SIEMENS

Wskazówki programowania " Styleguide"• 10/2016

Wskazówki programowania sterowników S7-1200/S7-1500

TIA Portal

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

Gwarancja i zrzeczenie się odpwiedzialności

Ważne

Informacje zawarte w niniejszym podręczniku mogą być niekompletne pod względem konfiguracji, wyposażenia jak również innych opcji. Należy je stosować wyłącznie do rozwiązań standardowych.
Odpowiedzialność za poprawną eksploatację opisanych produktów leży po stronie użytkownika. Poniższe przykłady nie zwalniają użytkownika z obowiązku bezpiecznego obchodzenia się ze sprzętem podczas instalacji, eksploatacji i konserwacji.
Firma Siemens nie może być pociągnięta do odpowiedzialności za ewentualne szkody wykraczające poza powyższą regulację.
Zastrzegamy sobie prawo do wprowadzenia zmian do zamieszczonych przykładów bez uprzedzenia. W przypadku różnic pomiędzy przykładami zamieszczonymi w niniejszym podręczniku, a innymi publikacjami, należy postępować zgodnie z informacjami zamieszczonymi w tychże publikacjach.

Zrzekamy się odpowiedzialności z tytułu informacji zawartych w niniejszym podręczniku.
Nasza odpowiedzialność, niezależnie od podstawy prawnej za szkody spowodowane korzystaniem z opisanych niżej przykładów, informacji, programów, danych projektowych itp. jest wykluczona, o ile nie występuje przypadek odpowiedzialności przymusowej np. zgodnie z ustawą o odpowiedzialności za produkty w przypadkach działania umyślnego, rażącej niedbałości, uszkodzenia życia lub zdrowia, albo z powodu przejęcia gwarancji za cechy i stan jakiegoś produktu, zatajenia jego wady lub naruszenia istotnych postanowień wynikających z umowy. Odszkodowanie z tytułu naruszenia kluczowych zobowiązań wynikających z umowy ograniczają się jednak do typowych dla umów, przewidywalnych szkód, o ile nie dotyczą odpowiedzialności przymusowej z tytułu działania umyślnego, rażącej niedbałości, lub uszkodzenia życia lub zdrowia.
Nie wiąże się to ze zmianą ciężaru dowodowego na Państwa niekorzyść.

Wzskazówki dotyczące bezpieczeństwa

Produkty firmy Siemens są opracowywane z myślą o bezpiecznym użytkowaniu w środowisku przemysłowym. Poszczególne elementy np. urządzenia, sieci oraz inne oferowane przez nas rozwiązania stanowią część całościowego systemu bezpieczeństwa przemysłowego. Mając to na uwadze, staramy się ciągle ulepszać nasze produkty. Warto regularnie odwiedzać naszą stronę internetową i być na bieżąco z najnowszymi aktualizacjami.

Aby zagwarantować bezpieczne użytkowanie naszych produktów należy podjąć odpowiednie środki ochronne i zintegrować wszystkie urządzenia we wspólnym systemie bezpieczeństwa przemysłowego. Dotyczy to również produktów innych producentów.
Więcej informacji można znaleźć na stornie internetowej:
http://www.siemens.com/industrialsecurity.

Jeżeli chcesz być na bieżąco z najnowszymi aktualizacjami, zapisz się do newslettera produktu, który Cię interesuje.
Więcej informacji można znaleźc na stronie:
http://www.siemens.com/industrialsecurity

Spis treści

1 Wstęp

Programując sterowniki SIMATIC, programista ma za zadanie stworzyć przejrzysty program. Aczkolwiek, każdy użytkownik podchodzi do tego zadania inaczej, wprowadzajac własny sposób nazywania zmiennych, blokow czy też komentowania. W rezultacie powstaje kod źródłowy, który może zostać rozszyfrowany jedynie przez autora.

Zalety przejrzystego stylu programowania

Jeżeli jeden program opracowywany jest przez kilku programistów, zaleca się trzymanie tego samego stylu programowania:

- Jednolity styl programowania
- Czytelny i zrozumiały kod
- Proste serwisowanie
- Szybkie znajdywanie i naprawianie błędów
- Wydajna praca nad projektem

Przeznaczenie podręcznika

Uwaga Wskazówki zawarte w ninijeszym podręczniku należy traktować na zasadzie podpowiedzi. Ostateczna decyzja odnośnie implementacji proponowanych rozwiązań leży po stronie użytkownika.

Stosując się do wskazówek zawartych w niniejszym podręczniku, programista będzie w stanie stworzyć jednolity oraz czytelny kod, który znacznie przyspieszy wykrycie błędów w programie. Aczkolwiek, pewne efekty wizualne, np. stała ilość odstępów przed przecinike, niewiele wnoszą do jakości oprpgramowania. O wiele bardziej ważne jest ustalenie zasad, które bedą wspierały progamistę w następujących kwestiach:

- Uniknięcie literówek, które prowadzą do błędów kompilatora.
  Cel: Kompilator będzie w stanie wykryć więcej błędów.
- Używanie kodu do diagnozowania błędów w programie: np. użycie zmiennej temp dla kilku cykli.
  Cel: Wcześniejsza detekcja błędów
- Jednolite aplikacje i biblioteki
  Cel: Możliwość ponownego wykorzystania kodu
- Proste serwisowanie i rozbudowa
  Cel: Modyfikacja kodu danego modułu, który zawiera: Funkcje (FC_, bloki funkcyjne (FB), bloki danych (DB), bloki organizacyjne (OB) w bibliotekach lub w projekcie będzie miała minimalny wpływ na program / bibliotekę. Modyfikacja kodu w poszczególnych modułach będzie przeprowadzana przez innych programistów.

Ważność dokumentu

Niniejszy dokument dotyczy przykładowych aplikacji, takich jak biblioteki stworzone wg. IEC 1131-3 (DIN EN 61131-3) w językach ST, LAD, FFB oraz STL.

Tematy nieporuszane w niniejszym podręczniku

Niniejszy podręcznik nie zawiera informacji na temat:

- Programowania w STEP 7
- Uruchamiania sterowników SIMATIC

2 Terminologia

Reguły / zalecenia

W niniejszym podręczniku można znaleźć dwa rodzaje informacji:

Reguły - informacje wiążące, których należy przestrzegać aby zagwarantować poprawną realizację programu. W wyjątkowych przypadkach reguły można złamać, jednak wymaga to odpowiedniego udokumentowania.

Zalecenia - informacje pomocnicze, dzięki którym kod jest jednolity oraz bardziej przejrzysty. W niektórych przypadkach, zalecenia można pominąć np. z uwagi na wydajność programu.

Wydajność (performance)

Wydajność systemu automatyki oznacza czas przetworzenia (czas cyklu) programu.

Spadek wydajności (performance loss) oznacza zredukowanie czasu przetworzenia programu poprzez zastosowanie odpowiednich reguł programistycznych i umiejętne zaprogramowanie programu użytkownika.

Identyfikator / nazwa

Ważne jest aby nie pomylić tych terminów. Nazwa stanowi część identyfikatora. Natomiast na identyfikator składają się następujące elementy:

- Przedrostek
- Nazwa
- Końcówka

Skróty

W poniższej tabeli znajdują się skróty wykorzystywane w niniejszej dokumentacji.

Tabela 2-1: Skróty

Skrót	Rozwinięcie
OB	Blok organizacyjny
FB	Blok funkyjny
FC	Funkcja
DB	Blok danych
TO	Obiekt technologiczny
SFB	Systemowy blok funkcyjny
SFC	Funkcja systemowa

Zmienne (tagi) i parametry

Pojęcia związane ze zmiennymi, blokami funkcyjnymi oraz funkcjami wielokrotnie używane są niepoprawnie, oraz w sposób niespójny. Poniższa tabela pozwoli ujednolicić tą terminologię.

Rysunek 2-1: Zmienne (tagi) i parametry

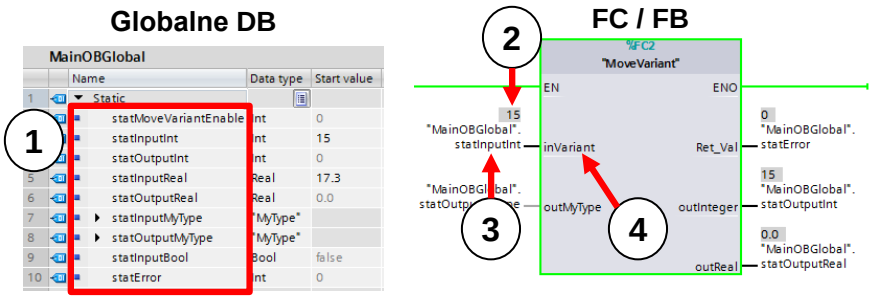


Tabela 2-2: Pojęcia związane ze zmiennymi i parametrami

	Pojęcie	Opis
1.	Zmienna (tag)	Zmienne to obszary pamięci sterownika zarezerwowane dla określonej wartości. Określane za pomocą rodzaju danych np. Bool, Integer, itp.: • zmienne PLC • pojedyncze zmienne w blokach danych • całe bloki danych
2.	Wartość zmiennych (tag value)	Są to wartości przechowywane w danej zmiennej (np. 15 jako wartość zmiennej Integer).
3.	Parametr aktualny	Są to zmienne powiązane ze sobą na poziomie interfejsów poleceń, funkcji oraz bloków funkcyjnych.
4.	Parametr formalny (parametr transferu, parametr blokowy)	Parametry formalne to parametry interfejsów dla poleceń, funkcji oraz bloków funkcyjnych (Input, Output, InOut, Temp, Static, oraz Return).

3 Specyfikacje

Nazwy używane w niniejszym podręczniku są zawsze unikalne pod względem funkcji oraz rodzaju interfejsu.

Oznacza to, że jeżeli dana nazwa pojawia się wielokrotnie, to powinna zawsze odnosić się do tej samej funkcjonalności.

Podstawę niniejszego dokumentu stanowi Podręcznik Programowania dla S7-1200/S7-1500, w którym można znaleźć informacje na temat poprawnego programowania sterowników S7-1200 oraz S7-1500.

Wskazówka Podręcznik programownia dlaS7-1200/S7-1500 jest dostępny na stronie:

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

3.1 Wymagania klienta

Reguła: dokumentowanie złamania reguły

Za każdym razem, kiedy reguła zostanie złamana, należy to udokumentować w kodzie programu.

