SIEMENS

Programmierstyleguide • 06/2015

Programmierstyleguide für S7-1200/S7-1500

TIA Portal

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

Gewährleistung und Haftung

Hinweis

Die Programmierrichtlinien sind unverbindlich und erheben keinen Anspruch auf Vollständigkeit hinsichtlich Konfiguration und Ausstattung sowie jeglicher Eventualitäten. Die Programmierrichtlinien stellen keine kundenspezifischen Lösungen dar, sondern sollen lediglich Hilfestellung bieten bei typischen Aufgabenstellungen. Sie sind für den sachgemäßen Betrieb der beschriebenen Produkte selbst verantwortlich. Diese Programmierrichtlinien entheben Sie nicht der Verpflichtung zu sicherem Umgang bei Anwendung, Installation, Betrieb und Wartung. Durch Nutzung dieser Programmierrichtlinien erkennen Sie an, dass wir über die beschriebene Haftungsregelung hinaus nicht für etwaige Schäden haftbar gemacht werden können. Wir behalten uns das Recht vor, Änderungen an diesen Programmierrichtlinien jederzeit ohne Ankündigung durchzuführen. Bei Abweichungen zwischen den Vorschlägen in diesem Programmierrichtlinien und anderen Siemens Publikationen, wie z.B. Katalogen, hat der Inhalt der anderen Dokumentation Vorrang.

Für die in diesem Dokument enthaltenen Informationen übernehmen wir keine Gewähr.

Unsere Haftung, gleich aus welchem Rechtsgrund, für durch die Verwendung der in diesem Programmierstyleguide beschriebenen Beispielen, Hinweisen, Programmen, Projektierungs- und Leistungsdaten usw. verursachte Schäden ist ausgeschlossen, soweit nicht z.B. nach dem Produkthaftungsgesetz in Fällen des Vorsatzes, der groben Fahrlässigkeit, wegen der Verletzung des Lebens, des Körpers oder der Gesundheit, wegen einer Übernahme der Garantie für die Beschaffenheit einer Sache, wegen des arglistigen Verschweigens eines Mangels oder wegen Verletzung wesentlicher Vertragspflichten zwingend gehaftet wird. Der Schadensersatz wegen Verletzung wesentlicher Vertragspflichten ist jedoch auf den vertragstypischen, vorhersehbaren Schaden begrenzt, soweit nicht Vorsatz oder grobe Fahrlässigkeit vorliegt oder wegen der Verletzung des Lebens, des Körpers oder der Gesundheit zwingend gehaftet wird. Eine Änderung der Beweislast zu Ihrem Nachteil ist hiermit nicht verbunden.

Security-hinweise

Siemens bietet Produkte und Lösungen mit Industrial Security-Funktionen an, die den sicheren Betrieb von Anlagen, Lösungen, Maschinen, Geräten und/oder Netzwerken unterstützen. Sie sind wichtige Komponenten in einem ganzheitlichen Industrial Security-Konzept. Die Produkte und Lösungen von Siemens werden unter diesem Gesichtspunkt ständig weiterentwickelt. Siemens empfiehlt, sich unbedingt regelmäßig über Produkt-Updates zu informieren.

Für den sicheren Betrieb von Produkten und Lösungen von Siemens ist es erforderlich, geeignete Schutzmaßnahmen (z. B. Zellenschutzkonzept) zu ergreifen und jede Komponente in ein ganzheitliches Industrial Security-Konzept zu integrieren, das dem aktuellen Stand der Technik entspricht. Dabei sind auch eingesetzte Produkte von anderen Herstellern zu berücksichtigen. Weitergehende Informationen über Industrial Security finden Sie unter http://www.siemens.com/industrialsecurity.

Um stets über Produkt-Updates informiert zu sein, melden Sie sich für unseren produktspezifischen Newsletter an. Weitere Informationen hierzu finden Sie unter http://support.industry.siemens.com/.

Inhaltsverzeichnis

1 Einleitung

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

Vorteile eines einheitlichen Programmierstils

Falls mehrere Programmierer am selben Programm arbeiten, empfiehlt es sich, dass ein gemeinsamer und abgestimmter Programmierstil eingehalten wird. Somit ergeben sich folgende Vorteile:

- Einheitlicher durchgängiger Stil
- Leicht lesbar und verständlich
- Einfache Wartung und Wiederverwendbarkeit
- Einfache und schnelle Fehlererkennung und –korrektur
- Effektive Arbeit am selben Projekt mit mehreren Programmieren

Ziel des Programmierstyleguides

Hinweis Die hier beschriebenen Programmierrichtlinien sollen Ihnen lediglich als Vorschlag dienen, einen einheitlichen Programmierstil einzuhalten. Es liegt bei Ihnen, welche Regeln und Empfehlungen Sie als sinnvoll erachten und welche Sie verwenden oder nicht.
Beachten Sie aber, dass die hier beschriebenen Regeln und Empfehlungen auf einander abgestimmt sind und sich gegenseitig nicht behindern.

Die hier beschriebenen Programmierrichtlinien helfen ihnen, einen einheitlichen Programmcode zu erstellen, der besser gewartet und wiederverwendet werden kann. Somit können möglichst frühzeitig Fehler erkannt (z.B. durch Compiler) bzw. vermieden werden.

Der Quellcode muss folgende Eigenschaften haben:

- Einheitlicher durchgängiger Stil
- Leicht lesbar und verständlich

Für die Wartbarkeit und Übersichtlichkeit des Quellcodes ist es zunächst erforderlich, sich an eine gewisse äußere Form zu halten. Optische Effekte - z.B. eine einheitliche Anzahl von Leerzeichen vor dem Komma - tragen allerdings nur unwesentlich zur Qualität der Software bei. Viel wichtiger ist es bspw. 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 Programmcodes bei der Diagnose von Programmfehlern, z.B. Wiederverwendung einer Temp-Variable über einen Zyklus hinaus.
  Ziel: Der Code weist frühzeitig auf Probleme hin.
- Vereinheitlichung von Standardapplikationen und –bibliotheken
  Ziel: Die Einarbeitung soll einfach sein und die Wiederverwendbarkeit von Programmcode erhöht werden.

- Einfache Wartung und Vereinfachung der Weiterentwicklung
  Ziel: Änderungen von Programmcode in den einzelnen Modulen, welche Funktionen (FCs), Funktionsbausteine (FBs), Datenbausteine (DBs), Organisationsbausteine (OBs) in Bibliotheken oder im Projekt umfassen können, sollen minimale Auswirkungen auf das Gesamtprogramm/ Gesamtbibliothek haben. Änderungen von Programmcode in den einzelnen Modulen sollen von verschiedenen Programmierern durchführbar sein.

Gültigkeit

Dieses Dokument gilt für (Kunden-)Anwendungsbeispiele sowie Bibliotheken, die in den Programmiersprachen der IEC 1131-3 (DIN EN 61131-3) Structured Text (ST), Kontaktplan (KOP), Funktionsplan (FUP) und Anweisungsliste (AWL) erstellt werden.

Abgrenzung

Dieses Dokument enthält keine Beschreibung von:

- STEP 7 Programmierung
- Inbetriebnahme von SIMATIC Steuerungen

Grundlegende Kenntnisse über diese Themen werden voraus gesetzt.

2 Begriffsklärung

Regeln / Empfehlungen

Vorgaben werden unterteilt in Empfehlungen und Regeln.

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 man eine Empfehlung nicht befolgt. Dies kann aus Gründen der Effizienz sein, aber auch weil der Code anders besser lesbar ist.

Performance

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

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

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

Abkürzungen

Folgende Abkürzungen werden innerhalb dieses Textes verwendet:

Tabelle 2-1: Abkürzungen Bausteine

Abk.	Typ
OB	Organisationsbaustein
FB	Funktionsbaustein
FC	Funktion
DB	Datenbaustein
TO	Technologieobjekt
SFB	Systemfunktionsbaustein
SFC	Systemfunktion

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.

Abbildung 2-1: Begriffe bei Variablen und Parametern

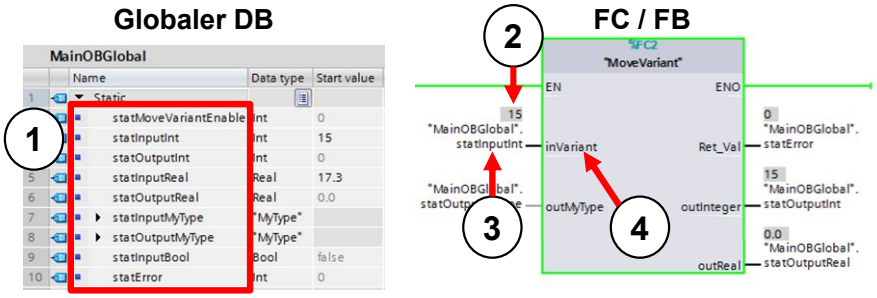


Tabelle 2-2: Begrffe bei Variablen und Parametern

	Begriff	Erklärung
1.	Variable	Variablen werden durch einen Namen/Identifier bezeichnet und belegen eine Adresse im Speicher in der Steuerung. Variablen werden immer mit einem bestimmten Datentyp (Bool, Integer, usw.) definiert: • PLC-Variablen • einzelne Variablen in Datenbausteinen • komplette Datenbausteine
2.	Variablenwert	Variablenwerte sind Werte, die in einer Variablen gespeichert sind (z.B. 15 als Wert einer Integer-Variablen)
3.	Aktualparameter	Aktualparameter sind Variablen, die an den Schnittstellen von Anweisungen, Funktionen und Funktionsbausteinen verschaltet sind.
4.	Formalparameter (Übergabeparameter, Bausteinparameter)	Formalparameter sind die Schnittstellenparameter von Anweisungen, Funktionen und Funktionsbausteinen (Input, Output, InOut und Ret_Val).

3 Allgemeine Vorgaben

Generell sollten Sie darauf achten, dass verwendete Namen bezogen auf Funktionalität und verwendeter Schnittstellenart immer eindeutig sind.

D.h. wenn derselbe Name verwendet wird, sollte auch die dahinter stehende Funktionalität dieselbe sein.

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

Hinweis Programmierleitfaden für S7-1200/1500

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

3.1 Vorgaben und Kundenanforderung

Regel: Dokumentation bei Regelverletzung

Jedes Mal wenn eine Regel verletzt wird, ist dies unbedingt an der entsprechenden Stelle im Programmcode zu dokumentieren.

Für Kundenprojekte geht der Wunsch des Endkunden vor. Wünscht der Kunde Änderungen oder Abweichungen von dieser Programmierrichtlinie, so hat dies Vorrang. Die vom Kunden definierten Regeln sind in geeigneter Form im Quelltext zu dokumentieren.

