



Dokumentation

TwinCAT Safety PLC

PC-basierte Sicherheitssteuerung

Version: 1.2.0
Datum: 29.06.2017

BECKHOFF

Inhaltsverzeichnis

1	Vorwort	5
1.1	Hinweise zur Dokumentation	5
1.2	Sicherheitshinweise	6
1.2.1	Auslieferungszustand	6
1.2.2	Sorgfaltspflicht des Betreibers	6
1.2.3	Erklärung der Sicherheitssymbole	7
1.3	Ausgabestände der Dokumentation	7
2	Systembeschreibung	8
2.1	Erweiterung des Beckhoff I/O-Systems mit Funktionen für die Sicherheitstechnik	8
2.2	TwinCAT Safety PLC	8
2.3	Sicherheitskonzept	8
3	Produktbeschreibung	10
3.1	Bestimmungsgemäße Verwendung	10
3.2	Technische Daten	12
3.3	Sicherheitstechnische Kenngrößen	13
3.4	Projektierungsgrenzen	13
4	Betrieb	14
4.1	Installation	14
4.1.1	Sicherheitshinweise	14
4.1.2	Transportvorgaben / Lagerung	14
4.1.3	Mechanische Installation	14
4.1.4	Elektrische Installation	15
4.1.5	Software Installation	15
4.1.6	Reaktionszeiten TwinSAFE	15
4.2	Konfiguration der TwinCAT Safety PLC in TwinCAT	17
4.2.1	Voraussetzung für die Konfiguration	17
4.2.2	Anlegen eines Safety Projektes in TwinCAT 3	17
4.2.3	CRC Verteilung	39
4.2.4	Download der Safety Applikation	40
4.2.5	Freischaltung der Safety Applikation	41
4.2.6	Safety und CRC Toolbar	42
4.2.7	Info-Daten	43
4.2.8	Task-Einstellungen	48
5	Anwendungsentwicklung in Safety C	50
5.1	Programmierung in Safety C	50
5.1.1	Abgrenzung der Programmierung in Safety C zu C/C++	50
5.1.2	Quellcode-Vorlagen	51
5.2	Sichere Codierregeln	55
5.2.1	Begriffserklärung	55
5.2.2	Allgemein	56
5.2.3	Strenge Typisierung	58
5.3	Zulässiger Sprachumfang	66
5.3.1	Einfache Datentypen	66
5.3.2	Aufzählungstypen	68
5.3.3	Datenstrukturen	68
5.3.4	Einfache Anweisungen	69
5.3.5	Kontrollstrukturen	70
5.3.6	Ausdrücke und Operatoren	73

5.3.7	Literale und Konstante	74
5.3.8	Funktionsaufrufe und benutzerdefinierte Funktionen.....	75
5.3.9	Asserts und Traces	76
5.4	Performance-Optimierungen	78
5.5	Anbindung an die E/A-Ebene	79
5.6	Verifikation und Validierung	81
5.7	Online-Diagnose	82
5.8	Sichere Hilfsfunktionen	86
5.8.1	Sichere Logikfunktionen.....	86
5.8.2	Sichere Ganzzahlarithmetik-Funktionen	89
5.8.3	Sichere Bit-Schiebe-Funktionen	97
5.8.4	Sichere Konvertierungsfunktionen (Boolesch zu Ganzzahl).....	100
5.8.5	Sichere Konvertierungsfunktionen (Ganzzahl zu Ganzzahl)	102
6	Graphische Anwendungsentwicklung	111
7	Anhang	112
7.1	Support und Service	112
7.2	Zertifikate	113

1 Vorwort

1.1 Hinweise zur Dokumentation

Zielgruppe

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen und internationalen Normen und Regeln vertraut ist.

Dieses Fachpersonal muss geschult sein, um gemäß der Anforderungen der EN 61508 eine sicherheitsrelevante Applikation in einer Hochsprache entsprechend des normativen Software-Lebenszyklusses, zu entwickeln, zu validieren und zu verifizieren.

Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.

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

Dokumentenursprung

Diese Dokumentation ist in deutscher Sprache verfasst. Alle weiteren Sprachen werden von dem deutschen Original abgeleitet.

Aktualität

Bitte prüfen Sie, ob Sie die aktuelle und gültige Version des vorliegenden Dokumentes verwenden. Auf der Beckhoff Homepage finden Sie unter <http://www.beckhoff.de/german/download/twinsafe.htm> die jeweils aktuelle Version zum Download. Im Zweifelsfall wenden Sie sich bitte an den technischen [Support](#) [► 112].

Produkteigenschaften

Gültig sind immer nur die Produkteigenschaften, die in der jeweils aktuellen Anwenderdokumentation angegeben sind. Weitere Informationen, die auf den Produktseiten der Beckhoff Homepage, in E-Mails oder sonstigen Publikationen angegeben werden, sind nicht maßgeblich.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte unterliegen zyklisch einer Revision. Deshalb ist die Dokumentation nicht in jedem Fall vollständig auf die Übereinstimmung mit den beschriebenen Leistungsdaten, Normen oder sonstigen Merkmalen geprüft. 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®, EtherCAT®, Safety over EtherCAT®, TwinSAFE®, XFC® und XTS® 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, DE102004044764, DE102007017835 mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

Die TwinCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente: EP0851348, US6167425 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

© Beckhoff Automation GmbH & Co. KG, Deutschland.

Weitergabe sowie Vervielfältigung dieses Dokuments, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet.

Zu widerhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten.

Lieferbedingungen

Es gelten darüber hinaus die allgemeinen Lieferbedingungen der Fa. Beckhoff Automation GmbH & Co. KG.

1.2 Sicherheitshinweise

1.2.1 Auslieferungszustand

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.






1.2.2 Sorgfaltspflicht des Betreibers

Der Betreiber muss sicherstellen, dass

- die TwinSAFE-Produkte nur bestimmungsgemäß verwendet werden (siehe Kapitel Produktbeschreibung).
- die TwinSAFE-Produkte nur in einwandfreiem, funktionstüchtigem Zustand betrieben werden.
- nur ausreichend qualifiziertes und autorisiertes Personal die TwinSAFE-Produkte betreibt.
- dieses Personal regelmäßig in allen zutreffenden Fragen von Arbeitssicherheit und Umweltschutz unterwiesen wird, sowie die Betriebsanleitung und insbesondere die darin enthaltenen Sicherheitshinweise kennt.
- die Betriebsanleitung stets in einem leserlichen Zustand und vollständig am Einsatzort der TwinSAFE-Produkte zur Verfügung steht.
- alle an den TwinSAFE-Produkten angebrachten Sicherheits- und Warnhinweise nicht entfernt werden und leserlich bleiben.

1.2.3 Erklärung der Sicherheitssymbole

In der vorliegenden Betriebsanleitung werden die folgenden Symbole mit einem nebenstehenden Sicherheitshinweis oder Hinweistext verwendet. Die Sicherheitshinweise sind aufmerksam zu lesen und unbedingt zu befolgen!

 GEFAHR	<p>Akute Verletzungsgefahr!</p> <p>Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht unmittelbare Gefahr für Leben und Gesundheit von Personen!</p>
 WARNUNG	<p>Verletzungsgefahr!</p> <p>Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht Gefahr für Leben und Gesundheit von Personen!</p>
 VORSICHT	<p>Schädigung von Personen!</p> <p>Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, können Personen geschädigt werden!</p>
 Achtung	<p>Schädigung von Umwelt oder Geräten</p> <p>Wenn der Hinweis neben diesem Symbol nicht beachtet wird, können Umwelt oder Geräte geschädigt werden.</p>
 Hinweis	<p>Tipp oder Fingerzeig</p> <p>Dieses Symbol kennzeichnet Informationen, die zum besseren Verständnis beitragen.</p>

1.3 Ausgabestände der Dokumentation

Version	Kommentar
1.2.0	<ul style="list-style-type: none"> • Anwendungsentwicklung in Safety C aktualisiert
1.1.0	<ul style="list-style-type: none"> • Beschreibung der sicheren Hilfsfunktionen hinzugefügt • Generelle Überarbeitung aller Kapitel
1.0.0	<ul style="list-style-type: none"> • Erste freigegebene Version • Zertifikat hinzugefügt • Diagnose-Daten aktualisiert
0.3	<ul style="list-style-type: none"> • Update Zielgruppe • Update Betrieb - Aufbau der Hardware-Plattform
0.2	<ul style="list-style-type: none"> • Vorläufige Version zur Zertifizierung
0.0.1	<ul style="list-style-type: none"> • Erster Entwurf. Nur für den internen Gebrauch.

2 Systembeschreibung

2.1 Erweiterung des Beckhoff I/O-Systems mit Funktionen für die Sicherheitstechnik

Beckhoff bietet mit den TwinSAFE Produkten die Möglichkeit, das Beckhoff I/O-System einfach mit Komponenten für die Sicherheitstechnik zu erweitern und die gesamte Verkabelung für den Sicherheitskreis mit in das vorhandene Feldbuskabel zu überführen. Die sicheren Signale lassen sich mit Standard-Signalen beliebig mischen. Das Übermitteln der sicherheitsgerichteten TwinSAFE Telegramme wird von der Standard-Steuerung durchgeführt. Die Wartung wird durch schnellere Diagnose und leichten Austausch der Komponenten deutlich vereinfacht.

Folgende Grundfunktionalitäten sind in den TwinSAFE-Komponenten enthalten: digitale Eingänge (z.B. EL19xx, EP1908), digitale Ausgänge (z.B. EL29xx), Antriebskomponenten (z.B. AX5805) und Logikeinheiten (z.B. EL6900, EL6910, TwinCAT Safety PLC). Bei einer Vielzahl von Anwendungen kann die gesamte sicherheitsgerichtete Sensorik und Aktorik auf diese Komponenten verdrahtet werden. Die notwendige logische Verknüpfung der Eingänge mit den Ausgängen führt die EL69xx oder die TwinCAT Safety PLC durch. Mit der EL6910 sind neben booleschen Operationen auch analoge Operationen möglich. Die TwinCAT Safety PLC bietet die Möglichkeit der Erstellung der sicherheitsgerichteten Logik in Safety C.

2.2 TwinCAT Safety PLC

Die TwinCAT Safety PLC dient zur Realisierung der Verknüpfungen zwischen sicherheitsgerichteten Ein- und Ausgängen über das Safety-over-EtherCAT Protokoll (FSoE).

Die TwinCAT Safety PLC erfüllt die Anforderungen der IEC 61508:2010 SIL 3 und EN ISO 13849-1:2015 (Cat 4, PL e).

Die TwinCAT Safety PLC realisiert eine sicherheitsgerichtete Laufzeitumgebung auf einem Standard Industrie-PC. Aktuell sind nur Beckhoff IPCs zulässig. Genauere Informationen zu zulässigen Konfigurationen finden Sie im Dokument „Liste der zulässigen Systemkonfigurationen“ auf der Beckhoff Homepage.

Die sicherheitsgerichtete Logik kann in Safety C oder zukünftig auch über den grafischen TwinSAFE Editor erstellt werden.

2.3 Sicherheitskonzept

TwinSAFE: Sicherheits- und I/O-Technik in einem System

- Erweiterung des bekannten Beckhoff I/O-Systems um TwinSAFE-Komponenten
- beliebige Mischung von sicheren und nicht-sicheren Komponenten
- logische Verknüpfung der I/Os in der TwinCAT Safety PLC
- geeignet für Anwendungen bis SIL 3 nach EN 61508:2010 und Cat 4, PL e nach EN ISO 13849-1:2015
- sicherheitsrelevante Vernetzung von Maschinen über Bussysteme realisierbar
- Jede TwinSAFE Komponente schaltet im Fehlerfall immer in den energielosen und somit sicheren Zustand
- Keine sicherheitstechnischen Anforderungen an das überlagerte Standard-TwinCAT-System

Safety-over-EtherCAT Protokoll (FSoE)

- Übertragung sicherheitsrelevanter Daten über beliebige Medien („echter schwarzer Kanal“)
- TwinSAFE-Kommunikation über Feldbussysteme, wie z.B. EtherCAT, Lightbus, PROFIBUS, PROFINET oder Ethernet

- erfüllt IEC 61508:2010 SIL 3
- FSoE ist IEC Standard (IEC 61784-3-12) und ETG Standard (ETG.5100)

Fail-Safe Prinzip (Fail Stop)

Der Grundsatz bei einem sicherheitstechnischen System wie TwinSAFE ist, dass ein Ausfall eines Bauteils, einer System-Komponente, oder des Gesamtsystems nie zu einem gefährlichen Zustand führen darf. Der sichere Zustand ist immer der abgeschaltete und energielose Zustand.





VORSICHT


Sicherer Zustand

Bei allen TwinSAFE-Komponenten ist der sichere Zustand immer der abgeschaltete und energielose Zustand.

3 Produktbeschreibung





 WARNUNG	<p>Systemgrenzen</p> <p>Die TwinCAT Safety PLC ist nur für Hardware-Plattformen freigegeben, die in der Liste der Systemkonfigurationen aufgeführt sind. Der TwinSAFE Editor für das Engineering und die TwinCAT Safety PLC Runtime müssen auf physikalisch unterschiedlichen PCs installiert und verwendet werden.</p>
 Hinweis	<p>Software-Umgebung</p> <p>Für die Nutzung des vollständigen Funktionsumfangs der TwinCAT Safety PLC wird Visual Studio 2015 Professional oder eine neuere Version benötigt.</p>



3.1 Bestimmungsgemäße Verwendung

 WARNUNG	<p>Vorsicht Verletzungsgefahr!</p> <p>Eine Verwendung der TwinCAT Safety PLC, die über den im Folgenden beschriebene bestimmungsgemäße Verwendung hinausgeht, ist nicht zulässig!</p>
---	--

Die TwinCAT Safety PLC erweitert das Einsatzfeld des Beckhoff I/O Systems um Funktionen, die es erlauben, diese auch im Bereich der Maschinensicherheit einzusetzen. Das angestrebte Einsatzgebiet der TwinCAT Safety PLC sind Sicherheitsfunktionen an Maschinen und die damit unmittelbar zusammenhängenden Aufgaben in der industriellen Automatisierung. Sie sind daher nur für Anwendungen mit einem definierten Fail-Safe-Zustand zugelassen. Dieser sichere Zustand ist der energielose Zustand. Dafür ist eine Fehlersicherheit entsprechend der zugrunde gelegten Normen erforderlich.

Beim Softwareanteil der TwinCAT Safety PLC handelt es sich um eine Software-Sicherheitssteuerung, welche nur auf zulässigen Systemkonfigurationen (bestehend aus Entwicklungsumgebung, Laufzeitumgebung und Hardware-Plattform) genutzt werden darf.

 VORSICHT	<p>Zulässige Systemkonfigurationen</p> <p>Vom Zertifikat der TwinCAT Safety PLC sind nur Systemkonfigurationen abgedeckt, die im Dokument „Liste der zulässigen Systemkonfigurationen“ aufgeführt sind. Alle nicht im Dokument „Liste der zulässigen Systemkonfigurationen“ aufgeführten Systemkonfigurationen fallen nicht unter die Gültigkeit des Zertifikates der TwinCAT Safety PLC. Der Nachweis eines erreichten Sicherheitslevels für Applikationen mit abweichenden Systemkonfigurationen ist kundenseitig zu erbringen.</p>
 VORSICHT	<p>Maschinenrichtlinie beachten</p> <p>Die TwinCAT Safety PLC und die TwinSAFE-Klemmen dürfen nur in Maschinen im Sinne der Maschinenrichtlinie eingesetzt werden.</p>
 VORSICHT	<p>Rückverfolgbarkeit sicherstellen</p> <p>Der Betreiber hat die Rückverfolgbarkeit der Geräte über die Seriennummer sicherzustellen.</p>
 VORSICHT	<p>Verwendeter Industrie-PC</p> <p>Bitte beachten Sie die technischen Daten des verwendeten Industrie-PCs und stellen Sie sicher, dass er nur bestimmungsgemäß verwendet wird.</p>

 VORSICHT	Security Die TwinCAT Safety PLC wird als abgeschlossenes System betrachtet. Entsprechend muss das Thema Security für die Einzelbestandteile Entwicklungsrechner und Laufzeitumgebung durch den Anwender bewertet und realisiert werden.
 Achtung	Benutzername und Kennwort Der Anwender hat dafür Sorge zu tragen, dass seine Zugangsdaten unberechtigten Personen nicht zugänglich sind.

3.2 Technische Daten

Produktbezeichnung	TwinCAT Safety PLC
Anzahl der Eingänge	0
Anzahl der Ausgänge	0
Statusanzeige	entsprechend genutzter Hardware-Plattform
Minimale/Maximale Zykluszeit	ca. 500 µs / entsprechend Projektgröße
Fehlerreaktionszeit	≤ Watchdog-Zeiten
Watchdog-Zeit	min. 1 ms, max. 60000 ms
Eingangsprozessabbild	Dynamisch entsprechend der TwinSAFE Konfiguration in TwinCAT 3
Ausgangsprozessabbild	Dynamisch entsprechend der TwinSAFE Konfiguration in TwinCAT 3
Versorgungsspannung (SELV/PELV)	entsprechend genutzter Hardware-Plattform (siehe Dokument „Liste der zulässigen Systemkonfigurationen“)
zulässige Umgebungstemperatur (Betrieb)	0 °C bis +55 °C (falls nicht in den technischen Daten der Hardware-Plattform anders angegeben)
zulässige Umgebungstemperatur (Transport/Lagerung)	-25 °C bis +65 °C (falls nicht in den technischen Daten der Hardware-Plattform anders angegeben)
zulässige Luftfeuchtigkeit	5% bis 95%, nicht kondensierend (falls nicht in den technischen Daten der Hardware-Plattform anders angegeben)
zulässiger Luftdruck (Betrieb/Lagerung/Transport)	750 hPa bis 1100 hPa (falls nicht in den technischen Daten der Hardware-Plattform anders angegeben) (diese Angabe entspricht einer Höhe von ca. -690 m bis 2450 m über N.N. bei Annahme einer internationalen Standardatmosphäre)
Klimaklasse nach EN 60721-3-3	3K3 (falls nicht in den technischen Daten der Hardware-Plattform anders angegeben)
zulässiger Verschmutzungsgrad nach EN 60664-1	Verschmutzungsgrad 2 (falls nicht in den technischen Daten der Hardware-Plattform anders angegeben)
Unzulässige Betriebsbedingungen	TwinSAFE-Komponenten dürfen unter folgenden Betriebsbedingungen nicht eingesetzt werden: <ul style="list-style-type: none"> • unter dem Einfluss ionisierender Strahlung (die das Maß der natürlichen Umgebungsstrahlung überschreitet) • in korrosivem Umfeld • in einem Umfeld, das zu unzulässiger Verschmutzung der Hardware-Plattform führt
Vibrations- / Schockfestigkeit	gemäß EN 60068-2-6 / EN 60068-2-27
EMV-Festigkeit / Aussendung	gemäß EN 61000-6-2 / EN 61000-6-4
Schocken	entsprechend genutzter Hardware-Plattform (siehe Dokument „Liste der zulässigen Systemkonfigurationen“)
Schutzart	IP20
zulässige Einbaulage	entsprechend genutzter Hardware-Plattform (siehe Dokument „Liste der zulässigen Systemkonfigurationen“)
Zulassungen	TÜV SÜD

3.3 Sicherheitstechnische Kenngrößen


Kennzahlen	TwinCAT Safety PLC
Lifetime [a]	nicht anwendbar (falls für Berechnungen ein Wert benötigt wird, können 20 a angenommen werden)
Proof-Test Intervall [a]	nicht erforderlich ¹⁾
PFH _D	5.5E-10
%SIL3 vom PFH _D	0.55%
PFD _{avg}	5.5E-10
%SIL3 vom PFD _{avg}	0.000055%
MTTF _D	hoch
DC	> 99%
Performance Level	PL e
Kategorie	4
HFT	0
Klassifizierung Element ²⁾	Typ B

1. Spezielle Proof-Tests während der gesamten Lebensdauer der TwinCAT Safety PLC sind nicht erforderlich.
2. Klassifizierung nach IEC 61508-2:2010 (siehe Kapitel 7.4.4.1.2 und 7.4.4.1.3)

Die TwinCAT Safety PLC kann für sicherheitsgerichtete Applikationen im Sinne der IEC 62061:2005/ A2:2015 SIL3, IEC 61508:2010 bis SIL3 und der EN ISO 13849-1:2015 bis PL e (Cat4) eingesetzt werden.

Zur Berechnung bzw. Abschätzung des MTTF_D Wertes aus dem PFH_D Wert finden Sie weitere Informationen im Applikationshandbuch TwinSAFE oder in der EN ISO 13849-1:2015 Tabelle K.1.

In den sicherheitstechnischen Kenngrößen ist die Safety-over-EtherCAT Kommunikation mit 1% des SIL3 entsprechend der Protokoll-Spezifikation bereits berücksichtigt.




 GEFAHR	<p>Auftreten schwerwiegender interner Fehler während der Abarbeitung</p> <p>Tritt mehr als ein erkennbarer schwerwiegender Fehler pro Stunde auf, darf die Anlage nicht weiter betrieben werden.</p> <p>In diesem Fall prüfen Sie bitte zunächst die unter Bestimmungsgemäße Verwendung aufgeführten Rahmenbedingungen und die technischen Daten der verwendeten Hardware-Plattform.</p> <p>Besteht das Problem weiterhin, kontaktieren Sie den Beckhoff Support.</p>
--	--

3.4 Projektierungsgrenzen

Die Projektierungsgrenzen sind abhängig von der Lizenzierung. Es gibt unterschiedliche Lizenzen, die jeweils eine unterschiedliche maximal zulässige Anzahl an FSoE-Verbindungen beinhalten.

4 Betrieb

Stellen Sie sicher, dass die TwinCAT Safety PLC nur bei jeweils für die Hardware-Plattform spezifizierten Umgebungsbedingungen (siehe technische Daten der entsprechenden Hardware-Plattform) transportiert, gelagert und betrieben werden!

 WARNUNG	<p>Verletzungsgefahr!</p> <p>Die TwinSAFE-Komponenten dürfen unter folgenden Betriebsbedingungen nicht eingesetzt werden.</p> <ul style="list-style-type: none"> • unter dem Einfluss ionisierender Strahlung (die das Maß der natürlichen Umgebungsstrahlung überschreitet) • in korrosivem Umfeld • in einem Umfeld, das zu unzulässiger Verschmutzung der Hardware-Plattform führt
 Achtung	<p>Elektromagnetische Verträglichkeit</p> <p>Die TwinSAFE-Komponenten entsprechen den Anforderungen der geltenden Normen zur elektromagnetischen Verträglichkeit in Bezug auf Störausstrahlung und insbesondere auf Störfestigkeit.</p> <p>Sollten jedoch in der Nähe der TwinSAFE-Komponenten Geräte (z.B. Funktelefone, Funkgeräte, Sendeanlagen oder Hochfrequenz-Systeme) betrieben werden, welche die in den Normen festgelegten Grenzen zur Störaussendung überschreiten, können diese ggf. die Funktion der TwinSAFE-Komponenten stören.</p>
 Achtung	<p>Verwendung der Hardware-Plattform</p> <p>Die Hardware-Plattform (siehe Liste der zulässigen Systemkonfigurationen), auf der die TwinCAT Safety PLC installiert und verwendet werden soll, darf nur in Maschinen verwendet werden, die gemäß den Anforderungen der Norm EN 60204-1 Kapitel 4.4.2 aufgebaut und installiert sind.</p>
 Achtung	<p>Aufbau der Hardware-Plattform</p> <p>Der Aufbau der Hardware-Plattform muss so realisiert werden, dass „Common Mode“-Störungen gemäß IEC 61000-4-16 im Frequenzbereich von 0 Hz bis 150 kHz nicht auftreten können.</p>

4.1 Installation

4.1.1 Sicherheitshinweise

Lesen Sie vor Installation und Inbetriebnahme der TwinCAT Safety PLC sowohl die Sicherheitshinweise im Vorwort dieser Dokumentation als auch die Sicherheitshinweise in der entsprechenden Dokumentation der verwendeten Hardware-Plattform.

4.1.2 Transportvorgaben / Lagerung

Hinweise zu den Transportvorgaben und der Lagerung finden Sie in der jeweiligen Dokumentation der eingesetzten Hardware-Plattform.

4.1.3 Mechanische Installation

Hinweise zur mechanischen Installation finden Sie in der jeweiligen Dokumentation der eingesetzten Hardware-Plattform. Achten Sie besonders auf die zulässige Einbaulage.

4.1.4 Elektrische Installation

Hinweise zur elektrischen Installation finden Sie in der jeweiligen Dokumentation der eingesetzten Hardware-Plattform.

4.1.5 Software Installation

Die TwinCAT Safety PLC wird immer zusammen mit TwinCAT installiert. Die Installation von TwinCAT 3.1 Build 4022 oder größerer Buildnummer enthält immer die jeweils aktuelle, freigegebene Version der TwinCAT Safety PLC. Über die Lizenzierung wird die Nutzung entsprechend freigeschaltet.

4.1.6 Reaktionszeiten TwinSAFE

Die TwinSAFE-Klemmen zusammen mit der TwinCAT Safety PLC bilden ein modular aufgebautes Sicherheitssystem, welches über das Safety-over-EtherCAT-Protokoll sicherheitsgerichtete Daten austauscht. Dieses Kapitel soll dabei helfen die Reaktionszeit des Systems vom Signalwechsel am Sensor bis zur Reaktion am Aktor zu bestimmen.

Typische Reaktionszeit

Die typische Reaktionszeit ist die Zeit, die benötigt wird, um eine Information vom Sensor zum Aktor zu übermitteln, wenn das Gesamtsystem fehlerfrei im Normalbetrieb arbeitet.

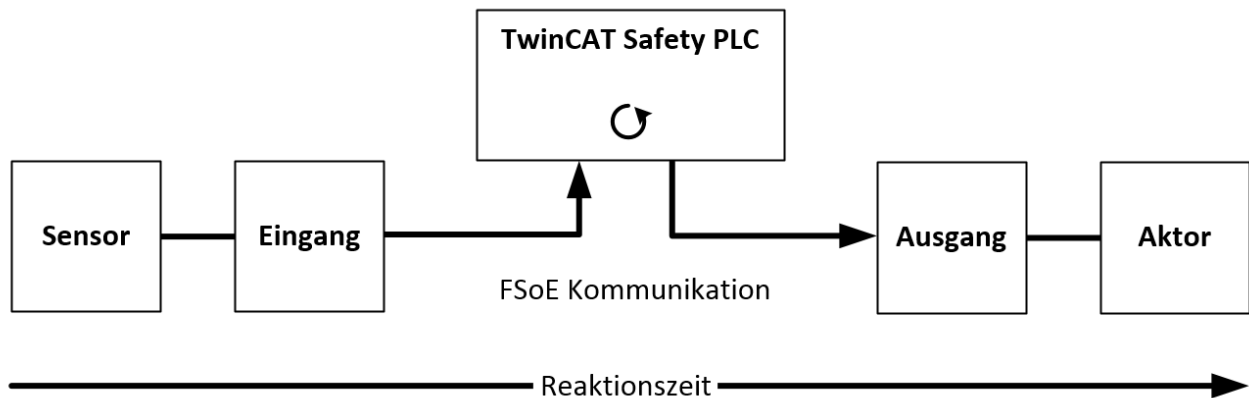


Abb. 1: Typische Reaktionszeit

Definition	Beschreibung
RTSensor	Reaktionszeit des Sensors, bis das Signal an der Schnittstelle zur Verfügung gestellt wird. Wird typischerweise vom Sensorhersteller geliefert.
RTInput	Reaktionszeit des sicheren Eingangs, wie z.B. EL1904 oder EP1908. Diese Zeit kann aus den technischen Daten entnommen werden. Bei der EL1904 sind dies z.B. 4 ms.
RTComm	Reaktionszeit der Kommunikation. Diese ist typischerweise 3x die EtherCAT Zykluszeit, da neue Daten immer erst in einem neuen Safety-over-EtherCAT Telegramm versendet werden können. Diese Zeiten hängen von der Standard-Steuerung direkt ab (Zykluszeit der PLC/ NC/SafetyTask). Hierbei ist zu berücksichtigen, welche Task den EtherCAT-Strang synchron ansteuert.
RTLogic	Reaktionszeit der TwinCAT Safety PLC. Dieses ist die Zykluszeit der Task, in der die TwinCAT Safety PLC ausgeführt wird, wenn keine Überschreitungszähler auftreten.
RTOutput	Reaktionszeit der Ausgangsklemme. Diese liegt typischerweise im Bereich von 2 bis 3 ms.
RTActor	Reaktionszeit des Aktors. Diese Information wird typischerweise vom Aktor-Hersteller geliefert
WDComm	Watchdog-Zeit der Kommunikation

Es ergibt sich für die typische Reaktionszeit folgende Formel:

$$Reaktionszeit_{typ} = RT_{Sensor} + RT_{Input} + 3 * RT_{Comm} + RT_{Logic} + 3 * RT_{Comm} + RT_{output} + RT_{Actor}$$

mit z.B.

$$Reaktionszeit_{typ} = 5ms + 4ms + 3 * 1ms + 1ms + 3 * 1ms + 3ms + 15ms = 34ms$$

Worst-Case-Reaktionszeit

Die Worst-Case-Reaktionszeit gibt die Zeit an, die maximal benötigt wird, um im Fehlerfall ein Abschalten des Aktors durchzuführen.

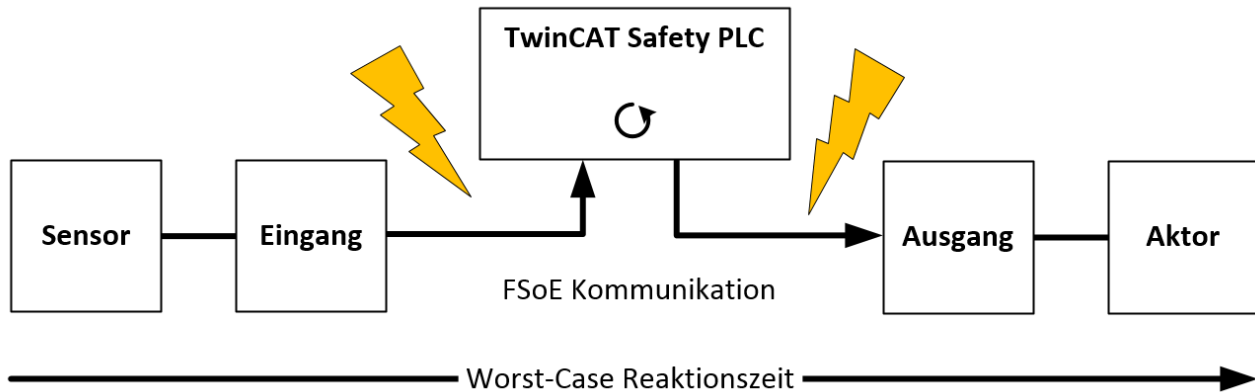


Abb. 2: Worst-Case Reaktionszeit

Dabei wird davon ausgegangen, dass am Sensor ein Signalwechsel erfolgt und dieser an den Eingang übermittelt wird. Gerade in dem Moment, wo das Signal an die Kommunikationsschnittstelle übergeben werden soll, tritt eine Kommunikationsstörung auf. Dies wird nach Ablauf der Watchdog-Zeit der Kommunikationsverbindung von der Logik detektiert. Diese Information soll dann an den Ausgang übergeben werden, wobei hier dann eine weitere Kommunikationsstörung auftritt. Diese Störung wird am Ausgang nach Ablauf der Watchdog-Zeit erkannt und führt dann zur Abschaltung.

Damit ergibt sich für die Worst-Case-Reaktionszeit folgende Formel:

$$Reaktionszeit_{max} = WD_{Comm} + WD_{Comm} + RT_{Actor}$$


mit z.B.

$$Reaktionszeit_{max} = 100ms + 100ms + 15ms = 215ms$$

4.2 Konfiguration der TwinCAT Safety PLC in TwinCAT


4.2.1 Voraussetzung für die Konfiguration

Zur Konfiguration der TwinCAT Safety PLC wird die Automatisierungs-Software TwinCAT Version 3.1 Build 4022 oder höher benötigt. Die jeweils aktuelle Version kann auf den Internetseiten der Firma Beckhoff unter www.beckhoff.de geladen werden.

 Hinweis	TwinCAT Unterstützung Eine Verwendung der TwinCAT Safety PLC unter TwinCAT 2 ist nicht möglich.
---	---

4.2.2 Anlegen eines Safety Projektes in TwinCAT 3

Ein Safety Projekt, welches in Safety C erstellt wird, muss entsprechend der anzuwendenden Normen entwickelt werden. Beachten Sie hierzu auch das Kapitel [Anwendungsentwicklung in Safety C](#) [► 50]

 GEFAHR	Quelltext der Safety Applikation Der Benutzerquelltext ist entsprechend der jeweils anzuwendenden Normen zu entwickeln mit der Grundnorm IEC 61508:2010. Beachten Sie hierzu auch das Kapitel Verifikation und Validierung [► 81].
--	--

4.2.2.1 Add new item

In TwinCAT 3 wird über das Kontextmenu des Knotens *Safety* ein neues Projekt über *Add New Item...* erstellt.

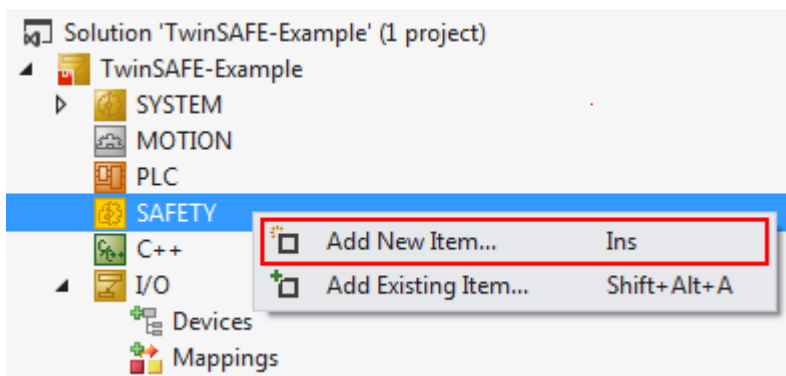


Abb. 3: Anlegen eines Safety Projektes - Add New Item

Der Projektname und das Verzeichnis können frei gewählt werden.

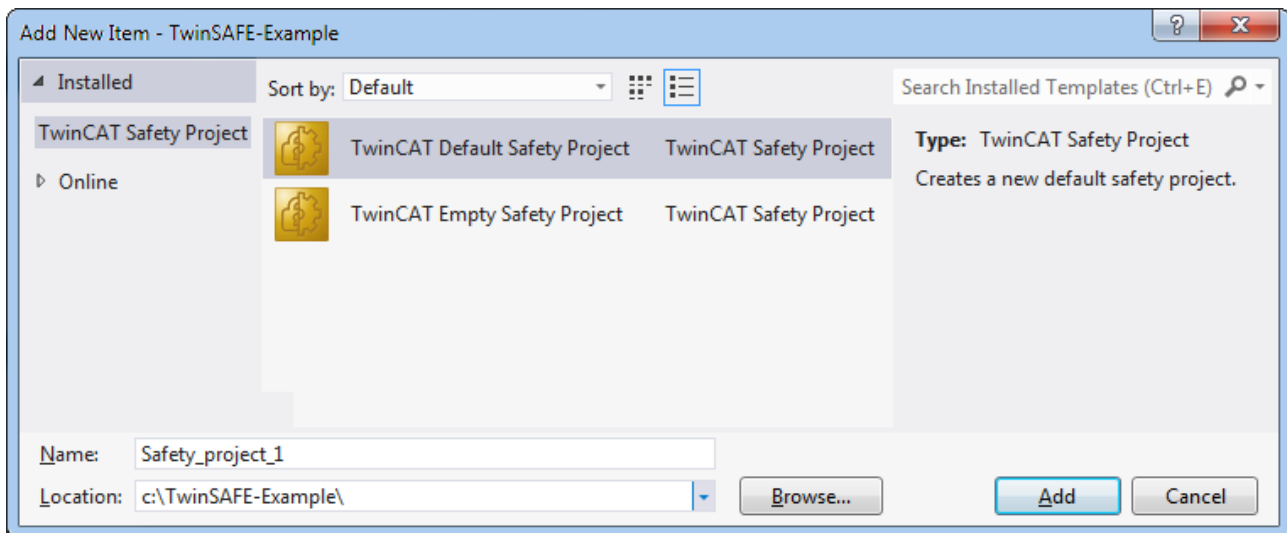


Abb. 4: Anlegen eines Safety Projektes - Projektname und Verzeichnis

4.2.2.2 TwinCAT Safety Project Wizard

Anschließend wählt man im TwinCAT Safety Project Wizard das Target System, die Programmiersprache, den Autor und den internen Projektname aus. Als Target System ist die Einstellung *TwinCAT Safety PLC* und als Programmiersprache *Safety C* zu wählen. Autor und interner Projektname können durch den Anwender frei gewählt werden.

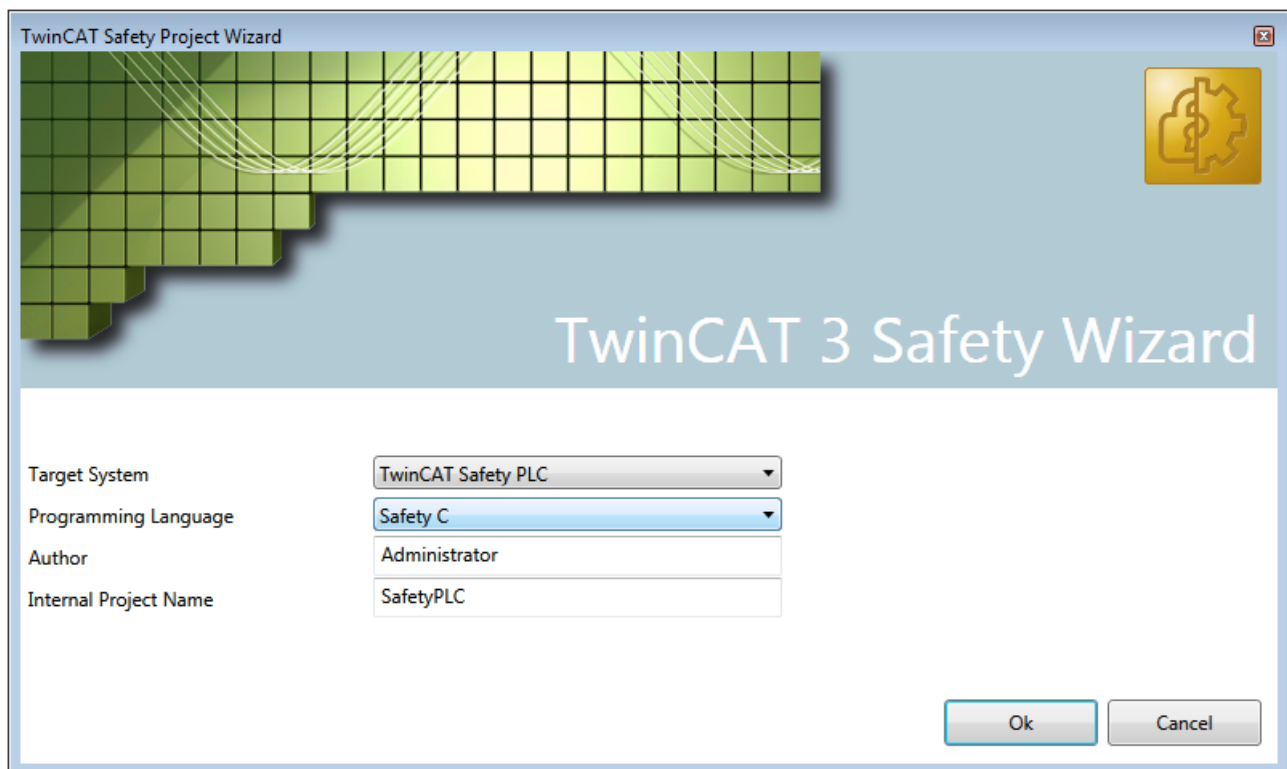


Abb. 5: TwinCAT Safety Project Wizard

4.2.2.3 Target System

Nach Erstellung des Projektes durch den Project Wizard, kann durch Auswahl des Knotens *Target System* eine Zuordnung des Safety Projektes zu der Task durchgeführt werden, mit der die Safety Applikation ausgeführt werden soll.

