Programmier-styleguide
für SIMATIC S7-1200/ S7-1500

TIA Portal

https://support.industry.siemens.com/cs/ww/de/view/81318674

Siemens Industry Online Support

Rechtliche Hinweise

Nutzung der Anwendungsbeispiele

In den Anwendungsbeispielen wird die Lösung von Automatisierungsaufgaben im Zusammenspiel mehrerer Komponenten in Form von Text, Grafiken und/oder Software-Bausteinen beispielhaft dargestellt. Die Anwendungsbeispiele sind ein kostenloser Service der Siemens AG und/oder einer Tochtergesellschaft der Siemens AG ("Siemens"). Sie sind unverbindlich und erheben keinen Anspruch auf Vollständigkeit und Funktionsfähigkeit hinsichtlich Konfiguration und Ausstattung. Die Anwendungsbeispiele stellen keine kundenspezifischen Lösungen dar, sondern bieten lediglich Hilfestellung bei typischen Aufgabenstellungen. Sie sind selbst für den sachgemäßen und sicheren Betrieb der Produkte innerhalb der geltenden Vorschriften verantwortlich und müssen dazu die Funktion des jeweiligen Anwendungsbeispiels überprüfen und auf Ihre Anlage individuell anpassen.
Sie erhalten von Siemens das nicht ausschließliche, nicht unterlizenzierbare und nicht übertragbare Recht, die Anwendungsbeispiele durch fachlich geschultes Personal zu nutzen. Jede Änderung an den Anwendungsbeispielen erfolgt auf Ihre Verantwortung. Die Weitergabe an Dritte oder Vervielfältigung der Anwendungsbeispiele oder von Auszügen daraus ist nur in Kombination mit Ihren eigenen Produkten gestattet. Die Anwendungsbeispiele unterliegen nicht zwingend den üblichen Tests und Qualitätsprüfungen eines kostenpflichtigen Produkts, können Funktions- und Leistungsmängel enthalten und mit Fehlern behaftet sein. Sie sind verpflichtet, die Nutzung so zu gestalten, dass eventuelle Fehlfunktionen nicht zu Sachschäden oder der Verletzung von Personen führen.

Haftungsausschluss

Siemens schließt seine Haftung, gleich aus welchem Rechtsgrund, insbesondere für die Verwendbarkeit, Verfügbarkeit, Vollständigkeit und Mangelfreiheit der Anwendungsbeispiele, sowie dazugehöriger Hinweise, Projektierungs- und Leistungsdaten und dadurch verursachte Schäden aus. Dies gilt nicht, soweit Siemens zwingend haftet, z. B. nach dem Produkthaftungsgesetz, in Fällen des Vorsatzes, der groben Fahrlässigkeit, wegen der schuldhaften Verletzung des Lebens, des Körpers oder der Gesundheit, bei Nichteinhaltung einer übernommenen Garantie, wegen des arglistigen Verschweigens eines Mangels oder wegen der schuldhaften Verletzung wesentlicher Vertragspflichten. Der Schadensersatzanspruch für die Verletzung wesentlicher Vertragspflichten ist jedoch auf den vertragstypischen, vorhersehbaren Schaden begrenzt, soweit nicht Vorsatz oder grobe Fahrlässigkeit vorliegen oder wegen der Verletzung des Lebens, des Körpers oder der Gesundheit gehaftet wird. Eine Änderung der Beweislast zu Ihrem Nachteil ist mit den vorstehenden Regelungen nicht verbunden. Von in diesem Zusammenhang bestehenden oder entstehenden Ansprüchen Dritter stellen Sie Siemens frei, soweit Siemens nicht gesetzlich zwingend haftet.
Durch Nutzung der Anwendungsbeispiele erkennen Sie an, dass Siemens über die beschriebene Haftungsregelung hinaus nicht für etwaige Schäden haftbar gemacht werden kann.

Weitere Hinweise

Siemens behält sich das Recht vor, Änderungen an den Anwendungsbeispielen jederzeit ohne Ankündigung durchzuführen. Bei Abweichungen zwischen den Vorschlägen in den Anwendungsbeispielen und anderen Siemens Publikationen, wie z. B. Katalogen, hat der Inhalt der anderen Dokumentation Vorrang.
Ergänzend gelten die Siemens Nutzungsbedingungen (https://support.industry.siemens.com).

Securityhinweise

Siemens bietet Produkte und Lösungen mit Industrial Security-Funktionen an, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen.
Um Anlagen, Systeme, Maschinen und Netzwerke gegen Cyber-Bedrohungen zu sichern, ist es erforderlich, ein ganzheitliches Industrial Security-Konzept zu implementieren (und kontinuierlich aufrechtzuerhalten), das dem aktuellen Stand der Technik entspricht. Die Produkte und Lösungen von Siemens formen nur einen Bestandteil eines solchen Konzepts.
Der Kunde ist dafür verantwortlich, unbefugten Zugriff auf seine Anlagen, Systeme, Maschinen und Netzwerke zu verhindern. Systeme, Maschinen und Komponenten sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn und soweit dies notwendig ist und entsprechende Schutzmaßnahmen (z. B. Nutzung von Firewalls und Netzwerksegmentierung) ergriffen wurden.
Zusätzlich sollten die Empfehlungen von Siemens zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Industrial Security finden Sie unter: https://www.siemens.com/industrialsecurity.
Die Produkte und Lösungen von Siemens werden ständig weiterentwickelt, um sie noch sicherer zu machen. Siemens empfiehlt ausdrücklich, Aktualisierungen durchzuführen, sobald die entsprechenden Updates zur Verfügung stehen und immer nur die aktuellen Produktversionen zu verwenden. Die Verwendung veralteter oder nicht mehr unterstützter Versionen kann das Risiko von Cyber-Bedrohungen erhöhen.
Um stets über Produkt-Updates informiert zu sein, abonnieren Sie den Siemens Industrial Security RSS Feed unter: https://www.siemens.com/industrialsecurity.

Inhaltsverzeichnis

1 Einleitung

Bei der Programmierung von SIMATIC-Steuerungen hat der Programmierer die Aufgabe, das Anwenderprogramm so übersichtlich und lesbar wie möglich zu gestalten. Jeder Programmierer wendet eine eigene Strategie an, wie z. B. Variablen oder Bausteine benannt werden oder in welcher Art und Weise kommentiert wird. Durch die unterschiedlichen Philosophien der Programmierer entstehen sehr unterschiedliche Anwenderprogramme, die häufig nur vom jeweiligen Ersteller interpretiert werden können.

Hinweis Als Grundlage für dieses Dokument gilt der Programmierleitfaden für SIMATIC S7-1200/ S7-1500. Dieser beschreibt Systemeigenschaften der Steuerungen S7-1200 und S7-1500 und wie diese optimal programmiert werden können:

https://support.industry.siemens.com/cs/ww/de/view/81318674

1.1 Ziele

Die hier beschriebenen Regeln und Empfehlungen helfen Ihnen, einen einheitlichen Programmcode zu erstellen, der besser gewartet und wiederverwendet werden kann. Falls mehrere Programmierer am selben Programm arbeiten, empfiehlt es sich deshalb, eine projektweit gültige Terminologie festzulegen sowie einen gemeinsamen und abgestimmten Programmierstil einzuhalten. Somit können möglichst frühzeitig Fehler erkannt bzw. vermieden werden.

Für die Wartbarkeit und Übersichtlichkeit des Quellcodes ist es zunächst erforderlich, sich an eine gewisse äußere Form zu halten. Optische Effekte tragen nur unwesentlich zur Qualität der Software bei. Viel wichtiger ist es beispielsweise auch Regeln zu finden, die die Entwickler folgendermaßen unterstützen:

- Vermeiden von Tipp- und Flüchtigkeitsfehlern, die der Compiler anschließend falsch interpretiert.
  Ziel: Der Compiler soll so viele Fehler wie möglich erkennen.
- Unterstützung des Programmierers bei der Diagnose von Programmfehlern, z. B. versehentliche Wiederverwendung einer temporären Variablen über einen Zyklus hinaus.
  Ziel: Der Bezeichner weist frühzeitig auf Probleme hin.
- Vereinheitlichung von Standardapplikationen und Standardbibliotheken
  Ziel: Die Einarbeitung soll einfach sein und die Wiederverwendbarkeit von Programmcode erhöht werden.
- Einfache Wartung und Vereinfachung der Weiterentwicklung
  Ziel: Änderungen von Programmcode in den einzelnen Modulen, die Organisationsbausteine, Funktionen, Funktionsbausteine und Datenbausteine in Bibliotheken oder im Projekt umfassen können, sollen minimale Auswirkungen auf das Gesamtprogramm/ die Gesamtbibliothek haben. Änderungen von Programmcode in den einzelnen Modulen sollen von verschiedenen Programmierern durchführbar sein.

Hinweis Beachten Sie, dass die in diesem Dokument beschriebene Regeln und Empfehlungen aufeinander abgestimmt sind und aufeinander aufbauen.

1.2 Vorteile eines einheitlichen Programmierstils

- Einheitlicher durchgängiger Stil
- Leicht lesbar und verständlich
- Einfache Wartung und Wiederverwendbarkeit
- Einfache und schnelle Fehlererkennung und -korrektur
- Effiziente Zusammenarbeit zwischen mehreren Programmierern

1.3 Gültigkeit

Dieses Dokument gilt für Projekte und Bibliotheken in TIA Portal, die in den Programmiersprachen der IEC 61131-3 (DIN EN 61131-3) Strukturierter Text (SCL/ ST), Kontaktplan (KOP) und Funktionsplan (FUP) erstellt werden.

Ebenso gilt dieses Dokument für Software Units, Ordner, Gruppen, Organisationsbausteine (OB), Funktionen (FC), Funktionsbausteine (FB), Technologieobjekte (TO), Datenbausteine (DB), PLC-Datentypen (UDT), Variablen, Konstanten, PLC-Variablentabellen, PLC-Variablen, Anwender-konstanten, PLC-Meldetextlisten, Beobachtungs- und Force-Tabellen sowie externe Quellen.

1.4 Abgrenzung

Dieses Dokument enthält keine Beschreibung von:

- STEP 7-Programmierung mit TIA Portal
- Inbetriebnahme von SIMATIC-Steuerungen

Ausreichende Erfahrung in diesen Themen wird vorausgesetzt, um die Regeln und Empfehlungen sinnvoll interpretieren und anwenden zu können.

Das Dokument ersetzt keinen Know-how Aufbau in Softwareentwicklung, sondern dient als Referenz.

1.5 Regelverletzung und andere Vorgaben

Bei Kundenprojekten sind die geforderten Normen sowie die kunden- oder branchenspezifischen Standards des Kunden oder der verwendeten Technologie (z. B. Safety, Motion, …) einzuhalten und besitzen Priorität gegenüber diesem Styleguide oder Teilen davon.

Bei einer Kombination von Kundenvorgaben und diesem Styleguide ist auf die Integrität der Kombination beider Vorgaben und des Gesamtprojekts zu achten.

Eine Regelverletzung ist an der entsprechenden Stelle im Anwenderprogramm zu begründen und zu dokumentieren.

Die vom Kunden definierten Regeln sind in geeigneter Form zu dokumentieren.

2 Begriffsklärung

2.1 Regeln/ Empfehlungen

Die Vorgaben dieses Dokuments werden in Empfehlungen und Regeln unterteilt:

- **Regeln** sind verbindliche Vorgaben und sind unbedingt einzuhalten. Sie sind für eine wiederverwendbare und performante Programmierung unabdingbar. Im Ausnahmefall können Regeln auch verletzt werden. Dies muss aber entsprechend dokumentiert werden.
- **Empfehlungen** sind Vorgaben, die zum einen der Einheitlichkeit des Codes dienen und zum anderen als Unterstützung und Hinweis gedacht sind. Empfehlungen sollten prinzipiell befolgt werden, es kann aber durchaus Fälle geben, in denen eine Empfehlung nicht befolgt wird. Gründe hierfür können bessere Effizienz oder bessere Lesbarkeit sein.

2.2 Regelnummerierung

Für eine eindeutige Regelzuordnung zwischen verschiedenen Kategorien werden die Regeln und Empfehlungen entsprechend ihrer Zugehörigkeit mit einem Präfix (zwei Zeichen) gekennzeichnet und nummeriert (3 Ziffern).

Entfällt eine Regel, wird die Nummer nicht neu vergeben. Wenn Sie weitere Regeln definieren, können Sie eine Nummerierung zwischen 901 und 999 nutzen.

Tabelle 2-1

Präfix	Kategorie
ES	Engineering System: Programmierumgebung
GL	Globalization: Globalisierung
NF	Nomenclature and formatting: Nomenklatur und Formatierung
RU	Reusability: Wiederverwendbarkeit
AL	Allocation: Referenzieren von Objekten (Allokieren)
SE	Security: Sicherheit
DA	Design and architecture: Design und Architektur
PE	Performance: Performanz

2.3 Performance

Unter Performance eines Automatisierungssystems wird die Bearbeitungszeit (Zykluszeit) eines Programms definiert.

Ist von einem Performancenachteil die Rede, so bedeutet dies, dass es möglich wäre, durch Anwendung der Programmierregeln und durch geschickte Programmierung des Anwenderprogramms die Bearbeitungszeit und somit die Zykluszeit eines Programmdurchlaufs zu verringern.

2.4 Bezeichner/ Name

Es ist wichtig, zwischen Namen und Bezeichnern (Identifier) zu unterscheiden. Der Name ist Teil eines Bezeichners, der die jeweilige Bedeutung beschreibt.

Der Bezeichner setzt sich zusammen aus:

- Präfix
- dem Namen
- Suffix

2.5 Abkürzungen

Folgende Abkürzungen werden innerhalb dieses Dokuments verwendet:

Tabelle 2-2

Abkürzung	Typ
OB	Organisationsbaustein
FB	Funktionsbaustein
FC	Funktion
DB	Datenbaustein
TO	Technologieobjekt
UDT	PLC-Datentyp

2.6 Begriffe bei Variablen und Parametern

Wenn es um Variablen, Funktionen und Funktionsbausteine geht, gibt es viele Begriffe, die immer wieder unterschiedlich oder sogar falsch benutzt werden. Die folgende Abbildung stellt diese Begriffe klar. Diese Klarstellung ist erforderlich, damit im Weiteren ein gemeinsames Verständnis über die verwendeten Begriffe herrscht.

Abbildung 2-1

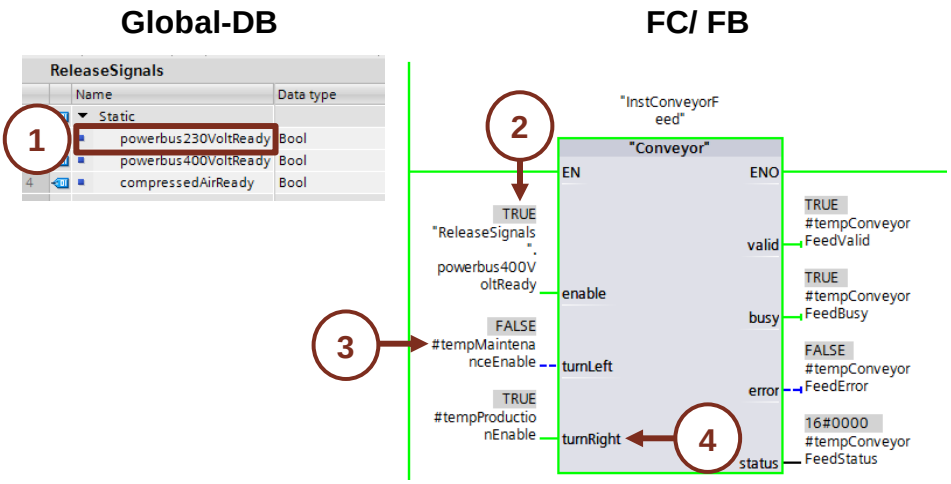


Tabelle 2-3

	Begriff	Erklärung
1.	Variable	Variablen werden durch einen Bezeichner deklariert und belegen eine Adresse im Speicher in der Steuerung. Variablen werden immer mit einem bestimmten Datentyp (Bool, Integer, usw.) definiert: • PLC-Variablen oder Anwenderkonstanten • Variablen oder Konstanten in Bausteinen • Variablen von Strukturen ("STRUCT") und PLC-Datentypen • Datenbausteine/ Instanz-Datenbausteine • Technologieobjekte
2.	Aktualwerte	Aktualwerte sind Werte, die in einer Variablen gespeichert sind (z. B. 15 als Wert einer Integer-Variablen)
3.	Aktualparameter	Aktualparameter sind Variablen, die an den Formalparametern von Bausteinen verschaltet sind.
4.	Formalparameter	Formalparameter sind die Schnittstellen von Bausteinen für Aufrufe im Programm. Formalparameter werden oft auch "Übergabeparameter" oder "Bausteinparameter" genannt. Diese Begriffe sind jedoch zu vermeiden.

Die Bausteinschnittstelle besteht aus zwei Teilen, den Formalparametern und den Lokaldaten.

Formalparameter

Tabelle 2-4

Typ	Abschnitt	Funktion
Eingangsparameter	Input	Parameter, deren Werte der Baustein liest.
Ausgangsparameter	Output	Parameter, deren Werte der Baustein schreibt.
Durchgangsparameter	InOut	Parameter, deren Werte der Baustein beim Aufruf liest und nach der Bearbeitung wieder in denselben Parameter schreibt.
Rückgabewert	Return	Wert, der an den aufrufenden Baustein zurückgeliefert wird.

Lokaldaten

Tabelle 2-5

Typ	Abschnitt	Funktion
Temporäre Variablen	Temp	Variablen, die zum Speichern von temporären Werten/ Zwischenergebnissen dienen.
Statische Variablen	Static	Variablen, die zum Speichern von statischen Werten/ Zwischenergebnissen im Instanz-Datenbaustein dienen.
Konstanten	Constant	Konstanten mit deklariertem symbolischem Bezeichner, die innerhalb des Bausteins verwendet werden.

3 Einstellungen in TIA Portal

Dieses Kapitel beschreibt die Regeln und Empfehlungen für die Grundeinstellungen in der Programmierumgebung TIA Portal.

Hinweis Die hier aufgezählten Regeln und Empfehlungen zu den Einstellungen in TIA Portal sind im TIA Portal Settings File (tps-File) gespeichert. Sie finden das tps-File als separaten Download im Beitrag. Um die Einstellungen zu übernehmen, können Sie das tps-File in TIA Portal importieren.

