



SIEMENS

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 kunden-spezifischen Lösungen dar, sondern bieten lediglich Hilfestellung bei typischen Aufgabenstellungen.** Die Anwendungsbeispiele unterliegen nicht den Standardtests und Qualitätsprüfungen eines kostenpflichtigen Produkts und können funktionale und Leistungsdefekte sowie andere Fehler und Sicherheitslücken enthalten. Sie sind verantwortlich für den ordnungsgemäßen und sicheren Betrieb der Produkte gemäß allen geltenden Vorschriften, einschließlich der Überprüfung und Anpassung des Anwendungsbeispiels für Ihr System, und stellen sicher, dass nur geschultes Personal es so verwendet, dass Sachschäden oder Verletzungen von Personen vermieden werden. Sie sind allein verantwortlich für jede produktive Nutzung.

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. Jede weitere Verwendung der Anwendungsbeispiele ist ausdrücklich nicht gestattet und es werden keine weiteren Rechte gewährt. Sie dürfen die Anwendungsbeispiele in keiner anderen Weise verwenden, insbesondere ist jegliches direkte oder indirekte Training oder Verbesserungen von KI-Modellen ausdrücklich untersagt.

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 Beweislage 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 und Ihnen die Lizenz jederzeit zu kündigen. 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://www.siemens.com/global/en/general/terms-of-use>).

Cybersecurity-Hinweise

Siemens bietet Produkte und Lösungen mit Industrial Cybersecurity-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 Cybersecurity-Konzept zu implementieren (und kontinuierlich aufrechtzuerhalten), das dem aktuellen Stand der Technik entspricht. Die Produkte und Lösungen von Siemens bilden einen Bestandteil eines solchen Konzepts.

Die Kunden sind dafür verantwortlich, unbefugten Zugriff auf ihre Anlagen, Systeme, Maschinen und Netzwerke zu verhindern. Diese Systeme, Maschinen und Komponenten sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn und soweit dies notwendig ist und nur wenn entsprechende Schutzmaßnahmen (z.B. Firewalls und/oder Netzwerksegmentierung) ergriffen wurden.

Weiterführende Informationen zu möglichen Schutzmaßnahmen im Bereich Industrial Cybersecurity finden Sie unter <https://www.siemens.com/cybersecurity-industry>.

Die Produkte und Lösungen von Siemens werden ständig weiterentwickelt, um sie noch sicherer zu machen. Siemens empfiehlt ausdrücklich, Produkt-Updates anzuwenden, sobald sie 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 Cybersecurity RSS Feed unter <https://www.siemens.com/cert>.

Inhaltsverzeichnis

Rechtliche Hinweise	2
1 Einleitung.....	4
1.1 Ziele	4
1.2 Vorteile eines einheitlichen Programmierstils.....	5
1.3 Gültigkeit.....	5
1.4 Abgrenzung	5
1.5 Regelverletzung und andere Vorgaben.....	5
2 Begriffsklärung	6
2.1 Regeln/Empfehlungen	6
2.2 Regelnummerierung	6
2.3 Performanz	6
2.4 Bezeichner/Name	7
2.5 Abkürzungen.....	7
2.6 Begriffe bei Variablen und Parametern	8
3 Einstellungen in TIA Portal	10
4 Globalisierung.....	14
5 Nomenklatur und Formatierung	16
6 Wiederverwendbarkeit.....	29
7 Referenzieren von Objekten (Allocieren)	34
8 Sicherheit.....	36
9 Designrichtlinien/Architektur.....	39
10 Performanz	54
11 CheatSheet	60
12 Anhang.....	61
12.1 Service und Support	61
12.2 Industry Mall	62
12.3 Links und Literatur	63
12.4 Changelog / History	64

1 Einleitung

Bei der Programmierung einer SIMATIC-Steuerung hat ein Entwickler die Aufgabe, das Anwendungsprogramm so lesbar und strukturiert wie möglich zu gestalten. Jeder Entwickler verwendet seine eigene Strategie, um diese Aufgabe zu erfüllen, z. B. die Benennung von Variablen, Bausteinen oder die Art und Weise, wie das Programm kommentiert wird. Die Entwickler verwenden unterschiedliche Philosophien, so dass es sehr unterschiedliche Anwendungsprogramme gibt, die oft nur vom jeweiligen Ersteller verstanden 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 neuer Mitarbeiter 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 beschriebenen 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) erstellt werden. Diese sind Strukturierter Text (SCL/ST), Kontaktplan (KOP), Funktionsplan (FUP) und Schrittketten (S7-GRAPH / SFC - Sequential Function Chart), sowie Signalflussplan (CFC - Continuous Function Chart) und Ursachen-Wirkungs-Matrix (CEM - Cause Effect Matrix).

Ebenso gilt dieses Dokument für Software Units, Ordner/Gruppen, Organisationsbausteine (OB), Funktionen (FC), Funktionsbausteine (FB), Technologieobjekte (TO), Datenbausteine (DB), PLC-Datentypen (UDT), Named value-Datentypen (NVT), Variablen, Konstanten, PLC-Variabellentabellen, PLC-Variablen, Anwenderkonstanten, PLC-Meldetextlisten, Beobachtungs- und Forcetabellen 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

Generell gilt, dass verwendete Namen und Bezeichner bezogen auf Funktionalität und verwendeter Schnittstellenart immer eindeutig sind. Das heißt, dass der verwendete Name auch einen Rückschluss auf die dahinterstehende Funktionalität zulässt. Es empfiehlt sich eine projektweit gültige Terminologie festzulegen.

Bei Kundenprojekten sind die geforderten Normen sowie die kunden- oder branchenspezifischen Standards des Kunden oder der verwendeten Technologie (z. B. Safety, Motion, Kommunikation, ...) 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 Ihre eigenen nummerierten 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 (Allocieren)
SE	Security: Sicherheit
DA	Design and architecture: Design und Architektur
PE	Performance: Performanz

2.3 Performanz

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

Ist von einem Performanznachteil 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	Kategorie
OB	Organisationsbaustein
FB	Funktionsbaustein
FC	Funktion
DB	Datenbaustein
TO	Technologieobjekt
UDT	PLC-Datentyp
NVT	Named value-Datentypen

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

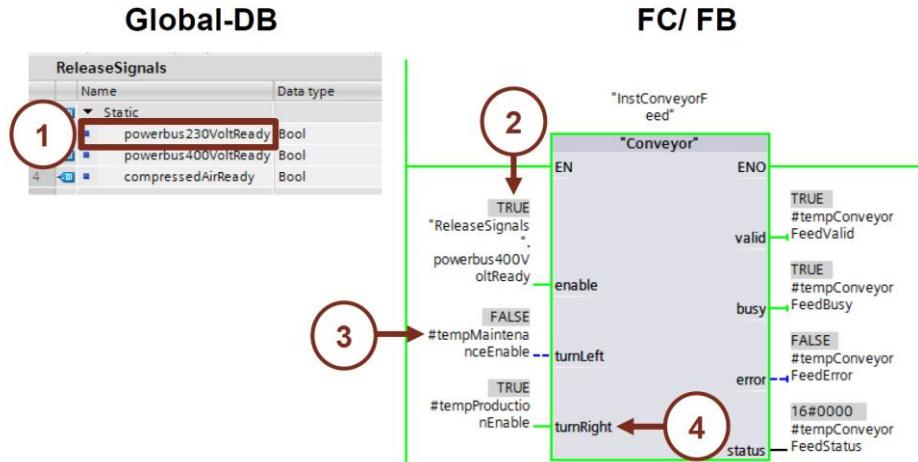


Tabelle 2-3

	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: <ul style="list-style-type: none"> - PLC-Variablen - Variablen in Bausteinen - Variablen von Strukturen ("STRUCT"), PLC-Datentypen, Named value-Datentypen - Datenbausteine/Instanz-Datenbausteine - Technologieobjekte
2	Aktualwerte	Aktualwerte sind Werte, die in einer Variablen oder Konstanten gespeichert sind (z. B. 15 als Wert einer Integer Variablen)
3	Aktualparameter	Aktualparameter sind Variablen oder Konstanten, die an den Formalparametern von Bausteinen verschaltet sind.
4	Formalparameter	Formalparameter sind die Schnittstellen von Bausteinen. Sie werden verwendet, um Daten bei Programmaufrufen zu übergeben. Formalparameter werden oft auch "Übergabeparameter" oder "Bausteinparameter" genannt.

2 Begriffsklärung

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

Formalparameter

Tabelle 2-4

Typ	Abschnitt	Funktion
Eingangs Parameter	Input	Parameter, deren Werte der Baustein liest.
Ausgangs Parameter	Output	Parameter, deren Werte der Baustein schreibt.
Durchgangs Parameter	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.

Wo Sie die Einstellungen finden: TIA Portal - Menü Extras - Eintrag Einstellungen

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.

Auswirkung auf: TIA Portal, TIA Portal Projekt, Erstellen neuer Objekte (z. B. HW Konstanten werden nach der eingestellten UI Sprache benannt)

Abbildung 3-1



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.

Auswirkung auf: TIA Portal, TIA Portal Projekt, Erstellen neuer Objekte

Abbildung 3-2



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 10pt.

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.

Auswirkung auf: TIA Portal

Abbildung 3-3

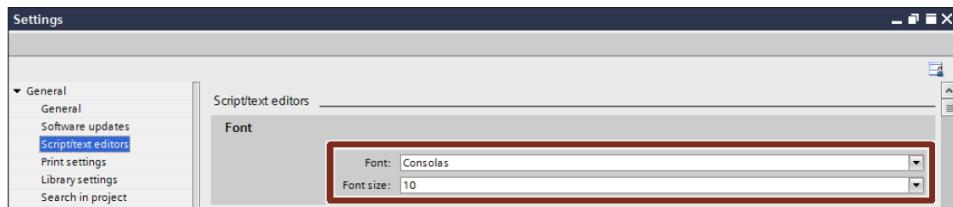


Abbildung 3-4

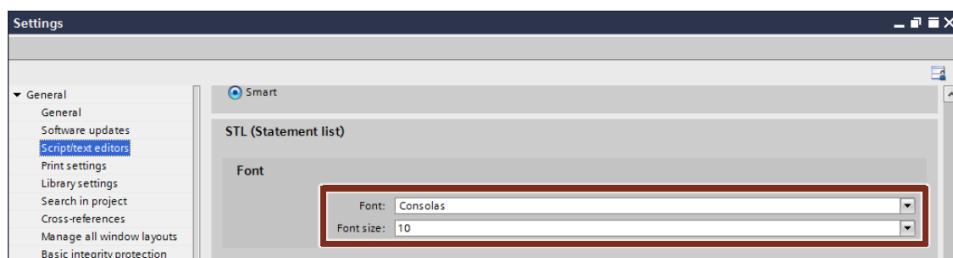
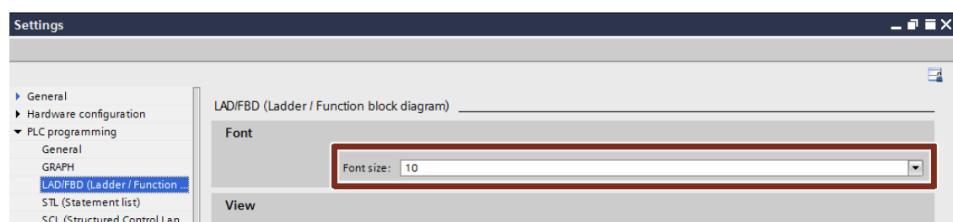


Abbildung 3-5



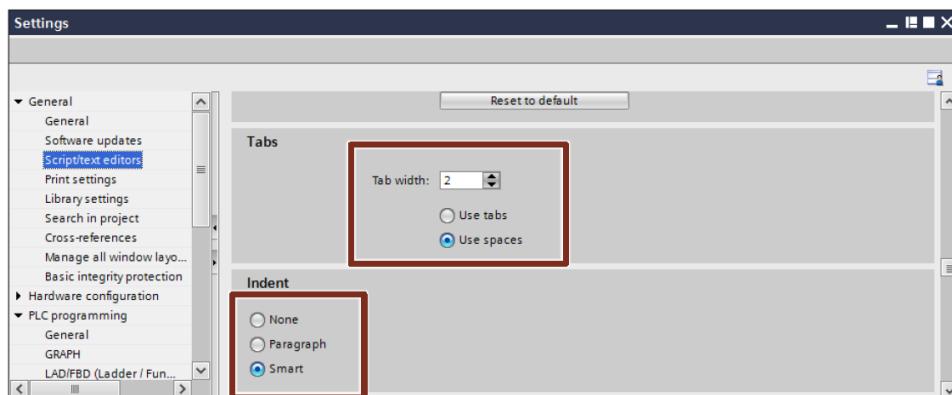
ES004 Regel: Smarte Einrückung mit zwei Leerzeichen

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.

Auswirkung auf: TIA Portal, TIA Portal Projekt, Automatische Einrückung

Abbildung 3-6



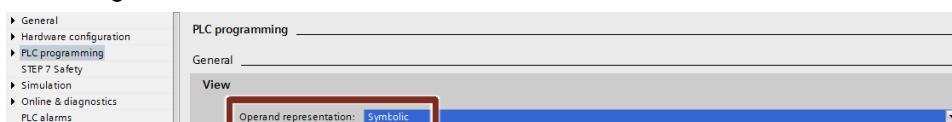
ES005 Regel: Symbolische Repräsentation von Operanden

Die Repräsentation der Operanden wird auf symbolisch festgelegt.

