# Programming Guideline for S7-1200/S7-1500

STEP 7 (TIA Portal)

Background and system description • October 2013

Applications & Tools

Answers for industry.

SIEMENS

# Warranty and Liability

## Caution

The functions and solutions described in this entry are mainly limited to the realization of the automation task. Please furthermore take into account that corresponding protective measures have to be taken in the context of industrial security when connecting your equipment to other parts of the plant, the enterprise network or the Internet. Further information can be found under the Entry ID 50203404.

http://support.automation.siemens.com/WW/view/en/50203404

## Siemens Industry Online Support

This document is an article from the Siemens Industry Online Support. The following link takes you directly to the download page of this document:

http://support.automation.siemens.com/WW/view/en/81318674

# Table of Contents

# 1 Preface

## Aims for the development of the new SIMATIC control generation

- An engineering framework for all automation components (controller, HMI, drives, etc.)
- Uniform programming
- Increased performance
- Full set of commands for every language
- Fully symbolic program generation
- Data handling even without pointer
- Reusability of created blocks

## Aim of the guideline

The new control generation SIMATIC S7-1200 and S7-1500 has an up-to-date system architecture, and together with the TIA Portal offers new and efficient options of programming and configuration. It is no longer the resources of the controller (e.g. data storage in the memory) that are paramount but the actual automation solution.

This document gives you many recommendations and tips on the optimal programming of S7-1200/1500 controllers. Some differences in the system architecture of the S7-300/400, as well as the thus connected new programming options are explained in an easy to understand way. This helps you to create a standardized and optimal programming of your automation solutions.

The examples described can be universally used for the controllers S7-1200 and S7-1500.

## Core content of this programming guideline

The following key issues on the TIA Portal are dealt with in this document:

- S7-1200/1500 innovations
  - Programming languages
  - Optimized blocks
- Recommendation on general programming
  - Operating system and user program
  - Storage concept
  - Symbolic addressing
  - Libraries
- Recommendations on hardware-independent programming

## Advantages and benefits

Numerous advantages arise by applying these recommendations and tips:

- Powerful user program
- Clear program structures
- Intuitive and effective programming solutions

# 2 S7-1200/1500 Innovations

## 2.1 Introduction

In general, the programming of SIMATIC controllers has stayed the same from S7-300/400 to S7-1500. There are the familiar programming languages such as LAD, FBD, STL, SCL or graph and blocks such as organization blocks (OBs), function blocks (FBs), functions (FCs) or data blocks (DBs). I.e. already created S7-300/400 programs can be implemented on S7-1500 and already created LAD, FBD and SCL programs on S7-1200 controller without any problems.

Additionally, there are many innovations that make programming easier for you and which allow a powerful and storage-saving code.

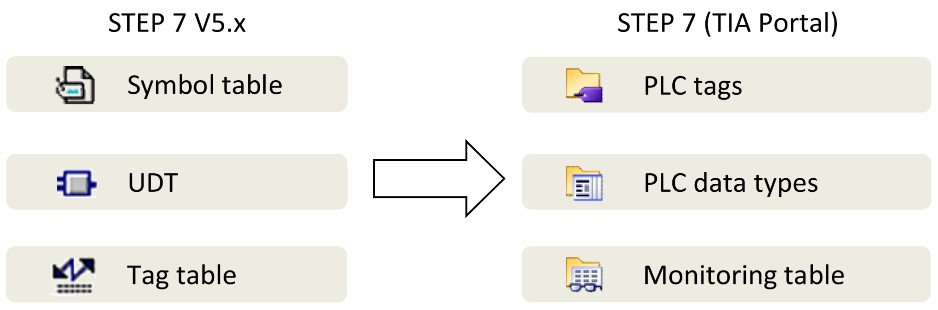
We not only recommend implementing programs that are implemented for S7-1200/1500 controllers 1:1 but also to check them for the new options and where applicable, to use them. The additional effort is often limited and you get a program code that is, for example,

- optimal in terms of memory and runtime for the newer CPUs
- easier to understand,
- and easier to maintain.

**Terms**

Some terms have changed in order to make better handling with the TIA Portal possible.

Figure 2-1: New terms in the TIA Portal



**Note**

You will find further information in the following entries:

What entries are available on the internet for the migration to STEP 7 (TIA Portal) and WinCC (TIA Portal)?
http://support.automation.siemens.com/WW/view/en/58879602

What prerequisites have to be fulfilled in order to migrate a STEP 7 V5.x project into STEP 7 Professional (TIA Portal)?
http://support.automation.siemens.com/WW/view/en/62101406

PLC migration for S7-1500 with STEP 7 (TIA Portal) V12
http://support.automation.siemens.com/WW/view/en/67858106

Programming recommendations for S7-1200 and S7-1500 with STEP 7 (TIA Portal) V12
http://support.automation.siemens.com/WW/view/en/67582299

Why is it not possible to mix register passing and explicit parameter transfer with the S7-1500 in STEP 7 (TIA Portal) V12?
Among others, the migration of STL programs to S7-1500 is described in this entry.
http://support.automation.siemens.com/WW/view/en/67655405

## 2.2 Programming languages

For the programming of a user program, various different programming languages are available. Each language has its own advantages, which can be variably used, depending on the application. Every block in the user program can therefore be created in any programming language.

- Contact plan (KOP or LAD)
- Function plan (FUP or FBD)
- Structured Control Language (SCL)
- Graph, only S7-1500, planned for S7-1200
- Instruction list (AWL or STL), only S7-1500

**Note**

You will find further information in the following entries:

What has to be observed when migrating a S7-SCL program in STEP 7 (TIA Portal)?
http://support.automation.siemens.com/WW/view/en/59784006

What instructions cannot be used in STEP 7 V11 in an SCL program?
http://support.automation.siemens.com/WW/view/en/58002710

How can the constants be defined under STEP 7 V11 in a S7-SCL program?
http://support.automation.siemens.com/WW/view/en/58065411

## 2.3 Optimized machine code

TIA Portal and S7-1200/1500 allow an optimized runtime performance in any programming language. All languages are compiled the same, directly into the machine code.

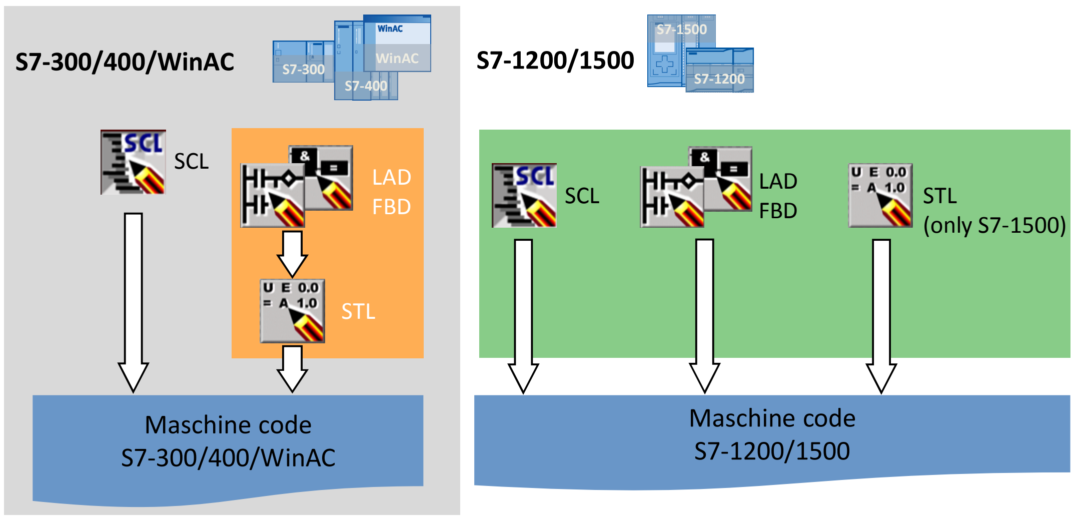
### Advantages

- All programming languages have the same high performance (with the same access types)
- No reduced performance through additional compiling with an intermediate step via STL

### Properties

The following figure displays the difference of the compilation of S7 programs into machine code.

