hxql

Programmier-Style-Guide

TIA Portal STEP 7

Ein Leitfaden für die S7-Programmierung im SIEMENS TIA-Portal

Dateiname: Programmier_Style_Guide_S7_hxql.docx
Version: 1.1.0
Status: Freigegeben
Geheimhaltungsstufe: Öffentlich

	Datum	Name	Abteilung	Telefon	Unterschrift
Erstellt	17.09.2025	M. Heymann	Entwicklung		
Geprüft	08.01.2026	F. Babel	Entwicklung		
Freigegeben	13.01.2026	S. Büttner	Entwicklung		

Inhalt

1 Einleitung 3
1.1 Ziel und Zweck 3
1.2 Geltungsbereich (Scope) 3
2 Allgemeine Vorschriften 4
3 Programmstruktur 5
3.1 Aufrufstruktur 5
3.2 Teilbereiche 5
3.3 Ordnerstruktur 5
3.4 Variablentabellen 6
3.5 Sicherheitsprogramm 7
4 Bausteine 8
4.1 Aufrufende Bausteine 8
4.2 SCL-Bausteine 9
4.3 Baustein-Kommentare 10
4.4 Beispiel-Baustein 10
5 Namenskonventionen für Bausteine 15
6 Namenskonventionen für Variablen 16
6.1 Allgemein 16
6.2 Präfix 16
6.2.1 Teil 1: Funktion 16
6.2.2 Teil 2: Datentyp 16
6.3 Festgelegte Abkürzungen 18
6.4 Variablen-Kommentare 19
6.5 Beispiele für Variablennamen 19
7 Verzeichnisse 20
7.1 Abbildungsverzeichnis 20
7.2 Tabellenverzeichnis 20
8 Dokumenteninformation 21

© hxql solutions GmbH

1 Einleitung

1.1 Ziel und Zweck

In diesem Dokument werden Vorgaben in Form eines Leitfadens zur Vereinheitlichung der S7-Programmierung im SIEMENS TIA-Portal festgehalten. Durch Einhaltung dieser Vorgaben sollen die Übersichtlichkeit, die Lesbarkeit sowie die Wartbarkeit der Softwareprojekte erhöht werden.

1.2 Geltungsbereich (Scope)

Dieses Dokument gilt als Leitfaden für alle Projekte, welche im SIEMENS TIA-Portal erstellt werden.

© hxql solutions GmbH

2 Allgemeine Vorschriften

Um eine Vermischung von Deutsch und Englisch zu vermeiden, werden generell alle Bezeichner und Kommentare in Englisch verfasst.

Der Programmaufbau, sowie die Bausteinaufrufe folgen einer festgelegten Struktur, damit man sich jederzeit schnell zurechtfinden kann, auch wenn man das Projekt nicht selbst erstellt hat.

Das Benennen von Bausteinen und Variablen erfolgt nach den hier im Dokument festgelegten Konventionen. Das erhöht nicht nur die Lesbarkeit, sondern bietet auch Vorteile bei der Autovervollständigung.

Außer den System- und Taktmerkerbits dürfen keine Merker verwendet werden. Variablen werden sinnvoll strukturiert in Datenbausteinen abgelegt.

Temporäre Variablen sind vor der Verwendung immer zu initialisieren, um unbestimmte Zustände zu vermeiden.

© hxql solutions GmbH

3 Programmstruktur

3.1 Aufrufstruktur

In der normalen Aufrufstruktur werden im „Main [OB1]“ alle Bausteine aufgerufen, welche wiederum alle Bausteine eines Teilbereichs aufrufen (CallSystem, CallFielLevel, ...). In den Teilbereichen können ebenfalls Bausteine sein, welche zum Beispiel mehrere Bausteine eines Typs aufrufen (z.B.: CallDigitalInputs).

3.2 Teilbereiche

Die Bereiche teilen sich auf in System, welcher alle Systemrelevanten Funktionen zugeordnet werden, wie zum Beispiel die Berechnung der Zykluszeit, danach folgt die Struktur der Automatisierungspyramide mit Feldebene, Steuerungsebene und Leitebene. Der Feldebene werden alle Bausteine für Sensorik und Aktorik zugeordnet, wie z.B. Analogeingang, Digitalausgang, Motor und Ventil. Die Steuerungsebene enthält alle Bausteine, welche alle logischen Verknüpfungen bzw. alle Abläufe der Maschine oder Anlage beinhalten. Zur Leitebene gehören Datenaustausch und -Aufbereitung zu HMIs oder übergeordneten Prozessleitsystemen. Außerdem gibt es Ordner für Standard- und Nichtstandardbibliotheksbausteine.

3.3 Ordnerstruktur

Da sonst alle Bausteine nur sortiert nach Bausteintyp untereinanderstehen würden, werden die Bausteine, entsprechend ihrem Teilbereich, auf Ordner (TIA-Portal: „Gruppe“) verteilt. Die vorgegebene Ordnerstruktur darf erweitert werden, um die Übersicht zu wahren. Möglich wäre zum Beispiel ein neuer Teilbereich **„40_RecipeManagement“** welcher Bausteine für eine Rezepturverwaltung enthält oder **„40_Safety“** für den fehlersicheren Programmteil. Es können aber auch Unterordner unterteil werden, wenn es der besseren Übersichtlichkeit dient.