ES001 Regel: Oberflächensprache "English"

Die Oberflächensprache in TIA Portal wird auf "English" eingestellt. Dadurch werden alle neu angelegten Projekte automatisch in der Editier- und Referenzsprache sowie die Systemkonstanten in "English" angelegt.

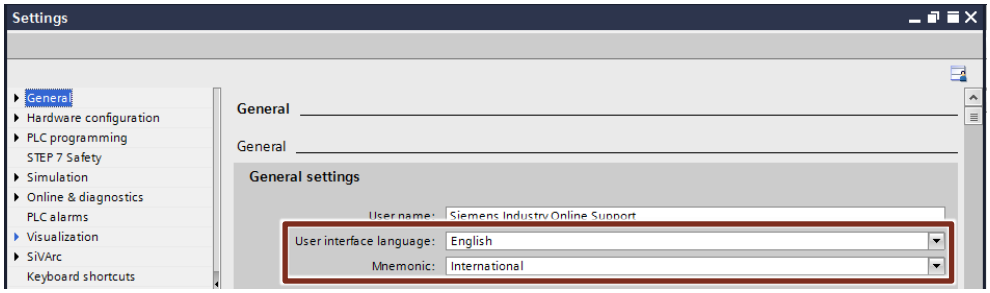
Begründung: Damit die Systemkonstanten in allen Projekten in derselben Sprache vorliegen, muss die Oberflächensprache einheitlich eingestellt sein.

ES002 Regel: Mnemonik "International"

Die Mnemonik (Spracheinstellung für Programmiersprachen) wird auf "International" eingestellt.

Begründung: Alle Systemsprachen und Systemparameter sind somit eindeutig und sprachunabhängig. Das gewährleistet eine reibungslose Zusammenarbeit im Team.

Abbildung 3-1



ES003 Empfehlung: Nichtproportionale Schriftart für Editoren

Für die Editoren empfiehlt es sich eine nichtproportionale Schriftart (Monospace Font) zu nutzen. Dadurch werden alle Zeichen in derselben Breite dargestellt und der Code, Wörter sowie Einrückungen sind gleichmäßig in ihrer Verteilung.

Die empfohlene Einstellung ist "Consolas" mit Schriftgröße 10 pt.

Begründung: Im Gegensatz zu "Courier New" wird bei "Consolas" verstärkt Wert auf die Unterscheidbarkeit ähnlicher Zeichen gelegt. Sie wurde vor allem für die Programmierung entworfen.

Abbildung 3-2

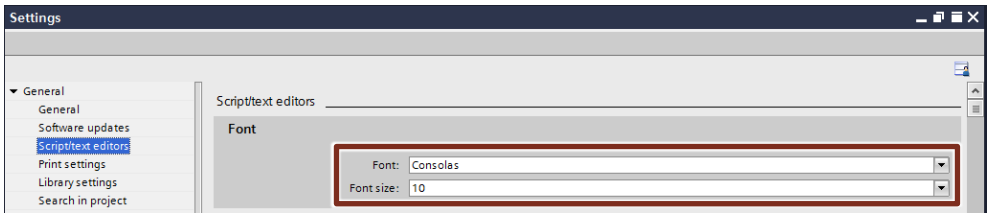


Abbildung 3-3

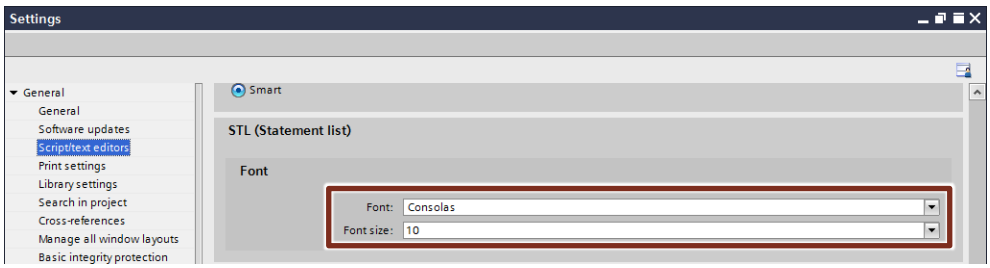
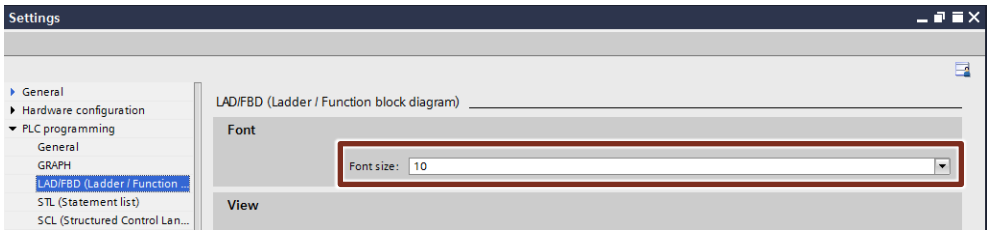


Abbildung 3-4

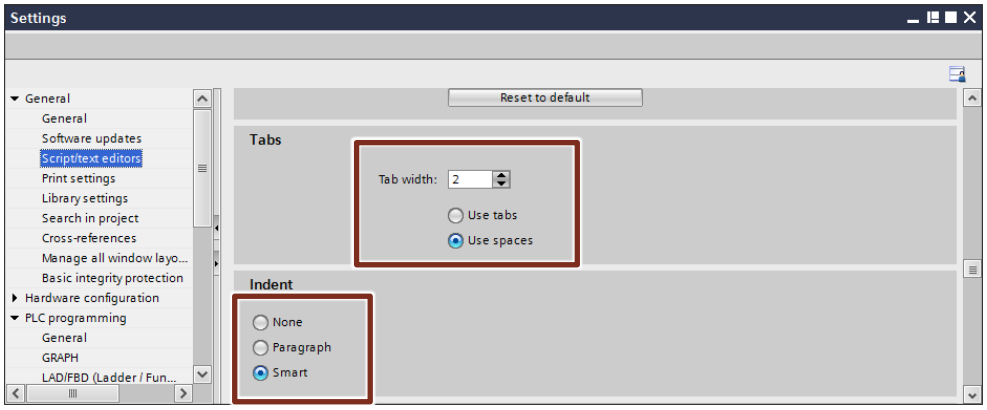


ES004 Regel: Smarte Einrückung mit zwei Leerzeichen

Für die Einrückung der Anweisungen werden zwei Leerzeichen und die Option "Indent" mit der Einstellung "Smart" verwendet. Tabulatorzeichen sind in den textuellen Editoren nicht zugelassen, da ihre Breite in verschiedenen Editoren unterschiedlich interpretiert und dargestellt wird.

Begründung: Damit wird ein einheitliches Aussehen in unterschiedlichen Editoren sichergestellt.

Abbildung 3-5

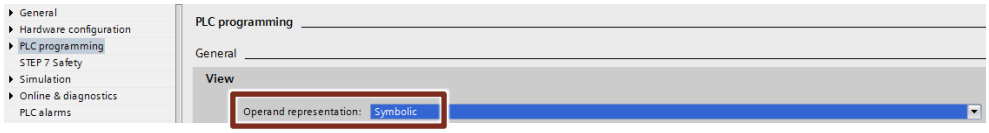


ES005 Regel: Symbolische Repräsentation von Operanden

Die Repräsentation der Operanden wird auf symbolisch festgelegt.

Begründung: Es wird vollsymbolisch programmiert.

Abbildung 3-6



ES006 Regel: IEC-konforme Programmierung

Um die IEC-konforme Programmierung zu gewährleisten, wird die IEC-Prüfung für die Bausteinerstellung standardmäßig aktiviert.

Begründung: Dadurch wird bei der Bausteinerstellung sichergestellt, dass der IEC-Check aktiviert ist und anschließend bei der Programmierung eine typ-konforme und typsichere Verwendung von Variablen gewährleistet wird.

Abbildung 3-7

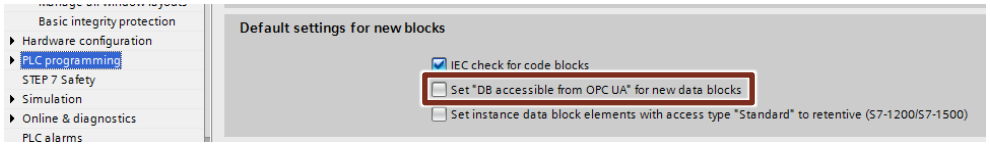
ES007 Regel: Expliziter Datenzugriff per HMI/ OPC UA/ Web API

Um die Zugriffsrechte von externen Applikationen wie HMI/ OPC UA/ Web API nicht grundsätzlich zuzulassen, werden diese in den Standardeinstellungen deaktiviert.

Begründung: Dadurch wird erreicht, dass nur Datenbereiche für externe Zugriffe freigeschaltet werden, bei denen diese explizit freigegeben werden.

Abbildung 3-8

Abbildung 3-9



ES008 Regel: Automatische Wertprüfung (ENO) aktiviert

Für die automatische Prüfung von Typgrenzen und Operationen ist der EN/ ENO Mechanismus zuständig. Dieser wird standardmäßig aktiviert.

Begründung: Somit werden die Wertprüfungen automatisch vom System durchgeführt, siehe dazu auch "SE003 Regel: ENO behandeln".

ES009 Regel: Automatische Prüfung von Arraygrenzen

Die automatische Prüfung der Arraygrenzen ist standardmäßig zu aktivieren.

Begründung: Die Einhaltung der Arraygrenzen werden bereits zum Compile-Zeitpunkt geprüft und somit Bereichslängenfehler im laufenden Betrieb verhindert.

4 Globalisierung

Dieses Kapitel beschreibt die Regeln und Empfehlungen für eine globale Zusammenarbeit.

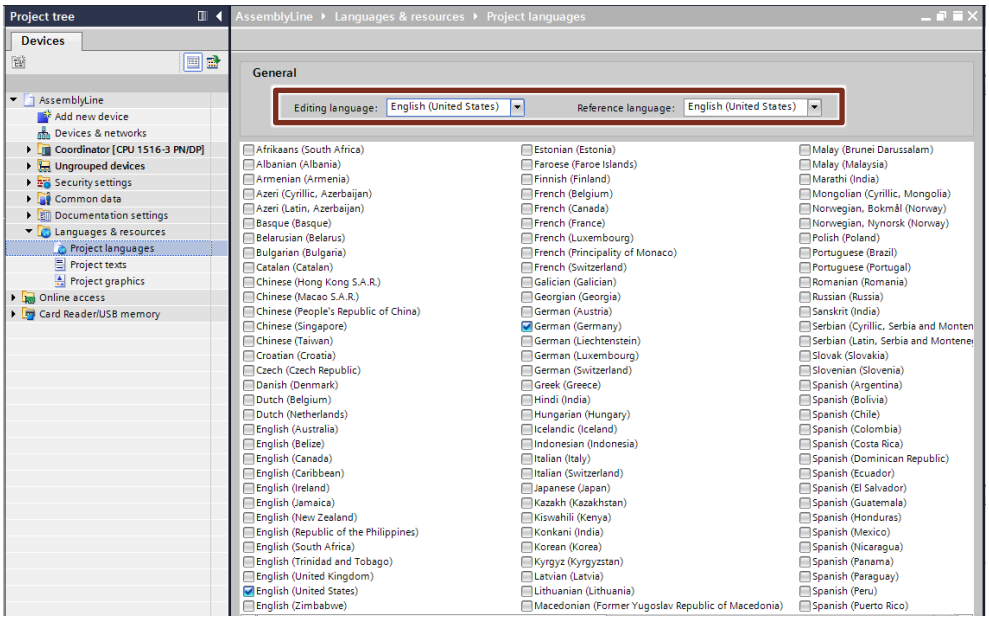
GL001 Regel: Einheitliche Sprache verwenden

Die Sprache muss sowohl in der PLC- als auch in der HMI-Programmierung immer konsistent sein. Das heißt, dass z. B. nur englische Texte im englischen Sprachbereich zu finden sind.

GL002 Regel: Editier- und Referenzsprache "English (US)" setzen

Wenn nicht ausdrücklich vom Auftraggeber anders gewünscht, ist die Sprache "English (United States)" als Editier- und Referenzsprache zu verwenden. Das komplette Programm inklusive aller Kommentare ist mindestens in Englisch zu verfassen.

Abbildung 4-1

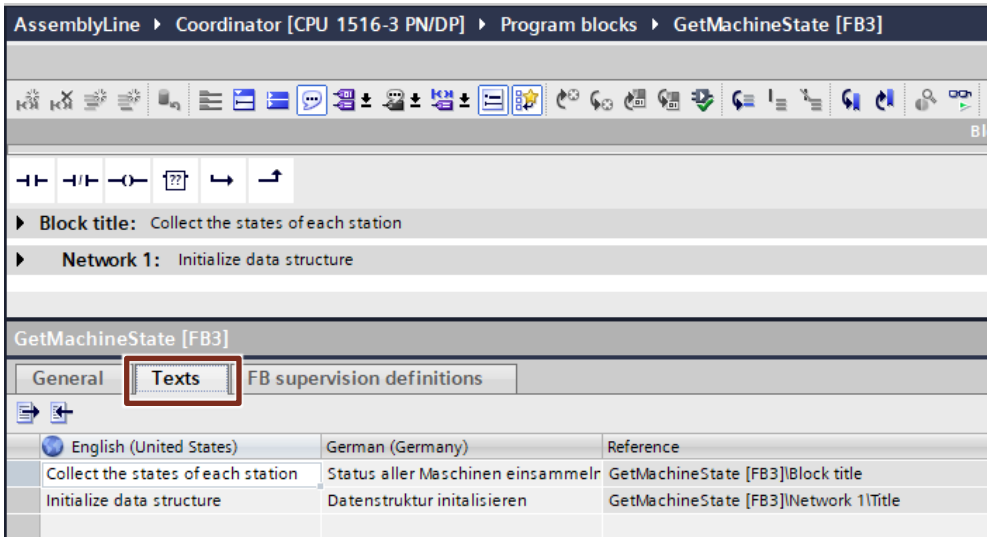


GL003 Regel: Texte in allen Projektsprachen hinterlegen

Projekttexte müssen immer mindestens in Englisch vorhanden sein, sowie in allen anderen verwendeten Projektsprachen in der entsprechenden Sprache.

Hinweis In einem Bausteineditor können über den Reiter "Texte" die Übersetzungen leicht verwaltet werden.

Abbildung 4-2



5 Nomenklatur und Formatierung

Dieses Kapitel beschreibt die Regeln und Empfehlungen für die Namensgebung und Schreibweise.

NF001 Regel: Eindeutig und einheitlich in Englisch bezeichnen

Der Name in Bezeichnern (Bausteine, Variablen etc.) ist in Englischer Sprache (English – United States) zu verfassen. Der Name gibt den Sinn und Zweck des Bezeichners im Kontext des Quellcodes wieder und lässt damit einen Rückschluss auf dessen Funktionalität bzw. Verwendung zu.

- Die gewählte Schreibweise der Bezeichner muss in allen Bausteinen und PLC-Datentypen beibehalten werden und ist so kurz wie möglich zu gestalten.
- Gleiche funktionale Bedeutung erhält namensgleiche Bezeichner. Dies gilt auch bezüglich Groß- und Kleinschreibung.
- Für Bezeichner, die aus mehreren Worten bestehen, ist die Reihenfolge der Worte wie die des gesprochenen Worts zu wählen.
- Funktionen/ Funktionsbausteine beginnen nach Möglichkeit mit einem Verb, z. B. "Get", "Set", "Put", "Find", "Search", "Calc".
- Ist der Bezeichner ein Arraybezeichner, ist er im Plural zu verwenden. Unzählbare Nomen bleiben im Singular ("data", "information", "content", "management").
- Statische und temporäre Boolesche Variablen sind häufig zustandsanzeigende Variablen. In einem solchen Falle sind Namen mit "is", "can" oder "has" am eingängigsten und verständlichsten.

Begründung: Ein schneller Überblick über das Programm und die Ein-/Ausgänge wird gewährleistet.

Hinweis Die von TIA Portal vorgeschlagenen Namen sind Platzhalter, die Sie mit Ihren Bezeichnern ersetzen müssen.

Tabelle 5-1

	Korrekte Namen	Falsche Namen
Für Arrays	data beltConveyors	datas beltConveyor
Für statische und temporäre Boolesche Variablen, die einen Zustand anzeigen	isConnected canScan	connected scan
Für weitere Boolesche Variablen	enable	setEnable
Für Funktionen/ Funktionsbausteine	GetMachineState SearchDevices	MachineStateFC FB_Device

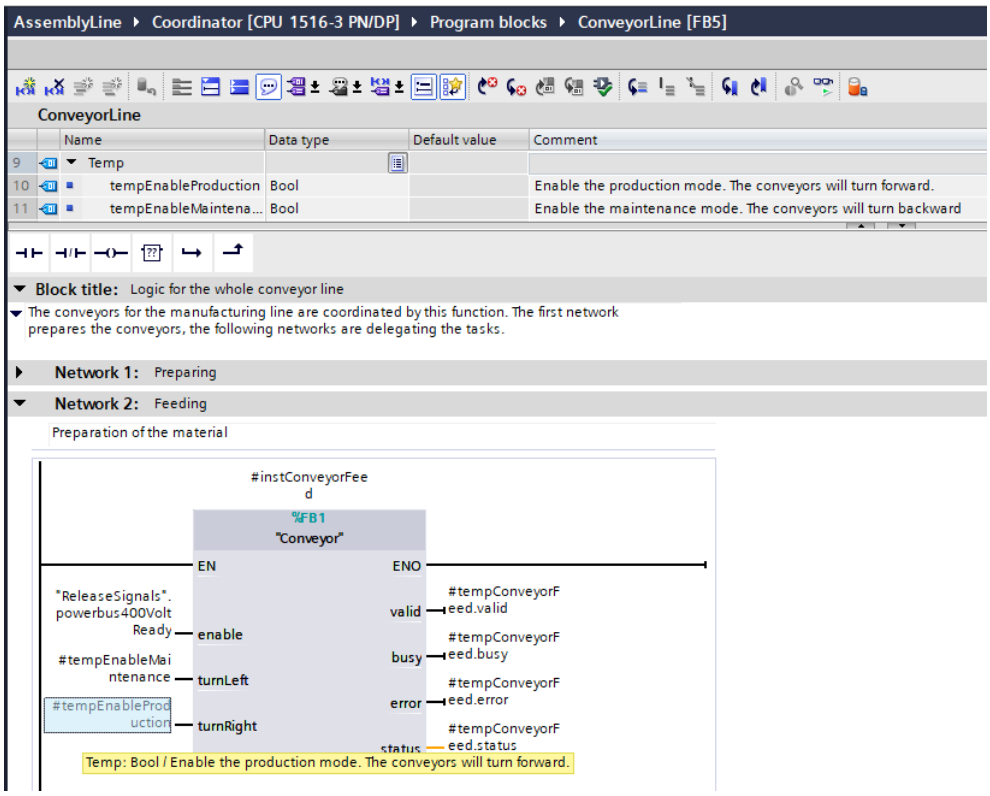
NF002 Regel: Sinnvolle Kommentare & Eigenschaften verwenden

Kommentarfelder und Eigenschaftsfelder sind zu verwenden und mit sinnvollen Kommentaren und Information zu füllen. Dazu zählen z. B.