W przypadku projektów na zlecenie, wymagania klienta końcowego mają najwyższy priorytet. Reguły zdefiniowane przez klienta należy udokumentować w odpowiednim formacie w tekście źródłowym.

3.2 Ustawienia w TIA Portal

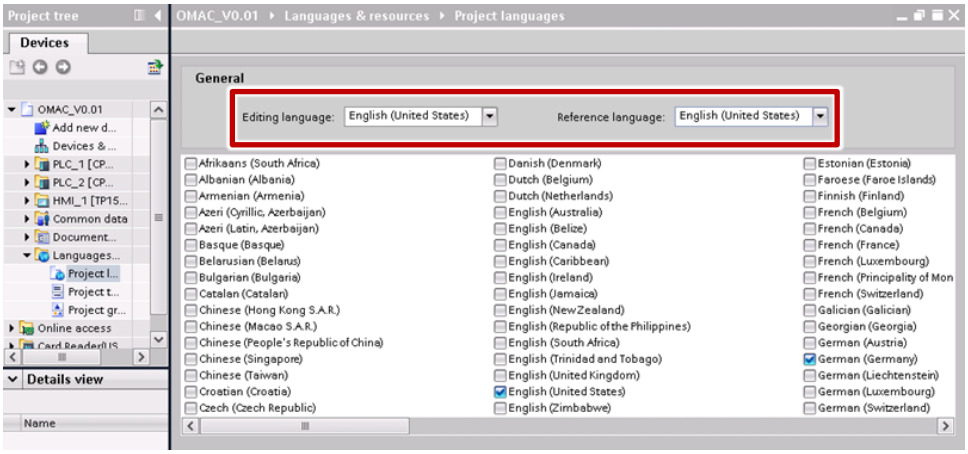
Reguła: Jeden język

Programując sterowniki PLC oraz panele HMI należy posługiwać się tym samym językiem. W obrębie danego projektu nie należy mieszać języków (np. używać j. angielskiego do edycji, j. niemieciego do komentarzy w blokach, czy też j. francuskiego do wyświetlanych tekstów.

Reguła: j. angielski - English (United States) do edycji oraz odnośników

Do edycji / opisywania kodu oraz komentarzy mależy posługiwać się j. angielskim “English (United States)”. Jedynym wyjątkiem od tej reguły jest wyraźna prośba klienta.

Rysunek 3-1: Ustawianie języka do edycji i odnośników



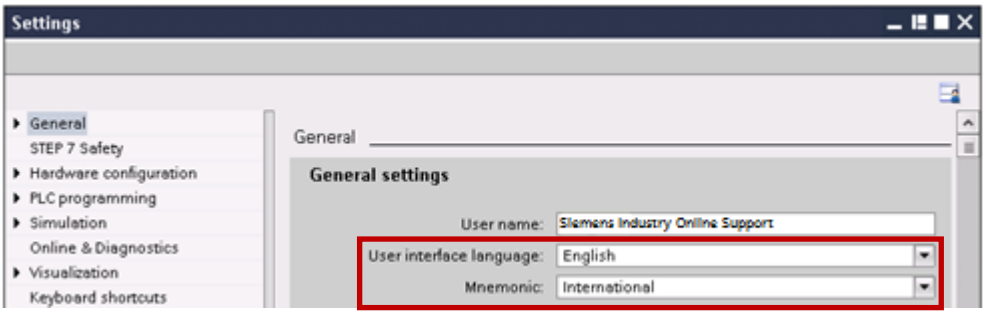
Zalecenie: j.angielski - English (United States) jako język interfejsu użytkownika

Interfejs użytkownika w TIA Portal powinien być ustawiony w języku angielskim. Wtenczas, dla wszystkich nowo-utworzonych projektów domyślnym językiem edycji będzie język angielski. Jeżeli np. ustawimy j. niemiecki jako język HMI, wszystkie nowe projekty będą utworzone w języku niemieckim, a język niemiecki będzie językiem odniesienia.

Zasada: Mnemonic: International

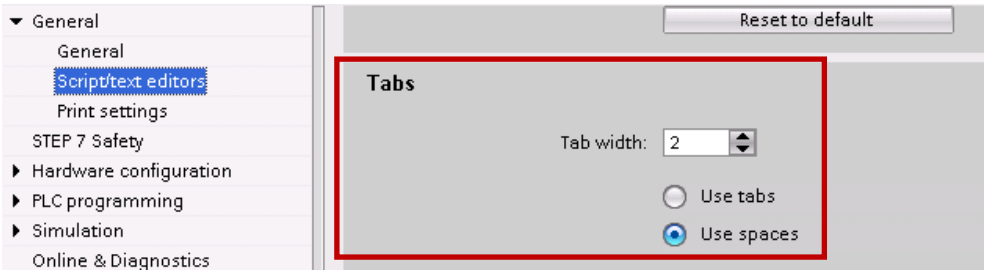
Ustawienie dla języków programowania "mnemonim" należy ustawić na międzynarodowy *International*.

Rysunek 3-2: Ustawienia języka w TIA Portal



Zasada: Wcięcia: 2 spacje

Używanie tabulatora w edytorach tekstu jest zabronione. Wcięcia należy robić za pomocą dwóch spacji. Należy dokonać poniższych ustawień w TIA Portal.



3.3 Identyfikator

3.3.1 Formatowanie

Reguła: angielski identyfikator

Nazwy wykorzystane w identyfikatorach (bloki, zmienne itp.) muszą być w języku angielskim (*English – United States*).

Reguła: identyfikatory unikalne

Nie należy używać identyfikatorów różniących się jedynie wielką / mała literą. Znaczenie identyfikatora pozostaje bez zmian dla wszystkich bloków i innych źródeł.

Reguła: identyfikatory w notacji camelCase

Jeżeli nie podano żadnej reguły dot. notacji identyfikatora, to identyfikator ten należy zapisać w notacji cameCase.

Zasady zapisu w notacji camelCase:

- pierwsza litera jest małą lietrą
- brak odstępów / separatorów (takich jak podkreślnik czy pauza)
- jeżeli identyfikator składa się z kilku słów, wtenczas każde nowe słowo zaczyna się od wielkiej litery

Zalecenie: długość identyfikatora: maksymalnie 24 znaki

Identyfikatory zmiennych, stałych oraz bloków nie powinny być dłuższe niż 24 znaki.

Reguła: nie używać znaków specjalnych

Nie należy używać spacji oraz znaków specjalnych dla danego języka takich jak: ä, ö, ü, à, itp.

Nie należy również używać znaków specjalnych pomiędzy prefiksem a identyfikatorem.

Przykład

Zmienna tymczasowa: tempMaxLength

Reguła: konstrukcja identyfikatora

W przypadku identyfikatorów, które składają się z kilku słów, sekwencja tych słów powinna być przypominać język mówiony.

3.3.2 Skróty

Zalecenie: dozwolone skróty

Aby zwiększyć przejrzystość programu, oraz zostawić więcej miejsca dla nazwy zmiennej, należy stosować ustandaryzowanych skrótów z tabeli poniżej.

Skrót	Rozwinięcie
Min	Minimum
Max	Maksimum
Act	Faktyczny, bieżący
Next	Następny
Prev	Poprzedni
Avg	Średnia
Diff	Różnica
Pos	Pozycja
Ris	Zbocze narastające
Fal	Zbocze opadające
Sim	Symulowane
Sum	Suma
Old	Stara wartość (np. dot. wykrycia zbocza)
Dir	Kierunek
Err	Błąd
Warn	Ostrzeżenie
Cmd	Polecenie

Zalecenie: Tylko jeden skrót w identyfikatorze

Nie zaleca się stosowania wielu skrótów jeden po drugim.

4 Programowanie sterowników PLC

4.1 Bloki programowe i kod źródłowy

4.1.1 Nazwy bloków i numeracja

Zalecenie: Krótkie nazwy bloków

Nazwy bloków powinny być krótkie, oraz nie zawierać informacji na temat pełnionych przez nie funkcji.

Reguła: Identyfikator musi rozpoczynać się od wielkiej litery

Identyfikator bloku musi rozpoczynać się od wielkiej litery.
Pozwoli to utrzymać spójne nazewnictwo bloków w TIA Portal.

Reguła: Prefiks "inst" / "Inst" dla instancji

Nazwa instancji powinna być poprzedzona prefiksem "inst" / "Inst".
Reguła ta dotyczy zarówno instancji pojedynczych jak i multi-instancji.

Przykład

Instancja pojedyncza:	InstHeater (wielka litera → własny blok)
Multi-instancja	instTimerMotor (mała litera → w obrębie jednej instancji)

Zalecenie: autonumeracja bloków

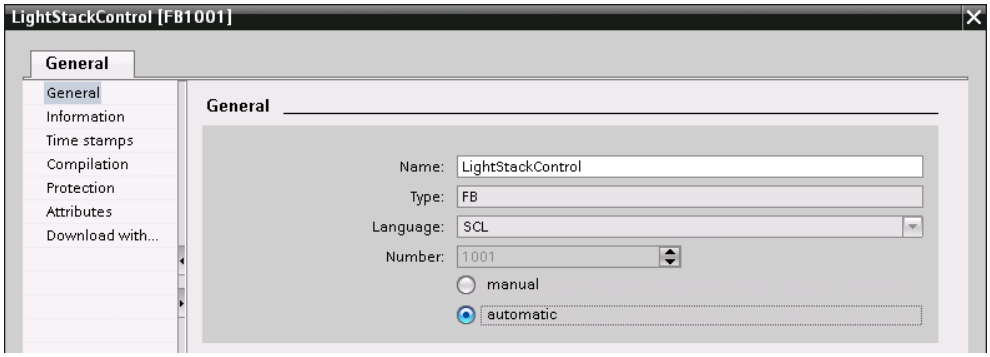
Bloki tego samego typu numerowane są automatycznie.

Instnieje możliwość ręcznej numeracji bloków. Patrz tabela poniżej.

Tabela 4-1: Procedura ręcznego numerowania bloków

Nr.	Procedura
1.	Zmień tryb numeracji na ręczny (*manual*)
2.	Nadaj blokowi numer (np. 1001)
3.	Zmień tryb numeracji na automatyczny (*automatic*)
4.	Potwierdź klikając *OK*

Rysunek 4-1: Zmiana trybu numeracji dla danego bloku



4.1.2 Formatowanie

Zalecenie: Długość linii w edytorze nie może przekraczać 80 znaków

Długość linii kodu źródłowego nie może przekraczać 80 znaków. W przeciwnym razie, kod może być nieczytelny.