Die Ordnerstruktur soll wie folgt aufgebaut sein (mit beispielhaften Bausteinen).

00_System (Ordner)
- CallSystem (aufrufende Funktion)
- **01_Logic** (Ordner)
  - CycleTime (Aufgerufene Funktion)
- **02_Data** (Ordner)
  - GdbCycleTime (Globaler Datenbaustein)

10_FieldLevel (Ordner)
- CallFieldLevel (aufrufende Funktion)
- **11_Logic** (Ordner)
  - CallAnalogInputs (Aufrufender Funktionsbaustein)
  - CallValves (Aufrufender Funktionsbaustein)
- **12_Data** (Ordner)

© hxql solutions GmbH

GdbAnalogInputsPlc (Globaler Datenbaustein)
GdbValvesPlc (Globaler Datenbaustein)
IdbAnalogInputs (Instanzdatenbaustein)
IdbValves (Instanzdatenbaustein)

20_ControlLevel (Ordner)
CallControlLevel (aufrufende Funktion)
21_Logic (Ordner)
ProcessA (Aufgerufene Funktion)
22_Data (Ordner)
GdbProcessA (Globaler Datenbaustein)

30_SuperVisoryLevel (Ordner)
CallSuperVisoryLevel (aufrufende Funktion)
31_Logic (Ordner)
SuperVisorySystemCom
32_Data (Ordner)
GdbSuperVisorySystemCom (Globaler Datenbaustein)
IdbSuperVisorySystemCom (Instanzdatenbaustein)

80_StandardLibrary (Ordner)
HSL_01_Signals (Ordner) (hxql Standard Library)
DigitalInput (Standard-Bibliotheksbaustein)
AnalogInput (Standard-Bibliotheksbaustein)
HSL_02_Objects (Ordner) (hxql Standard Library)
Motor (Standard-Bibliotheksbaustein)
Valve (Standard-Bibliotheksbaustein)
HSL_03_Diagnosis (Ordner) (hxql Standard Library)

90_NonStandardLibrary (Ordner)
Enthält mehrfach verwendete Bausteine, welche nicht Bestandteil der Standardbibliothek sind.

3.4 Variablentabellen

Für jedes Feldgerät ist eine separate Variablentabelle anzulegen, welche unter anderem mit dem entsprechenden Betriebsmittelkennzeichen benannt wird.

© hxql solutions GmbH

3.5 Sicherheitsprogramm

Da in der Praxis auch im Sicherheitsprogramm Informationen für das Diagnose- und Meldekonzept anfallen und auch betriebliche Informationen für das Sicherheitsprogramm relevant sind, können beide Programmteile nicht komplett getrennt werden.

Für einen Datenaustausch zwischen Standard-Anwenderprogramm und Sicherheitsprogramm sind Globale Datenbausteine zu verwenden. Um eine gute Übersicht zu bewahren, welcher Programmteil liest und welcher schreibt, empfiehlt es sich, für beide Richtungen je einen Datenbaustein anzulegen (siehe Abbildung).