- Bausteintitel und Bausteinkommentar (siehe dazu auch "NF003 Regel: Entwicklerinformationen dokumentieren")
- Bausteinschnittstellen
- Netzwerktitel und Netzwerkkommentare
- Bausteine, deren Variablen und Konstanten
- PLC-Datentypen und deren Variablen
- PLC-Variablentabellen, PLC-Variablen und Anwenderkonstanten
- PLC-Meldetextlisten
- Bibliothekseigenschaften

Begründung: Dadurch wird dem Anwender möglichst viel Information und Hilfestellungen bei der Verwendung zur Verfügung gestellt, z. B. durch die Tooltips.

Abbildung 5-1



Hinweis

Zusätzlich kann die Bausteinbeschreibung als Dokument (z. B. *.pdf, *.html) im TIA Portal-Projekt hinterlegt werden. Diese Beschreibung kann der Anwender bei Bedarf mit der Tastenkombination <Shift> <F1> als Online-Hilfe aufrufen.

Weitere Informationen dazu finden Sie in der Online-Hilfe unter dem Stichwort "Anwenderdefinierte Dokumentation bereitstellen":

https://support.industry.siemens.com/cs/ww/de/view/109773506/119453612171

NF003 Regel: Entwicklerinformationen dokumentieren

Jeder Baustein erhält einen Beschreibungskopf im Programmcode (SCL/ ST) bzw. im Blockkommentar (KOP, FUP). Darin müssen die wichtigsten Informationen zur Bausteinentwicklung hinterlegt werden. Durch die Platzierung im Programmcode werden diese entwicklerrelevanten Informationen bei Know-how geschützten Bausteinen verborgen.

Anwenderrelevante Informationen müssen hingegen in die Bausteineigenschaften eingetragen bzw. übernommen werden. Diese Informationen sind auch bei Know-how geschützten Bausteinen für den Anwender sichtbar.

Die nachfolgende Schablone für einen Beschreibungskopf enthält sowohl die Elemente aus den Baustein-Eigenschaften als auch entwicklerspezifische Elemente, die nicht in die Eigenschaften übernommen werden müssen.

Die Beschreibung enthält folgende Punkte:

- (Optional) Name der Firma / (C) Copyright (Jahr). All rights reserved.
- Titel/ Bausteinbezeichner
- Beschreibung der Funktionalität
- (Optional) Name der Bibliothek
- Abteilung/ Autor/ Kontakt
- Zielsystem – PLCs mit Firmware-Version (z. B. 1516-3 PN/DP V2.6)
- Engineering – TIA Portal mit Version bei Erstellung/ Änderungen
- Einschränkungen bei der Benutzung (z. B. bestimmte OB-Typen)
- Voraussetzungen (z. B. zusätzliche Hardware)
- (Optional) zusätzliche Informationen
- (Optional) Änderungslog mit Version, Datum, Autor und Änderungsbeschreibung (bei Safety-Bausteinen Safety-Signatur)

Vorlage für einen Beschreibungskopf in KOP und FUP:

```
(company) / (C) Copyright (year)
-------------------------------------------------------------------------------
Title:            (Title of this block)
Comment/Function: (that is implemented in the block)
Library/Family:   (that the source is dedicated to)
Author:           (department / person in charge / contact)
Target System:    (test system with FW version)
Engineering:      TIA Portal (SW version)
Restrictions:     (OB types, etc.)
Requirements:     (hardware, technological package, etc.)
-------------------------------------------------------------------------------
Change log table:
Version     | Date       | Signature  | Expert in charge | Changes applied
------------|------------|------------|------------------|-----------------------
001.000.000 | yyyy-mm-dd | 0x47F78CC1 | (name of expert) | First released version
```

Vorlage für einen Beschreibungskopf in SCL:

```
REGION Description header
  //================================================================================
  // (company) / (C) Copyright (year)
  //--------------------------------------------------------------------------------
  // Title:            (Title of this block)
  // Comment/Function: (that is implemented in the block)
  // Library/Family:   (that the source is dedicated to)
  // Author:           (department / person in charge / contact)
  // Target System:    (test system with FW version)
  // Engineering:      TIA Portal (SW version)
  // Restrictions:     (OB types, etc.)
  // Requirements:     (hardware, technological package, etc.)
  //--------------------------------------------------------------------------------
  // Change log table:
  // Version     | Date       | Expert in charge | Changes applied
  //-------------|------------|------------------|-----------------------------------
  // 001.000.000 | yyyy-mm-dd | (name of expert) | First released version
  //================================================================================
END_REGION
```

NF004 Regel: Präfixe und Struktur für Bibliotheken einhalten

Der Bibliotheksbezeichner hat das Präfix "L" und die Gesamtlänge umfasst maximal acht Zeichen

Der Bezeichner einer Bibliothek erhält das Präfix "L" plus maximal sieben Zeichen für den Namen (z. B. LGF, LCom). "L" steht für das Wort Library. Nach dem Bibliothekspräfix wird ein Unterstrich zur Trennung verwendet (z. B. LGF_).

Die maximale Länge des Bezeichners für einen Bibliotheksnamen (inklusive Präfix) wird auf acht Zeichen begrenzt.

Begründung: Diese Beschränkung dient zur kompakten Namensvergabe.

Alle in einer Bibliothek vorhandenen Elemente tragen das Präfix

Alle in einer Bibliothek vorhandenen Typen und Kopiervorlagen erhalten den Bezeichner der Bibliothek.

Ein Element, das nur die Verwendung der Bibliothek zeigt, ist kein Bibliothekselement im Sinne der standardisierenden Bibliothek, sondern ein Beispiel und trägt deshalb auch nicht zwingend das Präfix.

Begründung: Dadurch werden Kollisionen bei Bezeichnern vermieden.

Tabelle 5-2

Typ	Bezeichner nach Styleguide
Bibliothek, Hauptordner der Bibliothek	LExample
PLC-Datentyp	LExample_type<Name>
Funktionsbaustein	LExample_<Name>
Funktion	LExample_<Name>
Globaldatenbaustein	LExample_<Name>
Organisationsbaustein	LExample_<Name>
PLC-Variablentabelle	LExample_<Name>
Globale Konstante	LEXAMPLE_<NAME>

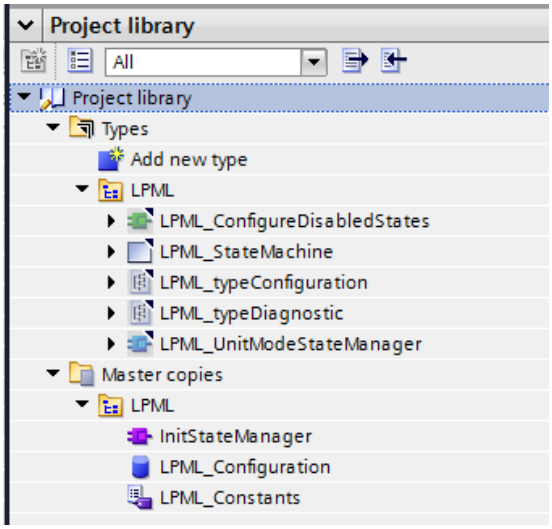
Typ	Bezeichner nach Styleguide
Globale Konstante für Fehler	LEXAMPLE_ERROR_<NAME> LEXAMPLE_ERR_<NAME>
Globale Konstante für Warnung	LEXAMPLE_WARNING_<NAME> LEXAMPLE_WARN_<NAME>
PLC-Meldetextliste	LExample_<Name>

Gruppieren in der Bibliothek

Alle Kopiervorlagen und Typen werden in einem Unterordner in der Bibliothek abgelegt, der den Bibliotheksbezeichner als Ordnerbezeichner trägt.

Begründung: Der Unterordner dient zum Zweck der Projektharmonisierung und zum Gruppieren von mehreren unabhängigen Bibliotheken in einem Projekt.

Abbildung 5-2



Hinweis Diese Regel enthält nur Angaben zur Nomenklatur für Bibliothekselemente. Ergänzend hierzu wird die Beachtung des Bibliothekleitfadens empfohlen, in dem der Umgang sowie die Verwendung von Bibliotheken ausführlich behandelt wird:

https://support.industry.siemens.com/cs/ww/de/view/109747503

NF005 Regel: Objekte in PascalCasing bezeichnen

Bezeichner von TIA Portal-Objekten, z. B.

- Bausteine
- Software Units, Technologieobjekte, Bibliotheken, Projekte
- PLC-Variablentabellen
- PLC-Meldetextlisten
- Beobachtungs- und Forcetabellen
- Traces und Messungen

werden in PascalCasing geschrieben.

Folgende Regeln gelten für das PascalCasing:

- Das erste Zeichen ist ein Buchstabe und wird groß geschrieben.
- Besteht ein Bezeichner aus mehreren Worten, wird der Anfangsbuchstabe der einzelnen Worte groß geschrieben.
- Es werden keine Trennzeichen (z. B. Binde- oder Unterstriche) für die optische Begriffstrennung verwendet. Zur Strukturierung und Spezialisierung von wiederkehrenden Strukturen bei Objektbezeichnern ist die maßvolle Nutzung von Unterstrichen (maximal drei) erlaubt.

Tabelle 5-3

Maßvoll	Nicht maßvoll
GetAxisData_PosAxis GetAxisData_SpeedAxis GetAxisData_SyncAxis	Get_Axis_Data_Pos_Axis

NF006 Regel: Codeelemente in camelCasing bezeichnen

Bezeichner von Codeelementen, z. B.

- Variablen
- PLC-Datentypen
- Strukturen ("STRUCT")
- PLC-Variablen
- Parameter

werden in camelCasing geschrieben.

Folgende Regeln gelten für das camelCasing:

- Das erste Zeichen ist ein Buchstabe und wird klein geschrieben.
- Besteht ein Bezeichner aus mehreren Worten, wird der Anfangsbuchstabe der folgenden einzelnen Worte groß geschrieben.
- Es werden keine Trennzeichen (z. B. Binde- oder Unterstriche) für die optische Begriffstrennung verwendet.

Abbildung 5-3

Conveyor

		Name	Datentyp
1		Input	
2		enable	Bool
3		turnLeft	Bool
4		turnRight	Bool
5		Output	
6		valid	Bool
7		busy	Bool
8		error	Bool
9		status	Word

NF007 Regel: Präfixe verwenden

Kein Präfix für Formalparameter

Formalparameter von Bausteinen werden ohne Präfixe verwendet. Auch bei der Übergabe eines PLC-Datentyps wird kein Präfix vergeben.

Temporäre und statische Variablen mit Präfix "temp" bzw. "stat"

Um statische und temporäre Variablen klar von Formalparametern im Code zu trennen, werden die Präfixe aus Tabelle 5-4 verwendet.

Begründung: Dies erleichtert dem Programmierer eines Bausteins die Unterscheidung zwischen Formalparametern und Lokaldaten. Dadurch können sofort die Zugriffsrechte für den Benutzer definiert und erkannt werden.

Hinweis Die statischen Variablen in Global-DBs und Array-DBs erhalten nicht das Präfix "stat".

Instanzdaten mit Präfix "inst" bzw. "Inst"

Sowohl Einzelinstanzen als auch Multiinstanzen und Parameterinstanzen erhalten ein Präfix. Bei Einzelinstanzen wird ein "Inst", bei Multi- und Parameterinstanzen ein "inst" vorangestellt.

Begründung: Es kann einfach erkannt werden, ob ein (unzulässiger) Zugriff auf Instanzdaten programmiert wurde.

PLC-Datentypen mit Präfix "type"

Einem PLC-Datentyp wird das Präfix "type" vorangestellt. Die Elemente im PLC-Datentyp erhalten wiederum kein Präfix.

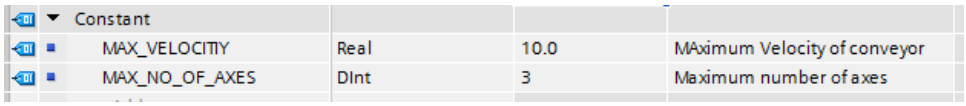
Tabelle 5-4

Präfix	Typ
Kein Präfix	**Eingangs- und Ausgangsparameter** Zugriff von außerhalb möglich ➔ enable ➔ error
Kein Präfix	**Durchgangsparameter** Änderung der verschalteten Daten sowohl durch Benutzer als auch durch Baustein jederzeit möglich ➔ conveyorAxis
Kein Präfix	**PLC-Variablen und Anwenderkonstanten** ➔ lightBarrierLeft (Bezeichner für PLC-Variable) ➔ MAX_BELTS (Bezeichner für Anwenderkonstante)
Kein Präfix	**Global-Datenbausteine** Weder der Global-DB noch die enthaltenen Elemente erhalten ein Präfix ➔ ReleaseSignals (Bezeichner für Global-DB) ➔ powerBusReady (Bezeichner für Variable in Global-DB)
temp	**Temporäre Variablen** Kein Zugriff auf die Lokaldaten von außerhalb möglich ➔ tempIndex
stat	**Statische Variablen** Kein Zugriff auf die Lokaldaten von außerhalb erlaubt ➔ statState
inst	**Variablen bei Multiinstanzen und Parameterinstanzen** ➔ instWatchdogTimer ➔ instWatchdogTimer**s** (bei Arrays von Instanzen)
Inst	**Einzelinstanz-Datenbausteine** ➔ InstConveyorFeed
type	**PLC-Datentyp** Nur der PLC-Datentyp erhält das Präfix, nicht die enthaltenen Elemente ➔ typeDiagnostic (Bezeichner für PLC-Datentyp) ➔ stateNumber (Bezeichner für Variable in PLC-Datentyp)

NF008 Regel: Bezeichner von Konstanten GROSS schreiben

Die Namen von Konstanten (globale und lokale Konstanten) werden durchgehend in Großschrift geschrieben (UPPER_CASING). Zum Trennen und Erkennen einzelner Worte oder Abkürzungen ist jeweils ein Unterstrich zwischen den einzelnen Worten oder Abkürzungen einzusetzen.

Abbildung 5-4: Konstanten in einem FB



Hinweis "TRUE" und "FALSE" sind ebenfalls Konstanten.

NF009 Regel: Zeichensatz für Bezeichner einschränken

Für Objekt- und Code-Bezeichner werden ausschließlich lateinische Buchstaben (a-z, A-Z), arabische Ziffern (0-9) sowie der Unterstrich (_) verwendet.

Tabelle 5-5

Korrekt bezeichnet	Falsch bezeichnet
tempMaxLength	temporäre Variable 1

NF010 Empfehlung: Zeichenlänge für Bezeichner einschränken

Die Gesamtlänge eines Bezeichners inkl. Präfix, Suffix oder Bibliotheksbezeichnung soll 24 Zeichen nicht überschreiten.

Begründung: Da Variablennamen aus Strukturen sich aus mehreren Bezeichnern zusammensetzen, wird sich in solchen Fällen ohnehin eine entsprechend lange Variablenbezeichnung an der Verwendungsstelle ergeben.

Beispiel:

```
instFeedConveyor024.releaseTransportSect1.gappingTimeLeft
```

NF011 Empfehlung: Nur eine Abkürzung pro Bezeichner nutzen

Mehrere Abkürzungen sollten nicht direkt hintereinander verwendet werden, um eine optimale Lesbarkeit zu gewährleisten. Um Zeichen bei einem Variablennamen zu sparen und die Lesbarkeit des Programms zu erhöhen, können die Abkürzungen aus Tabelle 5-6 verwendet werden.

Die Tabelle stellt nur eine gängige Auswahl dar. Die Schreibweise der Abkürzungen muss den Regeln für die jeweilige Verwendung entsprechend angepasst werden (Groß-/ Kleinschreibung).

Tabelle 5-6

Abk.	Typ
Min	Minimum
Max	Maximum
Act	Actual, Current
Next	Next value
Prev	Previous value
Avg	Average
Sum	Total sum
Diff	Difference
Cnt	Count
Len	Length
Pos	Position
Ris	Rising edge
Fal	Falling edge
Old	Old value (z. B. für Flankenerkennung)
Sim	Simulated
Dir	Direction
Err	Error
Warn	Warning
Cmd	Command
Addr	Address

NF012 Regel: Im konformen Format initialisieren

Die Initialisierung (Zuweisung von konstanten Daten) erfolgt in dem gebräuchlichen, konformen Format/ Darstellung ihres Datentyps (Literal). Somit wird eine Variable vom Typ WORD mit z. B. 16#0001 und nicht mit 16#01 initialisiert.

Bei Initialisierungen innerhalb des Bausteincodes sind lokale symbolische Konstanten zu verwenden, siehe dazu auch "RU005 Regel: Lokale symbolische Konstanten verwenden".

