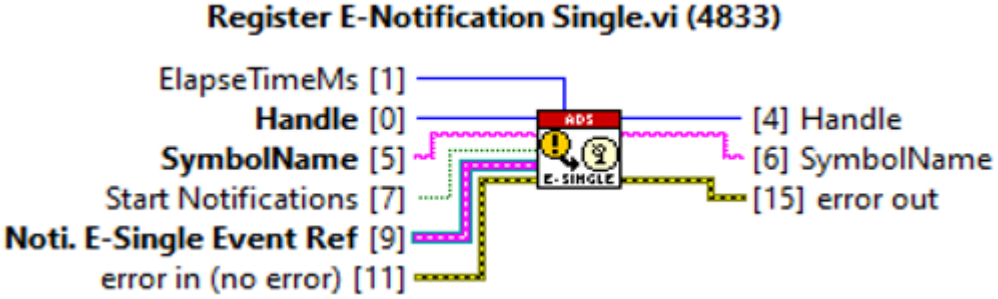
Eingang/Ausgang	Bedeutung
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[7] Send	Send Flag, TRUE registriert die ADS-Notification
[4] Handle	Handle auf den ADS-Client
[6] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[8] Read Status	Lese-Status
[10] Ads Read Data	Gepufferte ADS-Rohdaten

E-Noti. Single [▶ 34]

In dieser Betriebsart werden ADS-Notifications auf den betreffenden ADS-Symbolen in TwinCAT registriert und danach kontinuierlich als LabVIEW™-Event an LabVIEW™ übertragen. Die ADS-Notification meldet sich nach einer definierten Zeit, wenn ElapseTimeMs größer Null ist, oder bleibt bestehen, bis sie aktiv abgemeldet wird. Für die Verwendung von user events in LabVIEW™, siehe Dokumentation von LabVIEW™. Im Gegensatz zu E-Noti. Buffered (siehe unten) wird hier kein LabVIEW™-seitiger Pufferspeicher genutzt.

Relevante Parameter: Timeout, Transmode, SampleTime, TCBufferSize

Bei Transmode = “on change” ist die SampleTime sowie die TCBufferSize nicht relevant.



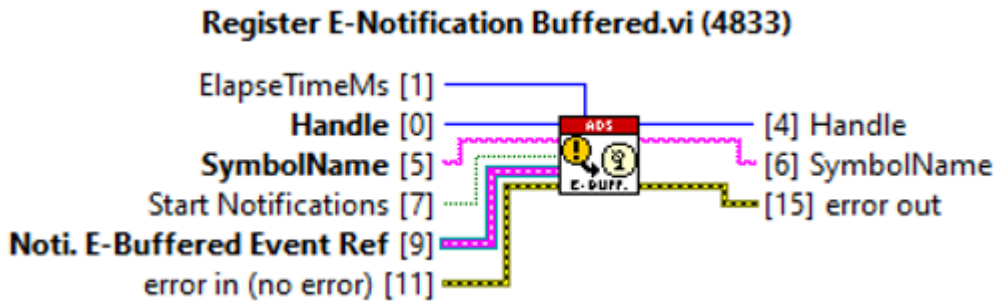
Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[1] ElapseTimeMs	Messdauer in Millisekunden: • ElapseTimeMs > 0: Die Notifications stoppen nach dem Ablauf der Zeit. • ElapseTimeMs = 0: Die Notifications müssen von außen gestoppt werden.
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[7] Send	Send Flag, TRUE startet die ADS-Notification
[9] user event	Referenz auf user event
[4] Handle	Handle auf den ADS-Client
[6] SymbolName	Der ADS-Symbol bestehend aus AMSNetId und Symbol-Name

E-Noti Buffered [▶ 35]

In dieser Betriebsart werden ADS-Notifications auf den betreffenden ADS-Symbolen in TwinCAT registriert und danach kontinuierlich als LabVIEW™-Event an LabVIEW™ übertragen. Die ADS-Notification meldet sich nach einer definierten Zeit, wenn ElapseTimeMs größer Null ist, oder bleibt bestehen, bis sie aktiv abgemeldet wird. Für die Verwendung von user events in LabVIEW™, siehe Dokumentation von LabVIEW™. Im Gegensatz zu E-Noti. Single (siehe oben) wird hier ein LabVIEW™-seitiger Pufferspeicher der Größe LVBufferSize verwendet. Nachdem der Puffer vollständig gefüllt ist, werden die gesammelten Daten per Event an LabVIEW™ übergeben.

Relevante Parameter: Timeout, Transmode, SampleTime, TCBufferSize, LVBufferSize

Bei Transmode = “on change” ist die SampleTime sowie die TCBufferSize nicht relevant.



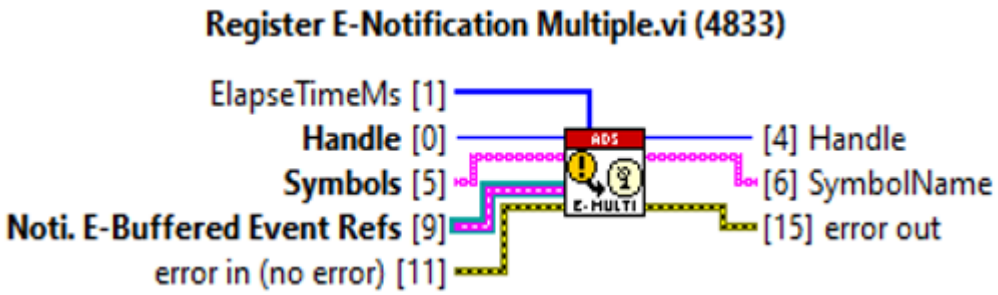
Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[1] ElapseTimeMs	Messdauer in Millisekunden: • ElapseTimeMs > 0: Die Notifications stoppen nach dem Ablauf der Zeit. • ElapseTimeMs = 0: Die Notifications müssen von außen gestoppt werden.
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[7] Send	Send Flag, TRUE startet die ADS-Notification
[9] user event	Referenz auf user event
[4] Handle	Handle auf den ADS-Client
[6] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name

E-Noti. Multiple Symbols

Diese Betriebsart funktioniert wie E-Noti. Buffered [▶ 70], mit der Ausnahme, dass dieses VI für mehrere ADS-Symbole gleichzeitig genutzt werden kann.

Relevante Parameter: Timeout, Transmode, SampleTime, TCBufferSize, LVBufferSize

Bei Transmode = “on change” ist die SampleTime sowie die TCBufferSize nicht relevant.



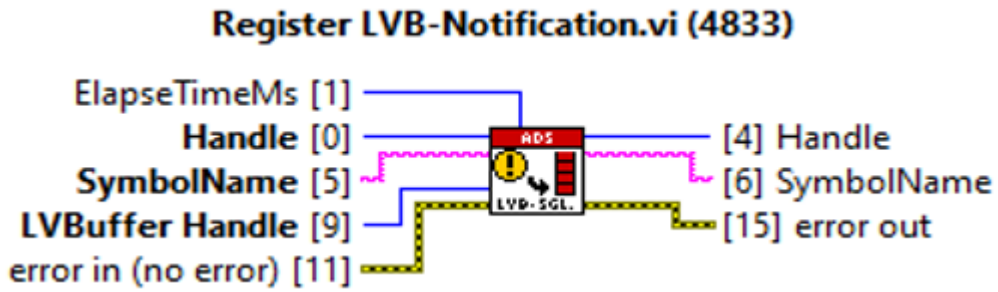
Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[1] ElapseTimeMs	Ein Array von U32 bestehend aus Messdauer in Millisekunden: • Element von ElapseTimeMs > 0: Die Notifications stoppen nach dem Ablauf der Zeit. • Element von ElapseTimeMs = 0: Die Notifications müssen von außen gestoppt werden.
[5] Symbols	Ein Array von LabVIEW™-Zeichenketten: • ADS-Symbol bestehend aus AMS-Adresse und Symbol-Name
[9] Noti. E-Buffered Event Refs	Ein Array von Referenzen auf user event
[4] Handle	Handle auf den ADS-Client
[6] SymbolName	ADS-Symbol bestehend aus AMS-Adresse und Symbol-Name

LVB-Noti. Single Symbol

Diese Betriebsart funktioniert wie E-Noti. Buffered [▶ 70], mit der Ausnahme, dass dieses VI statt über LabVIEW™-Events einen direkten Zugriff auf LVBuffer [▶ 81] ermöglicht. Infolgedessen werden auch keine LabVIEW™-Events benötigt, um die Notifications zu lesen.

Relevante Parameter: Timeout, Transmode, SampleTime, LVBufferSize, TCBufferSize

Bei Transmode = “on change” ist die SampleTime sowie die TCBufferSize nicht relevant.



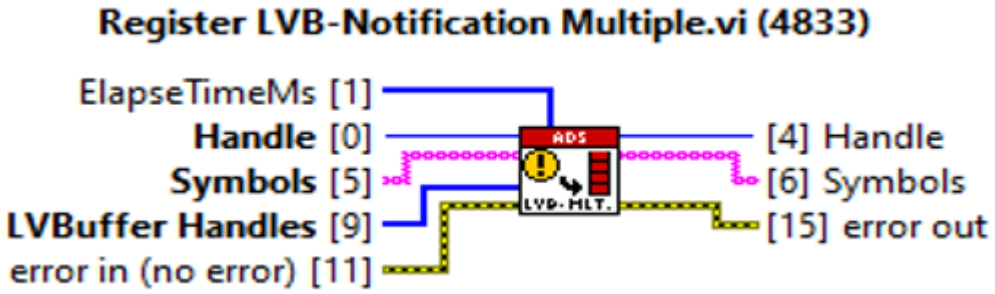
Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[1] ElapseTimeMs	Messdauer in Millisekunden: • ElapseTimeMs > 0: Die Notifications stoppen nach dem Ablauf der Zeit. • ElapseTimeMs = 0: Die Notifications müssen von außen gestoppt werden.
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[9] LVBuffer Handle	Handle auf den LVBuffer
[4] Handle	Handle auf den ADS-Client
[6] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name

LVB-Noti. Multiple Symbols

Diese Betriebsart funktioniert genau wie LVB-Noti. Single Symbol [▶ 72], mit dem Unterschied, dass dieses VI für mehrere ADS-Symbole gleichzeitig genutzt werden kann.

Relevante Parameter: Timeout, Transmode, SampleTime, LVBufferSize, TCBufferSize

Bei Transmode = “on change” ist die SampleTime sowie die TCBufferSize nicht relevant.



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[1] ElapseTimeMs	Ein Array von U32 bestehend aus Messdauer in Millisekunden: • Element von ElapseTimeMs > 0: Die Notifications stoppen nach dem Ablauf der Zeit. • Element von ElapseTimeMs = 0: Die Notifications müssen von außen gestoppt werden.
[5] Symbols	Ein Array von LabVIEW™-Zeichenketten: • ADS-Symbol bestehend aus AMS-Adresse und Symbol-Name
[9] LVBuffer Handle	Ein Array von U32 bestehend aus Handle auf LVBuffer

Eingang/Ausgang	Bedeutung
[4] Handle	Handle auf den ADS-Client
[6] Symbols	Ein Array von LabVIEW™-Zeichenketten: • ADS-Symbol bestehend aus AMS-Adresse und Symbol-Name

7.7 ADS-Write

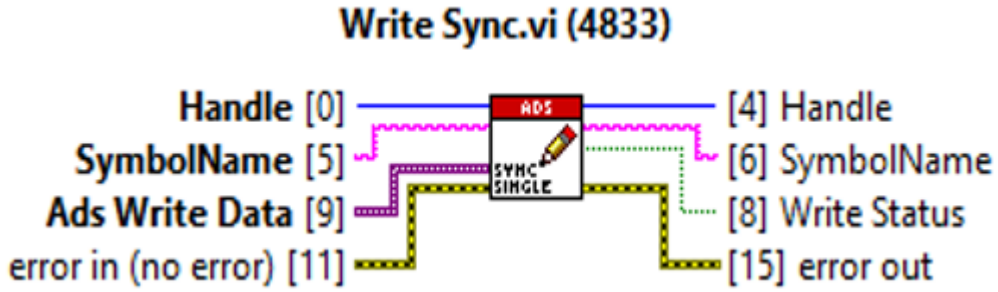
Der Block *ADS-Write* ist ein polymorphic VI und unterstützt folgende ADS-Kommunikations-Modi [▶ 25], um in TwinCAT Daten zu schreiben:

- Sync Single [▶ 30]
- Async Single [▶ 31]

Bei ADS-Write ist nur ein Parameter relevant, welcher im Symbol Interface [▶ 65] eingestellt werden kann und zwar der Parameter *Timeout*.

Sync Single [▶ 30]

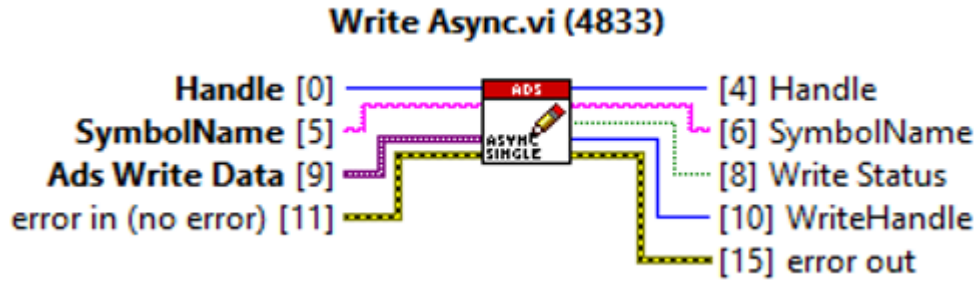
In dieser Betriebsart sendet der ADS-Client in LabVIEW™ eine Schreib-Anfrage mit dem type-resolved-ADS-Wert an den ADS-Server in TwinCAT und wartet auf eine Antwort, ob der Wert geschrieben worden ist.



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[9] Ads Write Data	Type-resolved ADS-Wert
[4] Handle	Handle auf den ADS-Client
[6] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[8] Write Status	Schreib-Status

Async Single [▶ 31]

In dieser Betriebsart sendet der ADS-Client in LabVIEW™ eine Schreib-Anfrage mit dem type-resolved-ADS-Wert an den ADS-Server in TwinCAT und wartet **nicht** auf eine Antwort, ob der Wert geschrieben worden ist.



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name

Eingang/Ausgang	Bedeutung
[9] Ads Write Data	Type-resolved ADS-Wert
[4] Handle	Handle auf den ADS-Client
[6] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[8] Write Status	Schreib-Status
[10] WriteHandle	Handle auf den ADS-Writer

7.8 TypeResolver

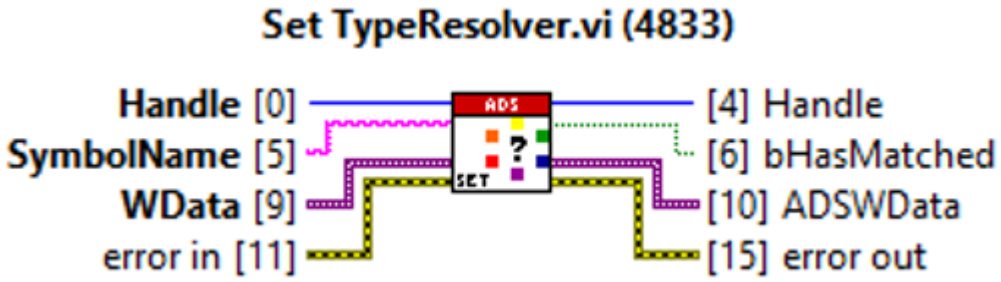
Der Block *TypeResolver* ist ein polymorphes VI und unterstützt folgende Betriebsarten, um Datentypen zu konvertieren:

- To TC (für das Schreiben von LabVIEW™ nach TwinCAT)
- From TC (für das Lesen von TwinCAT nach LabVIEW™)

To TC

Der Block *To TC* initialisiert den TypeResolver, vergleicht den LabVIEW™-Datentyp mit dem TC3-Datentyp und konvertiert die LabVIEW™-Daten in ADS-Rohdaten. Am Ende gibt der Block den TypeResolver aus dem Speicher frei.

Weitere Informationen finden Sie im Abschnitt Type Resolving [▶ 39].

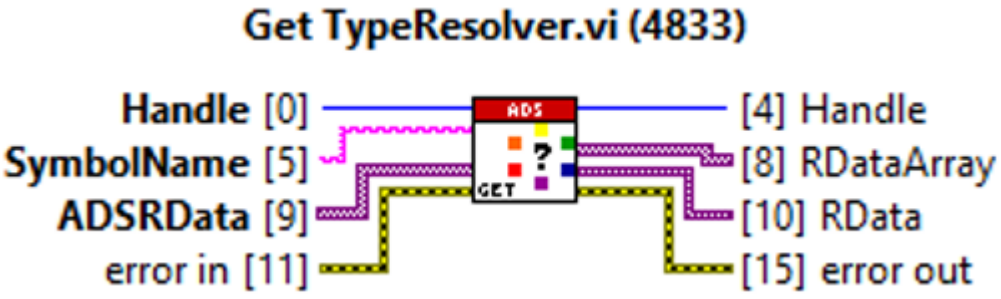


Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[9] WData	Rohdaten für ADS-Write
[4] Handle	Handle auf den ADS-Client
[6] bHasMatched	Flag (TRUE, wenn TC3- und LabVIEW™-Datentyp identisch sind, sonst FALSE)
[10] ADSWData	Type-resolved ADS-Write-Wert

From TC

Der Block *From TC* initialisiert den TypeResolver, parst und konvertiert die ADS-Daten, die mit ADS-Read von TwinCAT gelesen worden sind, in den LabVIEW™-Datentyp *Variant*. Der Block Type Release gibt den TypeResolver aus dem Speicher wieder frei.

Weitere Informationen finden Sie im Abschnitt Type Resolving [▶ 38].



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[9] ADSRData	Rohdaten aus ADS-Read
[4] Handle	Handle auf den ADS-Client
[8] RDataArray	Type-resolved ADS-Read-Array für den LabVIEW™-Datentyp
[10] RData	Type-resolved ADS-Read-Wert für den LabVIEW™-Datentyp

7.9 Release

Der Block *Release* gibt das Handle auf den ADS-Client aus dem Speicher frei.

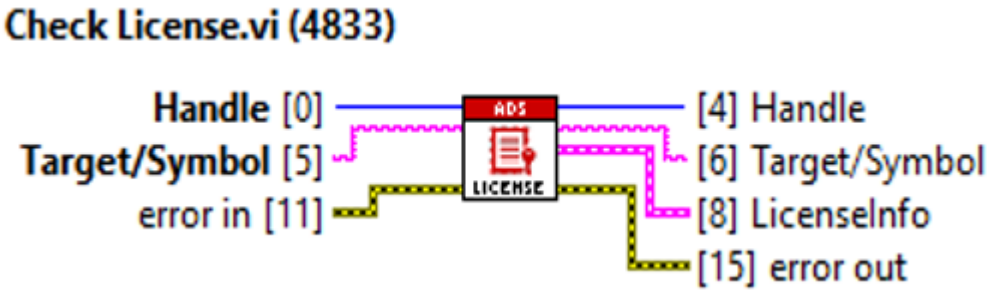
Eingang	Bedeutung
[0] Handle	Handle auf den ADS-Client

7.10 Utilities

Unter Utilities liegen zusätzliche VIs.

Check License

Der Block *Check License* überprüft das Zielsystem auf eine gültige TF3710 TwinCAT 3 Interface for LabVIEW™-Lizenz.



Eingang/Ausgang	Bedeutung
[0] Handle	Das Handle auf den Client
[5] Target/Symbol	ADS-Target als Zeichenkette bestehend auf AMSNetId oder ADS-Symbol als Zeichenkette bestehend auf AMSNetId und Symbol-Name
[4] Handle	Das Handle auf den Client
[6] Target/Symbol	ADS-Target als Zeichenkette bestehend auf AMSNetId oder ADS-Symbol als Zeichenkette bestehend auf AMSNetId und Symbol-Name
[8] LicenseInfo	LabVIEW™-Cluster bestehend aus drei Elementen: • Boolesche Flag (TRUE wenn Lizenz gültig ist, sonst FALSE) • AMSNetId des Zielsystems • Zeichenkette mit Nachricht über den Lizenz-Zustand, z. B. „license is valid“ oder „trial license denied“

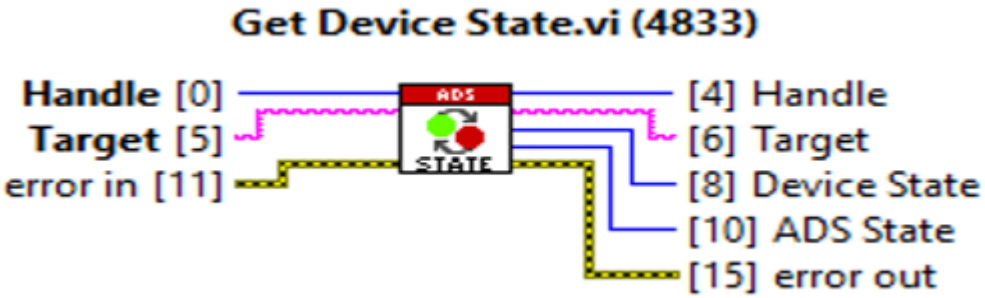
Get Device State

Der Block *Device State* ist ein polymorphes VI und kann als *Get Device State* und *Set Device State* betrieben werden.

Die Betriebsart *Get Device State* liest den aktuellen Zustand des Zielsystems und der PLC.

Die folgende Tabelle beschreibt den ADS Port und dessen Bedeutung:

ADS Port	ADS State	Zustand
TC3 PLC Port > 851	ADS_STATE_RUN	PLC läuft.
	ADS_STATE_STOP	PLC ist gestoppt.
	ADS_STATE_RESET	PLC ist resettet.
System Port = 10000	ADS_STATE_RECONFIG	TwinCAT ist im Config-Modus.
	ADS_STATE_RESET	TwinCAT ist im Run-Modus.



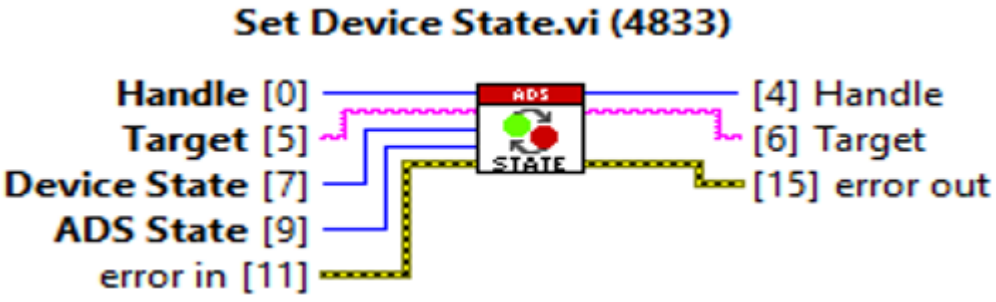
Eingang/Ausgang	Bedeutung
[0] Handle	Das Handle auf den Client
[5] Target	ADS-Target als Zeichenkette bestehend auf AMSNetId
[4] Handle	Das Handle auf den Client
[6] Target	ADS-Target als Zeichenkette bestehend auf AMSNetId
[8] Device State	Zustand des Zielsystems (ENUM)
[10] ADS State	Zustand der PLC (ENUM)

Set Device State

Der Block *Set Device State* ändert den aktuellen Zustand des Zielsystems und der PLC.

Mit einem bestimmten ADS Port und ADS State können die PLC oder TwinCAT in einen entsprechenden Zustand gesetzt werden. Die folgende Tabelle beschreibt den ADS Port und dessen Nutzung:

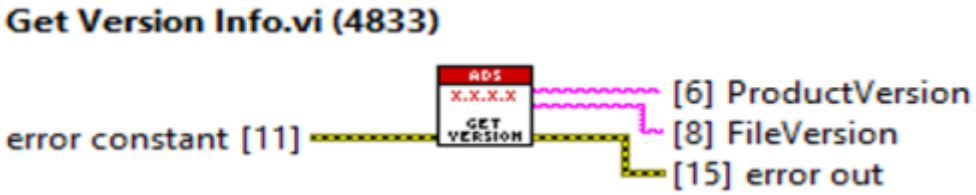
ADS Port	ADS State	Zustand
TC3 PLC Port > 851	ADS_STATE_RUN	Startet die PLC.
	ADS_STATE_STOP	Stoppt die PLC.
	ADS_STATE_RESET	Resettet die PLC.
System Port = 10000	ADS_STATE_RECONFIG	Führt TwinCAT in den Config-Modus.
	ADS_STATE_RESET	Führt TwinCAT in den Run-Modus.