4.1.3 Programowanie

Reguła: Nie należy edytować kodu źródłowego

Aby w pełni wykorzystać potencjał platformy TIA Portal, w tym funkcje autouzupełniania oraz naprawiania błędów, należy pracować głównie na blokach. Odradza się edytowanie kodu źródłowego w zewnętrznym edytorze oraz importowanie edytowanego kodu do programu użytkownika.

Zalecenie: Używanie języka SCL

Zaleca się korzystanie z języka SCL, jako głównego języka programowania bloków. SCL jest najbardziej przejrzysty z języków programowania, a przy tym nie obciąża tak sterowników SIMATIC PLC.

Jeżeli pojedyncze bloki mają być ze sobą połączone (np. w OB), należy skorzystać z języków LAD lub FBD, nawet jeżeli blok składa się głównie z binarnych operacji logicznych. Ułatwi to serwisantom przeprowadzenie diagnostyki.

Reguła: Multi-instancje zamiast instancji pojedynczych

Zaleca się stosowanie multi-instancji w miejsce instancji pojedynczych. Umożliwiają one tworzenie funkcji zamkniętych takich jak np. blok FB z wbudowanych timerem.

Reguła: Przechowywanie bloków danych w pamięci ładowania "load memory"

Bloki danych należy zawsze przechowywać w pamięci RAM. Poza pewnymi wyjątkami (np. przechowywanie danych pomiarowych oraz receptur), nie należy używać pamięci ładowania "load memory" do przechowywania bloków danych.

Reguła: Używanie zmiennych w obrębie danego bloku

Zmienne mogą być używane tylko lokalnie. W obrębie bloków danych dostęp do danych globalnych jest niedozwolony. Reguła ta dotyczy:

- Dostępu do globalnych DB oraz używania bloków instancji pojedynczych
- Dostępu do zmiennych (tablice zmiennych)

Zalecenie: Unikanie stałych globalnych

Nie należy stosować stałych globalnych. W przeciwnym razie blok nie będzie mógł być wykorzystany modułowo.

Reguła: Nie należy deklarować ważnych zmiennych testowych jako temp

Aby ułatwić proces testowania funkcji (FC) oraz bloków (FB) należy zwrócić uwagę na obserwowalność zmiennych w tablicach *watch table* oraz *force table.*

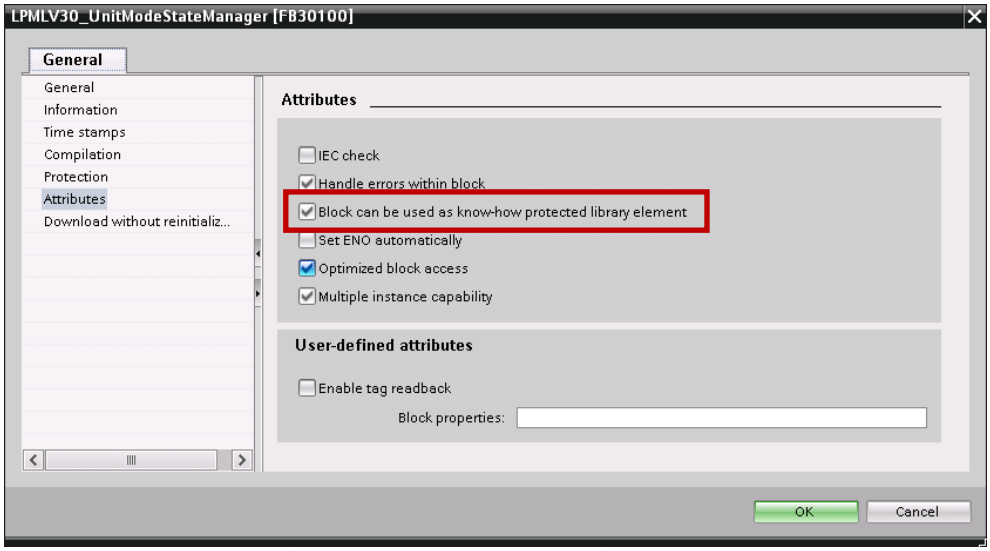
W tym celu, nie należy deklarować zmiennych wewnętrznych, a także parametrów wejściowych oraz wyjściowych jako zmienne temp, ponieważ zmiennych tych nie można śledzić w tablicach *watch table, force table* oraz na urządzeniu HMI.

Reguła: Pole "Block can be used as know-how protected library element”

Dla bloków FB oraz FC, w zakładce Attributes, należy upewnić się, że pole: “Block can be used as know-how protected library” zostało zaznaczone.

Oznacza to, że blok został zaprogramowany w sposób modułowy i nie będzie korzystał ze zmiennych oraz stałych globalnych

Rysunek 4-2: Właściwości bloku



Wskazówka Pole ”Block can be used as know-how protected library element" jest niedostępne w TIA Portal w wersji 14 lub wyższej.
Funkcję tę można znaleźć w zakładce: "Compilation" -> "Library conformance".

4.1.4 Komentarze

Istnieją dwa typu komentarzy:

- Komentarz blokowy (dot. funkcji lub części kodu)
- Komentarz liniowe (dot. linii kodu)

Zalecenie: Komentarze blokowe

Komentarz tego typu należy wstawić przed fragmentem kodu, którego dotyczą.

Zalecenie: Komentarze liniowe

Komentarz tego typu należy wstawić na końcu linii kodu, której dotyczą, lub w wyjątkowych przypadkach na początku.

Zalecenie: Komentarze rozpoczynające się od //

Aby usprawnić proces komentowania podczas debuggingu, należy wyłącznie stosować komentarze rozpoczynające się od //.

Reguła: Stosowanie szablonów do opisywania bloków

Bloki należy opisywać: w nagłówku kodu (SCL) lub w komentarzu blokowym (LAD, FBD). Opis powinien zawierać następujące informacje:

- Nazwę firmy
- (Opcja) © Copyright, Wszystkie prawa zastrzeżone …
- (Opcja) informacje dodatkowe
- Nazwę biblioteki
- Sterowniki, na których przeprowadzono test wraz z wersją Firmware (np. S7-1511 V1.6)
- Wersję TIA Portal, w której utworzono blok
- Ograniczenia dostępu (np. do pewnych typów bloków OB)
- Wymagania (np. dodatkowy sprzęt)
- Opis funkcjonalności
- Wersję bloku, autora oraz datę utworzenia

Modyfikowanie bloku: Wzór nagłówka

```
//===========================================================================
// Company// (c)Copyright (year)
//---------------------------------------------------------------------------
// Biblioteka:              (do której odnosi się kod)
// Testowano za pomocą:     (sterownik + wersja FW)
// Engineering:            TIA Portal (wersja FW)
// Ograniczenia: (typy bloków OB, itp.)
// Wymagania: (sprzęt, pamięć, itp.)
// Funkcjonalność:(funkcja jaką ma spełniać dany blok)
//---------------------------------------------------------------------------
// Zmiany w logu:
// Wersja Data          Autor             Data wprowadzenia zmian
// 01.00.00 dd.mm.yyyy (Imię i nazwisko) Data pierwszej wersji //
===========================================================================
```


4.1.5 Parametry formalne: Input, Output oraz InOut

Reguła: Nie należy stosować prefiksów dla parametrów formalnych

Nie należy stosować prefiksów dla parametrów formalnych (*Input, Output* oraz *InOut)* bloków FC/FB.

Reguła: Wymiana danych przez interfejs bloku

Wymiana danych pomiędzy blokami FB lub FC odbywa się przez interfejsy *Input*, *Output* oraz *InOut*.

Dostęp do zmiennych *Statycznych* poza blokami jest zabroniony.

Zalecenie: Używaj elementartnych typów danych np. In, Out lub InOut

Dla elementarnych typów danych (takich jak WORD, DWORD, REAL, INT, TIME), należy używać interfejsu *Input* lub *Output*.

Interfejs *InOut* należy używać wyłącznie, gdy wartość zostanie edytowana zarówno wewnątrz jak i poza blokiem.

Zalecenie: Podsumowanie parametrów rzeczywistych jako dane PLC

W przypadku transferu wielu parametrów, zaleca się podsumowanie tych danych jako dane PLC, a następnie zadeklarowanie ich jako zmienną *InOut*.

Do danych tego typu zaliczają się: dane konfiguracyjne, wartości rzeczywiste, wartości zadane, a także parametry wyjściowe oddające bieżący stan bloku funkcyjnego.

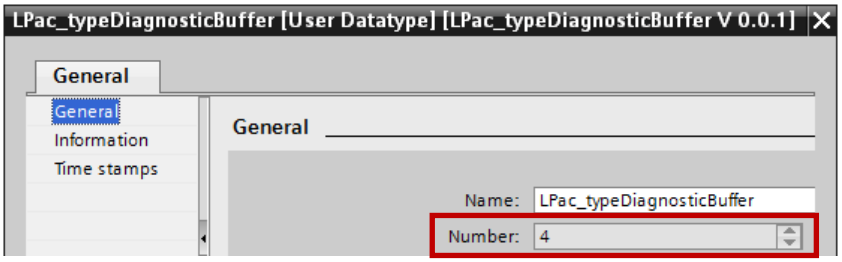
Zmienne sterujące oraz statusowe, które ulegają częstej zmianie, należy zadeklarować jako elementarne zmienne *Input* lub *Output*.

Reguła: Nie należy używać danych typu STRUCT

W programie PLC, w miejsce danych "struct" należy korzystać z danych PLC.

Wskazówka Wyjątkiem są bloki chronione (know-how protected blocks). W tym przypadku zastosowanie danych PLC może nie być najlepszym rozwiązaniem, gdyż dane tego typu otrzymują własną numerację w tle. Jeżeli następnie dane te zostaną użyte w bloku chronionym, wspomniany numer nie może zostać zmieniony podczas kopiowania bloku do innego projektu. W przeciwnym razie, będzie można dokonać kompilacji dopiero po podaniu hasła dla bloku chronionego.

Rysunek 4-3: Numeracja danych PLC



Zalecenie: Przesyłanie zmiennych strukturalnych jako InOut

W przypadku zmiennych strukturalnych (np. ARRAY, STRING, itp.) oraz danych PLC, należy korzystać z interfejsu *InOut*.

W przypadku identycznych ustawień optymalizacji dla połączonych danych oraz wywołanego bloku, dane nie są kopiowane. Zamiast tego system tworzy odnośnik do używanych danych. Rozwiązanie to pozwala zaoszczędzić miejscce wpamięci ładowania jednostki centralnej.