Begründung: Es wird vollsymbolisch programmiert.

Auswirkung auf: TIA Portal (Anpassbar im Editor)

Abbildung 3-7



ES006 Regel: IEC-konforme Programmierung

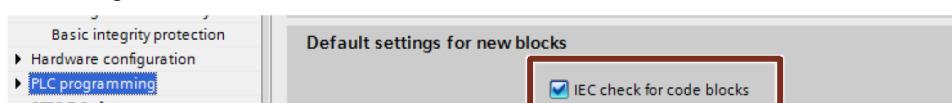
Um die IEC-konforme Programmierung zu gewährleisten, wird die IEC-Prüfung für die Bausteinerstellung standardmäßig aktiviert.

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

So werden beispielsweise implizite Typumwandlungen, bei denen die Gefahr eines Datenverlusts besteht (`Usint 0..255 <-> UInt -128..127`), oder Slice-Zugriffe auf numerische Datentypen vom Compiler als Fehler markiert.

Auswirkung auf: Erstellen neuer Objekte (Anpassbar in Objekteigenschaften)

Abbildung 3-8



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

Um den Zugriff auf eine PLC über externe Applikationen wie HMI/ OPC UA/ Web API so sicher wie möglich zu gestalten, wird die Freigabe des Zugriffs in den Standardeinstellungen deaktiviert.

Begründung: Dadurch wird erreicht, dass ein externer Zugriff nur auf Datenbereiche möglich ist, die explizit freigegeben werden.

Auswirkung auf: Erstellen neuer Objekte (Anpassbar in Objekteigenschaften)

Abbildung 3-9

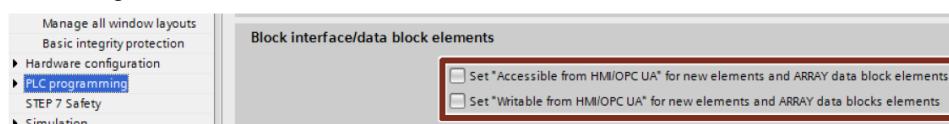
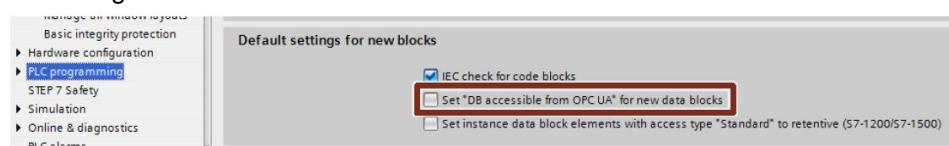


Abbildung 3-10



ES008 Regel: Automatische Wertprüfung (ENO) aktiviert

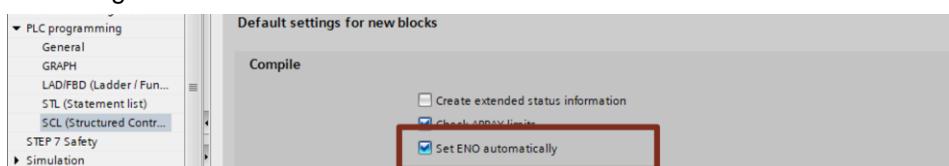
Für die automatische Prüfung von Typgrenzen und Operationen ist der EN/ENO Mechanismus zuständig. Dieser wird standardmäßig aktiviert.

Der EN/ENO Mechanismus kann später am Baustein selbst deaktiviert werden.

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

Auswirkung auf: Erstellen neuer Objekte (Anpassbar in Objekteigenschaften), Laufzeitumgebung der PLC

Abbildung 3-10



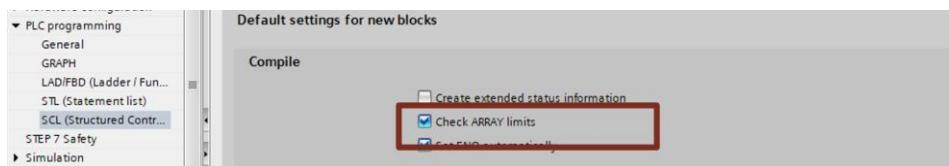
ES009 Regel: Automatische Prüfung von Arraygrenzen

Die automatische Prüfung der Arraygrenzen ist standardmäßig zu aktivieren.

Begründung: Prüfung zur Laufzeit, ob Array-Indizes innerhalb des deklarierten Bereichs für ein Array liegen. Wenn ein Array-Index den zulässigen Bereich überschreitet, wird der Freigabeausgang ENO des Bausteins auf FALSE gesetzt.

Auswirkung auf: TIA Portal, TIA Portal Projekt, Laufzeitumgebung der PLC

Abbildung 3-11



4 Globalisierung

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

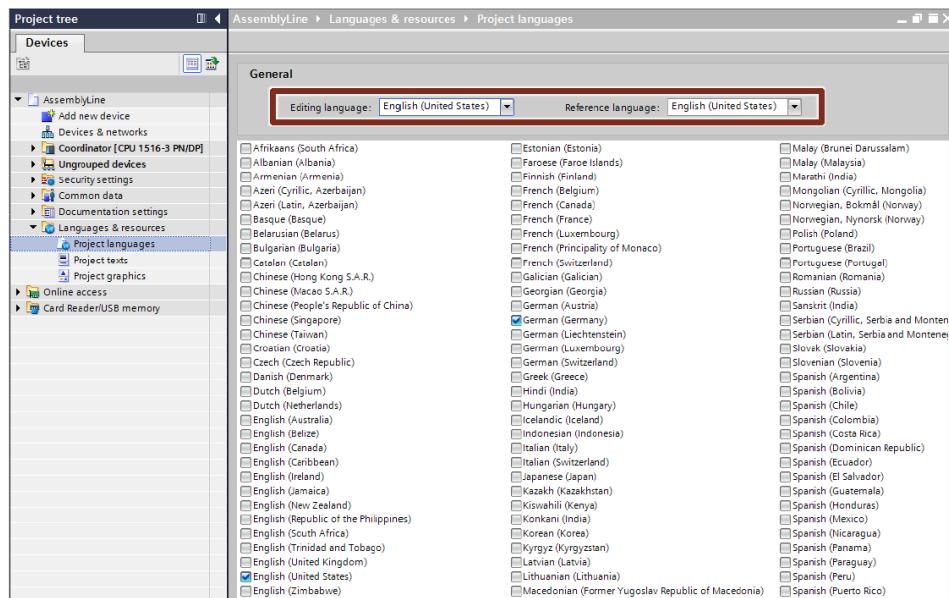
GL001 Regel: Einheitliche Sprache verwenden

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

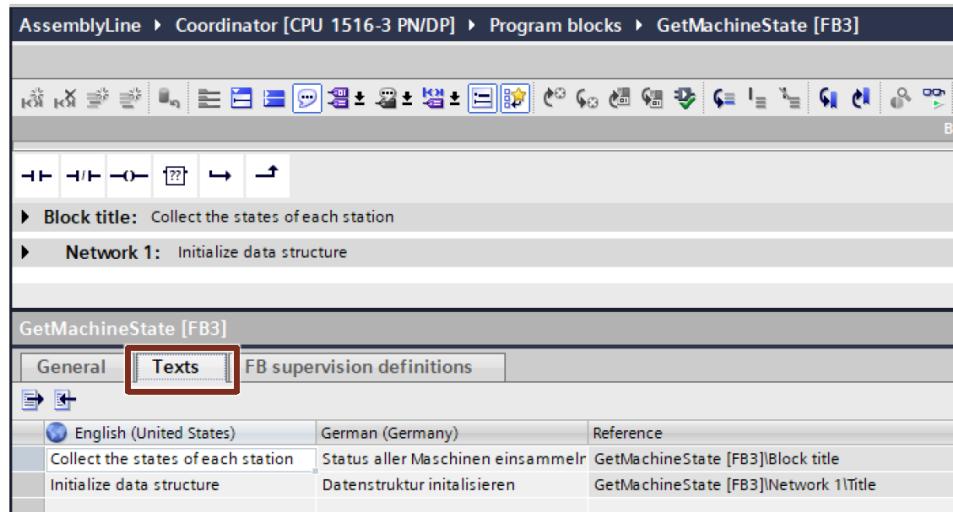


GL003 Regel: Texte in allen aktiven Projektsprachen hinterlegen

Projekttexte müssen immer in allen aktiven Projektsprachen übersetzt sein.

Hinweis In einem Bausteineditor können über den Reiter "Texte" die Übersetzungen leicht verwaltet werden.
Darüber hinaus ist es möglich, den Text zur externen Bearbeitung zu ex- und importieren.

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 Objekten 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 usw.
- Ist der Bezeichner ein Arraybezeichner, ist er im Plural zu verwenden. Unzählbare Nomen bleiben im Singular (data, information, content, management).
- Boolesche Variablen sind häufig Zustandsanzeigende Variablen. In einem solchen Falle sind Namen mit "is", "can" oder "has" am eingängigsten und verständlichsten.
- Namensräume dienen zur Strukturierung und Abgrenzung der Softwareelemente und werden entsprechend zur Projekt-Ordnerstruktur benannt.

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: Beispiel Tabelle

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

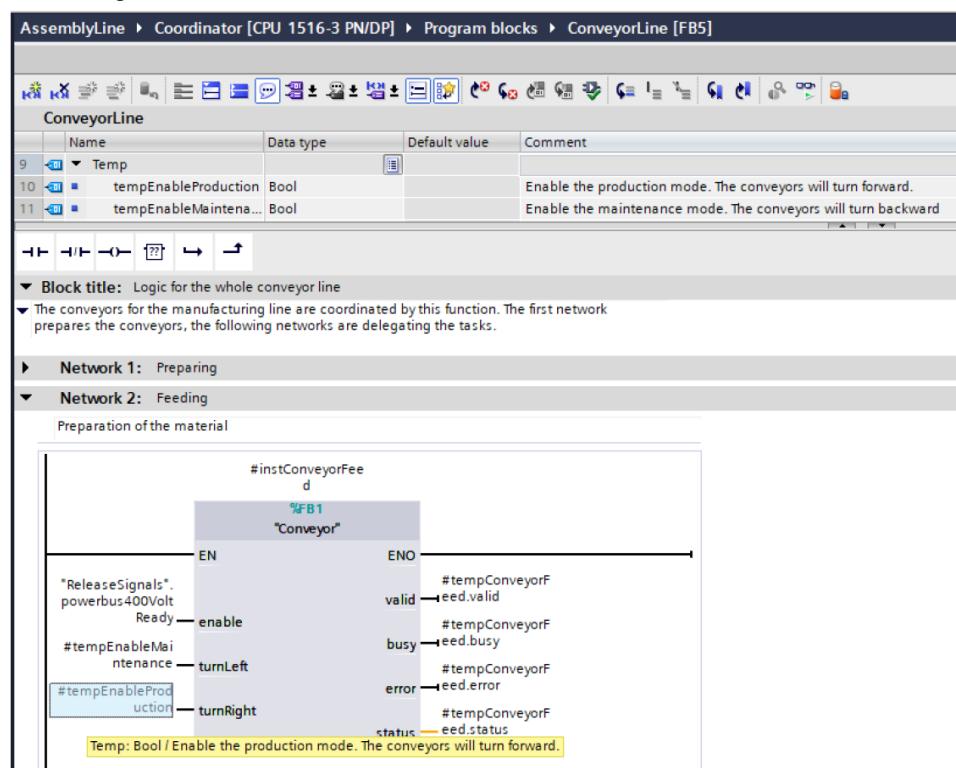
NF002 Regel: Sinnvolle Kommentare & Eigenschaften verwenden

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 (Variablen und Konstanten)
- Netzwerktitel und Netzwerkkommentare
- PLC-Datentypen, Named value-Datentypen und deren Variablen
- PLC-Variablenlisten, PLC-Variablen und Anwenderkonstanten
- PLC-Meldetextlisten, PLC-Überwachungen & -Meldungen
- 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. *.html / *.pdf) im TIA Portal Projekt hinterlegt werden. Diese Beschreibung kann der Anwender bei Bedarf mit der Tastenkombination **<Shift><F1>** als Onlinehilfe aufrufen.

Weitere Informationen dazu finden Sie in der Onlinehilfe unter dem Stichwort "Anwenderdefinierte Dokumentation bereitstellen":

<https://support.industry.siemens.com/cs/ww/de/view/109755202/114872699275>

Hinweis Mit dem Add-In *Code2Docu* ist es möglich, aus den Objekten in TIA Portal die Dokumentation zu generieren.