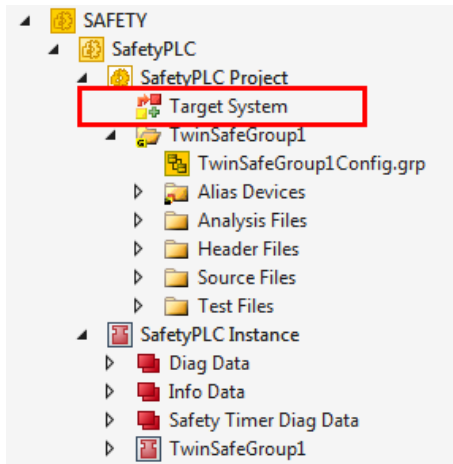


Abb. 6: Target System im Solution Explorer

Das Target System ist in der Drop-Down Liste fest auf *TwinCAT Safety PLC* eingestellt und wird über den Link-Button neben *Append to Task* mit der Task verknüpft, mit der die TwinCAT Safety PLC ausgeführt werden soll.

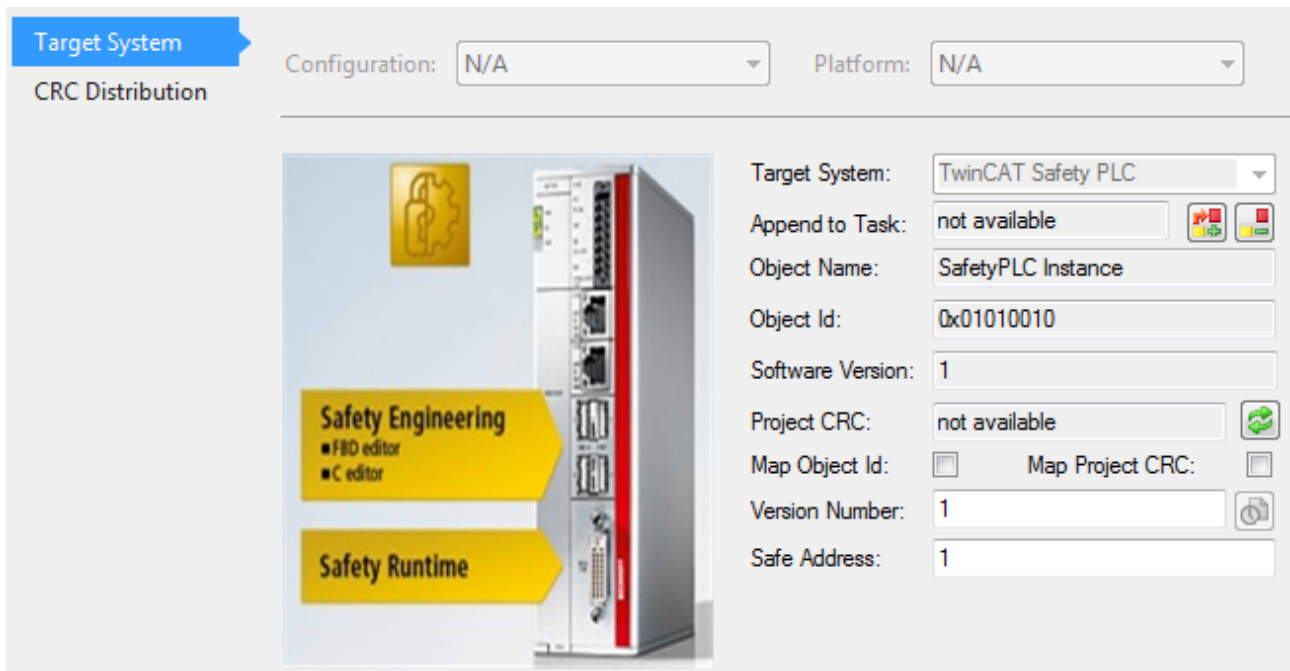


Abb. 7: Target System Property Page

4.2.2.4 TwinSAFE-Gruppen

Das Anlegen von TwinSAFE-Gruppen ist sinnvoll, wenn man unterschiedliche Sicherheitsbereiche einer Maschine realisieren möchte oder unterschiedliche C++ Quell-Dateien nutzen möchte. Innerhalb einer Gruppe führt ein Fehler einer Verbindung (hier Alias Device) zu einem ComError der Gruppe und somit zur Abschaltung aller Ausgänge dieser Gruppe.

Eine weitere Gruppe kann durch Öffnen des Kontextmenüs des Safety Projektes und Auswahl von *Add* und *New Item...* angelegt werden.

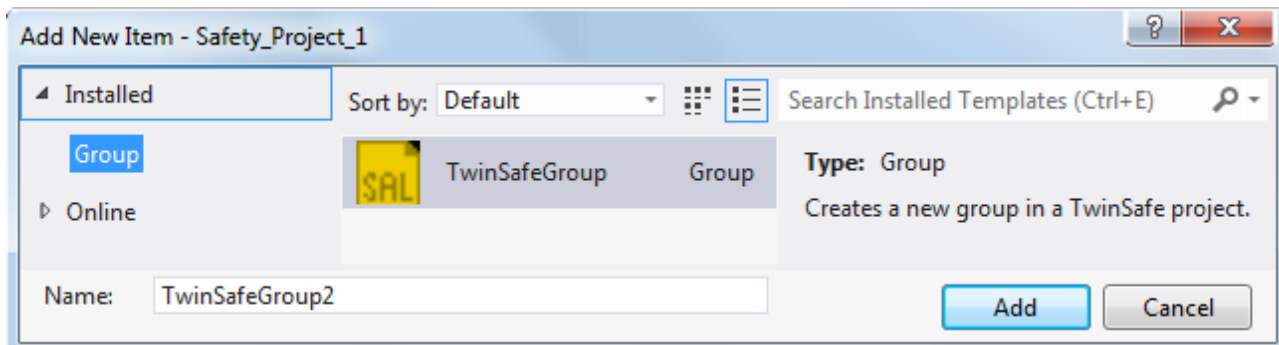


Abb. 8: Anlegen einer TwinSAFE -Gruppe

Eine Gruppe besteht aus Unterpunkten für die Gruppenkonfiguration (*.grp), für die Alias Devices (*.sds), für Header-Dateien (*.h) und für Source-Dateien (*.cpp). Zusätzlich gibt es noch Unterpunkte für Test- und für Analysis-Dateien.

Es gibt pro Gruppe nur genau eine Header-Datei und eine Source-Datei, die der Anwender für die Safety Applikation nutzen und bearbeiten kann. Dies sind die Dateien *<Gruppen-Name>.h* und *<Gruppen-Name>.cpp*.

Die Test-Dateien *ModuleTests.cpp* und *ModuleTests.h* können zum Debuggen der Safety Applikation verwendet werden. In diesen Dateien können die sicheren Ein- und Ausgänge gesetzt werden und bleiben auch gesetzt, wenn Breakpoints verwendet werden, ohne dass die gesamte Konfiguration aktiviert werden muss. In diesem Zustand wird nicht sicher kommuniziert!

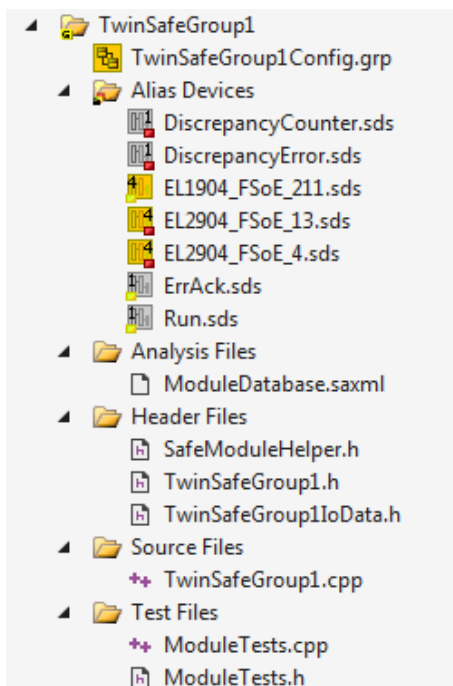


Abb. 9: TwinSAFE-Gruppe

In der Gruppenkonfiguration werden die generellen Einstellungen der Gruppe vorgenommen. Dies sind z.B. die Info-Daten oder Gruppen Ports für Error Acknowledge und Run/Stop.

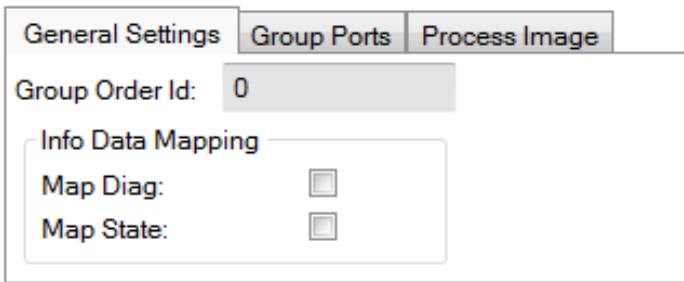


Abb. 10: TwinSAFE Group - General Settings

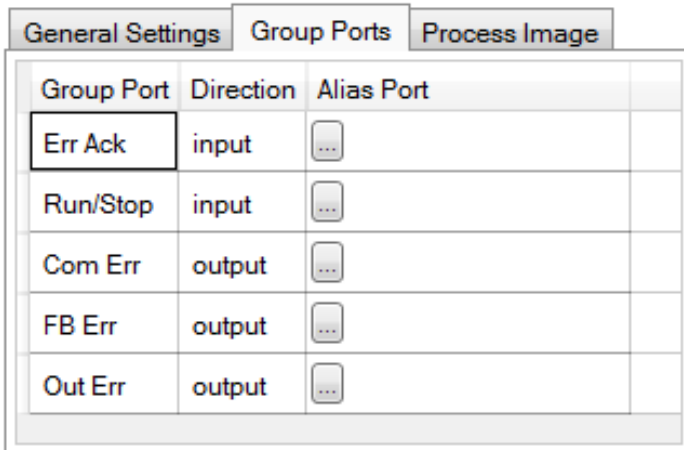



Abb. 11: TwinSAFE Group - Group Ports

Zusätzlich gibt es noch die Möglichkeit, ein internes Prozessabbild für die TwinSAFE-Gruppe zu erstellen. Dieses Prozessabbild enthält alle Signale, die in anderen TwinSAFE-Gruppen verwendet werden sollen. Die definierten Variablen werden in einer Struktur TSGData in der Header-Datei <GruppenName>IoData.h allen anderen Gruppen zur Verfügung gestellt.



Hinweis

TwinSAFE-Gruppen Ausgänge

Bitte achten Sie darauf, dass TwinSAFE-Gruppen nur Ausgänge im TSGData struct haben. Diese Ausgänge können von allen anderen Gruppen gelesen werden. Eingänge einer TwinSAFE Gruppe können nicht definiert werden.

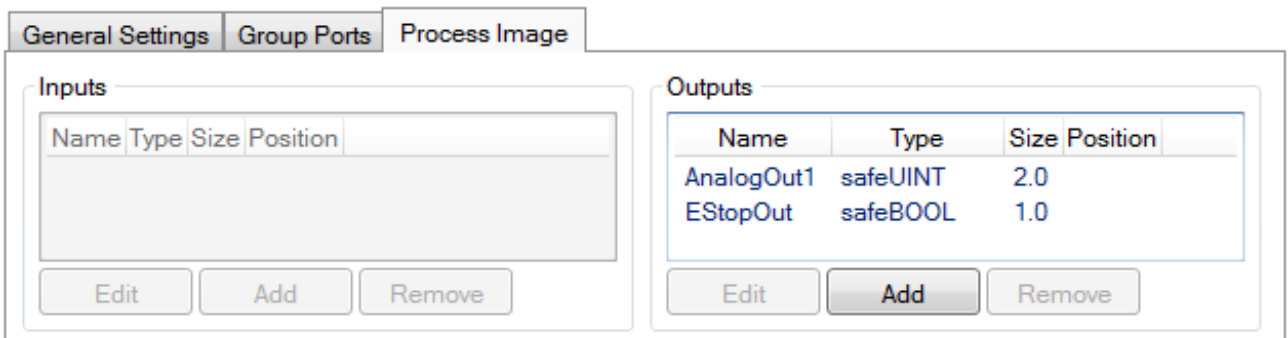


Abb. 12: TwinSAFE-Gruppe - Prozessabbild

```

//! Struct storing the TwinSAFE group exchange data
struct TSGData
{
    //! ..TwinSafeGroup: TwinSafeGroup1
    struct _TwinSafeGroup1
    {
        //! ..Outputs
        struct _Out
        {
            safeUINT AnalogOut1;
            safeBOOL EStopOut;
        } Out;
    } TwinSafeGroup1;
    //! ..TwinSafeGroup: TwinSafeGroup2
    struct _TwinSafeGroup2
    {
        //! ..Outputs
        struct _Out
        {
        } Out;
    } TwinSafeGroup2;
};

```

Abb. 13: TSGData struct

4.2.2.5 Alias Devices

Die Kommunikation zwischen der Safety Logic und der I/O-Ebene wird über einen Alias-Level realisiert. In diesem Alias-Level (Sub-Knoten *Alias Devices*) werden für alle sicheren Ein- und Ausgänge, aber auch für Standard-Signale entsprechende Alias Devices angelegt. Dies kann für die sicheren Ein- und Ausgänge auch automatisch anhand der I/O-Konfiguration durchgeführt werden.

Über die Alias Devices werden die Verbindungs- und Geräte-spezifischen Parameter eingestellt.

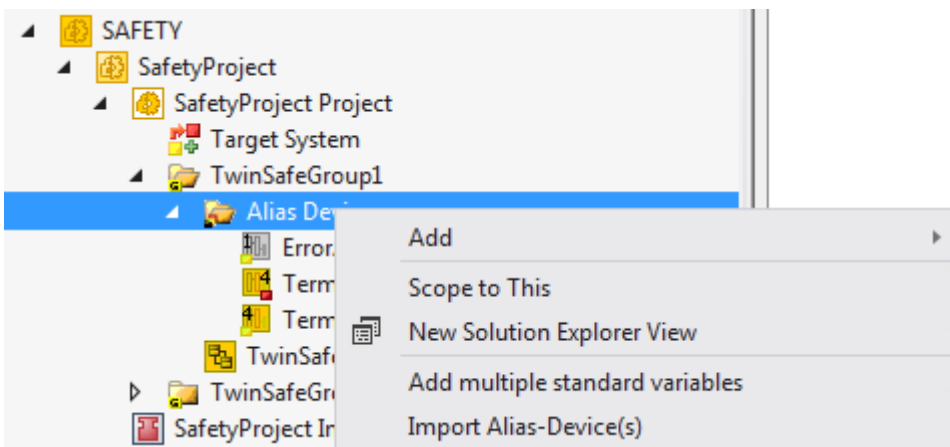


Abb. 14: Starten des automatischen Imports aus der I/O-Konfiguration

Wird der automatische Import aus der I/O-Konfiguration gestartet, wird ein Auswahldialog geöffnet, über den die einzelnen Klemmen, die automatisch importiert werden sollen, selektiert werden können.

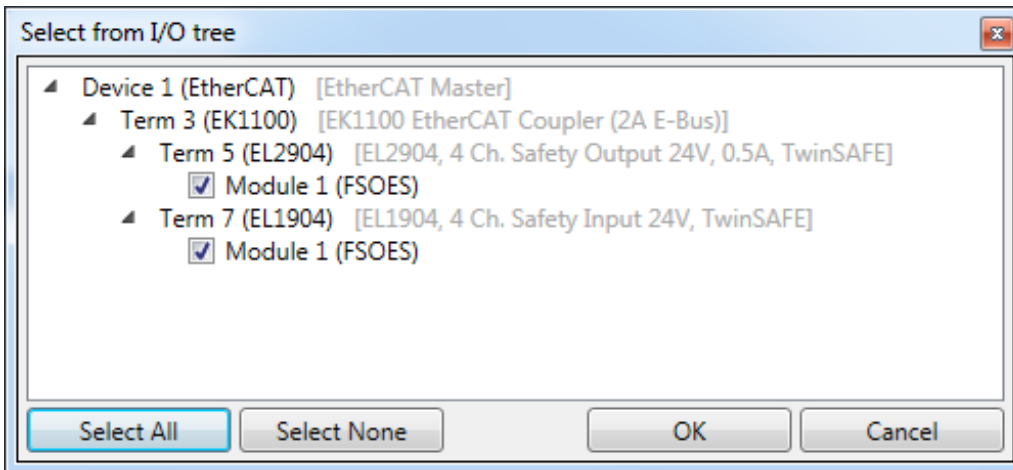


Abb. 15: Auswahl aus dem I/O Baum

Nach dem Schließen des Dialoges über OK, werden die Alias Devices im Safety Projekt angelegt.

Die Alias Devices können auch einzeln durch den Anwender angelegt werden. Dazu wird aus dem Kontextmenu der Eintrag *Add* und *New item* ausgewählt und das gewünschte Gerät ausgewählt.

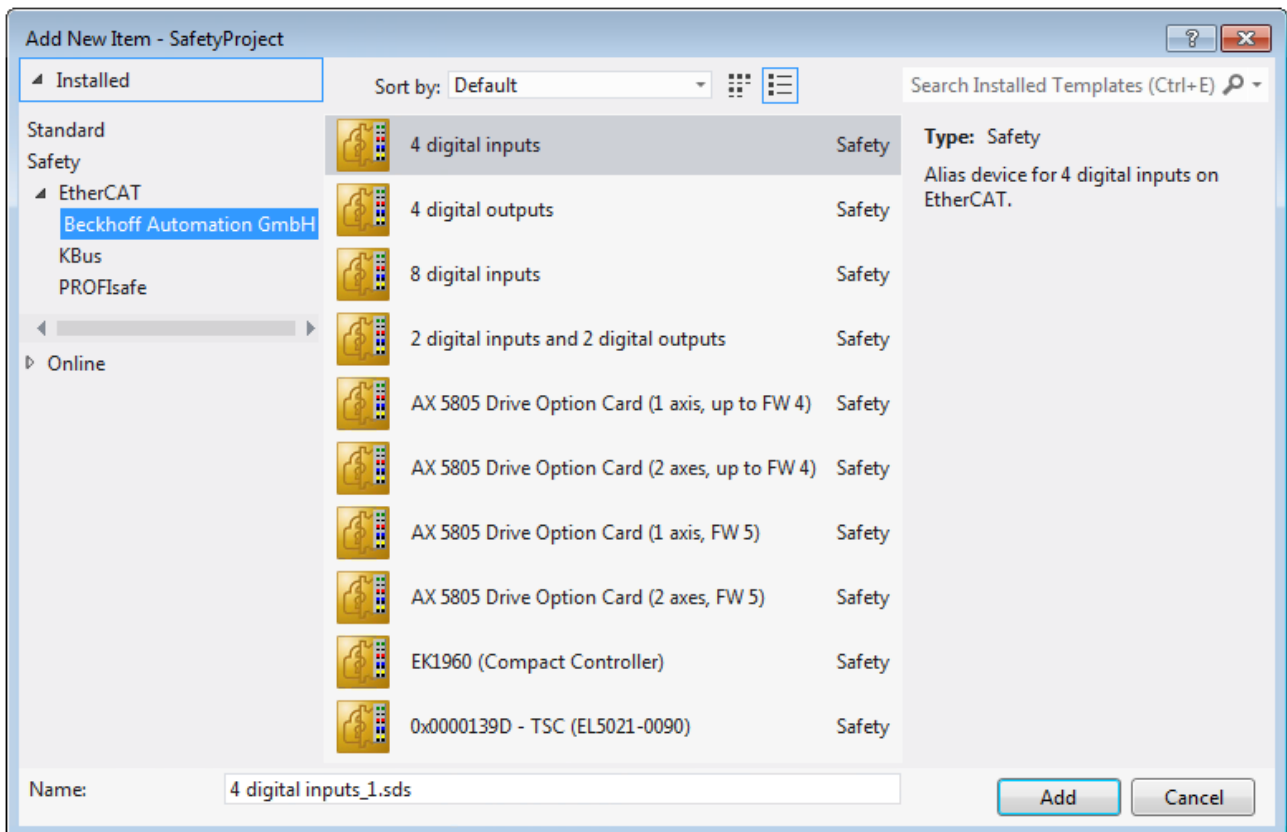


Abb. 16: Anlegen der Alias Devices durch den Anwender

4.2.2.6 Sicheres Zeitsignal

Ein Safety Projekt für die TwinCAT Safety PLC ist nur gültig, wenn für die Ausführung des Safety Projektes ein externes sicheres Zeitsignal zur Verfügung steht. Dazu muss mindestens eine der sicheren Kommunikationsverbindungen über die Funktionalität verfügen, ein sicheres Zeitsignal über die sichere Kommunikationsverbindung zu liefern. Dies kann z.B. eine TwinSAFE Komponente EL6910 sein. Am Beispiel einer TwinSAFE Komponente EL6910 muss in dem Alias Device auf dem Reiter *Process Image* der sichere Zeitwert in das Eingangsprozessabbild gelegt werden.

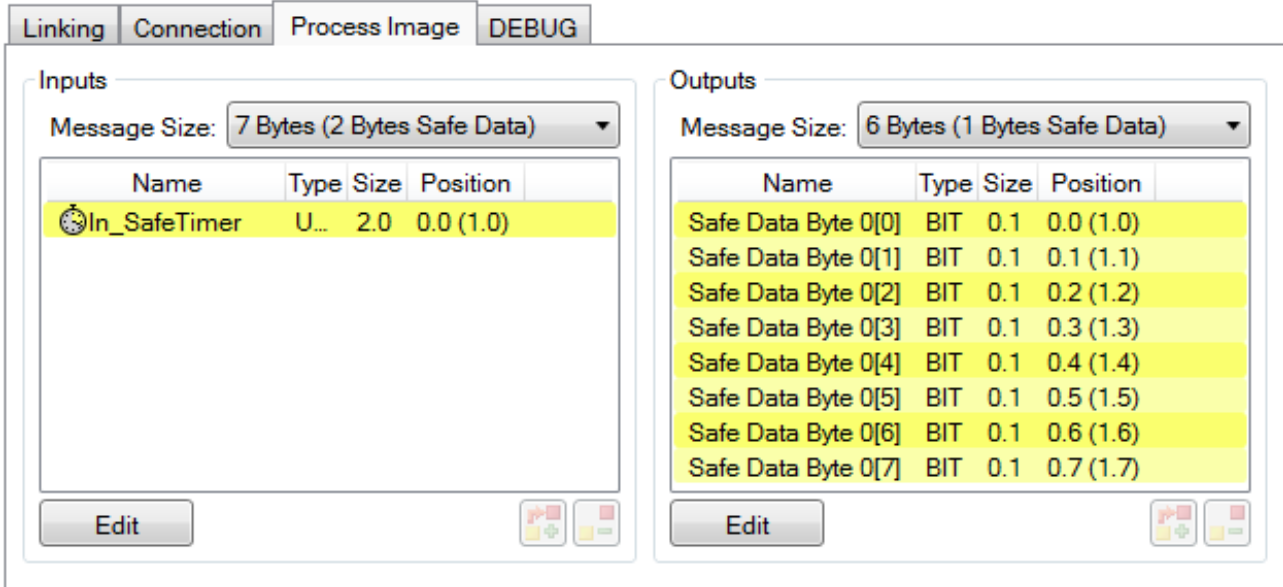


Abb. 17: Alias Device - Reiter Process Image

Durch Auswahl von *Edit* in diesem Dialog kann das Prozessabbild angepasst werden und der SafeTimer hinzugefügt werden.

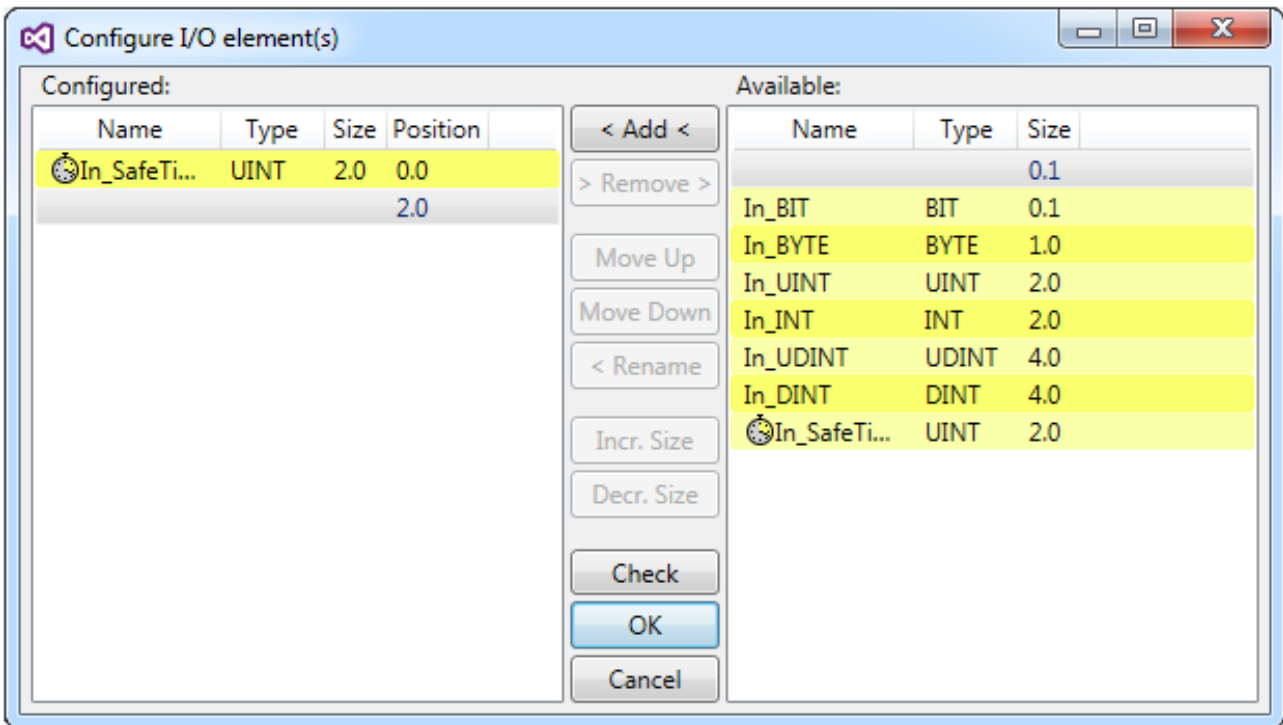


Abb. 18: Konfigurieren der I/O Elemente

Zusätzlich muss auf dem Reiter *Connection* die Checkbox *Use provided Safe Timer as reference* gesetzt werden.

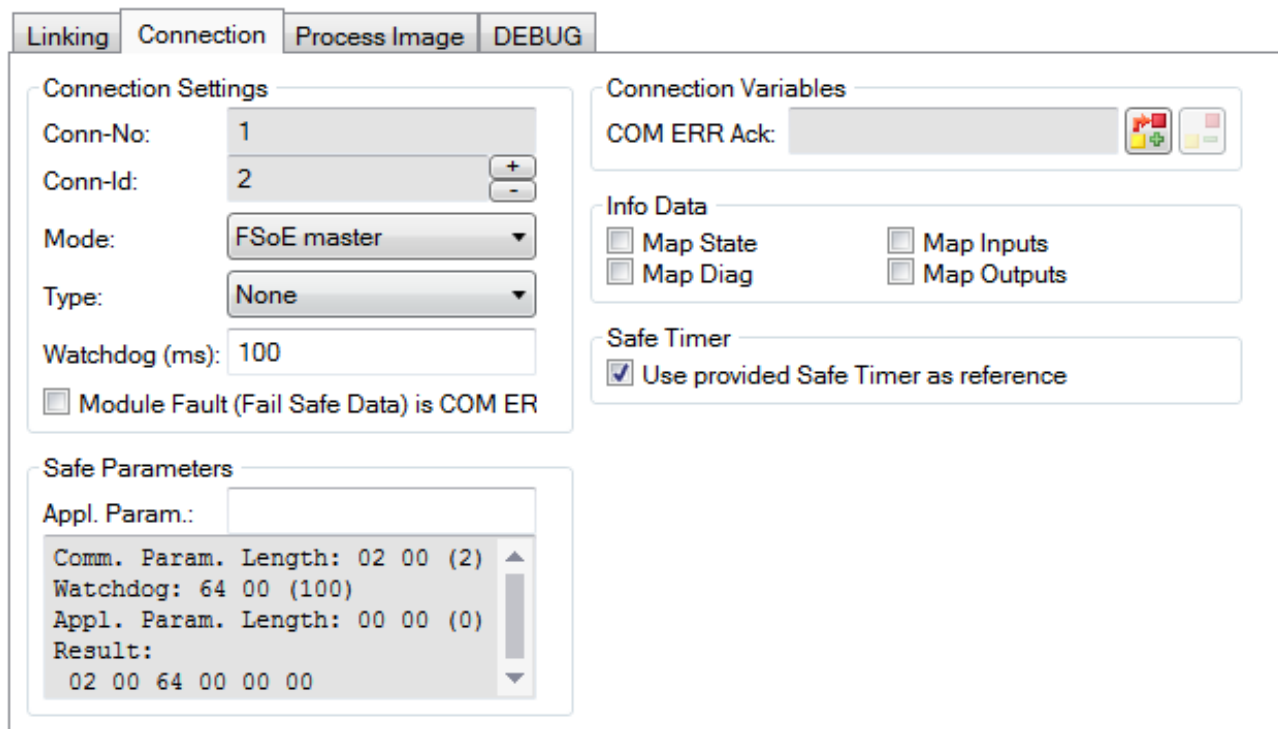


Abb. 19: Alias Device - Reiter Connection

Für ein Safety Projekt muss genau eine TwinSAFE-Komponente als Lieferant eines sicheren Zeitsignals gewählt werden, damit ein Safety Projekt erfolgreich geladen und gestartet werden kann. Das Safety Projekt wird nur ausgeführt, wenn das gelieferte sichere Zeitsignal vorhanden ist (Somit muss die entsprechende Kommunikationsverbindung im Zustand DATA sein.).

Ein Fehler im Kontext des sicheren Zeitsignals führt zum Einleiten des sicheren Zustands der TwinCAT Safety PLC.

4.2.2.7 Parametrierung des Alias Devices

Über einen Doppelklick auf das Alias Device in der Safety-Projektstruktur können die Einstellungen geöffnet werden.

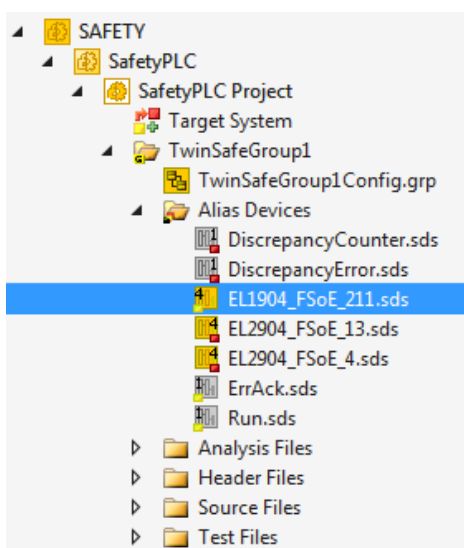



Abb. 20: Alias Device in der Safety-Projektstruktur

Der Reiter *Linking* enthält die FSoE-Adresse, die Checkbox zur Einstellung als *External Device* und den Link zum physikalischen I/O-Gerät. Besteht eine ADS-Online-Verbindung zu dem physikalischen I/O-Gerät, wird

die DIP-Schalter-Einstellung angezeigt. Ein erneutes Lesen der Einstellung kann über den Button  gestartet werden. Unter *Full Name (input)* und *Full Name (output)* werden die Verlinkungen zum TwinCAT Safety PLC-Prozessabbild angezeigt.

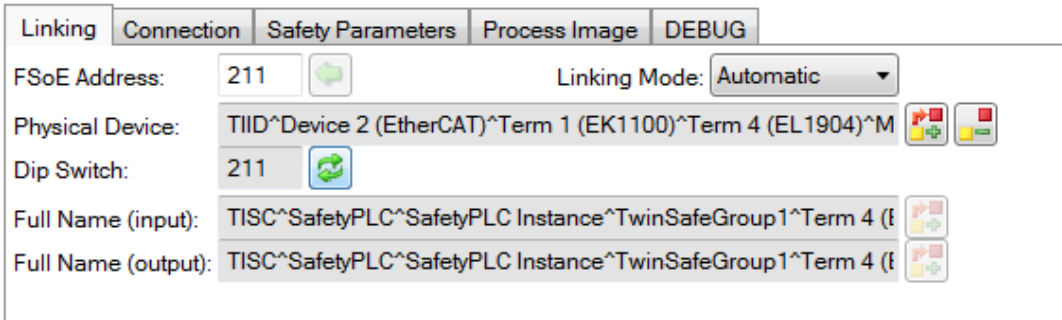


Abb. 21: Verlinkungen zum TwinCAT Safety PLC-Prozessabbild

Der Reiter *Connection* zeigt die verbindungsspezifischen Parameter.

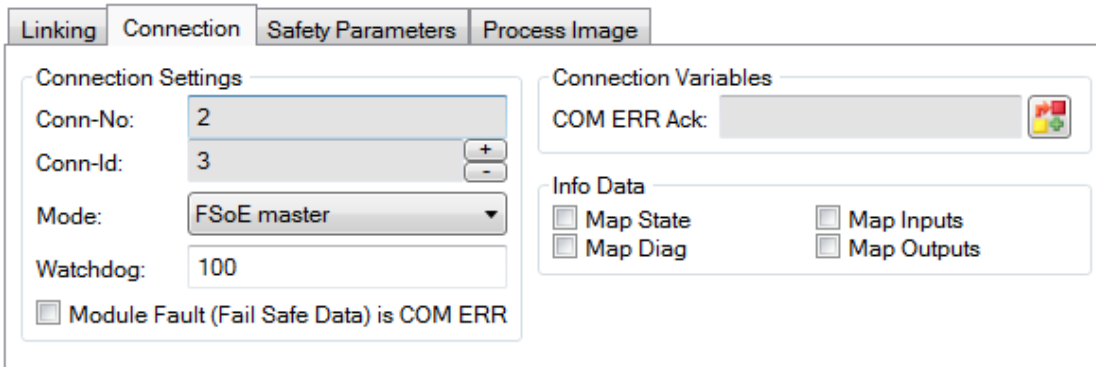



Abb. 22: Verbindungsspezifische Parameter der Connection

Parameter	Beschreibung	Anwender-Interaktion erforderlich
Conn-No.	Verbindungsnummer - wird vom TwinCAT System automatisch vergeben	Nein
Conn-ID	Verbindungs-ID: Wird durch das System vorbelegt, kann durch den Anwender jedoch geändert werden. Innerhalb einer Konfiguration darf eine Conn-ID nur einmal vorkommen. Doppelt vergebene Verbindungs-IDs führen zu einer Fehlermeldung.	Kontrolle
Mode	FSoE Master: TwinCAT Safety PLC ist FSoE-Master zu diesem Gerät. FSoE-Slave: TwinCAT Safety PLC ist FSoE-Slave zu diesem Gerät.	Kontrolle
Watchdog	Watchdog-Zeit für diese Verbindung. Wird innerhalb der Watchdog-Zeit kein gültiges Telegramm vom Gerät zurück zur TwinCAT Safety PLC gesendet, wird ein ComError generiert.	Ja
Module Fault is ComError	Über diese Checkbox stellt man das Verhalten im Fehlerfall ein. Ist die Checkbox gesetzt und tritt auf dem Alias Device ein Modulfehler auf, führt dies zusätzlich zu einem Fehler der Connection und somit zu einer Abschaltung der TwinSAFE-Gruppe, in der diese Verbindung definiert ist.	Ja
ComErrAck	Ist der ComErrAck mit einer Variablen verlinkt, muss die Verbindung im Falle eines Kommunikationsfehlers über dieses Signal zurückgesetzt werden, bevor die entsprechende Gruppe zurückgesetzt werden kann.	Ja
Info Data	Über diese Checkboxes können die Infodaten, die im Prozessabbild der TwinCAT Safety PLC eingeblendet werden sollen, definiert werden. Weitere Informationen finden Sie in der Dokumentation <i>TwinCAT-Funktionsbausteine für TwinSAFE-Logic-Klemmen</i> .	Ja

Die TwinCAT Safety PLC unterstützt an jeder Connection die Aktivierung eines ComErrAck. Ist dieses Signal beschaltet, muss nach einer Kommunikationsstörung zusätzlich zum ErrAck der TwinSAFE Gruppe auch die jeweilige Connection über das Signal ComErrAck zurückgesetzt werden. Dieses Signal wird über den Link

Button  neben COM ERR Ack verknüpft. Über den folgenden Dialog kann der Anwender ein Alias Device auswählen. Ein Löschen des Signals kann über den Button *Clear* im Link Dialog erfolgen.

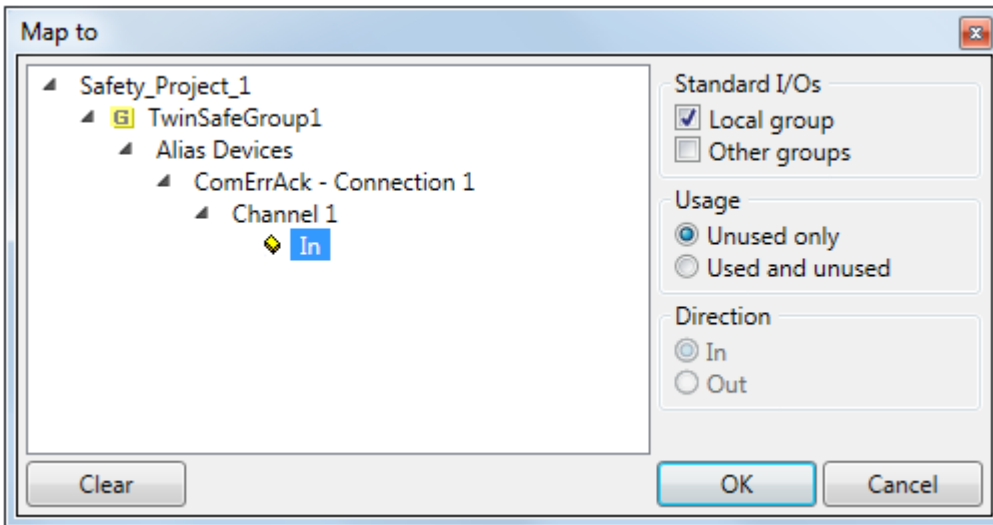


Abb. 23: Auswahl eines Alias Devices

Die zu dem Gerät passenden Safety Parameter werden unter dem Reiter *Safety Parameters* angezeigt. Diese müssen entsprechend des erforderlichen Performance Levels korrekt eingestellt werden. Weiterführende Informationen dazu finden sich auch im TwinSAFE-Applikationshandbuch.