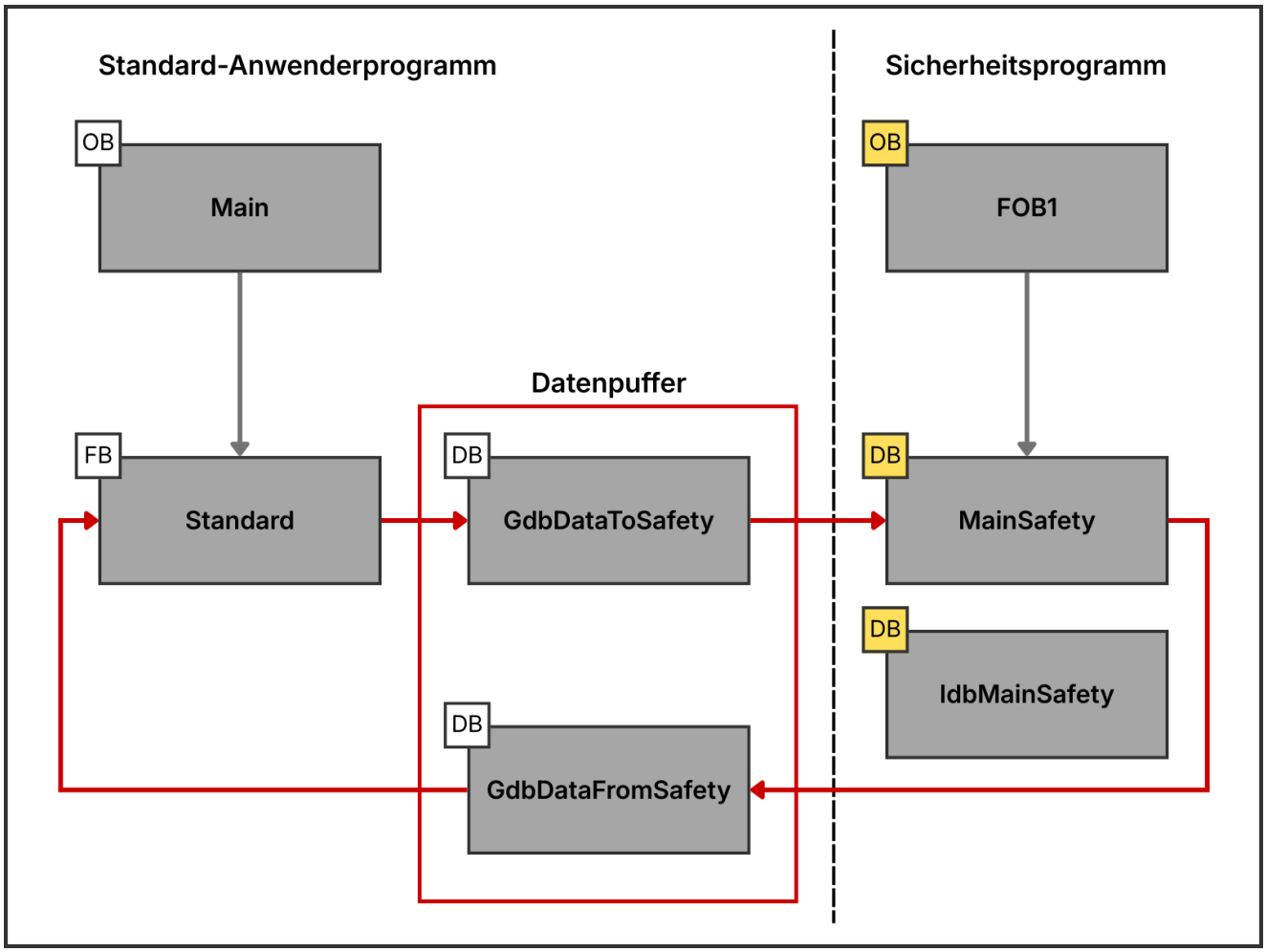


Abbildung 3-1: Datenaustausch mit Sicherheitsprogramm

© hxql solutions GmbH

4 Bausteine

Funktionen und Funktionsbausteine werden grundsätzlich in SCL geschrieben. Die einzige Ausnahme bilden Bausteine, deren Hauptfunktion der Aufruf anderer Bausteine ist (OB oder „Call…“), diese sind in FUP zu erstellen.

4.1 Aufrufende Bausteine

Der Baustein „Main [OB1]“ soll wie beschrieben nur die aufrufenden Bausteine der Teilbereiche aufrufen. Diese rufen wiederum die Bausteine des jeweiligen Teilbereichs auf.

Bausteine, welche nur einen Typ Standardbausteine mehrfach aufrufen, sollen so gestaltet sein, dass die Nummer des Netzwerks gleichzeitig den Index des verwendeten Speicherbereichs widerspiegelt. Also in Netzwerk 13 beispielsweise, wird der AnalogInput-Baustein aufgerufen, dessen Multiinstanzdatenbaustein im Array den Index 13 trägt und dessen udtAnalogInputPlc-Struktur im entsprechenden globalen Datenbaustein ebenfalls den Index 13 trägt. So kann die Stelle der Zuordnung einfacher gefunden werden.



Abbildung 4-1: Standardbaustein-Aufruf

© hxql solutions GmbH

4.2 SCL-Bausteine

In SCL-Bausteinen sind REGIONs zur logischen Gliederung zu Verwenden. Ein zusätzlicher Vorteil dabei ist, dass man in der REGIONs-Übersicht schnell zur gewünschten Stelle im Code navigieren kann. Eingeleitet werden die Bausteine mit folgender REGION in der zum einen die Funktion beschrieben und zum anderen die Versionierung eingetragen wird. - Kann als Kopiervorlage verwendet werden.

```
REGION Block Information
    //================================================================================
    // hxql solutions GmbH
    //--------------------------------------------------------------------------------
    // Description:
    // [function description]
    // Parameters:
    // [parameter information]
    //--------------------------------------------------------------------------------
    // Change log table:
    // Version  Date            Author          Changes applied
    // 1.0.0    23.08.2025      MHeymann        first version
    //================================================================================
END_REGION
```

Aber auch allgemein soll im Code auf gute Strukturierung und Lesbarkeit geachtet werden. Das kann unter anderem durch folgende Punkte erreicht werden.