<https://support.industry.siemens.com/cs/ww/de/view/109809007>

NF003 Regel: Entwicklerinformationen dokumentieren

Jeder Baustein erhält einen Beschreibungskopf im Programmcode (SCL/ST) bzw. im Bausteinkommentar (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 Baustineigenschaften 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) (C)Copyright (Name der Firma) [Jahr der Erstellung] - [Jahr der letzten Revision]
- (Optional) Titel/Bausteinbezeichner
- (Optional) Beschreibung der Funktionalität
- (Optional) Name der Bibliothek
- (Optional) Abteilung/Autor/Kontakt
- Testsystem – Getestet auf PLC(s) mit Firmware Version (z. B. 1516-3 PN/DP V4.0)
- 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:

```
(C)Copyright (company) (Year of first release) - (Year of last change)
```

```
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)
Tested on 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 release

Vorlage für einen Beschreibungskopf in SCL:

```

REGION Description header
//=====
// (C)Copyright (company) [year creation] - [year revision]
//-----
// 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)
// Tested on 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 Description header

```

NF004 Regel: Präfixe und Struktur für Bibliotheken einhalten

Der Bibliotheksbezeichner hat das Präfix L und die Gesamtlänge 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, welches 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: Beispiel Tabelle

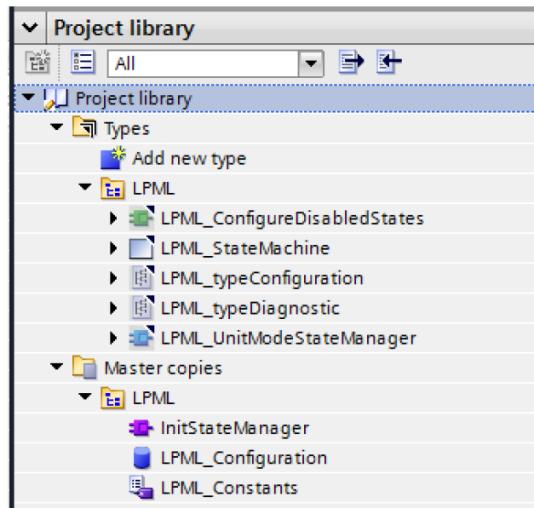
Typ	Bezeichner nach Styleguide
Bibliothek, Hauptordner der Bibliothek	LExample
PLC-Datentyp	LExample_type<Name>
Named value-Datentyp	LExample_nvt<Name>
Funktion/Funktionsbaustein	LExample_<Name>
Organisationsbaustein	LExample_<Name>
Globaldatenbaustein	LExample_<Name>
PLC-Variablenliste	LExample_<Name>
Globale Konstante	LEXAMPLE_<NAME>
Globale Konstante für Status/Fehler/...	LEXAMPLE_STATUS_<NAME>
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, welcher 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 UpperCamelCase bezeichnen

Bezeichner von TIA Portal Objekten z. B.

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

werden in UpperCamelCase (PascalCasing) geschrieben.

Folgende Regeln gelten für UpperCamelCase:

- Das erste Zeichen ist ein Buchstabe und wird großgeschrieben.
- Besteht ein Bezeichner aus mehreren Worten, wird der Anfangsbuchstabe der einzelnen Worte großgeschrieben.
- 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 lowerCamelCase bezeichnen

Bezeichner von Codeelementen z. B.

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

werden in lowerCamelCase (camelCasing) geschrieben.

Folgende Regeln gelten für lowerCamelCase:

- Das erste Zeichen ist ein Buchstabe und wird kleingeschrieben.
- Besteht ein Bezeichner aus mehreren Worten, wird der Anfangsbuchstabe der folgenden einzelnen Worte großgeschrieben.
- 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 oder Named value-Datentyps wird kein Präfix vergeben.

Hinweis Formalparameter in Bausteinen erhalten kein Präfix (z. B. in / out / inout).

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

Um temporäre und statische interne Variablen klar von Formalparametern im Code zu trennen, werden die Präfixe temp und stat verwendet.

Um statische Schnittstellen-Parameter zu PLC externen Systemen (z. B. HMI, MES,...) im statischen Bereich eines FBs zu deklarieren, werden diese mit dem Präfix ext gekennzeichnet.

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

Hinweis Statische Variablen in Global-DBs erhalten nicht das Präfix stat oder ext.

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 erfolgt.

Hinweis Datenbausteine, die von einem PLC-Datentyp abgeleitet sind, erhalten kein Präfix Inst.

PLC-Datentypen mit Präfix "type"

Der Deklaration des PLC-Datentypen wird das Präfix type vorangestellt. Die Definition einer Variablen von diesem PLC-Datentyp sowie die Elemente im PLC-Datentyp erhalten wiederum kein Präfix.

Begründung: Dies erleichtert dem Entwickler eines Bausteins die Unterscheidung zwischen PLC-Datentypen, Named value-Datentypen, Funktionsbausteinen und Basis- / System-Datentypen.

Hinweis Datenbausteine, die von einem PLC-Datentyp abgeleitet sind, erhalten kein Präfix Type oder type.

Named value-Datentypen mit Präfix "nvt"

Der Deklaration des Named value-Datentypen wird das Präfix nvt vorangestellt. Die Definition einer Variablen von diesem Named value-Datentypen sowie die Elemente im Named value-Datentypen erhalten wiederum kein Präfix.

Begründung: Dies erleichtert dem Entwickler eines Bausteins die Unterscheidung zwischen Named value-Datentypen, PLC-Datentypen, Funktionsbausteinen und Basis- / System-Datentypen.

Hinweis Variablen, die von einem Named value-Datentypen abgeleitet sind, erhalten kein Präfix nvt.

Für Anwender irrelevante Bausteine und PLC-Datentypen mit Präfix "unpub"

Sollen Bausteine und PLC-Datentypen von internen Bibliothekselementen für den Anwender in der Intellisenseliste "ausgeblendet" werden, kann dies durch Voranstellen des Präfix `unpub` umgesetzt werden.

Begründung: Durch die Verwendung des Präfixes werden die Objekte in der Intellisenseliste durch die alphabetische Reihenfolge ans Ende geschoben.

Beispiele:

`Lib_UnpubObject / Lib_unpubTypeDataType / Lib_unpubNvtNamedValue`

Tabelle 5-4: Präfix Übersicht

No.	Präfix	Typ
1	Kein Präfix	Eingangs- und Ausgangsparameter Zugriff von außerhalb möglich → enable → error
2	Kein Präfix	Durchgangsparameter Änderung der verschalteten Daten sowohl durch Benutzer als auch durch Baustein jederzeit möglich → conveyorAxis
3	Kein Präfix	PLC-Variablen und Anwenderkonstanten → lightBarrierLeft (Bezeichner für PLC-Variable) → MAX_BELTS (Bezeichner für Anwenderkonstante)
4	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)
5	temp	Temporäre Variablen Kein Zugriff auf die Lokaldaten von außerhalb möglich → templIndex
6	stat	Statische interne Variablen Kein Zugriff auf die Lokaldaten von außerhalb erlaubt → statState
7	ext	Statische Schnittstellen-Parameter Zugriff auf die Lokaldaten von PLC-Externen Systemen (z. B. HMI, MES,...) erlaubt → extInterface
8	inst	Variablen bei Multiinstanzen und Parameterinstanzen → instWatchdogTimer → instWatchdogTimers (bei Arrays von Instanzen)
9	Inst	Einzelinstanz Datenbausteine → InstConveyorFeed
10	type	PLC-Datentyp Nur der PLC-Datentyp erhält das Präfix, nicht die enthaltenen Elemente → typeDiagnostics (Bezeichner für PLC-Datentyp) → stateNumber (Bezeichner für Variable in PLC-Datentyp)
11	nvt	Named value-Datentypen Nur der Named Value Datentyp erhält das Präfix, nicht die enthaltenen Elemente → nvtStates (Bezeichner für NVT) → PROCESSING (Bezeichner für Konstante in NVT)
12	unpub	PLC-Baustein / PLC-Datentyp / Named value-Datentypen Das Präfix wird dem Namen vorangestellt, bei Bibliothekselementen nach dem Bibliothekspräfix → Lib_UnpubObject (Bezeichner für PLC-Baustein) → Lib_unpubTypeDataType (Bezeichner für PLC-Datentyp) → Lib_unpubNvtNamedValue (Bezeichner für Named value-Datentypen)

NF008 Regel: Bezeichner von Konstanten in UPPER_CASE schreiben

Die Namen von Konstanten werden in UPPER_CASING geschrieben - durchgehend in Großschrift (GROSS_SCHREIBUNG / UPPER_SNAKE_CASING / SCREAMING_SNAKE_CASE / ALL_CAPS). Zum Trennen und Erkennen einzelner Worte oder Abkürzungen ist jeweils ein Unterstrich zwischen den einzelnen Worten oder Abkürzungen einzusetzen.

Zu den Konstanten zählen folgende Elemente:

- Globale Konstanten
- Lokale Konstanten
- Elemente in Named value-Datentypen

Folgende Regeln gelten für Konstanten in UPPER_CASING:

- Das erste Zeichen ist ein Buchstabe und wird großgeschrieben.
- Alle folgenden Zeichen sind entweder Großbuchstaben oder Ziffern.
- Wenn ein Bezeichner aus mehreren Wörtern zusammengesetzt ist, werden die Wörter durch einen Unterstrich getrennt.

Abbildung 5-4: Konstanten in einem FB

Constant				
MAX_VELOCITY	Real	10.0	Maximum velocity of conveyor	
MAX_NO_OF_AXES	Dint	3	Maximum number of axes	

Abbildung 5-5: Konstanten in einem Named value-Datentypen

1	TYPE	
2	nvtStatus : Word (
3	STATUS_NO_ERROR := 16#0000,	
4	ERR_UNDEF := 16#8600	
5);	
6	END_TYPE	

Hinweis TRUE und FALSE sind ebenfalls Konstanten.

NF009 Regel: Zeichensatz für Bezeichner einschränken

Für Objekt- und Codebezeichner 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 einzelnen 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 Variablenamen 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: Beispiele für Abkürzungen

Abbrev.	Typ
Min	Minimum
Max	Maximum
Lim	Limit
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 5-6

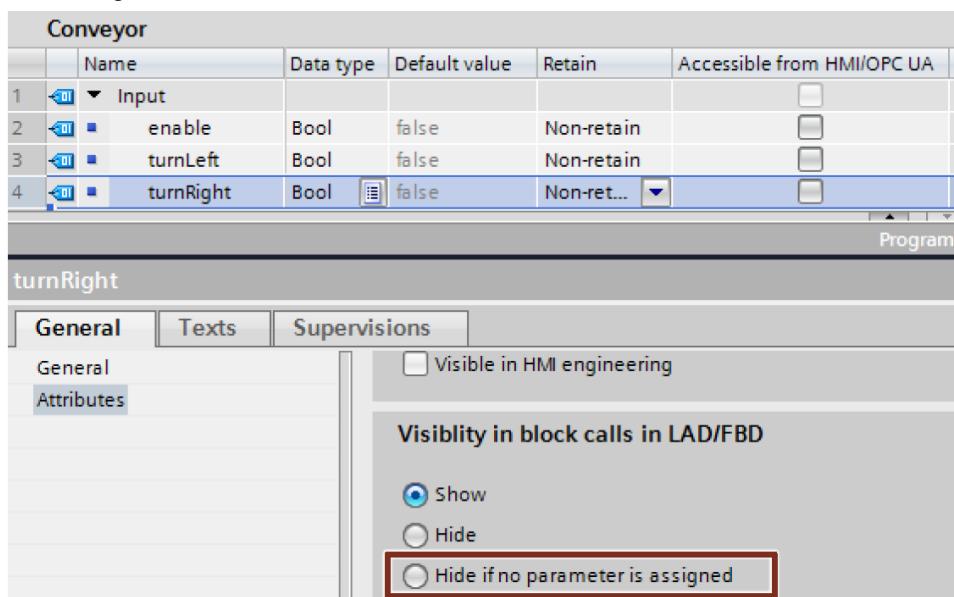
Korrekte Initialisierung			Falsche Initialisierung		
statTriggerOld	Bool	FALSE	statTriggerOld	Bool	false
statStatus	Word	16#7000	statStatus	Word	123
statStep	DInt	101	statStep	DInt	16#0
statVelocity	LReal	0.0	statVelocity	LReal	16#000
statCommand	Byte	16#01	statCommand	Byte	16#1
statFlags	Byte	2#1010_0101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

NF013 Empfehlung: Optionale Formalparameter ausblenden

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

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

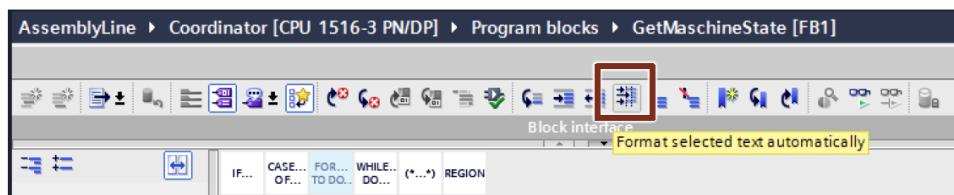
Abbildung 5-7



NF014 Regel: SCL-Code sinnvoll formatieren

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

Abbildung 5-8



1. Vorrangig Zeilenkommentar // verwenden

Es ist zwischen zwei Arten von Kommentaren zu unterscheiden:

- Ein Kommentarabschnitt `(*...*)` oder ein mehrsprachiger Kommentarabschnitt `(/*...*/)` ist in einer oder mehreren Zeilen vor dem entsprechenden Code Abschnitt zu setzen und beschreibt eine Funktion, oder einen Codeabschnitt.
- Ein Zeilenkommentar `//` beschreibt den Code einer einzelnen Zeile und ist, wenn möglich, an das Ende der Codezeile 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.

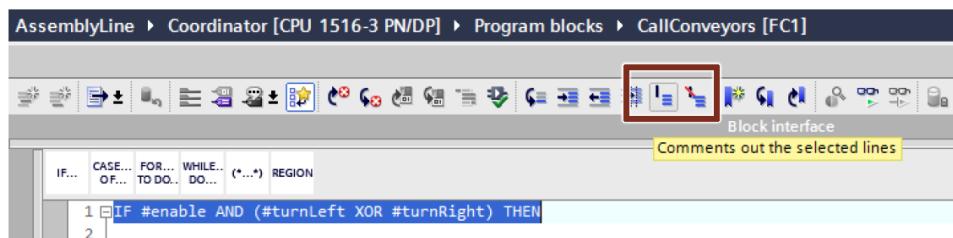
Ein Kommentar liefert dem Leser Information darüber, warum etwas an der betreffenden Stelle getan wird. Er darf nicht den Code als redundanten Klartext wiedergeben. Der Code beschreibt bereits was getan wird, daher ist es wichtig zu dokumentieren, warum es getan wird.

Begründung: Man vermeidet dadurch Syntaxprobleme durch Einfügen und möglicherweise Verschachteln von Kommentarabschnitten mit `(*...*)` oder `(/*...*/)`.

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 diese zu löschen.

Abbildung 5-9



2. Vor und nach Operatoren werden Leerzeichen gesetzt

Vor und nach Operatoren steht ein Leerzeichen.

3. Ausdrücke sind immer in Klammern

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

Beispiel:

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

4. 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. Dieses gilt ebenso 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;
```

5. Zeilenumbrüche bei Teilbedingungen

Bei komplexen Ausdrücken ist es sinnvoll, jede Teilbedingung durch einen Zeilenumbruch hervorzuheben. Operatoren stehen am Anfang der Zeile vor der Bedingung.