Figure 2-2: Machine code generation with S7-300/400 and S7-1200/1500

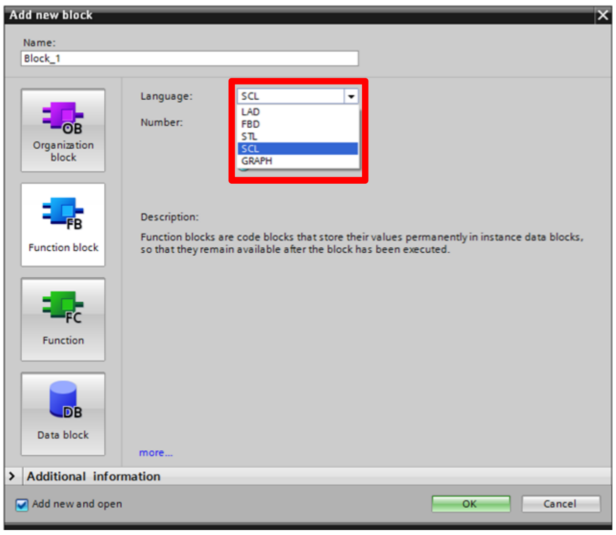


- For S7-300/400 controllers LAD and FDP programs are first of all compiled in STL before the machine code is created.
- For S7-1200/1500 controllers all programming languages are directly compiled into machine code.

## 2.4 Block creation

All blocks such as OBs, FBs and FCs can be programmed directly in the desired programming language. Thus no source has to be created for SCL programming. You only select the block, and SCL as programming language. The block can then be directly programmed.

Figure 2-3: “Add new block” dialog



Name: Block_1

Language: SCL

Number:

Description:

Function blocks are code blocks that store their values permanently in instance data blocks, so that they remain available after the block has been executed.

Add new and open

## 2.5 Optimized blocks

S7-1200/1500 controllers have optimized data storage. In optimized blocks all data types are automatically sorted. The sorting ensures that the data gaps between the data types are reduced to a minimum and that they are stored access-optimized for the processor.

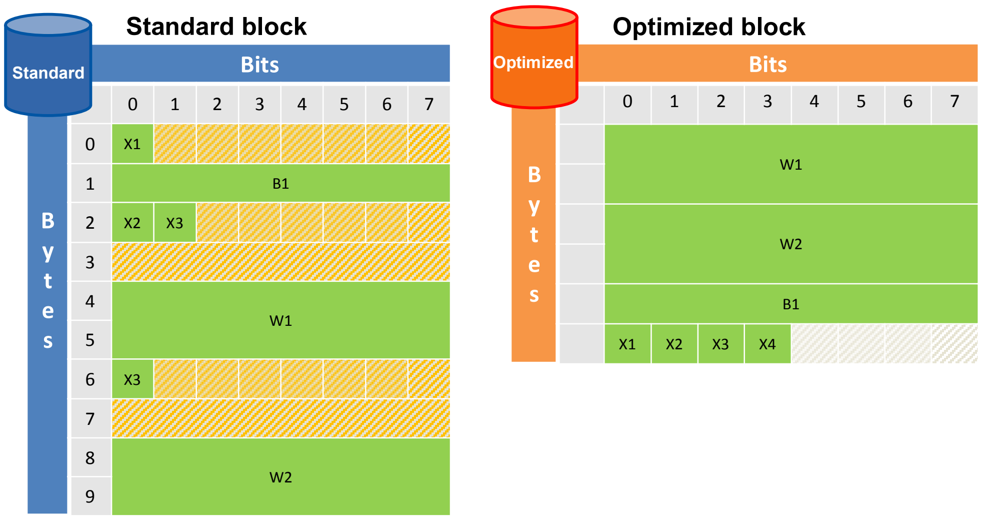
Non-optimized blocks only exist for reasons of compatibility in S7-1200/1500.

### Advantages

- The access is always as fast as possible, since the file storage is optimized by the system and is independent of the declaration.
- No danger of inconsistencies due to faulty, absolute accesses since the access is generally symbolic.
- Declaration changes do not lead to access errors since, for example, HMI accesses are performed symbolically.
- Individual tags can be specifically defined as “retain”.
- No settings in the instance data block are necessary. Everything is set in the assigned FB (e.g. retentivity).
- Memory reserves in the data block make it possible to change the actual values without any loss (see chapter 3.2.7 Downloading without reinitialization)

### 2.5.1 S7-1200: Setup of optimized blocks

Figure 2-4: Optimized block of S7-1200

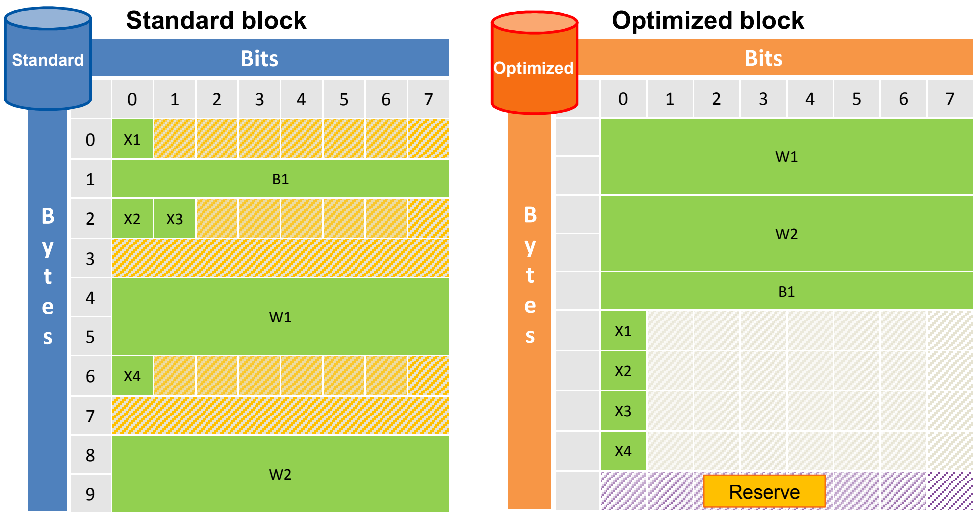


**Properties**

- No data gaps are formed since larger data types are located at the beginning of the block and smaller ones at the end.
- Only the symbolic access exists for optimized blocks.

### 2.5.2 S7-1500: Setup of optimized blocks

Figure 2-5: Optimized block of S7-1500



**Properties**

- No data gaps are formed since larger data types are located at the beginning of the block and smaller ones at the end.

- Fast access due the best possible storage in the processor (All data is stored in a way so that the processor of the S7-1500 can directly read or write all data with just one machine command).
- Boolean tags are stored as byte for faster access. The controller therefore does not have to mask the access.
- Optimized blocks have a memory reserves for reloading in running operation (see chapter 3.2.7 Downloading without reinitialization).
- Only the symbolic access exists for optimized blocks.

### 2.5.3 Best possible data storage in the processor on S7-1500

For reasons of compatibility to the first SIMATIC controllers the “Big-Endian” principle of data storage was adopted in the S7-300/400 controllers.

The new S7-1500 controller generation always accesses 4 byte (32 bit) in “Little-Endian” sequence due to the changed processor architecture. This results in the following system-specific properties.