- Bei If-Anweisungen wird „THEN“ nicht hintenangestellt, sondern steht allein auf der nächsten Zeile nach den Bedingungen und trennt diese so, gut sichtbar, von den Anweisungen.
- Ebenfalls sollen die Bedingungen einer If-Anweisung besser untereinander stehen statt auf einer Zeile, welche häufig so lang wird, dass man horizontal scrollen muss und somit nicht alle Bedingungen auf einen Blick erfassen kann.
- Vergleiche sollen in Klammern gesetzt werden. So ist besser zu erkennen, dass und was verglichen wird und wo dieser Vergleich endet.
- Bei allen Vergleichen erst die Variable, dann der Wert, um der natürlichen Lesrichtung zu entsprechen (Yoda Conditions sind in SCL nicht notwendig).
- Komplexe Bedingungen, z.B. mehrere Vergleiche, in separate benannte Booleans extrahieren, somit hat jede Bedingung einen Namen und liest sich wie Prosa. Der Vorteil geht dabei über die Lesbarkeit hinaus. Jede Bedingung kann an anderer Stelle erneut abgefragt werden und ändert sich etwas an der Bedingung, muss sie nur an einer Stelle angepasst werden.
- Konstanten statt Magic Numbers. „Nackte“ Zahlen sollten stets durch sprechende Konstanten ersetzt werden. Im Variablenkommentar sollte hinterlegt werden, wo der Wert herkommt (z.B. Grenzwert aus technischer Dokumentation).

© hxql solutions GmbH

- Bei Case-Anweisungen soll geprüft werden, ob man die Lesbarkeit erhöhen kann, indem man die Zahlen durch Konstanten ersetzt. Zum Beispiel bei einer Betriebsartenumschaltung, könnten die Zahlen durch „MANUAL_MODE“, „STEP_MODE“ und „AUTOMATIC_MODE“ ersetzt werden.
- Es sollen nach Möglichkeit Compound Assignment Operators (Zusammengesetzte Zuweisungsoperatoren bzw. kombinierten Wertzuweisungen) verwendet werden. Die Kombinationen +=, -=, *= und /= sind in SCL verfügbar. Gerade bei längeren Variablenbezeichnungen ist die Benutzung dieser Operatoren entschieden übersichtlicher.

  Beispiel: Aus „`udtStruktur.iVariableA := udtStruktur.iVariableA + 1;`“

  wird: „`udtStruktur.iVariableA += 1;`“
- Variablen, die nach außen gegeben werden (bzw. auch außerhalb des Bausteins verwendet werden) sollen, so möglich und sinnvoll, nur einmal beschrieben werden. Ist die mehrfache Zuweisung notwendig, sollte dafür eine temporäre Variable verwendet werden, welche am Bausteinende der ursprünglichen Variable einmal ihren Wert zuweist. Das hat zusätzlich den Vorteil, dass ein Rücksetzen nicht vergessen werden kann. Wird die temporäre Variable nach dem anfänglichen Initiieren nicht beschrieben, so entspricht ihr Wert im Moment der Zuweisung dem Defaultwert ihres Datentyps.

4.3 Baustein-Kommentare

Der Programmcode soll so ausführlich wie nötig und so knapp wie möglich kommentiert werden. Der Fokus liegt dabei aber auf Nachvollziehbarkeit. Bei mehrzeiligen Kommentaren soll auf „(*“ und „*)“ verzichtet werden und jede Zeile einzeln als Kommentar mit „//“ eingeleitet werden.

4.4 Beispiel-Baustein

AnalogInput

	Name	Datentyp	Defaultwert	Remanenz	Erreichbar a...	Schrei...	Sichtbar i...	Einstellwert	Überwachung	Kommentar
1	Input									
2	iRawInput	Int	0	Nicht rema...						raw input from analog card
3	Output									
4	<Hinzufügen>									
5	InOut									
6	ioudtPlc	"udtAnalogInputPlc"								
7	Static									
8	xSimulationModeOld	Bool	false	Nicht rema...						edge detection memory of SimulationMode
9	Temp									
10	trActualValue	Real								actual value in engineering units
11	trHysteresis	Real								local alias; hysteresis for limit monitoring (absolut value)
12	trMeasRangeStart	Real								local alias; engineering value for minimum (e.g. 0.0 bar, 0.0 °C)
13	trMeasRangeEnd	Real								local alias; engineering value for maximum (e.g. 10.0 bar, 100.0 °C)
14	tiHwZero	Int								local alias; hardware raw value for minimum (typically 0)
15	tiHwMax	Int								local alias; hardware raw value for maximum (typically 27648 or 32767)
16	tiHwRange	Int								hardware raw value range
17	txWireBreak	Bool								wire break error flag
18	txMeasuringRangeFault	Bool								measuring range fault flag
19	txLimitPlausibilityError	Bool								limit configuration plausibility error
20	txCollectiveFault	Bool								collective fault flag
21	txLimitHH	Bool								limit HH flag
22	txLimitH	Bool								limit H flag
23	txLimitL	Bool								limit L flag
24	txLimitLL	Bool								limit LL flag
25	txSimActive	Bool								local alias; simulation mode on (=TRUE)
26	<Hinzufügen>									
27	Constant									
28	<Hinzufügen>									

Abbildung 4-2: Beispiel-Baustein Teil 1

© hxql solutions GmbH

```
REGION Block Information
    //=============================================================================
    // hxql solutions GmbH
    //-----------------------------------------------------------------------------
    // Description:
    // Function block for processing of an analog input.
    // Supports simulation mode and comprehensive monitoring.
    // Includes wire break detection, measuring range monitoring,
    // and four-level limit monitoring with hysteresis.
    // Compatible with most analog input cards (Siemens, Phoenix, Wago, etc.)
    //-----------------------------------------------------------------------------
    // Log table for changes:
    // Version  Date           resp. expert    Changes applied
    // 1.0.0    23.08.2025     MHeymann        first version
    //=============================================================================
END_REGION