Wskazówka W przypadku interfejsów *InOut* należy zwracać szczególną uwagę na ustawienia optymalizacji dla bloku oraz połączonych danych.

Ustawienia te powinny być identyczne (blok i dane zoptymalizowane lub blok i dane niezoptymalizowane). W przeciwnym wypadku system będzie tworzył lokalną kopię danych, co odbije się negatywnie na wydajności interfejsu *InOut.*

Wskazówka Więcej informacji na ten temat można znaleźć w podręczniku: "Podręcznik Programowania dla S7-1200/S7-1500” w rozdziale "Interfejsy Bloków"
https://support.industry.siemens.com/cs/ww/en/view/81318674

Zalecenie: Edycja zmiennej *Output* tylko raz na cykl

Zaleca się przypisywanie nowej wartości dla zmiennej *Output* tylko jeden raz w trakcie cyklu.

4.2 Deklaracja zmiennych

4.2.1 Zmienne statyczne i tymczasowe

Reguła: Zmienne statyczne tylko dostępne lokalnie

Zmienne statyczne nie są dostępne poza blokiem.
Ma to znaczenie w przypadku wywoływanie oraz łączenia danych typu "instance" danego bloku.

Reguła: Prefiksy zmiennych statycznych i tymczasowych: stat; oraz temp;

Prefiksy służą do odróżnienia zmiennych statycznych oraz tymczasowych od parametrów transferu oraz parametrów wyjściowych. Listę prefiksów wraz z objaśnieniem można znaleźć w tabeli 4-2 poniżej.

Prefiksy te należy stosować wyłącznie dla bloków, które posiadają kompletny interfejs. Nie dotyczy to bloków globalnych (global DB) oraz danych typu PLC.

Tabela 4-2: Prefiksy zmiennych

Prefiks	Opis
stat	Zmienne statyczne → Dane typu "instance" niedostępne z zewnątrz
temp	Zmienne tymczasowe → Dane typu "instance" niedostępne z zewnątrz
	Zmienne *Input* oraz *Output* (bez prefiksu) → Dane typu "instance" dostępne z zewnątrz
	Zmienne *InOut* (bez prefiksu) → Można modyfikować połączone dane zarówno dla użytkownika jak i w obrębie bloku

4.2.2 Stałe

Reguła: Identyfikator stałych zawsze WIELKIMI LITERAMI oraz z podkreślnikiem

Nazwy stałych należy zawsze pisać WIELKIMI LITERAMI.
Jeżeli nazwa zawiera kilka słów lub skrótów, należy je oddzielić za pomocą podkreślników.

Reguła: Tylko stałe lokalne

Aby zagwarantować poprawne funkcjonowanie bloków w bibliotece, należy używać wyłącznie stałych lokalnych. Pozwoli to uniknąć błędów podczas kompilacji programu użytkownika.

Jeżeli stałe lokalne będą udostępnione dla użytkownika bloku, muszą być również utworzone jako stałe globalne. Nazwa stałych globalnych musi zawierać odniesienie do bloku lub biblioteki.

Tyczy się to w szczególności tych stałych, które odnoszą się do określonych wartości wyjściowych danego bloku np. numeracji błędu.

Stałe globalne (użytkownika) mogą być utworzone jako zmienne PLC. Aczkolwiek, stałe te nie będą automatycznie wykorzystywane przez sterownik podczas wywołania bloku.

Przykład

Rysunek 4-4: Stałe w bloku FB

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

Zalecenie: Stałe dla wartości różnych od 0

Dla zmiennych, których wartość numeryczna ma być różna od 0 należy używać stałych.

W ten sposób wartość ta będzie zmieniana centralnie (w stałej), a nie w poszczególnych miejscach kodu.