3.2 Einstellungen im TIA Portal

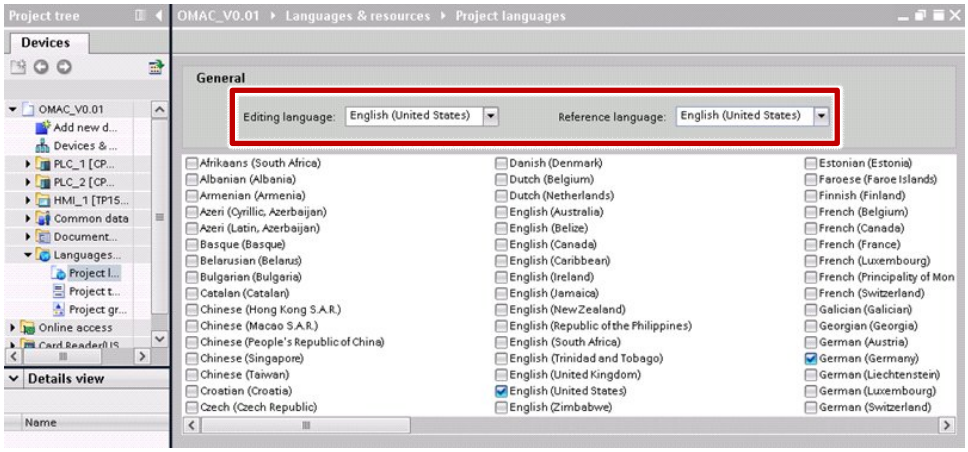
Regel: Einheitliche Sprache

Die Sprache muss sowohl in der PLC-Programmierung als auch im HMI immer konsistent sein. Das heißt, dass keinesfalls Sprachen in einem Projekt gemischt werden dürfen (z.B. Englisch als Editiersprache und deutsche Kommentare in Bausteinen oder französische Texte im englischen Sprachbereich des HMIs).

Regel: Editier- und Referenzsprache: English (United States)

Wenn nicht ausdrücklich vom Kunden anders gewünscht, ist die Sprache „English (United States)“ als Editier- und Referenzsprache zu verwenden. Somit werden auch der Programmcode und alle Kommentare in Englisch verfasst.

Abbildung 3-1: Einstellung Editier- und Referenzsprache



Empfehlung: Oberflächensprache: English (United States)

Die Oberflächensprache im TIA Portal sollte auf Englisch eingestellt werden. Dadurch werden alle neu angelegten Projekte automatisch in Editier- und Referenzsprache *English (United States)* angelegt. Ist dagegen als Oberflächensprache *Deutsch* gewählt, werden alle Projekte mit Editier- und Referenzsprache Deutsch angelegt.

Regel: Mnemonic: International

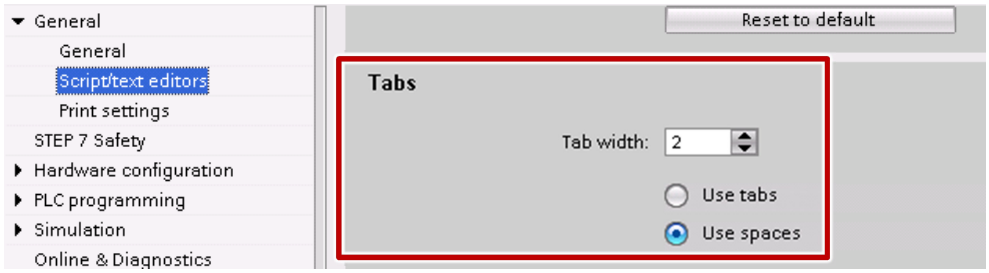
Die Mnemonik (Spracheinstellung für Programmiersprachen) muss auf *International* eingestellt werden.

Abbildung 3-2: Spracheinstellung und Mnemonik TIA Portal

Regel: Tabulatorzeichen: 2 Leerzeichen

Tabulatorzeichen sind in den textuellen Editoren nicht zugelassen. Einrückungen sind mit zwei Leerzeichen vorzunehmen. Die entsprechende Einstellung muss im TIA Portal vorgenommen werden.

Abbildung 3-3: TIA Portal Einstellung Tabulatoren



3.3 Bezeichner

3.3.1 Formatierung

Regel: Englische Bezeichner

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.

Regel: Eindeutige Bezeichner

Es dürfen nicht mehrere, namensgleiche Bezeichner verwendet werden, die sich nur bezüglich Groß- und Kleinschreibung unterscheiden. Die einmal gewählte Schreibweise eines Bezeichners wird in allen Bausteinen und Quellen beibehalten.

Regel: Bezeichner in camelCasing Schreibweise

Ist keine andere Regel für die Schreibweise eines Bezeichners im Programmierstyleguide vermerkt, wird der betreffende Bezeichner im camelCasing geschrieben.

Folgende Regeln gelten für das camelCasing:

- Anfangsbuchstabe wird klein geschrieben.
- Es werden keine Trennzeichen (wie Binde- oder Unterstriche) verwendet.
- Besteht ein Bezeichner aus mehreren Worten, wird der Anfangsbuchstabe der einzelnen Worte groß geschrieben.

Empfehlung: Bezeichnerlänge: max. 24 Zeichen

Der Bezeichner von Variablen, Konstanten oder Bausteinen sollte 24 Zeichen nicht überschreiten.

Regel: Keine Sonderzeichen

Es werden keine sprachspezifischen Sonderzeichen, wie z.B. ä, ö, ü, à, usw. und keine Leerzeichen verwendet.

Sonderzeichen sind zwischen Präfix und Bezeichner nicht erlaubt.

Empfehlung: Keine Trennzeichen

Trennzeichen (Unterstrich) sollten in Bezeichnern nicht verwendet werden, um die Länge des Bezeichners kurz zu halten.

Beispiel

temporäre Variable: tempMaxLength

Regel: Sinnvolle Bezeichner

Für Bezeichner, die aus mehreren Worten bestehen, ist die Reihenfolge der Worte wie die des gesprochenen Wortes zu wählen.

3.3.2 Abkürzungen

Empfehlung: Erlaubte Abkürzungen

Um Zeichen bei einem Variablennamen zu sparen und die Lesbarkeit des Programms zu erhöhen, sollten die vereinheitlichten Abkürzungen in Tabelle 3-1 verwendet werden.

Tabelle 3-1: Standardisierte Abkürzungen

Abk.	Typ
Min	Minimum
Max	Maximum
Act	Actual, Current
Next	Next
Prev	Previous
Avg	Average
Diff	Difference
Pos	Position
Ris	Rising Edge
Fal	Falling Edge
Sim	Simulated
Sum	Sum
Old	Old value (z.B. für Flankenerkennung)

Empfehlung: Nur eine Abkürzung pro Bezeichner

Mehrere Abkürzungen sollten nicht direkt hintereinander verwendet werden.

4 PLC Programmierung

4.1 Programmbausteine und Quellen

4.1.1 Bausteinnamen und -nummern

Empfehlung: Kurzer, funktionaler Bausteinname

Der Name des Bausteins wird so kurz wie möglich gehalten und enthält einen Hinweis auf dessen Funktionalität.

Regel: Bezeichner von Bausteinen beginnen mit einem Großbuchstaben

Bezeichner von Bausteinen (OBs, FBs, FCs, DBs, Instanz-DBs, TOs, usw.) beginnen mit einem Großbuchstaben, um eine einheitliche Darstellung der Namen im TIA Portal zu erreichen.

Regel: Präfix ‚inst' / ‚Inst' bei Instanzen

Instanzen (Einzelinstanz, Multiinstanzen) erhalten ‚inst' / ‚Inst' als Präfix.

Beispiel

Einzelinstanz: InstPidHeater (groß geschrieben → eigner Baustein)
Multiinstanz: instTimerMotor (klein geschrieben → innerhalb einer Instanz)

Empfehlung: Bausteine mit Autonummerierung

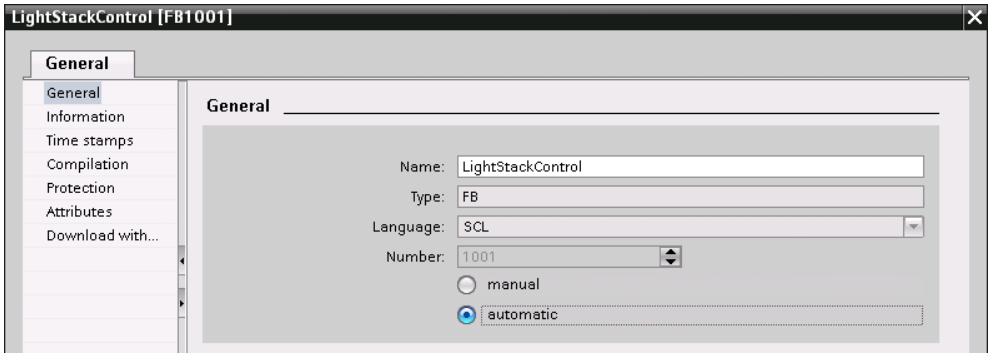
Bausteine werden nur mit aktivierter automatischer Nummernvergabe ausgeliefert.

Folgende Prozedur wird bei der Nummernvergabe empfohlen, wenn eine bestimmte Bausteinnummer an einem Baustein verwendet werden soll:

Tabelle 4-1: Prozedur Nummernvergabe

Nr.	Aktion
1.	Einstellung der Nummernvergabe auf *manuell*
2.	Vergabe der gewünschten Bausteinnummer (z.B. 1001)
3.	Übernehmen der Eigenschaften mit *OK*
4.	Erneutes Öffnen der Bausteineigenschaften
5.	Einstellung der Nummernvergabe auf *automatisch*
6.	Übernehmen der Eigenschaften mit *OK*

Abbildung 4-1: Bausteineigenschaften Nummernvergabe



4.1.2 Formatierung

Empfehlung: Zeilenlänge im Programmeditor: max. 80 Zeichen

Die Zeilenlänge des Quelltextes ist wegen der besseren Lesbarkeit in gedruckter Form auf 80 Zeichen zu begrenzen.

4.1.3 Programmierung

Regel: Keine Quellen verwenden

Um die volle Funktionalität des TIA Portals mit Auto-Vervollständigung nutzen zu können und ein einfaches Debugging zu garantieren, werden nur Bausteine verwendet. Der Umweg über Quellbearbeitung in einem externen Editor und späteren Quellimport darf nicht verwendet werden.

Empfehlung: Vorrangig SCL verwenden

Als Programmiersprache von Bausteinen sollte SCL gewählt werden. SCL bietet die beste Lesbarkeit unter den Programmiersprachen und hat zugleich keine Performance-Nachteile gegenüber anderen SIMATIC PLC-Programmiersprachen.

Soll eine Verschaltung einzelner Bausteine vorgenommen werden, z.B. in einem OB, sollte die Programmiersprache KOP oder FUP gewählt werden. Auch wenn ein Baustein größtenteils aus Binärverknüpfungen besteht, sollte 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.

Regel: Multiinstanzen statt Einzelinstanzen

Vorrangig werden Multiinstanzen verwendet anstatt Einzelinstanzen. Hiermit können in sich geschlossene Funktionen erstellt werden z.B. ein FB mit integriertem Timer für eine Zeitüberwachung.