Index	Name	Value	Unit
8000:0	FS Operating Mode	>1<	
8000:01	Operating Mode	digital (0)	
8001:0	FS Sensor Test	>5<	
8001:01	Sensor test Channel 1 active	TRUE (1)	
8001:02	Sensor test Channel 2 active	TRUE (1)	
8001:03	Sensor test Channel 3 active	TRUE (1)	
8001:04	Sensor test Channel 4 active	TRUE (1)	
8002:0	FS Logic of Input pairs	>5<	
8002:01	Logic of Channel 1 and 2	single logic ch...	
8002:03	Logic of Channel 3 and 4	single logic ch...	

Abb. 24: Safety-Parameter des Geräts

Zu jedem Alias Device wird in der Safety PLC Instanz ein Eintrag mit dem FSoE Stack zu dem Gerät angelegt. Er enthält die Verlinkungen zu den sicheren Ein- und Ausgangskomponenten und stellt auch den Data Pointer zur Verfügung, um auf die sicheren Ein- und Ausgänge innerhalb der Safety Applikation zuzugreifen.

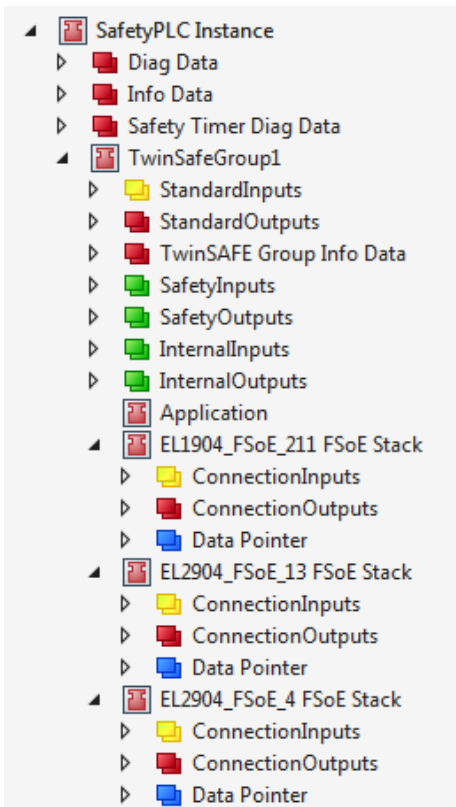


Abb. 25: Safety PLC Instanz - Alias Devices

Die Daten jeder einzelnen Verbindung werden als struct-Datentyp in der <Gruppenname>IoData.h deklariert und in der Header Datei <Gruppenname>.h wird dieser instanziiert.

Der Anwender kann dann über die Instanzvariable z.B. *sSafetyInputs.EL1904_FSoE_211.InputChannel1* direkt auf einen sicheren Eingang zugreifen.

```

//! Struct providing input data of the corresponding safety alias devices
struct SafetyInputs
{
    //! ..\Alias Devices\EL1904_FSoE_211.sds
    struct _EL1904_FSoE_211
    {
        safeBOOL InputChannel1;
        safeBOOL InputChannel2;
        safeBOOL InputChannel3;
        safeBOOL InputChannel4;
    } EL1904_FSoE_211;
};

```

Abb. 26: Struktur des Alias Devices

4.2.2.8 Verbindung zu AX5805/AX5806

Für eine Verbindung zu einer TwinSAFE-Drive-Optionskarte AX5805 bzw. AX5806 gibt es eigene Dialoge, über welche die Sicherheitsfunktionen der AX5000-Safety-Antriebsoptionen eingestellt werden können.

Nach dem Anlegen und Öffnen eines Alias Devices für eine AX5805 erhält man fünf Reiter, wobei die Reiter *Linking*, *Connection* und *Safety Parameters* identisch zu anderen Alias Devices sind.

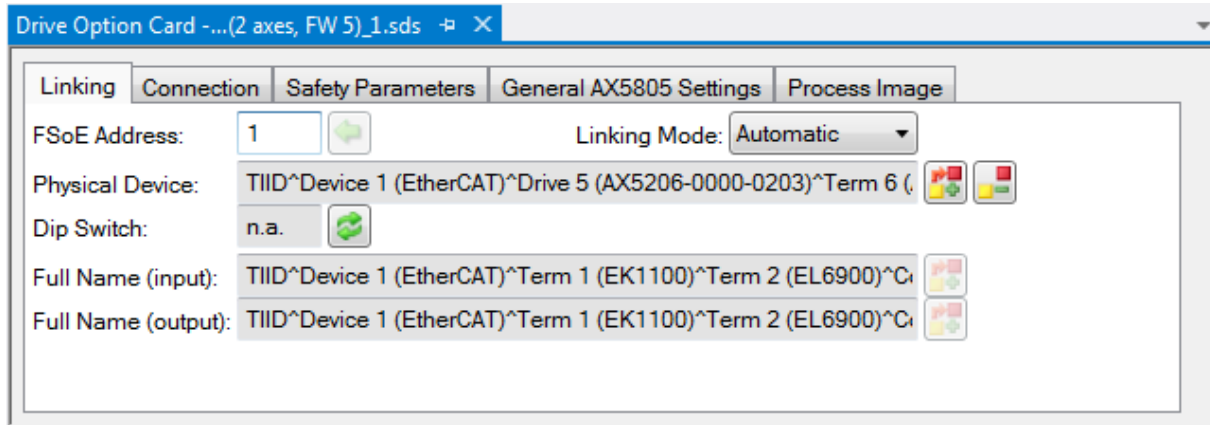


Abb. 27: AX5000-Safety-Antriebsoptionen

Über den Reiter *General AX5805 settings* kann man den Motorstring und die Funktionen SMS und SMA für eine oder zwei Achsen einstellen, je nach eingefügtem AliasDevice.

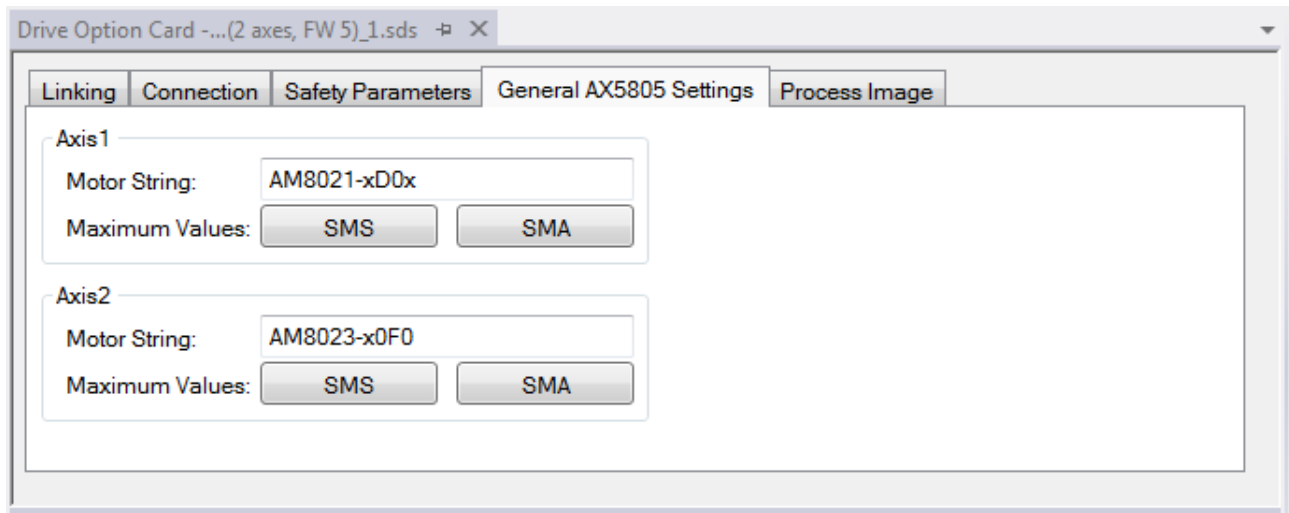


Abb. 28: AX5000-Safety-Antriebsoptionen - General AX5805 settings

Über den Reiter *Process Image* können die unterschiedlichen Sicherheitsfunktionen der AX5805 eingestellt werden.

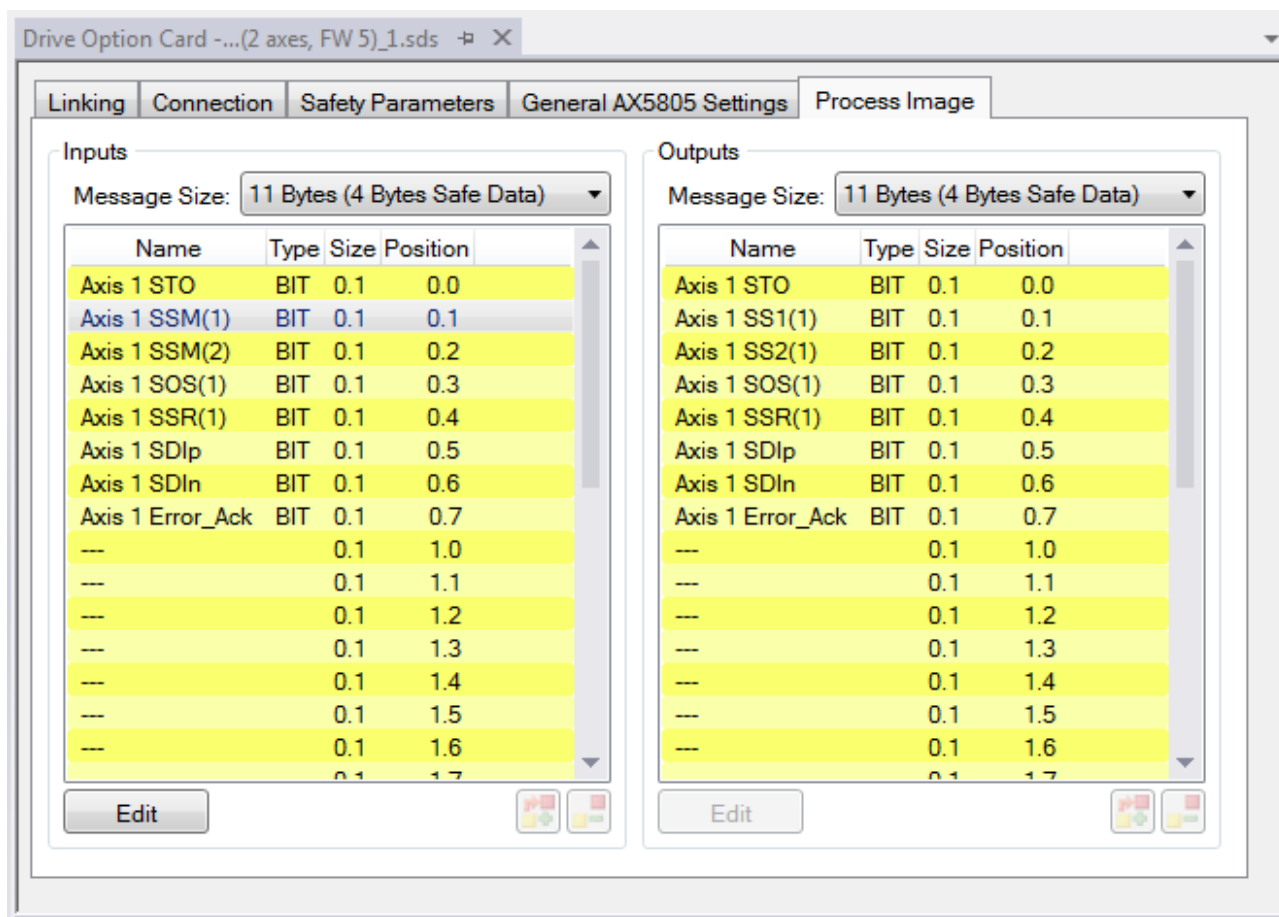


Abb. 29: AX5000-Safety-Antriebsoptionen - Process Image

Die Parameter unter den Reitern *General AX5805 Settings* und *Process Image* sind identisch zu den Parametern unter dem Reiter *Safety Parameters*. Es ist nur eine komfortablere Ansicht und Bearbeitung der Parameter. Eine Bearbeitung der Parameter unter dem Reiter *Safety Parameters* ist ebenfalls möglich.

Durch Markieren einer Funktion in den Inputs oder Outputs und Betätigen des *Edit* Buttons können die Parameter dieser Funktion eingestellt werden. Durch Markieren eines leeren Platzes (---) und Auswahl von *Edit* können neue Sicherheitsfunktionen in das Prozessabbild eingefügt werden.

Dabei kann entweder nur die zur Sicherheitsfunktion gehörige Parameterliste oder zusätzlich ein Diagramm der Funktion eingeblendet werden. Derzeit ist das Diagramm noch statisch und zeigt nicht die aktuell eingestellten Werte.

The screenshot displays the TwinCAT configuration environment. The main window shows the 'General AX5805 Settings' tab with an 'Inputs' table. A dialog box titled 'Configure I/O element(s)' is open, showing the configuration for the '0x66E0 Axis 1 SSM' function.

Inputs Table (Background):

Name	Type	Size	Position
Axis 1 STO	BIT	0.1	0.0
Axis 1 SSM(1)	BIT	0.1	0.1
Axis 1 SSM(2)	BIT	0.1	0.2
Axis 1 SOS(1)	BIT	0.1	0.3
Axis 1 SSR(1)	BIT	0.1	0.4
Axis 1 SDIp	BIT	0.1	0.5
Axis 1 SDIn	BIT	0.1	0.6
Axis 1 Error_Ack	BIT	0.1	0.7
---	---	0.1	1.0
---	---	0.1	1.1
---	---	0.1	1.2
---	---	0.1	1.3
---	---	0.1	1.4
---	---	0.1	1.5
---	---	0.1	1.6
---	---	0.1	1.7

Configure I/O element(s) Dialog:

Function: 0x66E0 Axis 1 SSM Instance: 1

Function Diagram:

The diagram shows the SSM signal (blue line) being activated by a parameter. It also shows the speed profile (n_UL_SSM_2, n_UL_SSM_1, n_LL_SSM_1, n_LL_SSM_2) and the SSM_1 and SSM_2 signals (red lines) over time (t).

Parameter Table:

Index	Name	Value	Unit
66E2:01	n_UL_SSM 32 Bit	0x000007D0 (2000)	Increments per millisecond
66E4:01	n_LL_SSM 32 Bit	0x000003E8 (1000)	Increments per millisecond

Abb. 30: AX5000-Safety-Antriebsoptionen - Function Diagram

4.2.2.9 Externe Verbindung

Für eine Verbindung zu einer weiteren EL69x0, EJ6910, KL6904 oder zu einem Fremdgerät, kann eine Externe Verbindung *Custom FSoE Connection* angelegt werden. Existiert zu einem Fremdgerät eine eigene ESI-Datei, wird das Gerät als auswählbares Safety Gerät aufgelistet und es wird nicht die Auswahl *Custom FSoE Connection* benötigt.

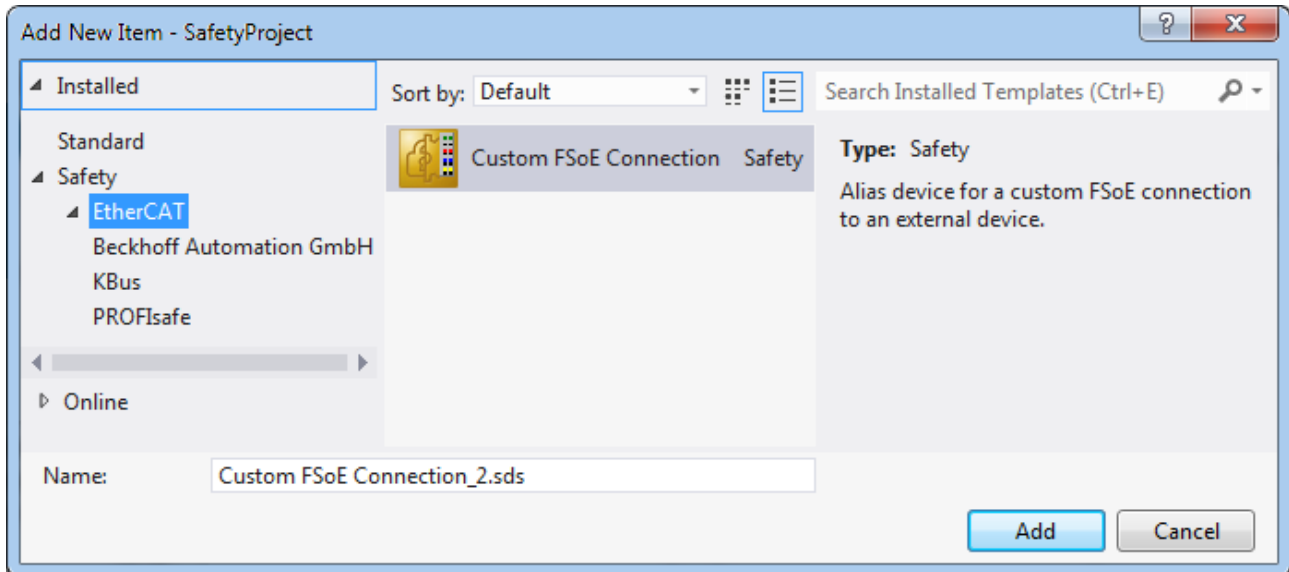


Abb. 31: Anlegen einer externen Verbindung (Custom FSoE Connection)

Bevor eine weitere Nutzung und Verlinkung der Verbindung stattfinden kann, muss die Prozessabbildgröße parametrisiert werden. Dies wird unter dem Reiter *Process Image* eingestellt. In den DropDown Listen für Input- und Output-Größe werden passende Datentypen für unterschiedliche Anzahl von Safety Daten zur Verfügung gestellt.

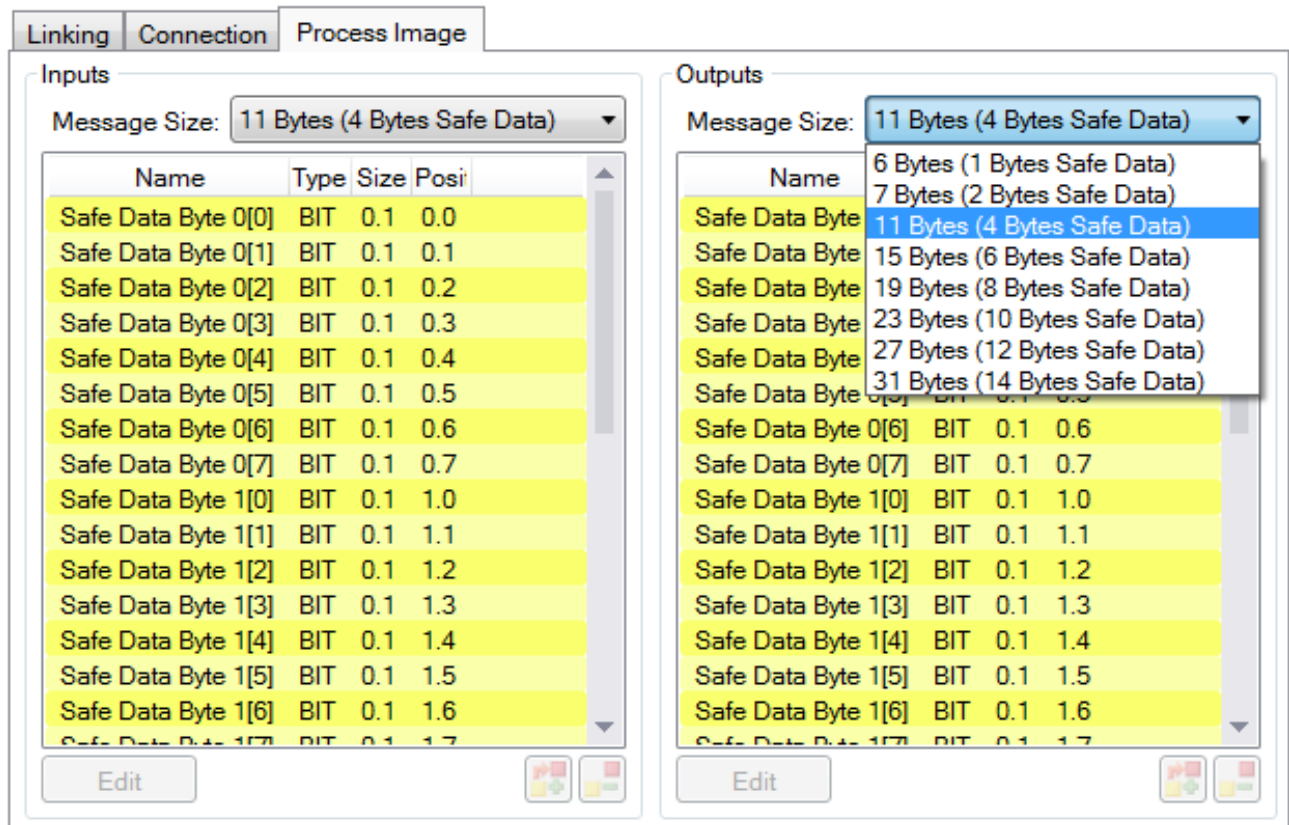


Abb. 32: Parametrierung der Prozessabbildgröße

Ist die Größe ausgewählt, können die einzelnen Signale innerhalb des Telegramms umbenannt werden, so dass bei Verwendung dieser Signale in der Logik ein entsprechender Klartext angezeigt wird. Werden die Signale nicht umbenannt, wird der Default-Name im Editor angezeigt (Safe Data Byte 0[0], ...).

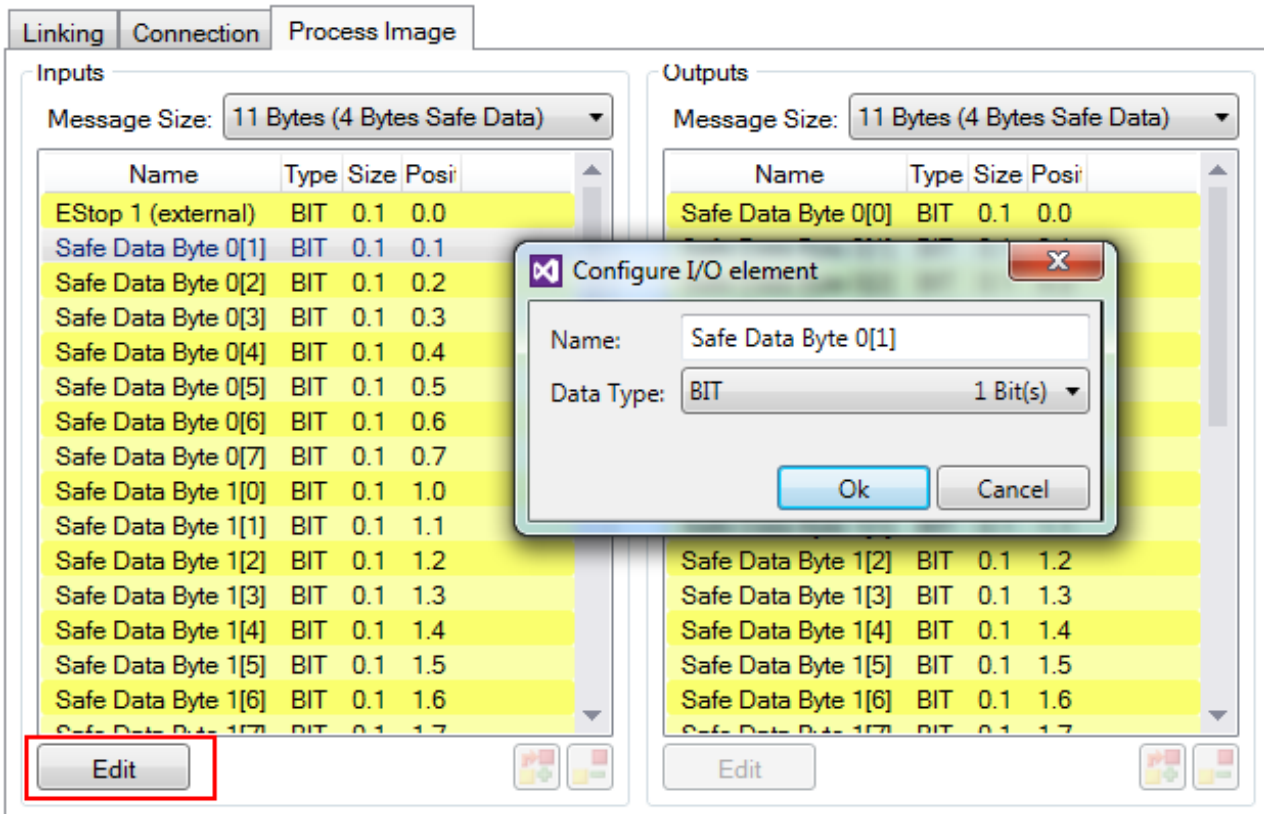



Abb. 33: Umbenennen der einzelnen Signale innerhalb des Telegramms

Die Verknüpfung der Verbindung erfolgt unter dem Reiter *Linking*. Über den Link Button  neben *Full Name (input)* und *Full Name (output)* kann die entsprechende Variable ausgewählt werden.

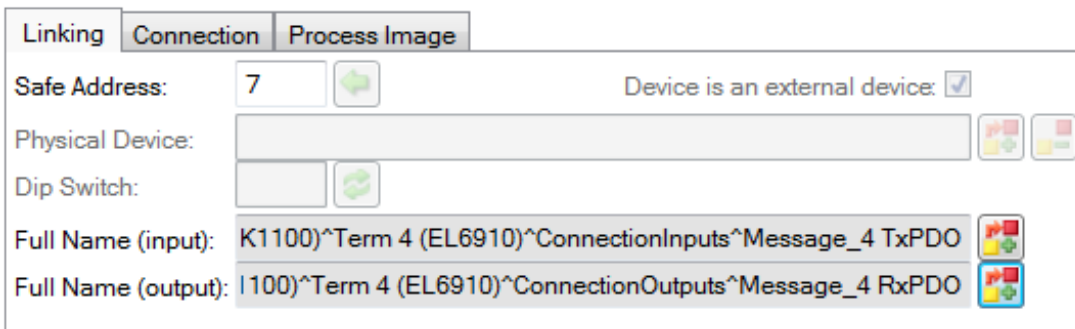


Abb. 34: Auswahl der Variablen

Dies kann z.B. eine SPS-Variable sein, die dann an das entfernte Gerät weitergeleitet wird oder kann auch direkt auf das Prozessabbild einer EtherCAT-Klemme (z.B. EL69x0 oder EL6695) verknüpft werden.

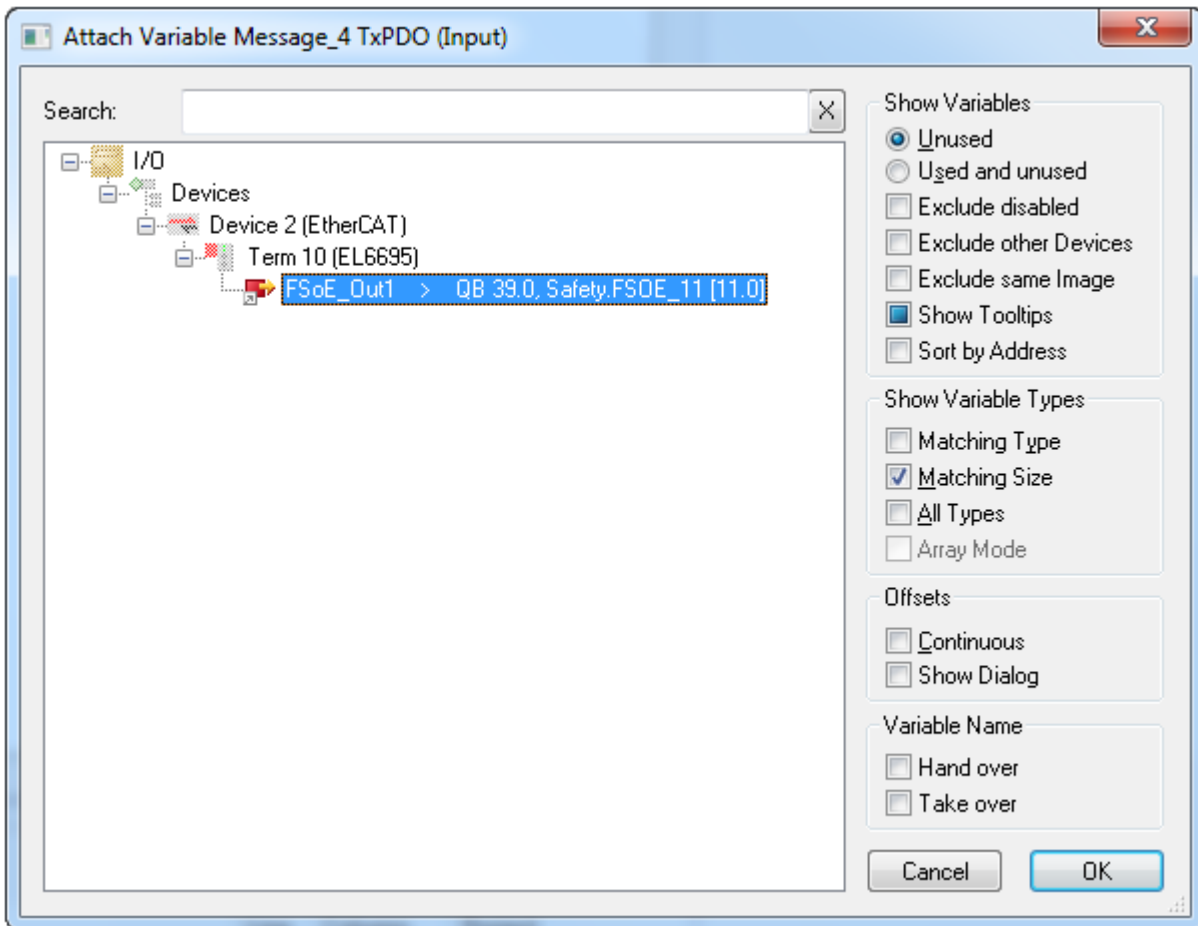


Abb. 35: Direkte Verknüpfung auf das Prozessabbild einer EtherCAT-Klemme

Weitere Informationen entnehmen Sie bitte der TwinCAT-Dokumentation zum Variablen Auswahldialog. Über den Reiter *Connection* werden die verbindungs-spezifischen Parameter eingestellt.

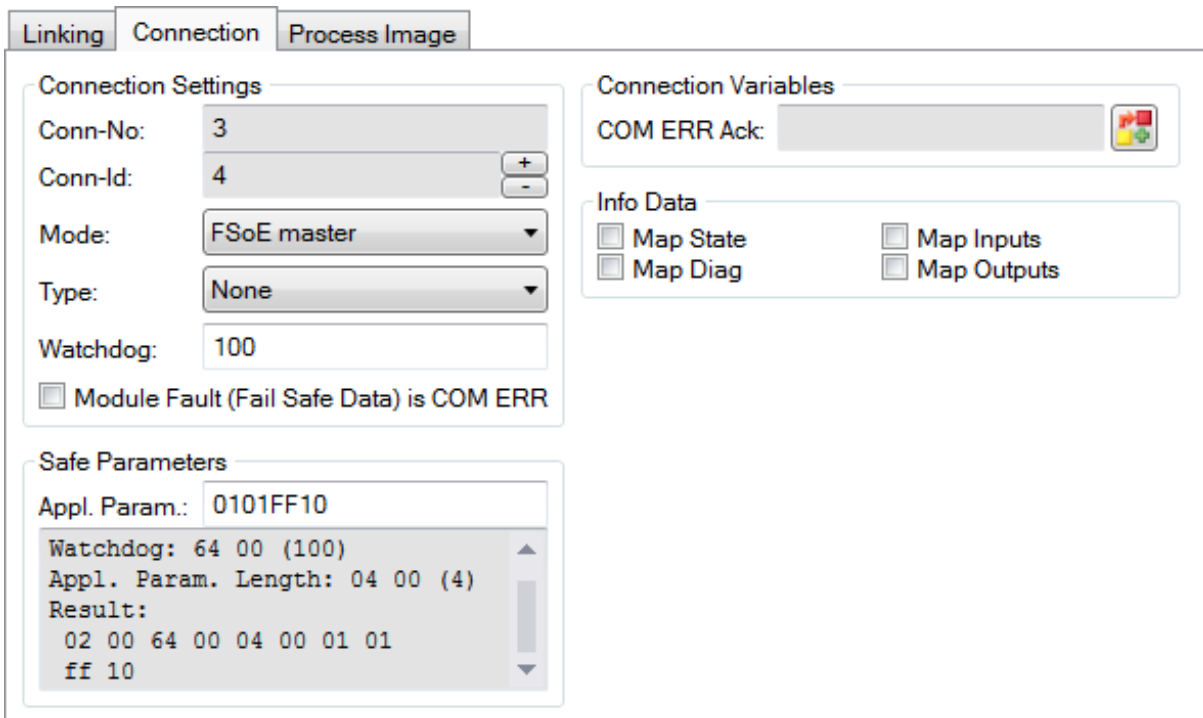


Abb. 36: Verbindungsspezifische Parameter

Detaillierte Informationen zu den einzelnen Einstellungen finden sich in der folgenden Tabelle.

Parameter	Beschreibung	Anwender-Interaktion erforderlich
Conn-No.	Verbindungsnummer: wird vom TwinCAT System automatisch vergeben	Nein
Conn-ID	Verbindungs-ID: Wird durch das System vorbelegt, kann durch den Anwender jedoch geändert werden. Innerhalb einer Konfiguration darf eine Conn-ID nur einmal vorkommen. Doppelt vergebene Verbindungs-IDs führen zu einer Fehlermeldung	Kontrolle
Mode	FSoE Master: TwinCAT Safety PLC ist FSoE-Master zu diesem Gerät. FSoE-Slave: TwinCAT Safety PLC ist FSoE-Slave zu diesem Gerät (Dies wird in der ersten Version der TwinCAT Safety PLC nicht unterstützt).	Kontrolle
Type	None: Einstellung für Fremdgeräte, für die keine ESI-Datei vorhanden ist. KL6904: Einstellung für KL6904 (Safety Parameter inaktiv) EL69XX: Einstellung für EL6900/EL6930/EL6910/EJ6910 (Safety Parameter inaktiv)	Ja
Watchdog	Watchdog-Zeit für diese Verbindung: Wird innerhalb der Watchdog-Zeit kein gültiges Telegramm von der Gerät zurück zur TwinCAT Safety PLC gesendet, wird ein ComError generiert.	Ja
Module Fault is ComError	Über diese Checkbox stellt man das Verhalten im Fehlerfall ein. Ist die Checkbox gesetzt und tritt auf dem Alias Device ein Modulfehler auf, führt dies zusätzlich zu einem Fehler der Connection und somit zu einer Abschaltung der TwinSAFE-Gruppe in der diese Verbindung definiert ist.	Ja
Safe Parameters (Appl. Param)	Geräte-spezifische Parameter: Die Länge der Parameter wird automatisch aus der eingegebenen Anzahl Zeichen berechnet. Diese Informationen liefert Ihnen typischerweise der Geräte-Hersteller.	Ja
ComErrAck	Ist der ComErrAck mit einer Variablen verlinkt, muss die Verbindung im Falle eines Kommunikationsfehlers über dieses Signal zurückgesetzt werden.	Ja
Info Data	Über diese Checkboxes können die Infodaten, die im Prozessabbild der TwinCAT Safety PLC eingeblendet werden sollen, definiert werden. Weitere Informationen finden Sie in der Dokumentation <i>TwinCAT-Funktionsbausteine für TwinSAFE-Logic-Klemmen</i> .	Ja

4.2.2.10 TwinSAFE-Gruppe - Header Files

In dem Unterordner *Header Files* der TwinSAFE-Gruppe werden alle Header-Dateien, die dieser Gruppe zugeordnet sind, aufgelistet.

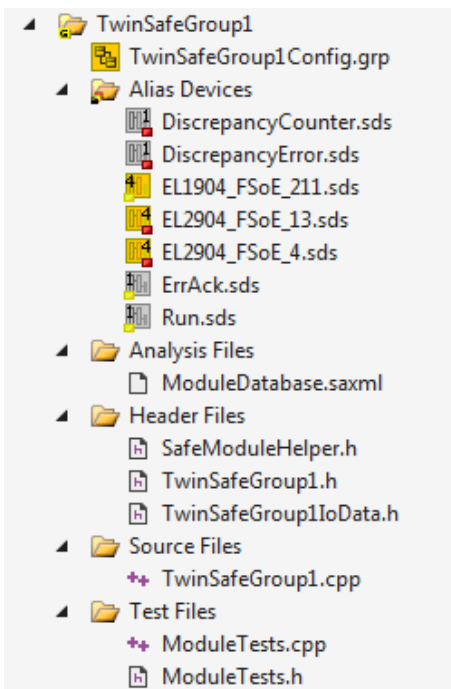


Abb. 37: TwinSAFE-Gruppe - Header Files

Die Header Dateien *SafeModuleHelper.h* und *<Gruppen-Name>IoData.h* werden automatisch durch den Safety Editor angelegt. Die Dateien sind nicht schreibgeschützt, somit könnte der Anwender die Dateien verändern, jedoch werden diese im Rahmen des Kompiliervorgangs erneut angelegt und somit alle Änderungen überschrieben.

SafeModuleHelper.h enthält vom Safety Editor angelegte Typ Definitionen, Makros und Funktionen.

<Gruppen-Name>IoData.h enthält die I/O Daten-Strukturen der Alias Devices und der TwinSAFE-Gruppen.

Die Header-Datei *<Gruppen-Name>.h* kann durch den Programmierer genutzt und erweitert werden. Hier können Typ Definitionen, Variablen und Funktionen für das Safety Applikationsmodul (*<Gruppen-Name>.cpp*) angelegt werden.

4.2.2.11 TwinSAFE-Gruppe - Source Files

Der Unterordner *Source Files* der TwinSAFE-Gruppe enthält die C++ Quell-Datei, die dieser Gruppe zugeordnet ist.

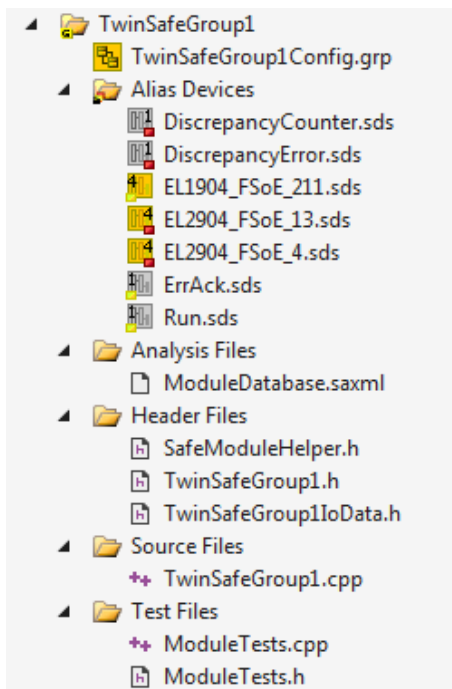


Abb. 38: TwinSAFE-Gruppe - Source Files

Die Datei <Gruppen-Name>.cpp ist in 4 Teile aufgeteilt. Davon sind 4 Modul-Funktionen fix vorgegeben. Dabei handelt es sich zum einen um die Funktionen Init, welche bei der Initialisierung der Benutzerapplikation aufgerufen wird. Zum anderen handelt es sich um die Funktionen Input-, Cycle- und OutputUpdate, welche der Einbindung der Benutzerapplikation in den zyklischen Ablauf dienen. In jedem zyklischen Ablauf wird demnach jede dieser Funktionen aufgerufen (in der Reihenfolge InputUpdate, CycleUpdate, OutputUpdate). Diese Funktionen dürfen nur durch die sichere Laufzeitumgebung aufgerufen werden.