Tabelle 5-7

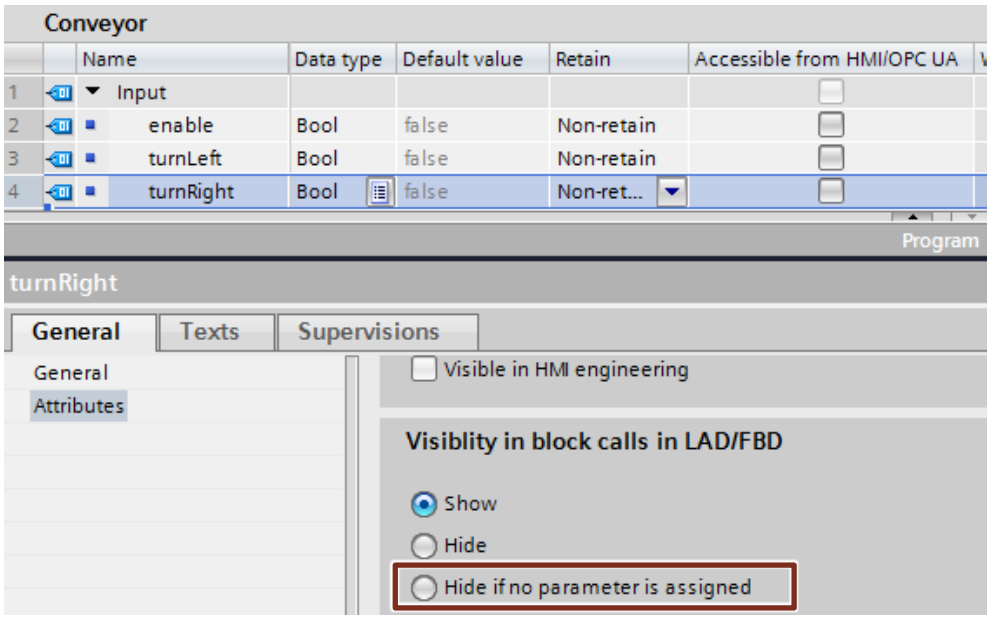
Korrekte Initialisierung			Falsche Initialisierung		
statTriggerOld	Bool	FALSE	statTriggerOld	Bool	false
statStatus	Word	16#7000	statStatus	Word	123
statStep	DInt	101	statStep	DInt	16#0
statVelocity	LReal	0.0	statVelocity	LReal	16#000
statCommand	Byte	16#01	statCommand	Byte	16#1
statFlags	Byte	2#1010_0101	statFlags	Byte	25

NF013 Empfehlung: Optionale Formalparameter ausblenden

Blenden Sie Formalparameter, die optional zu parametrieren sind, aus.

Begründung: So kann der Bausteinaufruf durch Einklappen der Bausteinansicht auf ein notwendiges Minimum reduziert werden.

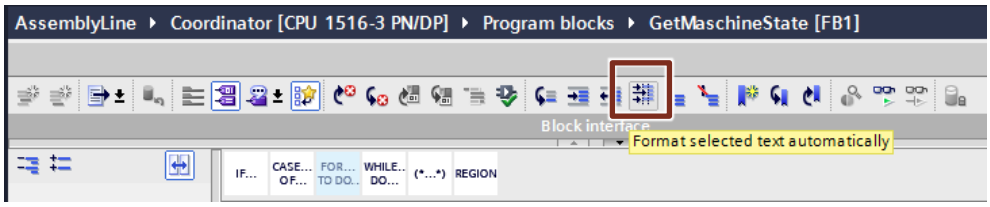
Abbildung 5-5



NF014 Regel: SCL-Code sinnvoll formatieren

Es wird empfohlen die Autoformat-Funktion von TIA Portal verwenden. Der Vorteil daraus ist, dass alle Benutzer mit einer einheitlichen Formatierung arbeiten und das Einrücken automatisch vorgenommen wird.

Abbildung 5-6



Nur Zeilenkommentar "//" verwenden

Es ist zwischen zwei Arten von Kommentaren zu unterscheiden:

- Ein Blockkommentar "(*…*)" oder ein mehrsprachiger Blockkommentar "(/*…*/)" ist in einer oder mehreren Zeilen vor dem entsprechenden Code-Abschnitt zu setzen und beschreibt eine Funktion, oder einen Code-Abschnitt.
- Ein Zeilenkommentar "//" beschreibt den Code einer einzelnen Zeile und ist, wenn möglich, an das Ende der Code-Zeile zu setzen, ansonsten vor die betreffende Code-Zeile.

Um das Auskommentieren von Codebereichen beim Debugging zu erleichtern, werden im PLC-Code nur Zeilenkommentare mit "//" verwendet.

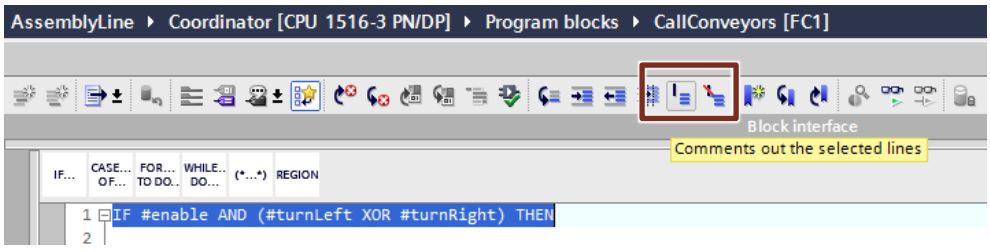
Ein Kommentar liefert dem Leser Information darüber, warum etwas an der betreffenden Stelle getan wird. Er darf nicht den Code als redundanten Klartext wiedergeben – also nicht was getan wird – das beschreibt bereits der Code – sondern warum es getan wird.

Hinweis

Sie können auch den Button "Auskommentieren" verwenden. Damit brauchen Sie die Kommentarzeichen nicht von Hand eintippen.

Das Engineering-System unterstützt hierbei über das Menü, um markierte Blöcke mit "//" zu versehen oder die Kommentierung zu löschen. Zum anderen vermeiden Sie Syntaxprobleme durch Einfügen und möglicherweise verschachtelte Kommentarabschnitte mit "(*...*)" oder "(/*...*/)".

Abbildung 5-7



Vor und nach Operatoren werden Leerzeichen gesetzt

Vor und nach Operatoren steht ein Leerzeichen.

Ausdrücke sind immer in Klammern

Um die Reihenfolge der Interpretation eindeutig zu machen, werden Ausdrücke entsprechend der gewünschten Interpretationsreihenfolge geklammert.

Beispiel:

```
#tempSetFlag := (#tempPositionAct < #MIN_POS)
                OR (#tempPositionAct > #MAX_POS);
```

Bedingung und Anweisungsteil werden mit Zeilenumbruch getrennt

Es ist eine klare Trennung zwischen Bedingung und Anweisungsteil herzustellen. Das heißt nach einer Bedingung (z. B. THEN) oder nach einem Alternativzweig (z. B. ELSE) muss immer ein Zeilenumbruch erfolgen, bevor eine Anweisung programmiert wird. Entsprechend gelten diese Regeln auch für den Umgang mit Bedingungen anderer Anweisungen (z. B. CASE, FOR, WHILE, REPEAT).

Beispiel:

```
IF #isConnected THEN // Comment
  ; // Statement section IF
ELSE
  ; // Statement section ELSE
END_IF;
```

Zeilenumbrüche bei Teil-Bedingungen

Bei komplexen Ausdrücken ist es sinnvoll, jede Teil-Bedingung durch einen Zeilenumbruch hervorzuheben. Operatoren werden dabei ebenfalls umgebrochen und an den Anfang der Teil-Bedingung gestellt.

Beispiel:

```
#tempResult := (#enable AND #tempEnableOld)
                OR (#enable AND #isValid
                  AND NOT (#hasError OR #hasWarning)
                );
```

Bedingungen und Anweisungen richtig einrücken

Jede Anweisung im Rumpf einer Kontrollstruktur wird eingerückt. Boolesche Verknüpfungen werden, wenn eine Zeile nicht für die gesamte Bedingung ausreicht, in einer neuen Zeile fortgesetzt.

Bei mehrzeiligen Bedingungen in IF-Anweisungen werden diese um zwei Leerzeichen eingerückt. THEN wird in eine eigene Zeile auf der gleichen Höhe wie IF platziert. Passen die Bedingungen einer IF-Anweisung in eine Zeile, kann THEN an das Zeilenende geschrieben werden. Falls tiefer geschachtelt wird, steht der Operand allein in einer Zeile. Eine alleinstehende Klammer zeigt das Ende der geschachtelten Bedingung. Operanden stehen immer am Anfang der Zeile.

Entsprechend gelten diese Regeln auch für den Umgang mit Bedingungen anderer Anweisungen (z. B. CASE, FOR, WHILE, REPEAT).

Beispiele:

```
IF #enable // Comment
  AND (
  (#turnLeft XOR #turnRight)
    OR (#statIsMaintenance AND #statIsManualMode)
  ) // Comment
  AND #tempIsConnected
THEN
  ; // Statement
ELSE
  ; // Statement
END_IF;
```

```
IF #enable THEN
  ; // Statement
  IF #tempIsReleased THEN
    ; // Statement
  END_IF;
ELSE
  ; // Statement
END_IF;
```

6 Wiederverwendbarkeit

Dieses Kapitel beschreibt die Regeln und Empfehlungen für die mehrfache Nutzung von Programmelementen.

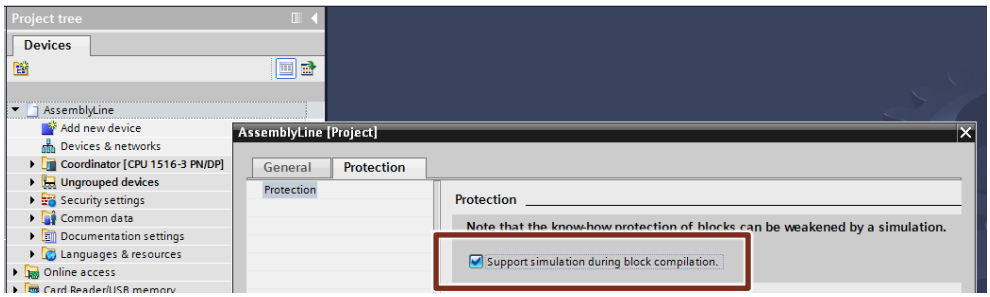
RU001 Regel: Simulierbare Bausteine bereitstellen

Aktivieren Sie die die Simulierbarkeit über die Projekteigenschaft.

Begründung: Damit wird eine vollständige und vollumfängliche Nutzbarkeit der Bausteine auch beim Simulieren sichergestellt.

Hinweis Beachten Sie, dass der Know-how-Schutz von Bausteinen durch eine Simulation geschwächt werden kann.

Abbildung 6-1



RU002 Regel: Vollständig mit Bibliotheken versionieren

Die Versionierung ist vollständig vorzunehmen. Das heißt, dass jegliche Änderungen am Baustein zu dokumentieren sind und die Versionsnummer zu pflegen ist. Es werden bei jeder Änderung der Version die Anpassungen an den entsprechenden Stellen, z. B. im Beschreibungskopf der Funktion, erklärt.

Unter Verwendung der Bibliothek und Bausteintypisierung wird die Bausteinversion zentral durch TIA Portal verwaltet. In diesem Fall ist es nicht notwendig, die Version manuell in den Bausteineigenschaften zu pflegen. Die Änderungshistorie bleibt hiervon unberührt.

Ein Hochrüsten der Bibliothek auf eine aktuelle TIA Portal-Version setzt keine Anpassung der Bausteine voraus und hat dementsprechend auch keine Versionsänderung zur Folge.

Hinweis Wird ein Baustein in eine Bibliothek eingefügt, müssen bereits vor dem Einfügen alle Eigenschaften des Bausteins, z. B. automatische Nummerierung, Know-how-Schutz, Simulierbarkeit (über die Projekteigenschaften) usw. gesetzt sein. Ist der Baustein einmal in einer Bibliothek, können die oben angeführten Eigenschaften nur mit Aufwand angepasst werden.

Die Eigenschaft "published" im Kontext der Software Units kann auch nachträglich editiert werden, ohne den Typ zu modifizieren.

Versionsnummern und deren Verwendung

Der erste freigegebene Stand beginnt mit der Version 1.0.0 (siehe Tabelle 6-1).

Die erste Stelle bezeichnet die Zahl ganz links.

Die dritte Stelle in der Software-Versionierung kennzeichnet Änderungen, die keine Auswirkung auf die Funktion und Dokumentation haben, z. B. reine Fehlerbehebungen.

Bei Erweiterung der bestehenden Funktionalität wird die zweite Stelle inkrementiert und die dritte Stelle zurückgesetzt.

Bei einer neuen Hauptversion, die neue Funktionalitäten aufweist und inkompatibel zu ihrer Vorgängerversion ist, wird die erste Stelle inkrementiert und die beiden letzten zurückgesetzt.

Jede Stelle hat einen Wertebereich von 0 bis 999.

Tabelle 6-1

Bibliothek	FB1	FB2	FC1	FC2	Kommentar
1.0.0	1.0.0	1.0.0	1.0.0		Freigegebener Stand
1.0.1	1.0.1	1.0.0	1.0.0		Fehlerbehebung von FB1
1.0.2	1.0.1	1.0.1	1.0.0		Optimierung von FB2
1.1.0	1.1.0	1.0.1	1.0.0		Erweiterung an FB1
1.2.0	1.2.0	1.0.1	1.0.0		Erweiterung an FB1
2.0.0	2.0.0	1.0.1	2.0.0		Neue und ggf. inkompatible Funktionalität an FB1 und FC1
2.0.1	2.0.0	1.0.2	2.0.0		Fehlerbehebung FB2
2.1.0 / 3.0.0	2.0.0	1.0.2	2.0.0	1.0.0	Neue Funktion FC2 / Ggf. größerer neuer Release

Typen für FC, FB und PLC-Datentypen verwenden

Wiederverwendbare Funktionen, Funktionsbausteine und PLC-Datentypen, die nicht vom Anwender angepasst werden müssen bzw. dürfen, werden als Typen in einer Bibliothek abgelegt.

Begründung: Damit können die Vorteile des Typenkonzepts in TIA Portal genutzt werden.

Hinweis Diese Regel enthält nur allgemeine Angaben zur Versionierung. Eine Erklärung für die automatische Versionierung von Bibliothekselementen ist im Bibliotheksleitfaden beschrieben:

https://support.industry.siemens.com/cs/ww/de/view/109747503

RU003 Regel: Nur freigegebene Typen in fertigen Projekten halten

Abgeschlossene Projekte enthalten keine typisierten Bibliothekselemente, die sich im Status "in test" befinden:

- Bausteine (nur Funktionen und Funktionsbausteine)
- PLC-Datentypen

RU004 Regel: Nur lokale Variablen verwenden

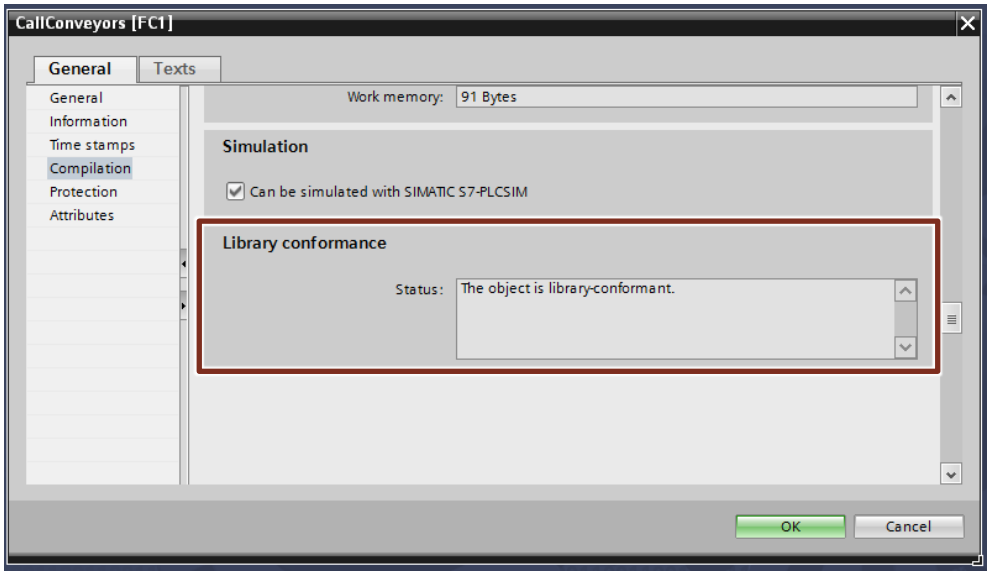
Innerhalb eines wiederverwendbaren Bausteins dürfen nur dessen lokale Variablen genutzt werden. Zugriffe auf globale Daten innerhalb von FCs und FBs sind nicht erlaubt. Globale Daten dürfen nur über die Formalparameter der Bausteinschnittstelle zu- oder abgeführt werden.

Von außen zu- oder abzuführen sind z. B.:

- Zugriff auf Global-DBs und Nutzung von Einzelinstanz-DBs
- Nutzung von globalen Zeiten und Zählern
- Nutzung von globalen Konstanten
- Zugriff auf PLC-Variablen

Werden die genannten Punkte eingehalten, setzt TIA Portal in den Eigenschaften des Bausteins automatisch den Status "Das Objekt ist bibliothekskonform" ("The object is library-conformant"). Dieser Status kann somit als einfache Überprüfungsmöglichkeit für die Einhaltung dieser Regel verwendet werden.

Abbildung 6-2



RU005 Regel: Lokale symbolische Konstanten verwenden

Im Sinne gekapselter Bausteine sind lokale Konstanten zu verwenden. Werden globale Konstanten verwendet, sind sie nach Möglichkeit über die Formalparameter zu übergeben. Globale Konstanten sind in eigenen PLC-Variablentabellen zu definieren.

Hinweis Wird in einem Baustein eine globale Konstante direkt verwendet und der Wert der Konstanten verändert, muss der Baustein neu übersetzt werden. Dies bedeutet, dass bei Know-how geschützten Bausteinen das Passwort bekannt sein muss.

Keine "Magic Numbers"

Wird eine Variable im Code mit einem Wert ungleich 0 (Integer), 0.0 (Real/ LReal), TRUE oder FALSE belegt oder verglichen, sind dafür symbolische Konstanten zu verwenden.

Begründung: Dadurch wird eine Änderung des Werts deutlich vereinfacht, da dieser nicht an mehreren Stellen im Code, sondern zentral in der Deklaration der Konstanten geändert wird.

Hinweis Konstanten sind textuelle Ersetzungen für numerische Werte, die vom Präprozessor ausgetauscht werden. Somit gibt es durch die Verwendung von Konstanten auf der PLC weder einen Performanceverlust, noch steigt der Speicherverbrauch im Arbeitsspeicher.