```

Abbildung 4-3: Beispiel-Baustein Teil 2

```
REGION initialization and local aliases
    //Initialize temporary variables
    #txCollectiveFault := FALSE;
    #txWireBreak := FALSE;
    #txMeasuringRangeFault := FALSE;
    #txLimitPlausibilityError := FALSE;

    //local aliases for readability
    #trActualValue := #ioudtPlc.stCtrl.rGetActualValue;
    #trHysteresis := #ioudtPlc.stCtrl.rSetHysteresis;
    #txLimitHH := #ioudtPlc.stCtrl.xGetLimitHH;
    #txLimitH := #ioudtPlc.stCtrl.xGetLimitH;
    #txLimitL := #ioudtPlc.stCtrl.xGetLimitL;
    #txLimitLL := #ioudtPlc.stCtrl.xGetLimitLL;
    #txSimActive := #ioudtPlc.stCtrl.xSetSimulationMode;
    #trMeasRangeStart := #ioudtPlc.stPar.rMeasuringRangeStart;
    #trMeasRangeEnd := #ioudtPlc.stPar.rMeasuringRangeEnd;
    #tiHwZero := #ioudtPlc.stPar.iHwZeroValue;
    #tiHwMax := #ioudtPlc.stPar.iHwMaxValue;
    #tiHwRange := #tiHwMax - #tiHwZero;
END_REGION

```

Abbildung 4-4: Beispiel-Baustein Teil 3

```
REGION input selection and scaling
    IF #txSimActive THEN
        //SIMULATION MODE
        //On rising edge: capture current actual value as simulation seed
        IF NOT #xSimulationModeOld THEN
            #ioudtPlc.stCtrl.rSetSimulationValue := #trActualValue;
        END_IF;

        #trActualValue := #ioudtPlc.stCtrl.rSetSimulationValue;
    ELSE
        //AUTOMATIC MODE
        //Scale raw input to engineering value
        IF (#tiHwRange <> 0)
        THEN
            #trActualValue :=
            #trMeasRangeStart
            + (INT_TO_REAL(#iRawInput) - #tiHwZero)
            / INT_TO_REAL(#tiHwRange)
            * (#trMeasRangeEnd - #trMeasRangeStart);
        ELSE
            //Division by zero guard → fall back to range start
            #trActualValue := #trMeasRangeStart;
        END_IF;
    END_IF;

    //Edge detection memory
    #xSimulationModeOld := #txSimActive;
END_REGION

```

Abbildung 4-5: Beispiel-Baustein Teil 4

```
REGION wire break monitoring
    //Wire break detection (only if monitoring enabled and not in simulation)
    IF #ioudtPlc.stPar.stMon.xMonWireBreak
        AND NOT #txSimActive
    THEN
        //Wire break typically detected when raw value is below minimum (e.g. < 3277 for 4-20mA)
        IF (#iRawInput < #ioudtPlc.stPar.iHwZeroValue)
        THEN
            #txWireBreak := TRUE;
        END_IF;
    END_IF;
END_REGION

```

Abbildung 4-6: Beispiel-Baustein Teil 5

© hxql solutions GmbH

```
84 REGION measuring range monitoring
85     //Measuring range fault detection (only if monitoring enabled and not in simulation)
86     IF #ioudtPlc.stPar.stMon.xMonMeasuringRange
87         AND NOT #ioudtPlc.stCtrl.xSetSimulationMode
88     THEN
89         //Check if raw input exceeds hardware limits
90         IF (#iRawInput < #ioudtPlc.stPar.iHwZeroValue)
91             OR (#iRawInput > #ioudtPlc.stPar.iHwMaxValue)
92         THEN
93             #txMeasuringRangeFault := TRUE;
94         END_IF;
95     END_IF;
96 END_REGION
97
```