Hinweis

Modul-Variablen

Alle Modul-Variablen (in der <TwinSAFE-Gruppenname>.h Datei definierten Variablen) müssen im Rahmen der Init-Funktion initialisiert werden.

```

/*<SafeUserApplicationCppFrontend>*/

#include "TwinSafeGroup1.h"          // Rename according to TwinSAFE group name

SAFE_MODULE_DEF(TwinSafeGroup1)    // Rename according to TwinSAFE group name
{
    //////////////////////////////////////
    /// \brief Implementation of the safe user module initialization function
    //////////////////////////////////////
    /*<TcInit>*/
    VOID CSafeModule::Init()
    {
        // Put your module initialization code here
        Counter = 0U;
        DiscrepancyError = false;
    }
    /*</TcInit>*/
}

```

Abb. 39: Init-Funktion

```

////////////////////////////////////
/// \brief Implementation of the safe user module input update function
////////////////////////////////////
/*<TcInputUpdate>*/
VOID CSafeModule::InputUpdate()
{
    // Put your module input update code here
    /*<IntentionallyEmpty/>*/
}
/*</TcInputUpdate>*/

```

Abb. 40: InputUpdate-Funktion

```

////////////////////////////////////
/// \brief Implementation of the safe user module output update function
////////////////////////////////////
/*<TcOutputUpdate>*/
VOID CSafeModule::OutputUpdate()
{
    // Put your module output update code here
    /*<IntentionallyEmpty/>*/
}
/*</TcOutputUpdate>*/

```

Abb. 41: OutputUpdate-Funktion

```

////////////////////////////////////
//! \brief Implementation of the safe user module cycle update function
////////////////////////////////////
/*<TcCycleUpdate>*/
VOID CSafeModule::CycleUpdate()
{
    // Put your cycle update code here

    // assign safe inputs to internal variable
    safeBOOL safeIn1 = sSafetyInputs.EL1904_FSoE_211.InputChannel1;
    safeBOOL safeIn2 = sSafetyInputs.EL1904_FSoE_211.InputChannel2;
    // write Counter value to standard alias device to main plc
    sStandardOutputs.DiscrepancyCounter.Out = Counter;
    sStandardOutputs.DiscrepancyError.Out = DiscrepancyError;

    // if discrepancy error occurred set outputs to false
    if (DiscrepancyError == true)
    {
        sSafetyOutputs.EL2904_FSoE_4.OutputChannel1 = false;
        sSafetyOutputs.EL2904_FSoE_13.OutputChannel14 = false;
    }
    // no discrepancy error - execute AND function
    else
    {
        if ((safeIn1 && safeIn2) == true)
        {
            sSafetyOutputs.EL2904_FSoE_4.OutputChannel1 = true;
            sSafetyOutputs.EL2904_FSoE_13.OutputChannel14 = true;
        }
        else
        {
            sSafetyOutputs.EL2904_FSoE_4.OutputChannel1 = false;
            sSafetyOutputs.EL2904_FSoE_13.OutputChannel14 = false;
        }
    }
    ...
}

```

Abb. 42: CycleUpdate-Funktion

4.2.3 CRC Verteilung

Die TwinCAT Safety PLC benötigt zum automatischen Aufstarten eine Prüfung, ob das aktuelle Projekt für die vorliegende Anlage freigeschaltet wurde. Dazu wird eine intern berechnete Prüfsumme auf durch den Anwender auswählbare TwinSAFE Komponenten verteilt und beim Starten der TwinCAT Safety PLC verifiziert. Schlägt der Vergleich fehl, startet die TwinCAT Safety PLC nicht auf. Ist der Vergleich erfolgreich, wird das Safety Projekt auf der TwinCAT Safety PLC ausgeführt.

Durch einen Doppelklick auf den Eintrag Target System öffnet sich der Dialog Target System. Hier kann neben dem Target System auch die CRC Distribution konfiguriert werden.

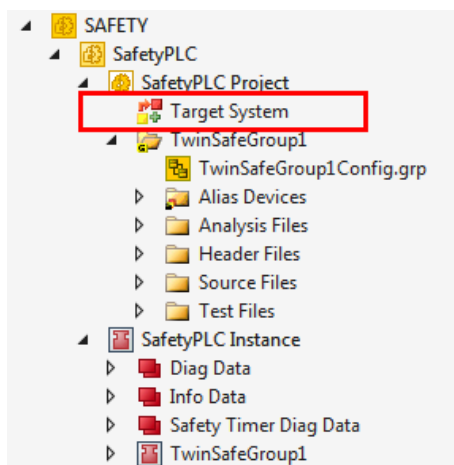


Abb. 43: Target System

In dem Dialog CRC Distribution werden alle sicheren Alias Devices aufgelistet, die für die CRC Verteilung verwendet werden können. Durch die Checkbox neben jedem Eintrag kann ausgewählt werden, ob die CRC auf der Komponente gespeichert werden soll. Zusätzlich kann angegeben werden, wie viele der ausgewählten Komponenten mindestens die korrekte CRC zurückliefern müssen, damit die TwinCAT Safety PLC aufstartet. Es muss an dieser Stelle mindestens eine Komponente ausgewählt werden, damit ein Safety Projekt für die TwinCAT Safety PLC erfolgreich runtergeladen und freigeschaltet werden kann.

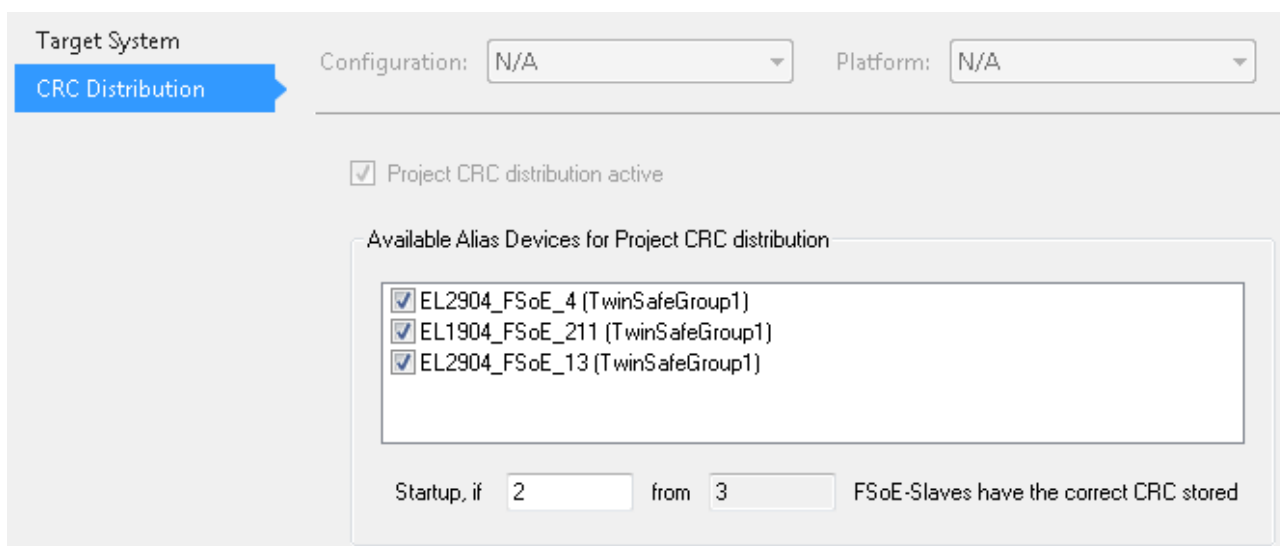



Abb. 44: Dialog CRC Distribution

4.2.4 Download der Safety Applikation

Der Download der Safety Konfiguration erfolgt 2-stufig - Laden (Download) und Freischaltung (Unlock). Für einen Download muss der Benutzer zunächst eine Verbindung zum gewünschten Zielsystem aufbauen. Dies erfolgt über die in TwinCAT 3 Standard-Mechanismen zum Verbinden zu einer nicht-lokalen Laufzeitumgebung (inklusive entsprechend geregelter Benutzerauthentifizierung). Der Download der Safety Applikation auf die Laufzeitumgebung kann im Anschluss durchgeführt werden (dabei wird der Zustand

Konfig der Laufzeitumgebung gefordert). Dies kann über den Button Download  aus der Safety Toolbar,

oder über das Menü  Download Safety Project erfolgen. Für den Download der Safety Applikation müssen keinerlei Eingaben durch den Anwender gemacht werden. Es wird im Download Dialog die Projekt CRC des Safety Projektes angezeigt, welches runtergeladen werden soll. Mit *Finish* kann diese Projekt CRC bestätigt und der eigentliche Download angestoßen werden.

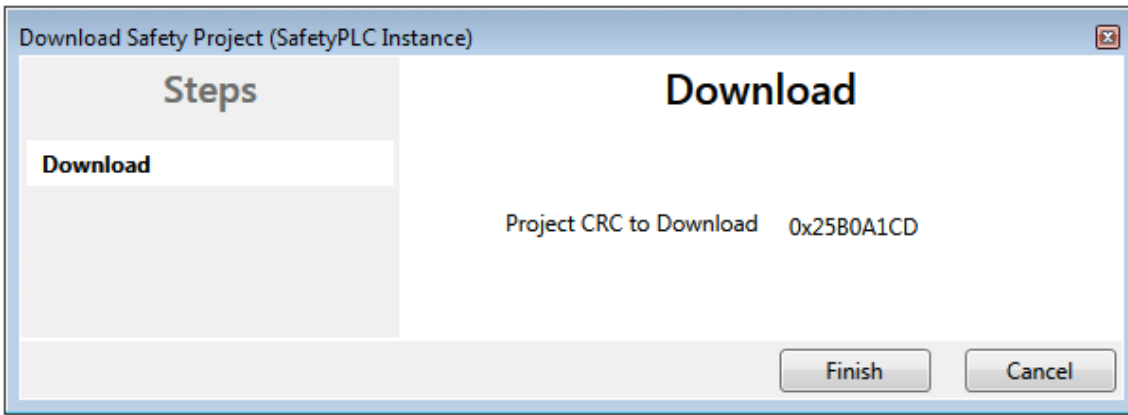


Abb. 45: Download der Safety Applikation

 VORSICHT	<p>Nur qualifizierte Tools zu benutzen</p> <p>Zum Laden und Freischalten des Projektes auf der TwinCAT Safety PLC ist ausschließlich ein qualifiziertes Tool (TwinCAT 3.1) zu benutzen!</p>
 GEFAHR	<p>Quelltext der Safety Applikation</p> <p>Der Benutzerquelltext ist entsprechend der jeweils anzuwendenden Normen zu entwickeln mit der Grundnorm IEC 61508:2010. Beachten Sie hierzu auch das Kapitel Verifikation und Validierung [► 81].</p>
 VORSICHT	<p>Identifikation des Zielsystems</p> <p>Vor dem Download und der Freischaltung des Safety Projektes muss der Anwender sicherstellen, dass es sich bei dem verbundenen Zielsystem um das gewünschte Zielsystem handelt. Ein Fehler bei diesem Vorgang kann erst bei der Verifikation und Validierung der Sicherheitsapplikation erkannt werden.</p>

4.2.5 Freischaltung der Safety Applikation

Nach einem erfolgreichen Download der Safety Applikation muss diese noch freigeschaltet werden, damit eine Ausführung möglich ist.

Hierzu muss die aktuelle Konfiguration zunächst aktiviert werden und TwinCAT im Run-Modus gestartet werden. Bei einem nicht freigeschalteten Safety Projekt erscheint im Ausgabefenster von TwinCAT 3 eine Meldung, dass das zu startende Safety Projekt auf eine Freischaltung wartet. Sobald die Konfiguration aktiv

ist, kann die Freischaltung über den Button Unlock  oder über das Menu gestartet werden.

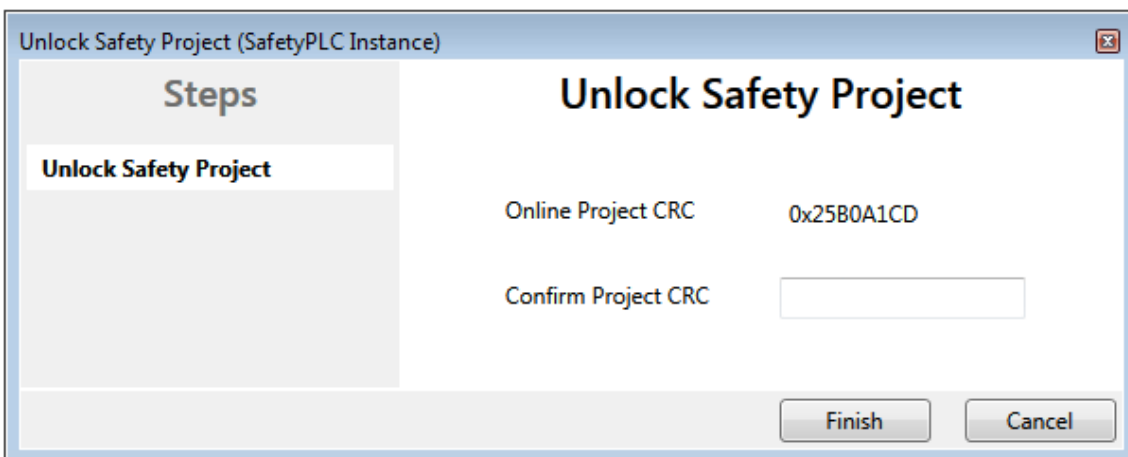


Abb. 46: Unlock Safety Project

Der Anwender muss nun die angezeigte Online CRC durch Eingabe bestätigen.

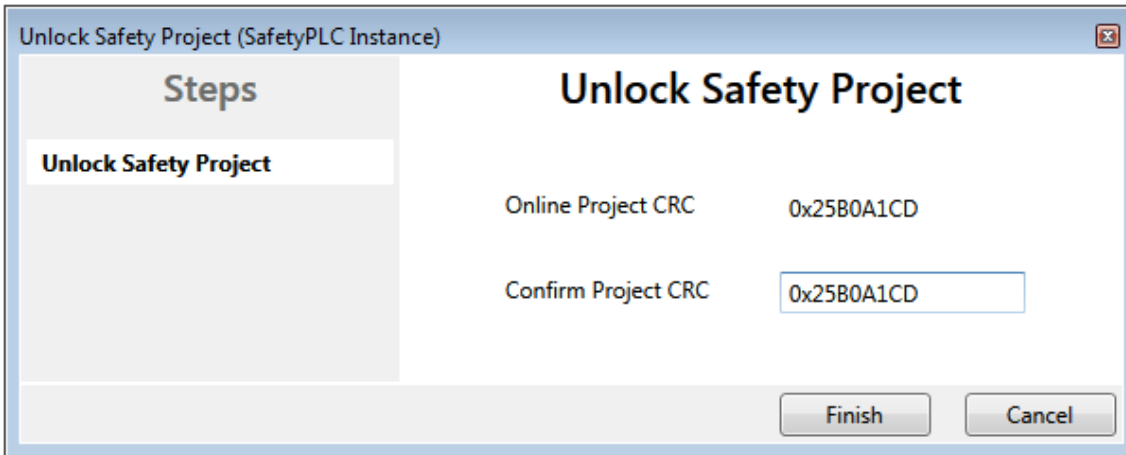


Abb. 47: Unlock Safety Project

Durch Bestätigen der CRC mit dem *Finish* Button wird die Safety Applikation freigeschaltet und ausgeführt. Im Zuge des Hochfahrens der Safety Applikation wird die CRC an die im Rahmen der CRC Distribution konfigurierten sicheren Kommunikationsteilnehmern verteilt. Bei einem erneuten Restart des TwinCAT Systems wird die Safety Applikation nun ohne erneute Freischaltung gestartet.

Nach der Freischaltung zeigt die TwinSAFE CRC Toolbar die identische Online und Offline CRC an.

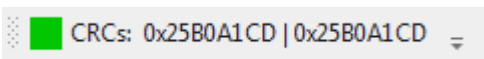


Abb. 48: Identische CRCs

4.2.6 Safety und CRC Toolbar

Über einen Rechtsklick auf den Toolbar Bereich von TwinCAT 3.1 können die Safety Toolbar und die Safety CRC Toolbar eingeschaltet werden.



Abb. 49: Aktivierung Safety und CRC Toolbar

Safety Toolbar	Beschreibung
	Prüfen der Safety Applikation
	Prüfen der Safety Applikation inklusive Hardware-Level
	Download der Safety Applikation auf die TwinCAT Safety PLC
	Löschen der Safety Applikation auf der TwinCAT Safety PLC
	Freischalten der Safety Applikation
	derzeit nicht verwendet

Die CRC Toolbar zeigt die Online und Offline CRC an. Zusätzlich wird durch ein Icon angezeigt, ob diese gleich oder ungleich sind.

CRC Toolbar	Beschreibung
	Grünes Icon: CRCs sind identisch
	Rotes Icon: CRCs sind unterschiedlich
	Online CRC (32-Bit)
	Offline CRC (32-Bit)

4.2.7 Info-Daten

Infodaten des Target System



Abb. 50: Target System - Map Object ID und Map Project CRC

Auf dem Target System kann über die Checkboxes *Map Object Id* und *Map Project CRC* festgelegt werden, dass die Object ID und die Projekt CRC in das Prozessabbild der TwinCAT Safety PLC kopiert werden. Von hier können die Einträge *Object Id* und *Project CRC* in die Standard-SPS verlinkt werden.

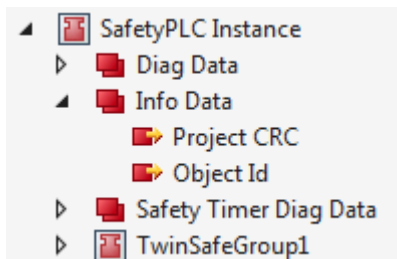


Abb. 51: Target System Info Data

Info Data	Beschreibung
Project CRC	32-bit Projekt CRC (Online)
Object Id	Eindeutige Id der TwinCAT Safety PLC Instanz (über das gesamte TwinCAT 3 Projekt eindeutig)

Diag Data - TwinCAT Safety PLC

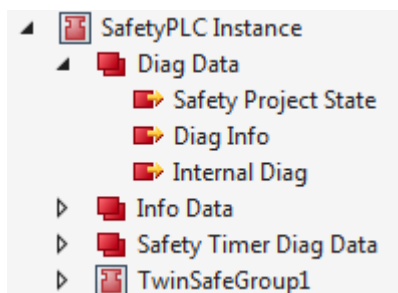


Abb. 52: Diag Data - TwinCAT Safety PLC

Diag Data	Beschreibung
Safety Project State	<ul style="list-style-type: none"> • 601 (0x0259) Init Der Zustand Init wird beim Aufstarten der Instanz eingenommen, um die interne Konfiguration zu testen, bevor letztlich wirklich aufgestartet werden kann. • 602 (0x025A) Run Der Zustand Run wird eingenommen, wenn im Rahmen des Aufstartens kein Fehler aufgetreten ist. Im Zustand Run werden die konfigurierten TwinSAFE Gruppen in den zyklischen Ablauf eingebunden. • 603 (0x025B) Error Der Zustand Error wird eingenommen, wenn ein interner Fehler auftritt. Dieser Zustand kann nur durch ein erneutes Starten der gesamten Konfiguration verlassen werden. Im Zustand Error werden die TwinSAFE Gruppen (und somit auch die untergeordneten Kommunikationsverbindungen und die untergeordnete Benutzerapplikation) nicht mehr abgearbeitet und somit der sichere Zustand eingenommen. • 604 (0x025C) Checking Download Completion In diesem Zustand wird geprüft, ob es sich beim zu startenden Safety Projekt um ein gültig heruntergeladenes Projekt handelt. Ein Fehler bei diesem Vorgangs führt zum Zustand Error. • 605 (0x025D) Checking Unlocking Data In diesem Zustand wird geprüft, ob es sich beim zu startenden Safety Projekt um ein im Vorfeld erfolgreich freigeschaltetes Projekt handelt. Ist die Prüfung nicht erfolgreich, wird der Zustand 606 eingenommen. • 606 (0x025E) Waiting for Activation Der Zustand wird eingenommen, wenn das Safety Projekt im Vorfeld noch nicht freigeschaltet wurde. Der Zustand kann nur verlassen werden, wenn aus dem entsprechenden TwinCAT 3-Projekt die Freischaltung des Safety Projektes manuell angestoßen wird. • 607 (0x025F) Writing Unlocking Data Nach erfolgreicher Freischaltung des Safety Projektes werden entsprechende Daten geschrieben, um für ein erneutes Hochstarten diese erfolgreiche Freischaltung zu signalisieren. • 608 (0x0260) Waiting for Project CRC Acknowledge Wird ein Safety Projekt gestartet, welches bereits freigeschaltet wurde, werden die Freischaltungsdaten mit den Daten verifiziert, welche von den zuvor im Rahmen der CRC Distribution konfigurierten sicheren Kommunikationsteilnehmern abgefragt werden.
Diag Info	Diagnoseinformation der Instanz
Internal Diag	Interne Diagnoseinformation (für Benutzer nicht von Relevanz)

Safety Timer Diag Data - TwinCAT Safety PLC

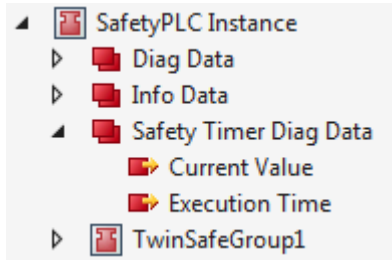


Abb. 53: Safety Timer Diag Data - TwinCAT Safety PLC

Safety Timer Diag Data	Beschreibung
Current Value	Aktueller Wert des sicheren Zeitsignals
Execution Time	Intern bestimmte Ausführungszeit von Anfang InputUpdate bis zu Ende OutputUpdate der gesamten Instanz

Gruppen Infodaten

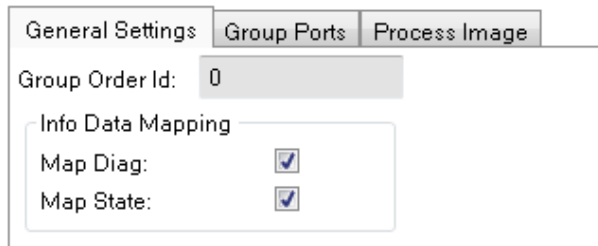


Abb. 54: Gruppen MapDiag MapState

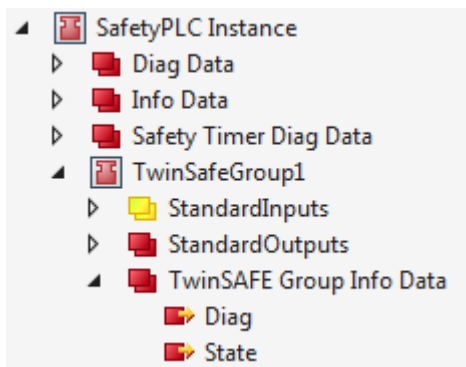


Abb. 55: Gruppen Infodaten

Group Info Data	Beschreibung
State	<ul style="list-style-type: none"> • 701 (0x02BD) Stop Der Zustand Stop wird beim Aufstarten der Instanz eingenommen. Im Betrieb wird dieser Zustand eingenommen, wenn das entsprechende Run/Stop-Signal konfiguriert ist und nicht auf True gesetzt ist. • 702 (0x02BE) Run Der Zustand Run wird eingenommen, wenn keine der beteiligten sicheren Verbindungen fehlerhaft ist und – sofern entsprechend konfiguriert – der Run/ Stop-Eingang auf True gesetzt ist. • 703 (0x02BF) Safe Der Zustand Safe wird eingenommen, wenn mindestens eine der Verbindungen nicht im Zustand Data ist. • 704 (0x02C0) Error Der Zustand Error wird eingenommen, wenn ein Applikationsfehler oder ein Kommunikationsfehler auftritt. • 705 (0x02C1) Reset Der Zustand Reset wird eingenommen, wenn im Zustand Error das ErrorAck-Signal eine steigende Flanke zeigt. • 706 (0x02C2) Global Error Der Zustand Global Error wird eingenommen, wenn in der internen Verarbeitung ein schwerwiegender Fehler auftritt. Dieser Zustand kann nur durch ein erneutes Starten der aktuellen Konfiguration wieder verlassen werden.
Diag	<ul style="list-style-type: none"> • Diagnoseinformation

Connection Infodaten

Abb. 56: FSoE Connection Map Info Data

Abb. 57: Infodaten FSoE Connection

Connection Info Data	Beschreibung
State	<ul style="list-style-type: none"> • 100 (0x64) Reset Der Zustand Reset dient dazu, nach dem Power-On oder einem Safety over EtherCAT Kommunikationsfehler die Safety over EtherCAT Connection neu zu initialisieren. • 101 (0x65) Session Beim Übergang in den bzw. im Zustand Session wird eine Session ID vom Safety over EtherCAT Master zum Safety over EtherCAT Slave übertragen, der wiederum mit einer eigenen Session ID antwortet. • 102 (0x66) Connection Im Zustand Connection wird eine Connection ID vom Safety over EtherCAT Master zum Safety over EtherCAT Slave übertragen. • 103 (0x67) Parameter Im Zustand Parameter werden sichere Kommunikations- und gerätespezifische Anwendungsparameter übertragen. • 104 (0x68) Data Im Zustand Data werden solange Safety over EtherCAT Cycles übertragen, bis entweder ein Kommunikationsfehler auftritt oder ein Safety over EtherCAT Node lokal gestoppt wird.
Diag	<ul style="list-style-type: none"> • xxxx 0001 - Ungültiges Kommando • xxxx 0010 - Unbekanntes Kommando • xxxx 0011 - Ungültige Connection ID • xxxx 0100 - Ungültige CRC • xxxx 0101 - Watchdog abgelaufen • xxxx 0110 - Ungültige FSoE Adresse • xxxx 0111 - Ungültige Daten • xxxx 1000 - Ungültige Kommunikationsparameterlänge • xxxx 1001 - Ungültige Kommunikationsparameter • xxxx 1010 - Ungültige Anwenderparameterlänge • xxxx 1011 - Ungültige Anwenderparameter • xxxx 1100 - FSoE Master Reset • xxxx 1101 - Modulfehler auf Slave erkannt, bei aktivierter Option "Modulfehler ist ComError" • xxxx 1110 - Modulfehler auf EL290x erkannt, bei aktivierter Option "Error acknowledge active" • xxxx 1111 - Slave noch nicht gestartet, oder unerwartetes Fehlerargument • xxx1 xxxx - Fehler beim FSoE Slave erkannt • xx1x xxxx - FSoE Slave meldet Failsafe Value aktiv • x1xx xxxx - StartUp • 1xxx xxxx - FSoE Master meldet Failsafe Value aktiv
Inputs	sicheren Eingänge der Connection
Outputs	sicheren Ausgänge der Connection

4.2.8 Task-Einstellungen

Durch einen Rechtsklick auf *Tasks* und Auswahl von *Add New Item...* kann eine neue Task angelegt werden.

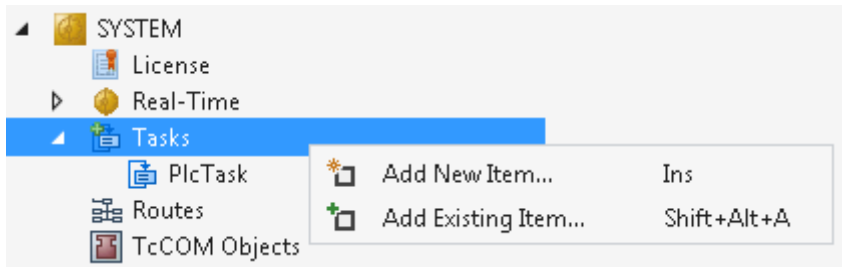


Abb. 58: Hinzufügen einer neuen Task

Im Insert Task Dialog kann der Taskname eingetragen werden und ausgewählt werden, ob die Task mit oder ohne Image angelegt werden soll. Für die TwinCAT Safety PLC kann beides ausgewählt werden, hier ist mit Image ausgewählt.

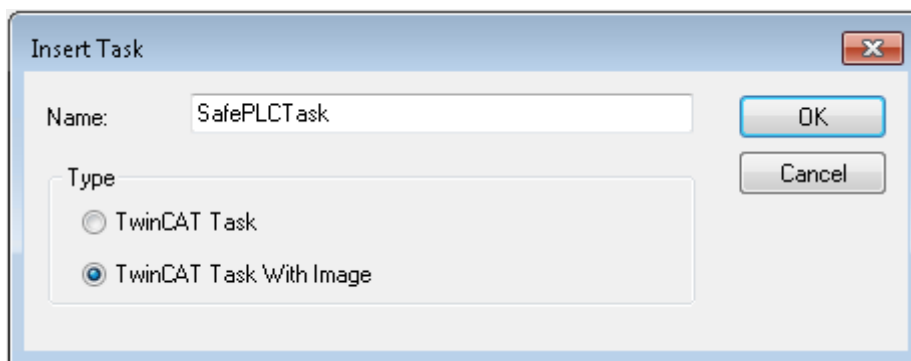



Abb. 59: Dialog Insert Task

Einstellungen

Durch einen Doppelklick auf die Task öffnen sich die Task-Einstellungen. Hier kann die Zykluszeit und auch die Priorität eingestellt werden.

 Hinweis	<p>Zykluszeit und Priorität</p> <p>Die Zykluszeit kann frei gewählt werden, sollte jedoch so gewählt werden, dass keine Überschreitungen (ExceedCounter inkrementiert nicht) auftreten.</p> <p>Die Priorität sollte möglichst hoch (niedrige Nummer) eingestellt werden, um Unterbrechungen und Jitter der TwinCAT Safety PLC möglichst gering zu halten.</p>
---	--

Task **Online** Parameter (Online)

Name: Port:

Auto start Object Id:

Auto Priority Management

Priority:

Cycle ticks: ms

Start tick (modulo):

Separate input update

Pre ticks:

Warning by exceeding

Message box

Watchdog Cycles:

Options

Disable

Create symbols

Include external symbols

...

Extern sync

Floating point exceptions

Watchdog stack

Comment:

Abb. 60: Einstellungen Task

ExceedCounter überprüfen

Über den Reiter Online der verwendeten Task können die Ausführungszeit und der Überschreitungszähler überprüft werden.

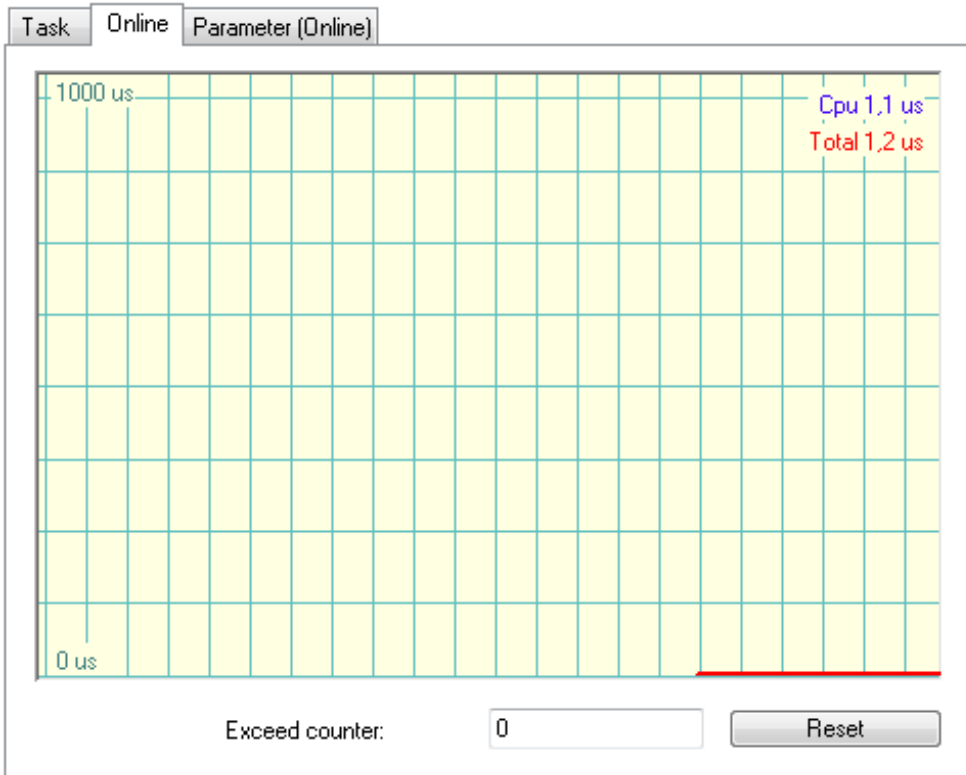




Abb. 61: Task Ausführungszeit und Überschreitungszähler

5 Anwendungsentwicklung in Safety C

 GEFAHR	Anwendungsentwicklung in Safety C Der Benutzer-Quelltext ist entsprechend der jeweils anzuwendenden Normen zu entwickeln, mit der Grundnorm IEC 61508:2010.
 GEFAHR	Warnungen des erweiterten Bauprozesses der sicherheitsgerichteten Applikation Alle im Rahmen des Bauprozesses auftretenden Warnungen müssen durch den Anwender bewertet werden und beseitigt oder ggf. kommentiert oder dokumentiert werden.

5.1 Programmierung in Safety C

5.1.1 Abgrenzung der Programmierung in Safety C zu C/C++

Safety C ist eine C++ basierte Hochsprache zur Programmierung von Sicherheitsanwendungen für die TwinCAT Safety PLC mit einem sicheren Sprachumfang, welcher einer streng typisierten und modularen Variante der Sprache C ohne dynamischen Speicher und Pointer(-Arithmetik) entspricht. Daher entspricht die Syntax und Semantik von Safety C grundsätzlich der entsprechenden, zulässigen C++ Sprachteilmenge (siehe dazu z.B. ISO-Standard C++11 N3242) wie sie durch den verwendeten C++ Compiler verarbeitet wird. Der Safety C Übersetzungsprozess der TwinCAT Safety PLC setzt dafür den Microsoft Visual C++ Compiler 2015 oder höher voraus.

C++ ist eine im Wesentlichen aufwärtskompatible Erweiterung von C mit Sprachunterstützung für Objektorientierung und Metaprogrammierung, sodass C Programme mit wenigen Einschränkungen auch durch C++ Compiler verarbeitet werden können. Die Programmierung in Safety C erlaubt daher auch die eingeschränkte Nutzung objektorientierter Erweiterungen von C++ zur Datenkapselung und Modularisierung von Programmmodulen durch den Anwendungsentwickler, verzichtet aber weitestgehend auf die typischen C++ Konzepte wie Vererbung, Polymorphie und generische Template-Programmierung. Daher ist aus Sicht des Anwendungsentwicklers die Programmierung in Safety C näher an der prozeduralen Entwicklung in C anzusiedeln.

Hochsprachen, wie etwa C/C++, erlauben dem Programmierer gegenüber funktionsblockbasierter Programmierung wesentlich mehr Freiheitsgrade, wodurch aber auch potentielle Quellen für systematische Anwendungsfehler entstehen. Zudem lassen die ISO-Standards für C/C++ bewusst Spielraum für die Implementierung effizienter Compiler, sodass ein standardkonformes C/C++ Programm undefiniertes oder plattformabhängiges Verhalten beinhalten kann (z.B. Datentypbreiten, Division durch Null oder Über-/Unterlauf vorzeichenbehalteter Ganzzahldatentypen).

Die anzuwendenden Sicherheitsnormen für speicherprogrammierbare Systeme fordern daher für die Software-Entwicklung von Sicherheitsfunktionen in Hochsprachen (siehe z.B. IEC 61508-3:2010) eine Einschränkung des vollen Umfangs von C/C++ durch die Einhaltung von Sprachteilmenge mit Codierregeln, um insbesondere Programme mit nicht eindeutiger Semantik zu vermeiden, aber auch das Risiko der Erzeugung von fehlerhaftem Programmcode oder Programmen mit unerwartetem Verhalten zu vermindern. Weiterhin sollen dadurch die werkzeuggestützte Programmanalyse und -verifikation, sowie manuelle Analysen durch Code-Inspektionen erleichtert werden.

Der zulässige Sprachumfang von Safety C vermeidet weitestgehend die Erzeugung von Quellcode mit undefiniertem Verhalten. An einigen Stellen werden alternative Hilfsfunktionen mit eindeutiger Semantik oder integrierter Erkennung von undefiniertem Verhalten angeboten, so dass der Anwendungsentwickler die Möglichkeit hat, zwischen nativen Operationen und Hilfsfunktionen zu wählen.

Die wesentlichen Einschränkungen von Safety C gegenüber C und C++ lassen sich wie folgt zusammenfassen:

- Keine Unterstützung (bzw. mit wenigen Ausnahmen) von Objektorientierung, Metaprogrammierung oder sonstige für C++ typische Spracherweiterungen gegenüber C

- Eingeschränkte Datentypen, sowie strenge Typisierung aller Datentypen zur Vermeidung impliziter Typumwandlungseffekte
- Einschränkung von einfachen Anweisungen, Operatoren und Ausdrücken zur Vermeidung unerwarteter Ergebnisse und Seiteneffekte
- Einschränkung von Kontrollflussanweisungen (*if-else*, *for*, *while*, und *switch-case*) für verständliche und analysierbare Programmabläufe
- Keine direkte oder indirekte Rekursion
- Keine dynamischen Datenstrukturen, sowie keine Pointer- bzw. adressbasierte Datenzugriffe oder Funktionsaufrufe
- Keine globalen oder statischen Funktionen und Variablen
- Vorgabe strukturierter Quellcode-Vorlagen als Basis für die Entwicklung von sicheren Programmmodulen (Applikation, Funktionsblock)
- Keine benutzerdefinierten Präprozessor- oder Compiler-Anweisungen zur bedingten Quellcode-Übersetzung oder Einbindung weiterer Quellcode-Dateien
- Keine Einbindung von Standard-Bibliotheken oder Legacy Code

5.1.2 Quellcode-Vorlagen

Mit dem Anlegen einer TwinSAFE Gruppe werden auch Header- und Quellcode-Dateien für das sichere Anwenderprogramm angelegt. Von besonderer Relevanz für den Programmierer sind dabei die Dateien <Gruppenname>.h und <Gruppenname>.cpp. Weiterhin wird dem Anwender die Möglichkeit gegeben auch benutzerdefinierte Funktionsblöcke zu definieren (in V1 noch nicht unterstützt).

Die Templates sind so angelegt, dass Änderungen und Erweiterungen nur innerhalb gekennzeichneteter Bereiche durch den Programmierer erlaubt sind. Das Anlegen weiterer *.h / *.cpp - Dateien innerhalb der TwinSAFE Gruppe ist nicht zulässig.

In der Modul-Header Datei (<Gruppenname>.h-Datei) können Präprozessor-Defines, Typdefinitionen (Structs, Enums) und Modul-Variablen angelegt werden (Enums werden in V1 noch nicht unterstützt). Es ist eine reine Deklaration der Modul-Klasse mit benutzerdefinierten Modul-Variablen und -Funktionen, es ist keine Implementierung erlaubt.

Die Implementierung der Modul-Klasse erfolgt in der <Gruppenname>.cpp-Datei.