Empfehlung: DBs nur in Ausnahmefällen im Ladespeicher

Datenbausteine werden immer im Arbeitsspeicher der CPU abgelegt. Die Benutzung des Ladespeichers für die Ablage von Datenbausteinen ist nur in Ausnahmefällen gestattet. Ausnahmen stellen beispielsweise das Abspeichern großer Mengen von Messdaten oder eine Rezepturverwaltung dar.

Regel: Innerhalb eines Bausteins nur lokale Variablen verwenden

Variablen werden nur lokal genutzt. Zugriffe auf globale Daten innerhalb von FCs und FBs sind nicht erlaubt. Dazu zählen:

- Zugriff auf globale DBs und Nutzung von Einzelinstanz-DBs
- Zugriff auf Tags (Variablentabellen)
- Zugriff auf globale Konstanten

Regel: Wichtige Testvariablen von Bausteinen nicht als Temp definieren

Um die Testbarkeit von FCs und FBs zu erleichtern, ist besondere Aufmerksamkeit auf die Beobachtbarkeit von Variablen in den Beobachtungs- und Forcetabellen zu legen.

Hierzu sind interne Variablen, Eingänge und Ausgänge in geeigneter Form (keine Temp-Variablen) zu definieren. Sie müssen ausreichende Auskunft über den

Zustand und Abläufe der Funktionen geben. Dazu gehört beispielsweise der letzte Bearbeitungszustand oder die aktuelle Schrittnummer.

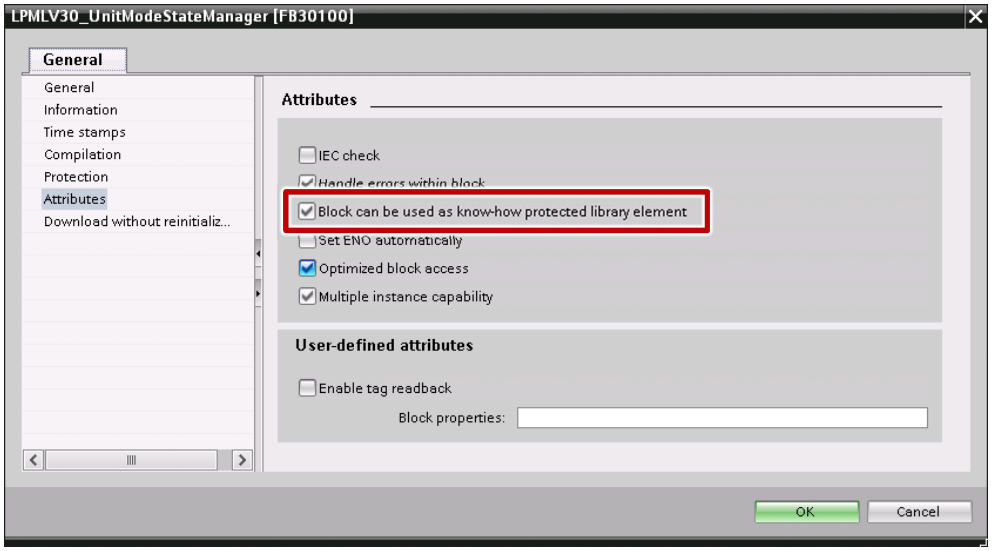
Temp-Variablen können nicht in Force- und Beobachtungstabellen oder HMI beobachtet werden.

Regel: ‚Baustein als Know-how geschütztes Bibliothekselement verwendbar'

Bei einem FB oder FC muss darauf geachtet werden, dass in den Eigenschaften des Bausteins unter den Attributen das Kontrollkästchen „Baustein als Know-how-geschütztes Bibliothekselement verwendbar" vom System (automatisch beim Kompilieren) gesetzt wurde.

Dazu muss der Baustein modular programmiert sein und darf z.B. keine globalen Konstanten oder Variablen verwenden.

Abbildung 4-2: Bausteineigenschaften - Attribute



4.1.4 Kommentare

Es ist zwischen zwei Arten von Kommentaren zu unterscheiden:

- Blockkommentar (beschreibt was eine Funktion, oder ein Code-Abschnitt tut)
- Zeilenkommentar (beschreibt den Code einer einzelnen Zeile)

Empfehlung: Blockkommentare

Ein Blockkommentar ist in einer oder mehreren Zeilen vor den entsprechenden Code-Abschnitt zu setzen.

Empfehlung: Zeilenkommentare

Ein Zeilenkommentar ist, wenn möglich, an das Ende der Code-Zeile zu setzen, ansonsten vor der betreffenden Code-Zeile.

Empfehlung: Nur // Kommentare verwenden

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

Regel: Schablone für Bausteinbeschreibung verwenden

Jeder Baustein wird in einem Beschreibungskopf im Programmcode (SCL) bzw. im Blockkommentar (KOP, FUP) beschrieben. Die Beschreibung enthält folgende Punkte:

- Name der Bibliothek
- getestete PLCs mit Firmware-Version (z.B. S7-1511 V1.6)
- TIA Portal Version bei Erstellung
- Einschränkungen bei der Benutzung (z.B. bestimmte OB-Typen)
- Voraussetzungen (z.B. zusätzliche Hardware)
- Beschreibung der Funktionalität
- Version des Bausteins mit Autor und Datum
- Vorgenommene Änderungen

Schablone für Bausteinkopf

```
//===============================================================================
// Company
// (c)Copyright (year) All Rights Reserved
//-------------------------------------------------------------------------------
// Library:       (that the source is dedicated to)
// Tested with:   (test system with FW version)
// Engineering:   TIA Portal (SW version)
// Restrictions:  (OB types, etc.)
// Requirements:  (hardware, technological package, memory needed, etc.)
// Functionality: (that is implemented in the block)
//-------------------------------------------------------------------------------
// Change log table:
// Version  Date        Expert in charge Changes applied
// 01.00.00 dd.mm.yyyy (Name of expert)  First released version
//===============================================================================
```


4.1.5 Formalparameter: Input, Output und InOut

Regel: Keine Präfixe bei Formalparameter

Es werden keine Präfixe für Formalparameter (*Input, Output* und *InOut)* von FCs/ FBs verwendet. Werden Strukturen für Übergabe- und Ausgangsvariablen verwendet, so besitzen die einzelnen Elemente ebenfalls keine Präfixe.

Regel: Datenaustausch über Bausteinschnittstellen

Werden Daten in mehreren FBs oder FCs benötigt, erfolgt der Datenaustausch über die Bausteinschnittstellen (*Input*-, *Output*- und *InOut*-Schnittstellen).

Ein direkter Zugriff auf S*tatic*-Variablen außerhalb des FBs ist verboten.

Empfehlung: Verwendung von elementaren Datentypen als In, Out oder InOut

Bei elementaren Datentypen (z.B. vom Typ WORD, DWORD, REAL, INT, TIME) sollte der *Input*- oder der *Output*-Schnittstellentyp verwendet werden.

Der *InOut*-Schnittstellentyp wird bei elementaren Datentypen nur dann verwendet, wenn ein Wert sowohl außerhalb als auch innerhalb eines Bausteines schreibend bearbeitet wird.

Empfehlung: Viele Variablen als strukturierte Variablen übergeben

Werden viele Parameter übergeben, sollte versucht werden diese in PLC-Datentypen zu kapseln. Dieser PLC-Datentyp sollte dann als *InOut*-Variable

deklariert werden. Beispiele für solche PLC-Datentypen sind Konfigurationsdaten, Istwerte, Sollwerte, Ausgabe des aktuellen Zustands des Funktionsbausteins etc.

Bei sich oft ändernden Steuer- bzw. Statusvariablen kann es sinnvoll sein, diese für einen einfachen Zugriff in KOP / FUP außerhalb eines solchen PLC-Datentyps direkt als elementare *Input*- bzw. *Output*-Variablen zu deklarieren.

Empfehlung: Strukturierte Variablen als InOut übergeben

Bei strukturierten Variablen (z.B. vom Typ ARRAY, STRUCT, STRING, …) und PLC-Datentypen sollte generell der *InOut*-Schnittstellentyp verwendet werden.

Dadurch wird bei gleicher Optimierungseinstellung der verschalteten Daten und des aufgerufenen Bausteins ein Umkopiervorgang, wie z.B. bei Input-Variablen, in der CPU eingespart. Anstatt des Umkopierens wird direkt mit der Zeigerreferenz zu den Daten gearbeitet. Außerdem wird durch Benutzung einer Referenz Speicherplatz im Ladespeicher gespart.

Hinweis

Bei *InOut*-Schnittstellen ist auf die Einstellung der Optimierung am Baustein und an den verschalteten Daten zu achten.

Nur wenn die Optimierungseinstellung gleich ist (Daten optimiert und Baustein optimiert oder Daten nicht optimiert und Baustein nicht optimiert) wird keine lokale Kopie der Daten vom System angelegt.

Ist die Optimierungseinstellung unterschiedlich, so wird von Arrays immer mindestens ein Element, von anderen Datentypen immer die kompletten Daten in den Stack kopiert. Dadurch verliert sich der Performancevorteil von InOut-Schnittstellen gänzlich.

Hinweis

Weitere Informationen finden Sie im „Programmierleitfaden für S7-1200/1500“ im Kapitel „Bausteinschnittstellen“.

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

Empfehlung: Outputvariablen nur einmal pro Zyklus schreiben

Einer *Output*-Variablen sollte pro Bearbeitungszyklus nur einmal am Ende des Bausteins ein neuer Wert zugewiesen werden. Dadurch stellt man sicher, dass alle Ausgänge so gut wie möglich konsistent gehalten werden.

Um dies zu erreichen, kann eine Variable im *Temp*- oder *Static*-Bereich angelegt werden, die dann innerhalb des Bausteins einen Stellvertreter des Ausgangs darstellt. Am Ende des Bausteins wird diese Stellvertreter-Variable dann der echten *Output*-Variable des Bausteins zugewiesen.

4.2 Variablendeklaration

4.2.1 Static und Temp

Regel: Statische Variablen werden nur lokal aufgerufen

Auf die statischen Daten (*Static*) eines Funktionsbausteins wird nicht außerhalb dieses Bausteins zugegriffen. Dies gilt insbesondere beim Aufruf und der damit verbundenen Verschaltung der Instanzdaten des Bausteins.

Regel: Präfix Static-Variablen: stat; Temp-Variablen: temp;

Um Static- und Temp-Variablen klar von Übergabe- und Ausgangsparametern im Code trennen zu können, werden die Präfixe aus Tabelle 4-2 verwendet. Dies erleichtert dem Benutzer eines Bausteins die Unterscheidung zwischen Schnittstellen-Variablen und internen Variablen. Somit können sofort die Zugriffsrechte für den Benutzer definiert und erkannt werden.

Die Präfixe gelten nicht für Global-DBs und PLC-Datentypen, sondern nur bei Bausteinen, die eine komplette Schnittstelle enthalten.