Eingang/Ausgang	Bedeutung
[0] Handle	Das Handle auf den Client
[5] Target	ADS-Target als Zeichenkette bestehend auf AMSNetId
[7] Device State	Neuer Zustand des Zielsystems (Numerisch)
[9] ADS State	Neuer Zustand der PLC (ENUM)
[4] Handle	Das Handle auf den Client
[6] Target	ADS-Target als Zeichenkette bestehend auf AMSNetId

Get Version Info

Der Block *Get Version Info* gibt die aktuelle Version des TF3710-Setups und der installierten TF3710 Bibliothek als LabVIEW™-Zeichenkette zurück.



Ausgang	Bedeutung
[6] ProductVersion	Setup-Version als Zeichenkette
[8] FileVersion	Version der installierten TF3710 Bibliothek als Zeichenkette

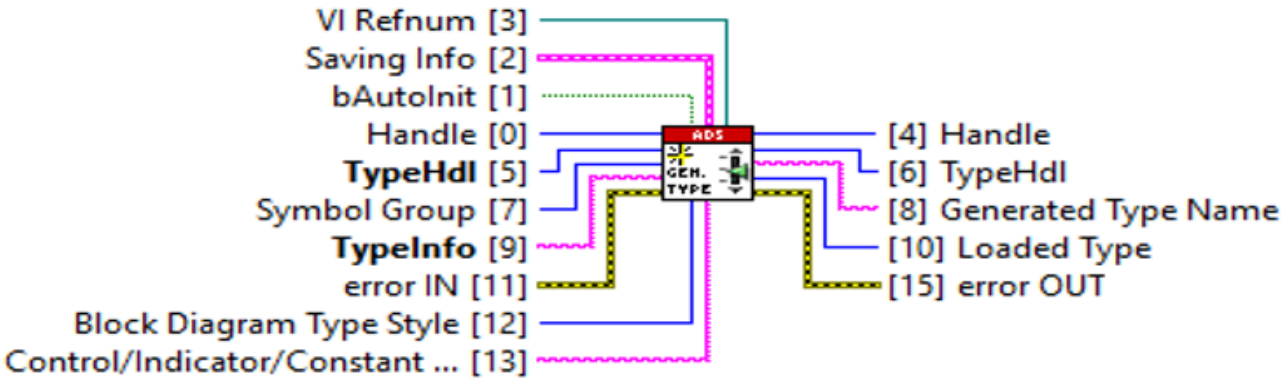
Generate Type

Der Block *Generate Type* erzeugt automatisch den LabVIEW™-Datentyp basierend auf einer TypeInfo-Beschreibung von einem TwinCAT-3-Datentyp.

Der LabVIEW™-Typ kann als Constant oder Steuer/Bedienelement auf einem Blockdiagramm eingefügt oder als LabVIEW™-ctl-Datei gespeichert werden. Der *Generate Type* Block ist ein Wrapper VI und nutzt Low-Level Klassen-Objekte von TypeGenerator [▶ 93] zusammen mit dem TypeResolver [▶ 91]. Es gibt verschiedene Möglichkeiten, wie der Wrapper Block parametrisiert werden kann.

1. **Der TypeResolver ist schon von außen initialisiert und LabVIEW™ hat schon ein TypeHandle. Der bAutoInit flag hat den Wert „False“**:
   - Der TypeResolver hat schon Informationen bezüglich des TwinCAT 3-Datentyps.
   - Der Wrapper Block kann den neuen Typ generieren.
   - Das Beispiel Generate Type using Symbol Interface or Symbol's file [▶ 110] nutzt dieses Verfahren.
2. **Der TypeResolver ist nicht von außen initialisiert und LabVIEW™ hat keinen TypeHandle. Der bAutoInit flag hat den Wert „True“**:
   - Der TypeResolver hat keine Informationen bezüglich des TwinCAT 3-Datentyps.
   - Der Wrapper Block benötigt die TypeInfo um den Typ zu generieren.
   - Das Beispiel Generate Type using TypeInfo file [▶ 110] nutzt dieses Verfahren.

Generate Type.vi (4833)



Eingang/Ausgang	Bedeutung
[0] Handle	Das Handle auf den Client
[1] bAutoInit	Boolesche flag (True = TypeResolver wird intern initialisiert, sonst False)
[2] Saving Info	LabVIEW™-Cluster bestehend aus drei Elementen: • CustomDir: Boolesche flag TRUE = LabVIEW™-Typ wird in einem bestimmten Ordner gespeichert (Dir Name). Wenn es diesen Ordner nicht gibt, erzeugt der Block den Ordner. FALSE = LabVIEW™-Typ wird im Projektordner, Unterordner Ctl, gespeichert. • Dir Name: LabVIEW™-Zeichenkette (Name des Ordners) • Control FileName: LabVIEW™-Zeichenkette (Name der neuen Ctl. Wenn leer, nutzt der Block den Namen aus der TypeInfo)
[3] VI Refnum	Eine statische VI Refnum Der Block nutzt die VI Refnum, um den LabVIEW™-Typ als Constant/ Control/Indicator auf dem Blockdiagramm einzufügen.
[5] TypeHdl	Das Handle auf den TypeResolver
[7] Symbol Group	LabVIEW™-Enum: • Read Symbol: Der Block generiert ein Bedienelement. • Write Symbol: Der Block generiert ein Steuerelement.
[9] TypeInfo	LabVIEW™-Zeichenkette als XML-Beschreibung vom TwinCAT 3-Datentyp
[12] Block Diagram Type Stype	LabVIEW™-Enum: • Constant • Control/Indicator
[13] Control/Indicator/ Constant Name	LabVIEW™-Zeichenkette. Leere Zeichenkette: Default Name der TypeInfo. Sonst: Name des Control/Indicator/Constant.
[8] Generated Type Name	LabVIEW™-Zeichenkette als Name von dem generierten Typ
[10] Loaded Type	LabVIEW™-Enum Beschreibt welcher Datentyp generiert wurde.

Parameter “Symbol Group”

Mit dem Parameter Symbol Group kann ausgewählt werden, ob ein Steuer-/Anzeige-Element generiert werden muss. Für das Steuerelement generiert der TypeGenerator den Typ ohne Namen, da der TypeResolver das Steuerelement ohne Namen unterstützt. Wenn das Steuerelement ein LabVIEW™-Cluster ist, dann werden die Cluster Member auch ohne Namen generiert.

7.10.1 Notification

Im Ordner *Notification* befinden sich zusätzliche Blöcke, die beim Lesen mit ADS-Notifications genutzt werden können. Der Ordner enthält die folgenden VIs:

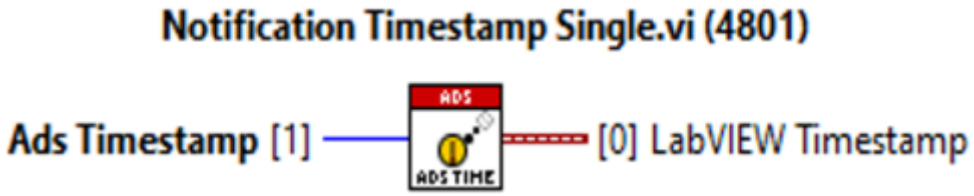
- ADS To LabVIEW Timestamp
- Notification Data To Variant Array
- Stop Notification
- Unregister Notification

ADS To LabVIEW Timestamp

Der Block *ADS To LabVIEW Timestamp* ist ein polymorpohic VI und unterstützt einzelne und mehrere ADS-Zeitstempel (als array). Der Block konvertiert ADS-Zeitstempel, so wie sie durch eine ADS-Notification von TwinCAT übertragen werden, in LabVIEW™-Zeitstempel.

Notification Timestamp Single

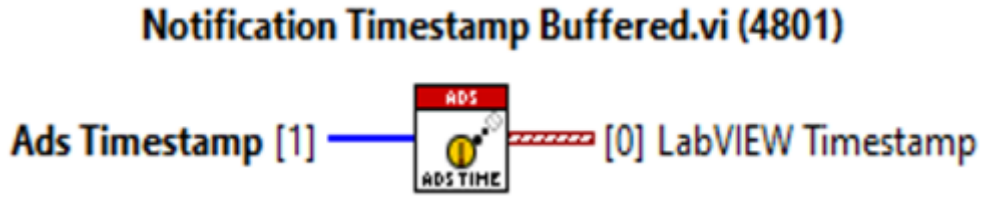
Der Block *Notification Timestamp Single* konvertiert einzelne ADS-Zeitstempel in LabVIEW™-Zeitstempel.



Eingang/Ausgang	Bedeutung
[1] ADS Timestamp	Einzelner ADS-Zeitstempel
[0] LabVIEW Timestamp	Einzelner konvertierter LabVIEW™-Zeitstempel

Notification Timestamp Buffered

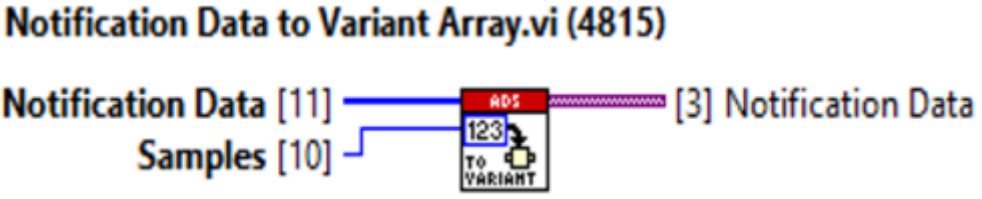
Der Block *Notification Timestamp Buffered* konvertiert mehrere ADS-Zeitstempel in LabVIEW™-Zeitstempel.



Eingang/Ausgang	Bedeutung
[1] ADS Timestamp	Array von ADS-Zeitstempeln
[0] LabVIEW Timestamp	Array von konvertierten LabVIEW™-Zeitstempeln

Notification Data To Variant Array

Der Block *Notification Data To Variant Array* konvertiert ADS-Rohdaten, die mit ADS-Notification gelesen worden sind, in ein passendes LabVIEW™ Variant.

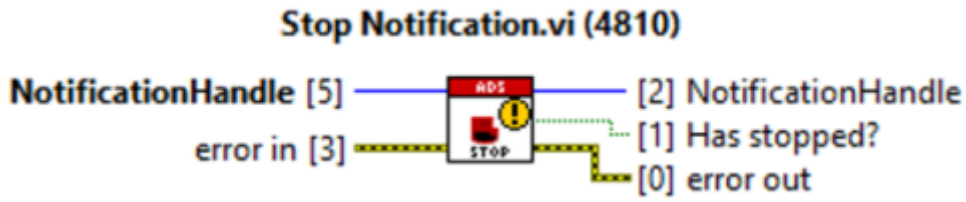


Eingang/Ausgang	Bedeutung
[11] Notification Data	ADS-Rohdaten
[10] Samples	Anzahl von Samples

Eingang/Ausgang	Bedeutung
[3] Notification Data	Konvertierte Rohdaten in LabVIEW™-Variant

Stop Notification

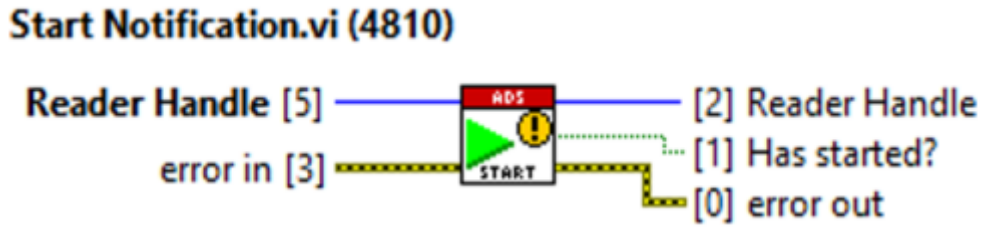
Der Block *Stop Notification* stoppt das Empfangen von ADS-Notifications.



Eingang/Ausgang	Bedeutung
[5] Notification-/Reader Handle	Handle auf die ADS Notification/Reader
[2] Notification-/Reader Handle	Handle auf die ADS-Notification/ADS-Reader
[1] Has stopped?	Boolesche Flag (TRUE, wenn Notification gestoppt ist, sonst FALSE.)

Start Notification

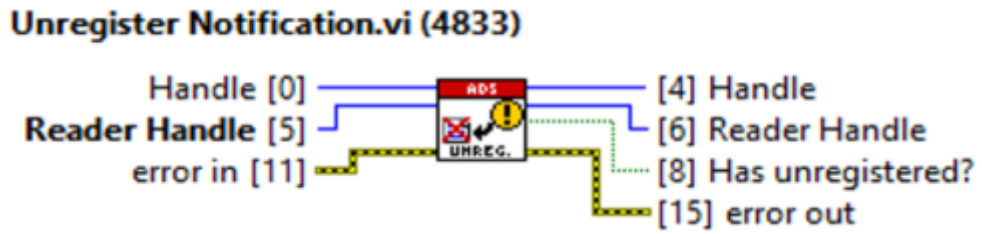
Der Block *Start Notification* startet die angemeldete Notification.



Eingang/Ausgang	Bedeutung
[5] Reader Handle	Handle auf den ADS-Reader
[2] Reader Handle	Handle auf den ADS-Reader
[1] Has started?	Boolesche Flag (TRUE, wenn Notification gestartet ist, sonst FALSE.)

Unregister Notification

Der Block *Unregister Notification* meldet die Notification am ADS-Server ab.



Eingang/Ausgang	Bedeutung
[0] Handle	Das Handle auf den Client (kann auch frei gelassen werden)
[5] Reader Handle	Das Handle auf den ADS-Reader
[4] Handle	Das Handle auf den Client
[6] Reader Handle	Das Handle auf den ADS-Reader
[8] Has unregistered?	Boolesche Flag (TRUE, wenn Notification registriert ist, sonst FALSE.)

7.10.1.1 Notification Steuer-Elemente

Zu finden sind diese Steuerelemente unter:
Front-Panel Palette > User Steuerelemente > Beckhoff-LabVIEW-Interface > Notification.

Single Buffer Info

Das Steuerelement *Single Buffer Info* beschreibt/definiert die Informationen, die für das Lesen des Noti. Buffered [▶ 69] benötigt werden.

Single User-Event Data

Das Steuerelement *Single User-Event Data* beschreibt/definiert Informationen, die für das kontinuierliche Lesen des E-Noti. Single [▶ 70] mit LabVIEW™-Events benötigt werden.

Buffered User-Event Data

Das Steuerelement *Buffered User-Event Data* beschreibt/definiert Information, die für das kontinuierliche Lesen des E-Noti. Buffered [▶ 70] mit LabVIEW™-Events benötigt werden.

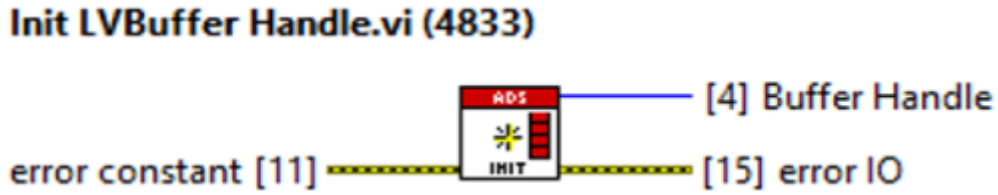
7.10.2 LVBuffer

Im Ordner LVBuffer befinden sich Blöcke, die beim Lesen einer ADS-Notification genutzt werden können. Der Ordner enthält foldende VIs:

- Init LVBuffer Handle
- Read From LVBuffer
- LVBuffer Status
- Release LVBuffer Handle

Init LVBuffer Handle

Der *Init LVBuffer Handle* Block initialisiert ein Handle auf dem LVBuffer.

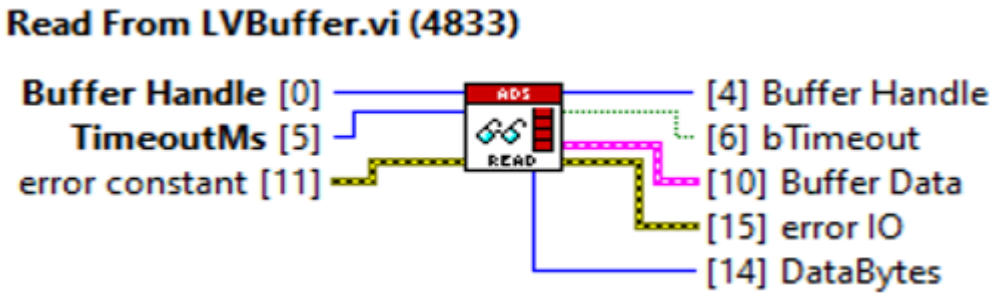


Ausgang	Bedeutung
[4] Buffer Handle	Das Handle auf den LVBuffer

Read From LVBuffer

Der *Read from LVBuffer* Block wartet auf Samples im LVBuffer (LabVIEW™-seitiger Datenpuffer, vgl. Kommunikations-Modi [▶ 25]). Der TimeoutMs beeinflusst das Warten. Der Block wartet für eine definierte Zeit, wenn TimeoutMs > 0 ist, sonst wartet der Block für immer. Bekommt der LVBuffer während des Wartens ein Sample, liest der Block das Sample und gibt es weiter an LabVIEW™.

Relevante Parameter: LVBufferSize

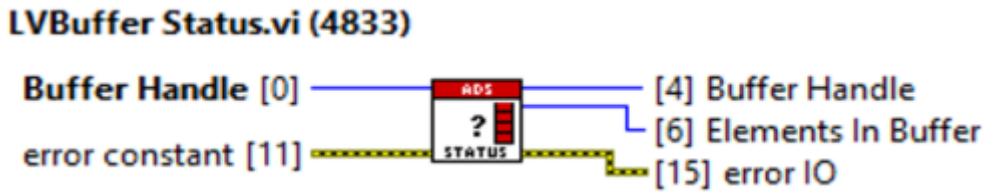


Eingang/Ausgang	Bedeutung
[0] Buffer Handle	Das Handle auf den LVBuffer

Eingang/Ausgang	Bedeutung
[5] TimeoutMs	Die Wartezeit in Millisekunden: • TimeoutMs>0: *Read From Buffer* Block wartet für die definierte Zeit. • TimeoutMs<0: *Read From Buffer* Block wartet für immer.
[4] Buffer Handle	Das Handle auf den LVBuffer
[6] bTimeout	Boolesche flag: • True: Wenn in der definierten Zeit kein Sample im LVBuffer liegt. • False: LVBuffer bekommt kontinuierlich neue Samples und *Read From Buffer* Block kann diese lesen.
[10] Buffer Data	Samples aus dem LVBuffer
[14] DataBytes	Anzahl von Bytes in LVBuffer

LVBuffer Status

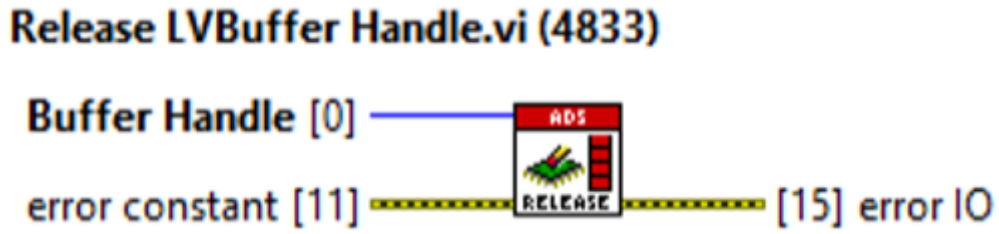
Der *LVBuffer Status* Block gibt den aktuellen Zustand des LVBuffer bezüglich Samples im LVBuffer zurück.



Eingang/Ausgang	Bedeutung
[0] Buffer Handle	Das Handle auf den LVBuffer
[4] Buffer Handle	Das Handle auf den LVBuffer
[6] Elements In Buffer	Anzahl von Samples in LVBuffer, die noch zu lesen sind.

Release LVBuffer Handle

Der Release LVBuffer Handle Block gibt das Handle auf den LVBuffer aus dem Speicher frei.



Eingang	Bedeutung
[0] Buffer Handle	Das Handle auf den LVBuffer

7.10.3 CoE

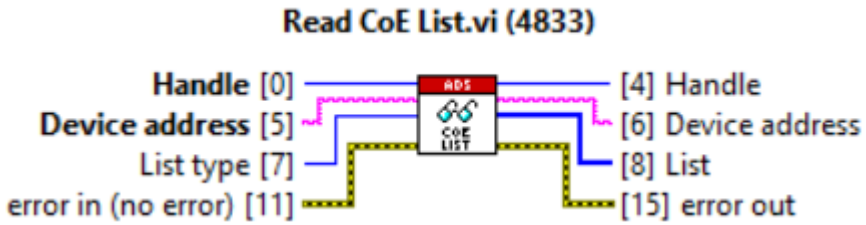
Im Abschnitt CoE (**C**ANopen **o**ver **E**therCAT) befinden sich Blöcke, die das Lesen und Schreiben der CoE-Objekte ermöglichen. Hierbei wird die AMS-Adresse des EtherCAT Teilnehmers genutzt. Die Adresse setzt sich zusammen aus Master AMS NetId und AMS Port des Clients. Der Order enthält die folgenden VIs:

- Read CoE List
- Read CoE Description
- Read CoE Entry
- Read CoE Value
- Write CoE Value

Das Beispiel CoE Lesen oder Schreiben [▶ 112] beschreibt die Nutzung der CoE Blöcke.

Read CoE List

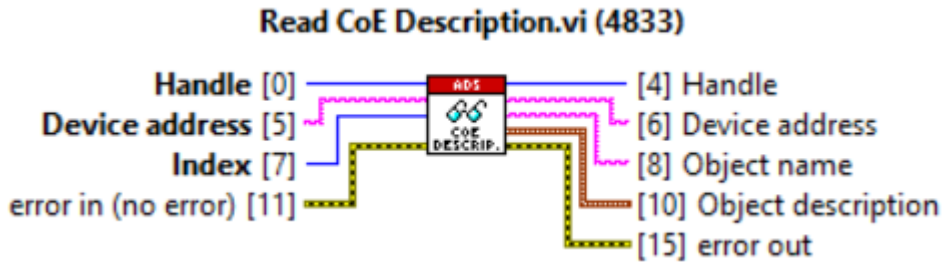
Das VI *Read CoE List* liest ein CoE-Verzeichnis eines Teilnehmers aus und listet alle Objekte in einem Array auf, die für das ausgewählte Gerät verfügbar sind. Die Objekte werden über Indizes identifiziert, die von nachfolgenden CoE VIs für den Zugriff verwendet werden.



Eingang/Ausgang	Bedeutung
[0][4] Handle	Handle auf den ADS-Client
[5][6] DeviceAddress	AMS-Adresse des Teilnehmers bestehend aus: • AMS NetId des Master • AMS Port des Clients
[7] ListType	LabVIEW™ Enum Beschreibt, welche Indizes aus dem Verzeichnis aufgelistet werden sollen: • TotalNumberOfLists: Listet die Anzahl der Elemente in den Verzeichnissen AllCoEObjects, RxPDOs, TxPDOs, StoredForDevice und StartUp auf. • AllCoEObjects: Liste aller CoE-Objekte • RxPDOs: Liste aller RxPDO-Objekte • TxPDOs: Liste aller TxPDO-Objekte • SettingObjects: Liste aller Objekte, die bei einem Austausch des Geräts relevant sind. • StartUp: Liste der Objekte, die als StartUp-Parameter genutzt werden können.
[8] List	1D-Array der Indizes der Objekte

Read CoE Description

Das VI *Read CoE Description* ruft die CoE-Objektbeschreibung des Teilnehmers auf und sendet diese an LabVIEW™. Die Beschreibung beinhaltet unter anderem den Objektnamen und die Anzahl der Einträge bzw. Subindizes des Objekts.