Przykład

Rysunek 4-5: Używanie stałych

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;
```

Wskazówka Stałe to identyfikatory, którym przypisana jest wartość tekstowa lub numeryczna. Używanie stałych nie powoduje spadku wydajności jednostki centralnej ani nie wpływa na wzrost zużycia pamięci danych.

Natomiast, z uwagi na większą ilość znaków w kodzie, zwiększeniu ulega wykorzystanie pamięci ładowania (load memory) jednostki centralnej.

4.2.3 Tablice

Zalecenie: Nazwa tablicy: liczba mnoga

Nazwa tablicy powinna być w liczbie mnogiej.

Przykład

Tablica struktury osi

axisData → niewłaściwa nazwa
axesData → właściwa nazwa

Zalecenie: Indeks tablicy rozpoczyna się od 0 i kończy stałą

Za dolną granicę tablicy należy przyjąć 0. Natomiast, górna granica tablicy to stała (np. DIAG_BUFFER_UPPER_LIM).

Tablice rozpoczynające się od 0 (tzw."zero-based") mają sens, ponieważ pewne komendy systemowe, np. MOVE_BLK_VARIANT działają z zerem. Zatem, indeks można wprowadzić bezpośrednio do funkcji systemowej bez potrzeby przeliczania.

Inną zaletą jest to, że programy takie jak WinCC (Comfort, Advanced oraz Professional) działają wyłącznie z tablicami "zero based".

Alternatywą dla tablic "zero based" są tablice, których obydwie granice określane są za pomocą stałej.

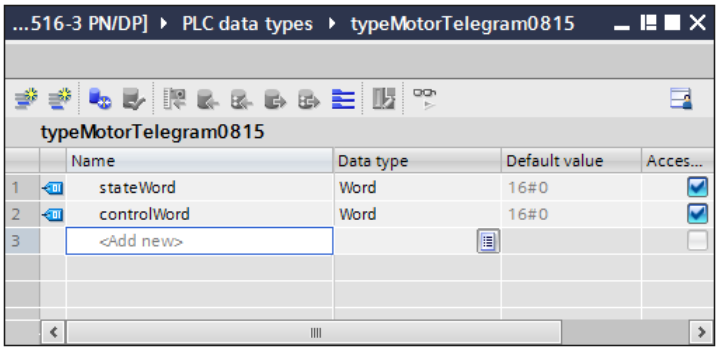
4.2.4 Dane PLC

Reguła: Prefiks "type"

Identyfikator danych użytkownika musi być poprzedzony prefiksem "type".

Przykład

Rysunek 4-6: Dane PLC



4.2.5 Inicjalizacja

Reguła: Inicjalizacja zmiennych tymczasowe w programie

Zmienne L stack (*Temp*) muszą zostać zainicjalizowane przez użytkownika. Należy pamiętać, że zmienne muszą zawsze być najpierw zapisane a potem odczytane.

Przykład

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

Reguła: Inicjalizacja w oparciu o typ zmiennej

Inicjalizacja zmiennej zawsze odbywa się w oparciu o typ zmiennej. Przykładowo, zmienna typu WORD będzie inicjalizowana wartością 16#0001, a nie 16#01.

Przykład

Rysunek 4-7: Inicjalizacja danych statycznych

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

Zalecenie: Inicjalizacja parametrów dla przerw (time-out): -1.0

Parametry użytkownika, które korzystają z wartości TO (np. prędkość, przyspieszenie, szarpnięcie), inicjalizowane są wartością -1.0. Pozwala to odróżnić, czy doszło do transferu wartości dla parametru.
Jeżeli wartość nie została przypisana przez użytkownika, przyjmowane są wartości domyślne dla TO.

4.3 Instrukcje

4.3.1 Operatory i wyrażenia

Zalecenie: Odstęp przed i po operatorach

Należy pamiętać o dodaniu spacji przed oraz po uzyciu operatorów binarnych.

Przykład

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

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

Zalecenie: Wyrażenia zawsze w nawiasach

Wyrażenia muszą zawsze występować w nawiasach.

Przykład

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

4.3.2 Instrukcje sterujące

Zalecenie: Odstępy przy warunkach składowych (partial conditions)

W przypadku bardziej złożonych wyrażeń należy zachować linię odstępu pomiędzy 'warunkami składowymi'.

Reguła: Odstępy

Pomiędzy warunkiem oraz poleceniem musi być wyraźny odstęp.

Np. po warunku THEN należy wstawić linię ostępu przed zaprogramowaniem polecenia.

Zalecenie: Wcięcia

Jeżeli warunek nie mieści się w danej linii, instrukcje logiczne należy umieścić na początku danego wiersza.

W przypadku warunków, które zajmują więcej niz jednen wiersz (np. w instrukcjach IF), każdy wiersz należy rozpocząć od wcięcia o długości dwóch spacji.

Instrukcję THEN należy umieścić w osobnym wierszu, na równi z instrukcją IF.

Jeżeli treść instrukcji IF mieści się w jednym wierszu, wtenczas instrukcję THEN można umieścić na końcu tego wiersza.

W przypadku głębszego zagnieżdżenia, operandy należy umieszczać na początku osobnego wiersza. Warunek zagnieżdżony można poznać po pojedynczym nawiasie.

Powyższe zalecenia dotyczą również innych instrukcji np. CASE, FOR, WHILE.

Przykład

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

Reguła: Instrukcja CASE musi zawierać gałąź ELSE

Instrukcje CASE muszą zawierać gałąź ELSE, aby umożliwić raportowanie błędów podczas realizacji programu.

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

Zaleceinia: Instrukcja CASE zamiast wielu gałęzi ELSIF

Aby program był bardziej przejrzysty, należy stosować instrukcję CASE zamiast instrukcji IF składającej się z wielu gałęzi ELSIF.

Reguła: Wcięcia w instrukcjach

Każda instrukcja w pionie instrukcji sterującej rozpoczyna się od wcięcia.

Przykład

Instrukcja IF

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

Przykład

Instrukcja CASE

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

Przykład

Instrukcja FOR

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

Przykład

Instrukcja FOR z długością skoku

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

Przykład

Warunkowe zakończenie pętli za pomocą instrukcji EXIT

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

Przykład

Sprawdzenie warunku pętli za pomocą instrukcji CONTINUE

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

Przykład

Instrukcja WHILE

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

Przykład

Instrukcja REPEAT

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

4.3.3 Postępowanie w przypadku błędów

Reguła: Kod błędu należy zawsze przeanalizować

Jeżeli Funkcje (FC), bloki funkcyjne (FB) lub funkcje systemowe zwracają kod błędu, należy go zawsze poddać analizie.

Więcej informacji na ten temat można znaleźć w rozdziale 4.5 Błędy i diagnostyka.

4.4 Programowanie w oparciu o standard PLCopen

Organizacja PLCopen opracowała standard programowania dla bloków Motion Control. W tym podręczniku standard ten występuje w formie uproszczonej i można go stosować do programowania wszystkich asynchronicznych bloków funkcyjnych. Opisuje on interfejs bloku oraz zachowanie sygnałów.

Dzięki tej standaryzacji można znacznie uprościć proces programowania bloków funkcyjnych.

Reguła: Standardowe identyfikatory dla PLCopen

Jeżeli funkcja zgdona z *PLCopen Function Blocks for Motion Control V2.0* wymaga użycia parametrów stadardowych, wtenczas należy używać również ich standardowych identyfikatorów.

Poniższe parametry uważa się za stanardowe

Tabela 4-3: Standardowe parametry zgodne z PLCopen

Standardowe sygnały zgodne z PLCopen	Znaczenie
Parametry *Wejściowe*	
execute	**execute bez aktualizacji:** Parametry rozpoczynają na zboczu narastającym wejścia *execute* - funkcja zostaje uruchomiona. W przypadku zmiany parametrów, należy reinicjalizować wejście.