Figure 2-6: Data access of a S7-1500 controller

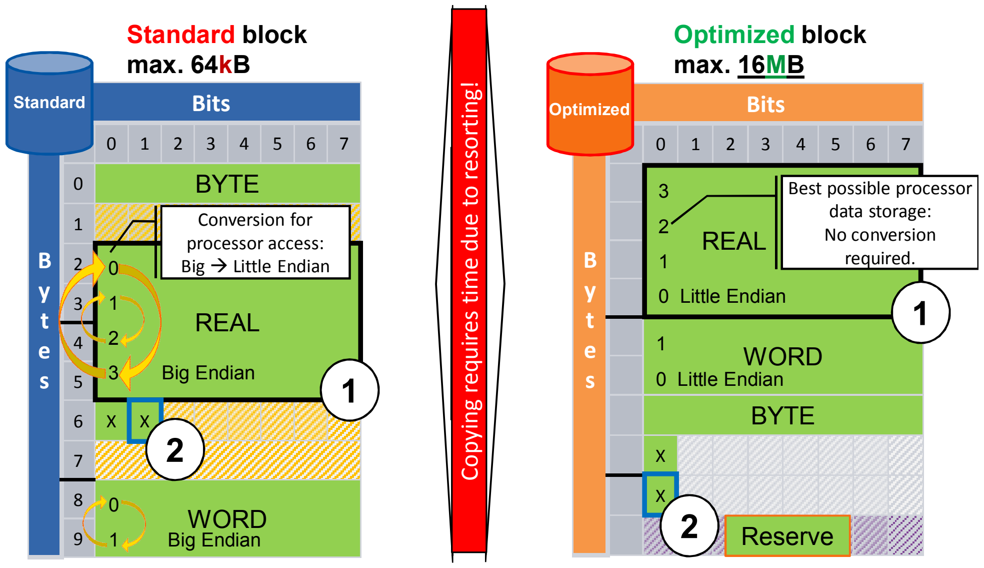


Table 2-1: Data access of a S7-1500 controller

| | Standard block | Optimized block |
|---|---|---|
| 1. | In the event of an unfavorable offset, the controller needs 2x16 bit accesses in order to be able to read a 4 byte value (e.g. REAL value).<br>In addition the bytes have to be changed. | The controller stores the tags, access optimized. An access is performed with 32 bit (REAL).<br>A changing of the bytes is not necessary. |
| 2. | The complete byte is read and masked per bit access.<br>The complete byte is blocked for any other access. | Each bit is assigned a byte.<br>When accessing, the controller does not have to mask the byte. |
| 3. | Maximum block size is 64kB. | Maximum block size can be up to 16MB. |

### Recommendation

- Always only use optimized blocks.
  - They do not require absolute addressing and can always be addressed with symbolic data (object related). Indirect addressing is also possible with symbolic data (see chapter 3.6.2 ARRAY data type and indirect field accesses).
  - The processing of optimized blocks in the controller is much faster than with standard blocks.
- Avoid the copying of data between optimized and non-optimized blocks. The required conversion between source and destination format requires high processing time.
- Avoid the assignment of optimized to non-optimized tags. Here, the storage formats are also converted.

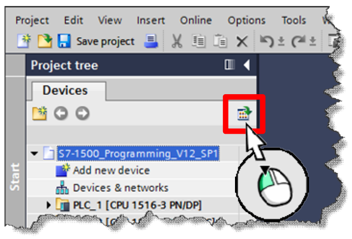
### Example: Setting optimized block access

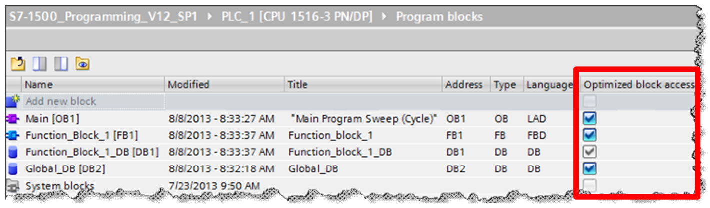
The optimized block accesses for all newly created blocks for S7-1200/1500 is enabled by default. Block access can be set for OBs, FBs and global DBs. For instance DBs, the setting depends on the respective FB.

The block access is not reset automatically when a block is migrated from a S7-300/400 controller to a S7-1200/1500. You can change the block access later on to “optimized block access”. You need to recompile the program after changing the block access. If you change the FBs to “optimized block access”, the assigned instance data blocks are automatically updated.

Follow the instructions below, in order to set the optimized block access.

Table 2-2: Setting optimized block access

| Step | Instruction |
|---|---|
| 1. | Click the “Maximizes/minimizes the Overview” button in the project navigation.   Project Edit View Insert Online Options Tools<br>Save project<br>Project tree<br>Devices<br>Start<br>S7-1500_Programming_V12_SP1<br>Add new device<br>Devices & networks<br>PLC_1 [CPU 1516-3 PN/DP] |
| 2. | Navigate to “Program blocks“. |

| Step | Instruction |
|---|---|
| 3. | This is where you see all blocks in the program and whether they are optimized or not. In this overview the “Optimized block access” status can be conveniently changed.<br><br>Note: Instance data blocks (here “Function_block_1_DB”) inherit the “optimized” status from the respective FB. This is why the “optimized” setting can only be changed on the FB. After the compilation of the project the DB accepts the status depending on the respective FB. |

S7-1500_Programming_V12_SP1 ‣ PLC_1 [CPU 1516-3 PN/DP] ‣ Program blocks

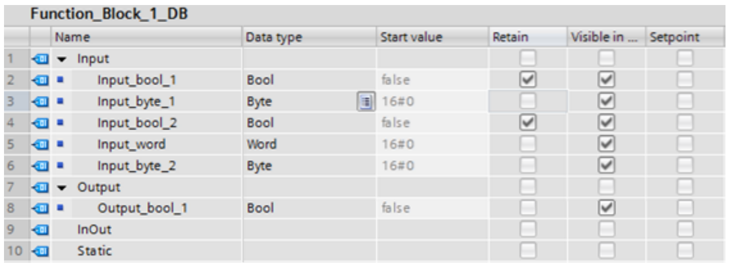
| Name | Modified | Title | Address | Type | Language | Optimized block access |
|---|---|---|---|---|---|---|
| Add new block | | | | | | ☐ |
| Main [OB1] | 8/8/2013 - 8:33:27 AM | "Main Program Sweep (Cycle)" | OB1 | OB | LAD | ✓ |
| Function_Block_1 [FB1] | 8/8/2013 - 8:33:37 AM | Function_block_1 | FB1 | FB | FBD | ✓ |
| Function_Block_1_DB [DB1] | 8/8/2013 - 8:33:37 AM | Function_block_1_DB | DB1 | DB | DB | ✓ |
| Global_DB [DB2] | 8/8/2013 - 8:32:18 AM | Global_DB | DB2 | DB | DB | ✓ |
| System blocks | 7/23/2013 9:50 AM | | | | | ☐ |

## Display of optimized and non-optimized blocks in the TIA Portal

In the two following figures the differences between an optimized and a non-optimized instance DB can be seen.

For a global DB there are the same differences.