So können für gleiche Werte (auch gleich 0) mit unterschiedlicher Bedeutung jeweils eigene symbolische lokale Konstanten definiert werden.

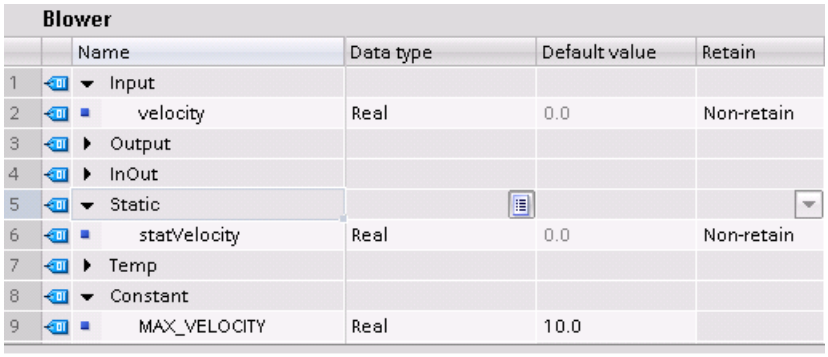
Beispiel:

```
STATUS_DONE             WORD    16#0000
STANDSTILL_SPEED        LREAL   0.0
FREEZING_TEMPERATURE    LREAL   0.0
```

Weiter werden auch die Lesbarkeit und Verständlichkeit erhöht, da ein symbolischer Bezeichner aussagekräftiger ist als eine Zahl.

Beispiel:

Abbildung 6-3




```
IF (#velocity < #MAX_VELOCITY) THEN
  #statVelocity := #velocity;
ELSE
  #statVelocity := #MAX_VELOCITY;
END_IF;
```

RU006 Regel: Vollsymbolisch programmieren

Es wird ausschließlich vollsymbolische Programmierung verwendet. Im Programm dürfen keine physikalischen Adressierungen, z. B. ANY-Pointer, verwendet werden.

Begründung: Das erhöht die Lesbarkeit und Wartbarkeit des Programms aufgrund der Symbolik deutlich.

Hinweis Die Alternative zu ANY ist VARIANT, optional mit Referenzen (REF_TO). Der Datentyp VARIANT erkennt Typfehler frühzeitig und bietet eine symbolische Adressierung.

RU007 Empfehlung: Hardwareunabhängig programmieren

Um die Kompatibilität zwischen den verschiedenen Systemen zu gewährleisten, wird empfohlen auf hardwareunabhängige Datentypen und Funktionalitäten zurückzugreifen.

Auf die Verwendung von globalen Merkern und Systemmerkern wird zu Gunsten der Wiederverwendbarkeit und Hardwareunabhängigkeit verzichtet.

Dazu zählen auch die systemseitigen Zähler und Zeiten, z. B. die S5-Timer. An Stelle dieser Datentypen werden die IEC-konformen Typen eingesetzt, z. B. der IEC-Timer, die auch als Multiinstanzen verwendbar sind.

Daten, die im gesamten Anwenderprogramm benötigt werden, werden in Global-Datenbausteinen gespeichert.

Hinweis Eine Vergleichstabelle der Systemfunktionen für die hardwareunabhängige Programmierung finden Sie im Siemens Industry Online Support unter folgendem Beitrag:

SIMATIC S7-1200/ S7-1500 Vergleichsliste für Programmiersprachen

https://support.industry.siemens.com/cs/ww/de/view/86630375

RU008 Empfehlung: Vorlagen verwenden

Durch die Verwendung von Vorlagen wird eine einheitliche Ausgangsbasis für alle Programmierer erreicht. Die durch die Vorlagen bereitgestellten Funktionalitäten können als validiert angesehen werden und tragen so erheblich zur Zeitersparnis bei.

Als weitere positive Aspekte sind die leichtere Nutzung und die verbesserte Außenwirkung durch eine einheitliche Bausteinbasis zu sehen, da die Bausteine aufgrund der Basis in ihrer Grundfunktionalität gleich funktionieren. Beispielhaft sei dazu auf den PLCopen-Standard verwiesen.

Hinweis Die hier erwähnten Vorlagen finden Sie als Kopiervorlagen in der Bibliothek mit generellen Funktionen (LGF):

https://support.industry.siemens.com/cs/ww/de/view/109479728

7 Referenzieren von Objekten (Allokieren)

Dieses Kapitel beschreibt die Regeln und Empfehlungen für die Speicherverwaltung und den Zugriff.

AL001 Regel: Multiinstanzen statt Einzelinstanzen nutzen

Im Programm sind vorrangig Multiinstanzen an Stelle von Einzelinstanzen zu verwenden.

Begründung: Dadurch können in sich geschlossene Module in Form von Funktionsbausteinen erstellt werden, z. B. ein FB mit integrierten Zeiten für eine Zeitüberwachung. Dadurch müssen keine zusätzlichen Instanzen in überlagerten oder globalen Strukturen angelegt werden.

AL002 Empfehlung: Arraygrenze von 0 bis Konstante definieren

Array-Grenzen beginnen mit 0 und enden mit einer Konstante für die Obergrenze des Arrays.

- Für Arrays in einem Baustein wird die Konstante in den Lokaldaten der Bausteinschnittstelle definiert.
- Für Arrays in Global-DBs und in PLC-Datentypen wird die Konstante in der PLC-Variablentabelle definiert.
- Für den Datentyp der Konstante sowie den Zugriff auf Arrayelemente (z. B. Laufvariable) ist aus Performancegründen DINT zu verwenden.

Beispiel:

```
BUFFER_UPPER_LIMIT   DINT   10


diagnostics          Array[0..BUFFER_UPPER_LIMIT] of typeDiagnostics
```

Begründung: Ein Array beginnend mit Index 0 ist sinnvoll, da bestimmte Systembefehle und mathematische Operationen null-basiert arbeiten, z. B. die Modulo-Funktion. Dadurch kann der gewünschte Index direkt in die Systemfunktion gegeben werden und muss nicht umgerechnet werden.

Ein weiterer Vorteil ist, dass auch WinCC (Comfort, Advanced, Professional und Unified) nur mit null-basierten Arrays arbeitet, z. B. für Skripte.

Wird dennoch mit nicht null-basierten Arrays gearbeitet, sollte ein Array mit jeweils einer symbolischen Konstante für die Unter- und Obergrenze deklariert werden.

AL003 Empfehlung: Array-Parameter als ARRAY[*] deklarieren

Wird ein Array über die Formalparameter übergeben, empfiehlt es sich diesen Parameter als Array mit unspezifizierter Größe zu deklarieren.

Die Größen und Grenzen werden innerhalb des Bausteins mit den Systemfunktionen "UPPER_BOUND" und "LOWER_BOUND" ermittelt und verarbeitet.

Beispiel:

```
diagnostics          Array[*] of typeDiagnostics
```

Begründung: Dadurch können generische Programmstrukturen aufgebaut werden, insbesondere auch mit Know-how geschützten Bausteinen, da die Arraygröße nicht explizit in der Bausteinschnittstelle festgelegt werden muss.

AL004 Empfehlung: Benötigte String-Länge festlegen

"String" und "WString" reservieren beim Anlegen standardmäßig immer die Länge von 254 Zeichen. Ein "String" kann bis zu 254 Zeichen enthalten, ein "WString" bis zu 16382 Zeichen. Es empfiehlt sich daher, soweit möglich, alle Zeichenfolgen per Konstante auf deren notwendige Länge zu begrenzen.

Begründung: Das verhindert zum einen unnötiges Allokieren von Speicher und zum anderen bringt es Performance-Vorteile bei der Übergabe von Zeichenfolgen über die Formalparameter.

Beispiel:

```
MAX_MESSAGE_LENGTH    DINT    24

errorMessage          String[#MAX_MESSAGE_LENGTH]
```

8 Sicherheit

Dieses Kapitel beschreibt die Regeln und Empfehlungen für das Gestalten eines möglichst sicheren, robusten Programmablaufs.

SE001 Regel: Aktualwerte auf Gültigkeit prüfen

Alle übergebenen Aktualwerte sind im Baustein auf deren Gültigkeit zu prüfen, um unkontrollierte Programmabläufe und Zustände zu verhindern.

Bei ungültigen oder nicht plausiblen Werten ist eine entsprechende Meldung für den Anwender bereit zu stellen, siehe auch "DA013 Regel: Status/ Fehler per "status"/ "error" zurückgeben".

SE002 Regel: Temporäre Variablen initialisieren

Alle im Baustein verwendeten temporären Variablen sind vor deren Verwendung mit einem entsprechenden Initialwert zu initialisieren. Die Initialisierung erfolgt direkt per Zuweisung einer Operation oder Konstanten, die in der gebräuchlichen Darstellung ihres Datentyps beschrieben werden (Literal).

Siehe auch "NF012 Regel: Im konformen Format initialisieren"

Begründung: Nur elementare Datentypen im optimierten Bereich werden vom System initialisiert, alle anderen haben einen undefinierten Initialwert.

Hinweis

Bei Variablen, die an Technologie-Bausteine verschaltet werden, haben Werte kleiner 0.0 eine besondere Bedeutung: Je nach Parameter wird stattdessen der im Technologieobjekt konfigurierte Standardwert verwendet.

Deshalb muss ein geeigneter Initialwert gewählt werden.

SE003 Regel: ENO behandeln

Mithilfe des Freigabeausgangs ENO können Sie bestimmte Laufzeitfehler erkennen und behandeln. Die Ausführung nachfolgender Anweisungen ist vom Signalzustand des Freigabeausgangs abhängig.

Begründung: Durch die Nutzung des EN/ ENO-Mechanismus wird vermieden, dass es zu Programmabbrüchen kommt. Der Bausteinstatus wird in Form einer Booleschen Variablen weitergegeben.

Um die Performance der PLC zu steigern, kann der automatische EN/ ENO-Mechanismus deaktiviert werden. Allerdings besteht dann nicht mehr die Möglichkeit auf Laufzeitfehler der Anweisung über den ENO-Wert zu reagieren. Die Freigabe nachfolgender Anweisungen muss daher manuell behandelt werden.

Deshalb muss in jedem Falle eine bewusste Zuweisung auf ENO im Programm vorhanden sein. Im einfachsten Falle:

```
ENO := TRUE;
```

SE004 Regel: Datenzugriff per HMI/ OPC UA/ Web API selektiv aktivieren

Standardmäßig ist der Zugriff per HMI/ OPC UA/ Web API auf Variablen zu deaktivieren. Der Zugriff auf statische Variablen eines FB ist nicht erlaubt. Entsprechende Formalparameter für den Schreib- bzw. Lesezugriff sind anzulegen.

Im Editor von PLC-Datentypen ist zumindest die Erreichbarkeit per HMI/ OPC UA/ Web API zu aktivieren, um zunächst grundsätzlich den Zugriff zu ermöglichen. Erst bei der Verwendung eines PLC-Datentyps wird in der Bausteinschnittstelle festgelegt, ob der Zugriff tatsächlich erlaubt werden soll.

SE005 Regel: Fehlercodes auswerten

Stellen im Programm aufgerufene FCs, FBs oder Systemfunktionen Fehlercodes oder Statusmeldungen bereit, sind diese auszuwerten.

Kann ein Fehler eines Bausteinaufruf nicht durch das Anwenderprogramm behandelt oder bearbeitet werden, so sind eindeutige Fehlermeldungen für den Anwender bereitzustellen, die eine Lokalisierung der Fehlerursache ermöglichen.

Hinweis Weitere Informationen zum Thema Fehlerbehandlung finden Sie in "DA013 Regel: Status/ Fehler per "status"/ "error" zurückgeben".

SE006 Regel: Fehler-OB mit Auswertelogik schreiben

Werden Organisationsbausteine zur Fehlerbehandlung eingesetzt, so haben diese auch eine entsprechende Aufgabe.

Die minimale Anforderung ist die Auswertung des Fehlers und die entsprechende Berücksichtigung im Anwenderprogramm, um Fehler aufzuzeigen und ggf. abzuarbeiten.

9 Designrichtlinien/ Architektur

Dieses Kapitel beschreibt die Regeln und Empfehlungen für Programmdesign und Programmarchitektur.

DA001 Regel: Projekt/ Bibliothek gruppieren und strukturieren

Strukturieren und gruppieren Sie Ihr Programm in logische Einheiten. Das System stellt dazu die verschiedensten Mittel zur Verfügung.

- Fassen Sie zusammengehörige Bausteine in einem Ordner/ einer Gruppe zusammen
- Strukturieren von technologischen Anlagenteilen in Software Units
- Gliedern von Programmen in logische funktionale Einheiten – FC/ FB
- Zusammenfassen von zusammengehörigen Daten in PLC-Datentypen
- Strukturieren von Programmcode durch Netzwerke bzw. Region

Hinweis

REGION in SCL ist vergleichbar mit Netzwerken in KOP/ FUP.

Der Name einer Region ist vergleichbar mit einem Netzwerktitel und ist daher wie ein Titel/ Kommentar zu schreiben.

Regionen bieten verschiedene Vorteile:

- Eine Übersicht aller Regionen im Editor am linken Rand
- Schnelle Navigation durch den Code mit Hilfe der Übersicht und Verlinkung
- Die Möglichkeit des Faltens und Ausblendens von Codefragmenten
- Schnelles Ein- und Ausklappen mit Hilfe der Navigation durch die Synchronisierung von Übersicht und Code

DA002 Empfehlung: Geeignete Programmiersprache verwenden

Es ist eine für den Anwendungsfall geeignete Programmiersprache zu wählen.

Standardbausteine – strukturierter Text (SCL/ ST)

Als Programmiersprache von Standardbausteinen ist SCL die bevorzugte Sprache. SCL bietet die kompakteste Lesbarkeit unter den Programmiersprachen und unterstützt zudem den Programmierer durch Automarkierung aller Verwendungsstellen bei Selektion eines Codeelements.

Aufrufumgebungen – grafisch/ blockorientiert (KOP, FUP)

Soll eine Verschaltung einzelner Bausteine vorgenommen werden, z. B. in einem OB als Aufrufumgebung, kann die Programmiersprache KOP oder FUP gewählt werden. Auch wenn ein Baustein größtenteils aus Binärverknüpfungen besteht, kann KOP oder FUP gewählt werden. In diesen Fällen sind durch die Wahl der Programmiersprache KOP oder FUP eine leichtere Diagnose und eine schnellere Übersicht durch Servicepersonal möglich.

Schrittketten – flussorientiert (GRAPH)

Bei Ablaufketten empfiehlt sich die Verwendung von GRAPH. So können sequenzielle Abläufe übersichtlich und schnell programmiert und nachverfolgt werden. Zusätzlich sind hier bereits "Interlocks" und "Supervisions" systemintegriert.

DA003 Regel: Bausteineigenschaften setzen/ prüfen

In den Bausteineigenschaften sind folgende Einstellungen zu aktivieren:

- **Autonummerierung:** Bausteine (OB, FC, FB, DB, TO) werden nur mit aktivierter automatischer Nummernvergabe ausgeliefert. Beachten Sie, dass die Abarbeitung eines Organisationsbausteins von seiner Nummer/Priorität abhängig ist.
- **IEC Check:** Um die IEC-konforme Programmierung zu gewährleisten, wird der IEC Check aktiviert. So wird die typkonforme und typsichere Programmierung sichergestellt.
- **Optimierter Zugriff:** Für die vollsymbolische Programmierung und maximale Performance ist der optimierte Bausteinzugriff zu aktivieren.

In den Bausteineigenschaften sind folgende Attribute zu prüfen:

- **ENO:** Siehe "SE003 Regel: ENO behandeln"
- **Multiinstanz-Fähigkeit:** Die Verwendung als Multiinstanz ist gewährleistet, wenn ein Baustein intern auch Multiinstanzen anstatt globaler Einzelinstanzen verwendet.
- **Bibliothekskonformität:** Siehe "RU004 Regel: Nur lokale Variablen verwenden"