or	**execute z aktualizacją (continuous update)** Parametry rozpoczynają na zboczu narastającym wejścia *exectue*. Parametry można modyfikować, pod warunkiem że wejście *continuousUpdate* zostało skonfigurowanie.
enable	**enable:** Parametry rozpoczynają na zboczu narastającym wejścia *execute* i mogą być zmieniane. Funkcja jest uruchamiania (TRUE) oraz wyłączana (FALSE).
Parametry *Wyjściowe*	
Wyłączność done busy valid commandAborted error	**Dla parametru wejściowego *execute*:** Sygnały zwracane: *done*, *busy*, *commandAborted* oraz *error* wykluczają się nawzajem. Dla parametru wejściowego exectute zwracany jest jeden z tych sygnałów. **Dla parametru wejściowego *enable*:** Sygnały *valid* oraz *error* wykluczają się nawzajem.
done	Wyjście zwraca sygnał *done,* gdy polecenie zakończy się pomyślnie.
busy	**Dla parametru wejściowego *execute*:** FB nie skończył przetwarzać polecenia, można spodziewać się nowych wartości wyjściowych. Jeżeli wyjście zwróci sygnał: *done, commandAborted* lub *error*, wtenczas na zboczu narastającym parametru execute zostanie zwrócony sygnał: *busy*. **Dla parametru wejściowego *enable*:** FB nie skończył przetwarzać polecenia.

Standardowe sygnały zgodne z PLCopen	Znaczenie
	Sygnał *busy* zostaje zwrócony na zboczu narastającym parametru i pozostaje aktywny dopóki FB nie zakończy przetwarzać polecenia.
active	**Sygnał opcjonalny,** w celu uzyskania zgodności z PLCopen (*tryb buforowania* bloków funkcyjnych). Sygnał jest zwracany jak tylko blok przejmie kontrolę nad osią. Jeżeli nie wybrano trybu buforowania sygnału *active* i *busy* mogą być identyczne.
commandAborted	**Sygnał opcjonalny** sygnalizuje, że realizacja bloku funkcyjnego została anulowana. Przykład: Jeden blok ustawia oś w pozycji prostej, podczas gdy inny blok hamuje oś. Dla bloku pozycjonującego zwracany jest sygnał *commandAborted* ponieważ polecenie zostało przerwane.
valid	Sygnał używany jest tylko w połączeniu z parametrem *enable*. Sygnał *valid* zwracany jest w momencie gdy dla parametru *enable* zostaną zwrócone poprawne wartości. W przypadku błędu, sygnał *valid* zostanie resetowany.
error	Sygnalizuje błąd podczas przetwarzania bloku funkcyjnego na zboczu wznoszącym sygnałów wyjściowych.
status (w miejsce errorID)	Zawiera informacje o błędzie / status bloku. Z uwagi na kompatybilność, identyfikator *errorID* nie jest używany w obecnych systemach oraz blokach SIMATIC. Zastąpił go identyfikator *status*.
diagnostics	**Sygnał opcjonlany**: Szczegółowy bufor błędu Informacje o błędach, ostrzeżenie oraz informacje o blokach zapisywane są w buforze pierścieniowym. Jego wielkość (ilość elementów w tablicach) uzależniona jest od pamięci PLC. Budowa komunikatów diagnostycznych:- patrz rozdział 4.5 Komunikaty o błędzie i diagnostyka

4.4.1 Bloki z *execute*

Funkcja rozpoczyna się na zboczu narastającym parametru *execute*, przyjmując wartości parametrów wejściowych.

Jeżeli nie wybrano funkcji *continuousUpdate*, wartości parametrów zostaną zmienione dopiero przy uruchomieniu kolejnego zadania.

Zresetowanie parametru *execute* nie przerywa realizacji zadania, ale wpływa na sygnalizację stanu danego zadania. Jeżeli parametr *execute* zostanie zresetowany przed ukończeniem zadania, sygnały *done*, *error* oraz *commandAborted* mogą zostać zwrócone przez czas trwania jednego cyklu.

Sygnały diagnostyczne (error, status, diagnostics) mogą zostać zresetowane wyłącznie nowym zboczem narastającym.

Po ukończeniu zadania, kolejne rozpocznie się na nowym zboczu narastającym parametru *execute*.

Dzięki temu, na początku każdego zadania blok znajduje się w stanie wyjściowym i jest przetwarzany niezależnie od poprzedniego zadania.

Reguła: Paramer: *execute* wymaga *busy* oraz *done*

Jeżeli programista skorzysta z parametru wejściowego *execute*, wtenczas musi również wykorzystać parametry wyjściowe *busy* oraz *done*.

Przykład

Rysunek 4-8: Reprezentacja LAD

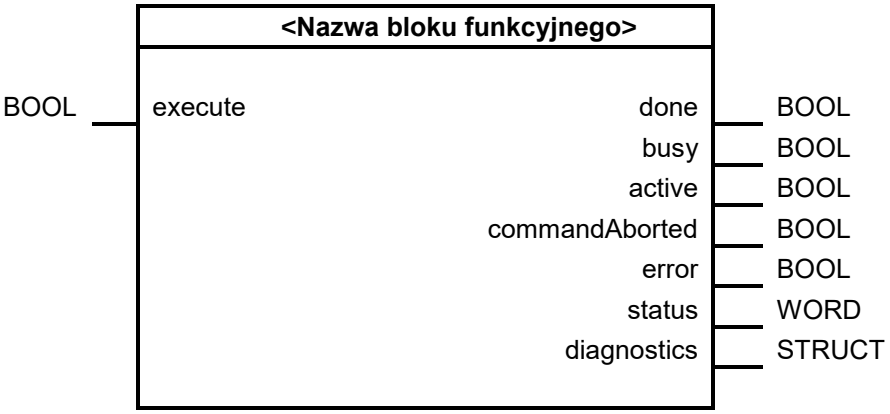
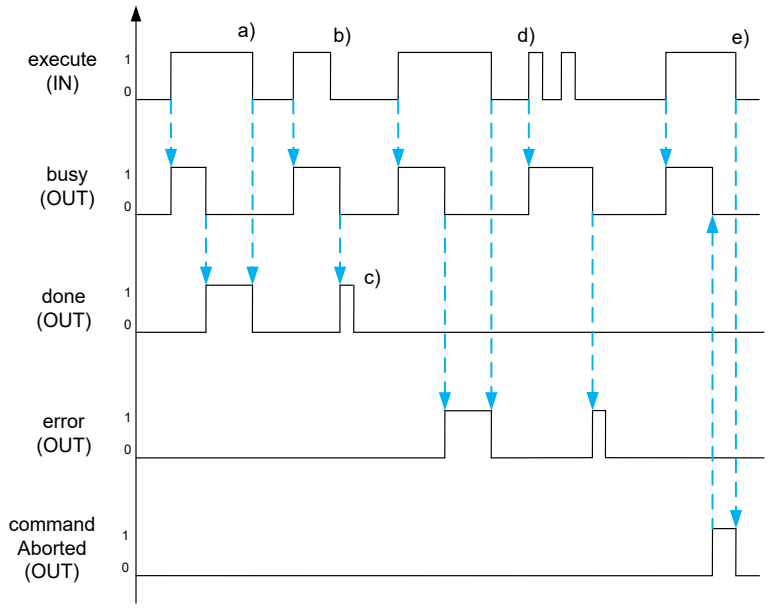


Diagram przejścia sygnału – blok z *execute*

UWAGA	Jeżeli parametr *execute* zostanie zresetowany przed zwróceniem sygnału *done* wtenczas sygnał done należy ustawić tylko dla jednego cyklu.

Rysunek 4-9: Diagram przejścia sygnału dla bloku funkcyjnego z parametrem wejściowym *execute*



a) *done*, *error* oraz *commandAborted* są resetowane w przypadku zbocza opadającego dla parametru *execute*.

b) Blok FB jest nadal przetwarzany w przypadku zbocza opadającego dla parametru *execute*.

c) Jeżeli *execute* zwraca FALSE, wtenczas *done*, *error* oraz *commandAborted* są zwracane przez tylko jeden czas cyklu.

d) Jeżeli nowe zadnie pojawi się na zboczu narastającym parametru ***execute,*** a blok jest nadal przetwarzany (*busy* = TRUE), wtenczas: 1) stare zadanie zostanie zakończone z parametrami początkowymi 2) stare zadanie zostanie anulowane, i uruchomione ponownie z nowym zestawem parametrów

Zachowanie to zależy od aplikacji i musi zostać odpowiednio opisane.

e) Jeżeli zadnie zostanie przerwane np. przez zadanie nadrzędne wtenczas blok zwróci sygnał *commandAborted*, a proces zostanie natychmiast przerwany.

Może do tego dojść np. w przypadku instrukcji AWARYJNY STOP dla danej osi, podczas gdy inny blok realizuje funkcję ruchu.

4.4.2 Bloki z parametrem *enable*

Parametr *enable* rozpoczyna przetwarzanie zadania. Zadanie będzie przetwarzane tak długo, jak parametr pozostanie aktywny; nowe wartości mogą być wprowadzane i przetwarzane.

Zresetowanie parametru *enable* kończy przetwarzanie zadanie.

Jeżeli rozpoczęto nowe zadanie, blok wraca do stanu wyjściowego - można go ponownie skonfigurować.

Reguła: Parametr: *enable* wymaga *valid*

Jeżeli programista korzysta z parametru wejściowego *enable* wtenczas musi również skorzystać z parametru wyjściowego *valid*.

Przykład

Rysunek 4-10: Reprezentacja LAD

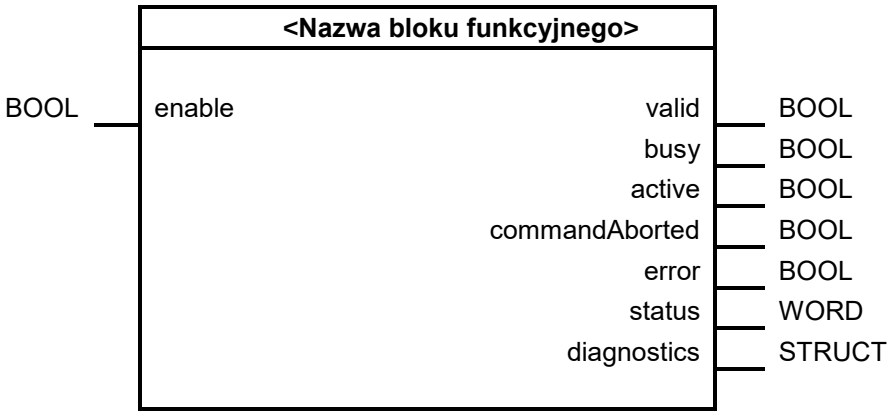
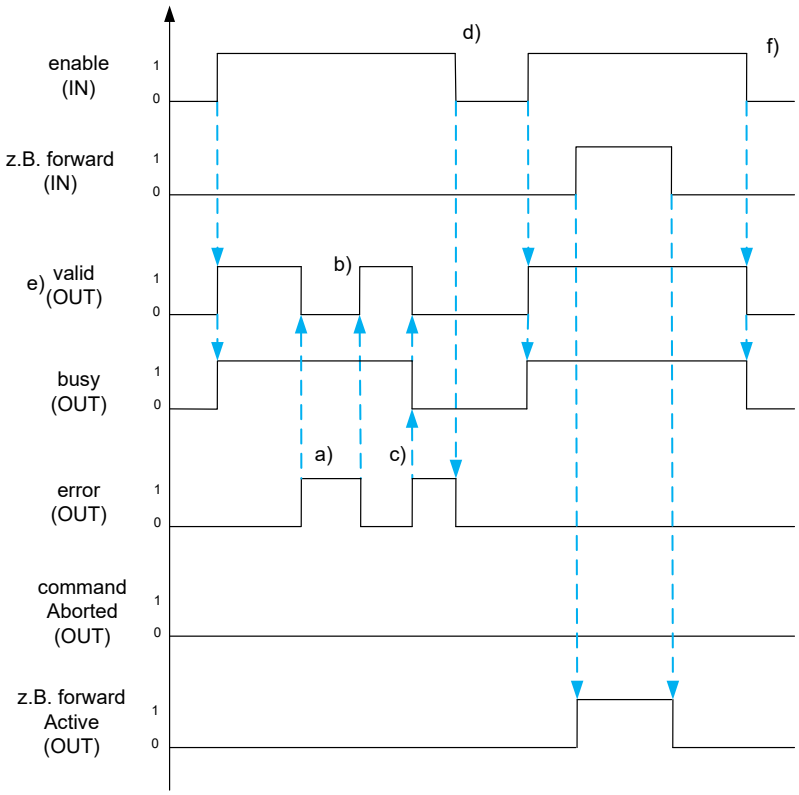


Diagram przejścia sygnału: blok z *enable*

Rynuek 4-11: Diagram przejścia sygnału dla bloku funkcyjnego z parametrem *enable*



a) Jeżeli sygnał *error* przyjmie wartość TRUE wtenczas sygnał *valid* zostaje przerwany, a blok przerywa realizację zadania. Ponieważ blok jest w stanie poradzić sobie z błędem, parametr wyjściowy zwraca sygnał *busy*.

b) Po usunięciu przyczyny błędu i przywróceniu komunikacji, parametr wyjściowy zwraca sygnał: *valid*

c) Wystąpił błąd, który może zostać usunięty tylko przez użytkownika. Blok zwraca sygnał *error*, parametry *busy* i *valid* zostają zresetowane.