Nach dem Anlegen einer Gruppe werden Änderungen an <Gruppenname>.h und <Gruppenname>.cpp nur durch den Anwender vorgenommen und unmittelbar nach dem Speichern durch Prüfsummen abgesichert (und somit auch für den Anwender indirekt als Änderung der Projekt-CRC sichtbar). Dadurch ergibt sich die Notwendigkeit, dass bei Umbenennung des Gruppennamens manuelle Anpassungen am Quellcode von <Gruppenname>.h und <Gruppenname>.cpp vorgenommen werden müssen. Das betrifft die zum Erstellungszeitpunkt generierten namensspezifischen Quellcode-Anteile (siehe dazu Kommentare in der Quellcode-Vorlage).

Die weiteren angelegten Header-Dateien werden durch den Safety Editor dynamisch angelegt und dürfen durch den Anwender nicht geändert werden. Alle Änderungen an diesen Dateien durch den Anwender werden im Rahmen des Übersetzungsprozesses überschrieben!

5.1.2.1 Anwendungsmodul für eine TwinSAFE Gruppe

Quellcode-Vorlage für die Modul-Deklaration eines Safety C Anwendungsmoduls <TwinSAFE-Gruppenname>.h

```

////////////////////////////////////
/// \file TwinSafeGroup1.h
/// \brief Header file of the TwinSafeGroup1 application module
/// \ingroup TwinSafeGroup1
/// \defgroup TwinSafeGroup1
/// \brief Put brief description of your application module here
/// \authors Administrator
/// \copyright Put affiliation and copyright notice here
/// \version V1.0

```

```

!!! \date 2016-09-29
!!! \ingroup Empty
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//\internal////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
!!! XML tags <...> enclosed by C style block comment markups are protected for
!!! structural and semantic code analysis. Do NOT remove or reorder any of the
!!! mandatory markups within the source code template as safe build process may
!!! fail otherwise! For further information on how to write compliant Safety C
!!! user code please refer to the provided Safety C coding guidelines document!
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/*<SafeUserApplicationHFrontend>*/
#pragma once

/*<UserDefinedIncludes>*/          // Include other safe module headers here
/*</UserDefinedIncludes>*/

#include "TwinSafeGroup1IoData.h" // Rename according to TwinSAFE group name

/*<UserDefinedDefines>*/          // Define preprocessor constants here
/*</UserDefinedDefines>*/

NAMESPACE(TwinSafeGroup1)        // Rename according to TwinSAFE group name
{
    /*<UserDefinedTypes>*/          // Define custom data types here
    /*</UserDefinedTypes>*/

    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    ///! \class TwinSafeGroup1
    ///! \brief Declaration of the Safety C user application module class
    ///! \details Put detailed description of your module functionality here
    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    SAFE_MODULE(TwinSafeGroup1) // Rename according to TwinSAFE group name
    {
        // Public module interface
        PUBLIC:
            VOID Init();                //!< Module initialization function
            VOID InputUpdate();         //!< Module input update function
            VOID OutputUpdate();        //!< Module output update function
            VOID CycleUpdate();         //!< Module cycle update function

            SafetyInputs sSafetyInputs; //!< Safe input data struct
            SafetyOutputs sSafetyOutputs; //!< Safe output data struct
            StandardInputs sStandardInputs; //!< Non-safe input data struct
            StandardOutputs sStandardOutputs; //!< Non-safe output data struct

            safeUINT16 ul6SafeTimer     //!< Safe external timer input (in ms)

            TSGData sTSGData;          //!< TwinSAFE group exchange data struct

        // Module internals
        PRIVATE:

            /*<UserDefinedVariables>*/ // Define internal variables here
            /*</UserDefinedVariables>*/

            /*<UserDefinedFunctions>*/ // Define internal functions here
            /*</UserDefinedFunctions>*/

            SAFE_MODULE_EXPORT();
        };

        ///! Reference to project FCS symbol
        extern UINT32 SAFETY_PROJECT_FCS; // Do NOT read, write or remove!
    };
}
/*</SafeUserApplicationHFrontend>*/

```



Hinweis

Zulässige Modifikationen <TwinSAFE-Gruppenname>.h

- NAMESPACE <TwinSAFE Gruppenname> (bei Änderung des TwinSAFE Gruppennamens im Projektbaum muss dieser Eintrag durch den Anwender ebenfalls angepasst werden)
- SAFE_MODULE(<TwinSAFE Gruppenname >) (bei Änderung des TwinSAFE Gruppennamens im Projektbaum muss dieser Eintrag durch den Anwender ebenfalls angepasst werden)
- Anwender Includes nur zwischen
/*<UserDefinedInclude>*/ ... /*</UserDefinedInclude>*/
- Anwender Defines nur zwischen
/*<UserDefinedDefines>*/ ... /*</UserDefinedDefines>*/
- Anwender Typdefinitionen nur zwischen
/*<UserDefinedTypes>*/ ... /*</UserDefinedTypes>*/
- Anwender Variablen nur zwischen
/*<UserDefinedVariables>*/ ... /*</UserDefinedVariables>*/
- Anwender Funktionen nur zwischen
/*<UserDefinedFunctions>*/ ... /*</UserDefinedFunctions>*/
- Kommentare können beliebig ergänzt/verändert werden (außer geschützte Kommentare der Form /*<...>*/)

Quellcode-Vorlage für die Modul-Implementierung eines Safety C Anwendungsmoduls: <TwinSAFE-Gruppenname>.cpp

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///! \file TwinSafeGroup1.cpp
///! \brief Source file of the TwinSafeGroup1 application module
///! \ingroup TwinSafeGroup1
///! \authors Administrator
///! \copyright Put affiliation and copyright notice here
///! \version V1.0
///! \date 2016-09-29
///! \details Put detailed description of your module implementation here
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///!\internal////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///! XML tags <...> enclosed by C style block comment markups are protected for
///! structural and semantic code analysis. Do NOT remove or reorder any of the
///! mandatory markups within the source code template as safe build process may
///! fail otherwise! For further information on how to write compliant Safety C
///! user code please refer to the provided Safety C coding guidelines document!
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/*<SafeUserApplicationCppFrontend>*/

#include "TwinSafeGroup1.h" // Rename according to TwinSAFE group name
SAFE_MODULE_DEF(TwinSafeGroup1) // Rename according to TwinSAFE group name
{
    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    ///! \brief Implementation of the safe user module initialization function
    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    /*<TcInit>*/
    VOID CSafeModule::Init()
    {
        // Put your module initialization code here
    }
    /*</TcInit>*/

    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    ///! \brief Implementation of the safe user module input update function
    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    /*<TcInputUpdate>*/
    VOID CSafeModule::InputUpdate()
    {
        // Put your module input update code here
    }
    /*</TcInputUpdate>*/

    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    ///! \brief Implementation of the safe user module output update function

```

```

////////////////////////////////////
/*<TcOutputUpdate>*/
VOID CSafeModule::OutputUpdate()
{
    // Put your module output update code here
}
/*</TcOutputUpdate>*/

////////////////////////////////////
//! \brief Implementation of the safe user module cycle update function
////////////////////////////////////
/*<TcCycleUpdate>*/
VOID CSafeModule::CycleUpdate()
{
    // Put your cycle update code here
}
/*</TcCycleUpdate>*/

/*<UserDefinedFunctionsDef>*/          // Implement internal module functions here
/*</UserDefinedFunctionsDef>*/

//! Reference to project FCS symbol
extern UINT32 SAFETY_PROJECT_FCS;      // Do NOT read, write or remove!
};

// Rename according to TwinSAFE group name
SAFE_MODULE_DEF_EXPORT(TwinSafeGroup1);

/*</SafeUserApplicationCppFrontend>*/

```



Hinweis

Zulässige Modifikationen <TwinSAFE-Gruppenname>.cpp



- SAFE_MODULE_DEF (<TwinSAFE-Gruppenname>) (bei Änderung des TwinSAFE Gruppennamens im Projektbaum, muss dieser Eintrag durch den Anwender ebenfalls angepasst werden)
- SAFE_MODULE_DEF_EXPORT (<TwinSAFE-Gruppenname>) (bei Änderung des TwinSAFE Gruppennamens im Projektbaum, muss dieser Eintrag durch den Anwender ebenfalls angepasst werden)
- Anwenderprogramm Initialisierung nur zwischen
/*<TcInnit>/ ... /*</TcInnit>*/
- Anwenderprogramm InputUpdate nur zwischen
/*<TcInputUpdate>/ ... /*</TcInputUpdate>*/
- Anwenderprogramm CycleUpdate nur zwischen
/*<TcCycleUpdate>/ ... /*</TcCycleUpdate>*/
- Anwenderprogramm OutputUpdate nur zwischen
/*<TcOutputUpdate>/ ... /*</TcOutputUpdate>*/
- Anwender-Funktionen nur im Bereich zwischen
/*<UserDefinedFunctions>*/ ... /*</UserDefinedFunctions>*/
- Nicht verwendete bzw. leere Bereiche müssen zur Vermeidung von Warnungen mit einem Kommentar versehen werden in der Form:
/*<IntentionallyEmpty>*/

5.2 Sichere Codierregeln

5.2.1 Begriffserklärung

Begriff	Erklärung
strenge/starke Typisierung	In Safety C sind implizite Typkonvertierungen nicht zulässig. Bei jeder Operation, Zuweisung oder Parameterübergabe muss der Datentyp aller Operanden zu dem Zieldatentyp passen (mit wenigen als sicher zu betrachtenden Ausnahmen). Eine Diskrepanz zwischen den Datentypen muss durch eine explizite Konvertierung behoben werden. (siehe Kapitel <u>Strenge Typisierung</u> [► 58])
pure functions	Die Funktion liefert immer das gleiche Ergebnis, wenn die gleichen Argumente an die Funktion übergeben werden. Das Ergebnis der Funktion hängt nicht von internen lokalen oder globalen bzw. Variablen, sonstigen internen Informationen oder Eingangssignalen ab. Eine pure function ist eine Funktion ohne Seiteneffekt. Alle bereitgestellten Hilfsfunktionen sind pure functions.
non-pure functions / impure functions	Jede Funktion, die Zustandsvariablen bzw. Ein-/Ausgangssignale oder sonstige interne Informationen verwendet ist potentiell als „impure“ oder „non-pure“ zu betrachten, da sie zu unterschiedlichen Aufrufzeitpunkten unterschiedliche Ergebnisse liefern kann. Eine non-pure / impure function ist eine Funktion mit Seiteneffekt. Benutzerdefinierte Funktionen werden grundsätzlich als impure betrachtet, selbst wenn sie die Kriterien einer pure function erfüllen.
Operatorpräzedenz	Bestimmt die anzuwendende Reihenfolge für die Operatoren einer Programmiersprache, wodurch ein zusammengesetzter Ausdruck in einen Syntaxbaum zur weiteren Verarbeitung durch den Compiler überführt werden kann. In Safety C sind die Operatorpräzedenzen durch die Ableitung vom C/C++ Standard eindeutig vorgegeben.
Operatorassoziativität	Bestimmt die anzuwendende Reihenfolge von Operatoren gleicher Präzedenz in einem zusammengesetzten Ausdruck, sofern diese nicht durch Klammerung explizit gegeben ist. In Safety C ist die Operatorassoziativität durch die Ableitung vom C/C++ Standard eindeutig bestimmbar. Der Ausdruck $(a + b + c)$ wird daher durch die Links-Assoziativität der Addition eindeutig mit folgender impliziter Klammerung verarbeitet: $((a + b) + c)$ Dieser Effekt ist insbesondere zu beachten, wenn die Reihenfolge der anzuwendenden Operatoren einen Effekt auf das Gesamtergebnis haben kann. In Safety C ist die Operatorassoziativität auch für die Einhaltung der strengen Typisierung zu berücksichtigen.
Auswertungsreihenfolge	Bestimmt die anzuwendende Reihenfolge bei der Auswertung von Operanden, wie z.B. das Ausführen von Funktionsaufrufen oder das Inkrementieren einer Variable als Teil eines zusammengesetzten Ausdrucks. Die Auswertungsreihenfolge kann Auswirkungen auf das Ergebnis eines Ausdrucks haben, wenn dieser mehrere voneinander abhängige Seiteneffekte enthält. Der C/C++ Standard definiert keine eindeutige Auswertungsreihenfolge! Deshalb werden zum Zwecke der Eindeutigkeit von Safety C Code Einschränkungen der Ausdrücke getroffen, um so die Effekte der nicht eindeutig bestimmten Auswertungsreihenfolge zu vermeiden.
Rekursion	Bei einer Rekursion ruft sich eine Funktion oder Prozedur selbst wieder auf. Dabei wird die direkte und die indirekte Rekursion unterschieden: direkte Rekursion: A() ruft A() auf indirekte Rekursion: A() ruft B() auf, B() ruft A() auf, ...
Kurzschluss-Auswertung	Die Kurzschluss-Auswertung oder auch bedingte Auswertung (short-circuit evaluation) bezeichnet ein Verhalten, was bei Booleschen Operationen dazu führt, dass die Auswertung vorzeitig abgebrochen wird. Wird schon durch einen Teil der Operation das Ergebnis eindeutig bestimmt, werden die folgenden Teiloperationen nicht mehr ausgeführt. Beispiel: $c = a \&\& b;$ Wenn a false ist, ist das Ergebnis c eindeutig bestimmbar und somit wird die Operation abgebrochen und c direkt auf false gesetzt ohne vorher b auszuwerten.

5.2.2 Allgemein

 Hinweis	<p>Datentypen</p> <p>Der Anwender sollte bei der Entwicklung der Safety Applikation darauf achten, dass zwischen den Datentypen BOOL und safeBOOL bzw. den weiteren safe und nonsafe Datentypen unterschieden wird. Dies vereinfacht die anschließende Verifikation und Validierung der realisierten Applikation. In V1 sind safe und nonsafe Datentypen aus Sicht des Typsystems identisch. Eine applikative Vermischung sicherer und nicht-sicherer Eingangssignale muss daher vom Anwender entsprechend der geltenden Normen bewertet werden.</p>
 Hinweis	<p>Kontrollstrukturen</p> <p>For-Schleifen sind While-Schleifen vorzuziehen, da die Terminierung einfacher zu erfassen ist.</p>

Es gelten folgende generelle Regeln für die kontextabhängige Einschränkung von Operatoren und Funktionsaufrufen in Ausdrücken:

- Die Logikoperatoren &&, || dürfen nur in Bedingungsausdrücken von if-, for-, und while-Anweisungen verwendet werden (sowie in den Bedingungsausdrücken von Assert-Anweisungen).
- Die Bedingungsausdrücke von if-, for-, und while-Anweisungen müssen das Ergebnis einer Vergleichsoperation (==, !=, <, >, <=, >=) liefern, komplexe zusammengesetzte Operanden dieser Vergleichsoperation müssen zudem geklammert sein, z.B.:

```
while(flag==true)           //Statt while(flag)
if (0>(a+b))                //Statt if(0>a+b) oder if(a+b)
```

- Funktionsaufrufe mit potentiellen Seiteneffekten (sogenannte „non-pure Functions“) dürfen nur als einfache Anweisungen entweder ohne Zuweisung eines Rückgabewertes oder (im Falle eines Rückgabewertes) mit direkter Zuweisung des Rückgabewertes zu einer Variablen aufgerufen werden, damit die Auswertungsreihenfolge eindeutig ist (d.h. sie dürfen nicht als Teil eines Bedingungs- oder Switch-Ausdrucks aufgerufen werden). Erlaubt ist z.B.:


```
INT32 r1 = MyNonPureFunc1();
INT32 r2 = MyNonPureFunc2();
INT32 r = r1 + r2;
```

 Nicht erlaubt ist z.B.:


```
INT32 r = MyNonPureFunc1() + MyNonPureFunc2();
```
- Funktionen ohne Seiteneffekte („pure Functions“) dürfen auch als Teil von Switch- oder Bedingungsausdrücken. Seiteneffektfreie Funktionen sind z.B. von der TwinCAT Safety PLC bereitgestellte Hilfsfunktionen wie Mathematik-Funktionen oder Konvertierungsfunktionen. Sie können auch mit „non-pure Functions“ kombiniert werden, so lange es nur einen Aufruf einer „non-pure Function“ pro Ausdruck gibt. Erlaubt ist z.B.:


```
INT32 r1 = MyPureFunc1() + MyNonPureFunc1(MyPureFunc2());
```
- Für die vorgegebenen Schnittstellenfunktionen der Module (Init, InputUpdate, CycleUpdate, OutputUpdate) sowie benutzerdefinierte Funktionen und Funktionsblöcke wird generell angenommen, dass diese potentiell Seiteneffekte haben. Von daher dürfen diese auch nur als einfache Zeilenanweisung mit oder ohne Zuweisung aufgerufen werden (siehe Punkt 3), z.B.:


```
MyFB1.CycleUpdate();
int y = MyFunction(42, y);
y = MyFunction2();
```

 Der Aufruf der Schnittstellenfunktionen als Einstiegspunkt der Benutzeranwendung ist nur durch die sichere Laufzeitumgebung zulässig. Benutzerdefinierte Aufrufe von Init() und CycleUpdate() sind nur für FB-Instanzen erlaubt (FBs sind in V1 nicht unterstützt).
- Die Operatoren Post-Inkrement (++) und Post-Dekrement (--) dürfen nur als einfache Zeilenanweisung (z.B. i++; i--;) verwendet werden (mit Ausnahme des dritten Ausdrucks der Anweisungszeile eines for-Schleifenkopfs).
- Der Zuweisungsoperator ist nur als Teil von einfachen Anweisungen mit einer Zuweisung erlaubt, mit Ausnahme des ersten und dritten Ausdrucks der Anweisungszeile eines for-Schleifenkopfs (z.B. INT32 i=0;). Mehrfachzuweisungen pro Zeilenanweisung sind nicht erlaubt (z.B. a=b=c;), ebenso sind Zuweisungen als Default-Wert-Initialisierungen in Funktionssignaturen verboten.

- Generell sind alle primitiven Datentypen bei Zuweisung sowie Parameterübergabe und Funktionsrückgabe im Gegensatz zu Standard C/C++ streng zu typisieren. Ausnahmen bilden hier die Kombination aus kleinerem Quelldatentyp und größerem Zieldatentyp mit gleicher Vorzeichenbehaftung (siehe Kapitel Strenge Typisierung [▶ 58]).
- Zur Vermeidung unbeabsichtigter Effekte sind weitere Kombinationen von Datentypen und Operatoren eingeschränkt (siehe Kapitel Strenge Typisierung [▶ 58]).
- Die Verwendung des expliziten Typumwandlungsoperators erfordert zusätzliche Klammerung, wenn der rechtsseitig vom Umwandlungsoperator stehende Ausdruck ein nicht geklammerter, komplexer Ausdruck ist:
`INT32 i32 = (INT32)u16 * -u16; // Unzulässig, da nicht eindeutig ist, welcher Ausdruck als Operand des Typumwandlungsoperators gemeint ist`
`INT32 i32 = (INT32)(u16) * -u16; // Zulässig, da eindeutig`

	<p>Beschränkung der Komplexität</p>
<p>Hinweis</p>	<p>Es werden Warnungen bei Überschreiten einer entsprechenden Komplexität herausgegeben (bekannt als McCabe-Index oder auch zyklomatische Komplexität). Die Empfehlung liegt bei einem Index < 10 pro Funktion.</p> <ul style="list-style-type: none"> • Index > 20 pro Modul-Funktion • Index > 50 pro Modul (Fehler bei Index > 1000)
	<p>Beschränkung der Anzahl Modulvariablen</p>
<p>Hinweis</p>	<p>Bei einer Anzahl von mehr als 50 Modulvariablen wird eine Warnung ausgegeben, bei Verwendung von mehr als 1000 Modulvariablen ein Fehler.</p>

5.2.3 Strenge Typisierung

Im Folgenden gilt für alle erwähnten Typbezeichner:

- BOOL ist äquivalent zu safeBOOL.
- INT8 ist äquivalent zu safeINT8, SINT, safeSINT.
- UINT8 ist äquivalent zu safeUINT8, USINT, safeUSINT.
- INT16 ist äquivalent zu safeINT16, INT, safeINT.
- UINT16 ist äquivalent zu safeUINT16, UINT, safeUINT.
- INT32 ist äquivalent zu safeINT32, DINT, safeDINT.
- UINT32 ist äquivalent zu safeUINT32, UDINT, safeUDINT.

Erläuterungen zu den in den folgenden Tabellen dargestellten Operator-Typeinschränkungen:

1. Logik-Operatoren (&&, ||, !) dürfen nur Variablen, Literale (*true*, *false*) oder Ausdrücke vom Typ BOOL als Operanden haben.
2. Arithmetik-Operatoren (+, -, *, /, %) dürfen nur Variablen, Literale (dezimal, hexadezimal, binär) oder Ausdrücke ganzzahliger arithmetischer Typen (U)INT(8/16/32) als Operanden haben.
3. Bitwise-Operatoren (&, |, ^, ~, <<, >>) dürfen nur Variablen, Literale (dezimal, hexadezimal, binär) oder Ausdrücke ganzzahliger vorzeichenloser arithmetischer Typen UINT(8/16/32) als Operanden haben.
4. Die Vergleichsoperatoren (<, >, <=, >=) dürfen keine Variablen, Literale (*true*, *false*) oder Ausdrücke vom Typ BOOL als Operanden haben.
5. Binäre Operatoren (Vergleich, Arithmetik, Bitweise, Logik) mit linkem und rechtem Operand (+, -, *, /, %, &, |, ^, <<, >>, ==, !=, <, >, <=, >=) dürfen nur Variablen, Literale (boolesch, dezimal, hexadezimal, binär) oder Ausdrücke gleichen Typs (sowie nach vorherigen Einschränkungen s.o.) haben.
6. „Integral Promotions“ (C++ Standardkonversionen zum Typ INT32) des Ergebnisausdrucks von Operationen mit Operanden kleiner ganzzahliger Datentypen (U)INT(8/16) müssen durch eine explizite Typkonversion neutralisiert werden, wenn dadurch eine der Regeln der Punkte 1.-5. verletzt wird.
z.B. im Fall einer Summenbildung von UIN8-Ausdrücken im UINT8-Zahlenraum:
1: UINT8 z = ...; UINT8 a = ...;
2: z = (UINT8)(a + (UINT8)(a + a)); //statt a = a + a + a;
Ohne Typkonversion des INT32-Ergebnisausdrucks von a+a zurück auf INT8 würde die Regel der Typgleichheit der anschließenden Addition verletzt. Gleiches gilt für die anschließende Zuweisung zum UINT8-Typ. Alternativ können alle UINT8-Ausdrücke zuvor in INT32-Ausdrücke konvertiert werden, um die Summenbildung im INT32-Zahlenraum durchzuführen:
3: UINT8 z = ...; UINT8 a = ...;
4: z = ((INT32)a) + ((INT32)a) + ((INT32)a);
Dabei sollte beachtet werden, dass dies ggf. zu einem anderen Ergebnis führt. Ziel der strengen Typisierung ist es, die Intention des Anwendungsentwicklers bzgl. der Typumwandlungseffekte explizit sichtbar zu machen.
Erlaubt ist z.B.:
INT16 i16; UINT8 u8; ...
INT32 i32 = i16; // Nicht typgleich, aber zulässig
UINT16 u16 = u8; // Nicht typgleich, aber zulässig
Weitere Ausnahmen gibt es bei der Initialisierung von Modulvariablen mit konstanten Literalen. Erlaubt ist z.B.:
INT8 i8; // Deklaration als Modulvariable
UNT16 u16; // Deklaration als Modulvariable
...
i8 = 0; // Zulässig obwohl konstantes Literal vom Typ INT32 ist
u16 = 42U; // Zulässig obwohl konstantes Literal vom Typ UINT32 ist
7. Der explizite Typkonversions-Operator () darf nur zur Umwandlung von einer Variable, einem Literal (dezimal, hexadezimal, binär) oder eines Ausdrucks von einem ganzzahligen arithmetischem Typ (U)INT(8/16/32) in einen anderen verwendet werden. Der explizite Typkonversions-Operator kann zu Daten- oder Vorzeichenverlust führen. Ist dies nicht gewünscht oder soll dies aufgedeckt werden, sollten die Hilfsfunktionen zur Konvertierung verwendet werden.
8. Die Zuweisung (=), Übergabe und Rückgabe von Funktionsparameter- und -ergebnissen darf nur von typgleichen Quelltyp zu einem Zieltyp bzw. Funktionssignatur verwendet werden oder aber von kleineren arithmetischen Datentypen zu einem größeren Datentypen ohne Wechsel zwischen vorzeichenbe-

haftetem und nicht vorzeichenbehaftetem Datentyp (sichere Ausnahmen der strengen Typisierung). Bei Initialisierungsstatements in der Art `UINT8 a = 10U`; wird durch den Compiler bereits geprüft, ob das Literal zum deklarierten Datentyp passt. Dies ist aber nur zulässig, wenn Deklaration und Initialisierung in einer einfachen Zeilenanweisung kombiniert sind oder wenn es sich um die Initialisierung einer einfachen Modulvariable handelt.

9. Komplexe Datentypen (structs) unterstützen nur die typgleiche Zuweisung (=), Übergabe als Funktionsparameter und die Rückgabe durch Funktionen, sowie den Zugriffsoperator (.) für Daten-Members. Die Anwendung expliziter Typkonversion auf komplexe Datentypen ist generell unzulässig.

Es sind nur Operator-Typ-Kombinationen zulässig, die einen Eintrag in den folgenden Tabellen haben. Der Typbezeichner in der Zelle ist der resultierende Ergebnistyp der Operation (unter Berücksichtigung von C++ Standardtypkonversionen). Der rechte Operand (RHS) ist in der Kopfzeile dargestellt, der linke Operand (LHS) in der ersten Spalte.

LHS/RHS: Linker/Rechter Operand unärer oder binärer Operatoren.

Typregeln für Arithmetik-Operatoren

+, -, *, /, %	BOOL	INT8	UINT8	INT16	UINT16	INT32	UINT32
BOOL							
INT8		INT32					
UINT8			INT32				
INT16				INT32			
UINT16					INT32		
INT32						INT32	
UINT32							UINT32

Unär	BOOL	INT8	UINT8	INT16	UINT16	INT32	UINT32
-x		INT32	INT32	INT32	INT32	INT32	
+x							

Unär	x++		x--	
BOOL				
INT8	INT8		INT8	
UINT8	UINT8		UINT8	
INT16	INT16		INT16	
UINT16	UINT16		UINT16	
INT32	INT32		INT32	
UINT32	UINT32		UINT32	

+=, -=, *=, /=, %=	BOOL	INT8	UINT8	INT16	UINT16	INT32	UINT32
BOOL							
INT8		INT8					
UINT8			UINT8				
INT16				INT16			
UINT16					UINT16		
INT32						INT32	
UINT32							UINT32

Typregeln für bit-weise Operatoren

&, , ^, <<, >>	BOOL	INT8	UINT8	INT16	UINT16	INT32	UINT32
BOOL							
INT8							
UINT8			INT32				
INT16							
UINT16					INT32		
INT32							
UINT32							UINT32

Unär	BOOL	INT8	UINT8	INT16	UINT16	INT32	UINT32
~x			INT32		INT32		UINT32

&=, =, ^=, <<=, >>=	BOOL	INT8	UINT8	INT16	UINT16	INT32	UINT32
BOOL							
INT8							
UINT8			UINT8				
INT16							
UINT16					UINT16		
INT32							
UINT32							UINT32

Typregeln für Logik Operatoren

&&,	BOOL	INT8	UINT8	INT16	UINT16	INT32	UINT32
BOOL	BOOL						
INT8							
UINT8							
INT16							
UINT16							
INT32							
UINT32							

Unär	BOOL	INT8	UINT8	INT16	UINT16	INT32	UINT32
!x	BOOL						

==, !=	BOOL	INT8	UINT8	INT16	UINT16	INT32	UINT32
BOOL	BOOL						
INT8		BOOL					
UINT8			BOOL				
INT16				BOOL			
UINT16					BOOL		
INT32						BOOL	
UINT32							BOOL

>, <, >=, <=	BOOL	INT8	UINT8	INT16	UINT16	INT32	UINT32
BOOL							
INT8		BOOL					
UINT8			BOOL				
INT16				BOOL			
UINT16					BOOL		
INT32						BOOL	
UINT32							BOOL

Typregeln für den expliziten Cast-Operator

()	BOOL	INT8	UINT8	INT16	UINT16	INT32	UINT32
BOOL							
INT8		INT8	INT8	INT8	INT8	INT8	INT8
UINT8		UINT8	UINT8	UINT8	UINT8	UINT8	UINT8
INT16		INT16	INT16	INT16	INT16	INT16	INT16
UINT16		UINT16	UINT16	UINT16	UINT16	UINT16	UINT16
INT32		INT32	INT32	INT32	INT32	INT32	INT32
UINT32		UINT32	UINT32	UINT32	UINT32	UINT32	UINT32

()	struct A	struct B	struct C
struct A			
struct B			
struct C			

Typregeln für den Memberzugriffs-Operator

.	struct A	struct B	struct C
BOOL	BOOL	BOOL	BOOL
INT8	INT8	INT8	INT8
UINT8	UINT8	UINT8	UINT8
INT16	INT16	INT16	INT16
UINT16	UINT16	UINT16	UINT16
INT32	INT32	INT32	INT32
UINT32	UINT32	UINT32	UINT32
struct A		struct B	struct C
struct B	struct A		struct C
struct C	struct A	struct B	

Typregeln für Zuweisungen, sowie Übergabe von Funktionsparametern und Rückgabe von Rückgabewerten

=	BOOL	INT8	UINT8	INT16	UINT16	INT32	UINT32
BOOL	BOOL						
INT8		INT8					
UINT8			UINT8				
INT16		INT16		INT16			
UINT16			UINT16		UINT16		
INT32		INT32		INT32		INT32	
UINT32			UINT32		UINT32		UINT32

=	struct A	struct B	struct C
struct A	struct A		
struct B		struct B	
struct C			struct C

5.2.3.1 Beispiele zum strengen Typsystem

Dieses Kapitel enthält Beispiele zum besseren Verständnis des strengen Typsystems, das bei der TwinCAT Safety PLC verwendet wird.



Betrachtung der Ausdrücke als Bäume

In C/C++ werden – vorgegeben durch den C/C++ Standard – komplexe Ausdrücke gemäß einer Baumstruktur entsprechend der impliziten Operator-Präzedenzen und Operator-Assoziativität verarbeitet.

Die Baumstruktur wird durch den C/C++ Compiler nach Standard bestimmt und kann z.T. implizite Typumwandlungen beinhalten, welche in Safety C verboten sind. In Safety C müssen die zulässigen Operator-Typ-Kombination bzw. Funktionssignaturen mit wenigen sicheren Ausnahmen exakt betrachtet werden (strenge Typisierung, siehe Matrix-Tabellen)

Durch explizite Klammerung können die impliziten Operator-Präzedenzen und die Operator-Assoziativität beeinflusst und damit die Baumstruktur verändert werden

Bäume bestehen aus folgenden Komponenten:

- **Blattknoten**  (sind Literale, Datenreferenzen und parameterlose Funktionsaufrufe)
- **Innere Knoten**  (sind Operatoren oder parametrisierte Funktionsaufrufe, die jeweils einen oder mehrere Teilausdrücke verarbeiten)
- **Baumkanten** INT32 (sind Zwischenergebnisse von ausgewerteten Teilausdrücken und haben jeweils einen Datentyp, welcher durch die Abarbeitung der Knoten gemäß der Baumstruktur vom Blattknoten (unten in der Baumstruktur) zum Wurzelknoten (oben in der Baumstruktur) statisch zur Übersetzungszeit bestimmbar ist)

Die Auswertungsreihenfolge der Blattknoten ist durch den C/C++ Standard nicht eindeutig definiert, und wird durch den Compiler festgelegt. In Safety C müssen Ausdrücke mit möglichen Seiteneffekten wegen der nicht definierten Auswertungsreihenfolge eingeschränkt werden (siehe auch [Ausdrücke und Operatoren](#) [▶ 73])

Summenbildung als Beispiel

Ausgangssituation: Der Benutzer will eine Summenbildung über ganzzahlige Ausdrücke vom Typ UINT8 in Safety C programmieren. Die Ausdrücke können Literale, Variablen oder Funktionsrückgaben sein, hier: a, b, c.

Intuitiver Ansatz in Standard C++:

Fall 1:

Wenn die Summe den UINT8-Zahlenraum überschreiten kann:

```
INT32 z = a + b + c;
```

Fall 2:

Wenn klar ist, dass die Summe den UINT8-Zahlenraum nicht überschreiten kann oder ein implizites Abschneiden (Modulo-Arithmetik) gewünscht ist:

```
UINT8 z = a + b + c;
```

5.2.3.1.1 Fall 1

Beispiel 1

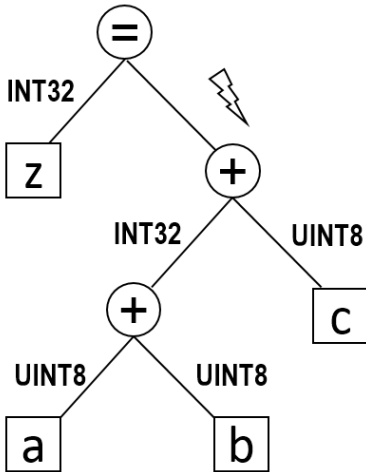


Abb. 62: Strenges Typsystem Fall 1 - Beispiel 1

```
UINT8 a; UINT8 b; UINT8 c;
...
INT32 z = a + b + c; // Type system error T8006 : binary '+ operator' is restricted to operands with equal type
```

Beispiel 2

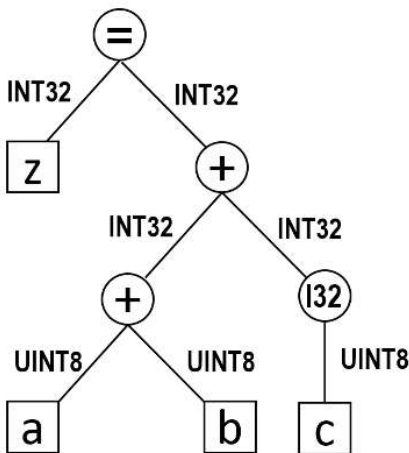


Abb. 63: Strenges Typsystem Fall 1 - Beispiel 2

```
UINT8 a; UINT8 b; UINT8 c;
...
INT32 z = a + b + (INT32)c; // OK but intention might be unclear
```

Beispiel 3

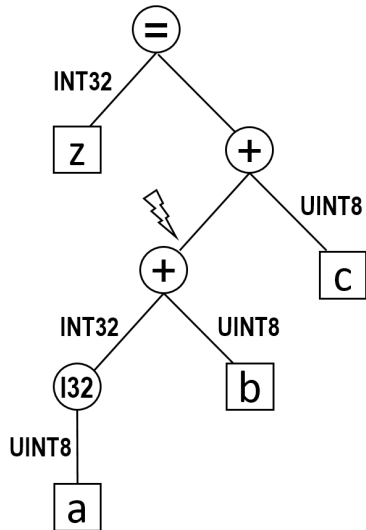


Abb. 64: Strenges Typsystem Fall 1 - Beispiel 3

```

UINT8 a; UINT8 b; UINT8 c;
...
INT32 z = ((INT32)a) + b + c; // Again, type system error T8006 as C/C++ binary addition operator is
left-associative
    
```

Beispiel 4

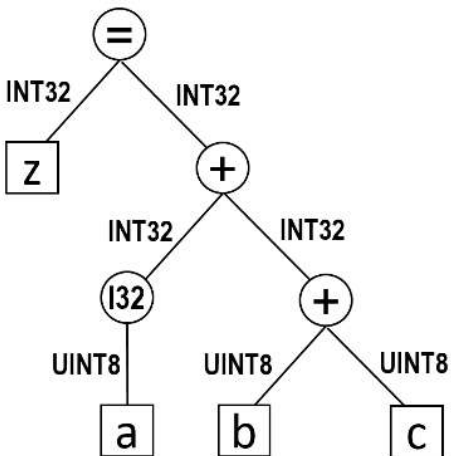


Abb. 65: Strenges Typsystem Fall 1 - Beispiel 4

```

UINT8 a; UINT8 b; UINT8 c;
...
INT32 z = ((INT32)a) + (b + c); // OK but intention might be unclear
    
```


Beispiel 5

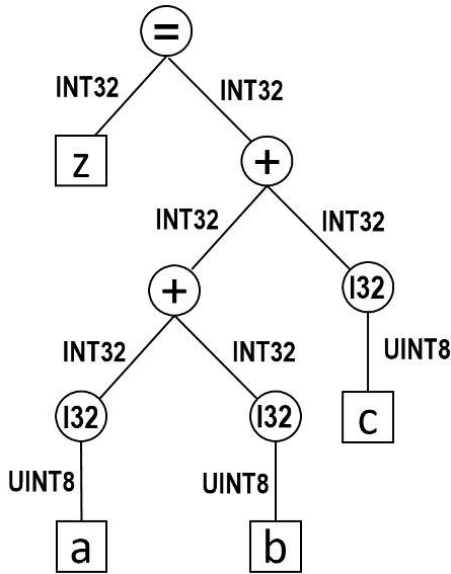


Abb. 66: Strenges Typsystem Fall 1 - Beispiel 5

```

UINT8 a; UINT8 b; UINT8 c;
...
INT32 z = ((INT32)a) + ((INT32)b) + ((INT32)c); // OK with clear intention
// Use this when lines get too long:
INT32 z = (INT32)a;
z += (INT32)b;
z += (INT32)c; // OK with clear intention and short lines
// Use this when overflow might accidentally occur and should be trapped:
INT32 z = (INT32)a;
... // Maybe lots of accumulations to z
z = ADDI32(z, (INT32)b);
z = ADDI32(z, (INT32)c);
    
```

5.2.3.1.2 Fall 2

Beispiel 1

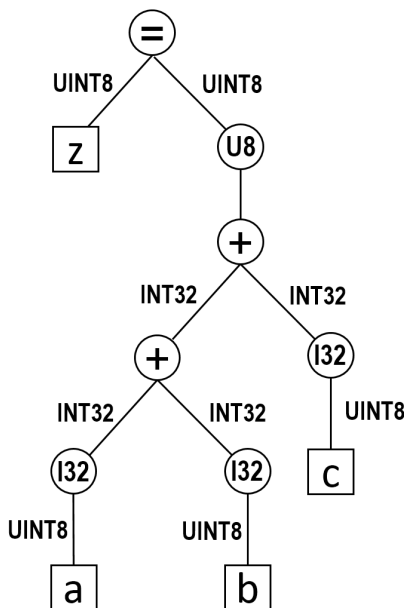


Abb. 67: Strenges Typsystem Fall 2 - Beispiel 1

```

UINT8 a; UINT8 b; UINT8 c;
...
UINT8 z = (UINT8)(((INT32)a) + ((INT32)b) + ((INT32)c)); // OK but lots of code to write
    
```

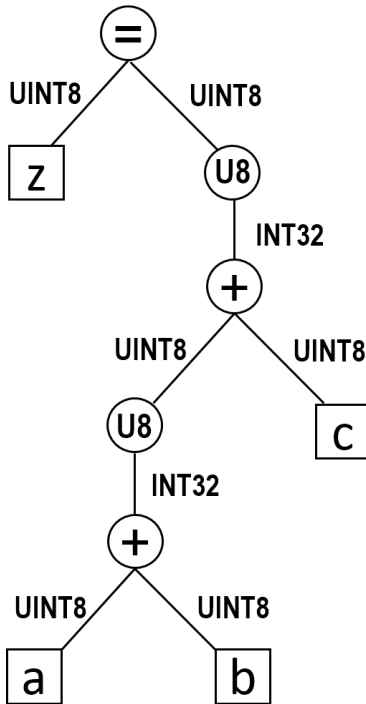
Beispiel 2

Abb. 68: Strenges Typsystem Fall 2 - Beispiel 2