Tabelle 4-2: Präfixe bei Variablen

Präfix	Typ
stat	*Static*-Variablen → Kein Zugriff in den Instanzdaten von außerhalb erlaubt
temp	Temp-Variablen → Kein Zugriff in den Instanzdaten von außerhalb möglich
	Input- und *Output*-Variablen (kein Präfix) → Zugriff in den Instanzdaten von außerhalb möglich
	InOut-Variablen (kein Präfix) → Änderung der verschalteten Daten sowohl durch Benutzer als auch durch Baustein jederzeit möglich

4.2.2 Konstanten

Regel: Bezeichner von Konstanten immer in GROSSSCHRIFT und Unterstriche

Die Namen von Konstanten werden immer in Großschrift geschrieben. Zum Erkennen einzelner Wörter oder Abkürzungen sind Unterstriche zwischen den einzelnen Wörtern oder Abkürzungen einzusetzen.

Regel: Nur lokale Konstanten verwenden

Um eine spätere Verwendung der Bausteine in einer Bibliothek zu garantieren, werden nur lokale Konstanten in Bausteinen verwendet.

Damit wird garantiert, dass keine Fehler bei der Kompilierung im Anwender-programm wegen fehlender Programmteile auftreten können.

Sollen lokale Konstanten dem Benutzer des Bausteins zur Verfügung gestellt werden, müssen diese auch als globale Konstanten angelegt werden. Hierbei sollte im Namen der globalen Konstanten ein Verweis auf den Baustein oder die Bibliothek vorhanden sein.

Dies gilt insbesondere auch für Konstanten, die definierte Werte an Baustein-ausgängen kennzeichnen, wie z.B. Fehlernummern.

Globale Anwenderkonstanten können als *PLC-Variablen* in den Kopiervorlagen der Bibliothek angelegt werden. Diese globalen Konstanten werden jedoch nicht bei einer Verwendung des typisierten Bausteins im Projekt automatisch mit in den Controller übernommen.

Beispiel

Abbildung 4-3: Konstanten in einem FB

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

Empfehlung: Konstanten bei Wertabfragen ungleich 0 verwenden

Soll eine Variable im Code mit einem numerischen Wert ungleich 0 belegt oder verglichen werden, sind dafür Konstanten zu verwenden.

Dadurch wird eine Änderung des numerischen Wertes deutlich vereinfacht, da dieser nicht an mehreren Stellen im Code, sondern zentral in der Konstante geändert wird.

Beispiel

Abbildung 4-4: Verwendung Konstanten

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	

```
#statVelocity := 0.0; //Correct, cause assignment with
                      //default value of data type 0.0

//Correct --> Working with constants
IF (ABS(#velocity) < #MAX_VELOCITY) THEN
  #statVelocity := #velocity;
ELSIF (#velocity < 0) THEN
  #statVelocity := -1.0 * #MAX_VELOCITY;
ELSE
  #statVelocity := #MAX_VELOCITY;
END_IF;

//Wrong --> Working with numerical values
IF (ABS(#velocity) < 10.0) THEN
  #statVelocity := #velocity;
ELSIF (#velocity < 0) THEN
  #statVelocity := -10.0;
ELSE
  #statVelocity := 10.0;
END_IF;
```

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

Einzig der Speicherverbrauch im Ladespeicher der CPU steigt dadurch, dass die Anzahl der Zeichen in den Bausteinquellen steigt.

4.2.3 Arrays

Empfehlung: Array-Name ist immer Mehrzahl

Der Name eines Arrays ist immer Mehrzahl.

Beispiel

Array einer Struktur aus Achsen
axisData → not okay

axesData → okay

Empfehlung: Array-Index beginnt mit 0 und endet mit einer Konstanten

Array-Grenzen beginnen mit 0 und enden mit einer Konstante für die Obergrenze des Arrays (z.B. DIAG_BUFFER_UPPER_LIM).

Eine Array ab 0 ist sinnvoll, da bestimmte Systembefehle wie z.B. MOVE_BLK_VARIANT null-basiert arbeiten. Dadurch kann man den gewünschten Index direkt in die Systemfunktion geben und muss nicht umrechnen.

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

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

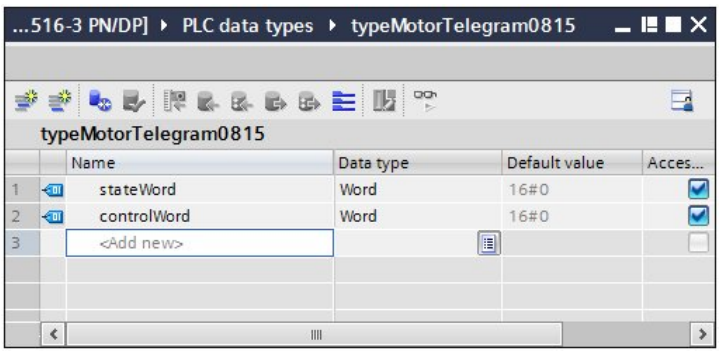
4.2.4 PLC-Datentypen

Regel: Präfix ‚type'

Dem Bezeichner eines anwenderdefinierten Datentyps wird das Präfix ‚type' vorangestellt.

Beispiel

Abbildung 4-5: Beispiel PLC-Datentyp



Regel: PLC-Datentypen statt Strukturen

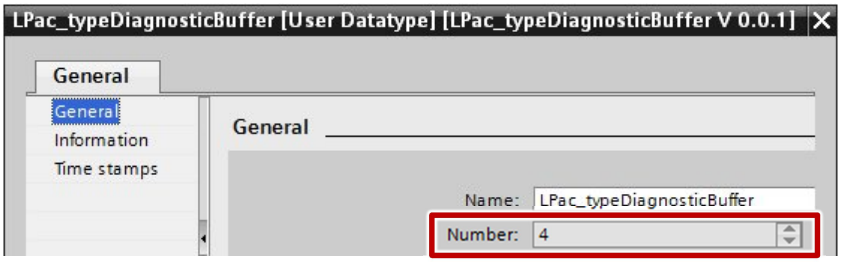
Es dürfen im PLC-Programm keine Strukturen mehr (wie in STEP 7 Classic üblich), sondern nur noch PLC-Datentypen verwendet werden.

Hinweis

Eine Ausnahme bilden Know-How-geschützte Bausteine. Hier sollte der Einsatz von PLC-Datentypen genau bedacht werden. Grund dafür ist, dass im Hintergrund für jeden PLC-Datentyp eine Nummer vergeben wird. Wird dieser PLC-Datentyp dann in einem Know-How-geschützten Baustein verwendet, muss diese Nummer beim Kopieren in ein anderes Projekt gleich bleiben. Ist dies nicht der Fall, lässt sich das neue Projekt nur mit Passwort für den Know-How-Schutz übersetzen.

Soll ein Baustein Know-How-geschützt werden, sollten PLC-Datentypen nur dort verwendet werden, wo typsichere Kopiervorgänge oder Verschaltungen mit dem entsprechenden strukturierten Datentyp vorgenommen werden.

Abbildung 4-6: Nummern bei PLC-Datentypen



4.2.5 Initialisierung

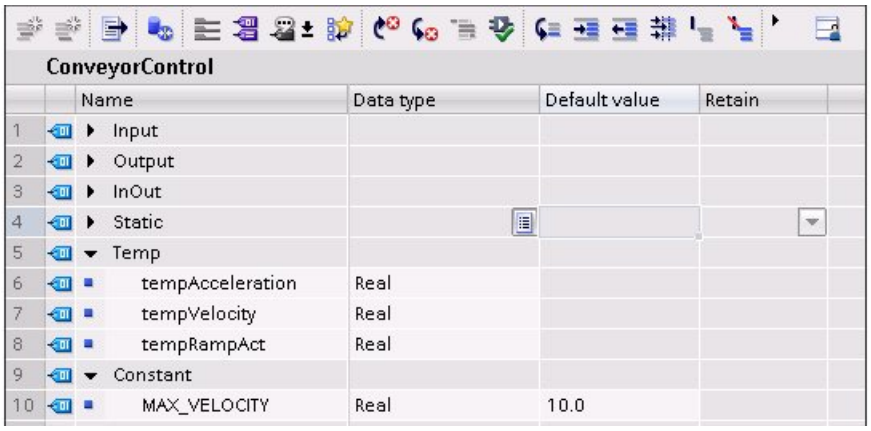
Regel: Temp-Variablen im Programm initialisieren

Variablen des L-Stacks (*Temp*) müssen am Anfang des Programms vom Anwender initialisiert werden.

Generell ist darauf zu achten, dass Variablen immer zuerst geschrieben bevor sie gelesen werden.

Beispiel

Abbildung 4-7: Initialisierung von temporären Daten




```
#tempAcceleration := 0.0;
#tempVelocity := #MAX_VELOCITY;
#tempRampAct := 0.0;
```

Regel: Initialisierung erfolgt in der gebräuchlichen Darstellung

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

Beispiel

Abbildung 4-8: Initialisierung von statischen Daten

Static			
statMask1	Word	16#01	not okay
statMask2	Word	16#0001	okay
statMask3	Byte	2#0000_1010	okay
statMask4	DWord	5	not okay
statCounter1	Int	16#00	not okay
statCounter2	Int	10	okay
statVelocity1	Real	16#0000	not okay
statVelocity2	Real	40.0	okay

Empfehlung: Parameterinitialisierung von TOs: -1.0

Anwenderdefinierte Parameterstrukturen, bei denen auch auf Werte eines TO zugegriffen werden soll (z.B. Geschwindigkeit, Beschleunigung, Ruck), werden mit dem Wert -1.0 initialisiert. Dies dient zur Unterscheidung, ob für den Parameter ein Wert übergeben wird. Bei keiner Belegung vom Anwender werden die Default-Einstellungen des TOs übernommen.

4.3 Anweisungen

4.3.1 Operatoren und Ausdrücke

Empfehlung: Vor und nach Operatoren ist ein Leerzeichen

Vor und nach binären Operatoren und dem Zuweisungsoperator steht ein Leerzeichen.

Beispiel

```
//Okay
#statSetValue := #statSetValue1 + #statSetValue2;

//Not okay
#statSetValue:=#statSetValue1+#statSetValue2;
```

Empfehlung: Ausdrücke immer in Klammern

Ausdrücke sind immer zu klammern, um die Reihenfolge der Interpretation eindeutig zu machen.

Beispiel

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


4.3.2 Programmsteueranweisungen

Empfehlung: Zeilenumbrüche bei Teil-Bedingungen

Bei komplexen Ausdrücken ist es sinnvoll jede „Teil-Bedingung“ durch einen Zeilenumbruch hervorzuheben. Damit sind auch übersichtliche Kommentare möglich.

Regel: Bedingungs- und Anweisungsteil werden mit Zeilenumbruch getrennt

Es ist eine klare Trennung von Bedingungsteil und Anweisungsteil einzuhalten.

D.h. nach einer Bedingung (z.B. nach THEN) muss immer ein Zeilenumbruch erfolgen, bevor eine Anweisung programmiert wird.

Empfehlung: IF-Anweisungen richtig einrücken

Boolesche Verknüpfungen werden, wenn eine Zeile nicht für die gesamte Bedingung ausreicht, an den Anfang der Zeile geschrieben.

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 alleine in einer Zeile. Eine alleinstehende Klammer zeigt das Ende der geschachtelten Bedingung. Operanden stehen immer am Anfang der Zeile.