Abbildung 4-7: Beispiel-Baustein Teil 6

```
98 REGION limit plausibility check
99     REGION hierarchy check OF enabled limits
100        IF #ioudtPlc.stPar.stMon.xMonLimitLL
101            AND #ioudtPlc.stPar.stMon.xMonLimitL
102            AND (#ioudtPlc.stCtrl.rSetLimitLL > #ioudtPlc.stCtrl.rSetLimitL)
103        THEN
104            #txLimitPlausibilityError := TRUE;
105        END_IF;
106
107        IF #ioudtPlc.stPar.stMon.xMonLimitLL
108            AND #ioudtPlc.stPar.stMon.xMonLimitH
109            AND (#ioudtPlc.stCtrl.rSetLimitLL > #ioudtPlc.stCtrl.rSetLimitH)
110        THEN
111            #txLimitPlausibilityError := TRUE;
112        END_IF;
113
114        IF #ioudtPlc.stPar.stMon.xMonLimitLL
115            AND #ioudtPlc.stPar.stMon.xMonLimitHH
116            AND (#ioudtPlc.stCtrl.rSetLimitLL > #ioudtPlc.stCtrl.rSetLimitHH)
117        THEN
118            #txLimitPlausibilityError := TRUE;
119        END_IF;
120
121        IF #ioudtPlc.stPar.stMon.xMonLimitL
122            AND #ioudtPlc.stPar.stMon.xMonLimitH
123            AND (#ioudtPlc.stCtrl.rSetLimitL > #ioudtPlc.stCtrl.rSetLimitH)
124        THEN
125            #txLimitPlausibilityError := TRUE;
126        END_IF;
127
128        IF #ioudtPlc.stPar.stMon.xMonLimitL
129            AND #ioudtPlc.stPar.stMon.xMonLimitHH
130            AND (#ioudtPlc.stCtrl.rSetLimitL > #ioudtPlc.stCtrl.rSetLimitHH)
131        THEN
132            #txLimitPlausibilityError := TRUE;
133        END_IF;
134
135        IF #ioudtPlc.stPar.stMon.xMonLimitH
136            AND #ioudtPlc.stPar.stMon.xMonLimitHH
137            AND (#ioudtPlc.stCtrl.rSetLimitH > #ioudtPlc.stCtrl.rSetLimitHH)
138        THEN
139            #txLimitPlausibilityError := TRUE;
140        END_IF;
141    END_REGION
142
```

Abbildung 4-8: Beispiel-Baustein Teil 7

```
143    REGION check if enabled limits are within measuring range
144        IF #ioudtPlc.stPar.stMon.xMonLimitHH
145            AND (#ioudtPlc.stCtrl.rSetLimitHH > #ioudtPlc.stPar.rMeasuringRangeEnd)
146        THEN
147            #txLimitPlausibilityError := TRUE;
148        END_IF;
149
150        IF #ioudtPlc.stPar.stMon.xMonLimitH
151            AND (#ioudtPlc.stCtrl.rSetLimitH > #ioudtPlc.stPar.rMeasuringRangeEnd)
152        THEN
153            #txLimitPlausibilityError := TRUE;
154        END_IF;
155
156        IF #ioudtPlc.stPar.stMon.xMonLimitL
157            AND (#ioudtPlc.stCtrl.rSetLimitL < #ioudtPlc.stPar.rMeasuringRangeStart)
158        THEN
159            #txLimitPlausibilityError := TRUE;
160        END_IF;
161
162        IF #ioudtPlc.stPar.stMon.xMonLimitLL
163            AND (#ioudtPlc.stCtrl.rSetLimitLL < #ioudtPlc.stPar.rMeasuringRangeStart)
164        THEN
165            #txLimitPlausibilityError := TRUE;
166        END_IF;
167    END_REGION
168 END_REGION
169
```

Abbildung 4-9: Beispiel-Baustein Teil 8

© hxql solutions GmbH

```
170 REGION limit monitoring with hysteresis
171     //Validate and correct hysteresis if negative
172     IF (#trHysteresis < 0.0)
173     THEN
174         #ioudtPlc.stCtrl.rSetHysteresis := 0.0;
175         #trHysteresis := 0.0;
176     END_IF;
177
178     REGION Limit HH
179         IF #ioudtPlc.stPar.stMon.xMonLimitHH
180         THEN
181             //Set alarm when limit exceeded
182             IF (#trActualValue >= #ioudtPlc.stCtrl.rSetLimitHH)
183             THEN
184                 #txLimitHH := TRUE;
185             //Reset alarm when value returns with hysteresis
186             ELSIF (#trActualValue < (#ioudtPlc.stCtrl.rSetLimitHH - #trHysteresis))
187             THEN
188                 #txLimitHH := FALSE;
189             END_IF;
190         ELSE
191             #txLimitHH := FALSE;
192         END_IF;
193     END_REGION
194
```

Abbildung 4-10: Beispiel-Baustein Teil 9

```
195     REGION Limit H
196         IF #ioudtPlc.stPar.stMon.xMonLimitH
197         THEN
198             //Set alarm when limit exceeded
199             IF (#trActualValue >= #ioudtPlc.stCtrl.rSetLimitH)
200             THEN
201                 #txLimitH := TRUE;
202             //Reset alarm when value returns with hysteresis
203             ELSIF (#trActualValue < (#ioudtPlc.stCtrl.rSetLimitH - #trHysteresis))
204             THEN
205                 #txLimitH := FALSE;
206             END_IF;
207         ELSE
208             #txLimitH := FALSE;
209         END_IF;
210     END_REGION
211
```

Abbildung 4-11: Beispiel-Baustein Teil 10

```
212     REGION Limit L
213         IF #ioudtPlc.stPar.stMon.xMonLimitL
214         THEN
215             //Set alarm when limit exceeded
216             IF (#trActualValue <= #ioudtPlc.stCtrl.rSetLimitL)
217             THEN
218                 #txLimitL := TRUE;
219             //Reset alarm when value returns with hysteresis
220             ELSIF (#trActualValue > (#ioudtPlc.stCtrl.rSetLimitL + #trHysteresis))
221             THEN
222                 #txLimitL := FALSE;
223             END_IF;
224         ELSE
225             #txLimitL := FALSE;
226         END_IF;
227     END_REGION
228
```