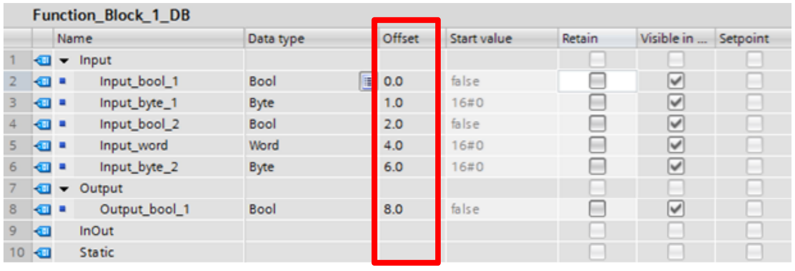
Figure 2-7: Optimized data block (without offset)



Function_Block_1_DB

| | Name | Data type | Start value | Retain | Visible in ... | Setpoint |
|---|---|---|---|---|---|---|
| 1 | ▼ Input | | | ☐ | ☐ | ☐ |
| 2 | Input_bool_1 | Bool | false | ✓ | ✓ | ☐ |
| 3 | Input_byte_1 | Byte | 16#0 | ☐ | ✓ | ☐ |
| 4 | Input_bool_2 | Bool | false | ✓ | ✓ | ☐ |
| 5 | Input_word | Word | 16#0 | ☐ | ✓ | ☐ |
| 6 | Input_byte_2 | Byte | 16#0 | ☐ | ✓ | ☐ |
| 7 | ▼ Output | | | ☐ | ☐ | ☐ |
| 8 | Output_bool_1 | Bool | false | ☐ | ✓ | ☐ |
| 9 | InOut | | | ☐ | ☐ | ☐ |
| 10 | Static | | | ☐ | ☐ | ☐ |

Figure 2-8: Non-optimized data blocks (with offset)



Function_Block_1_DB

| | Name | Data type | Offset | Start value | Retain | Visible in ... | Setpoint |
|---|---|---|---|---|---|---|---|
| 1 | ▼ Input | | | | ☐ | ☐ | ☐ |
| 2 | Input_bool_1 | Bool | 0.0 | false | ☐ | ✓ | ☐ |
| 3 | Input_byte_1 | Byte | 1.0 | 16#0 | ☐ | ✓ | ☐ |
| 4 | Input_bool_2 | Bool | 2.0 | false | ☐ | ✓ | ☐ |
| 5 | Input_word | Word | 4.0 | 16#0 | ☐ | ✓ | ☐ |
| 6 | Input_byte_2 | Byte | 6.0 | 16#0 | ☐ | ✓ | ☐ |
| 7 | ▼ Output | | | | ☐ | ☐ | ☐ |
| 8 | Output_bool_1 | Bool | 8.0 | false | ☐ | ✓ | ☐ |
| 9 | InOut | | | | ☐ | ☐ | ☐ |
| 10 | Static | | | | ☐ | ☐ | ☐ |

Table 2-3: Difference: optimized and non-optimized data block

| Optimized data block | Non-optimized data block |
|---|---|
| Optimized data blocks do **not** have an “offset” of the tags since the addressing is only symbolic. | Non-optimized blocks have an “offset” and can be addressed directly. |
| In optimized blocks **every** tag can be declared with “Retain”. | However, in non-optimized blocks only **all or no** tags can be declared with “Retain”. |

If you want to declare a tag in a global DB with retain, you can set this directly in the global DB. No data is retentive by default.

If you want to declare a tag in a global DB with retain, you can set this directly in the global DB.

### Access types for optimized and non-optimized blocks

The following table displays all access types to blocks.

Table 2-4: Access types

| Access type | Optimized block | Non-optimized block |
|---|---|---|
| Symbolic | ✓ | ✓ |
| Indexed (fields) | ✓ | ✓ |
| Slice accesses | ✓ | ✓ |
| AT instruction | ✗ (Alternatively: slice access) | ✓ |
| Direct absolute | ✗ (Alternatively: ARRAY with index) | ✓ |
| Indirect absolute (pointer) | ✗ (Alternatively: VARIANT / ARRAY with index) | ✓ |
| Downloading without reinitialization | ✓ | ✗ |

**Note**

You will find further information in the following entry:

What properties do you have to pay attention to in STEP 7 V11 for the instructions "READ_DBL" and "WRIT_DBL", when you are using DBs with optimized access?
http://support.automation.siemens.com/WW/view/en/51434748

## 2.6 Block sizes

For S7-1200/1500 controllers the maximum size of blocks was significantly increased in the main memory.

Table 2-5: Block sizes

| **Max. size and number** (regardless of the main memory size) | | S7 -300/400 | S7-1200 | S7-1500 |
|---|---|---|---|---|
| **DB** | Max. size | 64 kB | 64 kB | 64 kB (non-optimized)<br>**16 MB** (optimized) |
| | Max. number | 16.000 | 59.999 | 59.999 |
| **FC/FB** | Max. size | 64 kB | 64 kB | 512 kB |
| | Max. number | 7.999 | 65.535 | 65.535 |
| **FC / FB / DB** | Max. number | 4.096 (CPU319)<br>6.000 (CPU412) | 1.024 | 6.000 (CPU1516) |

**Recommendation**

- Use the DBs for S7-1500 controllers as data container of very large data volumes.
- Data volumes of > 64 kB can be stored in an optimized DB (max. size 16 MB) with S7-1500 controllers.

## 2.7 New data types for S7-1200/1500

S7-1200/1500 controllers support new data types in order to make programming more convenient. With the new 64 bit data types considerably larger and more accurate values can be used.

**Note**

You will find further information in the following entry:

How is the conversion of data types performed in the TIA Portal for the S7-1200/1500?
http://support.automation.siemens.com/WW/view/en/60546567

### 2.7.1 Elementary data types

Table 2-6: Integer data types

| Type | Size | Value range |
|---|---|---|
| USint | 8 bit | 0 .. 255 |
| SInt | 8 bit | -128 .. 127 |
| UInt | 16 bit | 0 .. 65535 |
| UDInt | 32 bit | 0 .. 4.3 million |
| ULInt* | 64 bit | 0 .. 18.4 billion |
| LInt* | 64 bit | -9.2 billion .. 9.2 billion |

* only for S7-1500

Table 2-7: Floating-point decimal data types

| Type | Size | Value range |
|---|---|---|
| Real | 32 bit (1 bit signs, 8 bit exponent, 23 bit mantissa), 8 digits | -3.40e+38 .. 3.40e+38 |
| LReal | 64 bit (1 bit signs, 11 bit exponent, 23 bit mantissa), 16 digits | -1.79e+308 .. 1.79e+308 |

### 2.7.2 Date_Time_Long data type

Table 2-8: Structure of DTL (Date_Time_Long)

| Year | Month | Day | Weekday | Hour | Minute | Second | Nanosecond |
|---|---|---|---|---|---|---|---|

DTL always reads the current system time. Access to the individual values is through the symbolic names (e.g. `My_Timestamp.Hour`)

**Advantages**

- All partial areas (e.g. Year, Month, …) can be addressed symbolically.

**Recommendation**

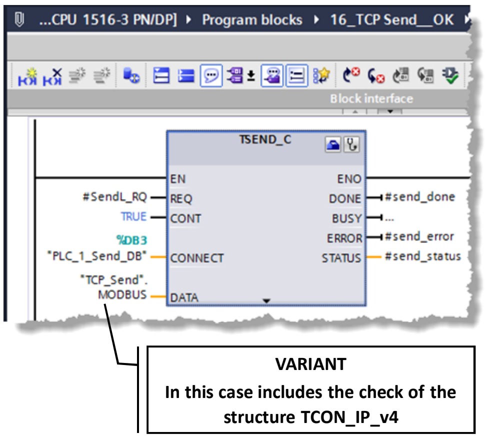
Use the new DTL data type instead of LDT and address symbolically.

### 2.7.3 VARIANT data type

A parameter of the VARIANT type is a pointer that can point to tags of different data types. In contrast to the ANY pointer the VARIANT is a pointer with type test. I.e. the target structure and source structure are checked at runtime and have to be identical.

VARIANT is used, for example, as input for communication blocks (TSEND_C).

Figure 2-9: VARIANT data type as input parameter for the TSEND_C instruction



**Advantages**

- Integrated type test
- Symbolic addressing for optimized blocks

**Recommendation**

- Use the VARIANT data type instead of the ANY pointer. Due to the integrated type test, errors are detected early on and due to the symbolic addressing the program code can be easily interpreted.
- As another alternative you can use the indexed ARRAYs for the ANY pointer (see chapter 3.6.2 ARRAY data type and indirect field accesses).