Schlecht lesbares Beispiel:

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

Besser lesbares Beispiel:

```
#tempResult := FALSE
OR (#enable AND #tempEnableOld)
OR (#enable AND #isValid AND NOT (#hasError OR #hasWarning))
; // semicolon in separate line, so each line can be commented out
```

6. 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`).

Beispiel A1:

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

Beispiel A2:

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

Beispiel B:

```
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.

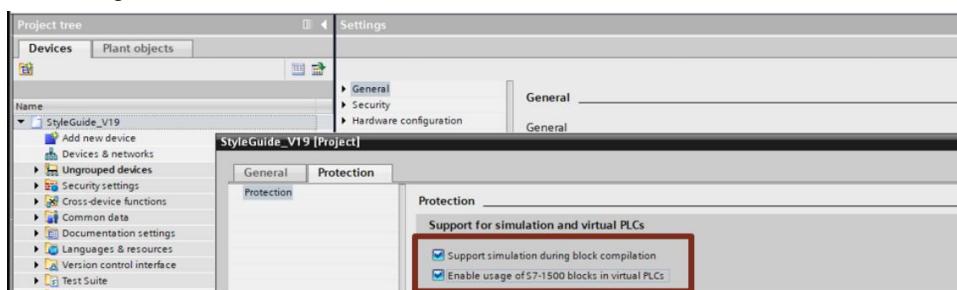
RU001 Empfehlung: Unterstützung für verschiedene Zielsysteme aktivieren

Aktivieren Sie die Simulierbarkeit und die Nutzbarkeit für virtuelle CPUs über die Projekteigenschaft.

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

Hinweis	Mit dem Hinzufügen dieser Optionen für Know-how geschützte Programmbausteine wird potentiell das Niveau des Schutzes reduziert, da der Know-how Schutz unter anderem von dem ausführenden Zielsystem abhängig ist. In der auf unterschiedlichsten PC-Systemen betreibbaren Laufzeitumgebungen kann der Schutz prinzipbedingt nicht so hoch sein, wie in einer dedizierten Hardware-CPU.
----------------	--

Abbildung 6-1



RU002 Regel: Mit Bibliotheken versionieren

Wiederverwendbare Elemente (z. B. FC, FB, PLC-Datentypen, Named value-Datentypen), die vom Anwender nicht verändert werden sollen, werden als Typen in einer Bibliothek bereitgestellt.

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 Baustieneigenschaften 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) 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 (Major) bezeichnet die Zahl ganz links.
- Bei kompatibler Erweiterung der bestehenden Funktionalität wird die zweite Stelle (Minor) inkrementiert und die dritte Stelle zurückgesetzt.
- Die dritte Stelle (Patch) in der Software-Versionierung kennzeichnet Änderungen, die keine Funktionserweiterung beinhalten und keine neue Dokumentation erforderlich machen, z. B. reine Fehlerbehebungen.
- 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: Versionsnummern und deren Verwendung

Bibliothek	FB1	FB2	FC1	FC2	Kommentar
1.0.0	1.0.0	1.0.0	1.0.0	-	Freigegeben
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
1.0.3	1.0.1	1.0.2	1.0.0	-	Anpassung Dokumentation von FB2
1.1.0	1.1.0	1.0.1	1.0.0	-	Kompatible Funktionserweiterung an FB1
1.2.0	1.2.0	1.0.1	1.0.0	-	Kompatible Funktionserweiterung an FB1
2.0.0	2.0.0	1.0.1	2.0.0	-	Inkompatible Funktionserweiterung 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, kann als Major oder als Minor Release gekennzeichnet werden
3.0.1	2.0.0	1.0.2	2.0.1	1.0.1	Fehlerbehebungen

Hinweis

Diese Regel enthält nur allgemeine Angaben zur Versionierung. Eine Erklärung für die automatische Versionierung von Bibliothekselementen ist im Bibliothekleitfaden 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
- Named value-Datentypen

RU004 Regel: Nur lokale Variablen verwenden

Innerhalb eines wiederverwendbaren Objekts (z. B. FCs, FBs, PLC Datentypen)

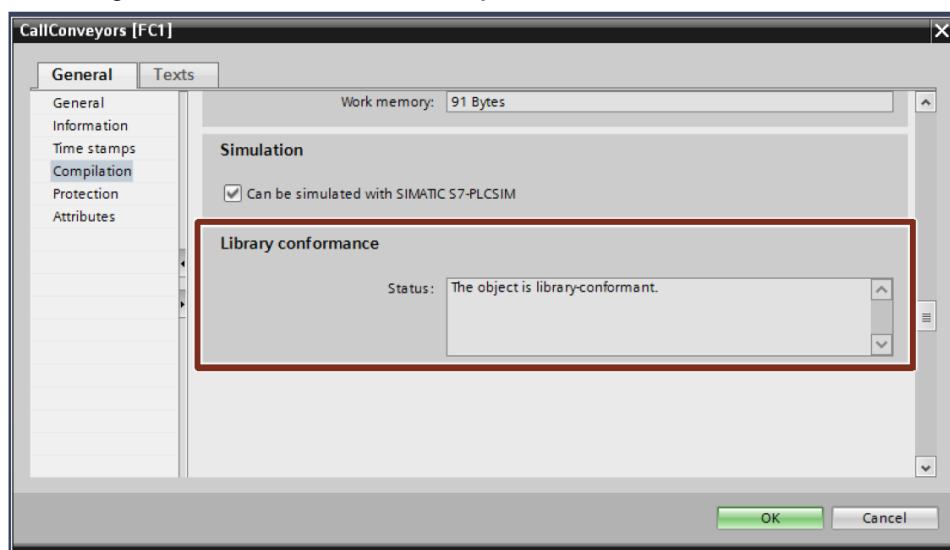
1. dürfen nur dessen lokale Variablen genutzt werden.
2. ist der Zugriff auf globale Konstanten lediglich für die Definition von Arraygrenzen zur Skalierung von unterschiedlichen Mengengerüsten zulässig.
Siehe auch "AL002 Empfehlung: Arraygrenze von 0 bis Konstante definieren"

Die Bibliothekskonformität eines Bausteins oder PLC-Datentyps zeigt an, dass keine der folgenden Zugriffsarten verwendet wird:

- 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 Objekteigenschaften automatisch den Status "Das Objekt ist bibliothekskonform". Dieser Status kann somit als einfache Überprüfungsmöglichkeit für die Einhaltung dieser Regel verwendet werden.

Abbildung 6-2: Bibliothekskonformes Objekt



RU005 Regel: Symbolische Konstanten verwenden

Im Sinne gekapselter Bausteine sind lokale Konstanten oder Named value-Datentypen zu verwenden. Werden globale Konstanten verwendet, sind sie nach Möglichkeit über die Formalparameter zu übergeben. Globale Konstanten sind in eigenen PLC-Variabellisten 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.

Hinweis Named value-Datentypen sind nur innerhalb von Software Units verfügbar.

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

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

Beispiel:

Abbildung 6-3

Blower				
	Name	Data type	Default value	Retain
1	Input			
2	velocity	Real	0.0	Non-retain
3	Output			
4	InOut			
5	Static			
6	statVelocity	Real	0.0	Non-retain
7	Temp			
8	Constant			
9	MAX_VELOCITY	Real	10.0	

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

Abbildung 6-4

DemoNVT				
	Name	Data type	Default value	Retain
1	Input			
2	enable	Bool	false	Non-retain
3	Output			
4	error	Bool	false	Non-retain
5	status	nvtStatus	nvtStatus#STATUS_SUCCESSFULL	Non-retain
6	hasWarning	Bool	false	Non-retain

```
IF #tempError THEN
    #status := nvtStatus#ERR_UNDEF;
ELSE
    #status := nvtStatus#STATUS_SUCCESSFUL;
END_IF;