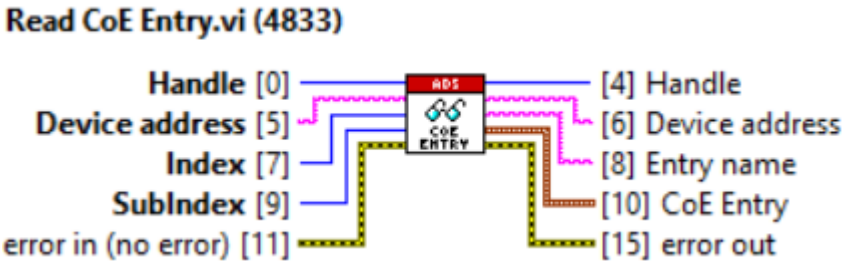


Eingang/Ausgang	Bedeutung
[0][4] Handle	Handle auf den ADS-Client
[5][6] DeviceAddress	AMS Adresse des Teilnehmers bestehend aus: • AMS NetId des Masters • AMS Port des Clients
[7] Index	Index des Objekts
[8] ObjectName	Name des Objekts
[10] ObjectDescription	LabVIEW™ Cluster bestehend aus folgenden Elementen:

Eingang/Ausgang	Bedeutung
	• Index: Index des Objekts • DataType: Datentyp des Objekts • MaxSubIndex: Anzahl von Subindizes des Objekts

Read CoE Entry

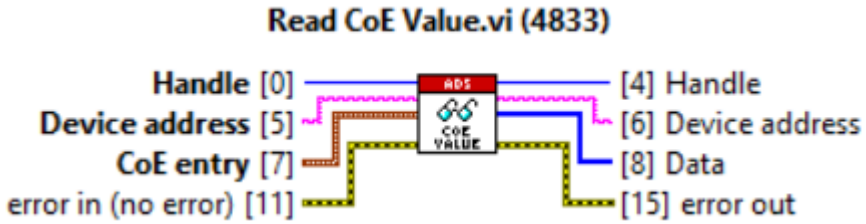
Das VI *Read CoE Entry* liest einen CoE-Eintrag eines Objekts aus. Das Objekts wird über den Index und Subindex referenziert. Zurückgegeben werden Informationen, wie z. B. der Name des Eintrags und der Zugriff.



Eingang/Ausgang	Bedeutung
[0][4] Handle	Handle auf den ADS-Client
[5][6] DeviceAddress	AMS Adresse des Teilnehmers bestehend aus: • AMS NetId des Masters • AMS Port des Slaves
[7] Index	Index des Objekts
[8] Entry Name	Name des Eintrags
[9] SubIndex	Subindex des Eintrags
[10] CoE Entry	LabVIEW™ Cluster bestehend aus folgenden Elementen: • Index: Index des Objekts • SubIndex: Subindex des Eintrags • DataType: Datentyp des Eintrags • BitLength: Die Länge des Eintrags in Bits • ObjectAccess: Angaben zur Veränderbarkeit, z. B., ob nur gelesen oder auch geschrieben werden kann.

Read CoE Value

Das VI *Read CoE Value* liest den Wert eines CoE-Eintrags. Zurückgegeben wird ein Array der Länge BitLength in Bytes. Das Array gibt den Wert in hexadezimalen Zahlen im Format Little-Endian an.



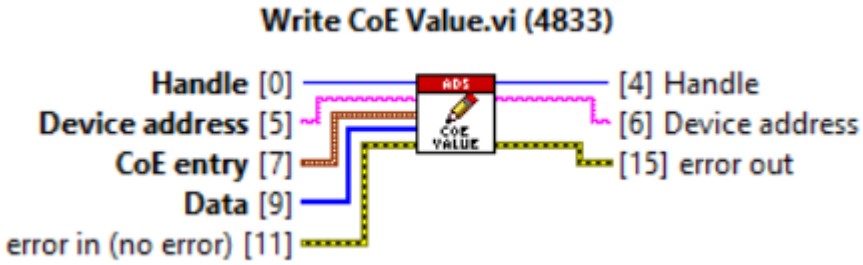
Eingang/Ausgang	Bedeutung
[0][4] Handle	Handle auf den ADS-Client
[5][6] DeviceAddress	AMS Adresse des Teilnehmers bestehend aus: • AMS NetId des Masters • AMS Port des Slaves
[7] CoE Entry	LabVIEW™ Cluster bestehend aus folgenden Elementen:

Eingang/Ausgang	Bedeutung
	• Index: Index des Objekts • SubIndex: Subindex des Eintrags • DataType: Datentyp des Eintrags • BitLength: Die Länge des Wertes des Eintrags in Bits • ObjectAccess: Beschreibt, in welchem EtherCAT-Zustand auf das Objekt zugegriffen werden kann.
[8] Data	1D Byte Array mit dem Wert des CoE-Eintrags

Der Wert des CoE-Eintrags wird hexadezimal dargestellt und die Bytes in der little-endian Reihenfolge übertragen. Zeichenketten und Arrays werden von links nach rechts übertragen, wobei die Elemente des Arrays im little-endian gespeichert sind.

Write CoE Value

Der Block *Write CoE Value* schreibt den Wert des CoE-Eintrags. Der zu schreibende Wert ist ein Array aus Bytes der Größe BitLength. Die Werte müssen in hexadezimalen Zahlen im Format Little-Endian angegeben werden.



[0][4] Handle	Handle auf den ADS-Client
[5][6] DeviceAddress	AMS Adresse des Teilnehmers bestehend aus: • AMS NetId des Masters • AMS Port des Slaves
[7] Object Entry	LabVIEW™ Cluster bestehend aus folgenden Elementen: • Index: Index des Objekts • SubIndex: SubIndex des Eintrags • DataType: Datentyp des Eintrags • BitLength: Die Länge des Wertes des Eintrags in Bits • ObjectAccess: Beschreibt, in welchem EtherCAT-Zustand auf das Objekt zugegriffen werden kann.
[9] Data	1D Byte Array mit dem Wert des CoE Eintrags

7.11 Low-Level

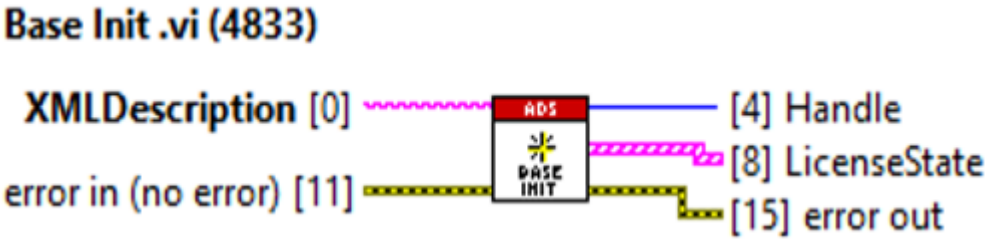
Low-Level VIs [▶ 41] können genutzt werden, um in bestimmten Situationen mehr Flexibilität in der Programmierung zu erhalten, oder die Lese- oder Schreibgeschwindigkeit zu erhöhen.

7.11.1 Init

Der Unterordner *Init* enthält Low-Level-Blöcke, die für das Initialisieren und Verbinden des ADS-Clients notwendig sind.

Base Init

Der Block *Base Init* initialisiert und baut eine Verbindung zwischen ADS-Client und ADS-Router auf. Der Block überprüft nach erfolgreicher Verbindung die Lizenzen auf den Zielsystemen. Nach erfolgreicher Überprüfung gibt der Block ein gültiges Handle auf den ADS-Client zurück.



Eingang/Ausgang	Bedeutung
[0] XML Description	LabVIEW™-XML-Zeichenkette mit ADS-Lese- und Schreib-Symbolen **oder** der Pfad als Zeichenkette zu einer vorhandenen, bereits erstellten (exportierten) XML Datei.
[4] Handle	Handle auf den ADS-Client
[8] LicenseState	Liste der Lizenzzustände der TwinCAT-Zielsysteme

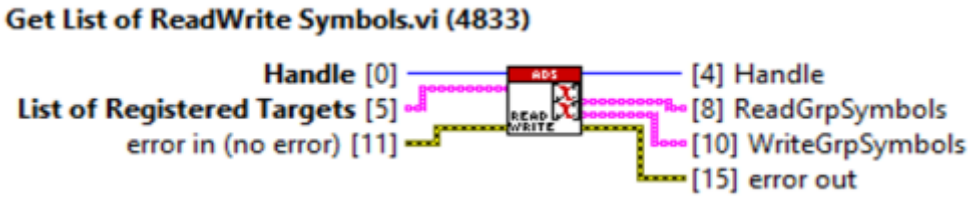
Get List of Registered Targets

Der Block *Get List of Registered Targets* erstellt eine Liste von ADS-Zielsystemen, welche vom Nutzer im Symbol Interface [▶ 65] gewählt worden oder in der direkt geladenen XML-Datei eingetragen sind.

Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[4] Handle	Handle auf den ADS-Client
[6] List of Registered Targets	Liste von angemeldeten Zielsystemen

Get List of ReadWrite Symbols

Der Block *Get List of ReadWrite Symbols* erstellt eine Liste von eingetragenen ADS-Lese-und Schreib-Symbolen, welche vom Nutzen im Symbol Interface [▶ 65] gewählt worden oder in der direkt geladenen XML-Datei eingetragen sind.



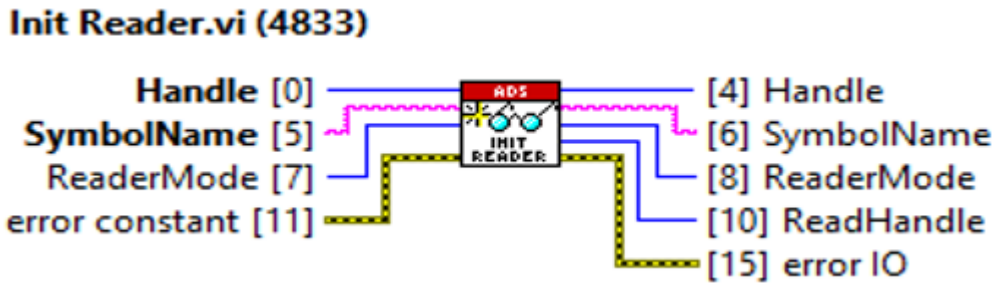
Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[5] List of Registered Targets	Liste von angemeldeten Zielsystemen
[4] Handle	Handle auf den ADS-Client
[8] ReadGrpSymbols	Liste von ADS-Lese-Symbolen
[10] WriteGrpSymbols	Liste von ADS-Schreib-Symbolen

7.11.2 Read

Der Unterordner *Read* enthält Low-Level Blöcke, welche für das Lesen via ADS notwendig sind.

Init Reader

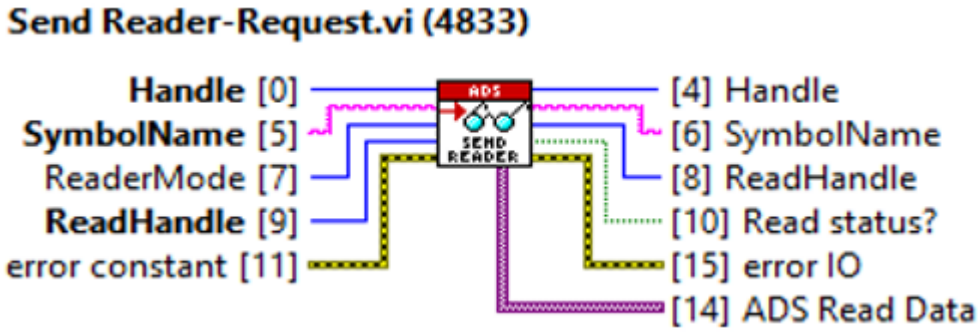
Der Block *Init Reader* initialisiert den ADS-Reader. Beim erfolgreichen Aufruf gibt der Block ein Handle auf den ADS-Reader zurück.



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[7] ReaderMode	Die Art des Lesens (ENUM: Sync/Async)
[4] Handle	Handle auf den ADS-Client
[6] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[8] ReaderMode	Die Art des Lesens (Sync/Async)
[10] ReadHandle	Handle auf den ADS-Reader

Send Reader-Request

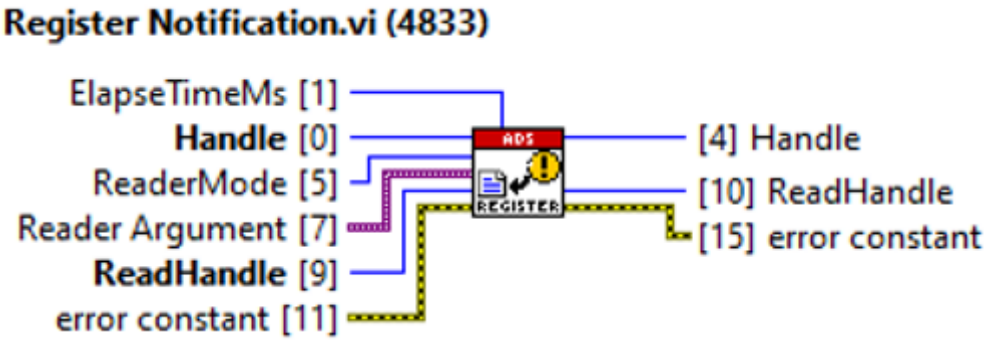
Der *Send Reader-Request* Block sendet eine Lese-Anfrage an den ADS-Server. Der Block wartet auf eine Antwort des Servers, wenn der Reader mit der Betriebsart „Synchron“ initialisiert worden ist. Sonst wartet der Block nicht auf die Antwort und gibt das ReadHandle weiter.



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[7] ReaderMode	Die Art des Lesens (ENUM: Sync/Async)
[9] ReadHandle	Handle auf den ADS-Reader
[4] Handle	Handle auf den ADS-Client
[6] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[8] ReadHandle	Handle auf den ADS-Reader
[10] Read status?	Lese-Status
[14] ADS-Read Data	ADS-Rohdaten

Register Notification

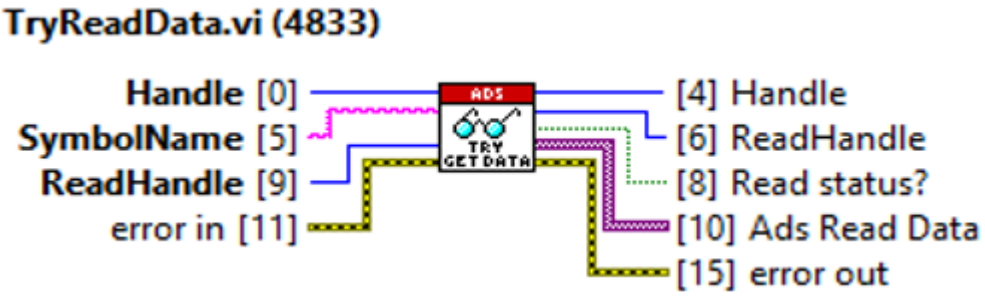
Der *Register Notification* Block meldet die ADS-Notification am ADS-Server an und wartet, bis die Notification von außen explizit abgemeldet (unregistered) wird. Die Anmeldung startet die Notifications nicht automatisch, sondern sie müssen explizit gestartet werden.



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[1] ElapseTimeMs	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[5] ReaderMode	Die Art des Lesens: LabVIEW™-ENUM: • Noti. Single [▶ 69] • Noti. Buffered [▶ 69] • E-Noti. Single [▶ 70] • E-Noti. Buffered [▶ 70] • LVB-Noti. Single Symbol [▶ 72]
[7] Reader Argument	Reader Argumente variieren mit dem ReaderMode: • Noti. Single: kein Argument • Noti. Buffered: Single Buffer Info [▶ 81] • E-Noti. Single: LabVIEW™ Event Ref auf Single User-Event Data [▶ 81] • E-Noti.Buffered: LabVIEW™ Event Ref auf Buffered User-Event Data [▶ 81] • LVB-Noti.Single Symbol: Handle auf LVBuffer auf Typ (UINT32)
[9] ReadHandle	Handle auf den ADS-Reader
[4] Handle	Handle auf den ADS-Client
[10] ReadHandle	Handle auf den ADS-Reader

TryReadData

Der Block *TryReadData* prüft im Zusammenhang mit der Betriebsart „Asynchron“ auf das erfolgreiche Erhalten einer Antwort (ADS-Datenpaket) vom ADS-Server.

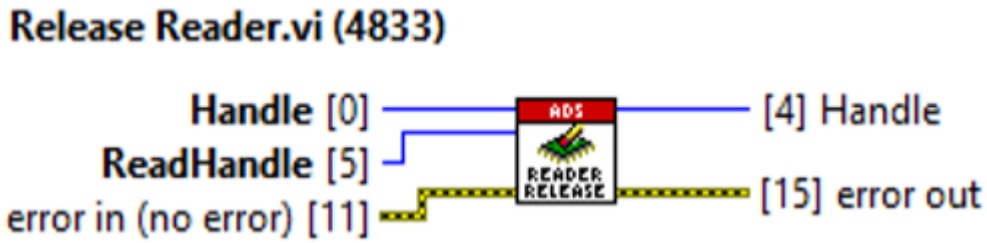


Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client

Eingang/Ausgang	Bedeutung
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[9] ReadHandle	Handle auf den ADS-Reader
[4] Handle	Handle auf den ADS-Client
[6] ReadHandle	Handle auf den ADS-Reader
[8] Read status?	Lese-Status
[10] ADS-Read Data	ADS-Rohdaten

Release Reader

Der *Release Reader* gibt das Handle auf den Reader aus dem Speicher frei.



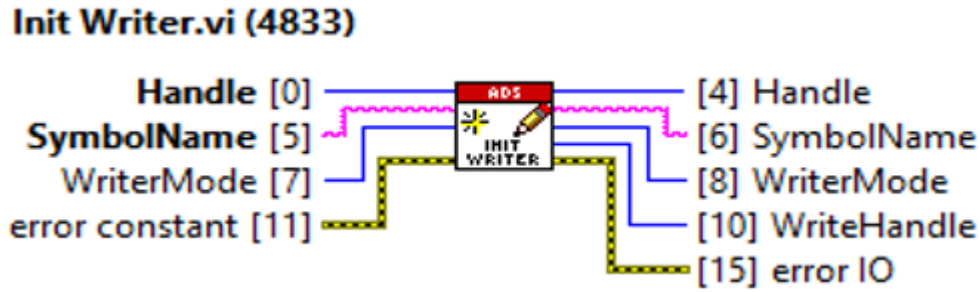
Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[5] ReadHandle	Handle auf den ADS-Reader
[4] Handle	Handle auf den ADS-Client

7.11.3 Write

Der Unterordner *Write* enthält Low-Level Blöcke, die für das Schreiben via ADS notwendig sind.

Init Writer

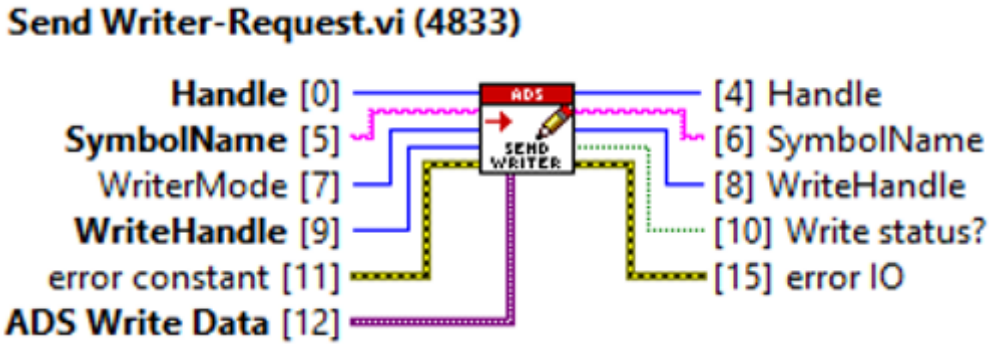
Der Block *Init Writer* initialisiert den ADS-Writer. Beim erfolgreichen Aufruf gibt der Block ein Handle auf den ADS-Writer zurück.



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[7] WriterMode	Art des Schreibens (Sync/Async)
[4] Handle	Handle auf den ADS-Client
[6] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[8] WriterMode	Art des Schreibens (Sync/Async)
[10] WriteHandle	Handle auf den ADS-Writer

Send Writer-Request

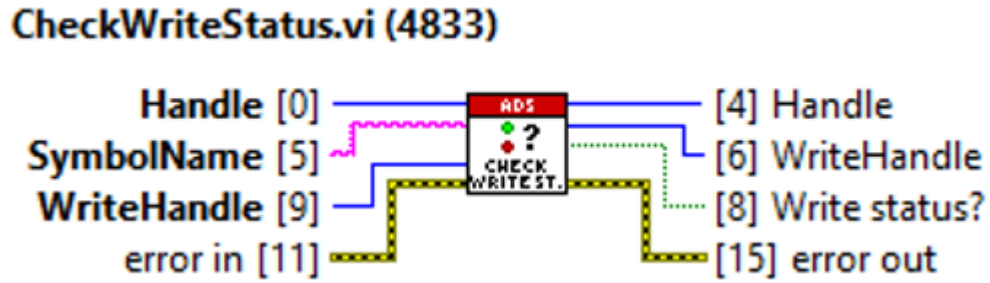
Der Block *Send Writer-Request* sendet eine Schreib-Anfrage an den ADS-Server. Der Block wartet auf eine Antwort des Servers, wenn der Writer mit der Betriebsart „Synchron“ initialisiert worden ist. Sonst wartet der Block nicht auf die Antwort und gibt das WriteHandle weiter.



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[7] WriterMode	Art des Schreibens (Sync/Async)
[9] WriteHandle	Handle auf den ADS-Writer
[12] ADS-Write Data	TypeResolved ADS-Datenpaket
[4] Handle	Handle auf den ADS-Client
[6] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[8] WriteHandle	Handle auf den ADS-Writer
[10] Write status?	Schreib-Status

CheckWriteStatus

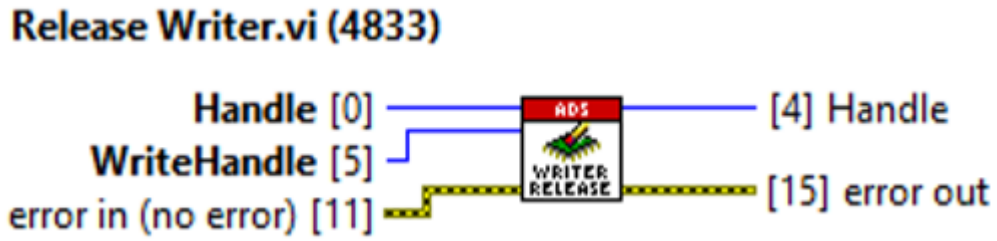
Der Block *CheckWriteStatus* prüft im Zusammenhang mit der Betriebsart „Asynchron“ den erfolgreichen Schreibzugriff auf den ADS-Server.



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[9] WriteHandle	Handle auf den ADS-Writer
[4] Handle	Handle auf den ADS-Client
[6] WriteHandle	Handle auf den ADS-Writer
[8] Write status?	Schreib-Status

Release Writer

Der *Release Writer* gibt das Handle auf den Writer aus dem Speicher frei.



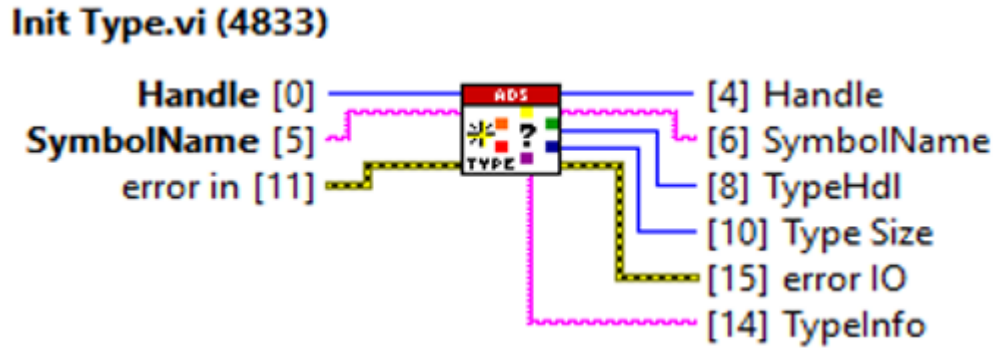
Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[5] WriteHandle	Handle auf den ADS-Writer
[4] Handle	Handle auf den ADS-Client

7.11.4 TypeResolver

Der Unterordner *TypeResolver* enthält Low-Level-Blöcke, die für das Konvertieren und den Vergleich zwischen dem LabVIEW™-Datentyp und dem TC3-Datentyp notwendig sind.