Abbildung 9-1: Autonummerierung

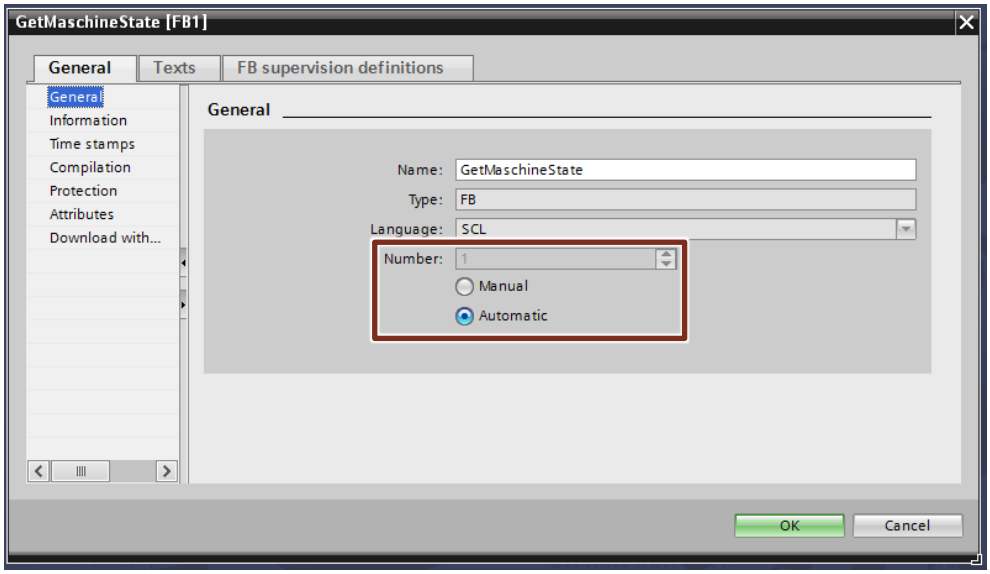


Abbildung 9-2: IEC Check, ENO, Optimierter Zugriff, Multiinstanz-Fähigkeit

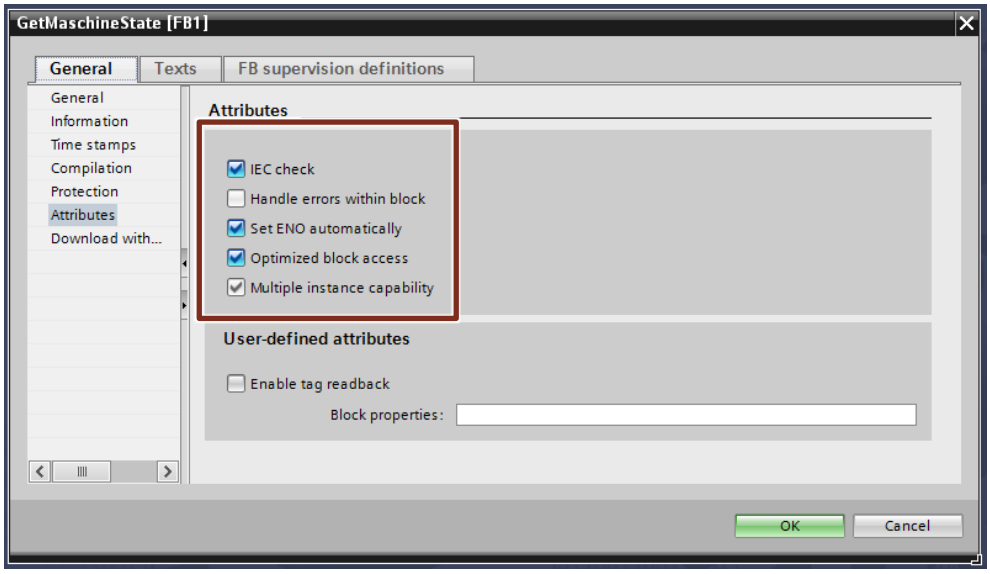
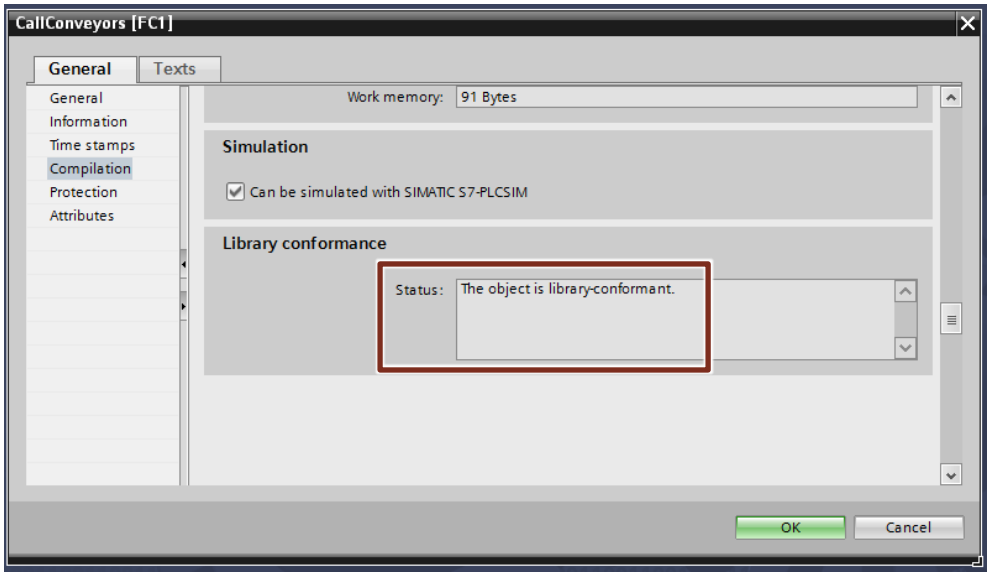


Abbildung 9-3: Bibliothekskonformität



DA004 Regel: PLC-Datentypen verwenden

Zum Strukturieren sind im Anwenderprogramm PLC-Datentypen zu verwenden. Auch in den Lokaldaten werden PLC-Datentypen eingesetzt, wenn sie als Ganzes ausgetauscht werden.

Strukturen (STRUCT) werden nur in den Lokaldaten eines Bausteines definiert, um bei Bedarf Variablen innerhalb des Bausteins durch die Gruppierung übersichtlicher darzustellen, nicht aber um sie strukturiert auszutauschen.

Begründung: Eine Änderung in einem PLC-Datentyp wird an allen Verwendungsstellen im Anwenderprogramm automatisch aktualisiert und der Datenaustausch über Formalparameter zwischen mehreren Bausteinen wird vereinfacht.

DA005 Regel: Daten nur über Formalparameter austauschen

Werden Daten in mehreren FBs oder FCs benötigt, erfolgt der Datenaustausch ausschließlich über die Eingangs-, Ausgangs- oder Durchgangsparameter.

Begründung: Im Sinne gekapselter Bausteine werden Daten von wiederverwendbaren Bausteinen entkoppelt und Abhängigkeiten aufgelöst. Der Baustein muss nicht modifiziert werden, da die Daten übersichtlich über Formalparameter ausgetauscht werden. Der Aufrufer hat die Kontrolle, an welchen Stellen die Daten verwendet werden. Die Datenkonsistenz wird bei mehrfachem Zugriff (ggf. an unterschiedlichen Programmstellen) sichergestellt.

DA006 Regel: Auf statische Variablen nur lokal zugreifen

Die statischen Daten eines Funktionsbausteins sind nur innerhalb des Bausteins zu verwenden in dem sie deklariert sind.

Begründung: Bei direktem Zugriff auf statische Variablen einer Instanz ist die Kompatibilität nicht gewährleistet, da keinerlei Einfluss auf zukünftige Updates besteht. Außerdem ist nicht klar, welchen Einfluss das Modifizieren statischer Variablen von außen auf die Abarbeitung innerhalb eines FB hat.

DA007 Empfehlung: Formalparameter zusammenfassen

Werden viele (z. B. mehr als zehn) Parameter übergeben, empfiehlt es sich, diese in PLC-Datentypen zusammenzufassen. Diese Parameter werden als Durchgangsparameter deklariert und daher mit "Call by reference" übergeben.

Beispiele für solche Parameter sind Konfigurationsdaten, Istwerte, Sollwerte oder Ausgabe aktueller Diagnosedaten des Funktionsbausteins.

Siehe auch "PE003 Empfehlung: Strukturierte Parameter als Referenz übergeben"

Hinweis Bei sich oft ändernden Steuer- bzw. Statusvariablen kann es sinnvoll sein, diese zum einfacheren Beobachten in KOP/ FUP direkt als elementare Eingangs- bzw. Ausgangsparameter zu deklarieren.

DA008 Regel: Ausgangsparameter genau einmal schreiben

Die Ausgangsparameter und Rückgabewerte werden pro Bearbeitungszyklus genau einmal mit einem Wert beschrieben. Dies muss möglichst gemeinsam und am Ende passieren.

Es ist nicht erlaubt, den eigenen Ausgangsparameter oder Rückgabewert zu lesen. Stattdessen muss eine temporäre oder statische Variable eingeführt werden.

Begründung: Dadurch wird sichergestellt, dass alle Ausgänge konsistent gehalten werden.

DA009 Regel: Nur genutzten Code beibehalten

Im fertigen Programm darf nur Code enthalten sein, der aufgerufen und somit in der PLC ausgeführt wird.

Beispiel für Verstöße:

- Nie aufgerufene Bausteine oder Technologieobjekte
- Nie verwendete Variablen oder Konstanten
- Nie verwendete Parameter
- Nie durchlaufener Code
- Auskommentierter Code
- Nie zugegriffene PLC-Variablen
- Nie genutzte Anwenderkonstanten
- Externe Quellen

Hinweis Produktivcode, der je nach Option zu einem anderen Zeitpunkt benötigt wird, ist davon nicht betroffen.

DA010 Regel: Asynchrone Bausteine nach PLCopen entwickeln

Die PLCopen Organisation hat einen Standard für Motion Control-Bausteine definiert. Dieser Standard kann soweit verallgemeinert werden, dass er auf alle asynchronen Funktionsbausteine angewendet werden kann. Asynchron heißt in diesem Zusammenhang: Alle Bausteine, die über mehrere Zyklen bearbeitet werden, folgen diesem Standard, z. B. für Kommunikation, Regler, Motion Control.

Begründung: Durch diese Standardisierung kann eine Vereinfachung der Programmierung und Anwendung von Funktionsbausteinen erreicht werden.

DA011 Regel: Kontinuierliche asynchrone Abarbeitung mit "enable"

Bausteine die einmalig gestartet und initialisiert werden, anschließend im Betrieb sind und auf weitere Eingaben reagieren, verwenden den Eingangsparameter "enable".

Beispiel: Ein Kommunikationsbaustein (Server) wartet nach erfolgter Initialisierung auf eingehende Verbindungen eines Clients. Nach erfolgreicher Datenübertragung wartet der Server wieder auf eine eingehende Verbindung.

Hinweis Die Bausteinvorlagen mit "enable" finden Sie als Kopiervorlagen in der Bibliothek mit generellen Funktionen (LGF):

https://support.industry.siemens.com/cs/ww/de/view/109479728

Mit dem Setzen des Parameters "enable" wird der Auftrag gestartet und asynchron abgearbeitet. Solange "enable" gesetzt bleibt, ist die Auftragsbearbeitung aktiv und es können fortlaufend neue Werte übernommen und verarbeitet werden.

Mit dem Rücksetzen des Parameters "enable" wird der Auftrag beendet.

Diagnoseinformationen ("diagnostics") werden erst durch eine erneute steigende Flanke an "enable" zurückgesetzt.

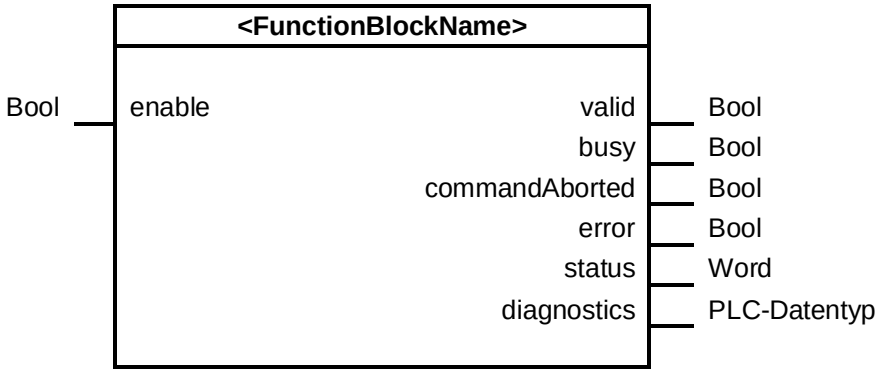
Wenn der Baustein nach PLCopen implementiert wird und den Eingangsparameter "enable" verwendet, müssen mindestens die Ausgangsparameter "valid", "error" und "busy" verwendet werden.

Tabelle 9-1

Bezeichner	Datentyp	Bedeutung
Eingangsparameter		
enable	Bool	Alle Parameter werden mit einer steigenden Flanke am Eingang "enable" übernommen und können fortlaufend verändert werden. Die Funktion wird pegelgesteuert aktiviert (bei TRUE) und deaktiviert (bei FALSE).
Ausgangsparameter		
valid	Bool	Die Ausgänge "valid" und "error" sind gegenseitig **exklusiv**. Der Ausgang ist gesetzt, solange gültige Ausgangswerte verfügbar sind und der "enable" Eingang gesetzt ist. Sobald ein Fehler ansteht, wird der Ausgang "valid" zurückgesetzt.
busy	Bool	Der FB ist gerade mit der Bearbeitung eines Kommandos beschäftigt. Neue Ausgangswerte können erwartet werden. "busy" wird mit einer steigenden Flanke von "enable" gesetzt und bleibt gesetzt, solange der FB Aktionen ausführt.
error	Bool	Die Ausgänge "valid" und "error" sind gegenseitig **exklusiv**. Eine steigende Flanke des Ausgangs signalisiert, dass ein Fehler während der Abarbeitung des FB aufgetreten ist.
commandAborted	Bool	**Optionaler** Ausgang, der anzeigt, dass der laufende Auftrag des FB durch eine andere Funktion bzw. durch einen anderen Auftrag an das gleiche Objekt abgebrochen wurde. Beispiel: Eine Achse wird über den Funktionsbaustein gerade positioniert, während über einen anderen Funktionsbaustein die gleiche Achse angehalten wird. Am Funktionsbaustein der Positionierung wird dann der Ausgang "commandAborted" gesetzt, da dieser Auftrag durch das Halt-Kommando abgebrochen wurde.
status	Word	**Optionaler** Ausgang: Fehler- oder Statusinformation des Bausteins: Dieser Parameter wird in Anlehnung an die vorhandenen Systembausteine benannt. ("errorID" nach PLCopen)
diagnostics	PLC-Datentyp	**Optionaler** Ausgang: Detaillierte Fehlerinformation Hier werden alle Fehler, Warnungen und Informationen des Bausteins abgelegt. Der Aufbau der Diagnose ist in der Empfehlung "Unterlagerte Statusinformationen durchreichen" beschrieben.

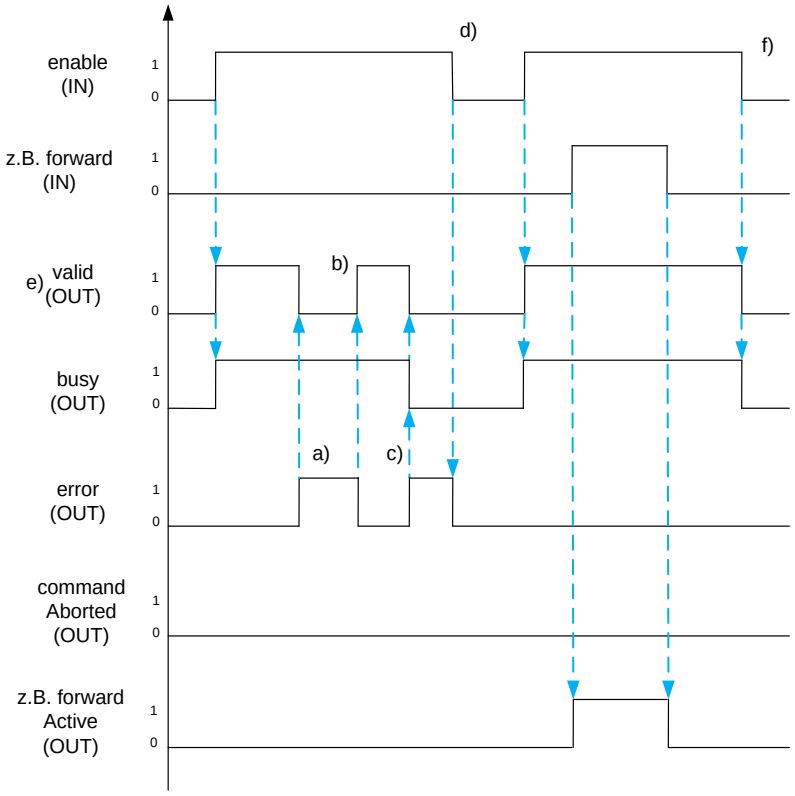
Beispiel:

Abbildung 9-4



Signalablaufdiagramm eines Bausteins mit "enable":

Abbildung 9-5



a) Mit "error" auf TRUE wird "valid" rückgesetzt und alle Funktionalitäten des FB gestoppt.
Da es sich um einen Fehler handelt, der vom Baustein selbst behoben werden kann, bleibt "busy" gesetzt.

b) Nach Behebung der Fehlerursache (z. B. Neuaufbau der Kommunikations-verbindung) wird "valid" wieder gesetzt.

c) Ein Fehler, der nur durch den Benutzer behoben werden kann, tritt ein. Hierbei muss "error" gesetzt, "busy" und "valid" zurückgesetzt werden.

d) Nur durch eine fallende Flanke an "enable" kann der anstehende Fehler, der durch den Benutzer behoben werden muss, quittiert werden.

e) "valid" auf TRUE bedeutet, dass der Baustein aktiviert ist, keine Fehler anstehen und somit die Ausgänge des FB gültig sind.

f) Wenn "enable" auf FALSE zurückgesetzt wird, werden "valid" und "busy" auch zurückgesetzt.

"commandAborded", "error" und "done" sind immer so lange gesetzt, wie das Signal "enable" ansteht, mindestens jedoch für einen Zyklus.

DA012 Regel: Einmalige asynchrone Abarbeitung mit "execute"

Bausteine, die einmalig abgearbeitet werden, besitzen den Eingangsparameter "execute".

Beispiel: Ein Kommunikationsbaustein (Client) fordert einmalig Daten von einem Server an, was durch die Flanke an "execute" signalisiert wird. Nach erfolgter Verarbeitung der Antwort ist die Abarbeitung beendet. Ein erneuter Auftrag erfordert eine erneute Flanke an "execute".

Hinweis Die Bausteinvorlagen mit "execute" finden Sie als Kopiervorlagen in der Bibliothek mit generellen Funktionen (LGF):

https://support.industry.siemens.com/cs/ww/de/view/109479728

Mit einer steigenden Flanke am Parameter "execute" wird der Auftrag gestartet und die an den Eingangsparametern anstehenden Werte übernommen.

Nachträglich geänderte Parameterwerte werden erst beim nächsten Auftragsstart übernommen, wenn kein "continuousUpdate" verwendet wird.

Das Rücksetzen des Parameters "execute" beendet die Bearbeitung des Auftrags nicht, hat aber Einfluss auf die Anzeigedauer des Auftragsstatus. Wenn "execute" vor Abschluss eines Auftrags rückgesetzt wird, werden die Parameter "done", "error" und "commandAborted" entsprechend nur für einen Aufrufzyklus gesetzt.

Diagnoseinformationen ("diagnostics") werden erst durch eine erneute steigende Flanke an "execute" zurückgesetzt.

Nach Ende des Auftrags ist eine neue steigende Flanke an "execute" notwendig, um einen neuen Auftrag zu starten. Somit ist gewährleistet, dass bei jedem Start eines Auftrags der Baustein im Initialzustand ist und die Funktion unabhängig von vorherigen Aufträgen bearbeitet.

Wenn der Baustein nach PLCopen implementiert wird und den Eingangsparameter "execute" benutzt, müssen mindestens die Ausgangsparameter "busy", "done" und "error" verwendet werden.