Abbildung 4-12: Beispiel-Baustein Teil 11

```
229     REGION Limit LL
230         IF #ioudtPlc.stPar.stMon.xMonLimitLL
231         THEN
232             //Set alarm when limit exceeded
233             IF (#trActualValue <= #ioudtPlc.stCtrl.rSetLimitLL)
234             THEN
235                 #txLimitLL := TRUE;
236             //Reset alarm when value returns with hysteresis
237             ELSIF (#trActualValue > (#ioudtPlc.stCtrl.rSetLimitLL + #trHysteresis))
238             THEN
239                 #txLimitLL := FALSE;
240             END_IF;
241         ELSE
242             #txLimitLL := FALSE;
243         END_IF;
244     END_REGION
245 END_REGION
246
```

Abbildung 4-13: Beispiel-Baustein Teil 12

© hxql solutions GmbH

```
247 REGION collective fault
248     #txCollectiveFault := #txWireBreak OR #txMeasuringRangeFault OR #txLimitPlausibilityError;
249 END_REGION
250
251 REGION status word generation
252     #ioudtPlc.stCtrl.wGetStatus.%X0 := #txCollectiveFault;       //Bit 0: collective fault
253     #ioudtPlc.stCtrl.wGetStatus.%X1 := #txWireBreak;             //Bit 1: wire break
254     #ioudtPlc.stCtrl.wGetStatus.%X2 := #txMeasuringRangeFault;   //Bit 2: measuring range fault
255     #ioudtPlc.stCtrl.wGetStatus.%X3 := #ioudtPlc.stCtrl.xSetSimulationMode;  //Bit 3: simulation active
256     #ioudtPlc.stCtrl.wGetStatus.%X4 := #txLimitHH;               //Bit 4: limit HH
257     #ioudtPlc.stCtrl.wGetStatus.%X5 := #txLimitH;                //Bit 5: limit H
258     #ioudtPlc.stCtrl.wGetStatus.%X6 := #txLimitL;                //Bit 6: limit L
259     #ioudtPlc.stCtrl.wGetStatus.%X7 := #txLimitLL;               //Bit 7: limit LL
260     #ioudtPlc.stCtrl.wGetStatus.%X8 := #txLimitPlausibilityError; //Bit 8: limit plausibility error
261     // Bits 9-15: reserved for future use
262 END_REGION
263
```

Abbildung 4-14: Beispiel-Baustein Teil 13

```
264 REGION write plc variables
265     #ioudtPlc.stCtrl.rGetActualValue := #trActualValue;
266     #ioudtPlc.stCtrl.xGetCollectiveFault := #txCollectiveFault;
267     #ioudtPlc.stCtrl.xGetWireBreak := #txWireBreak;
268     #ioudtPlc.stCtrl.xGetMeasuringRangeFault := #txMeasuringRangeFault;
269     #ioudtPlc.stCtrl.xGetLimitHH := #txLimitHH;
270     #ioudtPlc.stCtrl.xGetLimitH := #txLimitH;
271     #ioudtPlc.stCtrl.xGetLimitL := #txLimitL;
272     #ioudtPlc.stCtrl.xGetLimitLL := #txLimitLL;
273 END_REGION
274
275
276
```

Abbildung 4-15: Beispiel-Baustein Teil 14

© hxql solutions GmbH

5 Namenskonventionen für Bausteine

Die Bezeichnungen der Bausteine sollen die Funktion klar kommunizieren.

Alle Bausteine werden im upper CamelCase benannt.

Für Funktionen und Funktionsbausteine wird das Präfix „Call“ verwendet, wenn die Hauptaufgabe der Bausteine der Aufruf anderer Bausteine ist. Ansonsten bekommen sie kein Präfix, da diese Bausteintypen aus der Symbolik und der automatischen Beschriftung im TIA-Portal eindeutig erkenntlich sind.

Globale Datenbausteine werden mit dem Präfix „Gdb“ gekennzeichnet, Instanzdatenbausteine von Funktionsbausteinen mit „Idb“ bei Einzelinstanzen und mit „minst“ bei Multiinstanzen (siehe Punkt 6.2.2).

© hxql solutions GmbH

6 Namenskonventionen für Variablen

6.1 Allgemein

Die Bezeichnungen sollen, ebenso wie die der Bausteine, die Funktion oder den Inhalt der Variable klar kommunizieren. Das Präfix eingeschlossen werden Programm-Variablen immer im lower CamelCase und Hardware-Variablen im upper CamelCase benannt. Die einzige Ausnahme bilden Konstanten, welche ausschließlich mit Großbuchstaben geschrieben werden, wobei die Einzelworte durch Unterstriche getrennt werden. Allen anderen Variablen wird außerdem ein zweiteiliges Präfix vorangestellt, welcher Aufschluss über die Funktion im Baustein bzw. im Programm, sowie den Datentyp der Variable gibt.