Init Type

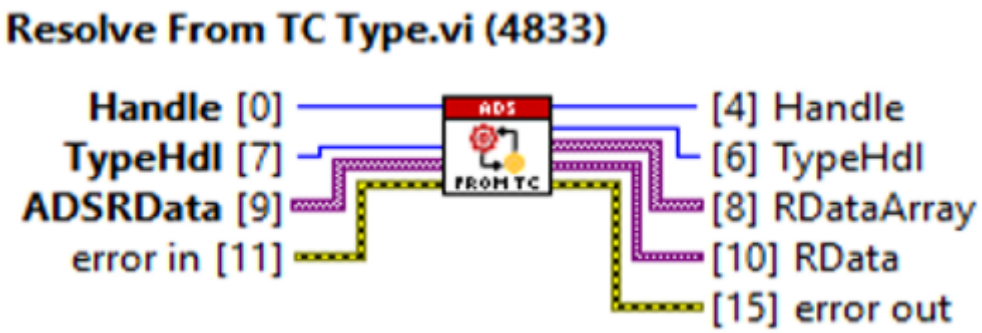
Der Block *Init Type* initialisiert den TypeResolver basierend auf SymbolName und Handle. Bei erfolgreicher Initialisierung gibt der Block das Handle auf den TypeResolver und den TC3-Datentyp des ADS-Symbols als LabVIEW™-Zeichenkette in XML-Beschreibung an den LabVIEW™-Prozess weiter.



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[4] Handle	Handle auf den ADS-Client
[6] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[8] TypeHdl	Handle auf den TypeResolver
[10] Type Size	Datentyp-Größe in Bytes
[14] TypeInfo	Typ Beschreibung in XML als LabVIEW™-Zeichenkette

Resolve From TC Type

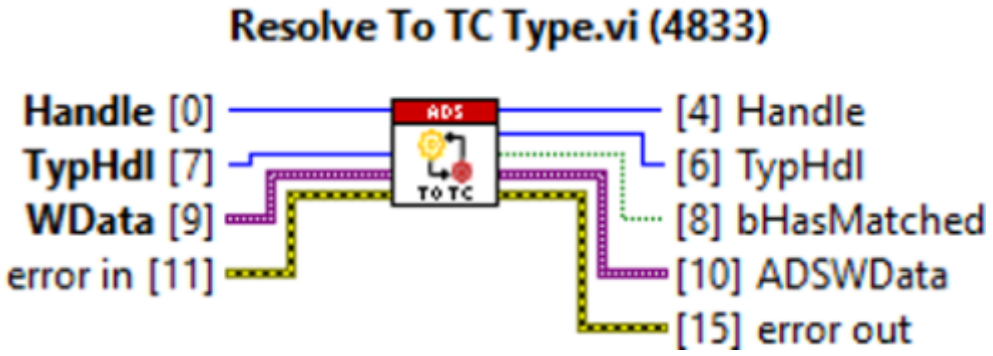
Der Block *Resolve From TC Type* vergleicht und konvertiert die Rohdaten aus dem ADS-Read in den entsprechenden LabVIEW™-Datentyp „Variant“. Die Konvertierung erfolgt nur dann, wenn der Vergleich zwischen beiden Datentypen erfolgreich war.



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[7] TypeHdl	Handle auf den TypeResolver
[9] ADSRData	ADS-Datenpaket als Rohdaten
[4] Handle	Handle auf den ADS-Client
[6] TypeHdl	Handle auf den TypeResolver
[8] RDataArray	TypeResolved ADS-Rohdaten als Variant Array
[10] RData	TypeResolved ADS-Rohdaten als Variant

Resolve To TC Type

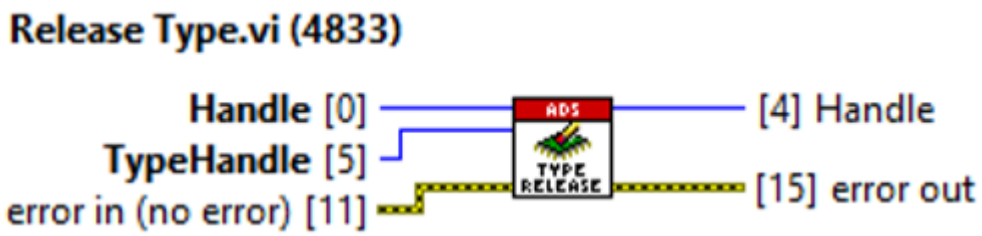
Der Block *Resolve To TC Type* konvertiert die Rohdaten für ADS-Read von einem LabVIEW™-Datentyp „Variant“ in den passenden TC3-Datentyp. Die Konvertierung erfolgt nur dann, wenn der Vergleich zwischen beiden Datentypen erfolgreich war.



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[7] TypeHdl	Handle auf den TypeResolver
[9] WData	ADS-Datenpaket als Rohdaten
[4] Handle	Handle auf den ADS-Client
[6] TypeHdl	Handle auf den TypeResolver
[8] bHasMatched	Flag (TRUE, wenn TC3- und LabVIEW™-Datentyp identisch sind, sonst FALSE)
[10] ADSWData	TypeResolved ADS-Rohdaten als Variant

Release Type

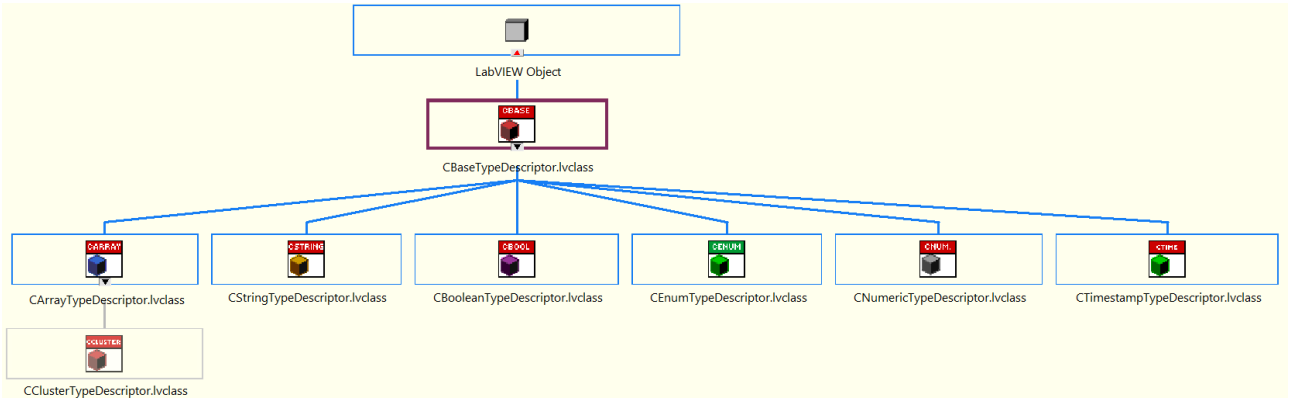
Der *Release Type* gibt das Handle auf den TypeResolver aus dem Speicher frei.



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf denADS-Client
[5] TypeHandle	Handle auf den TypeResolver
[4] Handle	Handle auf den Client

7.11.4.1 TypeGenerator

Der Unterordner *TypeGenerator* enthält LabVIEW™-Klassen, die für das Generieren eines unterstützten LabVIEW™-Datentypen [▶ 118] verantwortlich sind. Folgende Grafik zeigt die Beziehung bzw. Vererbung zwischen den *TypeGenerator*-Klassen.



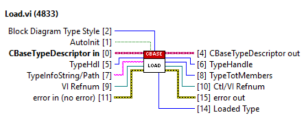
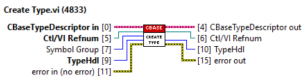
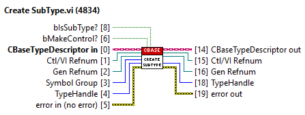
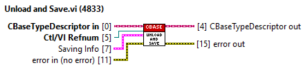
Nutzung der Klassen

Die TypeGnerator-Klassen der jeweiligen Datentypen wurden von der Base-Klasse abgeleitet und haben die Funktionen geerbt. Sehen Sie dafür auch die Informationen zum Wrapper Block Utilities [▶ 77].

CBaseTypeDescriptor Klasse

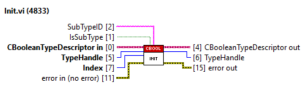
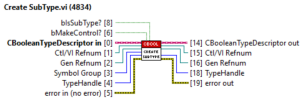
Die CBaseTypeDescriptor Klasse ist die oberste Klasse in der Hierarchie. Diese Klasse stellt allgemeine Methoden und Eigenschaften für die anderen Klassen, die davon erben, zur Verfügung. Die Tabelle stellt allgemeine Informationen zu den Public-Eigenschaften und -Methoden dar. Die LabVIEW™-Blöcke stellen weitere detaillierte Informationen zur Verfügung.

VI	Eigenschaft/Methode	Zugriff (Scope)	Bedeutung
	Read m_TCName	Public	Liest die Eigenschaft m_TCName (Name des geladenen Typen)
	Write m_TCName	Public	Schreibt die Eigenschaft m_TCName (Name des geladenen Typen)
	Read m_TypeCode	Public	Liest die Eigenschaft m_TypeCode (Definiert en numerischen Wert für den geladenen Typ)
	Read m_TypeID	Public	Liest die Eigenschaft m_TypeID (ID als GUID des geladenen Typ)

VI	Eigenschaft/Methode	Zugriff (Scope)	Bedeutung
	Read m_TypeStyle	Public	Liest die Eigenschaft m_TypeStyle (LabVIEW™-Enum: Beschreibt, wie der geladene Typ generiert werden soll.)
	Write m_TypeStyle	Public	Schreibt die Eigenschaft m_TypeStyle (LabVIEW™-Enum: Beschreibt, wie der geladene Typ generiert werden soll.)
	Load	Public, Static Dispatch	Lädt TwinCAT3 Typ-Information in den Speicher, basierend auf TypeResolver oder **TypeInfo.**
	Create Type	Public, Dynamic Dispatch	Die Methode ist leer. Die Klassen, die von dieser Klasse erben, implementieren diese Methode.
	Create SubType	Public, Dynamic Dispatch	Die Methode ist leer. Die Klassen, die von dieser Klasse erben, implementieren diese Methode.
	Unload and Save	Public, Static Dispatch	Entlädt den geladenen Typ aus dem Speicher und speichert den neuen Typ (wenn Ctl).

CBooleanTypeDescriptor Klasse

Die CBooleanTypeDescriptor Klasse generiert einen booleschen Typ als Constant oder Steuer/ Bedienelemente (in einem Blockdiagramm) oder als ctl auf der Festplatte. Die Tabelle stellt allgemeine Informationen zu Public-Methoden dar. Die LabVIEW™-Blöcke stellen weitere detaillierte Informationen zur Verfügung.

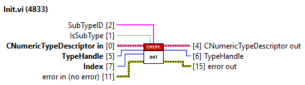
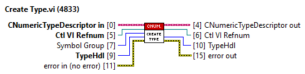
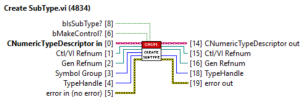
	Methode	Zugriff (Scope)	Bedeutung
	Init	Public, Static Dispatch	Initialisiert einen Booleschen Typ, basierend auf der geladenen TypeInfo.
	Create Type	Public, Static Dispatch	Generiert einen Booleschen Typ, basierend auf der geladenen TwinCAT 3 Typ-Informationen.
	Create SubType	Public, Static Dispatch	Generiert einen Booleschen Typ als Sub-Typ, z. B das Element eines Cluster.

IEC 61131-3 BIT Datentyp

Die CBooleanTypeDescriptor Klasse unterstützt BOOL- und BIT-Datentypen und kann daher auch für das Genenieren von beiden genutzt werden.

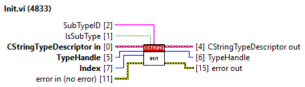
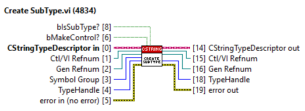
CNumericTypeDescriptor Klasse

Die CNumericTypeDescriptor Klasse generiert einen numerischen Typ mit korrekter Darstellung (I8, I16, I32, Single, …) als Constant oder Steuer/Bedienelemente (in einem Blockdiagramm) oder als ctl auf der Festplatte. Die Tabelle stellt allgemeine Informationen zu Public-Methoden dar. Die LabVIEW™-Blöcke stellen weitere detaillierte Informationen zur Verfügung.

	Methode	Zugriff (Scope)	Bedeutung
	Init	Public, Static Dispatch	Initialisiert den numerischen Typ, basierend auf der geladenen TypeInfo.
	Create Type	Public, Static Dispatch	Generiert einen numerischen Typ, basierend auf der geladenen TypeInfo.
	Create SubType	Public, Static Dispatch	Generiert einen numerischen Typ als Sub-Typ, z. B das Element eines Clusters.

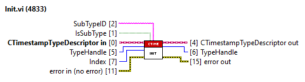
CStringTypeDescriptor Klasse

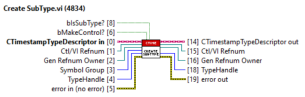
Die CStringTypeDescriptor Klasse generiert eine LabVIEW™-Zeichenkette als Constant oder Steuer/Bedienelemente (in einem Blockdiagramm) oder als ctl auf der Festplatte. Die Tabelle stellt allgemeine Informationen zu Public-Methoden dar. Die LabVIEW™-Blöcke stellen weitere detaillierte Informationen zur Verfügung.

	Eigenschaft/Methode	Zugriff (Scope)	Bedeutung
	Read m_Length	Public	Liest die Eigenschaft m_Length (Länge der TwinCAT 3-Zeichenkette der geladenen TypeInfo).
	Init	Public, Static Dispatch	Initialisiert eine LabVIEW™ Zeichenkette, basierend auf der geladenen TypeInfo.
	Create Type	Public, Static Dispatch	Generiert eine LabVIEW™-Zeichenkette, basierend auf geladene TypeInfo.
	Create SubType	Public, Static Dispatch	Generiert eine LabVIEW™-Zeichenkette als Sub-Typ, z. B ein Element eines LabVIEW™ Clusters.

CTimestampTypeDescriptor Klasse

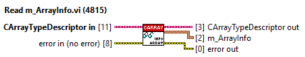
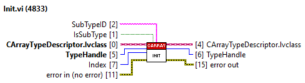
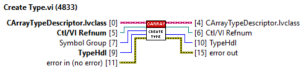
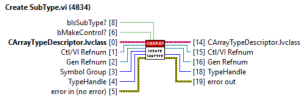
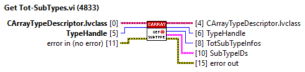
Die CTimestampTypeDescriptor Klasse generiert einen Zeitstempel LabVIEW™-Typ als Constant oder Steuer/Bedienelemente (in einem Blockdiagramm) oder als ctl auf der Festplatte. Die Tabelle stellt allgemeine Informationen zu Public-Methoden dar. Die LabVIEW™-Blöcke stellen weitere detaillierte Informationen zur Verfügung.

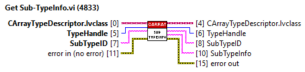
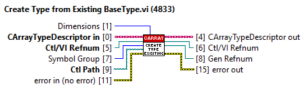
	Methode	Zugriff (Scope)	Bedeutung
	Init	Public, Static Dispatch	Initialisiert den Zeitstempel, basierend auf der geladenen TypeInfo.

	Methode	Zugriff (Scope)	Bedeutung
	Create Type	Public, Static Dispatch	Generiert einen Zeitstempel, basierend auf der geladenen TypeInfo.
	Create SubType	Public, Static Dispatch	Generiert einen Zeitstempel als Sub-Typ, z. B ein Element eines LabVIEW™-Clusters.

CArrayTypeDescriptor Klasse

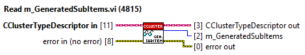
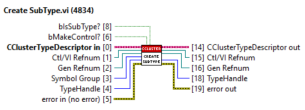
Die CArrayTypeDescriptor Klasse generiert einen Array LabVIEW™-Typ als Constant oder Steuer/ Bedienelemente (in einem Blockdiagramm) oder als ctl auf der Festplatte. Die Tabelle stellt allgemeine Informationen zu Public-Methoden dar. Die LabVIEW™-Blöcke stellen weitere detaillierte Informationen zur Verfügung.

	Eigenschaft/Methode	Zugriff (Scope)	Bedeutung
	Read m_ArrayInfo	Public	Liest die Eigenschaft m_ArrayInfo (Informationen bezüglich TwinCAT 3 Array Dimension Elemente)
	Read m_BaseTypeID	Public	Liest die Eigenschaft m_BaseTypeID (ID als GUID des geladenen Base-Typen)
	Read m_Dimensions	Public	Liest die Eigenschaft m_Dimensions (Anzahl von TwinCAT 3 Array Dimensions)
	Read m_SubTypes	Public	Liest die Eigenschaft m_SubTypes (Für Arrays von LabVIEW™-Clustern gibt diese Eigenschaft die Anzahl der generierten Cluster-Elemente an)
	Init	Public, Static Dispatch	Initialisiert das Array, basierend auf der geladenen TypeInfo.
	Create Type	Public, Static Dispatch	Generiert ein Array, basierend auf der geladenen TypeInfo.
	Create SubType	Public, Static Dispatch	Generiert ein Array als Sub-Typ, z. B. ein Element eines LabVIEW™ Clusters.
	Get Tot-SubTypes	Public, Static Dispatch	Liefert Informationen bezüglich Anzahl der komplexen Sub-Typen des Arrays sowie deren IDs als GUID (z. B.: Array mit komplexem BaseType, wobei der BaseType weitere Arrays mit komplexen BaseTypes enthält).

	Eigenschaft/Methode	Zugriff (Scope)	Bedeutung
	Get Sub-TypeInfo	Public, Static Dispatch	Liest TypeInfo des komplexen BaseType bzw. SubType mit eingegeber ID als GUID.
	Create Type From Existing BaseType	Public, Static Dispatch	Generiert ein Array von einem ctl aus einem Pfad.

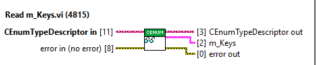
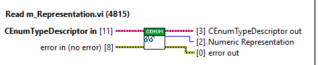
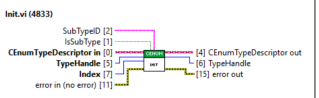
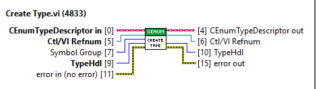
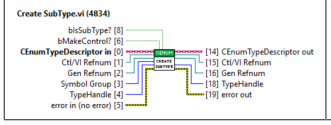
CClusterTypeDescriptor Klasse

Die CClusterTypeDescriptor Klasse generiert einen Cluster LabVIEW™-Typ als Constant oder Steuer/ Bedienelemente (in einem Blockdiagramm) oder als ctl auf die Festplatte. Die Tabelle stellt allgemeine Informationen zu Public-Methoden dar. Die LabVIEW™-Blöcke stellen weitere detaillierte Informationen zur Verfügung.

	Eigenschaft/Methode	Zugriff (Scope)	Bedeutung
	Read m_SubItems	Public	Liest die Eingenschaft m_SubItems (Die Anzahl von Elementen im aktuellen Cluster).
	Read m_GeneratedSubItems	Public	Liest die Eigenschaft m_GeneratedSubItems (Die Anzahl von generierten Elementen im aktuellen Cluster).
	Read m_NestedLevel	Public	Liest die Eigenschaft m_NestedLevel (Die geschachtelte Ebene des aktuellen Clusters).
	Read m_RemainingSubItems	Public	Liest die Eigenschaft m_RemainingSubItems (Die Anzahl von Elementen des aktuellen Clusters, die noch zu generieren sind).
	Read m_ActualPosition	Public	Liest die Eigenschaft m_ActualPosition (Iterator der SubTypes im Cluster).
	InitC	Public, Static Dispatch	Initialisiert die Cluster, basierend auf der geladenen TypeInfo.
	Create Type	Public, Static Dispatch	Generiert ein Cluster, basierend auf der geladenen TypeInfo.
	Create SubType	Public, Static Dispatch	Generiert ein Cluster als Sub-Typ, z. B. ein Element eines LabVIEW™ Clusters.

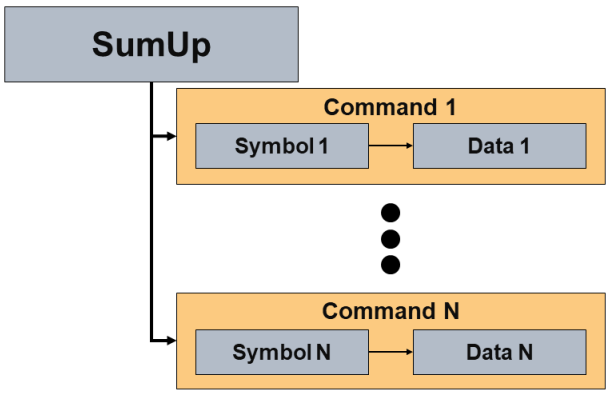
CEnumTypeDescriptor Klasse

Die CEnumTypeDescriptor Klasse generiert einen Enum LabVIEW™-Typ als Constant oder Steuer/ Bedienelemente (in einem Blockdiagramm) oder als ctl auf die Festplatte. Die Tabelle stellt allgemeine Informationen zu Public-Methoden dar. Die LabVIEW™-Blöcke stellen weitere detaillierte Informationen zur Verfügung.

	Eigenschaft/Methode	Zugriff (Scope)	Bedeutung
	Read m_Keys	Public	Liest die Eingenschaft m_Keys (Alle Text Elemente des Enums).
	Read m_Representation	Public	Liest die Eigenschaft m_Representation (Der unterlegende Speicher des Enums).
	Init	Public, Static Dispatch	Initialisiert die Enum, basierend auf der geladenen TypeInfo.
	Create Type	Public, Static Dispatch	Generiert ein Enum, basierend auf der geladenen TypeInfo.
	Create SubType	Public, Static Dispatch	Generiert ein Enum als Sub-Typ, z. B. ein Element eines LabVIEW™ Clusters.

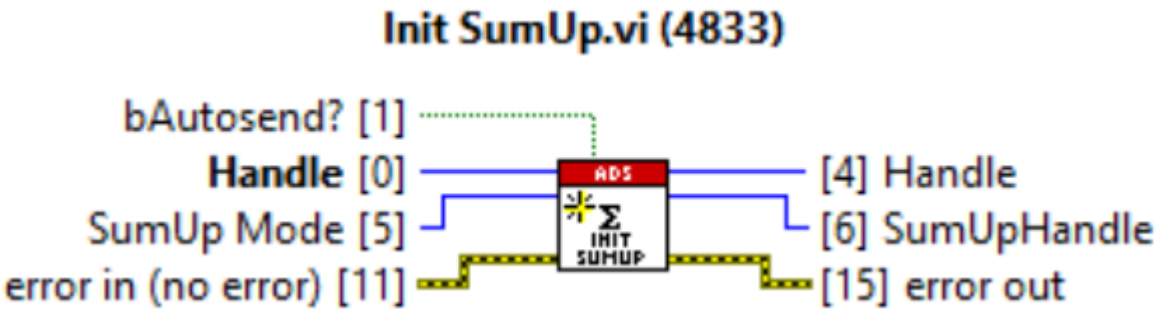
7.11.5 SumUp

Der Unterordner *SumUp* enthält Low-Level Blöcke, welche das Schreiben oder Lesen von mehreren ADS-Symbolen mit einem API Call ermöglichen. Im Vergleich zu einfachen ADS Read- oder Write-Anweisungen, wird für das Schreiben bzw. Lesen von mehreren Symbolen mit dem SumUp nur ein Handle benötigt. Das Handle kann wiederum mehrere Subkommandos enthalten, die gleichzeitig an die TwinCAT 3 Runtime gesendet werden.



Init SumUp

Der Block *Init SumUp* initialisiert den ADS SumUp. Bei erfolgreichem Aufruf gibt der Block ein Handle auf den ADS-SumUp zurück.