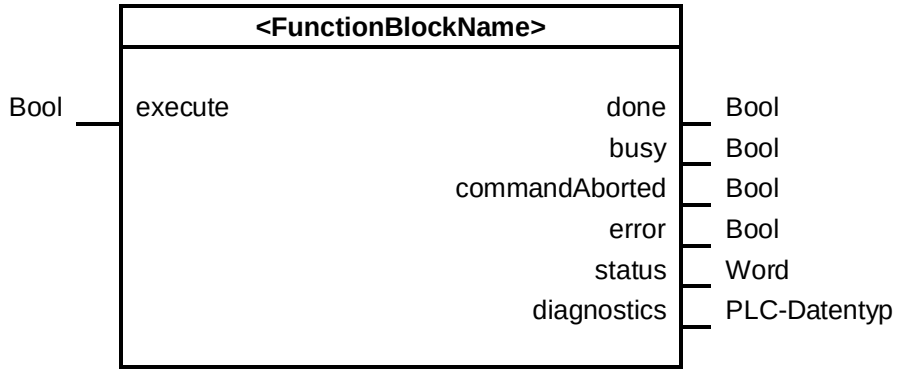
Tabelle 9-2

Bezeichner	Datentyp	Bedeutung
Eingangsparameter		
execute	Bool	**"execute" ohne "continuousUpdate":** Alle Parameter werden mit einer steigenden Flanke am Eingang "execute" übernommen und die Funktionalität wird gestartet. Ist eine Änderung der Parameter notwendig, muss der Eingang "execute" neu getriggert werden. **"execute" mit "continuousUpdate":** Alle Parameter werden mit einer steigenden Flanke am Eingang "execute" übernommen. Diese können solange angepasst werden, wie der Eingang "continuousUpdate" gesetzt ist.
continuousUpdate	Bool	**Optionaler** Eingang: Bedeutung siehe Eingang "execute"
Ausgangsparameter		
done	Bool	Die Ausgänge "done", "busy", "commandAborted" und "error" sind gegenseitig **exklusiv**. Der Ausgang "done" wird gesetzt, wenn das Kommando erfolgreich abgearbeitet wurde.

Bezeichner	Datentyp	Bedeutung
busy	Bool	Die Ausgänge "done", "busy", "commandAborted" und "error" sind gegenseitig **exklusiv**. Der FB ist noch nicht mit der Bearbeitung des Kommandos fertig und somit können neue Ausgangswerte erwartet werden. "busy" wird bei steigender Flanke von "execute" gesetzt und rückgesetzt, wenn einer der Ausgänge "done", "commandAborted" oder "error" gesetzt wird.
error	Bool	Die Ausgänge "done", "busy", "commandAborted" und "error" sind gegenseitig **exklusiv**. Eine steigende Flanke des Ausgangs signalisiert, dass ein Fehler während der Abarbeitung des FB aufgetreten ist.
commandAborted	Bool	Die Ausgänge "done", "busy", "commandAborted" und "error" sind gegenseitig **exklusiv**. **Optionaler** Ausgang, der anzeigt, dass der laufende Auftrag des Funktionsbausteins durch eine andere Funktion bzw. durch einen anderen Auftrag an das gleiche Objekt abgebrochen wurde. Beispiel: Eine Achse wird über den Funktionsbaustein gerade positioniert, während über einen anderen Funktionsbaustein die gleiche Achse angehalten wird. Am Funktionsbaustein der Positionierung wird dann der Ausgang "commandAborted" gesetzt, da dieser Auftrag durch das Halt-Kommando abgebrochen wurde.
status	Word	**Optionaler** Ausgang: Fehler- oder Statusinformation des Bausteins: Dieser Parameter wird in Anlehnung an die vorhandenen Systembausteine benannt. ("errorID" nach PLCopen)
diagnostics	PLC-Datentyp	**Optionaler** Ausgang: Detaillierte Fehlerinformation Hier werden alle Fehler, Warnungen und Informationen des Bausteins abgelegt. Der Aufbau der Diagnose ist in der Empfehlung "Unterlagerte Statusinformationen durchreichen" beschrieben.

Beispiel:

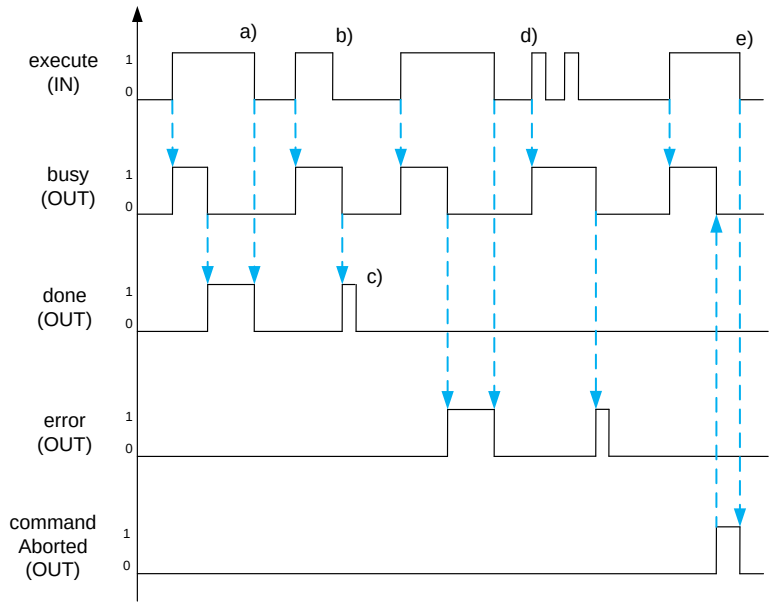
Abbildung 9-6



Signalablaufdiagramm eines Bausteins mit "execute":

Hinweis Wird der Eingangsparameter "execute" zurückgesetzt, bevor der Ausgangsparameter "done" oder "error" gesetzt ist, so ist der Ausgangsparameter "done" oder "error" nur einen Zyklus zu setzen.

Abbildung 9-7



a) "done", "error" und "commandAborted" werden mit fallender Flanke an "execute" zurückgesetzt.

b) Die Funktionalität des FB wird mit fallender Flanke an "execute" nicht gestoppt.

c) Wenn "execute" bereits FALSE ist, dann stehen "done", "error" und "commandAborted" nur für einen Zyklus an.

d) Es wird ein neuer Auftrag mit einer steigenden Flanke an "execute" angefordert, während der Baustein noch in Bearbeitung ist ("busy" = TRUE). Der alte Auftrag wird entweder mit den zu Auftragsbeginn anstehenden Parametern beendet, oder es wird der alte Auftrag abgebrochen und mit den neuen Parametern neu gestartet. Das Verhalten richtet sich nach dem Anwendungsfall und ist entsprechend zu dokumentieren.

e) Wird die Bearbeitung eines Auftrags durch einen Auftrag mit höherer oder gleicher Priorität (von einem anderen Baustein/ Instanz) unterbrochen, wird vom Baustein "commandAborted" gesetzt. Dieser unterbricht sofort die restliche Bearbeitung des Auftrags. Dieser Fall tritt z. B. ein, wenn ein Emergency-Stop an einer Achse ausgeführt werden soll, während ein Baustein einen Verfahrauftrag an dieser Achse ausführt.

DA013 Regel: Status / Fehler per "status"/ "error" zurückgeben

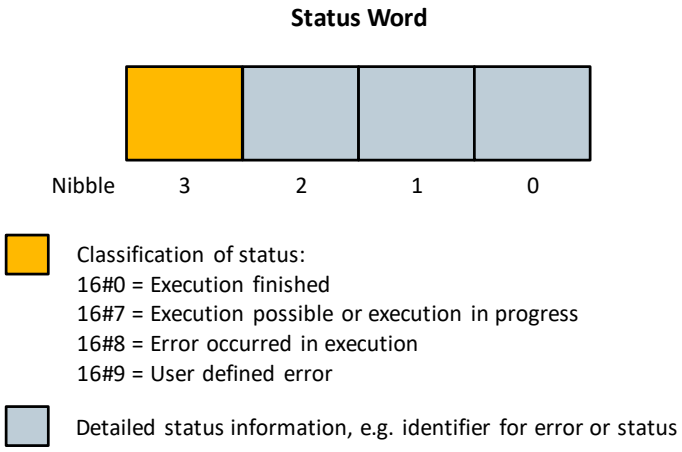
Der Baustein gibt eine eindeutige Auskunft am Ausgangsparameter "status", die auf den Zustand im Baustein schließen lässt. Die Werte sind als Konstanten in den Lokaldaten der Bausteinschnittstelle zu definieren, um doppelte Belegungen zu verhindern und die Lesbarkeit durch die Symbolik zu verbessern.

Hat der Baustein einen Fehler, sind hierfür die Ausgangsparameter "status" und "error" zu verwenden. Nach dem folgendem Statuskonzept sind die Parameter "error" und das hochwertigste Bit (MSB) von "status" (Bit 15) identisch. Die restlichen Bits werden für einen Fehlercode benutzt, der eindeutig auf die Ursache hinweist.

Aus Kompatibilitätsgründen zu bestehenden SIMATIC-Systembausteinen wird auf den Ausgang "errorID", der nach dem PLCopen Standard vorgeschrieben ist, zugunsten des Ausgangs "status" verzichtet.

Begründung: Dadurch können noch weitere Informationen des Bausteins nach außen über den Ausgang "status" ausgegeben werden, die keine Fehlerinformationen beinhalten.

Abbildung 9-8



DA014 Regel: Standardisierte Wertebereiche in "status" verwenden

Für eine Standardisierung des Ausgangsparameters "status" sind die in der folgenden Tabelle gezeigten Wertebereiche für Informationen und Fehler einzuhalten.

Tabelle 9-3

Information	Wertebereich
Auftrag abgeschlossen, keine Warnung und keine weitere Detaillierung	16#0000
Auftrag abgeschlossen, weitere Detaillierung	16#0001 ... 16#0FFF
Kein Auftrag in Bearbeitung (auch Initialwert)	16#7000
Erster Aufruf nach Eingang eines neuen Auftrags (steigende Flanke "execute")	16#7001
Folgeaufruf während aktiver Bearbeitung ohne weitere Detaillierung	16#7002
Folgeaufruf während aktiver Bearbeitung mit weiterer Detaillierung. Aufgetretene Warnungen, die den Betrieb nicht weiter beeinflussen.	16#7003 ... 16#7FFF

Tabelle 9-4

Fehler	Wertebereich
Falsche Bedienung des Funktionsbausteins	16#8001 ... 16#81FF
Fehler bei der Parametrierung	16#8200 ... 16#83FF
Fehler bei der Abarbeitung von außen (z. B. falsche I/O-Signale, Achse nicht referenziert)	16#8400 ... 16#85FF
Fehler bei der Abarbeitung intern (z. B. bei Aufruf einer Systemfunktion)	16#8600 ...16#87FF
Reserviert	16#8800 ... 16#8FFF
Benutzerdefinierte Fehlerklassen	16#9000 ... 16#FFFF

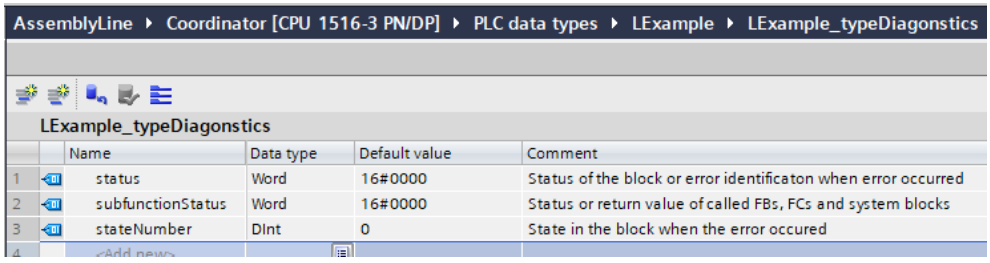
DA015 Empfehlung: Unterlagerte Informationen durchreichen

Werden in einem Baustein weitere Unterfunktionen aufgerufen, die detaillierte Status- und möglicherweise Diagnose-Informationen liefern, so sind diese über eine Diagnosestruktur am Ausgangsparameter "diagnostics" abzulegen. Des Weiteren können in der Diagnosestruktur auch weitere Begleitwerte zur Diagnose des aktuellen Bausteinverhaltens abgelegt werden, z. B. Laufzeitinformationen.

Hinweis Die Diagnosestruktur kann remanent angelegt werden, um eine Diagnose auch nach einem Spannungsausfall an der PLC zu ermöglichen.

Beispiel für eine einfache Diagnosestruktur:

Abbildung 9-9



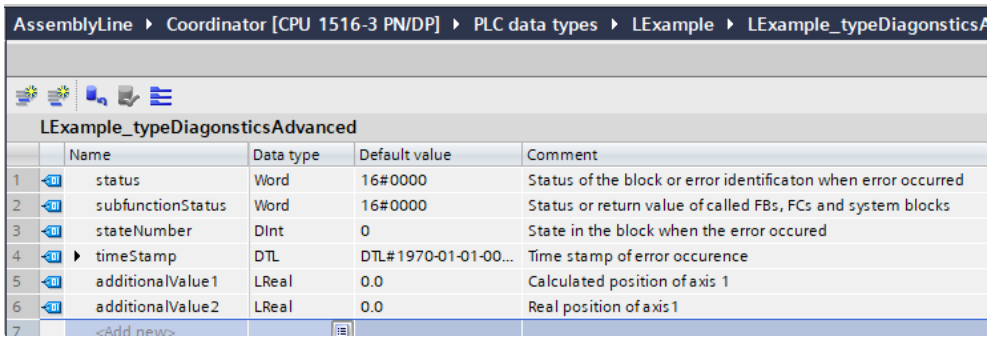
Die einfache Diagnosestruktur enthält drei Parameter:

Tabelle 9-5

Bezeichner	Datentyp	Beschreibung
status	Word	Status des Bausteins selbst
subfunctionStatus	Word/ DWord	Status von unterlagerten Funktionen
stateNumber	DInt	Die Nummer des Bearbeitungsschritts, in dem der Fehler aufgetreten ist

Beispiel für eine erweiterte Diagnosestruktur:

Abbildung 9-10



	Name	Data type	Default value	Comment
1	status	Word	16#0000	Status of the block or error identificaton when error occurred
2	subfunctionStatus	Word	16#0000	Status or return value of called FBs, FCs and system blocks
3	stateNumber	DInt	0	State in the block when the error occured
4	timeStamp	DTL	DTL#1970-01-01-00...	Time stamp of error occurence
5	additionalValue1	LReal	0.0	Calculated position of axis 1
6	additionalValue2	LReal	0.0	Real position of axis1
7	<Add new>			

In der Variable "timestamp" wird der Zeitpunkt abgespeichert, zu dem der Fehler aufgetreten ist.

In "stateNumber" wird der aktuelle State der internen State-Machine gespeichert.

Wurde ein Fehler einer Systemfunktion oder eines aufgerufenen FBs/ FCs festgestellt, wird dessen Status in der Variable "subfunctionStatus" gespeichert.

Der eindeutige Fehlercode vom Ausgangsparameter "status" wird zusätzlich in der Variable "status" der Diagnosestruktur gespeichert.

Zusätzliche Variablen zu einem Fehler (auch unterlagerte) können mit einem geeigneten Datentyp ergänzt werden, z. B. mit Hilfe von "additionalValueX". "X" wird durch eine fortlaufende Nummer, beginnend bei 1, ersetzt.

DA016 Empfehlung: CASE-Anweisung statt ELSIF-Zweige nutzen

Wenn möglich, soll anstatt einer IF-Anweisung mit mehreren ELSIF-Zweigen eine CASE-Anweisung verwendet werden.

Begründung: Damit wird das Programm übersichtlicher.

DA017 Regel: ELSE-Zweig bei CASE-Anweisungen erstellen

Eine CASE-Anweisung muss immer einen ELSE-Zweig aufweisen.

Begründung: Dies dient dazu, Fehler, die zur Laufzeit auftreten, melden zu können.

Beispiel:

```
CASE #stateSelect OF
  CMD_INIT: // Comment
    ; // Statement
  CMD_READ: // Comment
    ; // Statement
ELSE
  // default statement
  ; // Generate error message
END_CASE;
```

DA018 Empfehlung: Jump und Label vermeiden

Auf Sprünge im Programm ist zu verzichten. Verwendet werden Sprünge nur im Ausnahmefall, wenn der Programmablauf nicht durch andere Methoden zu realisieren ist.

Begründung: Sprünge führen mitunter zu einem unleserlichen Programm, da durch diese Befehle im Code hin und her gesprungen wird.

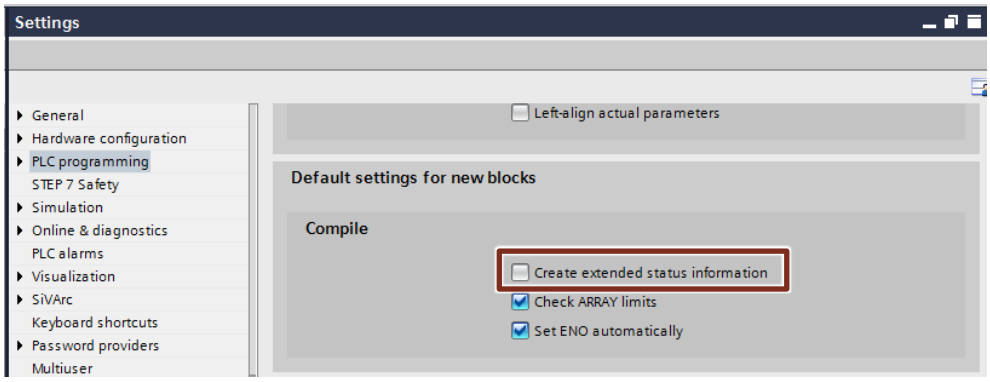
10 Performance

Dieses Kapitel beschreibt die Regeln und Empfehlungen für ein performantes Anwenderprogramm.

PE001 Empfehlung: "Create extended status info" deaktivieren

Das Deaktivieren von "Create extended status information" kann im produktiven Betrieb zu Performance-Verbesserungen führen. Für die Entwicklung und zum Debuggen des Ablaufprogramms kann es zu Beobachtungszwecken Vorteile bringen, die Option zu aktivieren.

Abbildung 10-1



PE002 Empfehlung: "Set in IDB" vermeiden

Aufgrund der Bausteinoptimierung und symbolischer Programmierung sollte auf die Verwendung der Funktionalität "Set in IDB" in der Bausteinschnittstelle verzichtet werden.

Begründung: Die Nutzung von "Set in IDB" hat zur Folge, dass ein Misch-DB aus optimierten und nichtoptimierten Daten generiert wird. Beim Zugriff auf die Daten muss daher ggf. ein Umkopieren in das jeweils andere Datenformat erfolgen.