Analog gelten diese Empfehlungen auch für Umgang mit anderen Anweisungen (z.B. WHILE, usw.)

Beispiel

```
IF (DriveStatus() = #OK) //Comment
  AND
  ((#statOldDrive XOR #tempActDrive)
  OR (#statOldPower XOR #tempActPower)
  ) //Comment
  AND (#start = True)
THEN
  ; //Statement
ELSE
  ; //Statement
END_IF;
```

Regel: Immer ELSE-Zweig bei CASE-Anweisungen erstellen

Eine CASE-Anweisung muss immer einen ELSE-Zweig aufweisen, um Fehler, die zur Laufzeit auftreten, melden zu können.

```
CASE #tempSelect OF
  1: //Comment
    ; //Statement
  4: //Comment
    ; //Statement
  2..5: //Comment
    ; //Statement
ELSE
  ; //Generate error message
END_CASE;
```

Empfehlung: CASE-Anweisung statt mehrerer ELSIF-Zweige

Wenn möglich, soll anstatt einer IF-Anweisung mit mehreren ELSIF-Zweigen eine CASE-Anweisung verwendet werden. Damit wird das Programm übersichtlicher.

Regel: Anweisungen einrücken

Jede Anweisung im Rumpf einer Kontrollstruktur wird eingerückt.

Beispiel

IF-Anweisung

```
//---------------
IF #tempCondition THEN
  ; //Statement
  IF #tempCondition2 THEN
    ; //Statement
  END_IF;
ELSE
  ; //Statement
END_IF;
```

Beispiel

CASE-Anweisung

```
//-------------------
CASE #statSelect OF
  CMD_INIT: //Comment
    ; //Statement
  CMD_READ: //Comment
    ; //Statement
  CMD_WRITE: //Comment
    ; //Statement
ELSE
  ; //Generate error message
END_CASE;
```

Beispiel

FOR-Anweisung

```
//---------------
FOR #tempIndex := 0 TO #MAX_NUMBER - 1 DO
  ; //Statement
END_FOR;
```

Beispiel

FOR-Anweisung mit Sprungweite

```
//---------------
FOR #tempIndex := 0 TO #MAX_NUMBER - 1 BY 2 DO
  ; //Statement
END_FOR;
```

Beispiel

Bedingte Beendigung einer Schleife mit EXIT

```
//---------------
FOR #tempIndex := 0 TO #MAX_NUMBER - 1 BY 2 DO
  IF #tempCondition THEN
    EXIT; //EXIT Loop
  END_IF;
END_FOR;
```

Beispiel

Schleifenbedingung erneut prüfen mit CONTINUE

```
//---------------
FOR #tempIndex := 0 TO #MAX_NUMBER - 1 BY 2 DO
  IF #tempCondition THEN
    CONTINUE; //loop condition
  END_IF;
END_FOR;
```

Beispiel

WHILE-Anweisung

```
//---------------
WHILE #tempCondition DO
  ; //Statement
END_WHILE;
```

Beispiel

REPEAT-Anweisung

```
//---------------
REPEAT
  ; //Statement
UNTIL #tempCondition END_REPEAT;
```

4.3.3 Fehlerbehandlung

Regel: Fehlercodes immer auswerten

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

Weitere Informationen zum Thema Fehlerbehandlung finden Sie im Kapitel 4.4.3 Fehlerrückgabe und Diagnose von Funktionsbausteinen.

4.4 Programmieren nach PLCopen

Die PLCopen Organisation hat einen Standard für Motion Control Bausteine definiert. Dieser Standard wird hier verallgemeinert und kann so auf alle asynchronen Funktionsbausteine angewendet werden. Beschrieben wird, wie die Schnittstelle eines Funktionsbausteins aussieht und sich die Signale dieser Schnittstelle verhalten.

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

Regel: Standardbezeichner bei PLCopen verwenden

Werden Parameter mit Standardbedeutung in Hinsicht auf Funktionalität nach *PLCopen Function Blocks for Motion Control V2.0* benötigt, sind die entsprechenden Standardbezeichner zu verwenden.

Bei folgenden Parametern handelt es sich um Standardparameter:

Tabelle 4-3: Standardparameter nach PLCopen

Signale Standardfunktion PLCopen-konform	Bedeutung
***Input*-Parameter**	
execute	**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 e*xecute*-Eingang neu getriggert werden.
oder	**execute mit ‚continuousUpdate':** Alle Parameter werden mit einer steigenden Flanke am *execute*-Eingang übernommen. Diese können solange angepasst werden, wie der Eingang *continuousUpdate* gesetzt ist.
enable	**enable:** 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).
***Output*-Parameter**	
Exklusivität: done busy valid commandAborted error	**Mit *execute*:** Die Ausgänge *done*, *busy*, *commandAborted* und *error* sind wechselseitig exklusiv, d.h. nur einer der Ausgänge kann zu einem Zeitpunkt gesetzt sein. Ist *execute* gesetzt, muss einer dieser Ausgänge gesetzt sein. **Mit *enable*:** Die Ausgänge *valid* und *error* sind gegenseitig exklusiv.
done	Der Ausgang *done* wird gesetzt, wenn das Kommando erfolgreich abgearbeitet wurde.
busy	***Mit execute*:** Der FB ist noch nicht mit der Bearbeitung des Kommandos fertig und somit können neue Ausgangswerte erwartet werden. *busy* wird bei steigender Flanke von *execute* gesetzt und rückgesetzt, wenn einer der Ausgänge *done*, commandA*borted* oder *error* gesetzt wird.

Signale Standardfunktion PLCopen-konform	Bedeutung
	Mit *enable*: Der FB ist gerade mit der Bearbeitung eines Kommandos beschäftigt. Neue Ausgangswerte können erwartet werden. *busy* wird mit einer steigenden Flanke von *enable* gesetzt und bleibt gesetzt, solange der FB Aktionen ausführt.
active	**Optionaler Ausgang** um Kompatibilität zu PLCopen (*buffered mode* von Funktionsbausteinen) herzustellen. Der Ausgang wird gesetzt, sobald der FB die Kontrolle über die Achse übernimmt. Ist kein *buffered mode* angewählt, kann *active* und *busy* identisch sein.
commandAborted	**Optionaler Ausgang**, der anzeigt, dass der laufende Auftrag des Funktionsbausteins durch eine andere Funktion bzw. durch einen anderen Auftrag an das gleiche Objekt abgebrochen wurde. Beispiel: Eine Achse wird über den Funktionsbaustein gerade positioniert, während über einen anderen Funktionsbaustein die gleiche Achse angehalten wird. Am Funktionsbaustein der Positionierung wird dann der Ausgang commandAborted gesetzt, da dieser Auftrag durch das Halt-Kommando abgebrochen wurde.
valid	Ausgang wird nur in Verbindung mit *enable* verwendet. Der Ausgang ist gesetzt, solange gültige Ausgangswerte verfügbar sind und der *enable* Eingang gesetzt ist. Sobald ein Fehler ansteht, wird der Ausgang *valid* zurückgesetzt.
error	Steigende Flanke des Ausgangs signalisiert, dass ein Fehler während der Abarbeitung des FB aufgetreten ist.
status (statt errorID)	Fehlerinformation oder Status des Bausteins Entgegen des PLCopen Standards wird aus Gründen der Kompatibilität mit bestehenden SIMATIC-Systemfunktionen und –Bausteinen auf den Bezeichner *errorID* verzichtet und stattdessen *status* verwendet.
diagnostics	**Optionaler Ausgang**: Detaillierter Fehlerpuffer Hier werden alle Fehler, Warnungen und Informationen des Bausteins in einem Ringpuffer abgelegt. Die Größe (Anzahl der Array-Elemente) orientiert sich an dem verfügbaren Speicher der durch die Applikation unterstützten PLCs. Der Aufbau der Diagnose ist in Kapitel 4.4.3 Fehlerrückgabe und Diagnose von Funktionsbausteinen beschrieben.

4.4.1 Bausteine mit execute

Mit einer steigenden Flanke am Parameter e*xecute* wird der Auftrag gestartet und die an den Eingangsparametern anstehenden Werte übernommen.

Nachträglich geänderte Parameterwerte werden erst beim nächsten Auftragsstart übernommen, wenn kein *continousUpdate* verwendet wird.

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

Nach Ende des Auftrags ist eine neue steigende Flanke an *execute* notwendig um einen neuen Auftrag zu starten.

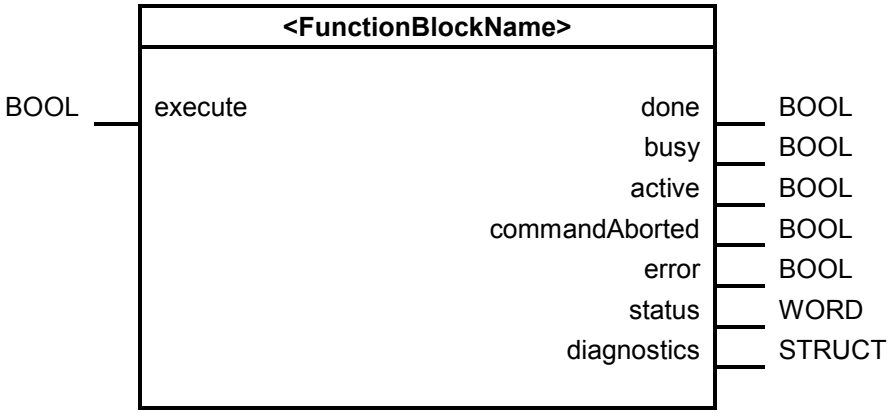
Somit ist gewährleistet, dass bei jedem Start eines Auftrags der Baustein im Initialzustand ist und die Funktion unabhängig von vorherigen Aufträgen bearbeitet.

Regel: Parameter: execute erfordert busy und done

Wenn der Programmierer den Input-Parameter *execute* benutzt, müssen die Output-Parameter *busy* und *done* verwendet werden.

Beispiel

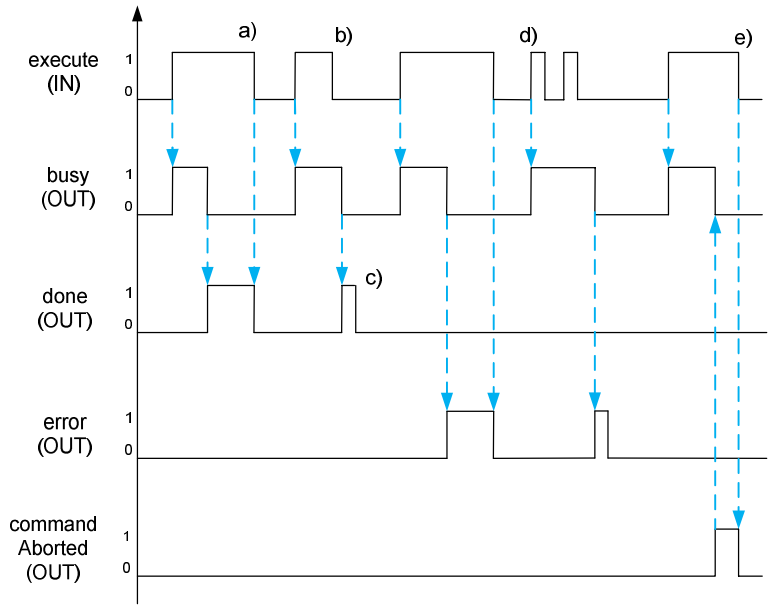
Abbildung 4-9: KOP-Darstellung



Signalablaufdiagramm Baustein mit execute

ACHTUNG	Wird der Eingang *execute* zurückgesetzt, bevor der Ausgang *done* gesetzt ist, so ist der Ausgang *done* nur einen Zyklus zu setzen.