d) Błąd oczekujący na interwencję użytkownika może zostać zatwierdzony tylko na zboczu opadającym parametru *enable*.

e) Jeżeli sygnał *valid* jest TRUE oznacza, że blok został aktywowany - nie wystąpiły żadne błędy

f) Jeżeli sygnał *enable* przyjmie wartość FALSE, wtenczas sygnały *valid* oraz *busy* zostaną zresetowane.

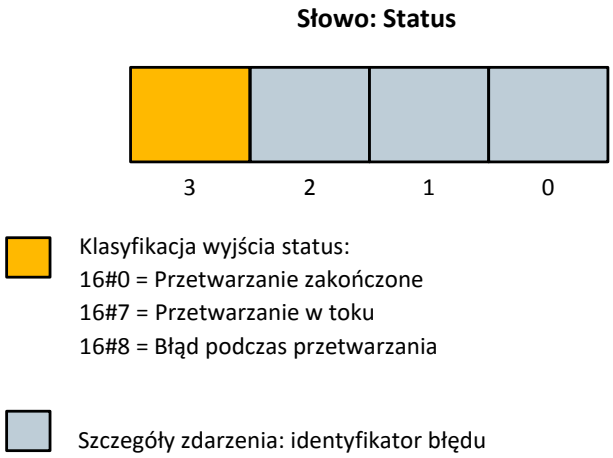
CommandAborted, *error* oraz *done* są zawsze zwracane pod warunkiem, że parametr *execute* jest przetwarzany przyanajmniej przez jeden cykl.

4.5 Komunikaty o błędzie i diagnostyka

Reguła: Status parametrów formalnych: zwracanie błędów

Błąd sygnalizowany jest przez zwrócenie sygnału *error* oraz zmianę najbardziej znaczącego bitu w obrazie wyjść procesu. Pozostałe bity wykorzystywane są do podania kodu błędu w celu identyfikacji przyczyny błędu. Ze względu na kompatybilność z poprzednimi blokami SIMATIC, wyjście *status* wykorzystywane jest do sygnalizacji błędu, w miejsce wyjścia *errorID* (wymaganego przez standard PLCopen).

Rysunek 4-12: Struktura wyjścia "status"



Zalecenie: Status parametrów - ustandaryzowane numery błędów

W celu ustandaryzowania informacji diagnostycznych, należy przestrzegać numeracji błędów zgodnie z tabelą poniżej.

Tabela 4-4: Numeracja błędów

Przyczyna błędu	Numeracja
Zadanie zakończone; brak szczegółów	16#0000
Zadanie zakończone; szczegóły	16#0001 ... 16#0FFF
Brak przetwarzanego zadania (**wartość początkowa**)	16#7000
Pierwsze wywołanie po przyjęciu zadania (*execute*: zbocze narastające)	16#7001
Kolejne wywołanie podczas przetwarzania; brak szczegółów	16#7002
Kolejne wywołanie podczas przetwarzania; szczegóły Ostrzeżenia nie wpływają na przetwarzanie.	16#7003 .. 16#7FFF
Błąd bloku	16#8001 .. 16#81FF

Przyczyna błędu	Numeracja
Błąd parametryzacji	16#8200 .. 16#83FF
Zewnętrzny błąd przetwarzania (np. podczas wywołania wej/wyj.	16#8400 .. 16#85FF
Wewnętrzny błąd przetwarzania (np. podczas wywołania funkcji systemowej)	16#8600 ..16#87FF
Zarezerwowane	16#8800 .. 16#8FFF
Błędy użytkownika	16#9000 .. 16#FFFF

Zalecenie: Błąd oczekuje na zatwierdzenie

Jeżeli błąd zostanie wykryty podczas przetwarzania bloku funkcyjnego, bieżące zadanie zostanie przerwane. Kod błędu będzie czekał na zatwierdzenie (zbocze ujemne parametru *execute*; zbocze opadające również konieczne dla parametru *enable*, w zależności od przyczyny błędu).

Zalecenie: Zwracanie kodów statusu instrukcji

Kody statusu instrukcji (bloków systemowych) są zwracane bez żadnych zmian na wyjściu *status*. Dlatego, jeżeli chodzi o dokumentację bloków, należy nanosić komentarze dot. kodów błędów w TIA Portal.

Zalecenie: wyjście *statusID* dla indentyfikacji przyczyny błędu

Aby określić przyczynę błędu, zaleca się korzystanie z dodatkowego wyjścia *statusID* o następujących właściwościach:

- Wskazuje, który blok, sub-blok / sub-instancja sygnalizuje błąd. Zaleca się przypisanie numeru*statusID* “1” dla bloków oraz numerów od “2” wzwyż dla sub-bloków.
- Jeżeli wyjście zwraca wartość "0" wtenczas nie wykryto żadnych błędów
- Jest to wyjście typu UINT

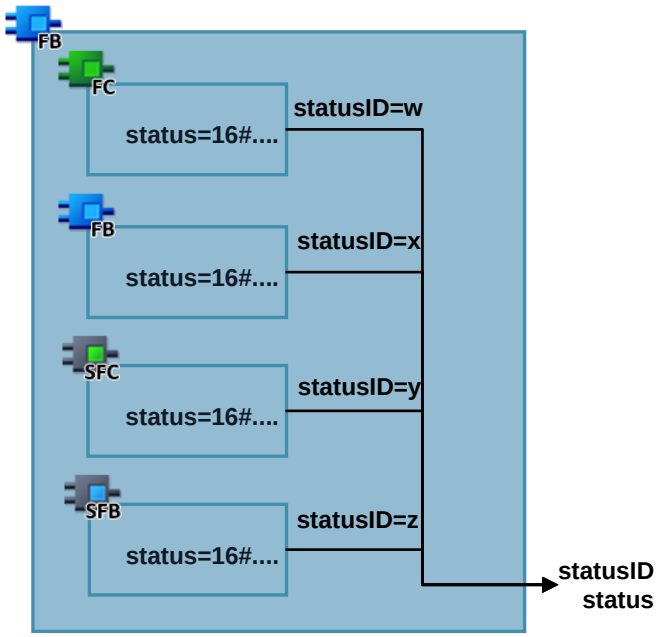
Wszystkie instancje mają przypisany unikalny *statusID* w obrębie wywołanego bloku.

Zalecenie: wyjście *statusID* output oraz offset dla bloków zagnieżdżonych

W przypadku bloków zagnieżdżonych, zaleca się zaprogramowanie unikalnego przyporządkowania przyczyn błędów do wyjścia *statusID* bloku wyższego rzędu. Można to zrobić dodając *offset* do wartości *statusID* bloku niższego rzędu w przypadku wielopoziomowego zagnieżdżenia.

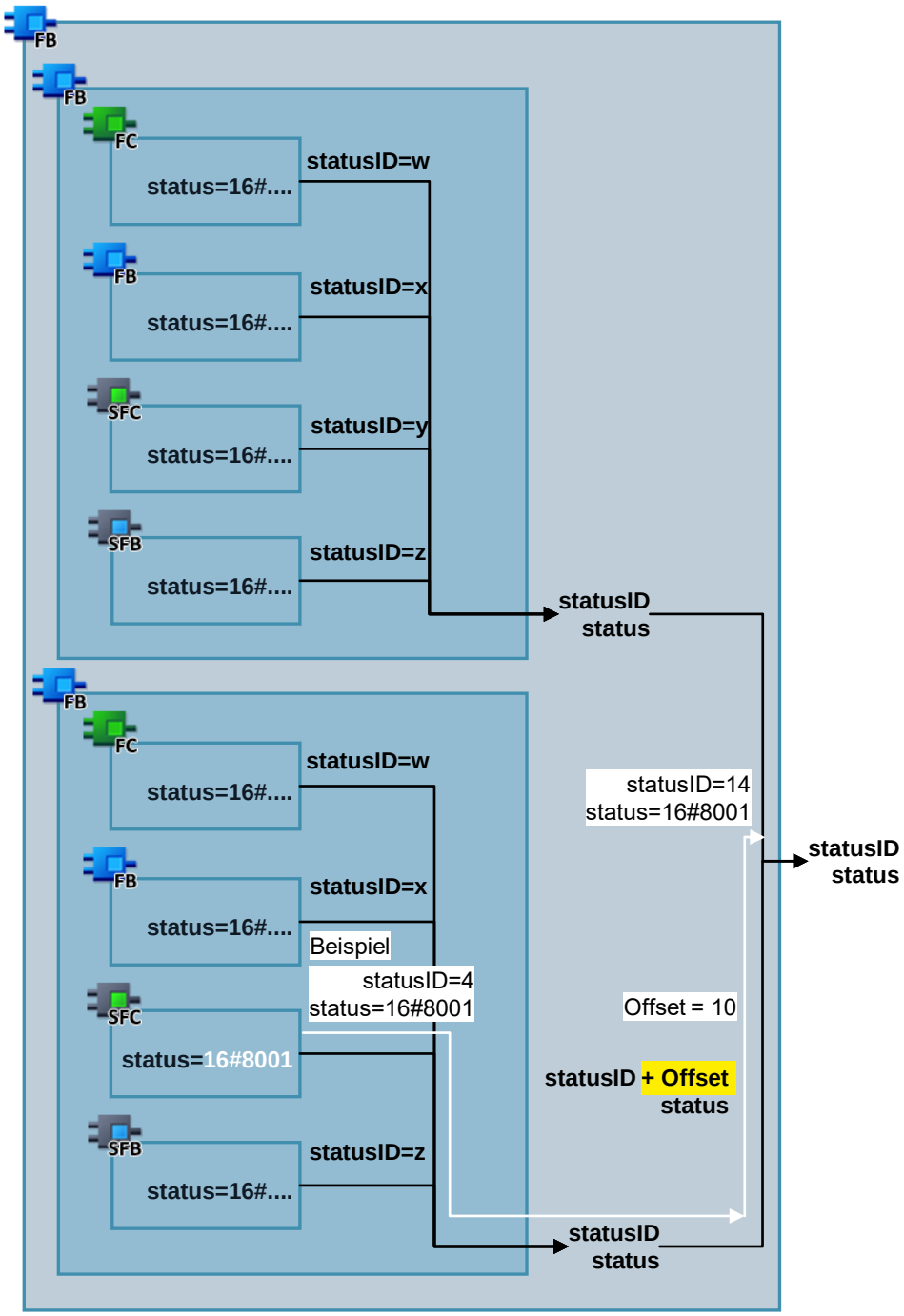
Przykład (głębokość zagnieżdżenia 1)

Rysunek 4-13: parametry *status* oraz *statusID* bloków zagnieżdżonych



Przykład (głębokość zagnieżdżenia 2)

Rysunek 4-14: parametry *status* oraz *statusID* dla bloków zagnieżdżonych



Zalecenie: Diagnostyka parametrów: struktura diagnostyki

W strukturze diagnostycznej, wszystkie poniższe informacje muszą zostać zapisane na wyjściu diagnostycznym. Co więcej, wartości konieczne do zdiagnozowania zachowania bloku, np. informacje runtime, również muszą zostać przechowane

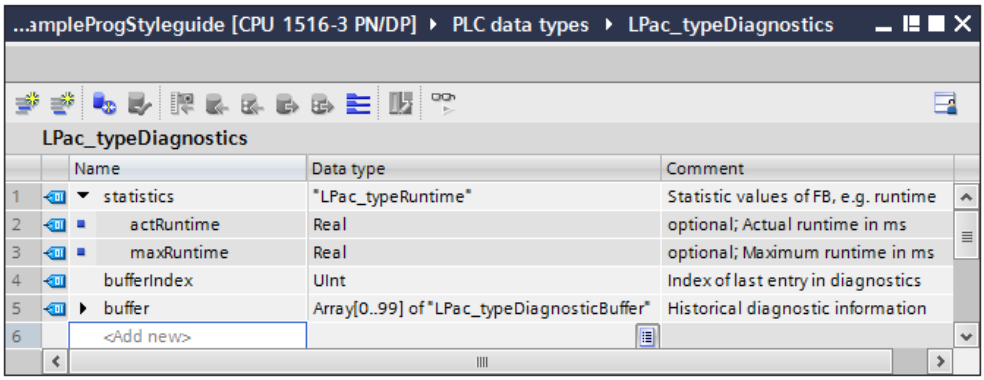


Tabela 4-5: Elementy struktury bufora diagnostycznego

Nazwa	Typ danych	Opcjonalne	Komentarz
timestamp	DATE_AND_TIME		Znacznik wystąpienia błędu
stateNumber	DINT		Stan urządzenia w momencie wystąpienia błędu
modeNumber	DINT	x	Tryb urządzenia w momencie wystąpienia błędu
subfunctionStatus	DWORD	x	Wartość zwracana błędów w wywołanych blokach
status	WORD		Status, który jednoznacznie identyfikuje błąd
additionalValueX	any	x	Wartości dodatkowe (X = numer) umożliwiające zapisanie danych diagnostycznych (np. położenie osi).

Parametr *timestamp* przechowuje informacje o czasie, w którym wystąpił błąd.

Parametr *stateNumber*, przechowuje informacje na temat stanu urządzenia w momencie wystąpienia błędu. W przypadku bloku funkcyjnego o innym trybie pracy, tryb pracy w którym doszło do błędu przechowywany jest w parametrze *modeNumber*.

W przypadku błędu funkcji systemowej lub wywołanego bloku FB / FC, kod błędu zwrócony jest w parametrze *subfunctionStatus*.

Kod błędu wyjścia *status* jest dodatkowo przechowywany w parametrze *status* komunikatu diagnostycznego.

Dodatkowe parametry dot. błędu przechowywane są w zmiennych *additionalValueX* .

W razie potrzeby, można dodać kolejne elementy.

Zalecenie: Retencyjność komunikatu diagnostycznego

Komunikat diagnostyczny powinien być retencyjny, aby umożliwić diagnostykę nawet w przypadku awarii zasilania sterownika.

4.6 Tablice, ślady, pomiary

Reguła: Notacja PascalCase dla tablic i śladów

Notacja PascalCase (rozpoczynanie od wielkiej litery) jest używana dla:

- Tablic zmiennych PLC
- Tablic: watch tables
- Śladów
- Pomiarów

4.7 Biblioteki

W tym rozdziale opisujemy reguły oraz zalecenia dotyczące programowania bibliotek. Reguły odnoszące się do kodu oraz nazywania zmiennych, które zostały poruszone w poprzednich rozdziałach obowiązują również w przypadku bibliotek.

Zalecenie: Dokumentowanie bibliotek

Zaleca się prowadzenie szczegółowej dokumentacji dla każdej biblioteki:

- Bloki oraz ich funckja
- Wersja systemu
- Historia zmian
- itp.

4.7.1 Przypisywanie nazw

Zalecenie: Nazwa biblioteki: Prefix L oraz maksymalnie 8 znaków

Nazwa każdej biblioteki rozpoczyna się od litery L (od angielskiego słowa "library" czyli biblioteka) np. LPac. Nazwa biblioteki nie powinna zawierać podkreślników (za wyjątkiem prefiksu biblioteki oraz nazwy bloku / zmiennej). Długość nazwy biblioteki nie może przekraczać 8 znaków.