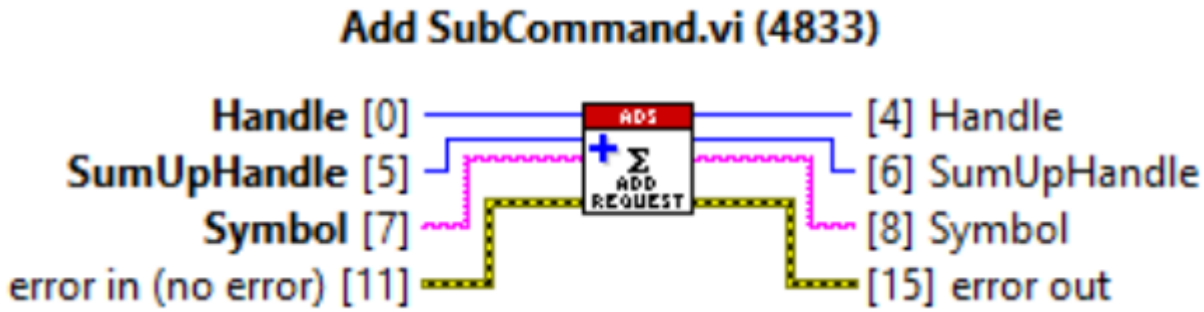


Eingang/Ausgang	Bedeutung
[0] [4] Handle	Handle auf den Client
[1] bAutosend	Autosend Flag ermöglicht das automatische Senden des SubCommandos

Eingang/Ausgang	Bedeutung
[5] SumUp Mode	SumUp Modi: • Schreiben • Lesen
[6] SumUp Handle	Handle auf den SumUp

Add SubCommand

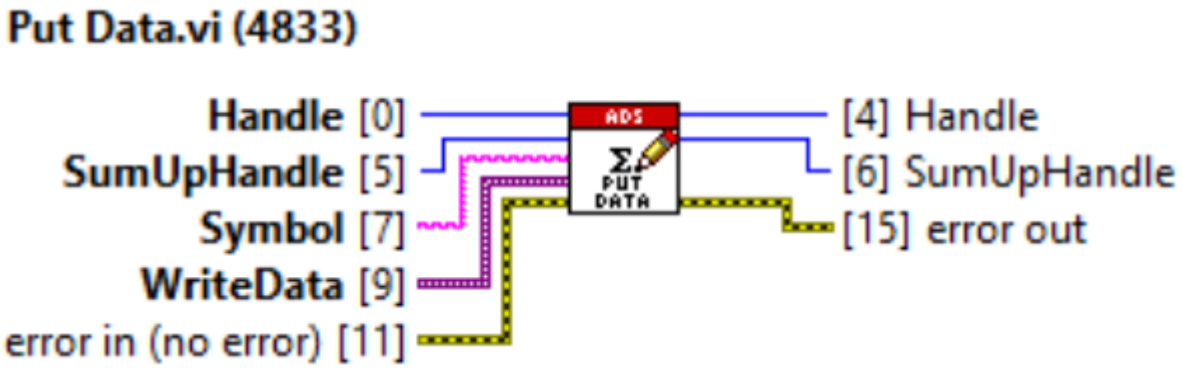
Der Block *Add SubCommand* initialisiert ein neues Sub-Kommando und fügt es dem SumUp Handle hinzu.



Eingang/Ausgang	Bedeutung
[0] [4] Handle	Handle auf den Client
[5] [6] SumUpHandle	Handle auf den SumUp
[7] [8] Symbol	Sub-Kommando Symbol

Put Data

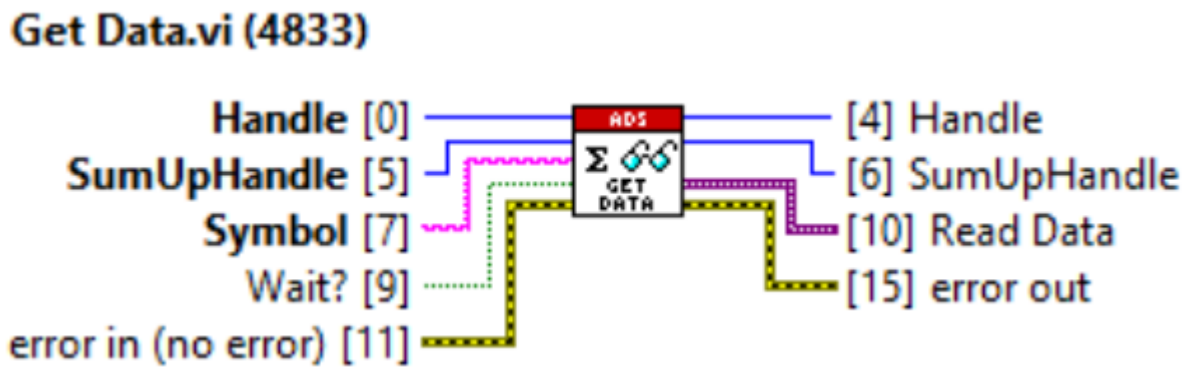
Der Block *Put Data* fügt dem initialisierten Sub Kommando neue Daten hinzu. Dazu benötigt der Block den Symbol Name um das Sub Kommando zu identifizieren. Der Block kann nur für schreibende SumUp Kommandos genutzt werden.



Eingang/Ausgang	Bedeutung
[0] [4] Handle	Handle auf den Client.
[5] [6] SumUpHandle	Handle auf den SumUp
[7] Symbol	Sub Kommando Symbol
[9] WriteData	Die zu schreibenden Daten

Get Data

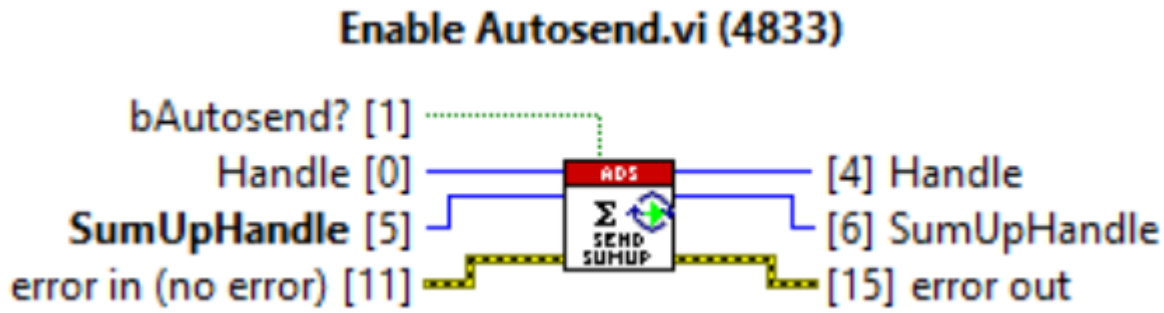
Der Block *Get Data* ruft dem initialisierten Sub Kommando neue Daten ab. Dazu benötigt der Block den Symbol Name, um das Sub Kommando zu identifizieren. Der Block kann nur für lesende SumUp Kommandos genutzt werden.



Eingang/Ausgang	Bedeutung
[0] [4] Handle	Handle auf den Client
[5] [6] SumUpHandle	Handle auf den SumUp
[7] Symbol	Sub Kommando Symbol
[9] Wait?	Wait Flag, das definiert, ob auf die zu lesenden Daten gewartet wird. • True: Der Block wartet bis zum Timeout, ob die neuen Daten reingekommen sind. • False: Der Block wartet nicht auf die neuen Daten.
[15] Read Data	Die zu lesenden Daten

Enable Autosend

Der Block *Enable Autosend* ermöglicht das automatische Senden des SumUp Kommandos.



Eingang/Ausgang	Bedeutung
[0] [4] Handle	Handle auf den Client
[1] bAutosend	Das Autosend Flag ermöglicht automatisches Schicken des SubCommandos
[5] [6] SumUpHandle	Handle auf den SumUp

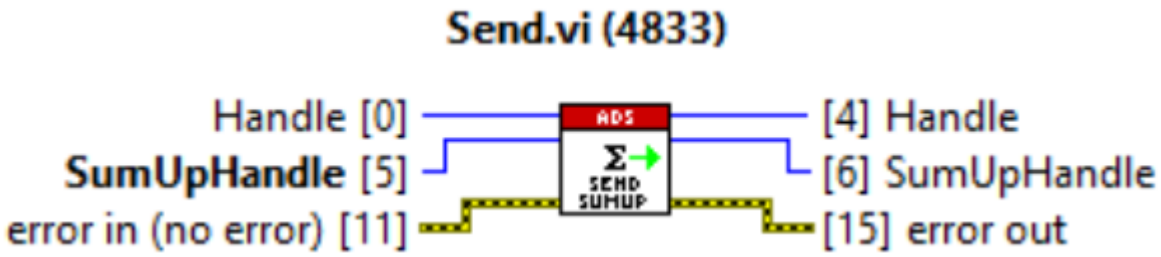
HINWEIS

Sub Kommandos ohne Daten

Das automatische Senden schlägt bei initialisierten Sub Kommandos fehl, die keine Daten beinhalten.

Send SumUp

Der Block *Send SumUp* sendet die dem SumUp Handle hinzugefügten Sub Kommandos an die TwinCAT 3 Runtime. Im Gegenteil zu SumUp [▶ 100], muss der Send SumUp explizit zyklisch aufgerufen werden, um die Daten an TwinCAT zu senden.



Eingang/Ausgang	Bedeutung
[0] [4] Handle	Handle auf den Client
[5] [6] SumUpHandle	Handle auf den SumUp

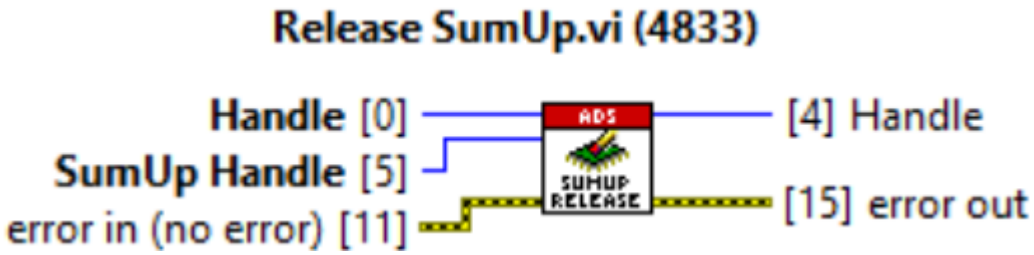
HINWEIS

Sub Kommandos ohne Daten

Das Senden schlägt bei initialisierten Sub Kommandos fehl, die keine Daten beinhalten.

Release SumUp

Der Block *Release SumUp* gibt den SumUp Handle aus dem Speicher frei.



Eingang/Ausgang	Bedeutung
[0] [4] Handle	Handle auf den Client
[5] SumUpHandle	Handle auf den SumUp

7.12 With TypeResolving

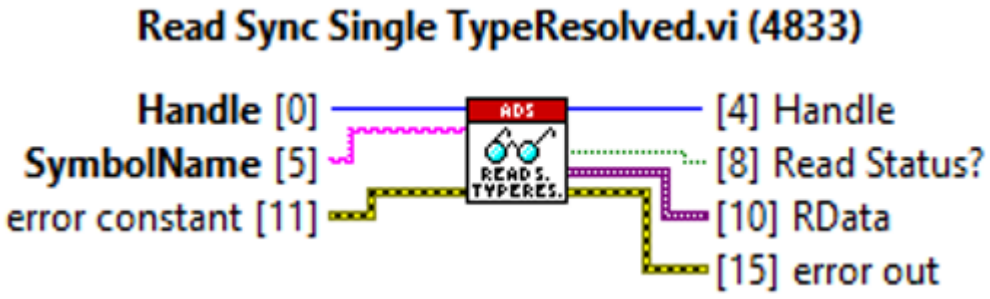
Der Ordner *With TypeResolving* enthält Blöcke, die das Lesen und Schreiben über ADS mit TypeResolver integrieren und dementsprechend das Programmieren in LabVIEW™ weiter vereinfachen.

Read TypeResolved

Der *Block Read TypeResolved* ist ein polymorphic Block, der das Lesen über ADS mit dem TypeResolver integriert. Der Block bietet synchrones und asynchrones Lesen über ADS.

Read Sync Single TypeResolved

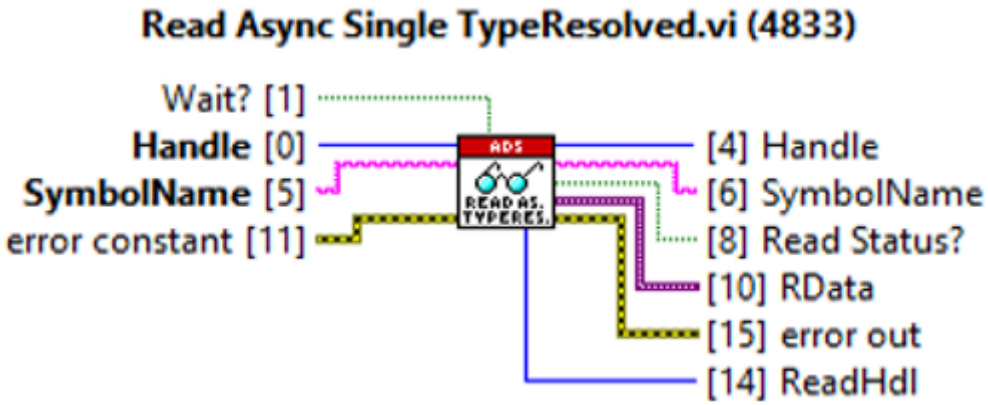
Der Block *Read Sync Single TypeResolved* ruft den Block ADS-Read [▶ 67] auf, um das ADS-Datenpaket (als ADS-Rohdaten), **synchron** vom ADS-Server zu empfangen und danach mit dem Block TypeResolver [▶ 74] in einen LabVIEW™-Datentyp zu konvertieren.



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[4] Handle	Handle auf den ADS-Client
[8] Read Status?	Lese-Status
[10] RData	TypeResolved ADS-Rohdaten als Variant

Read Async Single TypeResolved

Der Block *Read Async Single TypeResolved* ruft den Block ADS-Read [▶ 68] auf, um das ADS-Datenpaket (als ADS-Rohdaten) **asynchron** vom ADS-Server zu empfangen und danach mit dem Block TypeResolver [▶ 74] in einen LabVIEW™-Datentyp zu konvertieren.



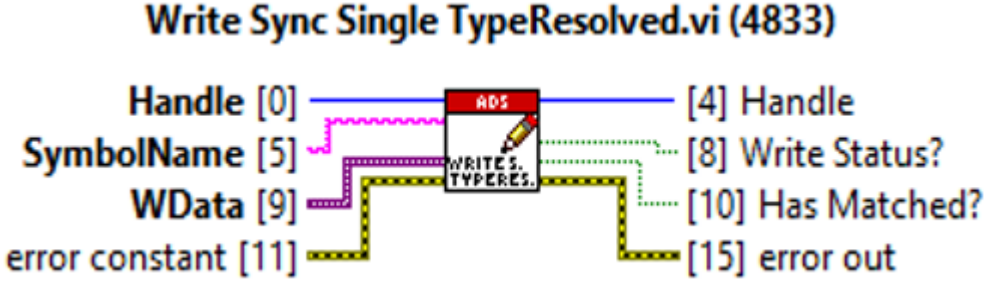
Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[1] Wait?	TRUE = Warte auf Server-Antwort (sync) FALSE (default) = Warte nicht auf Server-Antwort (async)
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[4] Handle	Handle auf den ADS-Client
[6] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[8] Read Status?	Lese-Status
[10] RData	TypeResolved ADS-Rohdaten als Variant
[14] ReadHdl	Handle auf den ADS-Reader

Write TypeResolved

Der *Block Write TypeResolved* ist ein polymorphic Block, der das Schreiben über ADS mit dem TypeResolver integriert. Der Block bietet synchrones und asynchrones Schreiben über ADS.

Write Sync Single TypeResolved

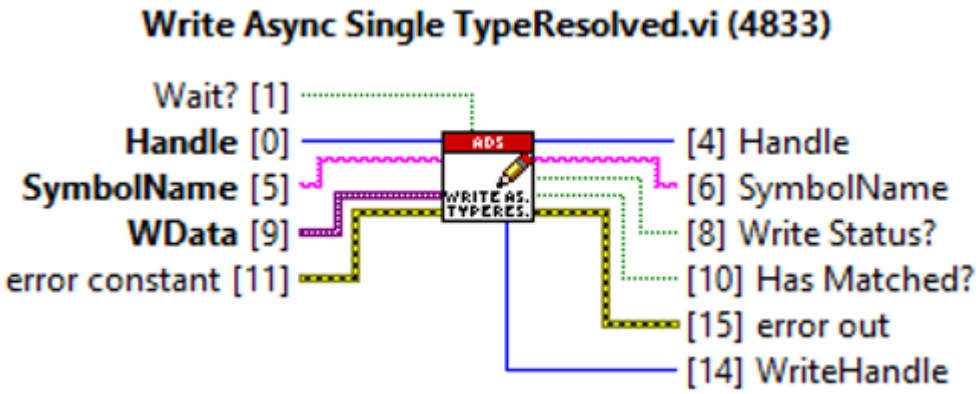
Der Block *Write Sync Single TypeResolved* ruft den TypeResolver [▶ 74] Block auf, um den LabVIEW™-Datentyp in einen TC3-Datentyp zu konvertieren und danach das konvertierte Datenpaket mit dem Aufruf von Block ADS-Write [▶ 73] **synchron** an den ADS-Server zu senden.



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[9] WData	ADS-Datenpaket als Rohdaten
[4] Handle	Handle auf den ADS-Client
[8] Write Status?	Schreib-Status
[10] Has Matched?	Flag, ob Konvertierung und Vergleich zwischen LabVIEW™ und TC3-Datentyp erfolgreich war

Write Async Single TypeResolved

Der Block *Write Async Single TypeResolved* ruft den TypeResolver [▶ 74] Block auf, um den LabVIEW™-Datentyp in einen TC3-Datentyp zu konvertieren und danach das konvertierte Datenpaket mit dem Aufruf vom Block ADS-Write [▶ 73] **asynchron** an den ADS-Server zu senden.



Eingang/Ausgang	Bedeutung
[0] Handle	Handle auf den ADS-Client
[1] Wait?	TRUE = Warte auf Server-Antwort (sync) FALSE (default) = Warte nicht auf Server-Antwort (async)
[5] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[9] WData	ADS-Datenpaket als Rohdaten
[4] Handle	Handle auf den ADS-Client
[6] SymbolName	ADS-Symbol bestehend aus AMSNetId und Symbol-Name
[8] Write Status?	Schreib-Status
[10] Has Matched?	Flag, ob Konvertierung und Vergleich zwischen LabVIEW™ und TC3-Datentyp erfolgreich war
[14] WriteHandle	Handle auf den ADS-Writer

8 Beispiele

Das Produkt TF3710 TwinCAT 3 Interface for LabVIEW™ kategorisiert die Beispiele in zwei verschiedene Gruppen:

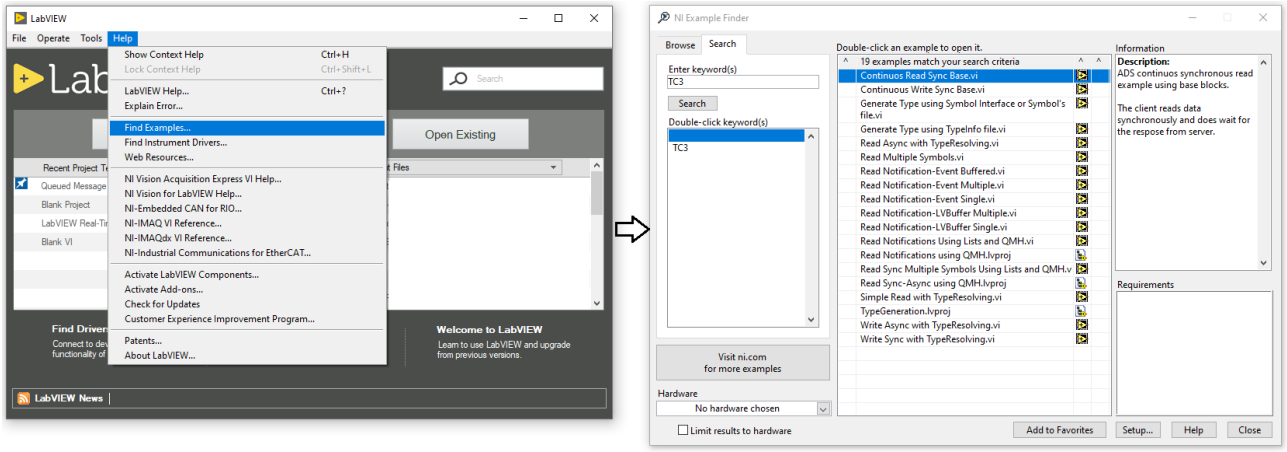
8.1 Grundlegende Beispiele

Eine große Anzahl von grundlegenden Beispielen wird während der Installation des Interface for LabVIEW™ mit in die LabVIEW™-Umgebung integriert. Diese können mit der Hilfe des **NI Example Finders** gefunden werden.

Folgende Schlagworte erleichtern die Suche nach den Beispielen:

- TC3
- Beckhoff
- Beckhoff-LabVIEW-Interface
- ADS
- TwinCAT
- TF3710

Der NI Example Finder kann in LabVIEW™ unter **Menu > Hilfe > Beispiele Finden** gestartet werden.



Ebenso können Sie alle Beispiele über die *Directory Structure* unter **Beckhoff Automation > Beckhoff-LabVIEW-Interface** finden.

Die Beispiele sind kategorisiert in die eingangs in der Dokumentation beschriebenen Kommunikations-Modi [▶ 25]:

- Einmaliges Lesen
- Einmaliges Schreiben
- Daten kontinuierlich lesen
- Daten kontinuierlich schreiben
- LabVIEW™-Typ generieren
- SumUp lesen oder schreiben
- CoE lesen oder schreiben

Einmaliges Lesen

Simple Read with TypeResolving.vi

Das Beispiel nutzt, wie die anderen Beispiele in Grundlegende Beispiele [▶ 104], die Grund-VIs aus den LabVIEW™-VIs [▶ 41] in den folgenden Schritten:

1. Im ersten Schritt wird der Grund-Block Symbol Interface [▶ 65] aufgerufen. Bei Ausführung des Beispiels öffnet sich das User-Interface und der Nutzer kann ein ADS-Symbol in seiner TwinCAT-Laufzeit auswählen.
   - Wenn Sie bereits eine XML-Beschreibung ihrer ADS-Symbole aus dem Symbol Interface exportiert haben, können Sie es als Pfad auswählen, sodass die XML geladen wird und nicht mehr das UI aufgerufen wird.
2. Im zweiten Schritt werden der Grund-Block Init [▶ 67] aufgerufen und der ADS-Client initialisiert.
3. Der Ausgang ReadGrpSymbols enthält ggf. mehr als ein ADS-Symbol (wenn Sie mehr als ein Symbol unter Schritt 1 ausgewählt haben). Im Beispiel wird mit *Index Array* das erste ADS-Symbol der Liste selektiert.
4. Nach der Initialisierung wird der polymorphic Block ADS-Read [▶ 67] aufgerufen. Mit dem Eingang **Read Choice** kann ausgewählt werden, welcher Read Mode genutzt werden soll. Der Eingang Read Choice ist ein LabVIEW™-Enum und bietet folgende Optionen:
   - **Read Sync**: Hierzu wird zuerst eine synchrone Leseanfrage an die TwinCAT-Laufzeit gesendet und auf eine Rückmeldung gewartet. Das empfangene ADS-Datenpaket wird danach, mit der Hilfe des TypeResolver, von Rohdaten in den LabVIEW™-Datentyp „Variant“ konvertiert.
   - **Noti. Single**: Hierzu wird eine ADS-Notification registriert/angemeldet und einzelne empfangene Notifications an LabVIEW™ weitergegeben. Nach Lesen der Notification werden die Notifications wieder abgemeldet. Die empfangenen ADS-Daten werden danach, mit der Hilfe des TypeResolver, von Rohdaten in den LabVIEW™-Datentyp „Variant“ konvertiert.
   - **Noti. Buffered**: Hierzu wird eine ADS-Notification registriert/angemeldet und ein Block von empfangenen Notifications an LabVIEW™ weitergegeben. Die Anzahl der im Block gepufferten Notifications wird durch den Symbol-Parameter `LVBufferSize` bestimmt. Nach dem Lesen der Notifications werden die Notifications wieder abgemeldet. Die empfangenen ADS-Daten werden danach, mit der Hilfe des TypeResolver, von Rohdaten in den LabVIEW™-Datentyp „Variant“ konvertiert.
5. **Interaktion**: Mit Variant-to-Data kann das Beispiel angepasst werden, um den Variant in einen passenden Datentyp zu übersetzen. Der Datentyp hängt ab vom gewählten ADS-Symbol im Symbol Interface. Siehe dazu Type Resolving [▶ 38].
6. Im letzten Schritt wird der Grund-Block Release [▶ 75] aufgerufen und der ADS-Client aus dem Speicher freigegeben.