#error := #tempStatus.%X15;
```

RU006 Regel: Vollsymbolisch programmieren

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

Weiter werden die Nummern von Systemobjekten (z. B. HW-ID, OB, DB, ...) nicht direkt verwendet (keine "Magic Numbers"), sondern es werden Systemkonstanten verwendet.

Begründung: Dies 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. Anstelle dieser Datentypen werden die IEC-konformen Typen, z. B. der IEC-Timer eingesetzt, die auch als Multiinstanzen verwendbar sind.

Das Speichern von Daten, die im gesamten Anwenderprogramm benötigt werden, erfolgt in Globaldatenbausteinen.

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 erreicht man eine einheitliche Ausgangsbasis für alle Programmierer. Die durch die Vorlagen bereitgestellten Funktionalitäten können als validiert angesehen werden und tragen so erheblich zur Zeitsparnis 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 findet man als Kopiervorlagen in der Bibliothek mit generellen Funktionen (LGF):

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

7 Referenzieren von Objekten (Allocieren)

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

Für Array-Grenzen gilt Folgendes:

1. Der Startindex ist 0 (Untergrenze)
2. Endet mit einer symbolischen Konstante (Obergrenze)

Beachten Sie die folgenden Regeln und Empfehlungen:

- Bei Arrays, die nur in einem Baustein verwendet werden, muss die Konstante als lokale Konstante definiert werden.
Der Zugriff auf globale Konstanten ist nur für die Definition von Array-Grenzen zur Skalierung verschiedener Mengengerüste zulässig.
Siehe „RU004-Regel: Nur lokale Variablen verwenden“
- Als Datentyp für die Konstante wird empfohlen DInt zu wählen, um den gleichen Datentyp wie für die Variable des Schleifenindex / Arrayzugriff zu verwenden.
Siehe „PE008 Empfehlung: Steuer-/Indexvariablen als „DInt“ deklarieren“

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

Muss ein Array unbekannter Größe über die Formalparameter übergeben werden können, 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...

1. alle Zeichenketten auf die erforderliche Länge zu begrenzen.
2. symbolische Konstanten zur Angabe der Länge zu verwenden.

Begründung: Das verhindert zum einen unnötiges Allokieren von Speicher und zum anderen bringt es Performanz 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 und 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 bereitzustellen.

Hinweis Weitere Informationen zum Thema Fehlerbehandlung finden Sie in “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 Objekte verschaltet werden, haben Werte kleiner 0.0 eine besondere Bedeutung. Je nach Parameter wird stattdessen der im Technologie Objekt 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, siehe auch "ES007 Regel: Expliziter Datenzugriff per HMI/OPC UA/Web API".

Im Folgenden die Übersicht für die zulässigen Zugriffseinstellungen:

(Zuordnung: ✓ = Aktiv / X = Inaktiv / ★ = optional)

No.	Typ	Erreichbar	Schreibbar	Sichtbar
1	Input / Eingang	✓	✓	★
2	Output / Ausgang	✓	X	★
3	InOut / Durchgang	✓	✓	★
4	Statisch Interface	✓	✓	★
5	Statisch Intern	X	X	X
6	PLC Datentyp	✓	✓	★
7	Global DB Interface	✓	✓	★
8	Global DB Intern	X	X	X

1. Eingabe-Parameter: Lesender und Schreibender Zugriff
2. Ausgabe-Parameter: Lesender Zugriff
3. Durchgangs-Parameter: Lesender und Schreibender Zugriff
4. Statische Interface-Variablen: Lesender und Schreibender Zugriff (Präfix ext)
5. Statische Interne-Variablen: Keinerlei Zugriff (Präfix inst, stat)
6. PLC Datentyp: Lesender und Schreibender Zugriff
7. Globaler DB - Interface-Variablen: Lesender und Schreibender Zugriff
8. Globaler DB - Interne-Variablen: Keinerlei Zugriff

Zur Unterscheidung Interface/Interne Variablen siehe Regeln:

- "DA006 Regel: Auf statische Variablen nur lokal zugreifen"
- "NF007 Regel: Präfixe verwenden"

Hinweis

Im Editor von PLC-Datentypen ist es sinnvoll 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 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 Bausteinaufrufs 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

Durch die Verwendung von Fehler OBs können unerwartete PLC Stops vermieden werden. Werden Organisationsbausteine zur Fehlerbehandlung eingesetzt, so müssen diese auch ausgewertet werden.

Beispiele für mögliche Fehler OBs:

- Zeitfehler-OB (Time error OB / CYCL_FLT [OB 80])
- Diagnosealarm-OB (Diagnostic interrupt OB / I/O_FLT1 [OB82])
- Ziehen/Stecken-OB (Pull/plug interrupt OB / I/O_FLT2 [OB83])
- Baugruppenträgerfehler-OB (Rack failure OB / RACK_FLT [OB86])
- Programmierfehler-OB (Programming error OB / PROG_ERR [OB121])
- Peripheriezugriffsfehler-OB (I/O access error OB / MOD_ERR [OB122])
- ProDiag OB

Beispiele für mögliche minimale Reaktionen:

- Bit Alarm
- ProDiag Alarm
- Program Alarm `Program_Alarm`
- Diagnose Puffer Eintrag `Gen_UsrMsg` / `GEN_DIAG`
- ...

Begründung: Damit Fehler nicht dauerhaft unerkannt bleiben und darauf reagiert werden kann, ist die minimale Anforderung die Auswertung eines Fehlers und eine Meldung an den Verantwortlichen.

SE007 Regel: Wertebereiche für Real/LReal-Vergleiche verwenden

Für den Vergleich von Real/LReal Werten werden nur Vergleichsoperatoren wie `<`, `>`, `<=`, `>=` verwendet. TIA Portal bietet auch `IN_RANGE` und `OUT_RANGE` für verschiedene Programmiersprachen an.

Begründung: Da Real/LReal im IEEE754-Format gespeichert sind, haben sie eine begrenzte Genauigkeit. Durch die Verwendung von toleranzbasierten Vergleichen vermeiden Sie unerwartete Ergebnisse aufgrund von möglichen Rundungsfehlern und Darstellungsdiskrepanzen. Deshalb sollte `=` nicht verwendet werden.

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.

- Zusammengehörige Bausteine in einem Ordner/einer Gruppe zusammenfassen
- 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. Regionen

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 Einblendens 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 werden hier bereits Verriegelungen und Überwachungen vom System bereitgestellt.

Signalflussorientiert (CFC - Continuous Function Chart)

CFC (Continuous Function Chart) wird insbesondere für verfahrenstechnische oder strukturierte Automatisierungslösungen eingesetzt. Sie verknüpfen z. B. selbst erstellte Bausteine oder von CFC gelieferte Anweisungen mit den Operanden Ihres Zielsystems. Mit Hilfe eines CFC-Charts bleibt auch ein komplexes Anwenderprogramm übersichtlich. Prozesswerte sowie Ein- und Ausgangsparameter der Anweisungen und Bausteine können Sie zum Testen online überwachen.

Ursachen-Wirkungs-Matrix (CEM - Cause Effect Matrix)

CEM (Cause-Effect-Matrix) ist eine Programmiersprache, mit der Sie übersichtlich und schnell unmittelbare Ursache-Wirkungs-Beziehungen definieren können. Dabei beschreiben Sie bestimmte Prozessereignisse und definieren mögliche Prozessreaktionen. Diese weisen Sie einander in einer zweidimensionalen Matrix zu.

DA003 Regel: Bausteineigenschaften setzen/prüfen

In den Bausteineigenschaften sind folgende Einstellungen zu aktivieren:

1. **Autonummerierung:** Bausteine (FC, FB, DB, TO) werden nur mit aktiver automatischer Nummernvergabe ausgeliefert. Beachten Sie, dass die Abarbeitung eines Organisationsbausteins von seiner Nummer/Priorität abhängig ist.
2. **IEC-Check:** Um die IEC konforme Programmierung zu gewährleisten wird der IEC-Check aktiviert. Die Kompatibilität von Operanden bei Vergleichsoperationen und arithmetischen Operationen wird gemäß den IEC-Regeln geprüft. So wird die typkonforme und typsichere Programmierung sichergestellt. Nicht kompatible Operanden müssen explizit konvertiert werden.
So werden beispielsweise implizite Typumwandlungen, bei denen die Gefahr eines Datenverlusts besteht (`USint 0..255 <-> Uint -128..127`), oder Slice-Zugriffe auf numerische Datentypen vom Compiler als Fehler markiert.
3. **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.
Siehe "AL001 Regel: Multiinstanzen statt Einzelinstanzen nutzen"
- **Bibliothekskonformität:** Siehe "RU004 Regel: Nur lokale Variablen verwenden".

Abbildung 9-1: Autonummerierung

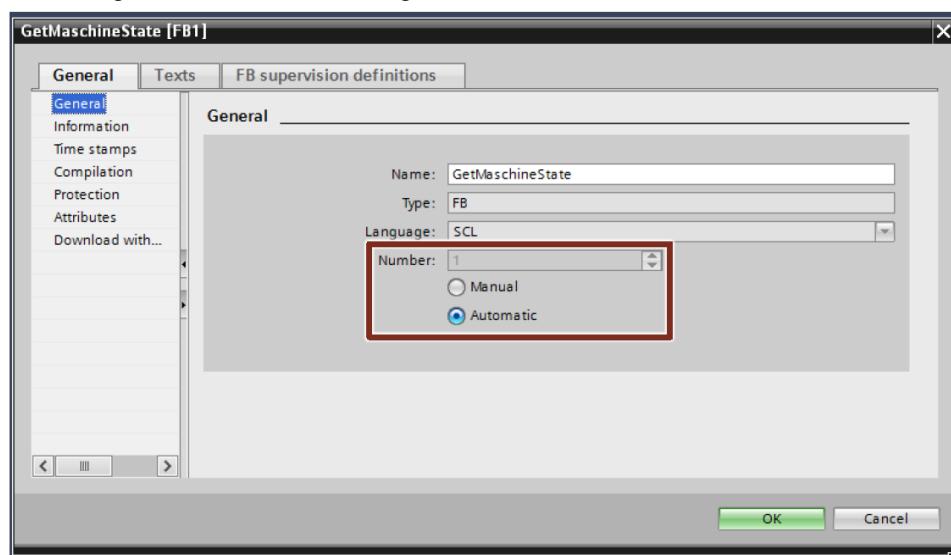


Abbildung 9-2: IEC-Check, ENO, Optimierter Zugriff, Multiinstanz Fähigkeit

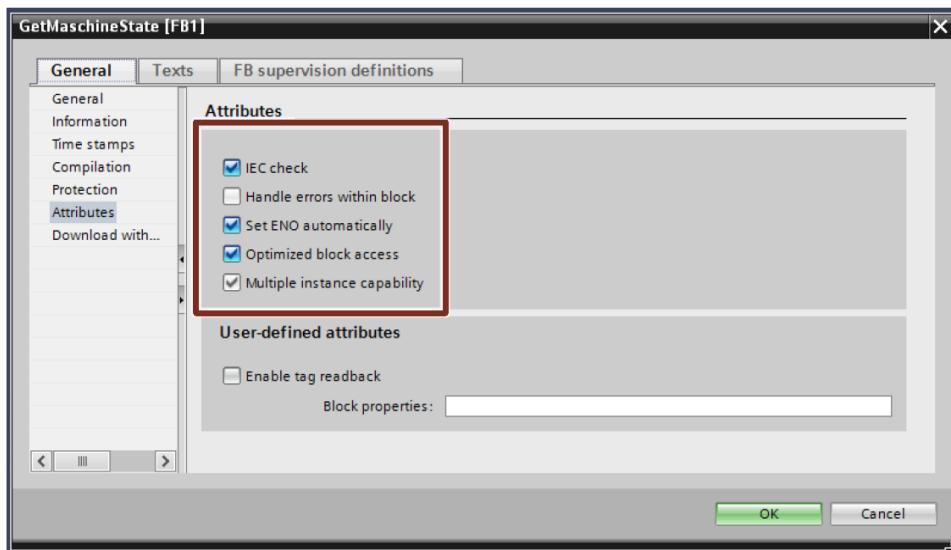
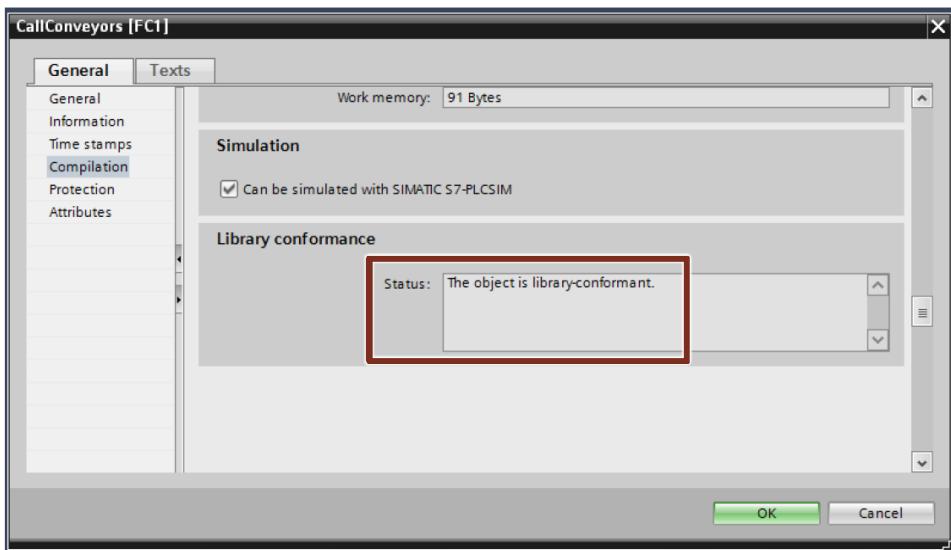


Abbildung 9-3: Bibliothekskonformität



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 oder innerhalb eines PLC-Datentyps definiert, um bei Bedarf Variablen innerhalb des Bausteins oder PLC-Datentyps durch die Gruppierung übersichtlicher darzustellen.

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.

Hinweis

Variablen von PLC-Datentypen können, durch Vorhalten einer leeren/ungenutzten Variablen vom gleichen Typ, mit einer Zuweisung einfach initialisiert und mit entsprechenden Startwerten vorbelegt werden. Bei einer Änderung am Datentyp ist dadurch keine weitere Nacharbeit für die Initialisierung erforderlich.

DA005 Regel: Daten nur über Formalparameter austauschen

Der Datenaustausch innerhalb der PLC bei FBs oder FCs erfolgt grundsätzlich über die Bausteinschnittstelle (Eingangs-, Ausgangs- oder Durchgangsparameter).

Werden FBs oder FCs als Aufrufumgebung oder zur Strukturierung der Software eingesetzt und benötigen keine Schnittstelle zur Wiederverwendung, kann auf Formalparameter verzichtet werden. Der Datenaustausch erfolgt in diesem Fall ausschließlich über den direkten Zugriff auf Globaldaten. Eine Vermischung beider Zugriffsarten in einem Baustein ist nicht zulässig.

Begründung: Im Sinne gekapselter Bausteine werden Daten über die Schnittstelle entkoppelt und Abhängigkeiten aufgelöst. Dadurch wird die Datenkonsistenz bei mehrfachen Aufrufen (ggf. an unterschiedlichen Programmstellen) sichergestellt.

Benötigt eine Aufrufumgebung/Strukturelement keine Schnittstelle zur Wiederverwendung, können durch den Verzicht auf Formalparameter Aufwände reduziert werden.

Hinweis

Es ist zulässig, direkt auf Eingangsvariablen unterlagerter Instanzen zu schreiben und von deren Ausgangsvariablen zu lesen. Dafür sind keine zusätzlichen Variablen notwendig und unnötige Kopiervorgänge werden vermieden.