```

UINT8 a; UINT8 b; UINT8 c;
...
UINT8 z = (UINT8)((UINT8)(a + b) + c); // OK but can get hard to read

// Better use this:
UINT8 z = a;
z = (UINT8)(z + b);
z = (UINT8)(z + c); // OK and with clear intention


// Or even that:
UINT8 z = a;
z += b;
z += c; // OK, compact and with clear intention

```


5.3 Zulässiger Sprachumfang


5.3.1 Einfache Datentypen


Folgende Tabelle listet die zulässigen, einfachen Datentypen und ihre alternativen Typbezeichner mit Wertebereichen auf.

 Achtung	<p>Über-/Unterläufe und Division durch Null</p> <p>Überläufe oder Unterläufe des zulässigen Wertebereichs vom Datentyp INT32 sind im C/C++ Standard nicht definiert und können zu Dateninkonsistenzen führen. Diese Inkonsistenzen führen zu einem sicheren Laufzeitfehler in der TwinCAT Safety PLC nur wenn sie sich auch auf ein Ausgangstelegramm auswirken. Wenn dieser Fall nicht applikativ oder durch Analyse durch den Anwender ausgeschlossen werden kann, sollten die sicheren Hilfsfunktionen an Stelle der nativen C/C++ Operatoren verwendet werden.</p> <p>Überläufe oder Unterläufe des zulässigen Wertebereichs vom Datentyp UINT32 sind dagegen im C/C++ Standard definiert und führen zu einem Durchlauf (Modulo-Arithmetik).</p> <p>Division durch Null ist im C/C++ Standard nicht definiert und kann zu Dateninkonsistenzen oder zum Abbruch der Sicherheitsapplikation führen, was wiederum zu einem sicheren Laufzeitfehler in der TwinCAT Safety PLC führt. Wenn dieser Fall nicht applikativ oder durch Analyse durch den Anwender ausgeschlossen werden kann, sollten die sicheren Hilfsfunktionen an Stelle der nativen C/C++ Operatoren verwendet werden.</p> <p>Der Anwender muss den korrekten Datentyp für die zu erwartenden Werte seiner Operationen entsprechend auswählen.</p>
---	---

Datentyp Standard	Datentyp Safe	Wertebereich
BOOL	safeBOOL	FALSE / TRUE
UINT8	safeUINT8	0 .. 255
USINT	safeUSINT	0 .. 255
INT8	safeINT8	-128 .. 127
SINT	safeSINT	-128 .. 127
UINT16	safeUINT16	0 .. 65535
UINT	safeUINT	0 .. 65535
INT16	safeINT16	-32768 .. 32767
INT	safeINT	-32768 .. 32767
UINT32	safeUINT32	0 .. 4.294.967.295
UDINT	safeUDINT	0 .. 4.294.967.295
INT32	safeINT32	-2.147.483.648 .. 2.147.483.647
DINT	safeDINT	-2.147.483.648 .. 2.147.483.647

 Hinweis	<p>Datentypen</p> <p>INT und UINT sind in der TwinCAT Safety PLC 16-Bit-Typen, wie sie in der IEC 61131-3 definiert sind. In TwinCAT C++ sind INT und UINT 32-Bit-Typen!</p>
---	--

 Hinweis	<p>Unterschied safe und standard</p> <p>Sicherheitstechnisch gibt es keinen Unterschied bei der Verarbeitung von Safe und Standard Datentypen (safe<TYP> / <TYP>), es wird jedoch empfohlen, zur Kennzeichnung sicherer Signale den Präfix „safe“ bei der Typdeklaration zu verwenden, um eine Validierung und Verifikation der Applikation zu erleichtern. Die applikative Verknüpfung sicherer und nicht-sicherer Daten muss durch Anwender gemäß geltender Normen bewertet werden.</p>
---	--

 Achtung	<p>Nicht zulässige Typen</p> <p>Alle nicht aufgelisteten C/C++ Typbezeichner, wie z.B. unsigned int, char usw. sind nicht zulässig. Pointer-Typen und Referenz-Typen sind ebenfalls nicht zulässig. Es muss immer eine Wertübergabe per Call-by-Value erfolgen.</p>
---	--

**Hinweis****Vorgaben**

- Lokale Variablen müssen vor ihrer Benutzung initialisiert werden und müssen verwendet werden
- Modulvariablen müssen in der Init-Funktion initialisiert werden und im Code gelesen werden
- Eingangsvariablen dürfen nicht geschrieben werden
- Ausgangsvariablen dürfen nicht gelesen werden

5.3.2 Aufzählungstypen

Enumerations (benutzerdefinierte enum-Typen) sind für V1 nicht vorgesehen. Alternativ können Konstanten-Defines genutzt werden, siehe [Literale und Konstante](#) [► 74].

5.3.3 Datenstrukturen

5.3.3.1 Structs

Struct-Typen kapseln die I/O-Daten (sichere, nicht-sichere Ein-/Ausgangsdaten, sowie gruppenübergreifende Daten). Diese Struct-Typen werden als Bestandteil der <TwinSAFEGruppenName>IoData.h aus der Aliasgeräte-Konfiguration generiert.

Default-Variablen dieser Struct-Typen sind im TwinSAFEGruppen-Template als Modulvariablen mit dem Namenspräfix „s“ angelegt.

Weiterhin können eigene Struct-Typen in der Modul-Header Datei im Bereich <UserDefinedTypes>...</UserDefinedTypes> angelegt werden, wobei diese Definition optional ein führendes „typedef“ haben kann (aber nicht muss). Eine globale Instanzvariable eines structs ist NICHT zulässig.

Diese können auch hierarchisch verschachtelt sein. Innere Struct-Typen von verschachtelten Struct-Typdefinitionen MÜSSEN hingegen eine innere Instanzvariable für den Zugriff haben. Innere Struct-Typen sind anonyme Typen und dürfen nicht unabhängig von ihrem hierarchisch übergeordneten Struct-Typ instanziiert werden.

Beispiele

```
typedef struct MyData
{
    INT32    a;
    UINT8    b;
    BOOL     c;
};

struct MyFunctionInterface
{
    struct
    {
        BOOL a;
        BOOL b;
    } In;
    struct
    {
        BOOL z;
    } Out;
}
```

Der Zugriff auf Variablen von Struct-Instanzen erfolgt über den „.“ Zugriffsoperator.

Selbst-definierte Struct-Typen können als bidirektionales Interface zur Übergabe von I/O-Daten für Funktionsaufrufe verwendet werden. Die Übergabe erfolgt aber weiterhin als Wertübergabe („Call-by-Value“).

Der Zugriff erfolgt dann z.B. über

```
MyFunctionInterface callFunc;
```

```
callFunc.In.a = true;  
callFunc.In.b = false;  
callFunc      = MyFunction(callFunc);  
BOOL result  = callFunc.Out.z;
```

Struct-Instanzen müssen wie einfache Variablen vor ihrer Verwendung initialisiert werden. Default-Initialisierungen innerhalb der Typ-Definitionen sind nicht erlaubt!

5.3.3.2 Arrays

Werden in V1 noch nicht unterstützt.

5.3.4 Einfache Anweisungen

Einfache Zeilanweisungen enden mit einem Semikolon.

Folgende Typen werden unterstützt als Teil eines Funktionsrumpfes oder des Rumpfes eines Kontrollstrukturblockes:

Typ-0

<Ausdruck>;

z.B. Funktionsaufruf ohne Rückgabe.

Typ-1

<Typ> <Bezeichner>;

z.B. Deklaration von Variablen ohne Initialisierung.

Typ-2

<Bezeichner> = <Ausdruck>;

z.B. Initialisierung von Variablen

z.B. Operationen/Funktionsaufruf mit Ergebniszuweisung

Typ-3

<Typ> <Bezeichner> = <Ausdruck>;

z.B. Deklaration von Variablen mit Initialisierung.

Sonderfälle

Anweisungen für Post-Inkrement/Dekrement (z.B. i++;) werden wie Typ-2-Anweisungen betrachtet.

Das gleiche gilt für Operationen mit Zuweisung (z.B. a += b).

Einschränkungen

Kombinierte Variablendeklarationen sind nicht zulässig:

z.B. INT32 i, j;

Mehrfachzuweisungen sind nicht zulässig:

z.B. a = b = c;

HINWEIS: Bestandteil 1 und 3 einer for-Kontrollflussanweisung (for (<1>; <2>; <3>)) gelten jeweils als einfache Anweisung, obwohl sie zu einer Zeilanweisung zu gehören scheinen.

Spezielle Anweisungen

return <Ausdruck>;

Ist nur am Ende einer Funktion mit Rückgabewert zulässig.

Break;

Ist nur am Ende eines „case:“ / „default:“ - Blocks zulässig.

Alle weiteren C/C++ Kontrollflussanweisungen, wie z.B. goto, continue, throw usw. sind nicht zulässig

5.3.5 Kontrollstrukturen

5.3.5.1 If-Else

Grundlegende Form

```
if (<COND>)
{
    <BLOCK>
}
else
{
    <BLOCK>
}
```

Richtlinien

- Der Else-Zweig ist zwingend anzulegen
(wenn dieser leer ist, muss zur Vermeidung einer Warnung der Spezialkommentar /
<IntentionallyEmpty/>/ eingefügt werden)
- Es sind keine Else-If-Zweige erlaubt
- Einschränkungen für den <COND>-Ausdruck:
 - Muss immer das Ergebnis einer Vergleichsoperation (<, >, <=, >=, ==, !=) sein, wobei linker und rechter Teilausdruck des Vergleichs ggf. geklammert sein müssen (wenn diese nicht aus einem einfachen Literal oder Bezeichner bestehen)
 - Keine Funktionsaufrufe mit potentiellen Seiteneffekten
 - Keine Zuweisungen, keine Post-/Pre-Inkrement/Dekrement

[CODEBEISPIEL]

```
if ((safeIn1 && safeIn2) == true)
{
    sSafeOutputs.EL2904_FSoE_4.OutputChannel1 = true;
    sSafeOutputs.EL2904_FSoE_13.OutputChannel4 = true;
}
else
{
    sSafeOutputs.EL2904_FSoE_4.OutputChannel1 = false;
    sSafeOutputs.EL2904_FSoE_13.OutputChannel4 = false;
}
```

5.3.5.2 While

Grundlegende Form

```
while (<COND>)
{
    <BLOCK>
}
```

Richtlinien

- Kein break Statement erlaubt
- Kein continue Statement erlaubt
- Einschränkungen für den <COND>-Ausdruck:

- Muss immer das Ergebnis einer Vergleichsoperation (<, >, <=, >=, ==, !=) sein, wobei linker und rechter Teilausdruck des Vergleichs ggf. geklammert sein müssen (wenn diese nicht aus einem einfachen Literal oder Bezeichner bestehen)
- Keine Funktionsaufrufe mit potentiellen Seiteneffekten
- Keine Zuweisungen, keine Post-/Pre-Inkrement/Dekrement
- Spezialkommentar /*<LoopBound max="N"/>*/ muss zu Beginn von <Block> eingefügt sein um Warnung zu verhindern, wobei N die Anzahl der erwarteten Durchläufe enthält (N>1).

[CODEBEISPIEL]

```
while (safeCounter < 10)
{
    /*<LoopBound max="10"/>*/
    safeCounter++;
}
```

5.3.5.3 For

Grundlegende Form

```
for (<STMT1>; <COND>; <STMT2>)
{
    <BLOCK>
}
```

- Kein break-Statement erlaubt
- Kein continue-Statement erlaubt
- Einschränkung für <STMT1>-Ausdruck
 - Typ-3 (siehe einfache Anweisungen)
- Einschränkungen für den <COND>-Ausdruck
 - Muss immer das Ergebnis einer Vergleichsoperation (<, >, <=, >=, ==, !=) sein, wobei linker und rechter Teilausdruck des Vergleichs ggf. geklammert sein müssen (wenn diese nicht aus einem einfachen Literal oder Bezeichner bestehen)
 - Keine Funktionsaufrufe mit potentiellen Seiteneffekten
 - Keine Zuweisungen, keine Post-/Pre-Inkrement/Dekrement
- Einschränkung für <STMT2>-Ausdruck
 - Typ-2 (siehe einfache Anweisung)
 - Post-Inkrement/Dekrement Anweisung
- Wenn for-Schleife nicht in Normalform ist, z.B. for (int i=0; i<10; i++), gilt: Spezialkommentar /*<LoopBound max="N"/>*/ muss zu Beginn von <Block> eingefügt sein um Warnung zu verhindern, wobei N die Anzahl der erwarteten Durchläufe enthält (N>1).

[CODEBEISPIEL]

```
for (INT32 i=N; i >= 0; i--=2)
{
    /*<LoopBound max="42"/>*/
    DoSomeComputations();
}
```

5.3.5.4 Switch-Case

Grundlegende Form

```
switch (<EXPR1>)
{
    case <EXPR2>:
        <BLOCK>
        break;
    ...
    default:
        <BLOCK>
        break; }
```

Richtlinien

- Mindestens ein case-Block erforderlich
- Default-Block ist zwingend erforderlich
- Case/default-Block müssen mit break-Statement enden
- Einschränkung für <EXPR1>-Ausdruck
 - Keine Funktionsaufrufe mit potentiellen Seiteneffekten
 - Kein logischer Ausdruck (kein Ausdruck vom Typ BOOL)
 - Keine Zuweisungen, keine Post-/Pre-Inkrement/Dekrement
- Einschränkung für <EXPR2>-Ausdruck
 - Konstanter Ausdruck (keine Variablen, keine Funktionsaufrufe)
 - Kein logischer Ausdruck (kein Ausdruck vom Typ BOOL)

5.3.6 Ausdrücke und Operatoren



Hinweis

Ausdrücke und Operatoren

Alle Operationen folgen der C++ Semantik für die entsprechenden einfachen Datentypen. Alle Operationen wenden die im C++ Standard definierten Typerweiterungen an (Promotion-Regeln), so dass der Ergebnisausdruck einer Operation ggf. nicht dem der/des Operanden entspricht. Dies muss ggf. mit einer expliziten Typkonvertierung, aufgrund der strengen Typisierung einfacher Datentypen in Safety C berücksichtigt werden (es sind mit wenigen sicheren Ausnahmen keine impliziten Typkonvertierungen zulässig).

Zulässige Operatoren

Zuweisungsoperatoren

binär	a=b
-------	-----

Einschränkungen:

- Nicht als Teil von Bedingungsausdrücken zulässig, keine Mehrfachzuweisungen.
- Operanden a und b müssen Typgleich sein, oder aber signed/signed bzw. unsigned/unsigned und mit Bitbreite a größer b.

Arithmetische Operatoren

unär	-a
binär	a+b, a-b, a*b, a/b, a%b
mit Zuweisung	a+=b, a-=b, a*=b, a/=b, a%=b
Post-Inkrement/Dekrement	a++, a--

Einschränkungen:

- Nur einfache, arithmetische Datentypen zulässig (kein BOOL, keine Structs).
- Die Operanden a, b müssen vom gleichen Typ sein.
- Mit Zuweisung nur als Teil von Typ-2-Anweisungen erlaubt.
- Inkrement/Dekrement nicht als Teil von Ausdrücken erlaubt (nur als einfache Zeilenanweisung).
- Über-/Unterläufe können undefiniertes Verhalten erzeugen. Entsprechende Prüfungen vornehmen oder sichere Hilfsfunktionen verwenden!

Bitweise Operatoren

unär	~a
binär	a&b, a b, a^b, a<<b, a>>b
mit Zuweisung	a&=b, a =b, a^=b, a<<=b, a>>=b

Einschränkungen:

- Nur einfache, vorzeichenlose arithmetische Datentypen zulässig (UINT8, UINT16, UINT32).
- Die Operanden a, b müssen vom gleichen Typ sein.
- Mit Zuweisung nur als Teil von Typ-2-Anweisungen erlaubt.
- Shift-Operationen können zu undefiniertem Verhalten führen. Entsprechende Prüfungen vornehmen oder sichere Hilfsfunktionen verwenden!

Logik Operatoren

unär	!a
binär	a&&b, a b, a!=a

Einschränkungen:

- Nur Typ BOOL zulässig.
- Short-Circuit-Operatoren &&, || sind nur als Teil von Bedingungsausdrücken erlaubt. Als Ersatz siehe dazu sichere Hilfsfunktionen mit vollständiger Auswertung: AND(a,b), AND3(a,b,c), AND4(a,b,c,d) sowie OR(a,b), OR3(a,b,c), OR4(a,b,c,d).

Vergleichsoperatoren

binär	a==b, a!=b, a<b, a>b, a<=b, a>=b
-------	----------------------------------

Einschränkungen:

- Nur einfache Datentypen zulässig (keine Structs).
- Die Operanden a, b müssen vom gleichen Typ sein.
- Vergleich von BOOL ist nur mit == und != zulässig.

Explizite Typumwandlung

binär	(<Typ><Ausdruck>
-------	------------------

Einschränkungen:

- Nur einfache Datentypen zulässig (keine Structs).
- Keine explizite Typkonvertierung von/nach BOOL zulässig.
Als Ersatz für Typkonvertierung von BOOL zu arithmetisch, siehe sichere Hilfsfunktionen mit eindeutiger Definition.
- Explizite Konvertierungen können zu Vorzeichen- und Datenverlust führen. Entsprechende Prüfungen vornehmen oder sichere Hilfsfunktionen verwenden!

Struct Zugriff

binär	a.b
-------	-----

5.3.7 Literale und Konstante

Literale können boolesch, dezimal, hexadezimal und binär angegeben werden. Es gelten die C/C++ Promotion-Regeln.

Ganzzahlige Literale

Wertebereich $0 \dots 2^{31}-1$ wird als Ausdruck vom Typ (safe)INT32 erfasst.

Wertebereich $2^{31} \dots 2^{32}-1$ wird als Ausdruck vom Typ (safe)UINT32 erfasst.

Mit dem Suffix „U“ werden Literale mit Typ UINT32 erfasst, auch wenn sie als INT32 darstellbar sind.



Hinweis

Vorzeichen

Literale werden ohne Vorzeichen erfasst, sprich ein Minus ist bereits eine Operation. Ein Plus als Vorzeichen ist nicht zulässig, da Literale implizit positiv erfasst werden.

Boolesche Literale

Literale false, true werden als Ausdruck vom Typ (safe)BOOL erfasst

Dezimal-Format

0-9 mit optionalem Suffix U

Hexadezimal-Format

0-9 und a-f bzw. A-F mit Präfix 0x oder 0X und optionalem Suffix U

Binär-Format

0-1 mit Präfix 0b oder 0B und optionalem Suffix U

Beispiele für zulässige Literale

- true

- false
- 0U
- 987654321
- 0xFF
- 0x0
- 0XFEDCBA98U
- 0b11010100
- 0B0U



Hinweis

Typumwandlung

Der Typ des erfassten Ausdrucks ist auch bei der Zuweisung von Literalen zu berücksichtigen, so führt z.B.

```
INT8 x;
```

```
x = 0;
```

zu einem Typfehler (eine Typumwandlung durch (INT8) ist hier notwendig)

Ausnahme stellt hier eine Deklarationsanweisung mit kombinierter Initialisierung dar, sofern das Literal zum Zieltyp passt und das Vorzeichensuffix richtig gewählt ist, also z.B.:

```
UINT16 x = 65535U;
```

Eine weitere Ausnahme ist die Initialisierung von einfachen Modulvariablen, welche auch Abseits ihrer Deklaration in der Modul-Header durch ein Typ-2-Statement mit passendem Literal zugewiesen werden können, also z.B.:

```
i8ModuleVar = 42;
```

Konstante / Präprozessor-Defines

Das Schlüsselwort „const“ ist nicht zulässig. Konstante sind mittels Präprozessor-Define zu definieren (siehe dafür vorgesehener Abschnitt in der Modul-Header Datei).

Eine Präprozessor-Direktive zum Definieren einer Konstante darf nur folgende Form besitzen (mit optionaler Klammerung):

```
#define <BEZEICHNER> [<TYPUMWANDLUNG>] <-><LITERAL>
```

Vordefinierte Konstante

Die SafeModuleHelper.h definiert Konstante für Minimal- und Maximalwerte der zulässigen Datentypen. Es ist zu beachten, dass der maximal negative Wert von INT32 (-2147483648) nicht direkt als Literal verwendet werden kann:

```
#define I8_MIN ((INT8) -128 )
#define I8_MAX ((INT8) 127 )
#define I16_MIN ((INT16) -32768 )
#define I16_MAX ((INT16) 32767 )
#define I32_MIN ( -2147483647-1 )
#define I32_MAX 2147483647
#define U8_MAX ((UINT8) 255U )
#define U16_MAX ((UINT16) 65535U )
#define U32_MAX 4294967295U
```

5.3.8 Funktionsaufrufe und benutzerdefinierte Funktionen

Ein TwinSafeGruppen-Modul gibt die Schnittstellen-Funktionen void Init(), void InputUpdate(), void OutputUpdate() und void CycleUpdate() vor. Diese dürfen nur von der TwinCAT Safety PLC Runtime aufgerufen werden.

Des Weiteren kann der Benutzer selbst Modul-Funktionen mit und ohne Rückgabewert deklarieren und definieren. Dafür gibt es vorgesehene Abschnitte in den .cpp/.h Dateien. Funktionen mit Rückgabewert dürfen und müssen lediglich ein finales „return“-Statement besitzen.

Selbstdefinierte Funktionen können aus den vier Schnittstellen-Funktionen heraus und aus den eigens definierten Funktionen heraus aufgerufen werden, sofern dadurch keine direkte/indirekte Rekursion entsteht. Des Weiteren stehen dem Benutzer Hilfsfunktionen über die SafeModuleHelper.h zur Verfügung.

Einschränkungen beim Aufruf von Funktionen

- Es wird unterschieden zwischen „pure Functions“ (ohne Seiteneffekte) und „impure Functions“ (mit möglichen Seiteneffekten).
- Alle vom Benutzer implementierten Funktionen werden grundsätzlich als „impure Functions“ betrachtet, da sie die Modulvariablen und Ausgänge potentiell verändern können.
- Als „pure Functions“ werden zunächst nur die von der SafeModuleHelper.h eingebundenen Funktionen betrachtet.
- Impure Functions können nur innerhalb einer Anweisung vom Typ-2 oder Typ-3 aufgerufen werden. Dabei darf eine Anweisung nicht mehr als einen Aufruf einer „impure Function“ haben.
- Pure Functions dürfen an beliebigen Stellen aufgerufen werden (Anweisungen und Bedingungsausdrücke)
- Der Rückgabewert einer Funktion mit Rückgabe muss immer verwendet werden, entweder durch Zuweisung oder als Parameter eines weiteren Funktionsaufrufes oder als Operand einer Operation.

5.3.9 Asserts und Traces

5.3.9.1 Asserts


FAILSAFE_ASSERT(<id>, <cond>)

Die Anweisung FAILSAFE_ASSERT() ist ein Mittel zur defensiven Programmierung für das Behandeln undefinierter Applikationszustände zur Laufzeit ohne Fallback-Strategie, für z.B. ungültige aber mögliche Eingaben. Ist eine Reaktion auf Applikationsebene möglich (z.B. durch setzen sicherer Standardwerte), sollte stattdessen eine Fallunterscheidung mit if-else/switch Kontrollstrukturen gewählt werden.

Ein FAILSAFE_ASSERT() sollte durch einen Fehlertest (Negativtest) auf Modultestebene auslösbar sein. Ist dies nicht der Fall, handelt es sich vermutlich um eine überflüssige Anweisung, da die Erkennung der fehlerhaften Ausführung des Benutzercodes bereits durch die sichere Laufzeitumgebung sichergestellt wird. Bei z.B. $c=a+b$; kann also davon ausgegangen werden, dass das Statement auch korrekt ausgeführt wird oder aber eine Inkonsistenz erkannt wird. Gleiches gilt für Kontrollstrukturen, Funktionsaufrufe usw.

Parameter	Beschreibung
<id>	Ein kurzer prägnanter C++ Identifier (wird als Klartext ausgegeben entweder in der Modultestausgabe oder per ADS-Nachricht im TwinCAT-Errorlist-Fenster)
<cond>	Boolescher Bedingungsausdruck für den die gleichen Einschränkungen gelten wie für if(), while() usw. Löst aus wenn <cond> FALSCH ist, d.h. <cond> muss im Gutfall halten!

Auslösung führt im nicht-sicheren Modultest zum Abbruch eines Testfalls mit Textausgabe. Wenn es sich um einen Fehlertestfall handelt, wird der Testfall als bestanden bewertet, sofern der Abbruch mit gegebener <id> und in gegebenem Testschritt erwartet wurde. Andernfalls gilt der Testfall bzw. Fehlertestfall als nicht bestanden.


 Hinweis	FAILSAFE_ASSERT() Die Anweisung FAILSAFE_ASSERT() setzt den sicheren Zustand der TwinSAFE Gruppe in der sicheren Laufzeitumgebung, wenn die Bedingung <cond> FALSCH liefert.
---	--

DEBUG_ASSERT(<id>, <cond>)

Die Anweisung DEBUG_ASSERT() ist ein Mittel zur Dokumentierung und Überprüfung interner Annahmen über den eigenen Programmcode (Vorbedingungen, Nachbedingungen, Invariante) während der Testphase. Beispiel: Test von Rückgabewerten oder Test von Parametern und Operanden VOR einem Funktionsaufruf oder einer Operation.

Parameter	Beschreibung
<id>	Ein kurzer prägnanter C++ Identifier (wird als Klartext ausgegeben entweder in der Modultestausgabe oder per ADS-Nachricht im TwinCAT-Errorlist-Fenster)
<cond>	Boolescher Bedingungsausdruck für den die gleichen Einschränkungen gelten wie für if(), while() usw. Löst aus wenn <cond> FALSCH ist, d.h. <cond> muss im Gutfall halten!


Auslösung führt im nicht-sicheren Modultest zum Abbruch eines Testfalls mit Textausgabe und mit dessen Bewertung als nicht bestanden.

 Hinweis	DEBUG_ASSERT() Die Anweisung DEBUG_ASSERT() führt in der sicheren Laufzeitumgebung zu einer Fehler- bzw. LogWindow-Meldung, wenn die Bedingung <cond> FALSCH liefert. Die Ausführung der sicherheitsgerichteten Applikation wird fortgesetzt!
---	---

TEST_ASSERT(<cond>)

Die Anweisung TEST_ASSERT() ist ein Mittel zur Überprüfung der Annahmen über die Ausgaben eines zu testenden Programmmoduls innerhalb eines Modultests. Dabei können sowohl konkrete Ergebnisse bewertet werden als auch allgemeine Annahmen definiert werden, um z.B. das Verhältnis von Ein- und Ausgängen sowie internen Modulvariablen zu überprüfen. Ein TEST_ASSERT() ist das Test-Pendant zum DEBUG_ASSERT() und sollte daher idealerweise von einer unabhängigen Person definiert werden.

Parameter	Beschreibung
<cond>	Boolescher Bedingungsausdruck für den die gleichen Einschränkungen gelten wie für if(), while() usw. Löst aus wenn <cond> FALSCH ist, d.h. <cond> muss im Gutfall halten!

 Hinweis	Verwendung von TEST_ASSERT() und DEBUG_ASSERT() Die Anweisung TEST_ASSERT() ist nur im Code der Modultestbench zulässig und führt zu einem Abbruch eines Testfalls (und dessen Bewertung als durchgefallen) , wenn die Bedingung <cond> FALSCH liefert. Die Ableitung von TEST_ASSERT() und DEBUG_ASSERT() Anweisung in Kombination mit Modultest und Testabdeckungsmessung sind ein wirkungsvolles Mittel um Implementierungsfehler und auch Spezifikationsfehler schon während des Entwurfs aufzudecken. Zudem ermöglichen generisch formulierte Assertions (Vorbedingungen, Nachbedingungen, Invariante) die Testabdeckung durch zusätzlich automatisch generierte Testfälle zu erhöhen (z.B. durch randomisierte Testdaten).
---	--

5.3.9.2 Traces

BRANCH_TRACE()

Der BRANCH_TRACE() ist für die Branch-Coverage-Messung (Zweigüberdeckungsmessung) in der Modultestumgebung notwendig, sofern dies nicht durch ein externes Werkzeug bewerkstelligt wird. Eine Branch-ID wird entsprechend der Dokumentenreihenfolge automatisch durchnummeriert. Die Ausgabe wird bei Auswahl eines entsprechenden Log-Levels durch Erreichen des Branches per ADS erzeugt. Diese Ausgabe erfolgt nur innerhalb der sicheren Laufzeitumgebung, jedoch nicht in der Modultestausgabe.

Ein BRANCH_TRACE(), wenn verwendet, muss am Ende eines Zweiges platziert werden. Bei Verwendung von return/break Statements direkt vor dem Statement.

Es wird eine Warnung generiert, wenn sie redundant gesetzt werden oder unvollständig sind. Diese Prüfung passiert, sobald ein BRANCH_TRACE() verwendet wird.

Die Abdeckung der Zweige wird am Ende der Modultestausführung in der Ausgabe angezeigt. Die ausgegebenen IDs nicht erreichter Zweige können über die Informationen in der ModuleDatabase.saxml im TwinSAFE-Gruppen-Ordner „Analysis Files“ den Quelltextzeilen zugeordnet werden. Eine Modultest-Zweigüberdeckung von 100% wird für sicherheitsgerichtete Applikationen gemeinhin als Minimalkriterium betrachtet! Mit dem Spezialkommentar /*<DefensiveBranch/>*/ können Zweige von der Testüberdeckungsmessung ausgenommen werden. Dies sollte aber nur dann erfolgen, wenn es sich

definitiv um nicht erreichbaren Code handelt, der begründet im Quelltext verbleiben soll! Zweige zum Abfangen fehlerhafter Eingaben sollten nicht dazu gehören, da sie durch einen Negativ-Test abgedeckt werden können.

DEBUG_TRACE(<expr>)

Die Anweisung DEBUG_TRACE() ist ein Mittel zur Testausgabe von lokalen Variablen und Zwischenergebnissen einfacher Datentypen, die nicht über das Prozessabbild ausgegeben werden können.

Die Log-Ausgabe erfolgt per ADS in TwinCAT bei Ausführung in der sicheren Laufzeitumgebung. Innerhalb des Modultests erfolgt die Ausgabe per einfacher Textausgabe.

Parameter	Beschreibung
<expr>	kann ein seiteneffekt-freier Ausdruck eines einfachen Datentypen sein, d.h. structs sind nicht erlaubt

5.4 Performance-Optimierungen

Aufgrund der Umsetzung notwendiger Sicherheitsmaßnahmen entsteht zur Ausführungszeit zusätzlicher Rechenaufwand. Um diesen zu minimieren, sollten folgende Codeoptimierungen beachtet werden.

Bedingungsaustrücke in Kontrollflussanweisungen

Komplexe Berechnungen in Kontrollflussanweisungen, insbesondere Funktionsaufrufe und reellwertige mathematische Funktionen (in V1 noch nicht unterstützt), sollten zunächst in einer Zeilenanweisung durchgeführt werden, um einer lokalen Variable zugewiesen zu werden. Das in der Variable gespeicherte Zwischenergebnis kann dann wiederum in die Bedingung einfließen, wie z.B. bei einer If-Else-Anweisung:

nicht optimiert	optimiert
<pre>if (SINF32(x) >= 0.0f) { ... } else { ... }...</pre>	<pre>FLOAT y = SINF32(x); if (y >= 0.0f) { ... } else { ... }</pre>

Bei Schleifenbedingungen sollten konstante Teilausdrücke, die aufwändige Teilberechnungen beinhalten können, ebenfalls als Zwischenergebnis in Zeilenanweisungen mit Zuweisung zu einer Variablen ausgelagert werden:

nicht optimiert	optimiert
<pre>#define _K_ 13U ... while (n < factorial(_K_ - 1U)) { ... n++; }</pre>	<pre>#define _K_ 13U ... UINT32 upper_limit = factorial(_K_ - 1U); while (n < upper_limit) { ... n++; }</pre>

Bei der Verwendung von Switch-Case-Konstrukten mit vielen Fällen sollte ebenfalls der Switch-Ausdruck ausgelagert werden. Folgendes Beispiel zeigt, in welchem Fall sich eine Optimierung (trotz reinem Integer-Ausdruck) lohnt:

nicht optimiert	optimiert
<pre> UINT32 w1; UINT32 w2; UINT32 w3; ... switch((((w1>>8U) & (w2>>16U)) (w3<<24U)) % 0xffU) { case 0x0U: ... break; case 0x1U: ... break; case 0x2U: ... break; ... case 0xfeU: ... break; default: ... break; } </pre>	<pre> UINT32 w1; UINT32 w2; UINT32 w3; ... UINT32 select = ((w1>>8U) & (w2>>16U)) (w3<<24U)) % 0xffU; switch(select) { case 0x0U: ... break; case 0x1U: ... break; case 0x2U: ... break; ... case 0xfeU: ... break; default: ... break; } </pre>

5.5 Anbindung an die E/A-Ebene

Interface zu Standard Ein- und Ausgängen

Standard Eingänge

```

/*! Struct providing input data of the corresponding standard alias devices
struct StandardInputs
{
  /*! ..\Alias Devices\ErrAck.sds
  struct _ErrAck
  {
    BOOL In;
  } ErrAck;
  /*! ..\Alias Devices\Run.sds
  struct _Run
  {
    BOOL In;
  } Run;
};

```

Standard Ausgänge

```

/*! Struct storing output data for the corresponding standard alias devices
struct StandardOutputs
{
  /*! ..\Alias Devices\DiscrepancyError.sds
  struct _DiscrepancyError
  {
    BOOL Out;
  } DiscrepancyError;
  /*! ..\Alias Devices\DiscrepancyCounter.sds
  struct _DiscrepancyCounter
  {
    UINT32 Out;
  } DiscrepancyCounter;
};

```

Interface zu sicheren Ein- und Ausgängen

Sichere Eingänge

```

    ///! Struct providing input data of the corresponding safety alias devices

    struct SafetyInputs
    {
        ///! ..\Alias Devices\EL1904_FSoE_211.sds
        struct _EL1904_FSoE_211
        {
            safeBOOL InputChannel1;
            safeBOOL InputChannel2;
            safeBOOL InputChannel3;
            safeBOOL InputChannel4;
        } EL1904_FSoE_211;
        ///! ..\Alias Devices\2 safe in 2 safe out.sds
        struct __2_safe_in_2_safe_out
        {
            safeBOOL InputChannel1;
            safeBOOL InputChannel2;
        } _2_safe_in_2_safe_out;
        ///! ..\Alias Devices\AX 5805 Drive Option.sds
        struct _AX_5805_Drive_Option
        {
            safeBOOL Axis_1_STO;
            safeBOOL Axis_1_SSM1;
            safeBOOL Axis_1_SSM2;
            safeBOOL Axis_1_SOS1;
            safeBOOL Axis_1_SSR1;
            safeBOOL Axis_1_SDIp;
            safeBOOL Axis_1_SDIin;
            safeBOOL Axis_1_Error_Ack;
        } AX_5805_Drive_Option;
    };

```

Sichere Ausgänge

```

    ///! Struct storing output data for the corresponding safety alias devices

    struct SafetyOutputs
    {
        ///! ..\Alias Devices\EL2904_FSoE_13.sds
        struct _EL2904_FSoE_13
        {
            safeBOOL OutputChannel1;
            safeBOOL OutputChannel2;
            safeBOOL OutputChannel3;
            safeBOOL OutputChannel4;
        } EL2904_FSoE_13;
        ///! ..\Alias Devices\EL2904_FSoE_4.sds
        struct _EL2904_FSoE_4
        {
            safeBOOL OutputChannel1;
            safeBOOL OutputChannel2;
            safeBOOL OutputChannel3;
            safeBOOL OutputChannel4;
        } EL2904_FSoE_4;
        ///! ..\Alias Devices\2 safe in 2 safe out.sds
        struct __2_safe_in_2_safe_out
        {
            safeBOOL OutputChannel1;
            safeBOOL OutputChannel2;
        } _2_safe_in_2_safe_out;
        ///! ..\Alias Devices\AX 5805 Drive Option.sds
        struct _AX_5805_Drive_Option
        {
            safeBOOL Axis_1_STO;
            safeBOOL Axis_1_SS11;
            safeBOOL Axis_1_SS21;
            safeBOOL Axis_1_SOS1;
            safeBOOL Axis_1_SSR1;
            safeBOOL Axis_1_SDIp;
            safeBOOL Axis_1_SDIin;
            safeBOOL Axis_1_Error_Ack;
        } AX_5805_Drive_Option;
    };

```


Interface zwischen TwinSAFE Gruppen

```

    //! Struct storing the TwinSAFE group exchange data
    struct TSGData
    {
        //! ..TwinSafeGroup: TwinSafeGroup1
        struct _TwinSafeGroup1
        {
            //! ..Outputs
            struct _Out
            {
                safeUINT AnalogOut1;
                safeBOOL EStopOut;
            } Out;
        } TwinSafeGroup1;
    };

```

Interface zum sicheren Zeitsignal