6.2 Präfix

6.2.1 Teil 1: Funktion

Tabelle 6-1: Präfix Teil 1

Präfix	Funktion
i	Bausteineingang (Input)
o	Bausteinausgang (Output)
io	Baustein-Ein-/Ausgang (InOut)
t	Temporäre Variable (Temp)
Hi	Hardware-Eingang
Ho	Hardware-Ausgang

Statische Variablen, weder in Funktionsbausteinen noch in Datenbausteinen, bekommen hier kein gesondertes Präfix.

6.2.2 Teil 2: Datentyp

Tabelle 6-2: Präfix Teil 2

Präfix	Datentyp
x	Bool
b	Byte
I	Int
si	SInt
di	DInt

© hxql solutions GmbH

Präfix	Datentyp
li	LInt
ui	UInt
usi	USInt
udi	UDInt
uli	ULInt
w	Word
dw	DWord
lw	LWord
r	Real
lr	LReal
str	String
wstr	WString
c	Char
wc	WChar
tim	Time
ltim	LTime
tod	Time_Of_Day
ltod	LTime_Of_Day
d	Date
dt	Date_And_Time
ldt	LDT
dtl	DTL
v	Variant
a	Array; immer gefolgt vom Datentyp der Array-Elemente z.B.: audt
udt	Benutzerdefinierter Datentyp/PLC-Datentyp
minst	Speicherbereich eines Multiinstanzdatenbausteins (siehe Punkt 5)
Hwid	Hardware-Identifikation

© hxql solutions GmbH

6.3 Festgelegte Abkürzungen

Tabelle 6-3: Festgelegte Abkürzungen

Abkürzung	Begriff
Ackn	Acknowledge
Act	Actual
Avg	Average
Bwd	Backward
Com	Communication
Ctrl	Control
Cnt	Counter
Diff	Difference
Dir	Direction
Err	Error
FEdge	Falling Edge
Fdbk	Feedback
Fwd	Forward
Get	Getter
Hw	Hardware
Max	Maximum
Min	Minimum
Mon	Monitoring
Neg	Negative
Num	Number
Op	Operation
Par	Parameter
Pstn	Position
Pos	Positive
Prev	Previous
Req	Request
REdge	Rising Edge

© hxql solutions GmbH

Abkürzung	Begriff
Set	Setter
Sim	Simulation
Val	Value
Vel	Velocity
Warn	Warning

6.4 Variablen-Kommentare

Variablen sind in der Kommentarspalte unmissverständlich zu erklären. Bei benutzerdefinierten Datentypen sind die jeweiligen Elemente ebenfalls zu kommentieren.

6.5 Beispiele für Variablennamen

Tabelle 6-4: Beispiele für Variablennamen

Variable	Funktion	Datentyp	Beschreibung
uiCntBaggages	statische Variable	UInt	Gepäckstück-Zähler
iuiCycleTime	Baustein-Eingang	UInt	Zykluszeit
trHwValuePos	temporäre Variable	Real	positiver Wert der Hardware
HixBG23SafetyLsExit	Hardware-Eingang	Bool	Sicherheitslichtschrankensignal
ioudtDigitalInputPlc	InOut-Variable	udt	Plc-Datenstruktur eines Digitaleingangs
aminstDigitalOutput	statische Variable	Array von Multiinstanzen Digitalausgangs-FBs	
MAX_VALUE	Konstante	unbekannt	Höchstwert

© hxql solutions GmbH

7 Verzeichnisse

7.1 Abbildungsverzeichnis

Abbildung 3-1: Datenaustausch mit Sicherheitsprogramm 7
Abbildung 4-1: Standardbaustein-Aufruf 8
Abbildung 4-2: Beispiel-Baustein Teil 1 10
Abbildung 4-3: Beispiel-Baustein Teil 2 11
Abbildung 4-4: Beispiel-Baustein Teil 3 11
Abbildung 4-5: Beispiel-Baustein Teil 4 11
Abbildung 4-6: Beispiel-Baustein Teil 5 11
Abbildung 4-7: Beispiel-Baustein Teil 6 12
Abbildung 4-8: Beispiel-Baustein Teil 7 12
Abbildung 4-9: Beispiel-Baustein Teil 8 12
Abbildung 4-10: Beispiel-Baustein Teil 9 13
Abbildung 4-11: Beispiel-Baustein Teil 10 13
Abbildung 4-12: Beispiel-Baustein Teil 11 13
Abbildung 4-13: Beispiel-Baustein Teil 12 13
Abbildung 4-14: Beispiel-Baustein Teil 13 14
Abbildung 4-15: Beispiel-Baustein Teil 14 14

7.2 Tabellenverzeichnis

Tabelle 6-1: Präfix Teil 1 16
Tabelle 6-2: Präfix Teil 2 16
Tabelle 6-3: Festgelegte Abkürzungen 18
Tabelle 6-4: Beispiele für Variablennamen 19

© hxql solutions GmbH

8 Dokumenteninformation

Änderungshistorie

Version	Datum	Autor	Änderungen
V1.0.0	17.09.2025	M. Heymann	Erste Fassung
V1.1.0	07.01.2026	M. Heymann	Dokument auf hxql-Vorlage übertragen

© hxql solutions GmbH