**Note**

You will find further information in the following entry:

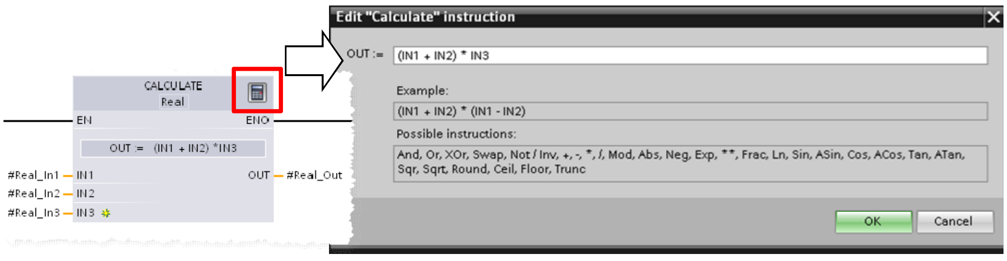
How can memory areas be copied in STEP 7 (TIA Portal)?
http://support.automation.siemens.com/WW/view/en/59886704

How do you program the "VARIANT" data type for indirect addressing for the S7-1200 in STEP 7 (TIA Portal) V11?
http://support.automation.siemens.com/WW/view/en/42603286

## 2.8 New CALCULATE instruction

With the CALCULATE instruction you can carry out mathematical calculations (e.g. (IN1 + IN2) * IN3) that are independent from the data type. The mathematical formula is programmed in the formula editor of the instruction.

Figure 2-10: CALCULATE instruction with formula editor



Edit "Calculate" instruction

OUT := (IN1 + IN2) * IN3

Example:
(IN1 + IN2) * (IN1 - IN2)

Possible instructions:
And, Or, XOr, Swap, Not / Inv, +, -, *, /, Mod, Abs, Neg, Exp, **, Frac, Ln, Sin, ASin, Cos, ACos, Tan, ATan, Sqr, Sqrt, Round, Ceil, Floor, Trunc

**Note** For more information refer to the Online Help of the TIA Portal with the “CALCULATE” instruction.

### Advantages

- A mathematical formula only needs one instruction
- Time saving due to simple configuration
- In SCL mathematical formula can be programmed even more clearly and efficiently

### Properties

- Supports bit sequences, integers, floating-point numbers
- Supports numerous mathematical functions (all basic arithmetic operations, trigonometric functions, rounding, logarithm, etc.)
- Supports any number of inputs

### Recommendation

- Always use the CALCULATE instruction for mathematical calculations instead of many calls of instructions, such as, e.g.  ADD, SUB, etc.

## 2.9 Symbolic and comments

### Advantages

You can make the code easy to understand and readable for your colleagues with the use of symbolic names and comments in your program.
The complete symbolic is saved together with the program code during the download to the controller and allows fast maintenance of the plant when no offline project is at hand.

### Recommendation

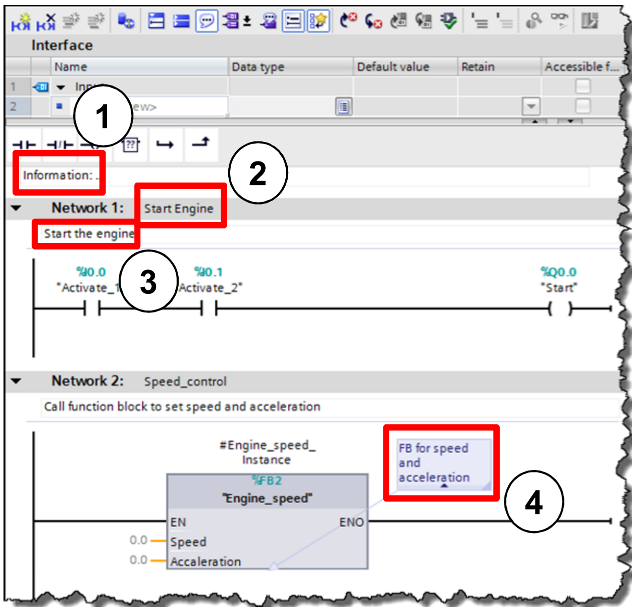
- Use the comments in the programs in order to improve readability.
- Design the program code in a way so that other users can understand the program straight away.

In the following example you can see the extensive options for commenting the program editors.

### Example

In the following figure you can see the options for commenting in the LAD editor (same functionality in FDB).

Figure 2-11: Commenting in the user program (LAD)



The following comments are possible:

1. Block comment
2. Network title comment
3. Network comment
4. Comment on instructions, blocks and functions (open, close, etc.)

In the programming languages SCL and STL, it can be commented with // in every row.

### Example

```
Filling level:= Radius * Radius * PI * height;   //
calculation of the filling level for medium tank
```

## 2.10 System constants

For S7-300/400 controllers the identification of hardware and software components is performed by logic address or diagnostic addresses.

For S7-1200/1500 the identification is by system constants. All hardware and software components (e.g. interfaces, modules, OBs, ...) of the S7-1200/1500 controllers have their own system constants. The system constants are automatically created during the setup of the device configuration for the central and distributed I/O.

### Advantages

- You can address via module names instead of hardware identification.

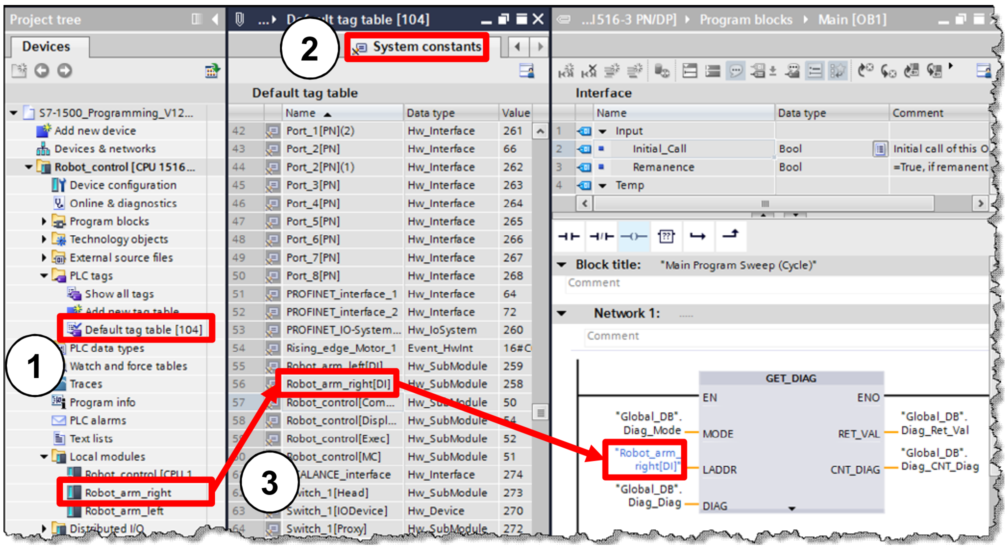
### Recommendation

- Assign function-related module names in order to identify them easily during programming.

### Example

In the following example you can see how system constants are used in the user program.