Reguła: Wszystkie elementy otrzymują nazwę biblioteki w formie prefiksu

Wszystkie elementy danej biblioteki (elementy PLC oraz HMI) otrzymują nazwę biblioteki. W ten sposób unika się duplikacji nazwy bloków.

UWAGA Przed dodaniem bloku do biblioteki należy ustawić wszystkie jego właściwości takie jak numer, czy ochrona dostępu. Po dodaniu bloku do biblioteki nie można modyfikować jego właściwości.

Przykład

Tabela 4-6: Przypisywanie nazw dla przykładowej biblioteki LExample

Typ	Nazwa według zaleceń Styleguide
Biblioteka	LExample
Dane PLC	LExample_type<Nazwa>
Blok funkcyjny	LExample_<Nazwa>
Funkcja	LExample_<Nazwa>
Blok organizacyjny	LExample_<Nazwa>
Zmienne PLC	LExample_<Nazwa>
Tablica zmiennych PLC	LExample_<Nazwa>
Stała globalna	LEXAMPLE_<NAZWA>
Stała globalna dla kodu błędu	LEXAMPLE_ERR_<NAZWA>

Przykład

Identyfikator:	LCom_CommToClient
Biblioteka:	LCom
Funkcja:	Komunikacja TCP/IP pomiędzy różnymi urządzeniami

4.7.2 Konfiguracja

Reguła: Funkcje (FC), bloki funkcyjne (FB) oraz dane PLC

Funkcje, bloki funkcyjne oraz dane PLC są dodawane do biblioteki jako typy. Wszystko inne jest dodawane do biblioteki jako kopiowany szablon, szczególnie bloki organizacyjne oraz tablice zmiennych.

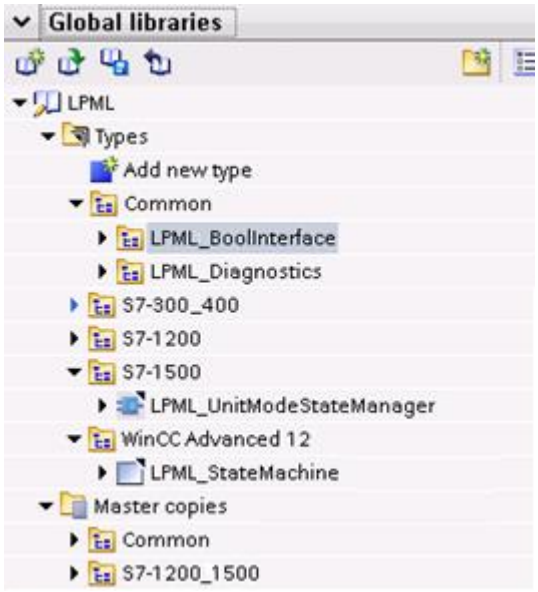
UWAGA	**Blok chroniony przed kopiowaniem zostaje przypisany do sterownika oraz wersji Firmware ostatniej kompilacji** Jeżeli blok został skompilowany na sterowniku S7-1500, wtenczas nie uruchomi się na S7-1200 pomimo tego, że obydwa sterowniki są kompatybilne. Warto zwrócić uwagę, że blok nie jest powiązany tylko ze sterownikiem, ale również z wersją firmware. Blok chroniony przed kopiowaniem nie może zostać ponownie skompilowany bez wprowadzenia hasła. W przypadku danych PLC należy pamiętać, aby ich nie modyfikować, ponieważ nie można ich zabezpieczyć przed kopiowaniem.

Zalecenie: Grupowanie w bibliotece

Bloki PLC oraz ekrany HMI danej biblioteki przypisane są do grupy o nazwie sterownika lub typu HMI. Jeżeli blok lub ekran HMI dotyczy wszystkich rodzajów sterowników (S7-300, S7-400, S7-1200 oraz S7-1500) lub ekranów HMI (…), wtenczas należy stworzyć wspólny folder.

Przykład

Rysunek 4-16: Budowa biblioteki



4.7.3 Wersja systemu

Reguła: Oznaczenie wersji systemu

Podstawową wersją systemu jest wersja V 1.0.0 (więcej szczegółów znajduje się w Tabeli 4-7). Wersje niższe niż 1.0 dotyczą wersji deweloperskich.

Zmiana trzeciej cyfry w oznaczeniu wersji informuje o zmianach, które nie miały wpływu na dokumentację (np. usunięcie błędów) oraz nie wprowadziły nowych funkcji.

Zmiana drugiej cyfry w oznaczeniu wersji informuje o wprowadzeniu nowych funkcji.

Zminan pierwszej cyfry w oznaczeniu wersji systemu informuje o wprowadzeniu nowych funkcji oraz , że wersja systemu nie jest kompatybilna z wersjami poprzednimi.

Reguła: Ciągłość wersji

Biblioteki numerowane są w sposób ciągły. W przypadku zmian, zmienia się numer biblioteki. Numerowanie bloków odbywa się na podobnej zasadzie jak numerowanie wersji systemu. Istnieje możliwość, że żadne z bloków nie będą mały tej samej numeracji co biblioteka do której są przypisane ponieważ numerowane są niezależnie (sprawdź w tabeli poniżej).

Przykład

Tabela 4-7: Numeracja bloków i bibliotek

Biblioteka	FB1	FB2	FC1	FC2	Komentarz
1.0.0	1.0.0	1.0.0	1.0.0		Pierwsza edycja
1.0.1	1.0.1	1.0.0	1.0.0		Usuwanie błędów w FB1
1.0.2	1.0.1	1.0.1	1.0.0		Optymalizacja FB2
1.1.0	1.1.0	1.0.1	1.0.0		Rozbudowa FB1
1.2.0	1.2.0	1.0.1	1.0.0		Rozbudowa FB1
2.0.0	2.0.0	1.0.1	2.0.0		Nowe funkcje dla FB1 oraz FC1
2.0.1	2.0.0	1.0.2	2.0.0		Usuwanie błędów w FB2
3.0.0	2.0.0	1.0.2	2.0.0	1.0.0	Nowa funkcja dla FC2

Reguła: Aktualizacja nagłówka

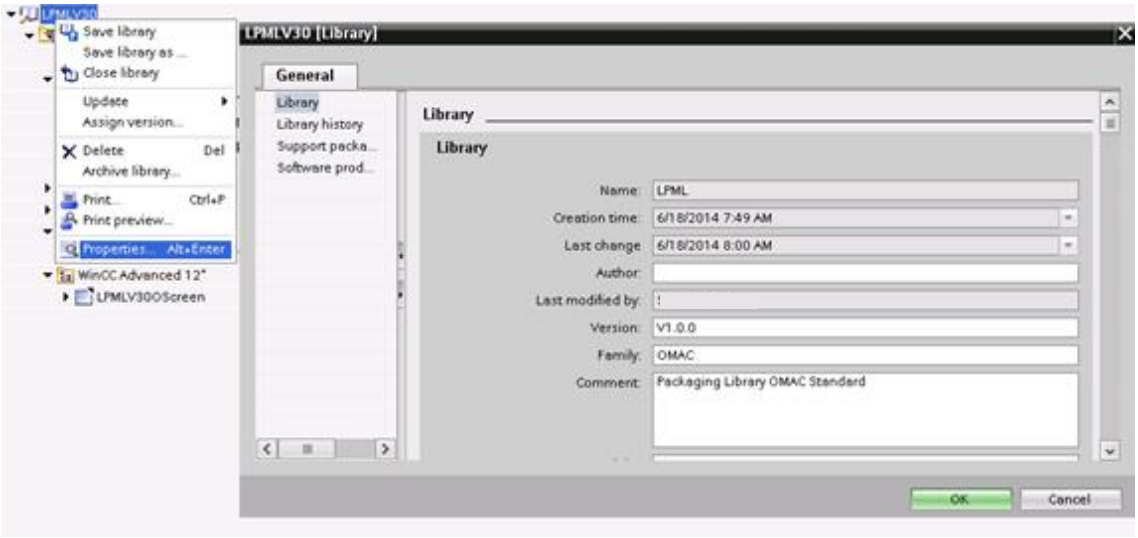
Każda zmiana wersji musi być odpowiednio opisana, np. w nagłówku bloku.

Reguła: Aktualizacja okna właściwości

Aktualną wersję biblioteki można znaleźć w oknie właściwości tej biblioteki.

Przykład

Rysunek 4-17: Okno właściwości biblioteki



Zalecenie: Szablony HMI OS

Dla przykładowych projektów, należy korzystać z szablonu HMI OS \6\ dostępnego do pobrania na stronie: https://support.industry.siemens.com/cs/ww/en/view/96003274

5 Podsumowanie

Zmienne (tagi)	Globalne	Stałe
	start	**MAX_SPEED**
Prefiks	-	-
Pierwsza litera	mała	-
Notacja	camelCasing	WIELKIE_LITERY

Podstawy
- unikalne, znaczące identyfikatory w j. angielskim
-Brak znaków specjalnych (§$%&…_)
- Maksymalnie 24 znaki
- Tablice: axesData [0..Max] of type…

Parametry formalne	IN	OUT	INOUT	STAT	TEMP	CONSTANTS
	enable	**done**	**motorData**	**statVelocity**	**tempVelocity**	**MAX_VELOCITY**
Prefiks	-	-	-	stat'	temp'	-
Pierwsza litera	mała	mała	mała	mała	mała	-
Notacja	camelCasing	camelCasing	camelCasing	camelCasing	camelCasing	WIELKIE_LITERY

Bloki danych DB	GLOBAL	Single Instance	Multi-instance
	MotorData	**InstHeater**	**instMotor**
Prefiks	-	'Inst'	'inst'
Pierwsza litera	wielka	wielka	wielka
Notacja	PascalCasing	PascalCasing	camelCasing

		Prefiks	Pierwsza litera	Notacja
Blok organizacyjny OB	**StationMain**	-	wielka	PascalCasing
Blok funkcyjny FB	**ConveyorControl**	-	wielka	PascalCasing
Funkcja FC	**Filling**	-	wielka	PascalCasing
Obiekt technologiczny	**PositioningAxis**	-	wielka	PascalCasing
Dane PLC	**typeMotor**	-	mała	camelCasing
Tablice: watch table	**MotorTags**	-	wielka	PascalCasing
Tablice: zmienne PLC	**InputTags**	-	wielka	PascalCasing
Ślady	**SpeedAxis**	-	wielka	PascalCasing
Pomiary	**HeaterControl**	-	wielka	PascalCasing

*Skróty (jeden / identyfikator)	
Min / Max	minimum / maksimum
Act	Faktyczny, bieżący
Next / Prev	Następny / poprzedni
Avg	Średnia
Diff / Sum	Różnica / suma
Pos	Pozycja
Ris / Fal	Narastające / opadające
Sim	Simulowane
Old	Stara wartość
Dir	Kierunek
Err	Błąd
Warn	Ostrzeżenie
Cmd	Polecenie
	*Zalecane

6 Dokumentacja pokrewna

Tabela 6-1: Dokumentacja pokrewna

	Temat	Link
\1\	Wsparcie Online	http://support.industry.siemens.com/
\2\	Strona do pobrania dokumentacji	https://support.industry.siemens.com/cs/ww/en/view/81318674
\3\	Totally Integrated Automation	http://www.siemens.de/tia
\4\	Podstawowe funkcje urządzeń modułowych	https://support.industry.siemens.com/cs/ww/en/view/61056223
\5\	Podręcznik programowania dla S7-1200/S7-1500	https://support.industry.siemens.com/cs/ww/en/view/81318674
\6\	Know-how	https://support.industry.siemens.com/cs/ww/en/view/96003274
\7	Standardy	Więcej informacji można znaleźć w standardach IEC 61131-8 lub EN 61131-3 załącznik 1.

7 Historia zmian

Tabela 7-1: Historia zmian

Wersja	Data	Modyfikacje
V1.0	10/2014	Pierwsza wersja
V1.1	06/2015	Poprawki
V1.2	10/2016	Poprawki Dodano rozdział 5 Podsumownie