Hinweis "Set in IDB" wird häufig im Zusammenhang mit dem "AT Konstrukt" verwendet. Stattdessen können Sie Slice-Zugriffe oder die Systemfunktionen "SCATTER" und "GATHER" verwenden.

PE003 Empfehlung: Strukturierte Parameter als Referenz übergeben

Um möglichst performant (speicher- und laufzeitoptimiert) Daten über die Formalparameter der Bausteinschnittstelle zu übergeben, wird empfohlen "Call by reference" zu nutzen.

Begründung: Beim Aufruf des Bausteins wird eine Referenz auf den Aktualparameter übergeben, wofür kein Kopiervorgang ausgeführt wird.

Hinweis Dadurch können die originalen Daten modifiziert werden.

Wenn beim Bausteinaufruf optimierte Daten an einen Baustein mit der deaktivierten Eigenschaft "Optimierter Bausteinzugriff" (oder auch umgekehrt) übergeben werden, werden diese grundsätzlich als Kopie übergeben. Wenn der Baustein viele strukturierte Parameter enthält, kann das dazu führen, dass der temporäre Speicherbereich des Bausteins überläuft. Das können Sie vermeiden, indem Sie für beide Bausteine die Zugriffsart "optimiert" einstellen.

Die folgende Tabelle gibt zusammenfassend einen Überblick wie Formalparameter in einer SIMATIC S7-1200/ S7-1500 PLC mit einem elementaren bzw. strukturierten Datentyp übergeben werden.

Tabelle 10-1

Bausteintyp/ Formalparameter		Elementarer Datentyp	Strukturierter Datentyp
FC	Input	Kopie	Referenz
	Output	Kopie	Referenz
	InOut	Kopie	Referenz
FB	Input	Kopie	Kopie
	Output	Kopie	Kopie
	InOut	Kopie	Referenz

PE004 Empfehlung: Formalparameter mit Variant vermeiden

Um Performanceeinbußen aufgrund der Verwendung von Variant zu vermeiden, wird empfohlen für unterschiedliche Datentypen eigene Bausteine vorzuhalten.

Die Verwendung von Variant wird nur benötigt, wenn z. B. Daten für die Kommunikation an einen Baustein übergeben werden, um diese an die systeminternen Kommunikationsbausteine oder zur Serialisierung durchzureichen.

PE005 Empfehlung: Formalparameter "mode" vermeiden

Es soll vermieden werden einen Baustein mit verschiedenen Funktionalitäten zu erstellen, die dann mit dem Parameter "mode" ausgewählt werden.

Begründung: Dadurch wird vermieden, dass Codefragmente, die nicht verwendet werden ("toter Code"), ins Anwenderprogramm gelangen, da der Modeparameter zumeist statisch verschaltet wird.

Stattdessen sollte die Funktionalitäten auf einzelne Module verteilt werden:

- Das spart Speicher und erhöht die Performance durch Codereduktion.
- Es erhöht die Lesbarkeit durch bessere Differenzierung in der Namensvergabe.
- Es erhöht die Wartbarkeit durch kleinere Codefragmente, die sich nicht beeinflussen.

PE006 Empfehlung: Temporäre Variablen bevorzugen

Grundsätzlich sind Variablen im temporären Bereich zu deklarieren, wenn sie nur im aktuellen Zyklus benötigt werden. Temporäre Variablen bieten im Baustein die beste Performance.

Falls sehr häufig auf Eingangs- oder Durchgangsparameter zugegriffen wird, soll eine temporäre Variable als Zwischenspeicher genutzt werden, um Laufzeit einzusparen.

Hinweis Temporäre Variablen können nicht in Force- und Beobachtungstabellen beobachtet werden.

PE007 Empfehlung: Wichtige Testvariablen statisch deklarieren

Wichtige Testvariablen sollen statisch deklariert werden. Sie müssen ausreichende Auskunft über den Zustand der Funktionen geben.

Begründung: Der Aktualwert einer statischen Testvariable bleibt für Diagnosezwecke auch nach Abarbeitung des Bausteins erhalten.

PE008 Empfehlung: Lauf-/ Index-Variablen als "DInt" deklarieren

Für Lauf- und Index-Variablen, die zur Verwendung von Schleifen, Iterationen und Array-Zugriffen verwendet werden, wird empfohlen den Datentyp "DInt" zu verwenden.

Begründung: Dieser Datentyp kann vom Prozessor am performantesten verarbeitet werden, da keine Typkonvertierungen durchgeführt werden müssen. Dementsprechend sollen auch die Definitionen für die Array-Größen und Schleifengrenzen als Konstanten vom Datentyp "DInt" angelegt werden.

PE009 Empfehlung: Mehrmaligen, gleichen Indexzugriff vermeiden

Vermeiden Sie den wiederholten, identischen Zugriff auf denselben Index eines Arrays. Eine temporäre Variable sollte als Zwischenspeicher verwendet werden.

Begründung: Dadurch werden die systeminternen Prüfungen der Arraygrenzen und deren Überschreitung auf ein Minimum reduziert.

Beispiel:

```
FOR #tempIndex := 0 TO #MAX_ARRAY_ELEMENTS DO
  // Copy to temporary variable
  #tempCurrentData := #statArray[#tempIndex];
  // Reset all member variables
  #tempCurrentData.element1 := FALSE;
  #tempCurrentData.element2 := FALSE;
  #tempCurrentData.element3 := FALSE;
  #tempCurrentData.element4 := FALSE;
  #tempCurrentData.element5 := FALSE;
  // Write back the changes made
  #statArray[#tempIndex] := #tempCurrentData;
END_FOR;
```

PE010 Empfehlung: Slice anstelle von Maskierungen verwenden

Anstelle von binären Maskierungen für wenige einzelne Bitzugriffe soll der Slice-Zugriff genutzt werden, um auf einzelne Bits zuzugreifen.

Begründung: Das erhöht die Performance und auch die Lesbarkeit des Quellcodes deutlich.

Beispiel: Prüfung auf Bit1 = TRUE und Bit0 = FALSE mit Slice

```
#tempIsTriggered := (#trigger.%X1 AND NOT #trigger.%X0);
```

Eine Maskierung empfiehlt sich immer dann, wenn Bitmuster mit der kompletten Variablen verglichen werden sollen.

Beispiel: Prüfung auf Bit1 = TRUE und Bit0 = FALSE – mit Maskierung

```
PATTERN_MASK    BYTE    2#00000011
PATTERN         BYTE    2#00000010

#tempIsTriggered := ((#trigger AND #PATTERN_MASK) = #PATTERN);
```

PE011 Empfehlung: IF/ ELSE-Anweisungen vereinfachen

Das Vereinfachen von unnötigen IF/ ELSE-Anweisungen zu einfachen binären Operationen bringt Performance und zugleich eine Optimierung des Speicherhaushalts.

Negativbeispiel anhand einer Flankenerkennung:

```
// Check for rising edge
IF #trigger AND NOT #statTriggerOld THEN
  #tempIsTrigger := TRUE;
ELSE
  #tempIsTrigger := FALSE;
END_IF;
// Store trigger for next cycle
#statTriggerOld := #trigger;
```

Korrektes Beispiel:

```
// Check for rising edge
#tempIsTrigger := #trigger AND NOT #statTriggerOld;
// Store trigger for next cycle
#statTriggerOld := #trigger;
```

PE012 Empfehlung: IF/ ELSIF-Fälle nach Erwartung sortieren

Bei IF/ ELSIF Anweisungen sollen die Fälle mit der größten Wahrscheinlichkeit zuerst geprüft werden, also nach Eintrittshäufigkeit absteigend sortiert.

Begründung: Dadurch erreicht man, dass seltenere Fälle nicht immer geprüft werden müssen und die Programmbearbeitung dadurch schneller ablaufen kann.

Beispiel: Davon ausgehend, dass der Ablauf fehlerfrei implementiert ist und im Regelfall der Idealzustand und Ablauf gegeben ist, wird der Gut-Fall immer zuerst geprüft.

```
// Check if connection is established
IF #instConnect.done = TRUE THEN
  // Connection is established - set next state
  ;
// Check if TCON throws an error
ELSIF #instConnect.error = TRUE THEN
  // TCON throws an error - do error handling
  ;
END_IF;
```

PE013 Empfehlung: Speicherintensive Anweisungen vermeiden

Auf die Verwendung speicherintensiver Anweisungen wie

- "GetSymbolName"
- "GetSymbolPath"
- "GetInstanceName"
- "GetInstancePath"

soll verzichtet werden.

Begründung: Die Verwendung der genannten Anweisungen führt zu einem erhöhten Arbeitspeicherbedarf. Er ist umso größer, je häufiger die Anweisung aufgerufen wird und je länger die symbolischen Bezeichner sind.

PE014 Empfehlung: Laufzeitintensive Anweisungen vermeiden

Die Verwendung von laufzeitintensiven Anweisungen soll auf ein Minimum beschränkt werden.

Laufzeitintensive Anweisungen sind z. B. das Verarbeiten großer Datenmengen mit "Serialize" und "Deserialize" oder Zugriffe auf die Speicherkarte.

Die Systemfunktionen "WRITE_DBL", "READ_DBL", "FileRead" und "FileWrite" greifen auf den vergleichsweisen langsamen Speicher der SIMATIC Memory Card zu. Das erfolgt typischerweise asynchron zur normalen Programmbearbeitung über mehrere Zyklen hinweg. Da jedoch größere Datenmengen übertragen werden, hat die Verwendung dieser Systemfunktionen einen negativen Einfluss auf die Laufzeit des Gesamtprogramms.

Weitere Beispiele für Zugriffe auf die SIMATIC Memory Card sind DataLog- und Recipe-Funktionen.

PE015 Empfehlung: SCL/ KOP/ FUP für zeitkritische Anwendungen nutzen

Bei zeitkritischen Programmen/ Programmteilen und Algorithmen wird empfohlen auf eine der drei Programmiersprachen SCL, KOP oder FUP zurückzugreifen.

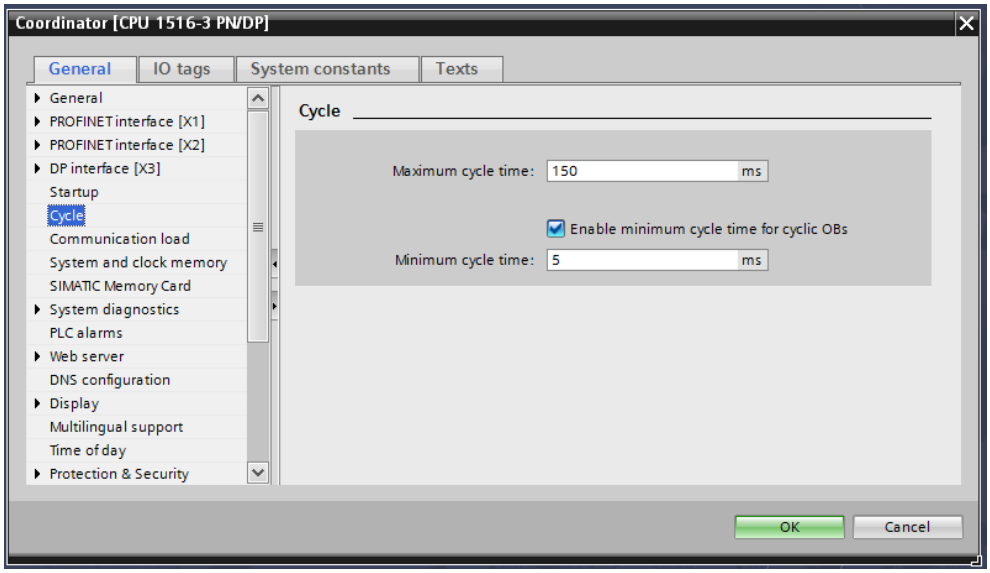
Begründung: GRAPH als Programmiersprache generiert zusätzlich Diagnose-information, die zusätzliche Laufzeit benötigen. GRAPH ist geeignet, um sequenzielle Maschinenabläufe zu erstellen.

PE016 Empfehlung: Einstellung der Mindestzykluszeit prüfen

Bei zeitkritischen Applikationen ohne hohes Kommunikationsaufkommen kann die "minimum cycle time" abgeschaltet werden, wenn eine schnelle Reaktionszeit erwartet wird.

Wird eine hohe Kommunikationslast erwartet und ist die Zykluszeit sehr gering, so kann durch Verlängern der Zykluszeit – z. B. durch Einschalten und Einstellen der "minimum cycle time" – mehr Kommunikationslast abgearbeitet werden.

Abbildung 10-2



11 Spickzettel

Block interface	*In*	*Out*	*InOut*	*Stat*	*Temp*	*Const*
	enable	**done**	**conveyorAxes** **instTimer**	**statState** **instTimer** **powerBusReady**	**tempIndex**	**MAX_VELOCITY**
Prefix	--	--	-- (default) "inst" (param-instance)	"stat" (default) "inst" (multi-instance) -- (in global data block)	"temp"	--
Casing	camelCasing	camelCasing	camelCasing	camelCasing	camelCasing	UPPER_CASING

Tag table	*PLC tag*	*User constant*
	lightBarrierLeft	**MAX_BELTS**
Prefix	--	--
Casing	camelCasing	UPPER_CASING

Programming styleguide for SIMATIC S7-1200/ S7-1500 in TIA Portal

- Unique, meaningful identifiers in English
- Only the characters a-z, A-Z, 0-9 and _
- Maximum 24 characters per identifier
- Array: axesData [0..#MAX] of type…
- Library: Name max. 8 chars; prefix "LExample_"

Object		Prefix	Casing
Project	**AssemblyLine**	--	PascalCasing
Library	**LCom**	"L"	PascalCasing
Organization block	**Main**	--	PascalCasing
Function block	**HeatTank**	--	PascalCasing
Function	**CalculateTime**	--	PascalCasing
Global data block	**MachineData**	--	PascalCasing
Single instance data block	**InstHeater**	"Inst"	PascalCasing
Technological object	**HeatingAxis**	--	PascalCasing
PLC tag table	**Sensors**	--	PascalCasing
Watch/Force table	**MachineState**	--	PascalCasing
Trace	**ConveyorSpeed**	--	PascalCasing
Measurement	**Temperature**	--	PascalCasing
PLC alarm text list	**ConveyorAlarms**	--	PascalCasing
Software unit	**Magazine**	--	PascalCasing
PLC datatype	**typeDiagnostics**	"type"	camelCasing
Element in a PLC datatype	**stateNumber**	--	camelCasing

Usual abbreviations
(maximum one per identifier)

Min / Max	Minimum / Maximum
Act	Actual, Current
Next / Prev	Next / Previous value
Avg	Average
Sum	Total sum
Diff	Difference
Cnt	Count
Len	Length
Pos	Position
Ris / Fal	Rising / falling edge
Old	Old value
Sim	Simulated
Dir	Direction
Err / Warn	Error / Warning
Cmd	Command
Addr	Address

12 Anhang

12.1 Service und Support

Industry Online Support

Sie haben Fragen oder brauchen Unterstützung?

Über den Industry Online Support greifen Sie rund um die Uhr auf das gesamte Service und Support Know-how sowie auf unsere Dienstleistungen zu.

Der Industry Online Support ist die zentrale Adresse für Informationen zu unseren Produkten, Lösungen und Services.

Produktinformationen, Handbücher, Downloads, FAQs und Anwendungsbeispiele – alle Informationen sind mit wenigen Mausklicks erreichbar:
https://support.industry.siemens.com

Technical Support

Der Technical Support von Siemens Industry unterstützt Sie schnell und kompetent bei allen technischen Anfragen mit einer Vielzahl maßgeschneiderter Angebote – von der Basisunterstützung bis hin zu individuellen Supportverträgen.

Anfragen an den Technical Support stellen Sie per Web-Formular:
www.siemens.de/industry/supportrequest

SITRAIN – Training for Industry

Mit unseren weltweit verfügbaren Trainings für unsere Produkte und Lösungen unterstützen wir Sie praxisnah, mit innovativen Lernmethoden und mit einem kundenspezifisch abgestimmten Konzept.

Mehr zu den angebotenen Trainings und Kursen sowie deren Standorte und Termine erfahren Sie unter:
www.siemens.de/sitrain

Serviceangebot

Unser Serviceangebot umfasst folgendes:

- Plant Data Services
- Ersatzteilservices
- Reparaturservices
- Vor-Ort und Instandhaltungsservices
- Retrofit- und Modernisierungsservices
- Serviceprogramme und Verträge

Ausführliche Informationen zu unserem Serviceangebot finden Sie im Servicekatalog:
https://support.industry.siemens.com/cs/sc

Industry Online Support App

Mit der App "Siemens Industry Online Support" erhalten Sie auch unterwegs die optimale Unterstützung. Die App ist für Apple iOS, Android und Windows Phone verfügbar:
https://support.industry.siemens.com/cs/ww/de/sc/2067

12.2 Links und Literatur

Tabelle 12-1

	Thema
\1\	Siemens Industry Online Support http://support.industry.siemens.com/
\2\	Downloadseite des Beitrags https://support.industry.siemens.com/cs/ww/de/view/81318674
\3\	Standardisierungsleitfaden https://support.industry.siemens.com/cs/ww/de/view/109756737
\4\	Bibliotheksleitfaden https://support.industry.siemens.com/cs/ww/de/view/109747503
\5\	Anwenderdefinierte Dokumentation bereitstellen https://support.industry.siemens.com/cs/ww/de/view/109773506/119453612171
\6\	SIMATIC S7-1200/ S7-1500 Vergleichsliste für Programmiersprachen https://support.industry.siemens.com/cs/ww/de/view/86630375
\7\	Bibliothek mit generellen Funktionen (LGF) für SIMATIC STEP 7 (TIA Portal) und SIMATIC S7-1200/ S7-1500 https://support.industry.siemens.com/cs/ww/de/view/109479728

12.3 Änderungshistorie

Tabelle 12-2

Version	Datum	Änderung
V1.0	10/2014	Erste Version nach interner Freigabe
V1.1	06/2015	Anpassungen und Korrekturen
V1.2	10/2016	Anpassungen und Korrekturen Neu: Spickzettel
V2.0	05/2020	Kategorisierung und Modernisierung • Kategorisierung nach Workflow • Erweiterung um Punkte für Performance und Design- / Architektur • Erweiterung um Programmierrichtlinien • Ergänzung von Begründungen • Überarbeitung des Spickzettels