Figure 2-12: “System constants” in the user program



Default tag table

| | Name | Data type | Value |
|---|---|---|---|
| 42 | Port_1[PN](2) | Hw_Interface | 261 |
| 43 | Port_2[PN] | Hw_Interface | 66 |
| 44 | Port_2[PN](1) | Hw_Interface | 262 |
| 45 | Port_3[PN] | Hw_Interface | 263 |
| 46 | Port_4[PN] | Hw_Interface | 264 |
| 47 | Port_5[PN] | Hw_Interface | 265 |
| 48 | Port_6[PN] | Hw_Interface | 266 |
| 49 | Port_7[PN] | Hw_Interface | 267 |
| 50 | Port_8[PN] | Hw_Interface | 268 |
| 51 | PROFINET_interface_1 | Hw_Interface | 64 |
| 52 | PROFINET_interface_2 | Hw_Interface | 72 |
| 53 | PROFINET_IO-System... | Hw_IoSystem | 260 |
| 54 | Rising_edge_Motor_1 | Event_HwInt | 16#C |
| 55 | Robot_arm_left[DI] | Hw_SubModule | 259 |
| 56 | Robot_arm_right[DI] | Hw_SubModule | 258 |
| 57 | Robot_control[Com... | Hw_SubModule | 50 |
| 58 | Robot_control[Displ... | Hw_SubModule | 54 |
| 59 | Robot_control[Exec] | Hw_SubModule | 52 |
| 60 | Robot_control[MC] | Hw_SubModule | 51 |
| 61 | ALANCE_interface | Hw_Interface | 274 |
| 62 | witch_1[Head] | Hw_SubModule | 273 |
| 63 | Switch_1[IODevice] | Hw_Device | 270 |
| 64 | Switch_1[Proxy] | Hw_SubModule | 272 |

Interface

| | Name | Data type | Comment |
|---|---|---|---|
| 1 | Input | | |
| 2 | Initial_Call | Bool | Initial call of this O |
| 3 | Remanence | Bool | =True, if remanent |
| 4 | Temp | | |

1. System constants of a controller can be found in the “PLC tags – Default tag table” folder.
2. The system constants are in a separate list in the “Default tag table”.
3. In this example the symbolic name “Robot_arm_left” was assigned for a DI module.
   You can also find the module under this name in the system constant table.
   In the user program “Robot_arm_left” is interconnected with the “GET_DIAG” diagnostic block.

**Note**

You will find further information in the following entry:

What meaning do the system constants have for the S7-1200/1500 in STEP 7 (TIA Portal)?
http://support.automation.siemens.com/WW/view/en/78782836

## 2.11 Internal reference ID for controller and HMI tags

STEP 7, WinCC, Startdrive, Safety and others integrate into the joint data base of the TIA Portal engineering framework. Changes of data are automatically accepted in all the locations in the user program, independent from whether this happens in a controller, a panel or a drive. Therefore no data inconsistencies can occur.

If you create a tag, the TIA Portal automatically creates a unique reference ID. The reference ID cannot be viewed or programmed by you. This procedure is internal referencing. When changing tags (address), the reference ID remains unchanged.

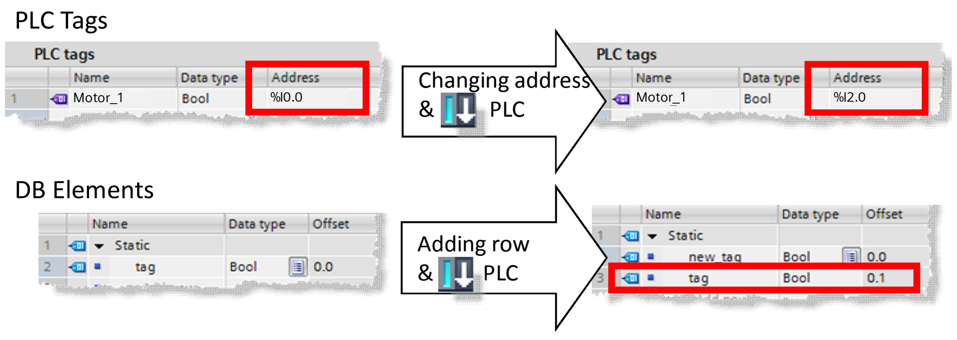
### Advantages

- You can rewire tags without changing internal relations. The communication between controller, HMI and drive also remains unchanged.
- The volume of the transferred data and the usage of main memory is independent from the length of the symbolic tag names.
- The length of the symbolic name does not have an influence on the communication load between controller and HMI.

### Properties

If you change addresses of PLC tags, you only have to reload the controller. It is not necessary to reload the HMI devices, since internally, the system addresses with the reference IDs (see Figure 2-13: Changing address or adding row).

Figure 2-13: Changing address or adding row



PLC tags

| | Name | Data type | Address |
|---|---|---|---|
| 1 | Motor_1 | Bool | %I0.0 |

PLC tags

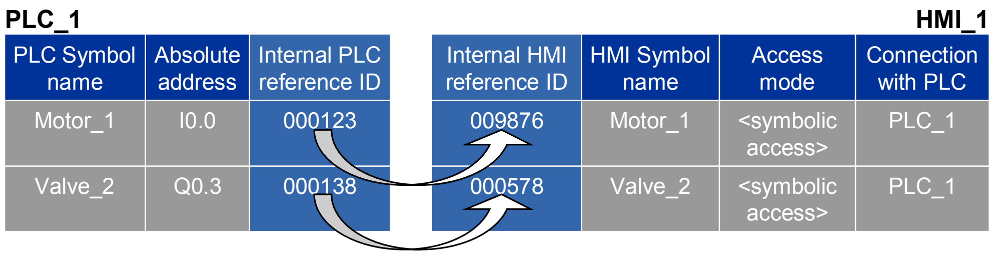
| | Name | Data type | Address |
|---|---|---|---|
| | Motor_1 | Bool | %I2.0 |

| | Name | Data type | Offset |
|---|---|---|---|
| 1 | ▼ Static | | |
| 2 | tag | Bool | 0.0 |

| | Name | Data type | Offset |
|---|---|---|---|
| 1 | ▼ Static | | |
| | new_tag | Bool | 0.0 |
| 3 | tag | Bool | 0.1 |

In the figure below the internal reference to the data is displayed schematically.

Figure 2-14: Internal reference ID for PLC and HMI



| PLC_1 | | | | | | HMI_1 |
|---|---|---|---|---|---|---|
| PLC Symbol name | Absolute address | Internal PLC reference ID | Internal HMI reference ID | HMI Symbol name | Access mode | Connection with PLC |
| Motor_1 | I0.0 | 000123 | 009876 | Motor_1 | <symbolic access> | PLC_1 |
| Valve_2 | Q0.3 | 000138 | 000578 | Valve_2 | <symbolic access> | PLC_1 |

**NOTE** The ID will be invalid if the name is changed, type is changed or tag is deleted

## 2.12 STOP mode in the event of errors

In comparison to S7-300/400 there are fewer criteria with the S7-1200/1500 that lead to the “STOP” mode.

Due to the changed consistency check in the TIA Portal, the “STOP” mode for S7-1200/1500 controllers can already be excluded in advance in most cases. The consistency of program blocks is already checked when compiling in the TIA Portal. This approach makes the S7-1200/1500 controllers more fault tolerant to errors than their predecessors.

### Advantages

There are only three fault situations that put the S7-1200/1500 controllers into the STOP mode. This makes the programming of the error management clearer and easier.

### Properties

Table 2-9: Responses to errors of S7-1200/1500

| | Error | S7-1200 | S7-1500 |
|---|---|---|---|
| 1. | Cycle monitoring time exceeded once | RUN | **STOP**, when OB80 is not configured |
| 2. | Cycle monitoring time exceeded twice | **STOP** | **STOP** |
| 3. | Programming errors | RUN | **STOP**, when OB121 is not configured |

Error OBs:

- OB80 “Time error interrupt” is called by the operating system when the maximum cycle time of the controller is exceeded.
- OB121 “Programming error” is called by the operating system when an error occurs during program execution.

For every error, in addition, an entry is automatically created in the diagnostic buffer.

**Note** For S7-1200/1500 controllers there are other programmable error OBs (diagnostic error, module rack failure, etc.).

More information on error responses of S7-1200/1500 can be found in the online help of the TIA Portal under “Events and OBs”.