Abbildung 4-10 : Signalablaufdiagramm eines Funktionsbausteins mit execute-Eingang



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

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

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

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

e) Wird die Bearbeitung eines Auftrags durch einen höher oder gleich prioren Auftrag (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 anderer Baustein einen Verfahrauftrag an dieser Achse ausführt.

4.4.2 Bausteine mit enable

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

Mit dem Rücksetzen des Parameters e*nable* wird der Auftrag beendet.

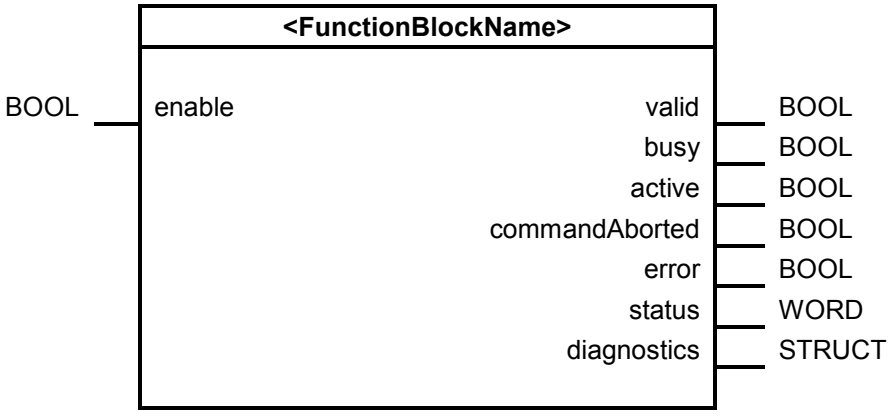
Wird ein neuer Auftrag gestartet, wird der Baustein in seinen Initialzustand versetzt und kann wieder völlig neu beschaltet und parametriert werden.

Regel: Parameter: enable erfordert valid

Wenn der Programmierer den Input-Parameter *enable* benutzt, muss mindestens der Output-Parameter *valid* verwendet werden.

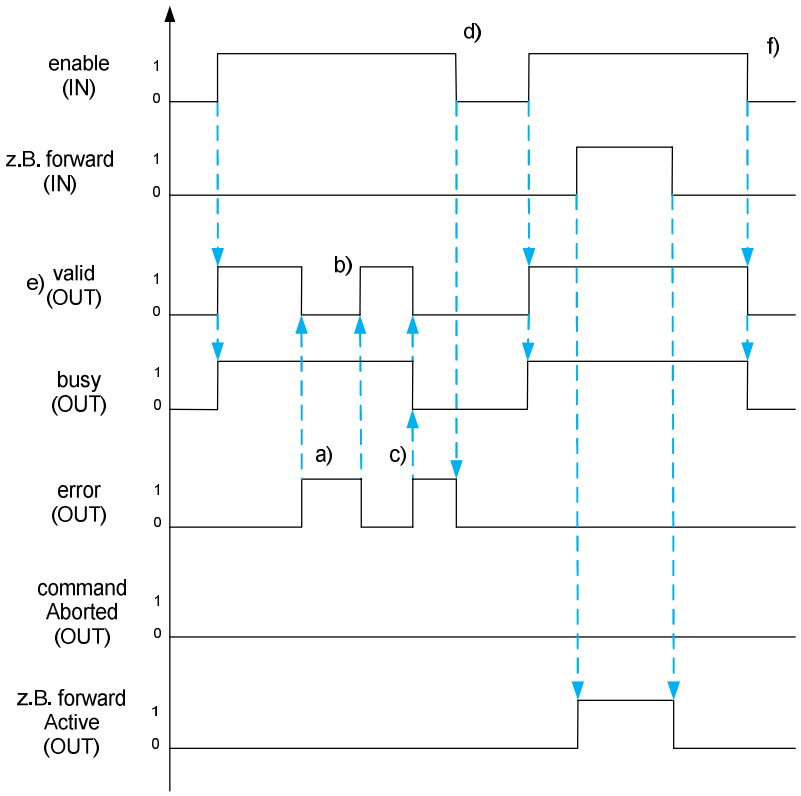
Beispiel

Abbildung 4-11: KOP-Darstellung



Signalablaufdiagramm Baustein mit enable

Abbildung 4-12: Signalablaufdiagramm eines Funktionsbausteins mit enable-Eingang



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

b) Nach Behebung der Fehlerursache (z.B. Neuaufbau der Kommunikationsverbindung) wird *valid* wieder gesetzt.

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

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

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

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

CommandAborded, error und done sind immer so lange gesetzt, wie das Signal execute ansteht, mindestens jedoch für einen Zyklus.

4.4.3 Fehlerrückgabe und Diagnose von Funktionsbausteinen

Hinweis

Dieses Kapitel ist nicht mehr Bestandteil des *PLCopen Function Blocks for Motion Control V2.0* Standards.

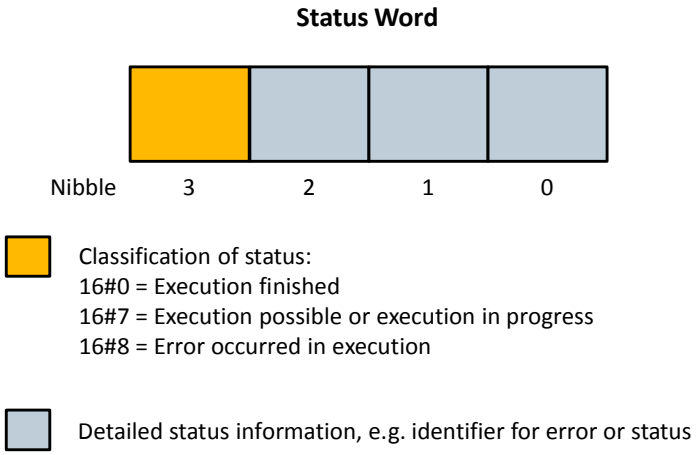
Folgende zusätzliche Regeln und Empfehlungen beschreiben weitere Vorgaben zum einheitlichen Programmieren von Fehlerrückgabe und Diagnose in Funktionsbausteinen.

Regel: Formalparameter status: generelle Fehlerrückgabe

Ein Fehler wird durch Setzen der booleschen Variablen *error* angezeigt. Gleichzeitig wird durch Setzen des höchstwertigen Bits im Ausgang *status* ein Fehler angezeigt. Die restlichen Bits werden für einen Fehlercode benutzt, der eindeutig auf die Ursache hinweist. Aus Kompatibilitätsgründen zu bisherigen SIMATIC-Systembausteinen wird auf den Ausgang *errorID,* der nach dem PLCopen Standard vorgeschrieben ist, zugunsten des Ausgangs *status* verzichtet.

Alternativ kann die Anbindung an ein Fehlerkonzept (z.B. Meldehandling) realisiert werden. Dann müssen die Variablen entsprechend des Konzepts realisiert werden. Z.B. Fehlernummer mit mehreren Begleitwerten, Diagnosestruktur, ...

Abbildung 4-13: Aufbau des Ausgangs *status*



Empfehlung: Parameter status: standardisierte Fehlernummern

Für eine Standardisierung der Fehler sind die in der folgenden Tabelle gezeigten Nummernbänder für Fehlergründe einzuhalten.

Tabelle 4-4: Nummernbänder für Fehler

Fehlergrund	Nummernband *status*
Auftrag abgeschlossen, keine Warnung oder weitere Detaillierung	16#0000
Auftrag abgeschlossen, weitere Detaillierung	16#0001 ... 16#0FFF
Kein Auftrag in Bearbeitung	16#7000

Fehlergrund	Nummernband *status*
(**auch Initialwert**)	
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
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

Empfehlung: Fehler steht an bis Quittierung

Wird ein Fehler bei der Bearbeitung eines Funktionsbausteins festgestellt, wird der aktuelle Auftrag und somit z.B. die Bewegung gestoppt. Der Fehlercode zum ersten Fehler bleibt solange anstehen, bis dieser quittiert wird (negative Flanke von *execute;* bei *enable* je nach Fehlertyp ebenfalls fallende Flanke notwendig).

Empfehlung: Statuscodes von Anweisungen am Ausgang *status* ausgeben

Statuscodes von Anweisungen (Systembausteinen) werden unverändert am Ausgang *status* ausgegeben. Somit kann bei der Dokumentation der Bausteine auf die Fehlercodes in der TIA Portal Hilfe verwiesen werden.

Empfehlung: Ausgang *statusID* zur Identifizierung der Fehlerquelle

Um die Fehlerquelle eindeutig identifizieren zu können, empfiehlt es sich, den zusätzlichen Ausgang *statusID* mit folgenden Eigenschaften zu verwenden:

- Er gibt an, welcher Baustein oder Subbaustein (Subinstanz) einen Fehler meldet. Es empfiehlt sich den aufrufenden Baustein die *statusID* „1“ und den Subbausteinen Nummern ab „2“ zuzuweisen.
- Er gibt den Wert „0“ zurück, wenn keine Fehler/Meldungen anstehen.
- Er ist ein Output vom Datentyp UINT.