HINWEIS

Noti. Single

Noti. Single unterstützt nur den `Transmode`=OnChange

Read Async with TypeResolving.vi

Das Beispiel nutzt, wie die anderen Beispiele in Grundlegende Beispiele [▶ 104], die Grund-VIs und die Low-Level-VIs aus LabVIEW™-VIs [▶ 41] in den folgenden Schritten:

1. Im ersten Schritt wird der Grund-Block Symbol Interface [▶ 65] aufgerufen. Bei Ausführung des Beispiels öffnet sich das User-Interface und der Nutzer kann ein ADS-Symbol in seiner TwinCAT-Laufzeit auswählen.
   - Wenn Sie bereits eine XML-Beschreibung der ADS-Symbole aus dem Symbol Interface exportiert haben, können Sie es als Pfad auswählen, sodass die XML geladen wird und nicht mehr das UI aufgerufen wird.
2. Im zweiten Schritt werden der Grund-Block Init [▶ 67] aufgerufen und der ADS-Client initialisiert.
3. Der Ausgang ReadGrpSymbols enthält ggf. mehr als ein ADS-Symbol (wenn Sie mehr als ein Symbol unter Schritt 1 ausgewählt haben). Im Beispiel wird mit *Index Array* das erste ADS-Symbol der Liste selektiert.
4. Nach der Initialisierung wird der Block With TypeResolving [▶ 102] aufgerufen. Hierzu wird eine asynchrone Leseanfrage an die TwinCAT-Laufzeit gesendet. Der Programmcode läuft weiter ohne auf Rückmeldung zu warten.

5. In diesem Beispiel wird der Low-Level-Block Read [▶ 88] in einer while-Schleife aufgerufen und wartet damit auf eine Rückmeldung vom angefragten ADS-Server. Dies ist als Beispiel zu verstehen und kann in Applikationen in ganz anderer Form implementiert werden. Nach empfangener Rückmeldung wird das empfangene Datenpaket in den passenden LabVIEW™-Datentyp Variant konvertiert.
6. **Interaktion**: Mit Variant-to-Data kann das Beispiel angepasst werden, um den Variant in einen passenden Datentyp zu übersetzen. Der Datentyp hängt ab vom gewählten ADS-Symbol im Symbol Interface. Siehe dazu Type Resolving [▶ 38].
7. Im nächsten Schritt wird ein Low-Level-Block Read [▶ 89] aufgerufen und damit der Reader aus dem Speicher freigegeben.
8. Im letzten Schritt wird der Grund-Block Release [▶ 75] aufgerufen und damit der ADS-Client aus dem Speicher freigegeben.

Einmaliges Schreiben

Write Sync with TypeResolving.vi

Das Beispiel nutzt, wie die anderen Beispiele in Grundlegende Beispiele [▶ 104], die Grund-VIs aus LabVIEW™-VIs [▶ 41] in den folgenden Schritten:

1. Im ersten Schritt wird der Grund-Block Symbol Interface [▶ 65] aufgerufen. Bei Ausführung des Beispiels öffnet sich das User-Interface und der Nutzer kann ein ADS-Symbol in seiner TwinCAT-Laufzeit auswählen.
   - Wenn Sie bereits eine XML-Beschreibung ihrer ADS-Symbole aus dem Symbol Interface exportiert haben, können Sie es als Pfad auswählen, sodass die XML geladen wird und nicht mehr das UI aufgerufen wird.
2. Im zweiten Schritt werden der Grund-Block Init [▶ 67] aufgerufen und der ADS-Client initialisiert.
3. Der Ausgang WriteGrpSymbols enthält ggf. mehr als ein ADS-Symbol (wenn Sie mehr als ein Symbol unter Schritt 1 ausgewählt haben). Im Beispiel wird mit *Index Array* das erste ADS-Symbol der Liste selektiert.
4. **Interaktion**: Fügen Sie an rot markierter Stelle den passenden Datentypen ein. Der Datentyp hängt ab vom gewählten ADS-Symbol im Symbol Interface. Siehe dazu Datentypen [▶ 118].
5. Nach der Initialisierung wird der Block Write Sync Single TypeResolved [▶ 102] aufgerufen. Hierzu werden zuerst die Rohdaten vom LabVIEW™-Datentyp in einen passenden TwinCAT 3-Datentyp konvertiert und danach eine synchrone Schreib-Anfrage an TwinCAT gesendet. Der Block wartet auf eine Rückmeldung vom Server, um sicherzustellen, ob die Daten erfolgreich gesendet wurden.
6. Im letzten Schritt wird der Grund-Block Release [▶ 75] aufgerufen und damit der ADS-Client aus dem Speicher freigegeben.

Write Async with TypeResolving.vi

Das Beispiel nutzt, wie die anderen Beispiele in Grundlegende Beispiele [▶ 104], die Grund-VIs und die Low-Level-VIs aus LabVIEW™-VIs [▶ 41] in den folgenden Schritten:

1. Im ersten Schritt wird der Grund-Block Symbol Interface [▶ 65] aufgerufen. Bei Ausführung des Beispiels öffnet sich das User-Interface und der Nutzer kann ein ADS-Symbol in seiner TwinCAT-Laufzeit auswählen.
   - Wenn Sie bereits eine XML-Beschreibung ihrer ADS-Symbole aus dem Symbol Interface exportiert haben, können Sie es als Pfad auswählen, sodass die XML geladen wird und nicht mehr das UI aufgerufen wird.
2. Im zweiten Schritt werden der Grund-Block Init [▶ 67] aufgerufen und der ADS-Client initialisiert.
3. Der Ausgang WriteGrpSymbols enthält ggf. mehr als ein ADS-Symbol (wenn Sie mehr als ein Symbol unter Schritt 1 ausgewählt haben). Im Beispiel wird mit *Index Array* das erste ADS-Symbol der Liste selektiert.
4. **Interaktion**: Fügen Sie an rot markierter Stelle den passenden Datentypen ein. Der Datentyp hängt ab vom gewählten ADS-Symbol im Symbol Interface. Siehe dazu Datentypen [▶ 118].

5. Nach der Initialisierung wird der Block Write Async Single TypeResolved [▶ 103] aufgerufen. Hierzu werden zuerst die Rohdaten vom LabVIEW™-Datentyp in einen passenden TwinCAT 3-Datentyp konvertiert und danach eine asynchrone Schreib-Anfrage an TwinCAT gesendet. Der Block wartet nicht auf eine Rückmeldung des ADS-Servers.
6. In diesem Beispiel wird der Low-Level-Block CheckWriteStatus [▶ 90] in einer while-Schleife aufgerufen und wartet damit auf eine Rückmeldung vom angefragten ADS-Server. Dies ist als Beispiel zu verstehen und kann in Applikationen in ganz anderer Form implementiert werden.
7. Im nächsten Schritt wird der Low-Level-Block Release Writer [▶ 90] aufgerufen und damit der Writer aus dem Speicher freigegeben.
8. Im letzten Schritt wird der Grund-Block Release [▶ 75] aufgerufen und damit der ADS-Client aus dem Speicher freigegeben.

Daten kontinuierlich lesen

Die Beispiele in dieser Gruppe nutzen das sogenannte Polling-Verfahren oder LabVIEW™ Event-basierte Verfahren, um die Datenpakete zyklisch anzufragen.

Continuos Read Sync Base.vi

Das Beispiel nutzt, wie die anderen Beispiele in Grundlegende Beispiele [▶ 104], die Grund-VIs und die Low-Level-VIs aus LabVIEW™-VIs [▶ 41] in den folgenden Schritten:

1. Im ersten Schritt wird der Grund-Block Symbol Interface [▶ 65] aufgerufen. Bei Ausführung des Beispiels öffnet sich das User-Interface und der Nutzer kann ein ADS-Symbol in seiner TwinCAT-Laufzeit auswählen.
   - Wenn Sie bereits eine XML-Beschreibung ihrer ADS-Symbole aus dem Symbol Interface exportiert haben, können Sie es als Pfad auswählen, sodass die XML geladen wird und nicht mehr das UI aufgerufen wird.
2. Im zweiten Schritt werden die Low-Level Blöcke Base Init [▶ 86], Get List of ReadWrite Symbols [▶ 86] und Get List of Registered Targets [▶ 86] aufgerufen und damit der ADS-Client initialisiert.
3. **Um das Lesen und Typeresolving zu beschleunigen**, werden im nächsten Schritt der ADS-Reader, mit dem Aufruf von Init Reader [▶ 87], und der TypeResolver, mit dem Aufruf Init Type [▶ 91], vorab und nur einmalig initialisiert.
4. Im nächsten Schritt fragt der Send Reader-Request [▶ 87] Block mit jedem Zyklus der Schleife ein neues Datenpaket aus TwinCAT an. Das empfangene Datenpaket wird mit dem Aufruf Resolve From TC Type [▶ 91] in einen passenden LabVIEW™-Datentyp Variant konvertiert.
5. **Interaktion**: Mit Variant-to-Data kann das Beispiel angepasst werden, um den Variant in einen passenden Datentyp zu übersetzen. Der Datentyp hängt ab vom gewählten ADS-Symbol im Symbol Interface. Siehe dazu Type Resolving [▶ 38].
6. Nach Erreichen der Abbruchbedingung der Schleife wird der ADS-Reader, mit dem Aufruf Release Reader [▶ 89], und der TypeResolver, mit dem Aufruf Release Type [▶ 92], aus dem Speicher freigegeben.
7. Im letzten Schritt wird der Grund-Block Release [▶ 75] aufgerufen und damit der ADS-Client aus dem Speicher freigegeben.

Read Notification-Event Single

Das Beispiel, wie die anderen Beispiele in Grundlegende Beispiele [▶ 104], nutzt die Grund-VIs und die Low-Level-VIs aus LabVIEW™-VIs [▶ 41] in den folgenden Schritten:

1. Im ersten Schritt wird der Grund-Block Symbol Interface [▶ 65] aufgerufen. Bei Ausführung des Beispiels öffnet sich das User-Interface und der Nutzer kann ein ADS-Symbol in seiner TwinCAT-Laufzeit auswählen. Beachten Sie bei der Auswahl die Paramatereinstellungen der Symbole für diese Betriebsart: E-Noti. Single.
   - Wenn Sie bereits eine XML-Beschreibung ihrer ADS-Symbole aus dem Symbol Interface exportiert haben, können Sie es als Pfad auswählen, sodass die XML geladen wird und nicht mehr das UI aufgerufen wird.

2. Im zweiten Schritt werden der Grund-Block Init [▶ 67] aufgerufen und der ADS-Client initialisiert.
3. Der Ausgang ReadGrpSymbols enthält ggf. mehr als ein ADS-Symbol (wenn Sie mehr als ein Symbol unter Schritt 1 ausgewählt haben). Im Beispiel wird mit *Index Array* das erste ADS-Symbol der Liste selektiert.
4. Es wird ein User Event erstellt und registriert. Siehe dazu auch die LabVIEW™-Dokumentation zu user events.
5. Mit dem Aufruf von E-Noti. Single [▶ 70] wird eine ADS-Notification am ADS-Server für das selektierte ADS-Symbol registriert.
6. Im nächsten Schritt wird in der Event Structure auf die ADS-Notifications gewartet. Wenn keine Notification empfangen wird, dann geht die Ereignisstruktur in einen Timeout. Mit Aufrufen des Blocks Stop Notification [▶ 80] wird die ADS-Notification gestoppt. Der Stopp erfolgt am ADS-Server, sodass keine Nachrichten mehr gesendet werden.
7. **Interaktion**: Type Resolving [▶ 38] ist in dem Beispiel nicht implementiert. Es gibt zwei Möglichkeiten. Entweder wird das Type Resolving innerhalb der Event-Loop realisiert, oder erst nach Beendigung/ Abmelden der ADS-Notification. Anknüpfungspunkt sin in der Event Structure die Punkte: Data sowie TimeStamps. Letztere sind die ADS timestamps.
8. Beim Aufrufen des Blocks Unregister Notification [▶ 80] wird die ADS-Notification des Symbols am ADS-Server abgemeldet und das Handle auf die ADS-Notification aus dem Speicher freigegeben.
9. Im letzten Schritt wird der Grund-Block Release [▶ 75] aufgerufen und damit der ADS-Client aus dem Speicher freigegeben.

Read Notification-Event Buffered.vi

Das Beispiel nutzt, wie die anderen Beispiele in Grundlegende Beispiele [▶ 104], die Grund-VIs und die Low-Level-VIs aus LabVIEW™-VIs [▶ 41]. Das Beispiel ist identisch im Aufbau zu Read Notification-Event Single [▶ 107]. Die einzigen Unterschiede sind:

- Das polymorphe VI wird auf E-Noti. Buffered [▶ 70] gestellt.
- Am Punkt Data in der Event-Loop wird immer ein Array der Größe `LVBufferSize` übergeben.

Read Notification-Event Multiple

Das Beispiel, wie die anderen Beispiele in Grundlegende Beispiele [▶ 104], nutzt die Grund-VIs und die Low-Level-VIs aus LabVIEW™-VIs [▶ 41]. Das Beispiel ist identisch im Aufbau zu Read Notification-Event Buffered [▶ 108]. Die einzigen Unterschiede sind:

- Das polymorphic VI wird auf E-Noti. Multiple Symbols [▶ 71] gestellt.
- Für jedes ADS-Symbol kann eine entsprechende Messdauer *ElapseTimeMs* (in Millisekunden) ausgewählt werden.
- Das Lesen von jedem ADS-Symbol benötigt eine entsprechende LabVIEW™-Event-Case.

Read Notification-LVBuffer Multiple

Das Beispiel nutzt, wie die anderen Beispiele in Grundlegende Beispiele [▶ 104], die Grund-VIs und die Low-Level-VIs aus LabVIEW™-VIs [▶ 41] in den folgenden Schritten:

1. Im ersten Schritt wird der Grund-Block Symbol Interface [▶ 65] aufgerufen. Bei Ausführung des Beispiels öffnet sich das User-Interface und der Nutzer kann ein ADS-Symbol in seiner TwinCAT-Laufzeit auswählen. Beachten Sie bei der Auswahl die Paramatereinstellungen der Symbole für die Betriebsart LVB-Noti. Multiple Symbols.
   - Wenn Sie bereits eine XML-Beschreibung ihrer ADS-Symbole aus dem Symbol Interface exportiert haben, können Sie es als Pfad auswählen, sodass die XML geladen wird und nicht mehr das UI aufgerufen wird.
2. Im zweiten Schritt werden der Grund-Block Init [▶ 67] aufgerufen und der ADS-Client initialisiert.
3. Der Ausgang ReadGrpSymbols enthält ggf. mehr als ein ADS-Symbol (wenn Sie mehr als ein Symbol unter Schritt 1 ausgewählt haben). Im Beispiel werden mit *Array Subset* die ersten 3 ADS-Symbole der Liste selektiert.

4. Es wird für jedes selektierte ADS-Symbol ein Handle auf das LVBuffer, mit dem Aufruf von Init LVBuffer Handle [▶ 81], initialisiert.
5. Mit dem Aufruf von LVB-Noti. Multiple Symbols [▶ 72] werden ADS-Notifications am ADS-Server für die selektierte ADS-Symbole registriert. Hierzu nimmt der Block ein Array von ADS Symbols als Eingang. Zusätzlich kann die Dauer der einzelnen Notification in Millisekunden eingegeben werden. So werden die Notifications nach dem Ablauf der Zeit automatisch gestoppt.
6. Im nächsten Schritt werden die einzelnen Symbole in einer while-Schleife gelesen. Auf das Lesen kann eine definierte Zeit (Timeout) oder unendlich lange gewartet werden.
7. Beim Aufrufen des Blocks Unregister Notification [▶ 80] wird die ADS-Notification des Symbols am ADS-Server abgemeldet und das Handle auf die ADS-Notification aus dem Speicher freigegeben.
8. Beim Aufrufen des Blocks Release LVBuffer Handle [▶ 82] wird das Handle auf den LVBuffer aus dem Speicher freigegeben.
9. Im letzten Schritt wird der Grund-Block Release [▶ 75] aufgerufen und damit der ADS-Client aus dem Speicher freigegeben.

Read Notification-LVBuffer Single

Das Beispiel, wie die anderen Beispiele in Grundlegende Beispiele [▶ 104], nutzt die Grund-VIs und die Low-Level-VIs aus LabVIEW™-VIs [▶ 41] in den folgenden Schritten:

1. Im ersten Schritt wird der Grund-Block Symbol Interface [▶ 65] aufgerufen. Bei Ausführung des Beispiels öffnet sich das User-Interface und der Nutzer kann ein ADS-Symbol in seiner TwinCAT-Laufzeit auswählen. Beachten Sie bei der Auswahl die Paramatereinstellungen der Symbole für die Betriebsart LVB-Noti. Single Symbol.
   - Wenn Sie bereits eine XML-Beschreibung ihrer ADS-Symbole aus dem Symbol Interface exportiert haben, können Sie es als Pfad auswählen, sodass die XML geladen wird und nicht mehr das UI aufgerufen wird.
2. Im zweiten Schritt werden der Grund-Block Init [▶ 67] aufgerufen und der ADS-Client initialisiert.
3. Der Ausgang ReadGrpSymbols enthält ggf. mehr als ein ADS-Symbol (wenn Sie mehr als ein Symbol unter Schritt 1 ausgewählt haben). Im Beispiel wird mit *Index Array* das erste ADS-Symbol der Liste selektiert.
4. Es wird für das selektierte ADS-Symbol ein Handle auf den LVBuffer, mit dem Aufruf von Init LVBuffer Handle [▶ 81], initialisiert.
5. Mit dem Aufruf von LVB-Noti. Single Symbol [▶ 72] werden ADS-Notifications am ADS-Server für das selektierte ADS-Symbol registriert. Zusätzlich ermöglicht es der Block, eine Messdauer in Millisekunden einzugeben. So werden die Notifications nach dem Ablauf der Zeit automatisch gestoppt.
6. Im nächsten Schritt kann das selektierte Symbol in einer while-Schleife gelesen werden. Auf das Lesen kann eine definierte Zeit (Timeout) oder unendlich lange gewartet werden. Wichtig dazu zu betrachten ist, ob die ADS-Notifications Cyclic oder OnChange vom Server gesendet werden.
7. Beim Aufrufen des Blocks Unregister Notification [▶ 80] wird die ADS-Notification des Symbols am ADS-Server abgemeldet und das Handle auf die ADS-Notification aus dem Speicher freigegeben.
8. Beim Aufrufen des Blocks Release LVBuffer Handle [▶ 82] wird das Handle auf den LVBuffer aus dem Speicher freigegeben.
9. Im letzten Schritt wird der Grund-Block Release [▶ 75] aufgerufen und damit der ADS-Client aus dem Speicher freigegeben.

Daten kontinuierlich schreiben

Continuos Write Sync Base.vi

Das Beispiel, wie die anderen Beispiele in Grundlegende Beispiele [▶ 104], nutzt die Grund-VIs und die Low-Level-VIs aus LabVIEW™-VIs [▶ 41] in den folgenden Schritten:

1. Im ersten Schritt wird der Grund-Block Symbol Interface [▶ 65] aufgerufen. Bei Ausführung des Beispiels öffnet sich das User-Interface und der Nutzer kann ein ADS-Symbol in seiner TwinCAT-Laufzeit auswählen.

- Wenn Sie bereits eine XML-Beschreibung ihrer ADS-Symbole aus dem Symbol Interface exportiert haben, können Sie es als Pfad auswählen, sodass die XML geladen wird und nicht mehr das UI aufgerufen wird.

2. Im zweiten Schritt werden der Grund-Block Init [▶ 67] aufgerufen und der ADS-Client initialisiert.
3. Der Ausgang WriteGrpSymbols enthält ggf. mehr als ein ADS-Symbol (wenn Sie mehr als ein Symbol unter Schritt 1 ausgewählt haben). Im Beispiel wird mit *Index Array* das erste ADS-Symbol der Liste selektiert.
4. Um das Schreiben und Type Resolving zu beschleunigen, werden im nächsten Schritt der ADS-Writer, mit dem Aufruf Init Writer [▶ 89], und der TypeResolver, mit dem Aufruf Init Type [▶ 91], einmalig vorher initialisiert.
5. Im nächsten Schritt sendet der Send Writer-Request [▶ 90]-Block mit jedem Zyklus der Schleife ein neues Datenpaket an TwinCAT. Vor dem Schreiben wird der LabVIEW™-Datentyp Variant, mit dem Aufruf von Resolve To TC Type [▶ 92] Block, in einem passenden TwinCAT 3-Datentyp konvertiert.
6. Nach Erreichen der Abbruchbedingung der Schleife wird der ADS-Reader, mit dem Aufruf von Release Writer [▶ 90], und der TypeResolver, mit dem Aufruf von Release Type [▶ 92], aus dem Speicher freigegeben.
7. Im letzten Schritt wird der Grund-Block Release [▶ 75] aufgerufen und der ADS-Client aus dem Speicher freigegeben.

LabVIEW™-Typ generieren

Um einen LabVIEW™-Datentyp automatisch zu einem ADS-Symbol zu generieren, werden der TypeResolver [▶ 91] und der Generate Type [▶ 77] Wrapper Block genutzt. Die folgenden Beispiele beschreiben zwei verschiedene Wege, um dies umzusetzen:

- Generate Type using Symbol Interface or Symbol's file
- Generate Type using TypeInfo file

Generate Type using Symbol Interface or Symbol's file

Das Beispiel nutzt, wie die anderen Beispiele in Grundlegende Beispiele [▶ 104], die Grund-VIs und die Low-Level-VIs aus LabVIEW™-VIs [▶ 41] in den folgenden Schritten:

1. Im ersten Schritt wird der Grund-Block Symbol Interface [▶ 65] aufgerufen. Bei Ausführung des Beispiels öffnet sich das User-Interface und der Nutzer kann ein ADS-Symbol in seiner TwinCAT-Laufzeit auswählen.
   - Wenn Sie bereits eine XML-Beschreibung ihrer ADS-Symbole aus dem Symbol Interface exportiert haben, können Sie es als Pfad auswählen, sodass die XML geladen wird und nicht mehr das UI aufgerufen wird.
2. Im zweiten Schritt werden der Grund-Block Init [▶ 67] aufgerufen und der ADS-Client initialisiert.
3. Der Ausgang ReadGrpSymbols und WriteGrpSymbols enthält ggf. mehr als ein ADS-Symbol (wenn Sie mehr als ein Symbol unter Schritt 1 ausgewählt haben).
4. In einer For-Schleife wird ausgeführt:
   - TypeResolver: Mit Init Type [▶ 91] wird die Typ-Beschreibung im TypeResolver geladen und an den LabVIEW™-Prozess als XML-Zeichenkette (**TypeInfo**) weitergegeben.
   - TypeGenerator: Mit dem Aufruf des Generate Type [▶ 77] Wrapper VI wird ein LabVIEW™-Typ generiert.
   - Release-Type: Gibt das Handle auf den TypeResolver aus dem Speicher frei.
5. Gibt das Handle auf den Client frei.

Generate Type using TypeInfo file

Das Beispiel nutzt, im Gegenteil zu Generate Type using Symbol Interface or Symbol's file [▶ 110], eine vorgenerierte **TypeInfo**-Datei, um einen LabVIEW™-Typ zu generieren.