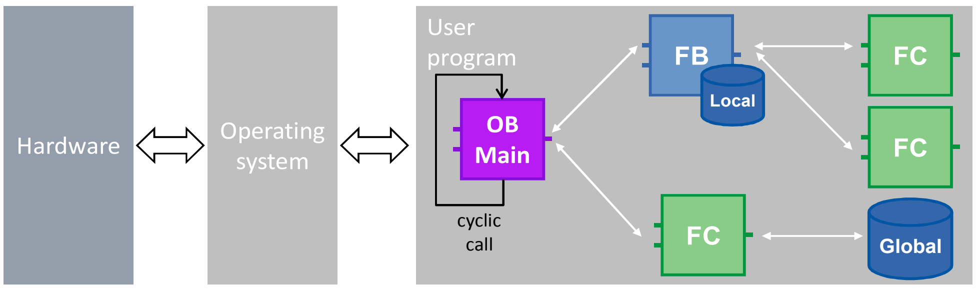
# 3 General Programming

## 3.1 Operating system and user program

SIMATIC controllers consist of operating system and user program.

- The operating system organizes all functions and sequences of the controller that are not connected with a specific control task (e.g. handling of restart, updating of process image, calling the user program, error handling, memory management, etc.). The operating system is an integral part of the controller.
- The user program includes all blocks that are required for the processing of your specific automation task. The user program is programmed with program blocks and loaded onto the controller.

Figure 3-1: Operating system and user program



For SIMATIC controllers the user program is always executed cyclically. The “Main” cycle OP already exists in the “Program blocks” folder after a controller was created in STEP 7. The block is processed by the controller and recalled in an infinite loop.

## 3.2 Program blocks

In STEP 7 (TIA Portal) there are all familiar block types from the previous STEP 7 versions:

- Organization blocks
- Function blocks
- Functions
- Data blocks

Experienced STEP 7 users will know their way around straight away and new users can very easily get familiar with the programming.

**Advantages**

- You can give your program a good and clear structure with the different block types.
- Due to a good and structured program you get many function units that can be multiply reused within a project and also in other projects. These function units then usually only differ by a different configuration (see chapter 3.2.8 Reusability of blocks).

- You project or your plant becomes more transparent. I.e. error states in a plant can be more easily detected, analyzed and removed. I.e. the maintainability of your plant becomes easier. This is also the case for errors in programming.

**Recommendation**

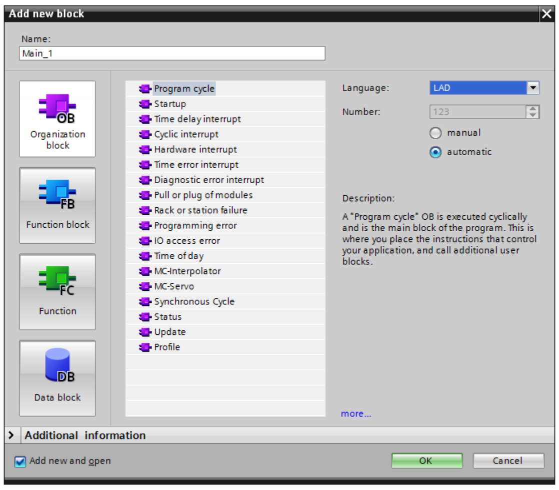
- Structure your automation task.
- Divide the entire function of your plant into individual areas and form sub-function units. Divide these function units again into smaller units and functions. Divide until you get functions that you can use several times with different parameters.
- Specify the interfaces between the function units. Define the unique interfaces for functionalities that are to be delivered by “external companies”.

All organization blocks, function blocks and functions can be programmed with the following languages:

- Contact plan (KOP or LAD)
- Function plan (FUP or FBD)
- Structured Control Language (SCL)
- Graph (GRAPH), only S7-1500, planned for S7-1200
- Instruction list (AWL or STL), only S7-1500

## 3.2.1 Organization blocks (OB)

Figure 3-2: “Add new block” dialog (OB)


Add new block

Name: Main_1

Language: LAD
Number: 123
( ) manual
(●) automatic

Description:
A "Program cycle" OB is executed cyclically and is the main block of the program. This is where you place the instructions that control your application, and call additional user blocks.

OBs are the interface between the operating system and the user program. They are called by the operating system and control, e.g. the following processes:

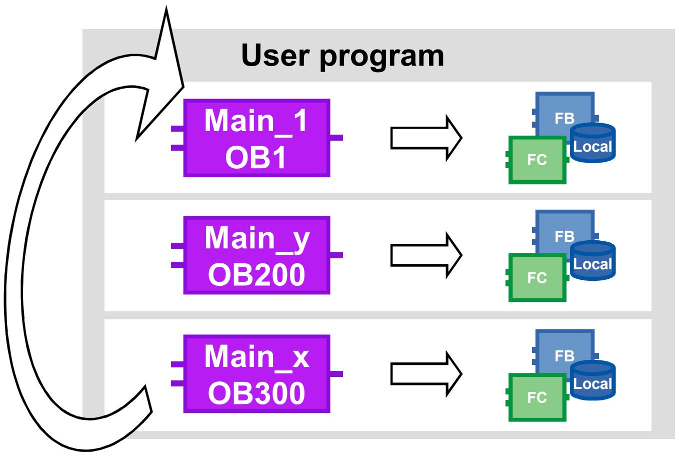
- Startup behavior of the controller
- Cyclic program processing
- Interrupt-controlled program processing
- Error handling

Depending on the controller a number of different OB types are available.

**Properties**

- OBs are called by the operating system of the controller
- Several Main OBs can be created in a program. The OBs are processed sequentially by OB number.

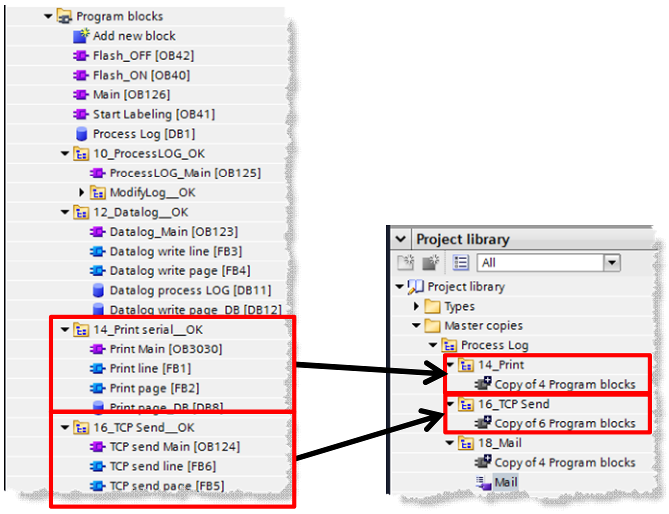
Figure 3-3: Using several Main OBs



**Recommendation**

- Encapsulate the different program parts which should maybe be replaceable from controller to controller, into several Main OBs.
- Avoid the communication between the different Main OBs. They can then be used independent from each other. If you nevertheless exchange data between the individual main OBs, use the global DBs (see chapter 4.2 No bit memory but global data blocks).
- Divide all program parts that belong to each other into folders and store them for reusability in the project or global library.

Figure 3-4: Storing program parts in order in the project library



```
Program blocks
    Add new block
    Flash_OFF [OB42]
    Flash_ON [OB40]
    Main [OB126]
    Start Labeling [OB41]
    Process Log [DB1]
    10_ProcessLOG_OK
        ProcessLOG_Main [OB125]
        ModifyLog__OK
    12_Datalog__OK
        Datalog_Main [OB123]
        Datalog write line [FB3]
        Datalog write page [FB4]
        Datalog process LOG [DB11]
        Datalog write page_DB [DB12]
    14_Print serial__OK
        Print Main [OB3030]
        Print line [FB1]
        Print page [FB2]
        Print page_DB [DB8]
    16_TCP Send__OK
        TCP send Main [OB124]
        TCP send line [FB6]
        TCP send page [FB5]
```

```
Project library
All
Project library
    Types
    Master copies
        Process Log
            14_Print
                Copy of 4 Program blocks
            16_TCP Send
                Copy of 6 Program blocks
            18_Mail
                Copy of 4 Program blocks
                Mail
```

For further information, please refer to chapter 3.7 Libraries.