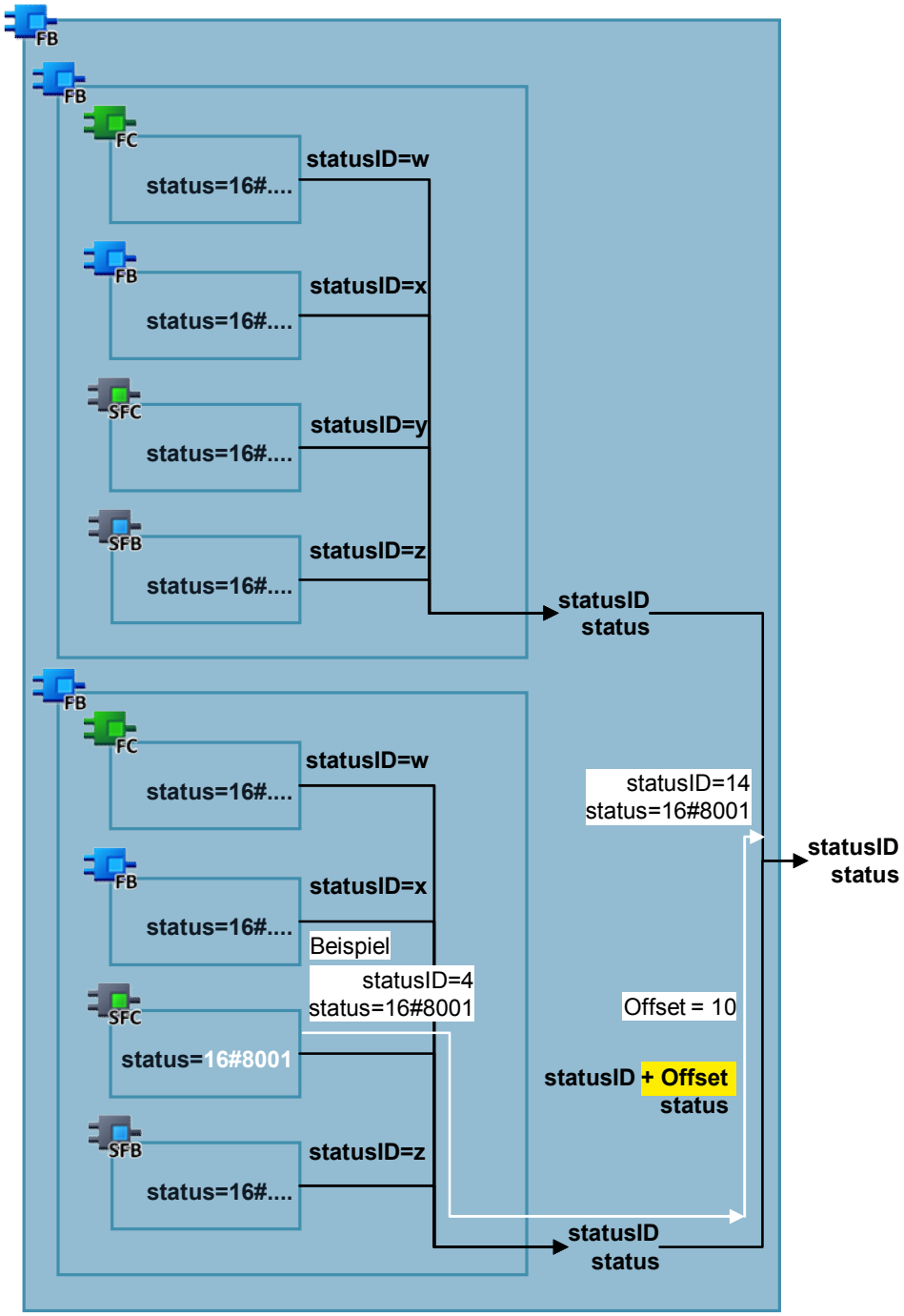
Alle Instanzen bekommen eine eindeutige *statusID* innerhalb des aufrufenden Bausteins zugeordnet.

Empfehlung: Ausgang *statusID* und Offset bei verschachtelten Bausteinen

Bei verschachtelten Bausteinen empfiehlt es sich, am übergeordneten Baustein dem Ausgang *statusID* eine eindeutige Zuordnung der Fehlerquelle (aufgerufene Instanz) zu programmieren. Dies geschieht, indem ein *Offset* auf den Wert von *statusID* der unterlagerten Bausteine addiert wird, falls mehrfach geschachtelt wird. Somit kann programmweit eine eindeutige *statusID* definiert werden.

Beispiel

Abbildung 4-14: *status* und *statusID* bei geschachtelten Bausteinen



Empfehlung: Parameter diagnostics: Diagnosestruktur

In einer Diagnosestruktur sind alle weiteren Informationen am Ausgang *diagnostics* über den aufgetretenen Fehler abzulegen. Des Weiteren können dort auch Werte zur Diagnose des aktuellen Bausteinverhaltens, wie z.B. Laufzeitinformationen abgelegt werden.

Abbildung 4-15: Aufbau der Diagnosestruktur

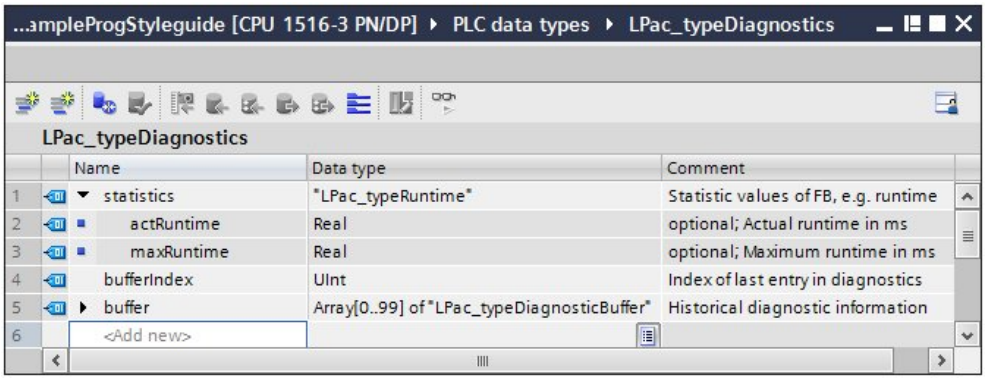


Tabelle 4-5: Elemente einer Struktur *diagnosticBuffer*

Name	Datentyp	Optional	Kommentar
timestamp	DATE_AND_TIME		Zeitstempel bei Auftritt des Fehlers
stateNumber	DINT		State aus der internen State-Machine bei Auftritt der Fehlers
modeNumber	DINT	x	Mode aus der internen Mode-State-Machine bei Auftritt des Fehlers
subfunctionStatus	DWORD	x	Rückgabewert bei Fehlern von aufgerufenen FBs und FCs und Systembausteinen
status	WORD		Status der den Fehler eindeutig identifiziert
additionalValueX	beliebig	x	Zusätzliche Werte (X = Nummer), um fehlerspezifische Informationen zur Diagnose speichern zu können (z.B. Achsposition).

Im Parameter *timestamp* wird der Zeitpunkt abgespeichert, zu dem der Fehler aufgetreten ist.

In *stateNumber* wird der aktuelle State der internen State Machine abgespeichert. Bei einem Funktionsbaustein mit verschiedenen Betriebsarten wird in der Variablen *modeNumber* die Betriebsart gespeichert, in welcher der Fehler aufgetreten ist.

Wurde ein Fehler einer Systemfunktion oder eines aufgerufenen FBs / FCs festgestellt, wird der Rückgabecode im Element *subfunctionStatus* gespeichert.

Der eindeutige Fehlercode vom Output *status* wird zusätzlich im Element *status* der Diagnosestruktur gespeichert.

Zusätzliche Parameter zu einem Fehler werden in den a*dditionalValueX*-Variablen gespeichert. Die neutrale Bezeichnung der *additionalValueX* Werte sollte

beibehalten werden, damit das Abspeichern verschiedenster Werte auf einem Speicherplatz möglich ist.

Sind weitere Elemente notwendig, können diese hinzugefügt werden.

Empfehlung: Remanente Diagnosestruktur

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

4.5 Tabellen, Traces, Messungen

Regel: PascalCase-Schreibweise bei Tabellen und Traces

PascalCase Schreibweise (erster Buchstabe ist groß) wird verwendet für:

- PLC-Variablen Tabellen
- Beobachtungstabellen
- Traces
- Messungen

4.6 Bibliotheken

In diesem Kapitel werden Regeln und Empfehlungen für die Programmierung von Bibliotheken vorgegeben. Die in den vorherigen Kapiteln aufgestellten Regeln für Quellcode und Variablennamen sind bindend für die Erstellung von Bibliotheken.

Empfehlung: Dokumentation für Bibliotheken

Generell wird empfohlen jede Bibliothe ausreichend in einer Dokumentation zu beschreiben:

- Bausteine und ihre Funktionen
- Versionierung
- Änderungshistorie
- usw.

4.6.1 Namensvergabe

Empfehlung: Bibliothekname: Präfix L und Länge max. 8 Zeichen

Der Name einer Bibliothek erhält das Präfix L (z.B. LPac). L steht für das englische Wort Library. Es wird, außer zwischen dem Präfix der Bibliothek und Baustein- / Konstanten-Namen, keine Unterstriche verwendet. Die maximale Zeichenlänge für einen Bibliotheksnamen und somit für das Präfix wird auf 8 Zeichen begrenzt.

Diese Beschränkung dient zur kompakten Namensvergabe.

Regel: Alle Elemente erhalten Name der Bibliothek als Präfix

Alle in einer Bibliothek vorhandenen Elemente (PLC- und HMI-Elemente) erhalten den Namen der Bibliothek. Dadurch wird sichergestellt, dass es keine Kollisionen bei Bausteinnamen gibt.

ACHTUNG	Wird ein Baustein in eine Bibliothek eingefügt, müssen bereits vor dem Einfügen alle Eigenschaften des Bausteins wie z.B. Bausteinnummer und Know-how-Schutz gesetzt sein. Ist der Baustein einmal in einer Bibliothek können seine Eigenschaften nicht mehr verändert werden.

Beispiel

Tabelle 4-6: Beispiel für Namensvergabe für Bibliothek LExample

Typ	Name nach Styleguide
Bibliothek	LExample
PLC-Datentyp	LExample_type<Name>
Funktionsbaustein	LExample_<Name>
Funktion	LExample_<Name>
Organisationsbaustein	LExample_<Name>
PLC-Variablen	LExample_<Name>
allgemeine Konstante	LEXAMPLE_<NAME>
Konstante für Fehlercode	LEXAMPLE_ERR_<NAME>

Beispiel

Bezeichner:	LCom_CommToClient
Bibliothek:	LCom
Funktionalität:	Kommunikation über TCP/IP zwischen verschiedenen Geräten

4.6.2 Aufbau

Regel: Typen: FC, FB, PLC-Datentypen

Funktionen, Funktionsbausteine und PLC-Datentypen werden als Typen in eine Bibliothek hinzugefügt. Alles andere wird als Kopiervorlage in eine Bibliothek hinzugefügt, v.a. Organisationsbausteine und Variablentabellen.

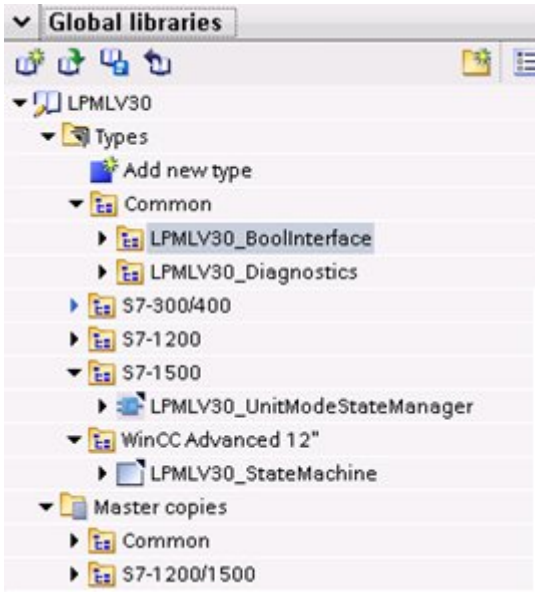
ACHTUNG	**Durch den Know-how-Schutz wird der Baustein an den Steuerungstyp und die Firmware gebunden, wie er zuletzt übersetzt wurde.** D.h. ist der Baustein in der Entwicklungsphase auf einer S7-1500 Steuerung übersetzt worden, kann er trotz S7-1200 kompatibler Programmierung nicht auf einer S7-1200 eingesetzt werden. Hierbei ist auch zu bedenken, dass der Baustein neben dem Steuerungstyp auch an die Firmware gebunden wird. Ein know-how-geschützter Baustein kann ohne Passwort nicht wieder übersetzt werden. Werden PLC-Datentypen verwendet, muss der Benutzer darauf achten diese nicht zu verändern. PLC-Datentypen können nicht know-how-geschützt werden.