```
instCall.execute := TRUE;
instCall();
IF instCall.done THEN
; // next step
ELSIF instCall.error THEN
; // do something....
END_IF;
```

DA006 Regel: Auf statische Variablen nur lokal zugreifen

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

Werden statische Variablen als Schnittstelle zu PLC-externen Systemen benötigt, z. B. zu dem Baustein zugehörigen HMI Faceplate, sind diese explizit zu kennzeichnen.

Siehe auch:

- “NF007 Regel: Präfixe verwenden”
- “SE004 Regel: Datenzugriff per HMI/OPC UA/Web API selektiv aktivieren”

Begründung: Bei direktem Zugriff auf interne 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 FBs 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

1. Die Ausgangsparameter und Rückgabewerte werden pro Bearbeitungszyklus genau einmal mit einem Wert beschrieben. Dies muss möglichst gemeinsam und am Ende passieren.
2. 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 Parameter
- Nie durchlaufener Code
- Auskommentierter Code
- Nie verwendete Variablen
- Nie verwendete Anwenderkonstanten
- Nie verwendete PLC-Datentypen
- Nie verwendete Named value-Datentypen
- 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 man ihn auf alle asynchronen Funktionsbausteine anwenden kann. Asynchron heißt in diesem Zusammenhang: Alle Bausteine, welche 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, besitzen 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 findet man 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 benutzt, müssen mindestens die Ausgangsparameter valid, error und busy verwendet werden.

Enthält ein Baustein, der das Enable-Verhalten verwendet, weitere unterlagerte Kommandos, so sind diese wiederum nach Enable- oder Execute-Verhalten zu implementieren. Unterlagerte Kommandos erhalten ebenfalls eine Rückmeldung mit welcher der Zustand des/der Kommandos erkennbar ist, z. B.:

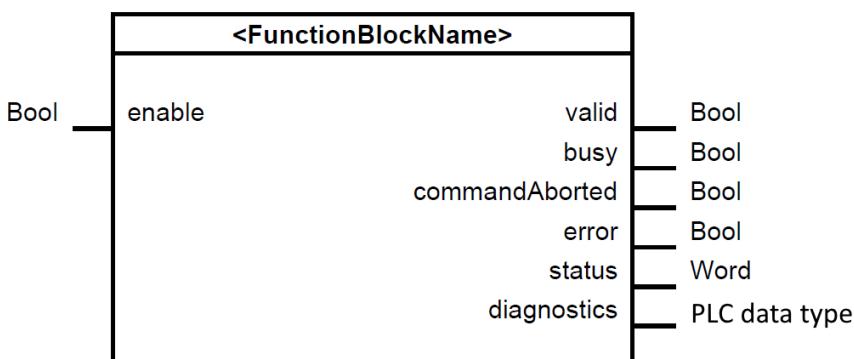
- Enable-Kommando:
commandA → commandBusy
- Execute-Kommando:
commandA → commandBusy & commandDone
- Mehrere Kommandos - zusätzlicher Parameter zur Unterscheidung des aktiven Kommandos
commandA → commandASelected
commandB → commandBSelected
commandAborted

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).
commandA	Bool	Optionaler Eingang (Name funktionsabhängig) für ein unterlagertes Kommando das mit einer steigenden Flanke aktiviert wird, kann nach Enable- oder Execute-Verhalten implementiert werden.

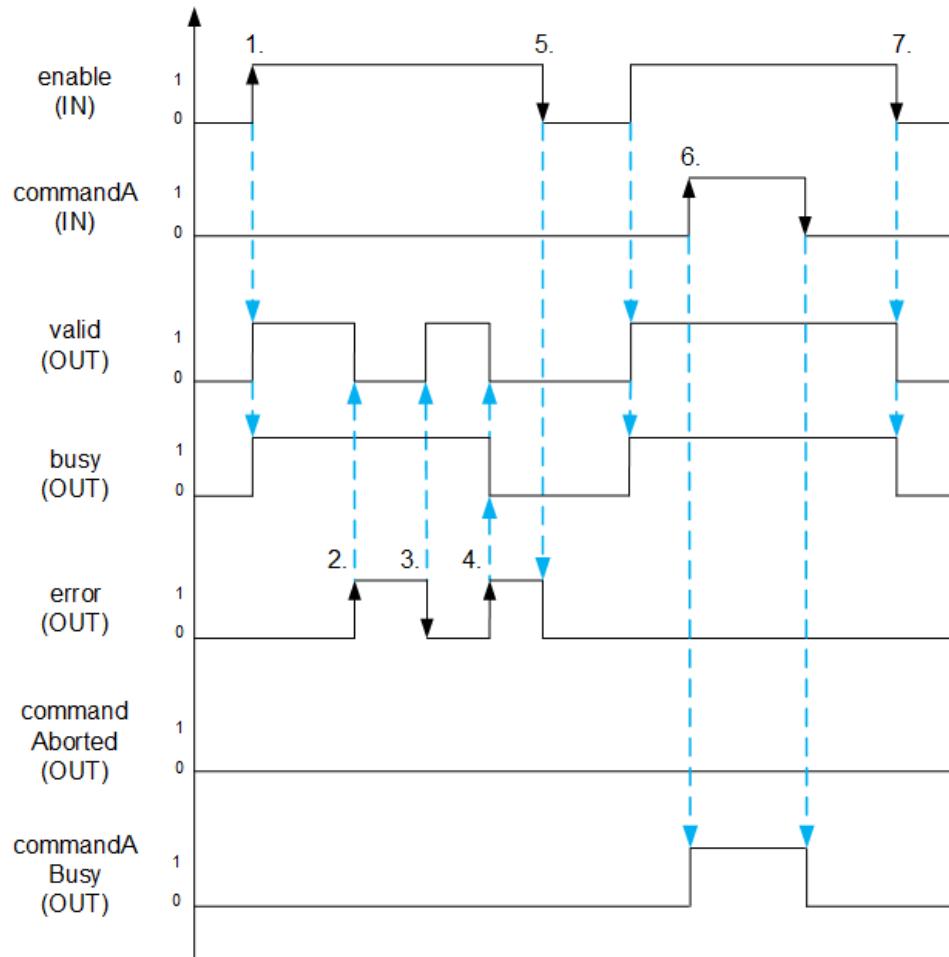
Bezeichner	Datentyp	Bedeutung
Ausgangsparameter		
valid oder enabled	Bool	Die Ausgänge valid/error sind gegenseitig exklusiv . Der Ausgang ist gesetzt, solange gültige Ausgangswerte verfügbar sind und der enable Eingang gesetzt ist. Sobald ein Fehler ansteht, wird der Ausgang valid zurückgesetzt. Nach PLCopen kann anstelle von valid auch enabled verwendet werden.
busy	Bool	Der FB ist gerade mit der Bearbeitung eines Kommandos beschäftigt. Neue Ausgangswerte können erwartet werden. Der Ausgang busy wird mit einer steigenden Flanke von enable gesetzt und bleibt gesetzt, solange der FB Aktionen ausführt.
error	Bool	Die Ausgänge valid/error sind gegenseitig exklusiv . Eine steigende Flanke des Ausgangs signalisiert, dass ein Fehler während der Abarbeitung des FB aufgetreten ist.
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 "DA015 Empfehlung: Unterlagerte Informationen durchreichen" beschrieben.
commandBusy	Bool	Optionaler Ausgang, der anzeigt, dass ein Kommando des FBs abgearbeitet wird.
commandDone	Bool	Optionaler Ausgang, der anzeigt, dass ein Kommando des FBs abgearbeitet wurde (bei Execute Kommandos)
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 gehalten wird. Am Funktionsbaustein der Positionierung wird dann der Ausgang commandAborted gesetzt, da dieser Auftrag durch das Halt Kommando abgebrochen wurde.
commandASelected	Bool	Optionaler Ausgang (Name kommandoabhängig), der das letzte aktive Kommando des FBs anzeigt und mit einem anderem Kommando oder commandDone zurückgesetzt wird. Dieser Parameter wird bei mehr als einem unterlagerten Kommando zur Unterscheidung benötigt.

Abbildung 9-4: Beispiel Baustein Schnittstelle mit enable



Signalablaufdiagramm eines Bausteins mit enable:

Abbildung 9-5



DA012 Regel: Einmalige asynchrone Abarbeitung mit "execute"

Bausteine, welche 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` findet man 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. Dadurch ist gewährleistet, dass bei jedem Start eines Auftrags der Baustein im Initialzustand ist und die Funktion unabhängig von vorherigen Aufträgen ausgeführt wird.

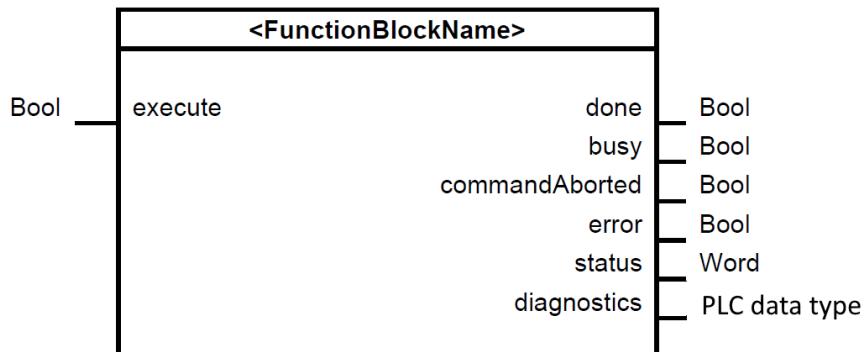
Wenn der Baustein nach PLCopen implementiert wird und den Eingangsparameter `execute` benutzt, müssen mindestens die Ausgangsparameter `busy`, `done` und `error` verwendet werden.

Tabelle 9-2

Bezeichner	Datentyp	Bedeutung
Eingangsparameter		
execute	Bool	<p>execute ohne continuousUpdate: Alle Parameter werden mit einer steigenden Flanke am execute-Eingang übernommen und die Funktionalität wird gestartet. Ist eine Änderung der Parameter notwendig, muss der execute-Eingang neu getriggert werden.</p> <p>execute mit continuousUpdate: Alle Parameter werden mit einer steigenden Flanke am execute-Eingang übernommen. Diese können solange angepasst werden, wie der Eingang <code>continuousUpdate</code> gesetzt ist.</p>
Ausgangsparameter		
done	Bool	<p>Die Ausgänge <code>done</code>, <code>busy</code>, <code>commandAborted</code> und <code>error</code> sind gegenseitig exklusiv. Der Ausgang <code>done</code> wird gesetzt, wenn das Kommando erfolgreich abgearbeitet wurde.</p>
busy	Bool	<p>Die Ausgänge <code>done</code>, <code>busy</code>, <code>commandAborted</code> und <code>error</code> sind gegenseitig exklusiv. Der FB ist noch nicht mit der Bearbeitung des Kommandos fertig und somit können neue Ausgangswerte erwartet werden. <code>busy</code> wird bei steigender Flanke von <code>execute</code> gesetzt und rückgesetzt, wenn einer der Ausgänge <code>done</code>, <code>commandAborted</code> oder <code>error</code> gesetzt wird.</p>

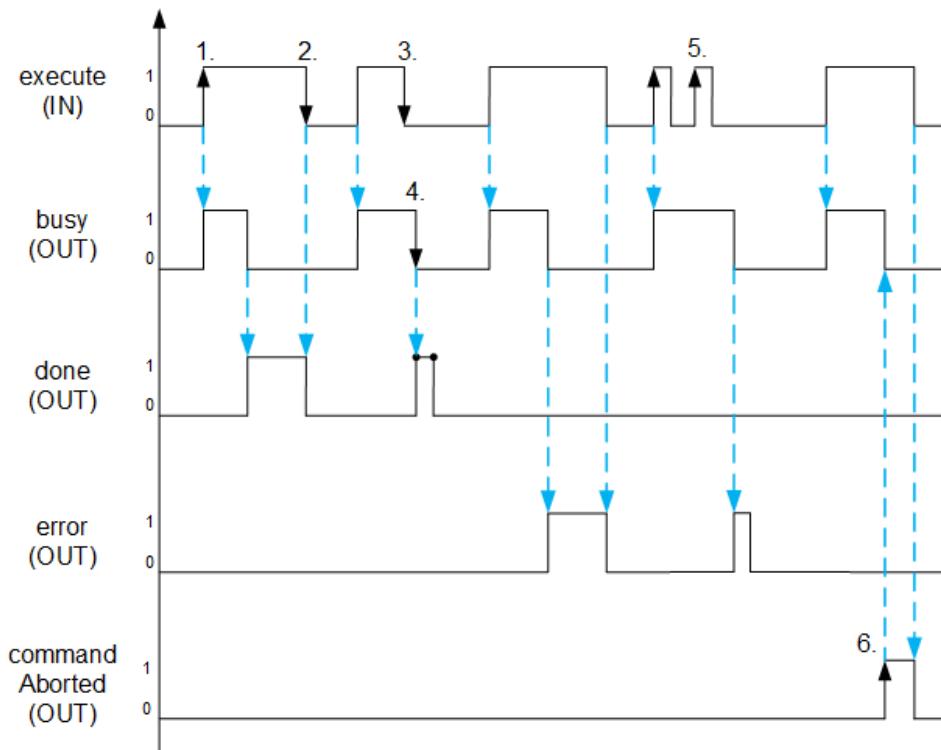
Bezeichner	Datentyp	Bedeutung
error	Bool	Die Ausgänge <code>done</code> , <code>busy</code> , <code>commandAborted</code> und <code>error</code> sind gegenseitig exklusiv . Eine steigende Flanke des Ausgangs signalisiert, dass ein Fehler während der Abarbeitung des FB aufgetreten ist.
commandAborted	Bool	Die Ausgänge <code>done</code> , <code>busy</code> , <code>commandAborted</code> und <code>error</code> sind gegenseitig exklusiv . Optionaler Ausgang: zeigt an, 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 <code>commandAborted</code> gesetzt, da dieser Auftrag durch das Halt Kommando abgebrochen wurde.
status	Bool	Optionaler Ausgang: Fehler- oder Statusinformation des Bausteins. Dieser Parameter wird in Anlehnung an die vorhandenen Systembausteine benannt. (<code>errorID</code> 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 "DA015 Empfehlung: Unterlagerte Informationen durchreichen" beschrieben.

Abbildung 9-6: Beispiel Baustein Schnittstelle mit execute



Signalablaufdiagramm eines Bausteins mit execute:

Abbildung 9-7



1. Mit einer steigenden Flanke an `execute` wird der Baustein aktiviert, mit `busy` auf `TRUE` wird angezeigt, dass der Baustein aktiviert ist, keine Fehler anstehen und somit die Ausgänge des FB gültig sind.
2. Die Ausgänge `done`, `error` und `commandAborted` werden mit fallender Flanke an `execute` zurückgesetzt.
3. Die Funktionalität des FB wird mit fallender Flanke an `execute` nicht gestoppt.
4. Wenn `execute` bereits `FALSE` ist, dann stehen `done`, `error` und `commandAborted` nur für einen Zyklus an.
5. 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.
6. 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.

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.

DA013 Regel: Status/Fehler per “status”/“error” zurückgeben

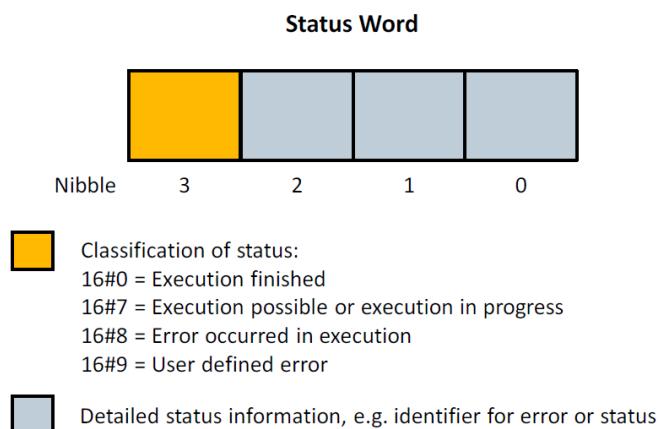
Der Baustein gibt eine eindeutige Auskunft am Ausgangsparameter `status`, die auf den Zustand im Baustein schließen lässt. Die Werte sind als Konstanten in den Lokaldaten der Bausteinschnittstelle zu definieren, um doppelte Belegungen zu verhindern und die Lesbarkeit durch die Symbolik zu verbessern.

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 (“most significant 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: Status codes

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: Error codes

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

Hinweis Es wird empfohlen, Named value-Datentypen (NVT) oder Konstanten innerhalb des Bausteins zu verwenden, um den Status-/Fehlercode zu definieren und sie korrekt zu benennen.
Daher sollte für die Namen ein Präfix verwendet werden, z. B.:

- Status codes: STATUS_, INFO_, WARN_, WARNING_
- Error codes: ERR_, ERROR_

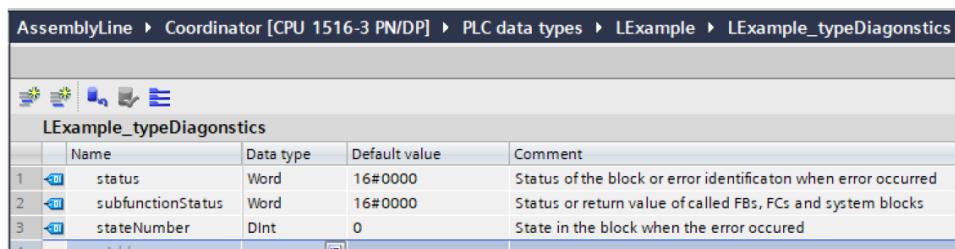
DA015 Empfehlung: Unterlagerte Informationen durchreichen

Werden in einem Baustein weitere Unterfunktionen aufgerufen, welche 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.
Mit Hilfe der Systembasierten Diagnose (ProDiag oder Program_Alarm) können ebenfalls Diagnose Informationen ausgegeben und protokolliert werden.

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	(D)Word	Status von unterlagerten Funktionen
stateNumber	DInt	Die Nummer des Bearbeitungsschritts, in welchem 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 identifier 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 occurred
4	timeStamp	DTL	DTL#1970-01-01-00...	Time stamp of error occurrence
5	additionalValue1	LReal	0.0	Calculated position of axis 1
6	additionalValue2	LReal	0.0	Real position of axis 1
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 abgespeichert.

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.

Hinweis

In den Bausteinvorlagen mit `enable` und `execute` wird die Verwendung einer Diagnosestruktur im Zusammenhang mit einem Zustandsautomaten gezeigt. Die Bausteinvorlagen sind in den Kopiervorlagen in der Bibliothek mit generellen Funktionen (LGF) zu finden:

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

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.

DA019 Empfehlung: Verwenden von Named value-Datentypen

Verwendung von Named value-Datentypen wenn möglich, z. B. für:

1. CASE-Anweisungen, z. B. in Zustandsautomaten.
2. Konfigurationsparametern anstelle von Integer
Beispiel: `direction := nvtDirection#FORWARD / direction := nvtDirection#BACKWARD`
3. Status- und Diagnosedaten

Siehe auch "RU005-Regel: Symbolische Konstanten verwenden"

Begründung: Dies gewährleistet die Lesbarkeit und Rückverfolgbarkeit der Software.

Hinweis

Named value-Datentypen sind nur innerhalb von Software Units verfügbar.

Beispiel:

```
CASE #statMainState OF
    nvtStates#NO_OPERATION:
        REGION NO_OPERATION
            ; // No operational state
        END_REGION NO_OPERATION

    nvtStates#ENABLING:
        REGION ENABLING
            ; // Enabling state
            #statMainState := nvtStates#PROCESSING;
        END_REGION ENABLING

    nvtStates#PROCESSING:
        REGION PROCESSING
            ; // Processing state
            #statMainState := nvtStates#DISABLING;
        END_REGION PROCESSING

    nvtStates#DISABLING:
        REGION DISABLING
            ; // Disabling state
            #statMainState := nvtStates#NO_OPERATION;
        END_REGION DISABLING

    ELSE // Statement section ELSE
        // Error, undefined state reached
        #status := nvtStates#ERR_UNDEF;
END_CASE;
```

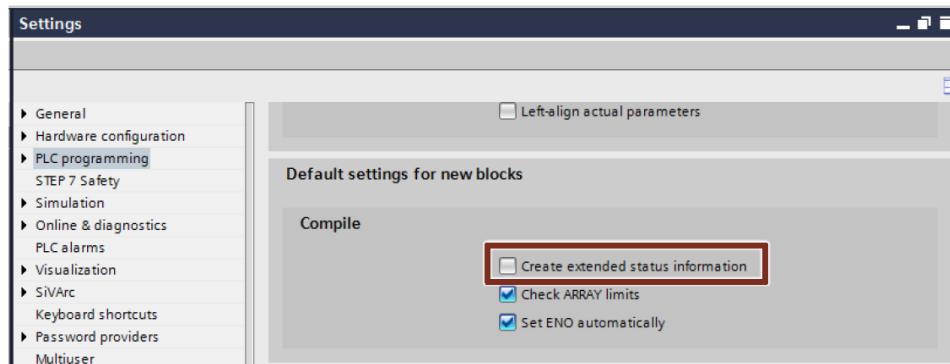
10 Performanz

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 Aktual Parameter übergeben, wofür kein Kopievorgang ausgeführt wird.

Hinweis

Dadurch können die originalen Daten modifiziert werden.

Hinweis

Wenn beim Baustainauftrag 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 als Formalparameter 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 sollten 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-/Indexvariablen als “DInt” deklarieren

Für Lauf- und Indexvariablen, 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 seltener 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 Arbeitsspeicherbedarf. Er ist umso größer, je häufiger die Anweisung aufgerufen wird und je länger die symbolischen Bezeichner sind.

PE014 Empfehlung: Laufzeitintensive Anweisungen vermeiden

Die Verwendung der laufzeitintensiven Befehle ist auf ein Minimum zu reduzieren, da sich die Verwendung dieser Systemfunktionen negativ auf die Gesamtdauer des Programms auswirken kann.

Laufzeitintensive Anweisungen sind Anweisungen, die

1. große Datenmengen verarbeiten, z. B.
 - "Serialize" / "Deserialize"
2. auf die Speicherkarte zugreifen, z. B.
 - "CREATE_DB" / "WRITE_DB" / "READ_DB" / "ATTR_DB" / "DELETE_DB"
 - "FileRead" / "FileWrite" / "FileDelete"
 - "RecipeExport" / "RecipeImport"
 - "DataLogCreate" / "DataLogOpen" / "DataLogClear" / "DataLogWrite" / "DataLogClose" / "DataLogDelete" / "DataLogNewFile"
 - Flash-Speicher wie die SIMATIC Memory Card sind vergleichsweise langsam, der Zugriff erfolgt typischerweise asynchron und kann mehrere Zyklen dauern, da größere Datenmengen übertragen werden.
3. Lesen / Auflösen von Symboldaten aus dem Ladespeicher, z. B.
 - "GetSymbolName" / "GetSymbolPath" / "GetInstanceName" / "GetInstancePath"
 - "ResolveSymbols" / "MoveToResolvedSymbol" / "MoveFromResolvedSymbol" / "MoveResolvedSymbolsToBuffer" / "MoveResolvedSymbolsFromBuffer"

Begründung: Die Nutzung dieser Systemfunktionen kann sich negativ auf die gesamte Programmduer auswirken.

Mögliche Lösungen:

- Sie nicht zyklisch aufrufen
- Sie nur bei Bedarf aufrufen
- Bei regelmäßigm Bedarf jeden n-ten Zyklus aufrufen
- Bei einer großen Anzahl von Aufrufen, diese in mehrere Gruppen aufteilen und in verschiedenen Zyklen aufrufen
- Wenn sie nur einmal benötigt werden, rufen Sie diese im Start-OB auf.

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.

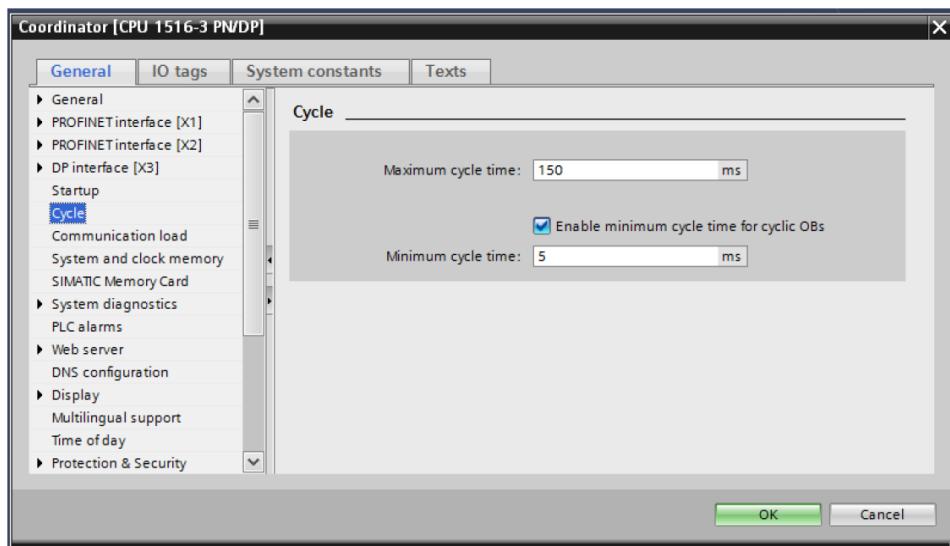
Begründung: GRAPH und CFC als Programmiersprachen erzeugen zusätzlichen Maschinencode und Diagnoseinformationen, die zusätzliche Laufzeit benötigen. GRAPH eignet sich für die Erstellung sequentieller Maschinenabläufe, während CFC für die signalflussorientierte Ablaufprogrammierung verwendet wird.

PE016 Empfehlung: Einstellung der Mindestzykluszeit prüfen

Bei zeitkritischen Applikationen ohne hohes Kommunikationsaufkommen kann die "Minimum cycle time" abgeschaltet werden, wenn eine schnelle Reaktionszeit erforderlich ist.

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



PE017 Empfehlung: Mehrfachzugriffe auf IO / TO-Variablen vermeiden

Bei mehrfachem Variablenzugriff auf I/O Bereiche oder Technologieobjekte wird empfohlen, die Werte einmalig auf interne temporäre oder statische Variablen umzukopieren und mit diesen zu arbeiten.

Darüberhinaus ist zu beachten, dass:

- Daten nicht häufiger ausgelesen werden, als ihre eigentliche Aktualisierungsrate (z.B. im Motion Takt -> Pre/Post Servo),
- Daten nur ausgelesen werden, wenn sie für die Programmabarbeitung benötigt werden.
- statische Daten nur auf gezielte Anforderung auslesen werden (z.B. Konfigurationswerte).

Begründung: Der Zugriff auf I/O Datenbereiche und insbesonders auf Technologieobjekte ist deutlich langsamer als der Zugriff auf optimierte Daten. Deshalb ist bei Mehrfachzugriffen das einmalige Umkopieren und Arbeiten mit der Kopie performanter.

Darüberhinaus wird durch das Arbeiten mit der Kopie ein Prozessabbild geschaffen und der Baustein arbeitet mit konsistenten Daten. Bei mehrmaligem Zugriff im Baustein könnte es vorkommen, dass die Daten eines Technologieobjekts wie z.B. die Istposition aus verschiedenen Interpolator-Takten stammen und die Applikation nicht das erwartete Ergebnis liefert.

11 CheatSheet

Block Interface	<code>enable</code>	<code>In</code>	<code>Out</code>	<code>In/Out</code>	<code>Stat</code>	<code>tempIndex</code>	<code>Temp</code>	<code>Const</code>
			<code>done</code>	<code>conveyorAxes</code>				<code>MAX_VELOCITY</code>
				<code>instTimer</code>				
				<code>extInterface</code>				
Prefix	--	--	--	-- (default)	"stat"	"temp"	--	
Casing	camelCasing	camelCasing	camelCasing	"inst" (parameter-instance)	"inst" (multi-instance)	--	--	
				"ext" (external accessible interf.)	"ext" (external accessible interf.)			
				camelCasing	camelCasing			<code>UPPER_CASING</code>

Tag table	<code>PLC tag</code>	<code>User constant</code>	Programming style guide for SIMATIC S7 in TIA Portal S7-1200 / S7-1500					
Prefix	<code>lightBarrierLeft</code>	<code>MAX_BELTS</code>						
Casing	camelCasing	UPPER_CASING						

Object	Prefix	Casing	Usual abbreviations					
Project		UpperCamelCase	Min / Max	Minimum / Maximum				
Library	<code>lConn</code>	UpperCamelCase	Act	Actual, Current				
Organization block	<code>Main</code>	UpperCamelCase	Next / Prev	Next / Previous value				
Function block	<code>heatTank</code>	UpperCamelCase	Avg	Average				
Global data block	<code>calcTime</code>	UpperCamelCase	Sum	Total sum				
Single instance data block	<code>machineData</code>	UpperCamelCase	Diff	Difference				
Technology Object	<code>instHeater</code>	UpperCamelCase	Cnt	Count				
PLC tag	<code>heatingAxis</code>	UpperCamelCase	Len	Length				
Sensors	<code>temp</code>	UpperCamelCase	Pcs	Position				
MachineState	<code>ris</code>	UpperCamelCase	Ris / Fal	Rising / falling edge				
ConveyorSpeed	<code>trace</code>	UpperCamelCase	Old	Old value				
Temperature	<code>measure</code>	UpperCamelCase	Sim	Simulated				
ConveyorAlarms	<code>plcTextList</code>	UpperCamelCase	Dir	Direction				
SoftwareUnit	<code>magazine</code>	UpperCamelCase	Err / Warn	Error / Warning				
PLC data type	<code>typeDiagnostics</code>	lowerCamelCasing	Cmd	Command				
Element in a PLC datatype	<code>stateNumber</code>	lowerCamelCasing	Addr	Address				
Named value data type	<code>nvStatus</code>	lowerCamelCasing						
Element in NVT	<code>status_no_Error</code>	UPPER_CASING						
User irrelevant PLC Object	<code>lib</code>	lower-/UpperCamelCasing						

12 Anhang

12.1 Service und Support

SiePortal

Die integrierte Plattform für Produktauswahl, Einkauf und Support – und Verbindung von Industry Mall und Online Support. Die neue Startseite ersetzt die bisherigen Startseiten der Industry Mall sowie des Online Support Portals (SIOS) und fasst diese zusammen.

- Produkte & Services
Unter Produkte & Services finden Sie alle unsere Angebote, die bisher im Mall Katalog verfügbar waren.
- Support
Im Bereich Support finden Sie alle Informationen, die für die Lösung technischer Probleme mit unseren Produkten hilfreich sind.
- mySiePortal
mySiePortal ist Ihr persönlicher Bereich, der Funktionen, wie z.B. die Warenkorbverwaltung oder die Bestellübersicht anzeigt. Den vollen Funktionsumfang sehen Sie hier erst nach erfolgtem Login.

Das SiePortal rufen Sie über diese Adresse auf: sieportal.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:

support.industry.siemens.com/cs/my/src

SITRAIN – Digital Industry Academy

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:

siemens.de/sitrain

Industry Online Support App

Mit der App “Siemens Industry Online Support” erhalten Sie auch unterwegs die optimale Unterstützung. Die App ist für iOS und Android verfügbar:



12.2 Industry Mall



Die Siemens Industry Mall ist die Plattform, auf der das gesamte Produktpotfolio von Siemens Industry zugänglich ist. Von der Auswahl der Produkte über die Bestellung und die Lieferverfolgung ermöglicht die Industry Mall die komplette Einkaufsabwicklung – direkt und unabhängig von Zeit und Ort:

mall.industry.siemens.com

12.3 Links und Literatur

Table 12-1: Links und Literatur

No.	Topic
\1\	Siemens Industry Online Support https://support.industry.siemens.com
\2\	Link auf die Beitragsseite des Anwendungsbeispiels https://support.industry.siemens.com/cs/ww/de/view/81318674
\3\	Programmierstyleguide für S7-1200/S7-1500 https://support.industry.siemens.com/cs/ww/de/view/109478084
\4\	Programmierleitfaden für S7-1200/S7-1500 https://support.industry.siemens.com/cs/ww/de/view/90885040
\5\	Programmierleitfaden Safety für S7-1200/S7-1500 https://support.industry.siemens.com/cs/ww/de/view/109750255
\5\	TIA Portal Settings für Style Guide (Login required) 81318674_TIAPortalSettingsForStyleGuide.zip
\5\	Test Suite Advanced: Beispiel für die Überprüfung von TIA Portal Projekten https://support.industry.siemens.com/cs/ww/de/view/109779806
\7\	Projekt Check für TIA Portal: Prüfen gegen Programmierstyleguides https://support.industry.siemens.com/cs/ww/de/view/109741418
\7\	SIMATIC Project Insight https://support.industry.siemens.com/cs/ww/de/view/109818320
\6\	SIMATIC S7-1200/ S7-1500 Vergleichsliste für Programmiersprachen https://support.industry.siemens.com/cs/ww/de/view/86630375
\7\	Standardisierungsleitfaden https://support.industry.siemens.com/cs/ww/de/view/109756737
\8\	Multiuser Engineering mit TIA Project Server https://support.industry.siemens.com/cs/ww/de/view/109740141
\9\	Leitfaden zur Bibliothekshandhabung https://support.industry.siemens.com/cs/ww/de/view/109747503
\10\	Anwenderdefinierte Dokumentation bereitstellen https://support.industry.siemens.com/cs/ww/de/view/109773506/119453612171
\11\	TIA Portal Add-In Code2Docu zur Generierung der Dokumentation https://support.industry.siemens.com/cs/ww/de/view/109809007
\12\	Bibliothek mit generellen Funktionen (LGF) für TIA Portal und SIMATIC S7-1200/ S7-1500 https://support.industry.siemens.com/cs/ww/de/view/109479728

12.4 Changelog / History

Table 12-2: Changelog

Version	Date	Changes
V2.1	04/2025	<p>General</p> <ul style="list-style-type: none"> - Wording alignments - Add programming languages SFC, CFC and CEM to style guide - Add Named value data type (NVT) to style guide - Update CheatSheet <p>NEW RULES</p> <p>SE007:</p> <ul style="list-style-type: none"> - Use value ranges for Real/LReal <p>NEW RECOMMENDATIONS</p> <p>DA019:</p> <ul style="list-style-type: none"> - Use of Named value data type <p>PE017:</p> <ul style="list-style-type: none"> - Avoid multiple Access for IO / TO variables <p>UPDATED RULES/RECOMMENDATIONS</p> <p>ES*:</p> <ul style="list-style-type: none"> - Add where the rules effects TIA Portal or the Project <p>ES007:</p> <ul style="list-style-type: none"> - Adjust wording to be more precise and clearer about the rule <p>GL003:</p> <ul style="list-style-type: none"> - Change from "all project languages" to "all active project languages" <p>NF001 / NF002:</p> <ul style="list-style-type: none"> - Wording adjustments to be more general <p>NF001 / NF005:</p> <ul style="list-style-type: none"> - Add namespace to rule description <p>NF003:</p> <ul style="list-style-type: none"> - Document developer information - remove / set to optional for redundant infos <p>NF004:</p> <ul style="list-style-type: none"> - Extend example table with... - Global constants table for status/error/... - Named value data type <p>NF005/NF006/NF008:</p> <ul style="list-style-type: none"> - Rename rules to more understandable names - PascalCasing → UpperCamelCase - camelCasing → lowerCamelCase - CAPITALS → UPPER_CASING - Extend texts with multiple names for the different namings <p>NF006:</p> <ul style="list-style-type: none"> - Add Named value data type <p>NF007:</p> <ul style="list-style-type: none"> - Add prefix <code>ext</code> for static interface variables - Add prefix <code>nvt</code> for Named value data types - Add prefix <code>unpub</code> for unpublished / internal PLC objects - Derived data blocks from PLC data type do not get a prefix <code>Inst</code> - PLC data type prefix is just needed for its declaration - Numbering the example table - Extend with notes for <ul style="list-style-type: none"> a. Formal parameters b. Internal static variables c. Interface statics (<code>ext</code>) d. Named value data type (<code>nvt</code>)

Version	Date	Changes
		<p>e. Unpublished / internal PLC objects (unpub)</p> <p>NF008:</p> <ul style="list-style-type: none"> - Add Named value data type members <p>NF010:</p> <ul style="list-style-type: none"> - Adjust wording to be more precise and clearer about the rule - “overall length of a single identifier” <p>NF011:</p> <ul style="list-style-type: none"> - Add Lim for Lmit to abbreviation table <p>NF014:</p> <ul style="list-style-type: none"> - Change from “Use of line comment // only” to “Preferable use of line comment //” - Insert justification and rework note box - Rework “Line breaks in partial conditions” - Number the topics and rework the examples <p>RU001:</p> <ul style="list-style-type: none"> - Rename to “Enable support for different target systems” because inserting virtual PLC support - Change from Rule to Recommendation because it potentially reduces the level of know-how protection if used <p>RU002:</p> <ul style="list-style-type: none"> - Change wording to precise the meaning for versioning with libraries - Improve list and table for “Version numbers and their use” - Add Named value data type <p>RU003:</p> <ul style="list-style-type: none"> - Add Named value data type <p>RU004:</p> <ul style="list-style-type: none"> - Add PLC data types as they belong as well to the rule - Split topics into ordered and unordered list <p>RU005:</p> <ul style="list-style-type: none"> - Remove Local and add Named value data type for symbolic constants - Add example for NVT <p>RU006:</p> <ul style="list-style-type: none"> - Extend examples with Pointer - Extend with system constants <p>AL002:</p> <ul style="list-style-type: none"> - Split topics into ordered and unordered list - Insert references to other rules <p>AL003:</p> <ul style="list-style-type: none"> - Adjust wording to be more precise and clearer about the rule - Applies for “Array of unknown size” <p>AL004:</p> <ul style="list-style-type: none"> - Split topics into numbered list <p>SE001:</p> <ul style="list-style-type: none"> - Adjust wording to be more precise and clearer about the rule <p>SE004:</p> <ul style="list-style-type: none"> - Adjust wording to be more precise and clearer about the rule - Insert table with maximum permitted settings and description - Insert differentiation for Interfaces within static area (e.g. PLC external systems) - Insert reference to dependent rules <p>SE006:</p> <ul style="list-style-type: none"> - Adjust wording to be more precise and clearer about the rule - Insert examples for Error OBs and Error handling <p>DA002:</p> <ul style="list-style-type: none"> - Add CEM and CFC as programming language to recommendation <p>DA003:</p>

Version	Date	Changes
		<ul style="list-style-type: none"> - Remove OB from auto numbering list - OB number need to be adjusted for call sequence - First part to numbered list to separate rules and user actions - Extend description regarding IEC Check setting - Insert Reference to dependent rules <p>DA004:</p> <ul style="list-style-type: none"> - Add <code>struct</code> for the use in PLC data types - Add note for initialization with empty PLC data types <p>DA005:</p> <ul style="list-style-type: none"> - Adjust wording to be more precise and clearer about the rule - Insert differentiation for interfaces within static area (e.g. PLC external systems) - Insert note and example for multi-instance access <p>DA006:</p> <ul style="list-style-type: none"> - Insert differentiation for Interfaces within static area (e.g. PLC external systems) <p>DA008:</p> <ul style="list-style-type: none"> - Split topics into ordered list <p>DA011/DA012:</p> <ul style="list-style-type: none"> - Refine description, timing diagrams and description listing - Insert description for subordinated commands <p>DA014:</p> <ul style="list-style-type: none"> - Extend Error and Status ranges with recommendation for constant prefixes <p>DA015:</p> <ul style="list-style-type: none"> - Extend note for system-based diagnostics (<code>ProDiag</code> or <code>Program_Alarm</code>) <p>PE014:</p> <ul style="list-style-type: none"> - Rework to list - Extend examples - Extend with resolve symbolic names <p>PE015:</p> <ul style="list-style-type: none"> - Add CFC to rule
V2.0	05/2020	<ul style="list-style-type: none"> - Categorize according to Workflow - Extend topics for Performance and Design-/ Architecture - Extend Programming guidelines - Amendment of Justifications - Review of Cheat sheet
V1.2	10/2016	Adaptations and Corrections