SumUp Lesen oder Schreiben

Read SumUp with TypeResolving.vi

Das Beispiel nutzt, wie die anderen Beispiele in Grundlegende Beispiele [▶ 104], die Grund-VIs aus den LabVIEW™-VIs [▶ 41] in den folgenden Schritten:

1. Im ersten Schritt wird der Grund-Block Symbol Interface [▶ 65] aufgerufen. Bei Ausführung des Beispiels öffnet sich das User-Interface und der Nutzer kann ein ADS-Symbol aus seiner TwinCAT-Laufzeit auswählen.
   - Wenn Sie bereits eine XML-Beschreibung ihrer ADS-Symbole aus dem Symbol Interface exportiert haben, können Sie auch vor dem Starten des VIs im Front Panel die exportierete Datei auswählen, sodass die XML geladen wird und nicht das UI aufgerufen wird.
2. Im zweiten Schritt werden der Grund-Block Init [▶ 67] aufgerufen und der ADS-Client initialisiert.
3. Der Ausgang ReadGrpSymbols enthält ggf. mehr als ein ADS-Symbol (wenn Sie mehr als ein Symbol unter Schritt 1 ausgewählt haben). Im Beispiel werden mit *Index Array* das erste und zweite ADS-Symbol der Liste selektiert.
4. Nach der Initialisierung des ADS-Clients werden die TypeResolvers [▶ 38] mit den selektierten Symbolen initialisiert.
5. Danach wird das SumUp Handle mit Hilfe Init SumUp [▶ 98] fürs Lesen initialisiert. Hier wird das flag *bAutosend* auf true gesetzt, damit neue Daten automatisch gelesen werden.
6. Der Block Add Request baut die SumUp Subkommandos zusammen.
7. Danach teilt sich das Beispiel in zwei verscheiden Threads:
   - **Loop 1**: Liest zyklisch die neuen Daten nacheinander aus den Subkommandos. Die gelesenen Daten werden mit dem initialisierten TypeResolver in entsprechende LabVIEW™-Datentypen konvertiert.
   - **Loop 2:** Ermöglicht das Aktivieren oder Deaktivieren des automatischen Sendens von ADS-Anfragen.
8. Im letzten Schritt werden die alle initialisierten Handles aus dem Speicher freigegeben.

HINWEIS

ADS-Fehlermeldung beim Lesen von Symbolen mit bAutosend auf false

In Loop 1 können ADS-Fehlermeldungen auftreten, wenn beim Initialisieren des SumUp Handle das Flag *bAutosend* auf false gesetzt ist. In diesem Fall fängt der Block **Get Data** an, die Daten aus den Subkommandos zu lesen, obwohl sie noch nicht an den ADS-Server geschickt wurden und daher keine Daten beinhalten.

Write SumUp with TypeResolving.vi

Das Beispiel nutzt, wie die anderen Beispiele in Grundlegende Beispiele [▶ 104], die Grund-VIs und die Low-Level-VIs aus LabVIEW™-VIs [▶ 41] in den folgenden Schritten:

1. Im ersten Schritt wird der Grund-Block Symbol Interface [▶ 65] aufgerufen. Bei Ausführung des Beispiels öffnet sich das User-Interface und der Nutzer kann ein ADS-Symbol aus seiner TwinCAT-Laufzeit auswählen.
   - Wenn Sie bereits eine XML-Beschreibung der ADS-Symbole aus dem Symbol Interface exportiert haben, können Sie es als Pfad auswählen, sodass die XML geladen wird und nicht das UI aufgerufen wird.
2. Im zweiten Schritt wird der Grund-Block Init [▶ 67] aufgerufen und der ADS-Client initialisiert.
3. Der Ausgang WriteGrpSymbols enthält ggf. mehr als ein ADS-Symbol (wenn Sie mehr als ein Symbol unter Schritt 1 ausgewählt haben). Im Beispiel werden zwei ADS Symbole von den Typen Zeitstempel und Zeichenkette erwartet. Mit *Index Array* werden das erste und das zweite ADS-Symbol der Liste selektiert.
4. Nach der Initialisierung werden die TypeResolvers [▶ 38] auf die selektierten Symbole initialisiert. Danach wird überprüft, ob das erste Symbol den Datentyp Zeitstempel und das zweite den Datentyp Zeichenkette hat.

5. Danach wird das SumUp Handle mit Hilfe Init SumUp [▶ 98] für das Schreiben initialisiert. Hier wird das flag *bAutosend* auf false gesetzt, um die neuen Daten nicht automatisch zu schreiben.
6. Danach teilt sich das Beispiel in zwei verscheiden Threads:
   - **Loop 1**: Wartet in einer Ereignis-Struktur auf neue Daten, die von Loop 2 generiert werden. Die neuen Daten werden zu den entsprechenden Subkommandos verteilt und auf das Senden gewartet. Davor werden die Rohdaten in einen TwinCAT 3 Datentyp transformiert. Beinhaltet das Subkommando schon Daten, werden diese durch die Neuen ersetzt. In der Ereignis-Struktur kann das automatische Senden aktiviert oder deaktiviert werden.
   - **Loop 2:** Erzeugt die Block Diagramm Ereignisse in einem Zyklus von 250 ms.
7. Im letzten Schritt werden alle initialisierten Handles aus dem Speicher freigegeben.

CoE lesen oder schreiben

Das Beispiel nutzt, wie die anderen Beispiele in diesem Kapitel, die Grund-VIs und die Low-Level-VIs aus LabVIEW™-VIs [▶ 41] in den folgenden Schritten:

1. Im ersten Schritt wird der Grund-Block Init [▶ 67] mit einer leeren Zeichenkette aufgerufen und der ADS-Client wird initialisiert.
2. Die CoE-Blöcke [▶ 82] erwarten die AMS-Adresse als Zeichenkette des EtherCAT-Teilnehmers, von dem die CoE-Konfiguration gelesen werden soll. Das Steuerelement Device Address erwartet eine Zeichenkette mit dem folgenden Format: AMS Net-Id des Masters und AMS Port des Teilnehmers (AMS NetId:Port).

Wenn das Steuerelement leer gelassen wird, öffnet das Beispiel ein Dialogfenster zur Auswahl eines EtherCAT-Teilnehmers.

1. Das Beispiel liest die CoE-Liste von dem EtherCAT-Teilnehmer.
2. **Interaktion**:
   - Mit einem Doppelklick auf ein Objekt in der CoE-Liste kann dieses ausgewählt werden, um die Objektbeschreibung zu lesen. Hierbei wird das Low-Level VI aufgerufen.
   - Über den Button **Check Entry** kann ein Eintrag mit Subindex gelesen werden.
   - Im nächsten Schritt können verschiedene CoE-Einträge gelesen oder geschrieben werden.
3. Im letzten Schritt wird das Client Handle aus dem Speicher freigegeben.

8.2 Applikationsbeispiel

Die TwinCAT-Solution sowie das zugehörige VI (für x86 und x64 LabVIEW™ Bit Version) kann hier heruntergeladen werden: https://infosys.beckhoff.com/content/1031/TF3710_TC3_Interface_for_LabVIEW/Resources/10083995531.zip.

Die VIs sind aktuell mit LabVIEW™ 2017 kompiliert und können für die nachfolgenden LabVIEW™-Versionen (2018, 2019, 2020, 2021, 2022) ebenfalls benutzt werden.

Beschreibung des Beispiels

TwinCAT: Das TwinCAT-Projekt generiert in der SPS Signale über Signalgenerator-Funktionsbausteine (Sinus, Amplitudenmoduliertes Signal, Dreieck Signal, ….). Die SPS läuft mit einer Zykluszeit von 5 ms. Der Signalgenerator erzeugt pro Zyklus 10 Werte für die generierten Signale. Entsprechend wird hier ein Oversampling von 10 mit den Beckhoff Messtechnik-Klemmen simuliert.

LabVIEW™: Das LabVIEW™-Projekt nutzt das Grundlegende Beispiele [▶ 108]-Beispiel, um die ADS-Notifications als LabVIEW™-Events zu lesen. Hier werden zwei while-Schleifen parallel genutzt.

1. Die erste while-Schleife empfängt die ADS-Notifications und konvertiert die ADS-Rohdaten mit Hilfe des TypeResolvers in einen passenden LabVIEW™-Datentyp „Variant“. Die konvertierten Rohdaten werden danach in eine Warteschlage eingefügt.
2. Die zweite while-Schleife liest die Elemente aus der Warteschlage und nutzt den Block „Variant to Data“, um die Daten schlussendlich in einem Graph anzuzeigen. Es werden die empfangenen Daten und die ADS-Zeitstempel in zwei unterschiedlichen Graphen angezeigt.

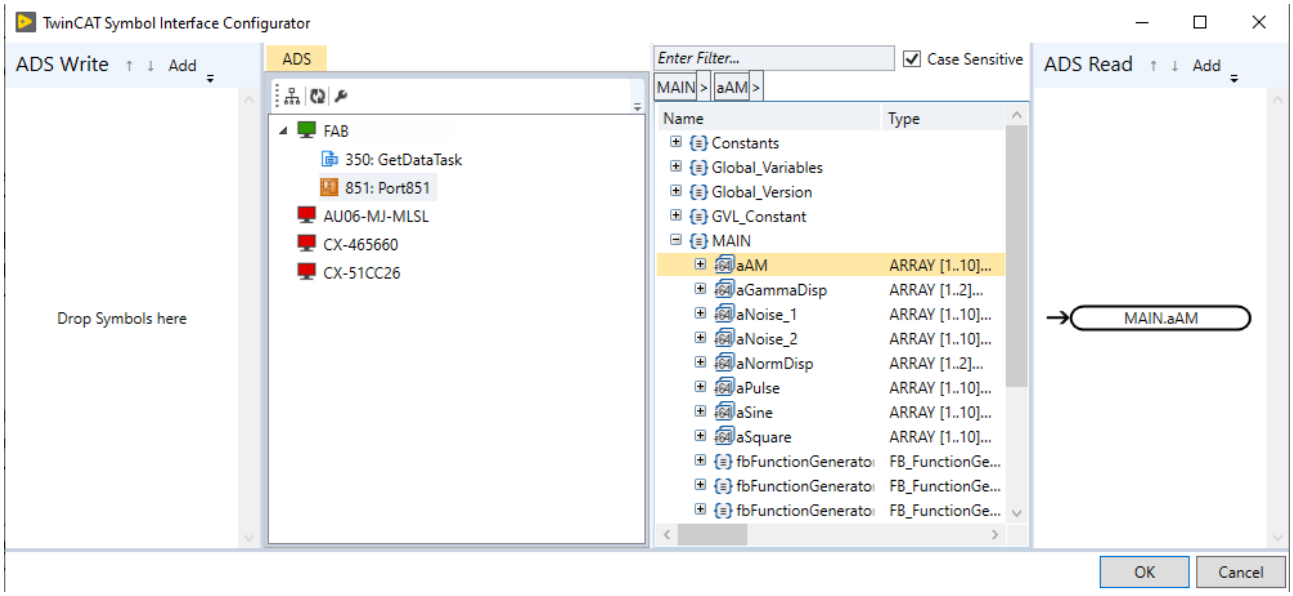
Öffnen und Starten des Beispiels

Das Beispiel enthält einen TC3- und einen LabVIEW-Ordner. Im TC3-Ordner befindet sich ein tnzip, welches Sie mit TwinCAT 3 über **File > Open > Open Solution From Archive** öffnen und dann als TwinCAT-Solution auf Ihrem PC speichern können.

Sie können die TwinCAT-Solution mit *activate configuration* auf dem Zielsystem Ihrer Wahl starten. Stellen Sie sicher, dass auf Ihrem Zielsystem eine TF3710-Lizenz vorhanden ist. Haben Sie keine gültige Lizenz, können Sie sich eine 7-Tage-Testlizenz erstellen.

Ist ihre TwinCAT-Laufzeit aktiv, können Sie die generierten Signale in TwinCAT mit dem TC3 Scope anzeigen lassen. Je nach gewähltem Zielsystem müssen Sie dazu nur unter **TC Signals > DataPool > aAM** (aNoise_1, aNoise_2, aSine) in den Properties Ihr Zielsystem auswählen.

Im Ordner LabVIEW wählen Sie zwischen 32bit (x86) und 64bit (x64) und öffnen das darin befindliche VI. Beim Starten des Programms öffnet sich das Symbol Interface. Browsen Sie in Ihr Zielsystem und wählen Sie eines der Signale aus.



9 Anhang

9.1 Übersicht der Fehlercodes

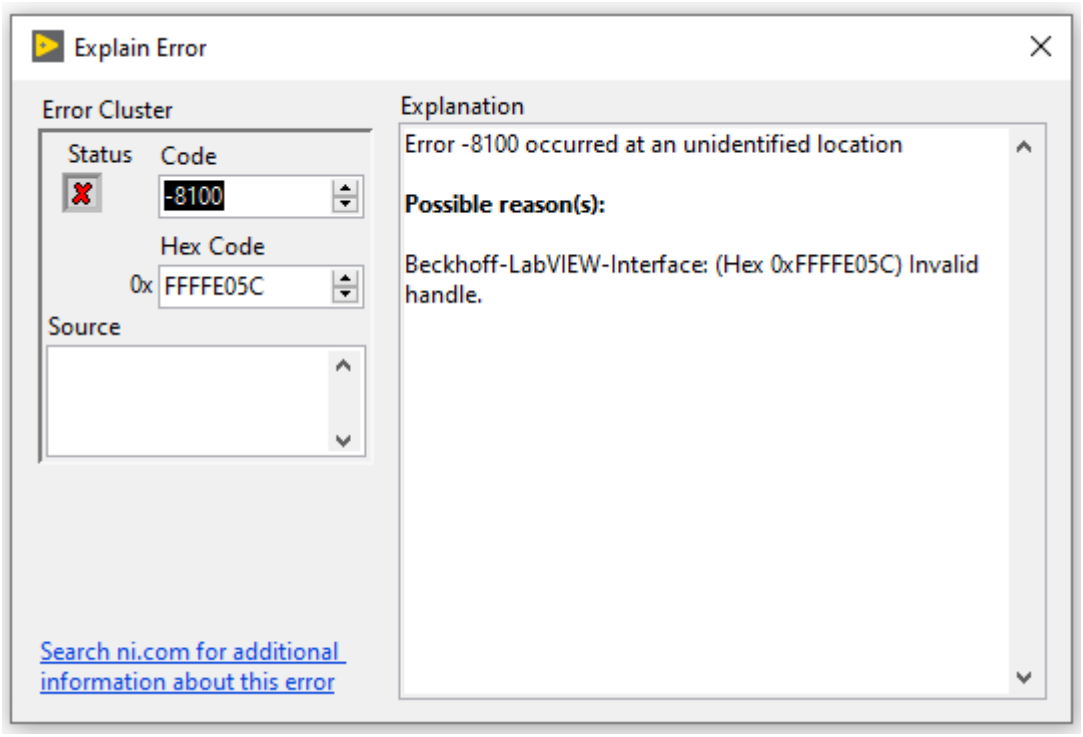
Die Fehlerprüfung im Produkt TF3710 TwinCAT 3 Interface for LabVIEW™ folgt dem Standard-LabVIEW™-Datenflussmodell. Während der Ausführung der LabVIEW™-VIs [▶ 41] findet an jedem ausgeführten Knoten eine Fehlerprüfung statt. Nur wenn kein Fehler auftritt, werden die Blöcke weiter durchgeführt. Wenn ein Fehler in einem Block auftritt, wird dieser an den nächsten Block übergeben und der betroffene Teil der Funktion oder der Block wird entsprechend nicht mehr ausgeführt.

Bei der Fehlerprüfung werden die LabVIEW™-Fehler-Cluster genutzt, die dann dementsprechend folgende Angaben weiterleiten:

- Status: Ein boolescher Wert, der TRUE ausgibt, wenn ein Fehler aufgetreten ist.
- Fehlercode: Eine Fehlerkennung in Form eines vorzeichenbehafteten 32-Bit-Integers. Wenn der Fehlercode nicht 0 lautet und der Status-Ausgang FALSE ausgibt, handelt es sich um eine Warnung und nicht um einen Fehler.
- Quelle: Gibt die Fehlerquelle in Form einer LabVIEW™-Zeichenkette an.

Die Beschreibung von dem Fehlercode befindet sich unter LabVIEW™

Menü > Hilfe > Fehler Beschreiben.



Um eine Kollision mit bestehenden LabVIEW™-Fehlern zu vermeiden, werden alle Fehlercodes in benutzerdefinierten Fehlercodes beschrieben. Beim Ausführen von LabVIEW™-VIs [▶ 41] können folgende Fehlercodes auftreten.

error value	Siehe…
16#FFFF_DE04 - 16#FFFF_DCD9	Gelistet in TwinCAT (ADS) Error Codes (dort ohne höherwertiges WORD). Weitere Informationen unten auf der Seite.
16#0000_1414 - 16#0000_1432	Gelistet in Support Return Codes [▶ 122].
16#0000_1464 - 16#0000_157C 16#FFFF_E05C – 16xFFFF_DFF9	Gelistet in Runtime Return Codes [▶ 119].

Wenn ein Fehler während der Initialisierung auftritt, kann der Funktionsbaustein nicht genutzt werden.

Weitere Informationen für Standard TwinCAT Error Codes:

error value	symbol	Fehlerbeschrei-bung	Behebungsmöglichkeit
16#FFFF_DDFA	NOMEMORY	Keinen Speicher	Falsche Speicher-Einstellungen => Router Speicher erhöhen.
16#FFFF_DDEB	TIMEOUT	Gerät hat einen Timeout	Ein Timeout kann während des Wartens von Rückmeldung oder Schickens von Anfrage auftreten. Bei einem überlasteten Netzwerk kann das häufig der Fall sein. Beim Einmaliges Lesen [▶ 29] oder Einmaliges Schreiben [▶ 30] ist das Timeout nicht so kritisch, da hier das Datenpaket in einem Durchgang gelesen oder geschrieben wird.

9.2 ADS Return Codes

Gruppierung der Fehlercodes: ADS Return Codes [▶ 116], ADS Return Codes [▶ 115], ADS Return Codes [▶ 116], ADS Return Codes [▶ 118], ADS Return Codes [▶ 118]...

Globale Fehlercodes

Hex	Dec	Name	Beschreibung
0xFFFFE504	-6908	ERR_NOERROR	Kein Fehler.
0xFFFFE503	-6909	ERR_INTERNAL	Interner Fehler.
0xFFFFE502	-6910	ERR_NORTIME	Keine Echtzeit.
0xFFFFE501	-6911	ERR_ALLOCLOCKEDMEM	Zuweisung gesperrt - Speicherfehler.
0xFFFFE500	-6912	ERR_INSERTMAILBOX	Postfach voll – Es konnte die ADS Nachricht nicht versendet werden. Reduzieren der Anzahl der ADS Nachrichten pro Zyklus bringt Abhilfe.
0xFFFFE4FF	-6913	ERR_WRONGRECEIVEHMSG	Falsches HMSG.
0xFFFFE4FE	-6914	ERR_TARGETPORTNOTFOUND	Ziel-Port nicht gefunden – ADS Server ist nicht gestartet oder erreichbar.
0xFFFFE4FD	-6915	ERR_TARGETMACHINENOTFOUND	Zielrechner nicht gefunden – AMS Route wurde nicht gefunden.
0xFFFFE4FC	-6916	ERR_UNKNOWNCMDID	Unbekannte Befehl-ID.
0xFFFFE4FB	-6917	ERR_BADTASKID	Ungültige Task-ID.
0xFFFFE4FA	-6918	ERR_NOIO	Kein IO.
0xFFFFE4F9	-6919	ERR_UNKNOWNAMSCMD	Unbekannter AMS-Befehl.
0xFFFFE4F8	-6920	ERR_WIN32ERROR	Win32 Fehler.
0xFFFFE4F7	-6921	ERR_PORTNOTCONNECTED	Port nicht verbunden.
0xFFFFE4F6	-6922	ERR_INVALIDAMSLENGTH	Ungültige AMS-Länge.
0xFFFFE4F5	-6923	ERR_INVALIDAMSNETID	Ungültige AMS Net ID.
0xFFFFE4F4	-6924	ERR_LOWINSTLEVEL	Installations-Level ist zu niedrig –TwinCAT 2 Lizenzfehler.
0xFFFFE4F3	-6925	ERR_NODEBUGINTAVAILABLE	Kein Debugging verfügbar.
0xFFFFE4F2	-6926	ERR_PORTDISABLED	Port deaktiviert – TwinCAT System Service nicht gestartet.
0xFFFFE4F1	-6927	ERR_PORTALREADYCONNECTED	Port bereits verbunden.
0xFFFFE4F0	-6928	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 Fehler.
0xFFFFE4EF	-6929	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0xFFFFE4EE	-6930	ERR_AMSSYNC_AMSERROR	AMS Sync Fehler.
0xFFFFE4ED	-6931	ERR_AMSSYNC_NOINDEXINMAP	Keine Index-Map für AMS Sync vorhanden.
0xFFFFE4EC	-6932	ERR_INVALIDAMSPORT	Ungültiger AMS-Port.
0xFFFFE4EB	-6933	ERR_NOMEMORY	Kein Speicher.

Hex	Dec	Name	Beschreibung
0xFFFFE4EA	-6934	ERR_TCPSEND	TCP Sendefehler.
0xFFFFE4E9	-6935	ERR_HOSTUNREACHABLE	Host nicht erreichbar.
0xFFFFE4E8	-6936	ERR_INVALIDAMSFRAGMENT	Ungültiges AMS Fragment.
0xFFFFE4E7	-6937	ERR_TLSSEND	TLS Sendefehler – Secure ADS Verbindung fehlgeschlagen.
0xFFFFE4E6	-6938	ERR_ACCESSDENIED	Zugriff Verweigert – Secure ADS Zugriff verweigert.

Router Fehlercodes

Hex	Dec	Name	Beschreibung
0xFFFFE004	-8188	ROUTERERR_NOLOCKEDMEMORY	Lockierter Speicher kann nicht zugewiesen werden.
0xFFFFE003	-8189	ROUTERERR_RESIZEMEMORY	Die Größe des Routerspeichers konnte nicht geändert werden.
0xFFFFE002	-8190	ROUTERERR_MAILBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0xFFFFE001	-8191	ROUTERERR_DEBUGBOXFULL	Das Debug Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0xFFFFE000	-8192	ROUTERERR_UNKNOWNPORTTYPE	Der Porttyp ist unbekannt.
0xFFFFDEEF	-8193	ROUTERERR_NOTINITIALIZED	Router ist nicht initialisiert.
0xFFFFDFFE	-8194	ROUTERERR_PORTALREADYINUSE	Die Portnummer ist bereits vergeben.
0xFFFFDFFD	-8195	ROUTERERR_NOTREGISTERED	Der Port ist nicht registriert.
0xFFFFDFFC	-8196	ROUTERERR_NOMOREQUEUES	Die maximale Portanzahl ist erreicht.
0xFFFFDFFB	-8197	ROUTERERR_INVALIDPORT	Der Port ist ungültig.
0xFFFFDFFA	-8198	ROUTERERR_NOTACTIVATED	Der Router ist nicht aktiv.
0xFFFFDFF9	-8199	ROUTERERR_FRAGMENTBOXFULL	Das Postfach hat die maximale Anzahl für fragmentierte Nachrichten erreicht.
0xFFFFDFF8	-8200	ROUTERERR_FRAGMENTTIMEOUT	Fragment Timeout aufgetreten.
0xFFFFDFF7	-8201	ROUTERERR_TOBEREMOVED	Port wird entfernt.

Allgemeine ADS Fehlercodes

Hex	Dec	Name	Beschreibung
0xFFFFDE04	-8700	ADSERR_DEVICE_ERROR	Allgemeiner Gerätefehler.
0xFFFFDE03	-8701	ADSERR_DEVICE_SRVNOTSUPP	Service wird vom Server nicht unterstützt.
0xFFFFDE02	-8702	ADSERR_DEVICE_INVALIDGRP	Ungültige Index-Gruppe.
0xFFFFDE01	-8703	ADSERR_DEVICE_INVALIDOFFSET	Ungültiger Index-Offset.
0xFFFFDE00	-8704	ADSERR_DEVICE_INVALIDACCESS	Lesen oder Schreiben nicht gestattet.
0xFFFFDDFF	-8705	ADSERR_DEVICE_INVALIDSIZE	Parametergröße nicht korrekt.
0xFFFFDDFE	-8706	ADSERR_DEVICE_INVALIDDATA	Ungültige Daten-Werte.
0xFFFFDDFD	-8707	ADSERR_DEVICE_NOTREADY	Gerät nicht betriebsbereit.
0xFFFFDDFC	-8708	ADSERR_DEVICE_BUSY	Gerät beschäftigt.
0xFFFFDDFB	-8709	ADSERR_DEVICE_INVALIDCONTEXT	Ungültiger Kontext vom Betriebssystem - Kann durch Verwendung von ADS Bausteinen in unterschiedlichen Tasks auftreten. Abhilfe kann die Multitasking-Syncronisation in der SPS geben.
0xFFFFDDFA	-8710	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher.
0xFFFFDDF9	-8711	ADSERR_DEVICE_INVALIDPARM	Ungültige Parameter-Werte.
0xFFFFDDF8	-8712	ADSERR_DEVICE_NOTFOUND	Nicht gefunden (Dateien,...).
0xFFFFDDF7	-8713	ADSERR_DEVICE_SYNTAX	Syntax-Fehler in Datei oder Befehl.
0xFFFFDDF6	-8714	ADSERR_DEVICE_INCOMPATIBLE	Objekte stimmen nicht überein.
0xFFFFDDF5	-8715	ADSERR_DEVICE_EXISTS	Objekt ist bereits vorhanden.
0xFFFFDDF4	-8716	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol nicht gefunden.
0xFFFFDDF3	-8717	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Symbol-Version ungültig – Kann durch einen Online-Change auftreten. Erzeuge einen neuen Handle.
0xFFFFDDF2	-8718	ADSERR_DEVICE_INVALIDSTATE	Gerät (Server) ist im ungültigen Zustand.
0xFFFFDDF1	-8719	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode nicht unterstützt.