Empfehlung: Gruppieren in der Bibliothek

Die PLC-Bausteine und HMI-Bilder einer Bibliothek werden einer Gruppe zugeordnet, die den Namen des Steuerungs- oder HMI-Typs trägt. Ist ein Baustein (z.B. PLC-Datentyp) oder HMI-Bild für alle Typen der Steuerungen (S7-300, S7-400, S7-1200 und S7-1500) oder HMIs (…) gültig, ist der Ordner Common zu verwenden.

Beispiel

Abbildung 4-16: Aufbau einer Bibliothek



4.6.3 Versionierung

Regel: Definition der Versionierung

Die offizielle Versionierung (erster freigegebener Stand) beginnt mit der Version V1.0.0 (siehe Tabelle 4-7). Versionsstände kleiner als 1.0 kennzeichnen Entwicklungsstände.

Die dritte Stelle in der Softwareversionierung kennzeichnet Änderungen, die keine Auswirkung auf die Dokumentation haben, wie beispielsweise reine Fehlerbehebungen, die keine neuen Funktionen beinhalten.

Bei Erweiterung der bestehenden Funktionalität wird die zweite Stelle hoch gezählt.

Bei einer neuen Hauptversion, die neue Funktionalitäten aufweist und inkompatibel ist, wird die erste Stelle inkrementiert.

Regel: Lückenlose Versionierung

Die Versionierung der Bibliothek ist lückenlos. Bei Änderung wird immer die Versionsnummer der Bibliothek hoch gezählt. Zusätzlich werden die Bausteine nach obigem Versionsschema versioniert. Hierbei ist es möglich, dass keiner der Bausteine die Bibliotheksversion trägt, da diese teilweise unabhängig voneinander versioniert werden (siehe Beispiel unten).

Beispiel

Tabelle 4-7: Beispiel für Änderung der Version

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
1.1.0	1.1.0	1.0.1	1.0.0		Erweiterung an FB1
1.2.0	1.2.0	1.0.1	1.0.0		Erweiterung an FB1
2.0.0	2.0.0	1.0.1	2.0.0		neue Funktionalität an FB1 und FC1
2.0.1	2.0.0	1.0.2	2.0.0		Fehlerbehebung FB2
3.0.0	2.0.0	1.0.2	2.0.0	1.0.0	Neue Funktion FC2

Regel: Änderungen und Version im Bausteinkopf pflegen

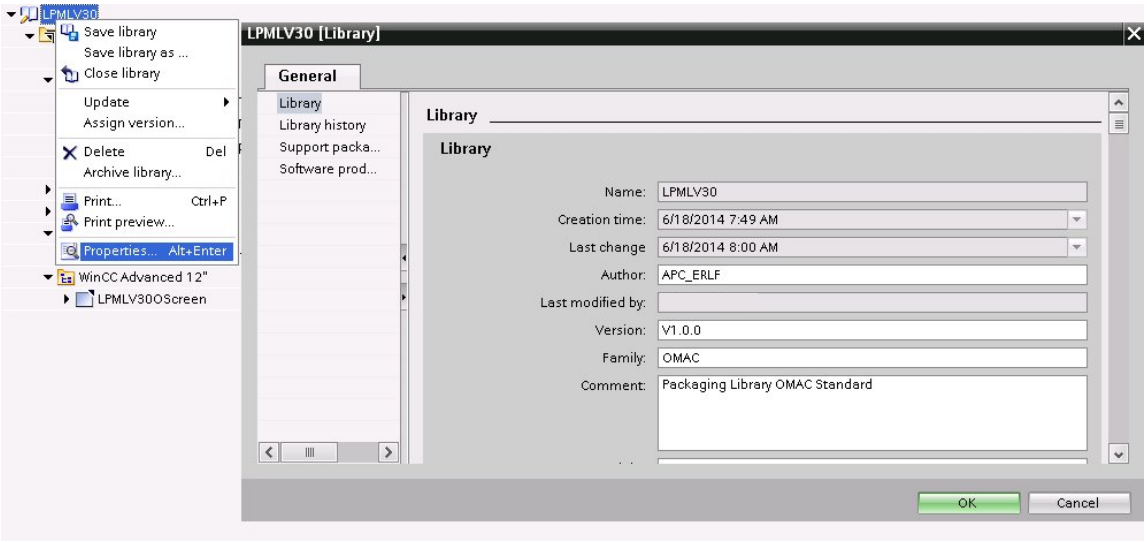
Es werden bei jeder Änderung der Version die Anpassungen an den entsprechenden Stellen, z.B. im Bausteinkopf der Funktion, beschrieben.

Regel: Eigenschaftendialog pflegen: Version, Abteilungskürzel

Die aktuelle Version der Bibliothek wird im Eigenschaften-Dialog der Bibliothek eingetragen. Bei Standardbibliotheken wird im Eigenschaftsfenster ein eindeutiges Kürzel für die entsprechende Abteilung hinterlegt.

Beispiel

Abbildung 4-17: Eigenschaften einer Bibliothek



4.6.4 Performancetest

Empfehlung: Performancetest

Bevor eine Bibliothek ausgeliefert wird, sollen die Funktionsbausteine auf einer CPU im mittleren Leistungsbereich mit kleinen-mittleren Mengengerüst (z.B. CPU 1212 oder CPU 1511-1 PN) getestet werden, um Performance- und Speicherprobleme frühzeitig zu erkennen.

4.6.5 Auslieferung

Empfehlung: Auslieferung als zip-Datei

Eine Bibliothek sollte nicht als Ordnerstruktur sondern als Archiv (Dateiformat .zip oder TIA Portal Archiv) ausgeliefert werden.

4.6.6 Beispielprojekt

Empfehlung: Beispielprojekt bei komplexen Bibliothekselementen

Ist ein Baustein komplexer oder besitzt die Applikation neben PLC-Code auch HMI Seiten, sollte zusätzlich zu einer Bibliothek auch noch ein Beispielprojekt erstellt werden.

Das Beispielprojekt muss ohne Anpassungen auf der für diese Applikation gebräuchlichen Steuerung ablauffähig sein. Das HMI soll ebenfalls ohne Anpassung sofort im Beispielprojekt verwendbar sein.

Empfehlung: HMI-Bilder möglichst nur in Beispielprojekten

Sind HMI-Seiten nicht zwingender Bestandteil einer Bibliothek oder einer Applikation, sollten diese nur im Beispielprojekt vorkommen. Dadurch kann man geschickt die Problematik umgehen, HMI-Seiten für die verschiedenen Panelarten (WinCC Runtime, Comfort- und Basic-Panels) und Panelgrößen anbieten zu müssen.

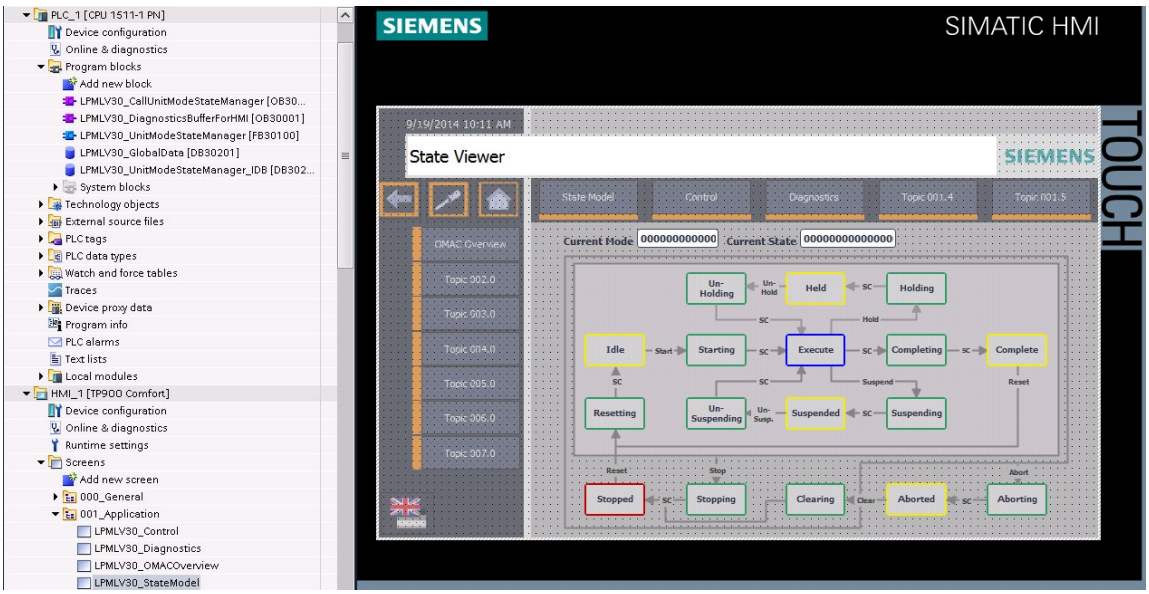
Empfehlung: HMI OS-Templates verwenden

Für Beispielprojekte sollte das HMI OS-Template \6\ verwendet werden.

https://support.industry.siemens.com/cs/ww/de/view/96003274

Beispiel

Abbildung 4-18



Empfehlung: Standardgeräte verwenden, wenn möglich

Sind keine Vorgaben bezüglich der Hardware vorhanden, sollten folgende Geräte verwendet werden:

Tabelle 4-8

Typ	Gerät
S7-1500	CPU 1513-1PN
S7-1200	CPU 1214
S7-300	CPU 315-2 PN/DP
Comfort Panel	TP900

5 Literaturhinweise

Tabelle 5-1

	Themengebiet	Titel
\1\	Siemens Industry Online Support	http://support.industry.siemens.com/
\2\	Downloadseite des Beitrages	https://support.industry.siemens.com/cs/ww/de/view/81318674
\3\	Totally Integrated Auomation	http://www.siemens.de/tia
\4\	Basisfunktionen für modulare Maschinen	https://support.industry.siemens.com/cs/ww/de/view/61056223
\5\	Programmierleit-faden für S7-1200/1500	https://support.industry.siemens.com/cs/ww/de/view/81318674
\6\	Know-how im Online Support	https://support.industry.siemens.com/cs/ww/de/view/96003274
\7	Normen	Weitere Informationen finden Sie in der Norm IEC 61131-8 bzw. EN 61131-3 Beiblatt 1.

6 Historie

Tabelle 6-1

Version	Datum	Änderung
V1.0	10/2014	Erste Version nach interner Freigabe
V1.1	06/2015	Anpassungen und Korrekturen