```
safeUINT16 u16SafeTimer    //!< Safe external timer input (in ms)
```

Über die Modul-Variable `u16SafeTimer` kann auf das sichere Zeitsignal zugegriffen werden. Dabei handelt es sich um einen 16 Bit Timer-Wert, welcher für zeitabhängige Funktionalitäten innerhalb der Sicherheitsapplikation verwendet werden kann. Innerhalb der Init-Funktion darf dieser Zeitwert nicht verwendet werden (da zu diesem Zeitpunkt noch nicht verfügbar). Der Timer eignet sich nur für Zeitmessungen über Applikationszyklen hinweg, da die Timer-Variable innerhalb eines Applikationszyklus konstant bleibt. Auf die Timer-Variable darf nicht schreibend zugegriffen werden.

5.6 Verifikation und Validierung



Anwendungsentwicklung in Safety C

Der Benutzer-Quelltext ist entsprechend der jeweils anzuwendenden Normen zu entwickeln, mit der Grundnorm IEC 61508:2010. Falls die anzuwendende Norm die IEC 61508:2010 ist, können die Erklärungen der Begriffe Verifikation und Validierung dem Teil 4 dieser Norm entnommen werden.

Verifikation

Bestätigen aufgrund einer Untersuchung und durch Bereitstellung eines Nachweises, dass die Anforderungen erfüllt worden sind.

ANMERKUNG

In Zusammenhang mit dieser Norm ist Verifikation die Tätigkeit, die in jeder Phase des relevanten Sicherheitslebenszyklus (Gesamt, E/E/PE-System und Software) durch Analyse, mathematische Schlussfolgerung und/oder Prüfung darlegt, dass für die speziellen Eingaben die Ergebnisse in jeder Hinsicht die Ziele und Anforderungen erfüllen, die für diese Phase festgelegt wurden.

BEISPIEL

Zu den Tätigkeiten der Verifikation gehören:

- die Überprüfungen der Ergebnisse (Dokumente aus allen Phasen des Sicherheitslebenszyklus), um unter Berücksichtigung der jeweiligen Eingaben der Phase die Übereinstimmung mit den Zielen und Anforderungen der Phase sicherzustellen;
- Entwurfsüberprüfungen;
- ausgeführte Prüfungen an entwickelten Produkten, um sicherzustellen, dass sie gemäß ihrer Spezifikation arbeiten;
- Ausführung von Integrationsprüfungen, wobei unterschiedliche Teile eines Systems Schritt für Schritt zusammengesetzt und Prüfungen unter Umgebungsbedingungen durchgeführt werden, um sicherzustellen, dass alle Teile in der spezifizierten Art zusammenarbeiten.

Validierung

Bestätigen aufgrund einer Untersuchung und durch Bereitstellung eines objektiven Nachweises, dass die besonderen Anforderungen für eine spezielle beabsichtigte Verwendung erfüllt worden sind.

ANMERKUNG 1

In dieser Norm gibt es drei Validierungsphasen:

- Validierung der Gesamtsicherheit (siehe IEC 61508-1, Bild 2);
- Validierung des E/E/PE-Systems (siehe IEC 61508-1, Bild 3);
- Validierung der Software (siehe IEC 61508-1, Bild 4).

ANMERKUNG 2

Die Validierung ist die Tätigkeit, die darlegt, dass das betrachtete sicherheitsbezogene System vor und nach der Installation in jeder Hinsicht der Spezifikation der Anforderungen an die Sicherheit des sicherheitsbezogenen Systems entspricht. Deshalb bedeutet zum Beispiel Validierung der Software die Bestätigung durch Untersuchung und Bereitstellung eines Nachweises, dass die Software die Spezifikation der Anforderungen an die Sicherheit der Software erfüllt.

5.7 Online-Diagnose

Modul-Tests

Der Entwickler kann die erstellte Safety Applikation über die ModuleTests.cpp im Standard C++ Modus testen. Dabei wird die Safety Applikation nicht im sicherheitsrelevanten Kontext, sondern direkt im Rahmen einer Standard C++-Umgebung übersetzt und ausgeführt. Eine Zuordnung zur Task mit der die TwinCAT Safety PLC im Release Modus ausgeführt wird, ist ebenfalls nicht gegeben.

Der Einstieg in den Modultest erfolgt über `MODULE_TEST_BENCH_DEF(<id>)`, in dem die Testüberdeckungsmessung gesteuert wird und die zu testenden Test-Gruppen über die Anweisung `MODULE_TEST_GROUP(<id>)` festgelegt werden. Jede Test-Gruppe benötigt eine eindeutige ID. Eine Test-Gruppe fasst definierte Testfälle (Test Cases) einer Gruppe über die Anweisung `MODULE_TEST_CASE(<id>)` zusammen, welche zuvor mittels `MODULE_TEST_CASE_DEF(<id>)` definiert wurden. Ein Testfall kann über die Anweisung `MODULE_TEST_STEP(<id>)` in weitere logische Testschritte unterteilt werden. Ein mit `MODULE_TEST_CASE_DEF(<id>)` definierter Testfall kann in eine Testgruppe als Fehlertest/Negativtestfall eingebunden werden, in dem er über die Anweisung `MODULE_FIT_CASE(<id>, step-id, <assert-id>)` aufgerufen wird. Hier wird erwartet, dass der Testfall in einem gegebenen Testschritt mit gegebener Assertion-ID fehlschlägt, um als bestanden bewertet zu werden. Dieser Mechanismus dient dem Test von FAILSAFE_ASSERT-Anweisungen z.B. durch unzulässige Eingabewerte.

Beim Anlegen einer TwinSAFE Gruppe wird bereits eine Modultestbench-Vorlage mit einem allgemeinen Testfall angelegt, welche auch direkt zum Debuggen verwendet werden kann. Dabei werden die Schnittstellenfunktionen eines TwinSAFE Gruppenmoduls zyklisch aufgerufen.

Das zu testende Modul (TwinSAFE-Gruppe) ist jeweils als Testinstanz innerhalb der Modultests bereits über die Variable DUT (Design Under Test) angelegt und verfügbar. Über `DUT.<Variablenname/Funktionsname>` sind alle (auch als *private* deklarierte) Modulvariablen und Modulfunktionen eines TwinSAFE Gruppenmoduls zugreifbar.

In dem Beispiel-Testfall wird einmal `DUT.Init();` aufgerufen und danach in einer For Schleife `DUT.Input-`, `DUT.Cycle-` und `DUT.OutputUpdate();` aufgerufen.

Der Anwendungsentwickler oder Tester kann vor dem Aufruf die internen Variablen setzen und nach dem Aufruf über eine `TEST_ASSERT`-Anweisung das Ergebnis der Berechnung bewerten.

Der ModuleTest wird über das Kontextmenü unter Test Files übersetzt (Build) und im Debug Modus gestartet. Über Run/Analyze soll die Darstellung der Testergebnisse im Safety Editor erfolgen (wird in V1 nicht unterstützt).

Zudem kann der Benutzer das entsprechende Visual Studio Projekt `ModuleTests.vcxproj` im Ordner `TwinSAFE-Gruppenordner „Test Files“` direkt öffnen, erweitern und ausführen. Für den Quellcode der Modultests gibt es keine Beschränkungen bzgl. Codierregeln oder der Einbindung von Standardbibliotheken.

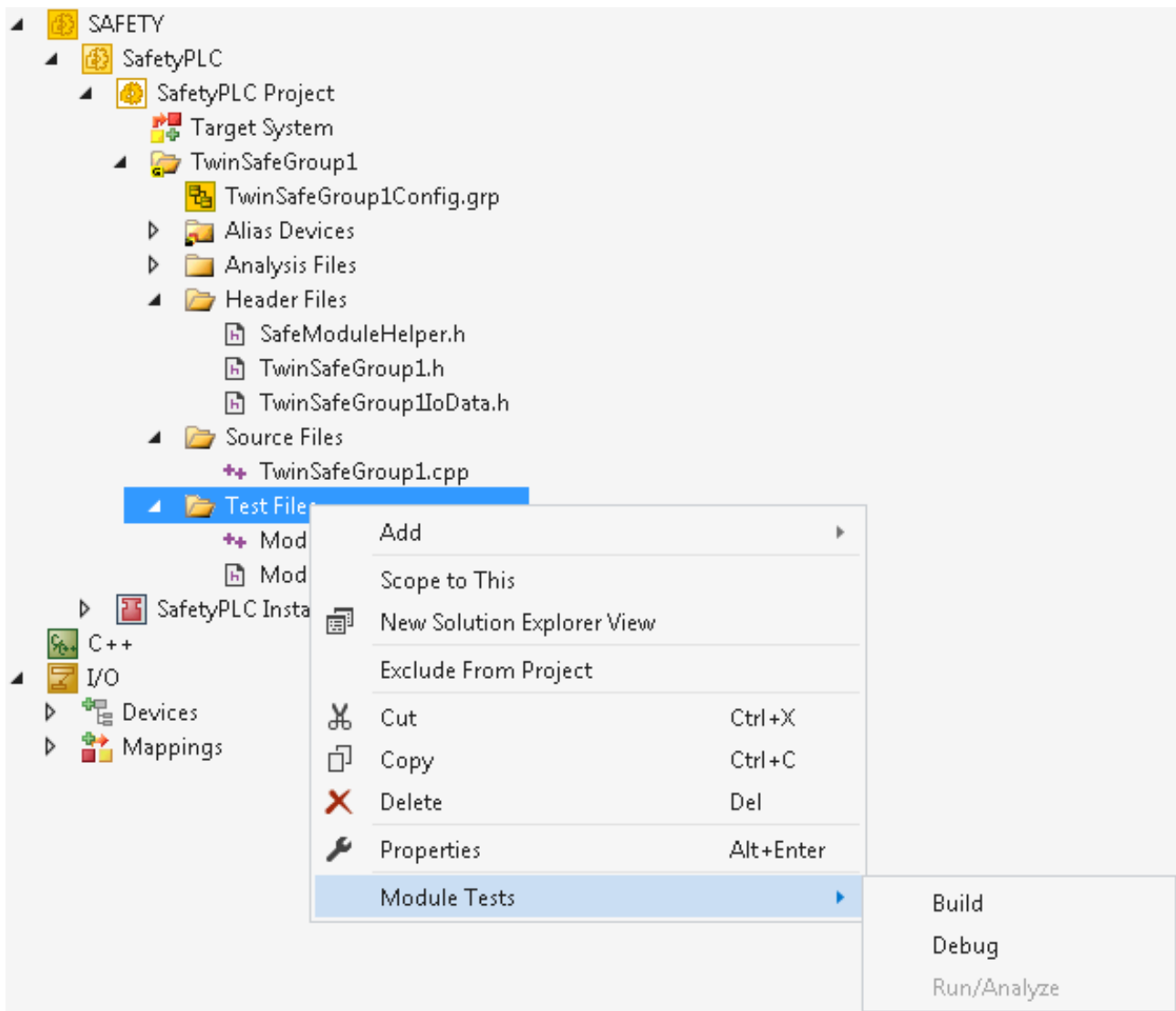


Abb. 69: Kontextmenu ModuleTests

Im Debug-Modus können die üblichen Visual Studio Mechanismen, wie Breakpoints, Step Into, Step Over, usw. aus dem TwinCAT Safety Editor heraus verwendet werden.

Die Variablenwerte können über das Locals Fenster oder über Data Tips im Visual Studio online überwacht werden. Die Ausgabe erfolgt in das Fenster Output (Debug).

Beispiel**Auszug aus TwinSafeGroup1.h**

```
// Module internals
PRIVATE:

/*<UserDefinedVariables>*/      // Define internal variables here
    INT32 a;
    INT32 b;
    INT32 z;
    BOOL neg;
/*</UserDefinedVariables>*/
```

Auszug aus TwinSafeGroup1.cpp

```
////////////////////////////////////
//! \brief Implementation of the safe user module initialization function
////////////////////////////////////
/*<TcInit>*/
VOID CSafeModule::Init()
{
    // Put your module initialization code here
    a = 0;
    b = 1;
    z = 0;
    neg = false;
    BRANCH_TRACE();
}
/*</TcInit>*/

////////////////////////////////////
//! \brief Implementation of the safe user module input update function
////////////////////////////////////
/*<TcInputUpdate>*/
VOID CSafeModule::InputUpdate()
{
    // Put your module input update code here
    BRANCH_TRACE();
}
/*</TcInputUpdate>*/

////////////////////////////////////
//! \brief Implementation of the safe user module output update function
////////////////////////////////////
/*<TcOutputUpdate>*/
VOID CSafeModule::OutputUpdate()
{
    // Put your module output update code here
    BRANCH_TRACE();
}
/*</TcOutputUpdate>*/

////////////////////////////////////
//! \brief Implementation of the safe user module cycle update function
////////////////////////////////////
/*<TcCycleUpdate>*/
VOID CSafeModule::CycleUpdate()
{
    // Put your cycle update code here
    FAILSAFE_ASSERT(DIV_BY_ZERO, b != 0);
    z = a / b;
    if (z >= 0)
    {
        neg = false;
        BRANCH_TRACE();
    }
    else
    {
        neg = true;
        BRANCH_TRACE();
    }
    BRANCH_TRACE();
}
/*</TcCycleUpdate>*/
```

ModuleTests.cpp

```

////////////////////////////////////
/// \file      ModuleTests.cpp
/// \brief     Source file with module test definitions for TwinSafeGroup1
/// \authors   User01
/// \copyright Put affiliation and copyright notice here
/// \version   V1.0
/// \date      2016-10-20
/// \details   Put detailed description of your module tests here
////////////////////////////////////

///! Define name of the safe module under test
#define MODULE_NAME TwinSafeGroup1::CSafeModule

#include "TwinSafeGroup1.h"
#include "ModuleTests.h"

///! Definition of test case IDs
#define TC_ID_0      0
#define TC_ID_1      1

////////////////////////////////////
///! \brief Test bench definition containing testsets triggered by TwinCAT3
////////////////////////////////////
MODULE_TEST_BENCH_DEF()
{
    // Reset branch counters for coverage measurement
    START_COVERAGE_MEASUREMENT();

    // Run test group TG_ID_0
    MODULE_TEST_GROUP(TG_ID_0);

    // Compute branch coverage and identify uncovered branches
    STOP_COVERAGE_MEASUREMENT();
}

////////////////////////////////////
///! \brief TC_ID_0 (put a reference to your test specification here)
///! \test   Generic module test sequence calling init and 1000 task cycles
////////////////////////////////////
MODULE_TEST_CASE_DEF(TC_ID_0)
{
    // Test case starts with an initial test step to prepare preconditions
    MODULE_TEST_STEP(0);
    DUT.Init(); // e.g., call Init() to set state variables to default values

    // Perform post initialization checks on module state variables here,
    // e.g., using TEST_ASSERT(<condition>) statements

    for (int nCycle = 1; nCycle <= 1000; nCycle++)
    { // Execute a test sequence consisting of 1000 module execution cycles

        MODULE_TEST_STEP(nCycle);
        // Apply test step stimuli to safe and non-safe module inputs here
        DUT.ul6SafeTimer = (nCycle * 5) % 65536U; // e.g. 5ms task period

        DUT.a = 2*cycle;
        DUT.b = cycle;

        // Perform a single cycle of a periodic task execution
        DUT.InputUpdate();
        DUT.CycleUpdate();
        DUT.OutputUpdate();

        // Perform invariant checks on safe and non-safe module outputs and
        // also state variables here, e.g., using TEST_ASSERT(<condition>)

        // Perform checks on test step response w.r.t. test step stimuli
        // here, e.g., using TEST_ASSERT(<condition>) statements
        TEST_ASSERT(DUT.z == 2); // As (2*cycle)/cycle is always 2

    }

    // Perform checks on the final module state w.r.t. test specification
    // here, e.g., using TEST_ASSERT(<condition>) statements
}

```

```

////////////////////////////////////
/// \brief   TC_ID_1 (put a reference to your test specification here)
/// \test    Negative test case for invalid input b=0 (division by zero)
////////////////////////////////////
MODULE_TEST_CASE_DEF(TC_ID_1)
{
    // Test case starts with an initial test step to prepare preconditions
    MODULE_TEST_STEP(0);
    DUT.Init(); // e.g., call Init() to set state variables to default values

    // Perform post initialization checks on module state variables here,
    // e.g., using TEST_ASSERT(<condition>) statements

    // Apply test step stimuli to safe and non-safe module inputs here
    MODULE_TEST_STEP(1);
    DUT.a = 1;
    DUT.b = 0;

    // Perform a single execution cycle
    DUT.InputUpdate();
    DUT.CycleUpdate();
    DUT.OutputUpdate();

    // Should not reach here as CycleUpdate is expected to trigger fail safe!
    TEST_ASSERT(false);
}

////////////////////////////////////
/// \brief Test group TG_ID_0 definition containing a set of test cases
////////////////////////////////////
MODULE_TEST_GROUP_DEF(TG_ID_0)
{
    // Run positive example test case TC_ID_0
    MODULE_TEST_CASE_RUN(TC_ID_0);
    // Run negative example test case TC_ID_1 expecting fail-safe
    // assertion DIV_BY_ZERO being triggered at test step 1!
    MODULE_FIT_CASE_RUN(TC_ID_1, 1 /* Step */, DIV_BY_ZERO /* ID */);
}

```

5.8 Sichere Hilfsfunktionen

Die sicheren Hilfsfunktionen bieten dem Benutzer sichere Erweiterungen des eingeschränkten Safety C Sprachumfangs an, welcher einer Teilmenge des nativen Sprachumfangs von C/C++ entspricht und somit keine Einbringung von nicht-sicheren Standardbibliotheken erlaubt.

Hilfsfunktionen sind frei von Seiteneffekten, sodass sie in allen Safety C Ausdrücken und Anweisungen ohne Einschränkungen verwendet werden können, d.h. der Rückgabewert einer Hilfsfunktion hängt nur von den eigenen Funktionsparametern ab. Des Weiteren ändert keine Hilfsfunktion direkt die Moduldaten eines sicheren Applikationsmoduls oder andere globale Applikationsdaten.

Hilfsfunktionen reagieren im Falle undefinierter Eingaben mit der Einnahme des sicheren Zustands der jeweiligen TwinSAFE Gruppe, in dessen Applikationsausführung sie eingebunden sind. Undefinierte Eingaben sind solche Eingaben, für die eine Hilfsfunktion keine gültige Ausgabe erzeugen kann (im Weiteren als *undef.* bezeichnet). Da es aber aufgrund der Funktionssignaturen möglich ist, dass solche Eingaben als Parameter an Hilfsfunktionen übergeben werden, werden derartige Eingaben intern mit einer **FAILSAFE_ASSERT**-Anweisung abgefangen und führen programmatisch den sicheren Fehlerzustand der TwinSAFE Gruppe herbei. Dieser Fall wird über eine entsprechende Log-Nachricht für den Anwender sichtbar.

5.8.1 Sichere Logikfunktionen

Die sicheren Logikfunktionen werten alle booleschen Operanden aus, d.h., es wird keine Kurzschlussauswertung angewendet, anders als es bei den nativen C/C++ Operatoren `&&` und `||` der Fall ist.

5.8.1.1 AND

Führt ein sicheres logisches Und für zwei boolesche Ausdrücke durch.

Safety C Funktionsschnittstelle

BOOL AND(BOOL xa, BOOL xb)
 safeBOOL AND(safeBOOL xa, safeBOOL xb)

Funktionsspezifikation

$y = \text{AND}(x_a, x_b): f(x_a, x_b) \rightarrow y$

$x_a \in \{false, true\}$
 $x_b \in \{false, true\}$
 $y \in \{false, true\}$
 $f(x_a, x_b) = x_a \wedge x_b$

5.8.1.2 AND3

Führt ein sicheres logisches Und für drei boolesche Ausdrücke durch.

Safety C Funktionsschnittstelle

BOOL AND3(BOOL xa, BOOL xb, BOOL xc)
 safeBOOL AND3(safeBOOL xa, safeBOOL xb, safeBOOL xc)

Funktionsspezifikation

$y = \text{AND3}(x_a, x_b, x_c): f(x_a, x_b, x_c) \rightarrow y$

$x_a \in \{false, true\}$
 $x_b \in \{false, true\}$
 $x_c \in \{false, true\}$
 $y \in \{false, true\}$
 $f(x_a, x_b, x_c) = x_a \wedge x_b \wedge x_c$

5.8.1.3 AND4

Führt ein sicheres logisches Und für vier boolesche Ausdrücke durch.

Safety C Funktionsschnittstelle

BOOL AND4(BOOL xa, BOOL xb, BOOL xc, BOOL xd)
 safeBOOL AND4(safeBOOL xa, safeBOOL xb, safeBOOL xc, safeBOOL xd)

Funktionsspezifikation

$y = \text{AND4}(x_a, x_b, x_c, x_d): f(x_a, x_b, x_c, x_d) \rightarrow y$

$x_a \in \{false, true\}$
 $x_b \in \{false, true\}$
 $x_c \in \{false, true\}$
 $x_d \in \{false, true\}$
 $y \in \{false, true\}$
 $f(x_a, x_b, x_c, x_d) = x_a \wedge x_b \wedge x_c \wedge x_d$

5.8.1.4 OR

Führt ein sicheres logisches Oder für zwei boolesche Ausdrücke durch.

Safety C Funktionsschnittstelle

```
BOOL      OR(BOOL xa,      BOOL xb)
safeBOOL  OR(safeBOOL xa, safeBOOL xb)
```

Funktionspezifikation

$y = \text{OR}(x_a, x_b): f(x_a, x_b) \rightarrow y$

$x_a \in \{false, true\}$
 $x_b \in \{false, true\}$
 $y \in \{false, true\}$
 $f(x_a, x_b) = x_a \vee x_b$

5.8.1.5 OR3

Führt ein sicheres logisches Oder für drei boolesche Ausdrücke durch.

Safety C Funktionsschnittstelle

```
BOOL      OR3(BOOL xa,      BOOL xb,      BOOL xc)
safeBOOL  OR3(safeBOOL xa, safeBOOL xb, safeBOOL xc)
```

Funktionspezifikation

$y = \text{OR3}(x_a, x_b, x_c): f(x_a, x_b, x_c) \rightarrow y$

$x_a \in \{false, true\}$
 $x_b \in \{false, true\}$
 $x_c \in \{false, true\}$
 $y \in \{false, true\}$
 $f(x_a, x_b, x_c) = x_a \vee x_b \vee x_c$

5.8.1.6 OR4

Führt ein sicheres logisches Oder für vier boolesche Ausdrücke durch.

Safety C Funktionsschnittstelle

```
BOOL      OR4(BOOL xa,      BOOL xb,      BOOL xc,      BOOL xd)
safeBOOL  OR4(safeBOOL xa, safeBOOL xb, safeBOOL xc, safeBOOL xd)
```

Funktionspezifikation

$y = \text{OR4}(x_a, x_b, x_c, x_d): f(x_a, x_b, x_c, x_d) \rightarrow y$

$x_a \in \{false, true\}$
 $x_b \in \{false, true\}$
 $x_c \in \{false, true\}$
 $x_d \in \{false, true\}$
 $y \in \{false, true\}$
 $f(x_a, x_b, x_c, x_d) = x_a \vee x_b \vee x_c \vee x_d$

5.8.2 Sichere Ganzzahlarithmetik-Funktionen

Die sicheren Arithmetikfunktionen für Ganzzahldatentypen detektieren undefiniertes Verhalten, welches bei entsprechenden C/C++ Operatoren und Standardfunktionen durch Überläufe des vorzeichenbehafteten 32 Bit-Ganzzahltypen sowie bei Modulo/Division durch Null auftreten kann. Bei Eingabe ungültiger Werte wird der sichere Gruppenzustand eingenommen.

HINWEIS: Eine UINT16-Multiplikation mit dem C/C++ Operator „*“ kann zu einem undefinierten Überlauf im resultierenden INT32-Ausdruck bei einem Ergebnis größer $2^{31}-1$ führen. In V1 wird dafür keine Hilfsfunktion bereitgestellt.

5.8.2.1 ADDI32

Führt eine sichere Addition für den vorzeichenbehafteten 32-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```
INT32      ADDI32(INT32 xa,      INT32 xb)
DINT       ADDI32(DINT xa,      DINT xb)
safeINT32  ADDI32(safeINT32 xa, safeINT32 xb)
safeDINT   ADDI32(safeDINT xa,  safeDINT xb)
```

Funktionsspezifikation

$y = \text{ADDI32}(x_a, x_b): f(x_a, x_b) \rightarrow y$

$x_a \in \mathbb{Z},$ $-2^{31} \leq x_a \leq 2^{31} - 1$
 $x_b \in \mathbb{Z},$ $-2^{31} \leq x_b \leq 2^{31} - 1$
 $y \in \mathbb{Z},$ $-2^{31} \leq y \leq 2^{31} - 1$

$$f(x_a, x_b) = \begin{cases} x_a + x_b, & -2^{31} \leq x_a + x_b \leq 2^{31} - 1 \\ \text{undef.}, & (x_a + x_b < -2^{31}) \vee (2^{31} \leq x_a + x_b) \end{cases}$$

5.8.2.2 SUBI32

Führt eine sichere Subtraktion für den vorzeichenbehafteten 32-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```
INT32      SUBI32(INT32 xa,      INT32 xb)
DINT       SUBI32(DINT xa,      DINT xb)
safeINT32  SUBI32(safeINT32 xa, safeINT32 xb)
safeDINT   SUBI32(safeDINT xa,  safeDINT xb)
```

Funktionsspezifikation

$y = \text{SUBI32}(x_a, x_b): f(x_a, x_b) \rightarrow y$

$x_a \in \mathbb{Z},$ $-2^{31} \leq x_a \leq 2^{31} - 1$
 $x_b \in \mathbb{Z},$ $-2^{31} \leq x_b \leq 2^{31} - 1$
 $y \in \mathbb{Z},$ $-2^{31} \leq y \leq 2^{31} - 1$

$$f(x_a, x_b) = \begin{cases} x_a - x_b, & -2^{31} \leq x_a - x_b \leq 2^{31} - 1 \\ \text{undef.}, & (x_a - x_b < -2^{31}) \vee (2^{31} \leq x_a - x_b) \end{cases}$$

5.8.2.3 MULI32

Führt eine sichere Multiplikation für den vorzeichenbehafteten 32-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

INT32	MULI32(INT32 xa,	INT32 xb)
DINT	MULI32(DINT xa,	DINT xb)
safeINT32	MULI32(safeINT32 xa,	safeINT32 xb)
safeDINT	MULI32(safeDINT xa,	safeDINT xb)

Funktionsspezifikation

$$y = \text{MULI32}(x_a, x_b): f(x_a, x_b) \rightarrow y$$

$x_a \in \mathbb{Z},$	$-2^{31} \leq x_a \leq 2^{31} - 1$
$x_b \in \mathbb{Z},$	$-2^{31} \leq x_b \leq 2^{31} - 1$
$y \in \mathbb{Z},$	$-2^{31} \leq y \leq 2^{31} - 1$

$$f(x_a, x_b) = \begin{cases} x_a \cdot x_b, & -2^{31} \leq x_a \cdot x_b \leq 2^{31} - 1 \\ \text{undef.}, & (x_a \cdot x_b < -2^{31}) \vee (2^{31} \leq x_a \cdot x_b) \end{cases}$$
5.8.2.4 DIVI32

Führt eine sichere Division für den vorzeichenbehafteten 32-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

INT32	DIVI32(INT32 xa,	INT32 xb)
DINT	DIVI32(DINT xa,	DINT xb)
safeINT32	DIVI32(safeINT32 xa,	safeINT32 xb)
safeDINT	DIVI32(safeDINT xa,	safeDINT xb)

Funktionsspezifikation

$$y = \text{DIVI32}(x_a, x_b): f(x_a, x_b) \rightarrow y$$

$x_a \in \mathbb{Z},$	$-2^{31} \leq x_a \leq 2^{31} - 1$
$x_b \in \mathbb{Z},$	$-2^{31} \leq x_b \leq 2^{31} - 1$
$y \in \mathbb{Z},$	$-2^{31} \leq y \leq 2^{31} - 1$

$$f(x_a, x_b) = \begin{cases} \lfloor \frac{x_a}{x_b} \rfloor, & (-2^{31} < x_b < 0) \wedge (0 < x_b) \\ \text{undef.}, & (x_b = -2^{31}) \vee (x_b = 0) \end{cases}$$
5.8.2.5 DIVU32

Führt eine sichere Division für den vorzeichenlosen 32-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

UINT32	DIVU32(UINT32 xa,	UINT32 xb)
UDINT	DIVU32(UDINT xa,	UDINT xb)
safeUINT32	DIVU32(safeUINT32 xa,	safeUINT32 xb)
safeUDINT	DIVU32(safeUDINT xa,	safeUDINT xb)

Funktionsspezifikation

$$y = \text{DIVU32}(x_a, x_b): f(x_a, x_b) \rightarrow y$$

$$\begin{aligned} x_a \in \mathbb{Z}, & & 0 \leq x_a \leq 2^{32} - 1 \\ x_b \in \mathbb{Z}, & & 0 \leq x_b \leq 2^{32} - 1 \\ y \in \mathbb{Z}, & & 0 \leq y \leq 2^{32} - 1 \end{aligned}$$

$$f(x_a, x_b) = \begin{cases} \lfloor \frac{x_a}{x_b} \rfloor, & 0 < x_b \\ \text{undef.}, & x_b = 0 \end{cases}$$

5.8.2.6 MODI32

Führt eine sichere Restwertberechnung für den vorzeichenbehafteten 32-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```
INT32      MODI32(INT32 xa,      INT32 xb)
DINT      MODI32(DINT xa,      DINT xb)
safeINT32 MODI32(safeINT32 xa, safeINT32 xb)
safeDINT  MODI32(safeDINT xa,  safeDINT xb)
```

Funktionsspezifikation

$$y = \text{MODI32}(x_a, x_b): f(x_a, x_b) \rightarrow y$$

$$\begin{aligned} x_a \in \mathbb{Z}, & & -2^{31} \leq x_a \leq 2^{31} - 1 \\ x_b \in \mathbb{Z}, & & -2^{31} \leq x_b \leq 2^{31} - 1 \\ y \in \mathbb{Z}, & & 0 \leq y \leq 2^{31} - 2 \end{aligned}$$

$$f(x_a, x_b) = \begin{cases} x_a - \lfloor \frac{x_a}{x_b} \rfloor \cdot x_b, & (0 \leq x_a) \wedge (0 < x_b) \\ \text{undef.}, & (x_a < 0) \vee (x_b \leq 0) \end{cases}$$

5.8.2.7 MODU32

Führt eine sichere Restwertberechnung für den vorzeichenlosen 32-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```
UINT32     MODU32(UINT32 xa,     UINT32 xb)
UDINT     MODU32(UDINT xa,     UDINT xb)
safeUINT32 MODU32(safeUINT32 xa, safeUINT32 xb)
safeUDINT  MODU32(safeUDINT xa,  safeUDINT xb)
```

Funktionsspezifikation

$$y = \text{MODU32}(x_a, x_b): f(x_a, x_b) \rightarrow y$$

$$\begin{aligned} x_a \in \mathbb{Z}, & & 0 \leq x_a \leq 2^{32} - 1 \\ x_b \in \mathbb{Z}, & & 0 \leq x_b \leq 2^{32} - 1 \\ y \in \mathbb{Z}, & & 0 \leq y \leq 2^{32} - 2 \end{aligned}$$

$$f(x_a, x_b) = \begin{cases} x_a - \lfloor \frac{x_a}{x_b} \rfloor \cdot x_b, & 0 < x_b \\ \text{undef.}, & x_b = 0 \end{cases}$$

5.8.2.8 DIVI16

Führt eine sichere Division für den vorzeichenbehafteten 16-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```

INT16      DIVI16(INT16 xa,      INT16 xb)
INT        DIVI16(INT xa,      INT xb)
safeINT16  DIVI16(safeINT16 xa, safeINT16 xb)
safeINT    DIVI16(safeINT xa,   safeINT xb)

```

Funktionsspezifikation

$y = \text{DIVI16}(x_a, x_b): f(x_a, x_b) \rightarrow y$

$x_a \in \mathbb{Z},$ $-2^{15} \leq x_a \leq 2^{15} - 1$
 $x_b \in \mathbb{Z},$ $-2^{15} \leq x_b \leq 2^{15} - 1$
 $y \in \mathbb{Z},$ $-2^{15} \leq y \leq 2^{15} - 1$

$$f(x_a, x_b) = \begin{cases} \lfloor \frac{x_a}{x_b} \rfloor, & (-2^{15} < x_b < 0) \wedge (0 < x_b) \\ \text{undef.}, & (x_b = -2^{15}) \vee (x_b = 0) \end{cases}$$

5.8.2.9 DIVU16

Führt eine sichere Division für den vorzeichenlosen 16-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```

UINT16     DIVU16(UINT16 xa,    UINT16 xb)
UINT       DIVU16(UINT xa,     UINT xb)
safeUINT16 DIVU16(safeUINT16 xa, safeUINT16 xb)
safeUINT   DIVU16(safeUINT xa,  safeUINT xb)

```

Funktionsspezifikation

$y = \text{DIVU16}(x_a, x_b): f(x_a, x_b) \rightarrow y$

$x_a \in \mathbb{Z},$ $0 \leq x_a \leq 2^{16} - 1$
 $x_b \in \mathbb{Z},$ $0 \leq x_b \leq 2^{16} - 1$
 $y \in \mathbb{Z},$ $0 \leq y \leq 2^{16} - 1$

$$f(x_a, x_b) = \begin{cases} \lfloor \frac{x_a}{x_b} \rfloor, & 0 < x_b \\ \text{undef.}, & x_b = 0 \end{cases}$$

5.8.2.10 MODI16

Führt eine sichere Restwertberechnung für den vorzeichenbehafteten 16-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```

INT16      MODI16(INT16 xa,      INT16 xb)
INT        MODI16(INT xa,      INT xb)
safeINT16  MODI16(safeINT16 xa, safeINT16 xb)
safeINT    MODI16(safeINT xa,   safeINT xb)

```

Funktionsspezifikation

$$y = \text{MODI16}(x_a, x_b): f(x_a, x_b) \rightarrow y$$

$$\begin{aligned} x_a \in \mathbb{Z}, & & -2^{15} \leq x_a \leq 2^{15} - 1 \\ x_b \in \mathbb{Z}, & & -2^{15} \leq x_b \leq 2^{15} - 1 \\ y \in \mathbb{Z}, & & 0 \leq y \leq 2^{15} - 2 \end{aligned}$$

$$f(x_a, x_b) = \begin{cases} x_a - \lfloor \frac{x_a}{x_b} \rfloor \cdot x_b, & (0 \leq x_a) \wedge (0 < x_b) \\ \text{undef.}, & (x_a < 0) \vee (x_b \leq 0) \end{cases}$$

5.8.2.11 MODU16

Führt eine sichere Restwertberechnung für den vorzeichenlosen 16-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```

UINT16      MODU16(UINT16 xa,      UINT16 xb)
UINT        MODU16(UINT xa,       UINT xb)
safeUINT16  MODU16(safeUINT16 xa, safeUINT16 xb)
safeUINT    MODU16(safeUINT xa,   safeUINT xb)
    
```

Funktionsspezifikation

$$y = \text{MODU16}(x_a, x_b): f(x_a, x_b) \rightarrow y$$

$$\begin{aligned} x_a \in \mathbb{Z}, & & 0 \leq x_a \leq 2^{16} - 1 \\ x_b \in \mathbb{Z}, & & 0 \leq x_b \leq 2^{16} - 1 \\ y \in \mathbb{Z}, & & 0 \leq y \leq 2^{16} - 2 \end{aligned}$$

$$f(x_a, x_b) = \begin{cases} x_a - \lfloor \frac{x_a}{x_b} \rfloor \cdot x_b, & 0 < x_b \\ \text{undef.}, & x_b = 0 \end{cases}$$

5.8.2.12 DIVI8

Führt eine sichere Division für den vorzeichenbehafteten 8-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```

INT8      DIVI8(INT8 xa,      INT8 xb)
SINT      DIVI8(SINT xa,     SINT xb)
safeINT8  DIVI8(safeINT8 xa, safeINT8 xb)
safeSINT  DIVI8(safeSINT xa, safeSINT xb)
    
```

Funktionsspezifikation

$$y = \text{DIVI8}(x_a, x_b): f(x_a, x_b) \rightarrow y$$

$$\begin{aligned} x_a \in \mathbb{Z}, & & -2^7 \leq x_a \leq 2^7 - 1 \\ x_b \in \mathbb{Z}, & & -2^7 \leq x_b \leq 2^7 - 1 \\ y \in \mathbb{Z}, & & -2^7 \leq y \leq 2^7 - 1 \end{aligned}$$

$$f(x_a, x_b) = \begin{cases} \lfloor \frac{x_a}{x_b} \rfloor, & (-2^7 < x_b < 0) \wedge (0 < x_b) \\ \text{undef.}, & (x_b = -2^7) \vee (x_b = 0) \end{cases}$$

5.8.2.13 DIVU8

Führt eine sichere Division für den vorzeichenlosen 8-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```

UINT8      DIVU8(UINT8 xa,      UINT8 xb)
USINT      DIVU8(USINT xa,     USINT xb)
safeUINT8  DIVU8(safeUINT8 xa, safeUINT8 xb)
safeUSINT  DIVU8(safeUSINT xa, safeUSINT xb)

```

Funktionsspezifikation

$y = \text{DIVU8}(x_a, x_b): f(x_a, x_b) \rightarrow y$

$x_a \in \mathbb{Z}, \quad 0 \leq x_a \leq 2^8 - 1$
 $x_b \in \mathbb{Z}, \quad 0 \leq x_b \leq 2^8 - 1$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 2^8 - 1$

$$f(x_a, x_b) = \begin{cases} \lfloor \frac{x_a}{x_b} \rfloor, & 0 < x_b \\ \text{undef.}, & x_b = 0 \end{cases}$$

5.8.2.14 MODI8

Führt eine sichere Restwertberechnung für den vorzeichenbehafteten 8-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```

INT8       MODI8(INT8 xa,      INT8 xb)
SINT       MODI8(SINT xa,     SINT xb)
safeINT8   MODI8(safeINT8 xa, safeINT8 xb)
safeSINT   MODI8(safeSINT xa, safeSINT xb)

```

Funktionsspezifikation

$y = \text{MODI8}(x_a, x_b): f(x_a, x_b) \rightarrow y$

$x_a \in \mathbb{Z}, \quad -2^7 \leq x_a \leq 2^7 - 1$
 $x_b \in \mathbb{Z}, \quad -2^7 \leq x_b \leq 2^7 - 1$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 2^7 - 2$

$$f(x_a, x_b) = \begin{cases} x_a - \lfloor \frac{x_a}{x_b} \rfloor \cdot x_b, & (0 \leq x_a) \wedge (0 < x_b) \\ \text{undef.}, & (x_a < 0) \vee (x_b \leq 0) \end{cases}$$

5.8.2.15 MODU8

Führt eine sichere Restwertberechnung für den vorzeichenlosen 8-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```

UINT8      MODU8(UINT8 xa,      UINT8 xb)
USINT      MODU8(USINT xa,     USINT xb)
safeUINT8  MODU8(safeUINT8 xa, safeUINT8 xb)
safeUSINT  MODU8(safeUSINT xa, safeUSINT xb)

```

Funktionsspezifikation

$$y = \text{MODU8}(x_a, x_b): f(x_a, x_b) \rightarrow y$$

$$\begin{aligned} x_a &\in \mathbb{Z}, & 0 \leq x_a \leq 2^8 - 1 \\ x_b &\in \mathbb{Z}, & 0 \leq x_b \leq 2^8 - 1 \\ y &\in \mathbb{Z}, & 0 \leq y \leq 2^8 - 2 \end{aligned}$$

$$f(x_a, x_b) = \begin{cases} x_a - \lfloor \frac{x_a}{x_b} \rfloor \cdot x_b, & 0 < x_b \\ \text{undef.}, & x_b = 0 \end{cases}$$

5.8.2.16 NEGI32

Führt eine sichere arithmetische Negation eines vorzeichenbehafteten 32-Bit-Ganzzahltyps durch.

Safety C Funktionsschnittstelle

INT32 NEGI32 (INT32 x)
 DINT NEGI32 (DINT x)
 safeINT32 NEGI32 (safeINT32 x)
 safeDINT NEGI32 (safeDINT x)

Funktionsspezifikation

$$y = \text{NEGI32}(x): f(x) \rightarrow y$$

$$\begin{aligned} x &\in \mathbb{Z}, & -2^{31} \leq x \leq 2^{31} - 1 \\ y &\in \mathbb{Z}, & -2^{31} + 1 \leq y \leq 2^{31} - 1 \end{aligned}$$

$$f(x) = \begin{cases} -x, & -2^{31} + 1 \leq x \leq 2^{31} - 1 \\ \text{undef.}, & x = -2^{31} \end{cases}$$

5.8.2.17 NEGI16

Führt eine sichere arithmetische Negation eines vorzeichenbehafteten 16-Bit-Ganzzahltyps durch.

Safety C Funktionsschnittstelle

INT16 NEGI16 (INT16 x)
 INT NEGI16 (INT x)
 safeINT16 NEGI16 (safeINT16 x)
 safeINT NEGI16 (safeINT x)

Funktionsspezifikation

$$y = \text{NEGI16}(x): f(x) \rightarrow y$$

$$\begin{aligned} x &\in \mathbb{Z}, & -2^{15} \leq x \leq 2^{15} - 1 \\ y &\in \mathbb{Z}, & -2^{15} + 1 \leq y \leq 2^{15} - 1 \end{aligned}$$

$$f(x) = \begin{cases} -x, & -2^{15} + 1 \leq x \leq 2^{15} - 1 \\ \text{undef.}, & x = -2^{15} \end{cases}$$

5.8.2.18 NEGI8

Führt eine sichere arithmetische Negation eines vorzeichenbehafteten 8-Bit-Ganzzahltyps durch.

Safety C Funktionsschnittstelle