Hex	Dec	Name	Beschreibung
0xFFFFDDF0	-8720	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification Handle ist ungültig.
0xFFFFDDEF	-8721	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert.
0xFFFFDDEE	-8722	ADSERR_DEVICE_NOMOREHDLS	Keine weiteren Notification Handles verfügbar.
0xFFFFDDED	-8723	ADSERR_DEVICE_INVALIDWATCHSIZE	Größe der Notification zu groß.
0xFFFFDDEC	-8724	ADSERR_DEVICE_NOTINIT	Gerät nicht initialisiert.
0xFFFFDDEB	-8725	ADSERR_DEVICE_TIMEOUT	Gerät hat einen Timeout.
0xFFFFDDEA	-8726	ADSERR_DEVICE_NOINTERFACE	Interface Abfrage fehlgeschlagen.
0xFFFFDDE9	-8727	ADSERR_DEVICE_INVALIDINTERFACE	Falsches Interface angefordert.
0xFFFFDDE8	-8728	ADSERR_DEVICE_INVALIDCLSID	Class-ID ist ungültig.
0xFFFFDDE7	-8729	ADSERR_DEVICE_INVALIDOBJID	Object-ID ist ungültig.
0xFFFFDDE6	-8730	ADSERR_DEVICE_PENDING	Anforderung steht aus.
0xFFFFDDE5	-8731	ADSERR_DEVICE_ABORTED	Anforderung wird abgebrochen.
0xFFFFDDE4	-8732	ADSERR_DEVICE_WARNING	Signal-Warnung.
0xFFFFDDE3	-8733	ADSERR_DEVICE_INVALIDARRAYIDX	Ungültiger Array-Index.
0xFFFFDDE2	-8734	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol nicht aktiv.
0xFFFFDDE1	-8735	ADSERR_DEVICE_ACCESSDENIED	Zugriff verweigert.
0xFFFFDDE0	-8736	ADSERR_DEVICE_LICENSENOTFOUND	Fehlende Lizenz.
0xFFFFDDDF	-8737	ADSERR_DEVICE_LICENSEEXPIRED	Lizenz abgelaufen.
0xFFFFDDDE	-8738	ADSERR_DEVICE_LICENSEEXCEEDED	Lizenz überschritten.
0xFFFFDDDD	-8739	ADSERR_DEVICE_LICENSEINVALID	Lizenz ungültig.
0xFFFFDDDC	-8740	ADSERR_DEVICE_LICENSESYSTEMID	Lizenzproblem: System-ID ist ungültig.
0xFFFFDDDB	-8741	ADSERR_DEVICE_LICENSENOTIMELIMIT	Lizenz nicht zeitlich begrenzt.
0xFFFFDDDA	-8742	ADSERR_DEVICE_LICENSEFUTUREISSUE	Lizenzproblem: Zeitpunkt in der Zukunft.
0xFFFFDDD9	-8743	ADSERR_DEVICE_LICENSETIMETOLONG	Lizenz-Zeitraum zu lang.
0xFFFFDDD8	-8744	ADSERR_DEVICE_EXCEPTION	Exception beim Systemstart.
0xFFFFDDD7	-8745	ADSERR_DEVICE_LICENSEDUPLICATED	Lizenz-Datei zweimal gelesen.
0xFFFFDDD6	-8746	ADSERR_DEVICE_SIGNATUREINVALID	Ungültige Signatur.
0xFFFFDDD5	-8747	ADSERR_DEVICE_CERTIFICATEINVALID	Zertifikat ungültig.
0xFFFFDDD4	-8748	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public Key vom OEM nicht bekannt.
0xFFFFDDD3	-8749	ADSERR_DEVICE_LICENSERESTRICTED	Lizenz nicht gültig für diese System.ID.
0xFFFFDDD2	-8750	ADSERR_DEVICE_LICENSEDEMODENIED	Demo-Lizenz untersagt.
0xFFFFDDD1	-8751	ADSERR_DEVICE_INVALIDFNCID	Funktions-ID ungültig.
0xFFFFDDD0	-8752	ADSERR_DEVICE_OUTOFRANGE	Außerhalb des gültigen Bereiches.
0xFFFFDDCF	-8753	ADSERR_DEVICE_INVALIDALIGNMENT	Ungültiges Alignment.
0xFFFFDDCE	-8754	ADSERR_DEVICE_LICENSEPLATFORM	Ungültiger Plattform Level.
0xFFFFDDCD	-8755	ADSERR_DEVICE_FORWARD_PL	Kontext – Weiterleitung zum Passiv-Level.
0xFFFFDDCC	-8756	ADSERR_DEVICE_FORWARD_DL	Kontext – Weiterleitung zum Dispatch-Level.
0xFFFFDDCB	-8757	ADSERR_DEVICE_FORWARD_RT	Kontext – Weiterleitung zur Echtzeit.
0xFFFFDDC4	-8764	ADSERR_CLIENT_ERROR	Clientfehler.
0xFFFFDDC3	-8765	ADSERR_CLIENT_INVALIDPARM	Dienst enthält einen ungültigen Parameter.
0xFFFFDDC2	-8766	ADSERR_CLIENT_LISTEMPTY	Polling-Liste ist leer.
0xFFFFDDC1	-8767	ADSERR_CLIENT_VARUSED	Var-Verbindung bereits im Einsatz.
0xFFFFDDC0	-8768	ADSERR_CLIENT_DUPLINVOKEID	Die aufgerufene ID ist bereits in Benutzung.
0xFFFFDDBF	-8769	ADSERR_CLIENT_SYNCTIMEOUT	Timeout ist aufgetreten – Die Gegenstelle antwortet nicht im vorgegebenen ADS Timeout. Die Routeneinstellung der Gegenstelle kann falsch konfiguriert sein.

Hex	Dec	Name	Beschreibung
0xFFFFDDBE	-8770	ADSERR_CLIENT_W32ERROR	Fehler im Win32 Subsystem.
0xFFFFDDBD	-8771	ADSERR_CLIENT_TIMEOUTINVALID	Ungültiger Client Timeout-Wert.
0xFFFFDDBC	-8772	ADSERR_CLIENT_PORTNOTOPEN	Port nicht geöffnet.
0xFFFFDDBB	-8773	ADSERR_CLIENT_NOAMSADDR	Keine AMS Adresse.
0xFFFFDDB4	-8780	ADSERR_CLIENT_SYNCINTERNAL	Interner Fehler in Ads-Sync.
0xFFFFDDB3	-8781	ADSERR_CLIENT_ADDHASH	Überlauf der Hash-Tabelle.
0xFFFFDDB2	-8782	ADSERR_CLIENT_REMOVEHASH	Schlüssel in der Tabelle nicht gefunden.
0xFFFFDDB1	-8783	ADSERR_CLIENT_NOMORESYM	Keine Symbole im Cache.
0xFFFFDDB0	-8784	ADSERR_CLIENT_SYNCRESINVALID	Ungültige Antwort erhalten.
0xFFFFDDAF	-8785	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port ist verriegelt.

RTime Fehlercodes

Hex	Dec	Name	Beschreibung
0xFFFFD504	-11004	RTERR_INTERNAL	Interner Fehler im Echtzeit-System.
0xFFFDD503	-11005	RTERR_BADTIMERPERIODS	Timer-Wert nicht gültig.
0xFFFFD502	-11006	RTERR_INVALIDTASKPTR	Task-Pointer hat den ungültigen Wert 0 (null).
0xFFFFD501	-11007	RTERR_INVALIDSTACKPTR	Stack-Pointer hat den ungültigen Wert 0 (null).
0xFFFFD500	-11008	RTERR_PRIOEXISTS	Die Request Task Priority ist bereits vergeben.
0xFFFFD4FF	-11009	RTERR_NOMORETCB	Kein freier TCB (Task Control Block) verfügbar. Maximale Anzahl von TCBs beträgt 64.
0xFFFFD4FE	-11010	RTERR_NOMORESEMAS	Keine freien Semaphoren zur Verfügung. Maximale Anzahl der Semaphoren beträgt 64.
0xFFFFD4FD	-11011	RTERR_NOMOREQUEUES	Kein freier Platz in der Warteschlange zur Verfügung. Maximale Anzahl der Plätze in der Warteschlange beträgt 64.
0xFFFFD4FC	-11012	RTERR_EXTIRQALREADYDEF	Ein externer Synchronisations-Interrupt wird bereits angewandt.
0xFFFFD4FB	-11013	RTERR_EXTIRQNOTDEF	Kein externer Sync-Interrupt angewandt.
0xFFFFD4FA	-11014	RTERR_EXTIRQINSTALLFAILED	Anwendung des externen Synchronisierungs-Interrupts ist fehlgeschlagen.
0xFFFFD4F9	-11015	RTERR_IRQLNOTLESSOREQUAL	Aufruf einer Service-Funktion im falschen Kontext
0xFFFFD4F8	-11016	RTERR_VMXNOTSUPPORTED	Intel VT-x Erweiterung wird nicht unterstützt.
0xFFFFD4F7	-11017	RTERR_VMXDISABLED	Intel VT-x Erweiterung ist nicht aktiviert im BIOS.
0xFFFFD4F6	-11018	RTERR_VMXCONTROLSMISSING	Fehlende Funktion in Intel VT-x Erweiterung.
0xFFFFD4F5	-11019	RTERR_VMXENABLEFAILS	Aktivieren von Intel VT-x schlägt fehl.

TCP Winsock-Fehlercodes

Hex	Dec	Name	Beschreibung
0xFFFFBDB8	-16968	WSAETIMEDOUT	Verbindungs Timeout aufgetreten - Fehler beim Herstellen der Verbindung, da die Gegenstelle nach einer bestimmten Zeitspanne nicht ordnungsgemäß reagiert hat, oder die hergestellte Verbindung konnte nicht aufrecht erhalten werden, da der verbundene Host nicht reagiert hat.
0xFFFFBDB7	-16969	WSAECONNREFUSED	Verbindung abgelehnt - Es konnte keine Verbindung hergestellt werden, da der Zielcomputer dies explizit abgelehnt hat. Dieser Fehler resultiert normalerweise aus dem Versuch, eine Verbindung mit einem Dienst herzustellen, der auf dem fremden Host inaktiv ist—das heißt, einem Dienst, für den keine Serveranwendung ausgeführt wird.
0xFFFFBDB6	-16970	WSAEHOSTUNREACH	Keine Route zum Host - Ein Socketvorgang bezog sich auf einen nicht verfügbaren Host.
Weitere Winsock-Fehlercodes: Win32-Fehlercodes			

9.3 Datentypen

TwinCAT Datentyp	Speicherbedarf	LabVIEW™ Datentyp
BIT	1 Bit	LabVIEW™ Boolean
BOOL	8 Bits	LabVIEW™ Boolean
BYTE/USINT	8 Bits	LabVIEW™ U8
WORD/UINT	16 Bits	LabVIEW™ U16
DWORD/UDINT	32 Bits	LabVIEW™ U32

TwinCAT Datentyp	Speicherbedarf	LabVIEW™ Datentyp
ULINT	64 Bits	LabVIEW™ U64
SINT	8 Bits	LabVIEW™ I8
INT	16 Bits	LabVIEW™ I16
DINT	32 Bits	LabVIEW™ I32
LINT	64 Bits	LabVIEW™ I64
REAL	32 Bits	LabVIEW™ Single (sgl)
LREAL	64 Bits	LabVIEW™ Double (dbl)
String	-	LabVIEW™ String
TIME	-	LabVIEW™ Zeitstempel
DATE	-	LabVIEW™ Zeitstempel
TOD/ TIME_OF_DAY	-	LabVIEW™ Zeitstempel
DT/DATE_AND_TIME	-	LabVIEW™ Zeitstempel
DCTIME	-	LabVIEW™ Zeitstempel
FILETIME	-	LabVIEW™ Zeitstempel
LTIME	-	LabVIEW™ Zeitstempel
ARRAY	-	LabVIEW™ ARRAY
STRUCT	-	LabVIEW™ Cluster
Enum	16 Bits (Default)	LabVIEW™ Enum

9.4 Runtime Return Codes

Fehler die während der Initialisierung des ADS Clients, Aufbauen der Verbindung, Senden von Anfragen oder Erhalten den Datenpaketen, Konvertierung des LabVIEW™ Datentyp in TC3-Datentyp oder umgekehrt auftauchen, sind als Laufzeitfehler beschrieben.

error value	symbol	Fehlerbeschreibung	Behebungsmöglichkeit
16#00001464	5220	TF3710_ERR_E_POINTER	Ungültige Pointer 0 (null).
16#00001465	5221	TF3710_ERR_E_OUT_OF_RANGE	Der Wert liegt nicht in der Spannweite
16#00001466	5222	TF3710_ERR_E_INVALID_ARG	Die Parameter sind ungültig
16#00001467	5223	TF3710_ERR_E_OUT_OF_MEMORY	Der Speicher ist voll
16#00001468	5224	TF3710_ERR_E_NOT_A_NUMBER	Ist keine Zahl
16#00001469	5225	TF3710_ERR_E_BAD_ALLOCATION	Falsche Allokation
16#0000146A	5226	TF3710_ERR_E_NOT_A_STRING	Ist keine Zeichenkette
16#0000146B	5227	TF3710_ERR_E_UNINITIALIZED_TIMESTAMP	Zeitstempel ist nicht initialisiert
16#0000146D	5229	TF3710_ERR_E_CONFIGURATION_NON_EXISTENT	Die Konfiguration wurde nicht gefunden
16#0000146E	5230	TF3710_ERR_E_FLOATING_OBJECT	Floating/Nichtgespeichertes Objekt (VI, Project, ...)
16#0000146F	5231	TF3710_ERR_E_INVALID_CONFIGURATION	Die Konfiguration ist inkorrekt
16#0000146C	5228	TF3710_ERR_E_LOCKFAILED	Lock ist fehlgeschlagen
16#00001479	5241	TF3710_ERR_FAILED_WAIT_ON_A_REQUEST	Warten auf die Anfrage ist fehlgeschlagen

error value	symbol	Fehlerbeschreibung	Behebungsmöglichkeit
16#00001483	5251	TF3710_ERR_INVALID_PORT_GROUP	Ungültiger ADS Port Typ
16#00001484	5252	TF3710_ERR_NEW_NODE_FAILED	Der neue Knotenpunkt ist fehlgeschlagen
16#00001485	5253	TF3710_ERR_APPEND_BUFFER_FAILED	Schreiben an dem Knotenpunkt ist fehlgeschlagen
16#00001486	5254	TF3710_ERR_MISSING_NODE	Knotenpunkt fehlt
16#00001487	5255	TF3710_ERR_PARSE_ERROR	Parsen fehlgeschlagen
16#0000148D	5261	TF3710_ERR_PORT_ALREADY_EXIST	ADS Port ist noch vorhanden
16#0000148E	5262	TF3710_ERR_PORT_OPEN_FAILED	Öffnen von ADS Port ist fehlgeschlagen
16#0000148F	5263	TF3710_ERR_INVALID_TARGET_ID	AmsNetID vom Zielsystem ist ungültig
16#00001490	5264	TF3710_ERR_UNDEFINED_TARGET_PORT	ADS Port des Zielsystems ist ungültig
16#00001491	5265	TF3710_ERR_CONNECTION_FAILED	Verbinden mit dem Router ist fehlgeschlagen
16#00001492	5266	TF3710_ERR_NO_PORT_OPENED	Port ist nicht geöffnet
16#00001493	5267	TF3710_ERR_DISCONNECT_FAILED	Disconnect vom Router ist fehlgeschlagen
16#00001494	5268	TF3710_ERR_READ_STATE_REQUEST_FAILED	Read-State Anfrage ist fehlgeschlagen
16#00001495	5269	TF3710_ERR_VARHANDLE_FAILED	Das Laden von Var-Handler ist fehlgeschlagen
16#00001496	5270	TF3710_ERR_VARHANDLE_ALREADY_EXIST	Var-Handler ist bereits vorhanden
16#00001497	5271	TF3710_ERR_VARHANDLE_RELEASE_FAILED	Freigeben von Var-Handler ist fehlgeschlagen
16#00001498	5272	TF3710_ERR_VARHANDLE_ALREADY_RELEASED	Var-Handler wurde bereits freigegeben
16#00001499	5273	TF3710_ERR_INVALID_VAR_HANDLER	Var-Handler ist ungültig
16#0000149A	5274	TF3710_ERR_MISSING_ROUTE	ADS Route ist nicht vorhanden
16#0000149B	5275	TF3710_ERR_CONNECTION_ALREADY_EXISTS	Die Verbindung besteht bereits
16#000014A6	5286	TF3710_ERR_EMPTY_SYMBOL_TABLE	Aktuell keine Symbole in Tabelle vorhanden
16#000014A7	5287	TF3710_ERR_MISSING_TARGETS	Zielsysteme fehlen
16#000014A8	5288	TF3710_ERR_TYPESYSTEM_PROVIDER_FAILED	Der TypeSystem Provider ist fehlgeschlagen
16#000014A9	5289	TF3710_ERR_TYPERESOLVERPROXY_FAILED	Der TypeResolverProxy hat fehlgeschlagen
16#000014AA	5290	TF3710_ERR_TYPERESOLVERPROXY_LOAD_GUID_FAILED	Das Laden von TC-GUID ist fehlgeschlagen.
16#000014AB	5291	TF3710_ERR_TYPERESOLVERPROXY_LOAD_TYPE_COLLECTION_FAILED	Das Laden von TypeCollection ist fehlgeschlagen.

error value	symbol	Fehlerbeschreibung	Behebungsmöglichkeit
16#000014AC	5292	TF3710_ERR_TYPERESOLVERPROXY_SYMBOL_GUID_ERROR	GUID ist ungültig
16#000014AD	5293	TF3710_ERR_LVTYPEDESCRIPTOR_INVALID_ELEMENT_RANGE	Anzahl der Elemente im TypeResolver stimmen nicht überein.
16#000014AE	5294	TF3710_ERR_LVTYPEDESCRIPTOR_TYPE_NOT_SUPPORTED	Datentyp wird aktuelle nicht unterstützt
16#000014AF	5295	TF3710_ERR_LVTYPEDESCRIPTOR_INVALID_ARRAY_SYMBOLINFO	Im Array Datentyp fehlen Elemente
16#000014B0	5296	TF3710_ERR_LVTYPEDESCRIPTOR_MISSING_ARRAY_SYMBOLINFO_ELEMENTS	Symbol-Info für Array Datentyp ist ungültig
16#000014B1	5297	TF3710_ERR_LVTYPEDESCRIPTOR_MISSING_ARRAY_BASE_TYPE	Array Datentyp hat keinen Base-Type
16#000014B2	5298	TF3710_ERR_LVTYPEDESCRIPTOR_LOAD_FAILDED	Laden von TypeResolver ist fehlgeschlagen
16#000014B3	5299	TF3710_ERR_LVTYPEDESCRIPTOR_INVALID_PRIMARY_TYPE	Der Primär-Datentyp ist ungültig
16#000014B4	5300	TF3710_ERR_LVTYPEDESCRIPTOR_TYPE_UNINITIALIZED	TypeResolver ist nicht initialisiert
16#000014C4	5316	TF3710_ERR_LVTYPEDESCRIPTOR_INVALID_CLUSTER_SYMBOLINFO	Symbol-Info für Cluster Datentyp ist ungültig
16#000014C5	5317	TF3710_ERR_LVTYPEDESCRIPTOR_NON_CLUSTER_TYPE	Datentyp ist nicht vom Datentyp LabVIEW™-Cluster
16#000014C6	5318	TF3710_ERR_LVTYPEDESCRIPTOR_INVALID_CLUSTER_SYMBOLPOSITION	Im Cluster-Datentyp fehlen Elemente
16#000014D4	5331	TF3710_ERR_LVTYPEDESCRIPTOR_INVALID_TIMESTAMP_VALUE	Ungültiger TC-Zeitstempel Wert
16#000014D5	5332	TF3710_ERR_LVTYPEDESCRIPTOR_UNSUPORTED_TIMESTAMP	Dieser TC Zeitstempel Datentyp ist aktuelle nicht unterstützt
16#000014D9	5337	TF3710_ERR_LVTYPEDESCRIPTOR_TYPE_SIZE_MISMATCH	Datentypen sind nicht identisch
16#000014DA	5338	TF3710_ERR_LVTYPEDESCRIPTOR_TYPE_MISMATCH	Datentypen sind nicht identisch
16#000014DB	5339	TF3710_ERR_LVTYPEDESCRIPTOR_DATA_SIZE_MISMATCH	Datentypen-Inhalt stimmt nicht über
16#00001504	5380	TF3710_ERR_TYPEGENERATOR_SUBTYPE_NOT_FOUND	TypeGenerator kann den SubType oder den BaseType nicht finden
16#00001519	5401	TF3710_ERR_READ_REQUEST_FAILED	Read-Anfrage ist fehlgeschlagen

error value	symbol	Fehlerbeschreibung	Behebungsmöglichkeit
16#0000151A	5402	TF3710_ERR_INVALID_READ_BUFFER	ADS Rohdaten sind falsch gepackt
16#0000151B	5403	TF3710_ERR_INVALID_READ_DATA_FORMAT	ADS Rohdaten sind ungültig
16#0000151C	5404	TF3710_ERR_NOTIFICATION_HANDLER_FAILED	Notification-Handler ist fehlgeschlagen
16#0000151D	5405	TF3710_ERR_ATLEAST_ONE_MISSED_NOTIFICATION	Mindestens ein/mehrere Notifications ist/sind nicht richtig empfangen
16#0000151E	5406	TF3710_ERR_NOTIFICATION_REQUEST_FAILED	Notification-Anfrage ist fehlgeschlagen
16#0000151F	5407	TF3710_ERR_UNSUPPORTED_NOTIFICATION_MODE	Notification-Mode ist aktuelle nicht unterstützt
16#00001520	5408	TF3710_ERR_NOTIFICATION_BUFFER_FAILED	Buffered-Notification ist fehlgeschlagen
16#00001534	5428	TF3710_ERR_WRITE_REQUEST_FAILED	Write-Anfrage ist fehlgeschlagen
16#00001565	5429	TF3710_ERR_WRITE_BUFFER	Write-Buffer ungültig
16#FFFFE05C	-8100	TF3710_ERR_INVALID_HANDLE	Ungültiges Handle
16#FFFFE05B	-8101	TF3710_ERR_HANDLE_IN_USE	Das Handle wird noch genutzt
16#FFFFE052	-8110	TF3710_ERR_INVALID_ARRAY	Ungültiges Array
0xFFFFE051	-8111	TF3710_ERR_INVALID_ARRAY_SIZE	Array Größe ungültig
0xFFFFE050	-8112	TF3710_ERR_SYMBOL_NOT_FOUND	Symbol nicht gefunden
0xFFFFE04F	-8113	TF3710_ERR_LVBUFFER_NOT_REGISTERED	LVBuffer wurde nicht registriert

9.5 Support Return Codes

error value	symbol	Fehlerbeschreibung	Behebungsmöglichkeit
16#00001414	5140	TF3710_ERR_INVALID_REQUEST	Die Anfrage ist aktuell nicht unterstützt.
16#00001415	5141	TF3710_ERR_BETA_TRIAL_EXPIRED	Beta-Trial Zeitraum ist abgelaufen.
16#00001416	5142	TF3710_ERR_LICENSE_STATE_ISSUE	Der Lizenzzustand ist ungültig.

Mehr Informationen:
www.beckhoff.com/TF3710

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