```

INT8      NEGI8(INT8 x)
SINT      NEGI8(SINT x)
safeINT8  NEGI8(safeINT8 x)
safeSINT  NEGI8(safeSINT x)

```

Funktionsspezifikation

$$y = \text{NEGI8}(x): f(x) \rightarrow y$$

$$x \in \mathbb{Z}, \quad -2^7 \leq x \leq 2^7 - 1$$

$$y \in \mathbb{Z}, \quad -2^7 + 1 \leq y \leq 2^7 - 1$$

$$f(x) = \begin{cases} -x, & -2^7 + 1 \leq x \leq 2^7 - 1 \\ \text{undef.}, & x = -2^7 \end{cases}$$

5.8.2.19 ABSI32

Führt eine sichere Absolutwertberechnung eines vorzeichenbehafteten 32-Bit-Ganzzahltyps durch.

Safety C Funktionsschnittstelle

```

INT32      ABSI32(INT32 x)
DINT       ABSI32(DINT x)
safeINT32  ABSI32(safeINT32 x)
safeDINT   ABSI32(safeDINT x)

```

Funktionsspezifikation

$$y = \text{ABSI32}(x): f(x) \rightarrow y$$

$$x \in \mathbb{Z}, \quad -2^{31} \leq x \leq 2^{31} - 1$$

$$y \in \mathbb{Z}, \quad 0 \leq y \leq 2^{31} - 1$$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^{31} - 1 \\ -x, & -2^{31} + 1 \leq x < 0 \\ \text{undef.}, & x = -2^{31} \end{cases}$$

5.8.2.20 ABSI16

Führt eine sichere Absolutwertberechnung eines vorzeichenbehafteten 16-Bit-Ganzzahltyps durch.

Safety C Funktionsschnittstelle

```

INT16      ABSI16(INT8 x)
INT         ABSI16(INT x)
safeINT16  ABSI16(safeINT16 x)
safeINT     ABSI16(safeINT x)

```

Funktionsspezifikation

$$y = \text{ABSI16}(x): f(x) \rightarrow y$$

$$x \in \mathbb{Z}, \quad -2^{15} \leq x \leq 2^{15} - 1$$

$$y \in \mathbb{Z}, \quad 0 \leq y \leq 2^{15} - 1$$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^{15} - 1 \\ -x, & -2^{15} + 1 \leq x < 0 \\ \text{undef.}, & x = -2^{15} \end{cases}$$

5.8.2.21 ABSI8

Führt eine sichere Absolutwertberechnung eines vorzeichenbehafteten 8-Bit-Ganzzahltyps durch.

Safety C Funktionsschnittstelle

```
INT8      ABSI8(INT8 x)
SINT      ABSI8(SINT x)
safeINT8  ABSI8(safeINT8 x)
safeSINT  ABSI8(safeSINT x)
```

Funktionsspezifikation

$y = \text{ABSI8}(x) : f(x) \rightarrow y$

$x \in \mathbb{Z}, \quad -2^7 \leq x \leq 2^7 - 1$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 2^7 - 1$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^7 - 1 \\ -x, & -2^7 + 1 \leq x < 0 \\ \text{undef.}, & x = -2^7 \end{cases}$$

5.8.3 Sichere Bit-Schiebe-Funktionen

Die sicheren Bit-Schiebefunktionen vermeiden undefiniertes C/C++ Verhalten der nativen Operatoren >> und <<. Dazu wird zum einen über die Funktionssignatur sichergestellt werden, dass keine vorzeichenbehafteten Operanden verwendet werden können (hiermit ist auch ein arithmetischer Shift mit Vorzeichen ausgeschlossen). Zum anderen wird das Verschieben um grösser-gleich der Wortbreite des linken Operanden (Datenwort) durch den rechten Operand (Schiebeoperand) abgefangen.

5.8.3.1 SHLU32

Verschiebt die Bits eines vorzeichenlosen 32-Bit-Wertes um maximal 31 Bits nach links.

Safety C Funktionsschnittstelle

```
UINT32     SHLU32(UINT32 xa,      UINT32 xb)
UDINT      SHLU32(UDINT xa,      UDINT xb)
safeUINT32 SHLU32(safeUINT32 xa, safeUINT32 xb)
safeUDINT  SHLU32(safeUDINT xa,  safeUDINT xb)
```

Funktionsspezifikation

$y = \text{SHLU32}(x_a, x_b) : f(x_a, x_b) \rightarrow y$

$x_a \in \mathbb{Z}, \quad 0 \leq x_a \leq 2^{32} - 1$
 $x_b \in \mathbb{Z}, \quad 0 \leq x_b \leq 2^{32} - 1$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 2^{32} - 1$

$$f(x_a, x_b) = \begin{cases} (x_a \cdot 2^{x_b}) \bmod 2^{32}, & 0 \leq x_b \leq 31 \\ \text{undef.}, & 32 \leq x_b \leq 2^{32} - 1 \end{cases}$$

5.8.3.2 SHLU16

Verschiebt die Bits eines vorzeichenlosen 16-Bit-Wertes um maximal 15 Bits nach links.

Safety C Funktionsschnittstelle

UINT16	SHLU16(UINT16 xa,	UINT32 xb)
UINT	SHLU16(UINT xa,	UDINT xb)
safeUINT16	SHLU16(safeUINT16 xa,	safeUINT32 xb)
safeUINT	SHLU16(safeUINT xa,	safeUDINT xb)

Funktionsspezifikation

$$y = \text{SHLU16}(x_a, x_b): f(x_a, x_b) \rightarrow y$$

$x_a \in \mathbb{Z},$	$0 \leq x_a \leq 2^{16} - 1$
$x_b \in \mathbb{Z},$	$0 \leq x_b \leq 2^{32} - 1$
$y \in \mathbb{Z},$	$0 \leq y \leq 2^{16} - 1$

$$f(x_a, x_b) = \begin{cases} (x_a \cdot 2^{x_b}) \bmod 2^{16}, & 0 \leq x_b \leq 15 \\ \text{undef.}, & 16 \leq x_b \leq 2^{32} - 1 \end{cases}$$

5.8.3.3 SHLU8

Verschiebt die Bits eines vorzeichenlosen 8-Bit-Wertes um maximal 7 Bits nach links.

Safety C Funktionsschnittstelle

UINT8	SHLU8(UINT8 xa,	UINT32 xb)
USINT	SHLU8(USINT xa,	UDINT xb)
safeUINT8	SHLU8(safeUINT8 xa,	safeUINT32 xb)
safeUSINT	SHLU8(safeUSINT xa,	safeUDINT xb)

Funktionsspezifikation

$$y = \text{SHLU8}(x_a, x_b): f(x_a, x_b) \rightarrow y$$

$x_a \in \mathbb{Z},$	$0 \leq x_a \leq 2^8 - 1$
$x_b \in \mathbb{Z},$	$0 \leq x_b \leq 2^{32} - 1$
$y \in \mathbb{Z},$	$0 \leq y \leq 2^8 - 1$

$$f(x_a, x_b) = \begin{cases} (x_a \cdot 2^{x_b}) \bmod 2^8, & 0 \leq x_b \leq 7 \\ \text{undef.}, & 8 \leq x_b \leq 2^{32} - 1 \end{cases}$$

5.8.3.4 SHR32

Verschiebt die Bits eines vorzeichenlosen 32-Bit-Wertes um maximal 31 Bits nach rechts.

Safety C Funktionsschnittstelle

UINT32	SHR32(UINT32 xa,	UINT32 xb)
UDINT	SHR32(UDINT xa,	UDINT xb)
safeUINT32	SHR32(safeUINT32 xa,	safeUINT32 xb)
safeUDINT	SHR32(safeUDINT xa,	safeUDINT xb)

Funktionsspezifikation

$y = \text{SHRU32}(x_a, x_b): f(x_a, x_b) \rightarrow y$

$x_a \in \mathbb{Z}, \quad 0 \leq x_a \leq 2^{32} - 1$
 $x_b \in \mathbb{Z}, \quad 0 \leq x_b \leq 2^{32} - 1$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 2^{32} - 1$

$$f(x_a, x_b) = \begin{cases} \lfloor x_a \cdot 2^{-x_b} \rfloor, & 0 \leq x_b \leq 31 \\ \text{undef.}, & 32 \leq x_b \leq 2^{32} - 11 \end{cases}$$

5.8.3.5 SHRU16

Verschiebt die Bits eines vorzeichenlosen 16-Bit-Wertes um maximal 15 Bits nach rechts.

Safety C Funktionsschnittstelle

UINT16 SHRU16(UINT16 x_a, UINT32 x_b)
 UINT SHRU16(UINT x_a, UDINT x_b)
 safeUINT16 SHRU16(safeUINT16 x_a, safeUINT32 x_b)
 safeUINT SHRU16(safeUINT x_a, safeUDINT x_b)

Funktionsspezifikation:

$y = \text{SHRU16}(x_a, x_b): f(x_a, x_b) \rightarrow y$

$x_a \in \mathbb{Z}, \quad 0 \leq x_a \leq 2^{16} - 1$
 $x_b \in \mathbb{Z}, \quad 0 \leq x_b \leq 2^{32} - 1$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 2^{16} - 1$

$$f(x_a, x_b) = \begin{cases} \lfloor x_a \cdot 2^{-x_b} \rfloor, & 0 \leq x_b \leq 15 \\ \text{undef.}, & 16 \leq x_b \leq 2^{32} - 1 \end{cases}$$

5.8.3.6 SHRU8

Verschiebt die Bits eines vorzeichenlosen 8-Bit-Wertes um maximal 7 Bits nach rechts.

Safety C Funktionsschnittstelle

UINT8 SHRU8(UINT8 x_a, UINT32 x_b)
 USINT SHRU8(USINT x_a, UDINT x_b)
 safeUINT8 SHRU8(safeUINT8 x_a, safeUINT32 x_b)
 safeUSINT SHRU8(safeUSINT x_a, safeUDINT x_b)

Funktionsspezifikation

$y = \text{SHRU8}(x_a, x_b): f(x_a, x_b) \rightarrow y$

$x_a \in \mathbb{Z}, \quad 0 \leq x_a \leq 2^8 - 1$
 $x_b \in \mathbb{Z}, \quad 0 \leq x_b \leq 2^{32} - 1$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 2^8 - 1$

$$f(x_a, x_b) = \begin{cases} \lfloor x_a \cdot 2^{-x_b} \rfloor, & 0 \leq x_b \leq 7 \\ \text{undef.}, & 8 \leq x_b \leq 2^{32} - 1 \end{cases}$$

5.8.4 Sichere Konvertierungsfunktionen (Boolesch zu Ganzzahl)

Die sicheren Konvertierungen von booleschen Ausdrücken in Ganzzahldatentypen bilden den booleschen Wahrheitswert FALSE auf eine arithmetische 0 und den booleschen Wahrheitswert TRUE auf eine arithmetische 1 im jeweiligen Zieldatentyp ab. Dadurch wird die etwaige Mehrdeutigkeit der expliziten Typumwandlung mit dem Cast-Operator vermieden, welche möglicherweise zu unerwartetem Verhalten beim Anwendungsentwickler führen kann.

5.8.4.1 BTOI32

Führt eine sichere Konvertierung eines booleschen Ausdrucks in einen vorzeichenbehafteten 32-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```
INT32      BTOI32(BOOL x)
DINT      BTOI32(BOOL x)
safeINT32 BTOI32(safeBOOL x)
safeDINT  BTOI32(safeBOOL x)
```

Funktionsspezifikation

$$y = \text{BTOI32}(x): f(x) \rightarrow y$$

$$\begin{aligned} x &\in \{false, true\} \\ y &\in \mathbb{Z}, \quad 0 \leq y \leq 1 \end{aligned}$$

$$f(x) = \begin{cases} 1, & x \\ 0, & \neg x \end{cases}$$

5.8.4.2 BTOI16

Führt eine sichere Konvertierung eines booleschen Ausdrucks in einen vorzeichenbehafteten 16-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```
INT16      BTOI16(BOOL x)
INT        BTOI16(BOOL x)
safeINT16 BTOI16(safeBOOL x)
safeINT    BTOI16(safeBOOL x)
```

Funktionsspezifikation

$$y = \text{BTOI16}(x): f(x) \rightarrow y$$

$$\begin{aligned} x &\in \{false, true\} \\ y &\in \mathbb{Z}, \quad 0 \leq y \leq 1 \end{aligned}$$

$$f(x) = \begin{cases} 1, & x \\ 0, & \neg x \end{cases}$$

5.8.4.3 BTOI8

Führt eine sichere Konvertierung eines booleschen Ausdrucks in einen vorzeichenbehafteten 8-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

INT8 BTOI8(BOOL x)
 SINT BTOI8(BOOL x)
 safeINT8 BTOI8(safeBOOL x)
 safeSINT BTOI8(safeBOOL x)

Funktionsspezifikation

$y = \text{BTOI8}(x): f(x) \rightarrow y$

$x \in \{false, true\}$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 1$

$$f(x) = \begin{cases} 1, & x \\ 0, & \neg x \end{cases}$$

5.8.4.4 BTOU32

Führt eine sichere Konvertierung eines booleschen Ausdrucks in einen vorzeichenlosen 32-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

UINT32 BTOU32(BOOL x)
 UDINT BTOU32(BOOL x)
 safeUINT32 BTOU32(safeBOOL x)
 safeUDINT BTOU32(safeBOOL x)

Funktionsspezifikation

$y = \text{BTOU32}(x): f(x) \rightarrow y$

$x \in \{false, true\}$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 1$

$$f(x) = \begin{cases} 1, & x \\ 0, & \neg x \end{cases}$$

5.8.4.5 BTOU16

Führt eine sichere Konvertierung eines booleschen Ausdrucks in einen vorzeichenlosen 16-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

UINT16 BTOU16(BOOL x)
 UINT BTOU16(BOOL x)
 safeUINT16 BTOU16(safeBOOL x)
 safeUINT BTOU16(safeBOOL x)

Funktionsspezifikation

$y = \text{BTOU16}(x): f(x) \rightarrow y$

$x \in \{false, true\}$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 1$

$$f(x) = \begin{cases} 1, & x \\ 0, & \neg x \end{cases}$$

5.8.4.6 BTOU8

Führt eine sichere Konvertierung eines booleschen Ausdrucks in einen vorzeichenlosen 8-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```
UINT8      BTOU8(BOOL x)
USINT     BTOU8(BOOL x)
safeUINT8 BTOU8(safeBOOL x)
safeUSINT BTOU8(safeBOOL x)
```

Funktionsspezifikation

$y = \text{BTOU8}(x) : f(x) \rightarrow y$

$x \in \{false, true\}$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 1$

$$f(x) = \begin{cases} 1, & x \\ 0, & \neg x \end{cases}$$

5.8.5 Sichere Konvertierungsfunktionen (Ganzzahl zu Ganzzahl)

Die sicheren Konvertierungsfunktionen zwischen Ganzzahltypen sind werterhaltend und vorzeichenerhaltend. Daher fangen die Konvertierungsfunktionen für alle potentiell verlustbehafteten Kombinationen von Quell- und Zieltyp ab, dass sich der Wert des Quelltyps nicht innerhalb des Wertebereichs des Zieltyps darstellen lässt. Bei gefahrlosen Typumwandlungen oder bei möglicherweise beabsichtigtem Wert- oder Vorzeichenverlust muss der native C/C++ Typkonvertierungs-Operator verwendet werden.

5.8.5.1 I8TOU8

Führt eine sichere Konvertierung des vorzeichenbehafteten 8-Bit-Ganzzahltyps zum vorzeichenlosen 8-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```
UINT8      I8TOU8(INT8 x)
USINT     I8TOU8(SINT x)
safeUINT8 I8TOU8(safeINT8 x)
safeUSINT I8TOU8(safeSINT x)
```

Funktionsspezifikation

$y = \text{I8TOU8}(x) : f(x) \rightarrow y$

$x \in \mathbb{Z}, \quad -2^7 \leq x \leq 2^7 - 1$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 2^7 - 1$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^7 - 1 \\ undef., & -2^7 \leq x < 0 \end{cases}$$

5.8.5.2 I8TOU16

Führt eine sichere Konvertierung des vorzeichenbehafteten 8-Bit-Ganzzahltyps zum vorzeichenlosen 16-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

UINT16 I8TOU16(INT8 x)
 UINT I8TOU16(SINT x)
 safeUINT16 I8TOU16(safeINT8 x)
 safeUINT I8TOU16(safeSINT x)

Funktionsspezifikation

$y = I8TOU16(x): f(x) \rightarrow y$

$x \in \mathbb{Z}, \quad -2^7 \leq x \leq 2^7 - 1$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 2^7 - 1$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^7 - 1 \\ undef., & -2^7 \leq x < 0 \end{cases}$$

5.8.5.3 I8TOU32

Führt eine sichere Konvertierung des vorzeichenbehafteten 8-Bit-Ganzzahltyps zum vorzeichenlosen 32-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

UINT32 I8TOU32(INT8 x)
 UDINT I8TOU32(SINT x)
 safeUINT32 I8TOU32(safeINT8 x)
 safeUDINT I8TOU32(safeSINT x)

Funktionsspezifikation

$y = I8TOU32(x): f(x) \rightarrow y$

$x \in \mathbb{Z}, \quad -2^7 \leq x \leq 2^7 - 1$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 2^7 - 1$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^7 - 1 \\ undef., & -2^7 \leq x < 0 \end{cases}$$

5.8.5.4 U8TOI8

Führt eine sichere Konvertierung des vorzeichenlosen 8-Bit-Ganzzahltyps zum vorzeichenbehafteten 8-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

INT8 U8TOI8(UINT8 x)
 SINT U8TOI8(USINT x)
 safeINT8 U8TOI8(safeUINT8 x)
 safeSINT U8TOI8(safeUSINT x)

Funktionsspezifikation

$y = U8TOI8(x): f(x) \rightarrow y$

$x \in \mathbb{Z}, \quad 0 \leq x \leq 2^8 - 1$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 2^7 - 1$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^7 - 1 \\ undef., & 2^7 \leq x \leq 2^8 - 1 \end{cases}$$

5.8.5.5 I16TOI8

Führt eine sichere Konvertierung des vorzeichenbehafteten 16-Bit-Ganzzahltyps zum vorzeichenbehafteten 8-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```
INT8      I16TOI8(INT16 x)
SINT     I16TOI8(INT x)
safeINT8 I16TOI8(safeINT16 x)
safeSINT I16TOI8(safeINT x)
```

Funktionsspezifikation

$y = I16TOI8(x): f(x) \rightarrow y$

$x \in \mathbb{Z}, \quad -2^{15} \leq x \leq 2^{15} - 1$
 $y \in \mathbb{Z}, \quad -2^7 \leq y \leq 2^7 - 1$

$$f(x) = \begin{cases} x, & -2^7 \leq x \leq 2^7 - 1 \\ \text{undef.}, & (-2^{15} \leq x < -2^7) \vee (2^7 \leq x \leq 2^{15} - 1) \end{cases}$$

5.8.5.6 I16TOU8

Führt eine sichere Konvertierung des vorzeichenbehafteten 16-Bit-Ganzzahltyps zum vorzeichenlosen 8-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```
UINT8     I16TOU8(INT16 x)
USINT     I16TOU8(INT x)
safeUINT8 I16TOU8(safeINT16 x)
safeUSINT I16TOU8(safeINT x)
```

Funktionsspezifikation

$y = I16TOU8(x): f(x) \rightarrow y$

$x \in \mathbb{Z}, \quad -2^{15} \leq x \leq 2^{15} - 1$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 2^8 - 1$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^8 - 1 \\ \text{undef.}, & (-2^{15} \leq x < 0) \vee (2^8 \leq x \leq 2^{15} - 1) \end{cases}$$

5.8.5.7 I16TOU16

Führt eine sichere Konvertierung des vorzeichenbehafteten 16-Bit-Ganzzahltyps zum vorzeichenlosen 16-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```
UINT16    I16TOU16(INT16 x)
UINT      I16TOU16(INT x)
safeUINT16 I16TOU16(safeINT16 x)
safeUINT   I16TOU16(safeINT x)
```


Funktionsspezifikation

$$y = \text{I16TOU16}(x): f(x) \rightarrow y$$

$$\begin{aligned} x \in \mathbb{Z}, & & -2^{15} \leq x \leq 2^{15} - 1 \\ y \in \mathbb{Z}, & & 0 \leq y \leq 2^{15} - 1 \end{aligned}$$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^{15} - 1 \\ \text{undef.}, & -2^{15} \leq x < 0 \end{cases}$$

5.8.5.8 I16TOU32

Führt eine sichere Konvertierung des vorzeichenbehafteten 16-Bit-Ganzzahltyps zum vorzeichenlosen 32-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```
UINT32      I16TOU32(INT16 x)
UDINT       I16TOU32(INT x)
safeUINT32  I16TOU32(safeINT16 x)
safeUDINT   I16TOU32(safeINT x)
```

Funktionsspezifikation

$$y = \text{I16TOU32}(x): f(x) \rightarrow y$$

$$\begin{aligned} x \in \mathbb{Z}, & & -2^{15} \leq x \leq 2^{15} - 1 \\ y \in \mathbb{Z}, & & 0 \leq y \leq 2^{15} - 1 \end{aligned}$$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^{15} - 1 \\ \text{undef.}, & -2^{15} \leq x < 0 \end{cases}$$

5.8.5.9 U16TOI8

Führt eine sichere Konvertierung des vorzeichenlosen 16-Bit-Ganzzahltyps zum vorzeichenbehafteten 8-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```
INT8        U16TOI8(UINT16 x)
SINT        U16TOI8(UINT x)
safeINT8    U16TOI8(safeUINT16 x)
safeSINT    U16TOI8(safeUINT x)
```

Funktionsspezifikation

$$y = \text{U16TOI8}(x): f(x) \rightarrow y$$

$$\begin{aligned} x \in \mathbb{Z}, & & 0 \leq x \leq 2^{16} - 1 \\ y \in \mathbb{Z}, & & 0 \leq y \leq 2^7 - 1 \end{aligned}$$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^7 - 1 \\ \text{undef.}, & 2^7 \leq x \leq 2^{16} - 1 \end{cases}$$

5.8.5.10 U16TOU8

Führt eine sichere Konvertierung des vorzeichenlosen 16-Bit-Ganzzahltyps zum vorzeichenlosen 8-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```

UINT8      U16TOU8(UINT16 x)
USINT      U16TOU8(UINT x)
safeUINT8  U16TOU8(safeUINT16 x)
safeUSINT  U16TOU8(safeUINT x)

```

Funktionsspezifikation

$$y = \text{U16TOU8}(x): f(x) \rightarrow y$$

$$x \in \mathbb{Z}, \quad 0 \leq x \leq 2^{16} - 1$$

$$y \in \mathbb{Z}, \quad 0 \leq y \leq 2^8 - 1$$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^8 - 1 \\ \text{undef.}, & 2^8 \leq x \leq 2^{16} - 1 \end{cases}$$

5.8.5.11 U16TOI16

Führt eine sichere Konvertierung des vorzeichenlosen 16-Bit-Ganzzahltyps zum vorzeichenbehafteten 16-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```

INT16      U16TOI16(UINT16 x)
INT         U16TOI16(UINT x)
safeINT16  U16TOI16(safeUINT16 x)
safeINT     U16TOI16(safeUINT x)

```

Funktionsspezifikation

$$y = \text{U16TOI16}(x): f(x) \rightarrow y$$

$$x \in \mathbb{Z}, \quad 0 \leq x \leq 2^{16} - 1$$

$$y \in \mathbb{Z}, \quad 0 \leq y \leq 2^{15} - 1$$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^{15} - 1 \\ \text{undef.}, & 2^{15} \leq x \leq 2^{16} - 1 \end{cases}$$

5.8.5.12 I32TOI8

Führt eine sichere Konvertierung des vorzeichenbehafteten 32-Bit-Ganzzahltyps zum vorzeichenbehafteten 8-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```

INT8       I32TOI8(INT32 x)
SINT       I32TOI8(DINT x)
safeINT8   I32TOI8(safeINT32 x)
safeSINT   I32TOI8(safeDINT x)

```

Funktionsspezifikation

$$y = \text{I32TOI8}(x): f(x) \rightarrow y$$

$$x \in \mathbb{Z}, \quad -2^{31} \leq x \leq 2^{31} - 1$$

$$y \in \mathbb{Z}, \quad -2^7 \leq y \leq 2^7 - 1$$

$$f(x) = \begin{cases} x, & -2^7 \leq x \leq 2^7 - 1 \\ \text{undef.}, & (-2^{31} \leq x < -2^7) \vee (2^7 \leq x \leq 2^{31} - 1) \end{cases}$$

5.8.5.13 I32TOU8

Führt eine sichere Konvertierung des vorzeichenbehafteten 32-Bit-Ganzzahltyps zum vorzeichenlosen 8-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```
UINT8      I32TOU8(INT32 x)
USINT      I32TOU8(DINT x)
safeUINT8  I32TOU8(safeINT32 x)
safeUSINT  I32TOU8(safeDINT x)
```

Funktionsspezifikation

$y = I32TOU8(x): f(x) \rightarrow y$

$x \in \mathbb{Z}, \quad -2^{31} \leq x \leq 2^{31} - 1$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 2^8 - 1$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^8 - 1 \\ undef., & (-2^{31} \leq x < 0) \vee (2^8 \leq x \leq 2^{31} - 1) \end{cases}$$

5.8.5.14 I32TOI16

Führt eine sichere Konvertierung des vorzeichenbehafteten 32-Bit-Ganzzahltyps zum vorzeichenbehafteten 16-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```
INT8       I32TOI16(INT32 x)
SINT       I32TOI16(DINT x)
safeINT8   I32TOI16(safeINT32 x)
safeSINT   I32TOI16(safeDINT x)
```

Funktionsspezifikation

$y = I32TOI16(x): f(x) \rightarrow y$

$x \in \mathbb{Z}, \quad -2^{31} \leq x \leq 2^{31} - 1$
 $y \in \mathbb{Z}, \quad -2^{15} \leq y \leq 2^{15} - 1$

$$f(x) = \begin{cases} x, & -2^{15} \leq x \leq 2^{15} - 1 \\ undef., & (-2^{31} \leq x < -2^{15}) \vee (2^{15} \leq x \leq 2^{31} - 1) \end{cases}$$

5.8.5.15 I32TOU16

Führt eine sichere Konvertierung des vorzeichenbehafteten 32-Bit-Ganzzahltyps zum vorzeichenlosen 16-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```
UINT16     I32TOU16(INT32 x)
UINT       I32TOU16(DINT x)
safeUINT16 I32TOU16(safeINT32 x)
safeUINT   I32TOU16(safeDINT x)
```

Funktionsspezifikation

$$y = \text{I32TOU16}(x): f(x) \rightarrow y$$

$$\begin{aligned} x &\in \mathbb{Z}, & -2^{31} \leq x \leq 2^{31} - 1 \\ y &\in \mathbb{Z}, & 0 \leq y \leq 2^{16} - 1 \end{aligned}$$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^{16} - 1 \\ \text{undef.}, & (-2^{31} \leq x < 0) \vee (2^{16} \leq x \leq 2^{31} - 1) \end{cases}$$

5.8.5.16 I32TOU32

Führt eine sichere Konvertierung des vorzeichenbehafteten 32-Bit-Ganzzahltyps zum vorzeichenlosen 32-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```

UINT32      I32TOU32(INT32 x)
UDINT       I32TOU32(DINT x)
safeUINT32  I32TOU32(safeINT32 x)
safeUDINT   I32TOU32(safeDINT x)

```

Funktionsspezifikation

$$y = \text{I32TOU32}(x): f(x) \rightarrow y$$

$$\begin{aligned} x &\in \mathbb{Z}, & -2^{31} \leq x \leq 2^{31} - 1 \\ y &\in \mathbb{Z}, & 0 \leq y \leq 2^{31} - 1 \end{aligned}$$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^{31} - 1 \\ \text{undef.}, & (-2^{31} \leq x < 0) \vee (2^{31} \leq x \leq 2^{31} - 1) \end{cases}$$

5.8.5.17 U32TOI8

Führt eine sichere Konvertierung des vorzeichenlosen 32-Bit-Ganzzahltyps zum vorzeichenbehafteten 8-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```

INT8        U32TOI8(UINT32 x)
SINT        U32TOI8(UDINT x)
safeINT8    U32TOI8(safeUINT32 x)
safeSINT    U32TOI8(safeUDINT x)

```

Funktionsspezifikation

$$y = \text{U32TOI8}(x): f(x) \rightarrow y$$

$$\begin{aligned} x &\in \mathbb{Z}, & 0 \leq x \leq 2^{32} - 1 \\ y &\in \mathbb{Z}, & 0 \leq y \leq 2^7 - 1 \end{aligned}$$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^7 - 1 \\ \text{undef.}, & 2^7 \leq x \leq 2^{32} - 1 \end{cases}$$

5.8.5.18 U32TOU8

Führt eine sichere Konvertierung des vorzeichenlosen 32-Bit-Ganzzahltyps zum vorzeichenlosen 8-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

UINT8 U32TOU8(UINT32 x)
 USINT U32TOU8(UDINT x)
 safeUINT8 U32TOU8(safeUINT32 x)
 safeUSINT U32TOU8(safeUDINT x)

Funktionsspezifikation

$y = U32TOU8(x): f(x) \rightarrow y$

$x \in \mathbb{Z}, \quad 0 \leq x \leq 2^{32} - 1$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 2^8 - 1$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^8 - 1 \\ undef., & 2^8 \leq x \leq 2^{32} - 1 \end{cases}$$

5.8.5.19 U32TOI16

Führt eine sichere Konvertierung des vorzeichenlosen 32-Bit-Ganzzahltyps zum vorzeichenbehafteten 16-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

INT8 U32TOI16(UINT32 x)
 SINT U32TOI16(UDINT x)
 safeINT8 U32TOI16(safeUINT32 x)
 safeSINT U32TOI16(safeUDINT x)

Funktionsspezifikation

$y = U32TOI16(x): f(x) \rightarrow y$

$x \in \mathbb{Z}, \quad 0 \leq x \leq 2^{32} - 1$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 2^{15} - 1$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^{15} - 1 \\ undef., & 2^{15} \leq x \leq 2^{32} - 1 \end{cases}$$

5.8.5.20 U32TOU16

Führt eine sichere Konvertierung des vorzeichenlosen 32-Bit-Ganzzahltyps zum vorzeichenlosen 16-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

UINT16 U32TOU16(UINT32 x)
 UUINT U32TOU16(UDINT x)
 safeUINT16 U32TOU16(safeUINT32 x)
 safeUUINT U32TOU16(safeUDINT x)

Funktionsspezifikation

$y = U32TOU16(x): f(x) \rightarrow y$

$x \in \mathbb{Z}, \quad 0 \leq x \leq 2^{32} - 1$
 $y \in \mathbb{Z}, \quad 0 \leq y \leq 2^{16} - 1$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^{16} - 1 \\ undef., & 2^{16} \leq x \leq 2^{32} - 1 \end{cases}$$

5.8.5.21 U32TOI32

Führt eine sichere Konvertierung des vorzeichenlosen 32-Bit-Ganzzahltyps zum vorzeichenbehafteten 32-Bit-Ganzzahltyp durch.

Safety C Funktionsschnittstelle

```
INT32      U32TOI32(UINT32 x)
DINT      U32TOI32(UDINT x)
safeINT32 U32TOI32(safeUINT32 x)
safeDINT  U32TOI32(safeUDINT x)
```

Funktionsspezifikation

$y = \text{U32TOI32}(x): f(x) \rightarrow y$

$x \in \mathbb{Z}, \quad 0 \leq x \leq 2^{32} - 1$

$y \in \mathbb{Z}, \quad 0 \leq y \leq 2^{31} - 1$

$$f(x) = \begin{cases} x, & 0 \leq x \leq 2^{31} - 1 \\ \text{undef.}, & 2^{31} \leq x \leq 2^{32} - 1 \end{cases}$$

6 Graphische Anwendungsentwicklung



Hinweis

Verwendung des TwinCAT Safety Editors

Die Verwendung des TwinCAT Safety Editors zusammen mit der TwinCAT Safety PLC wird in einem der nächsten Releases umgesetzt. Derzeit ist eine Verwendung nicht möglich.

7 Anhang

7.1 Support und Service

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

Beckhoff Support

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

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

Hotline: +49(0)5246/963-157
Fax: +49(0)5246/963-9157
E-Mail: support@beckhoff.com

Beckhoff Service

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

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

Hotline: +49(0)5246/963-460
Fax: +49(0)5246/963-479
E-Mail: service@beckhoff.com

Weitere Support- und Serviceadressen finden Sie auf unseren Internetseiten unter <http://www.beckhoff.de>.

Beckhoff Firmenzentrale

Beckhoff Automation GmbH & Co. KG

Hülshorstweg 20
33415 Verl
Deutschland

Telefon: +49(0)5246/963-0
Fax: +49(0)5246/963-198
E-Mail: info@beckhoff.com

Die Adressen der weltweiten Beckhoff Niederlassungen und Vertretungen entnehmen Sie bitte unseren Internetseiten:

<http://www.beckhoff.de>

Dort finden Sie auch weitere Dokumentationen zu Beckhoff Komponenten.

7.2 Zertifikate

ZERTIFIKAT ◆ CERTIFICATE ◆ 認証証書 ◆ CERTIFICADO ◆ CERTIFICAT



Product Service

CERTIFICATE

No. Z10 16 12 62386 035

Holder of Certificate: Beckhoff Automation GmbH & Co. KG
 Hülshorstweg 20
 33415 Verl
 GERMANY

Factory(ies): 62386

Certification Mark:



Product: Safety-Related Programmable Systems

Model(s): TwinCAT Safety PLC

Parameters:
 Supply voltage: SELV/PELV
 Protection class: IP 20
 Ambient temperature: 0°C ... +55°C

The report referenced below and the user documentation in the currently valid revision are mandatory part of this certificate. The product complies with the following listed safety requirements only if the specifications documented in the currently valid revisions of this report are met.

Tested according to:
 IEC 61508-1(ed.2) (SIL 3)
 IEC 61508-2(ed.2) (SIL 3)
 IEC 61508-3(ed.2) (SIL 3)
 IEC 61508-4(ed.2) (SIL 3)
 EN ISO 13849-1:2015 (Cat 4, PL e)

The product was tested on a voluntary basis and complies with the essential requirements. The certification mark shown above can be affixed on the product. It is not permitted to alter the certification mark in any way. In addition the certification holder must not transfer the certificate to third parties. See also notes overleaf.

Test report no.: BV90306C

Valid until: 2021-12-08

Date, 2016-12-12

J. Blum
 (Jürgen Blum)



Page 1 of 1

TÜV SÜD Product Service GmbH · Zertifizierstelle · Ridlerstraße 65 · 80339 München · Germany

TUV®

A1 / 04.11

Abbildungsverzeichnis

Abb. 1	Typische Reaktionszeit.....	15
Abb. 2	Worst-Case Reaktionszeit	16
Abb. 3	Anlegen eines Safety Projektes - Add New Item	17
Abb. 4	Anlegen eines Safety Projektes - Projektname und Verzeichnis.....	18
Abb. 5	TwinCAT Safety Project Wizard	18
Abb. 6	Target System im Solution Explorer	19
Abb. 7	Target System Property Page	19
Abb. 8	Anlegen einer TwinSAFE -Gruppe	20
Abb. 9	TwinSAFE-Gruppe.....	20
Abb. 10	TwinSAFE Group - General Settings.....	21
Abb. 11	TwinSAFE Group - Group Ports	21
Abb. 12	TwinSAFE-Gruppe - Prozessabbild.....	21
Abb. 13	TSGData struct	22
Abb. 14	Starten des automatischen Imports aus der I/O-Konfiguration.....	22
Abb. 15	Auswahl aus dem I/O Baum	23
Abb. 16	Anlegen der Alias Devices durch den Anwender.....	23
Abb. 17	Alias Device - Reiter Process Image	24
Abb. 18	Konfigurieren der I/O Elemente	24
Abb. 19	Alias Device - Reiter Connection.....	25
Abb. 20	Alias Device in der Safety-Projektstruktur	25
Abb. 21	Verlinkungen zum TwinCAT Safety PLC-Prozessabbild	26
Abb. 22	Verbindungsspezifische Parameter der Connection.....	26
Abb. 23	Auswahl eines Alias Devices	27
Abb. 24	Safety-Parameter des Geräts	27
Abb. 25	Safety PLC Instanz - Alias Devices	28
Abb. 26	Struktur des Alias Devices.....	28
Abb. 27	AX5000-Safety-Antriebsoptionen	29
Abb. 28	AX5000-Safety-Antriebsoptionen - General AX5805 settings	29
Abb. 29	AX5000-Safety-Antriebsoptionen - Process Image	30
Abb. 30	AX5000-Safety-Antriebsoptionen - Function Diagram.....	31
Abb. 31	Anlegen einer externen Verbindung (Custom FSoE Connection)	32
Abb. 32	Parametrierung der Prozessabbildgröße.....	32
Abb. 33	Umbenennen der einzelnen Signale innerhalb des Telegramms	33
Abb. 34	Auswahl der Variablen.....	33
Abb. 35	Direkte Verknüpfung auf das Prozessabbild einer EtherCAT-Klemme	34
Abb. 36	Verbindungsspezifische Parameter	34
Abb. 37	TwinSAFE-Gruppe - Header Files	36
Abb. 38	TwinSAFE-Gruppe - Source Files	37
Abb. 39	Init-Funktion	38
Abb. 40	InputUpdate-Funktion	38
Abb. 41	OutputUpdate-Funktion	38
Abb. 42	CycleUpdate-Funktion	39
Abb. 43	Target System	40
Abb. 44	Dialog CRC Distribution.....	40

Abb. 45	Download der Safety Applikation.....	41
Abb. 46	Unlock Safety Project	41
Abb. 47	Unlock Safety Project	42
Abb. 48	Identische CRCs.....	42
Abb. 49	Aktivierung Safety und CRC Toolbar.....	42
Abb. 50	Target System - Map Object ID und Map Project CRC	43
Abb. 51	Target System Info Data.....	43
Abb. 52	Diag Data - TwinCAT Safety PLC.....	44
Abb. 53	Safety Timer Diag Data - TwinCAT Safety PLC	45
Abb. 54	Gruppen MapDiag MapState	45
Abb. 55	Gruppen Infodaten.....	45
Abb. 56	FSoE Connection Map Info Data.....	46
Abb. 57	Infodaten FSoE Connection.....	46
Abb. 58	Hinzufügen einer neuen Task.....	48
Abb. 59	Dialog Insert Task.....	48
Abb. 60	Einstellungen Task	49
Abb. 61	Task Ausführungszeit und Überschreitungszähler	49
Abb. 62	Strenges Typsystem Fall 1 - Beispiel 1	63
Abb. 63	Strenges Typsystem Fall 1 - Beispiel 2	63
Abb. 64	Strenges Typsystem Fall 1 - Beispiel 3	64
Abb. 65	Strenges Typsystem Fall 1 - Beispiel 4	64
Abb. 66	Strenges Typsystem Fall 1 - Beispiel 5	65
Abb. 67	Strenges Typsystem Fall 2 - Beispiel 1	65
Abb. 68	Strenges Typsystem Fall 2 - Beispiel 2	66
Abb. 69	Kontextmenu ModuleTests	83