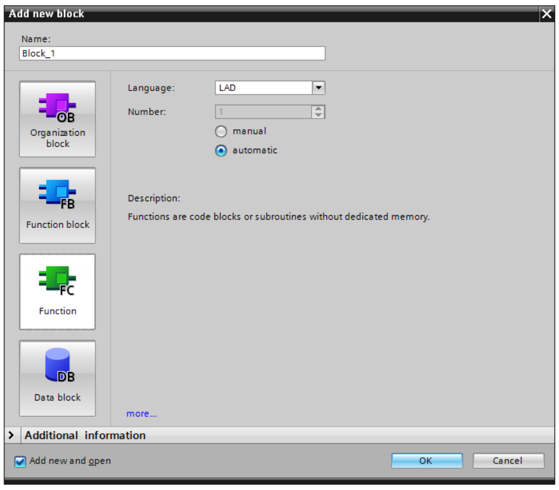
**Note**

You will find further information in the following entry:

Which organization blocks can be used in STEP 7 (TIA Portal)?
http://support.automation.siemens.com/WW/view/en/58235745

## 3.2.2 Functions (FC)

Figure 3-5: “Add new block” dialog (FC)



Add new block

Name: Block_1

Language: LAD

Number: 1

manual [✓ automatic]

Description:

Functions are code blocks or subroutines without dedicated memory.

[✓ Add new and open]

FCs are blocks without cyclic data storages. This is why the values of block parameters cannot be saved until the next call and have to be provided with actual parameters when called.

### Properties

- FCs are blocks without cyclic data storages.
- Temporary and out tags are lost after the processing of the function. They are preassigned with an undefined value when calling the function in non-optimized blocks, and for optimized blocks they are preassigned with “0” (S7-1500 and S7-1200 firmware V4).
- In order to permanently save the data of an FC, the functions of the global data blocks are available.
- FCs can have several outputs.
- The function value can be directly reused in SCL in a formula.

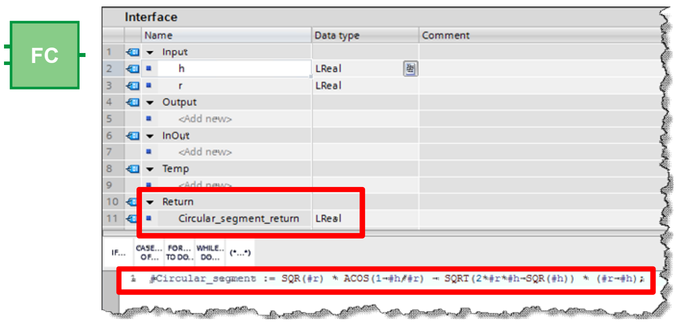
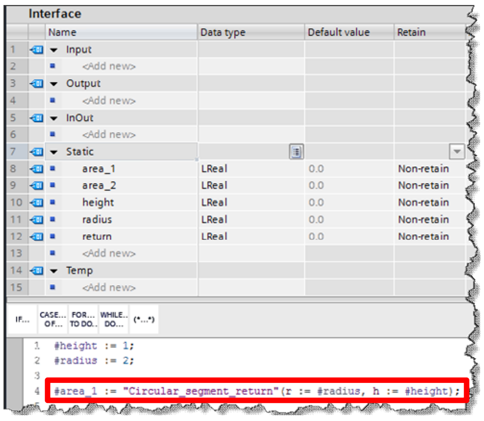
### Recommendation

- Use the functions for continuously recurring applications that are called several times in different locations of the user program.
- Use the option to directly reuse the function value in SCL.
  ```
  <Operand> := <FC name> (parameter list)
  ```

### Example

In the following example a mathematical formula is programmed in a FC. The result of the calculation is directly declared as return value and the function value can be directly reused.

Table 3-1: Reusing function value

| Step | Instruction |
|---|---|
| 1. | Create an FC with the mathematical formula (circular segment) and define the “Return” value as the result for the formula.  |
| 2. | Call the FC with the circular segment calculation in any block (SCL). `<Operand> := <FC name> (parameter list);`  |

Interface

|  | Name | Data type | Comment |
|---|---|---|---|
| 1 | Input |  |  |
| 2 | h | LReal |  |
| 3 | r | LReal |  |
| 4 | Output |  |  |
| 5 | <Add new> |  |  |
| 6 | InOut |  |  |
| 7 | <Add new> |  |  |
| 8 | Temp |  |  |
| 9 | <Add new> |  |  |
| 10 | Return |  |  |
| 11 | Circular_segment_return | LReal |  |

```
#Circular_segment := SQR(#r) * ACOS(1-#h/#r) - SQRT(2*#r*#h-SQR(#h)) * (#r-#h);
```

Interface

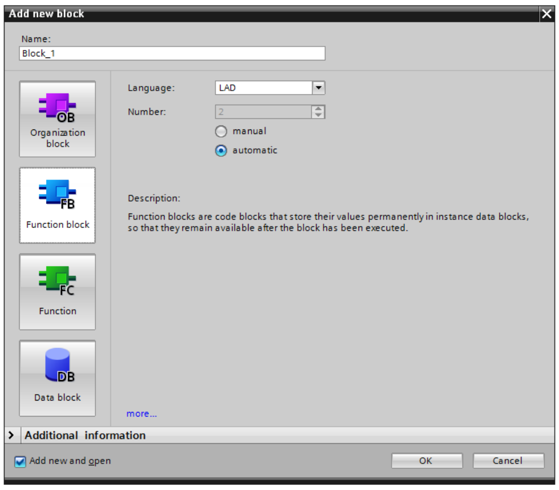
|  | Name | Data type | Default value | Retain |
|---|---|---|---|---|
| 1 | Input |  |  |  |
| 2 | <Add new> |  |  |  |
| 3 | Output |  |  |  |
| 4 | <Add new> |  |  |  |
| 5 | InOut |  |  |  |
| 6 | <Add new> |  |  |  |
| 7 | Static |  |  |  |
| 8 | area_1 | LReal | 0.0 | Non-retain |
| 9 | area_2 | LReal | 0.0 | Non-retain |
| 10 | height | LReal | 0.0 | Non-retain |
| 11 | radius | LReal | 0.0 | Non-retain |
| 12 | return | LReal | 0.0 | Non-retain |
| 13 | <Add new> |  |  |  |
| 14 | Temp |  |  |  |
| 15 | <Add new> |  |  |  |

```
#height := 1;
#radius := 2;

#area_1 := "Circular_segment_return"(r := #radius, h := #height);
```

### 3.2.3 Function blocks (FB)

Figure 3-6: “Add new block” dialog (FB)



Add new block

Name: Block_1

Language: LAD

Number: 2

manual [✓ automatic]

Description:

Function blocks are code blocks that store their values permanently in instance data blocks, so that they remain available after the block has been executed.

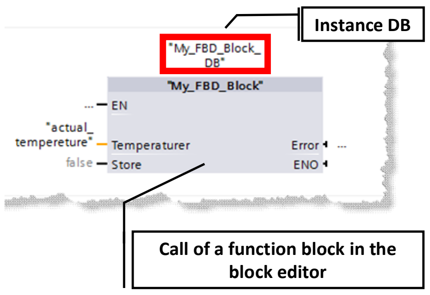
more...

Additional information

[✓] Add new and open

FBs are blocks with cyclic data storage, in which values are permanently stored. The cyclic data storage is realized in an instance DB.

Figure 3-7: Calling a function block



#### Properties

- FCs are blocks with cyclic data storage.
- Temporary and out tags are lost after the processing of the function. They are preassigned with an undefined value when calling the function in non-optimized blocks, and for optimized blocks they are preassigned with “0” (S7-1500 and S7-1200 firmware V4).
- Static tags keep the value from cycle to cycle

#### Recommendation

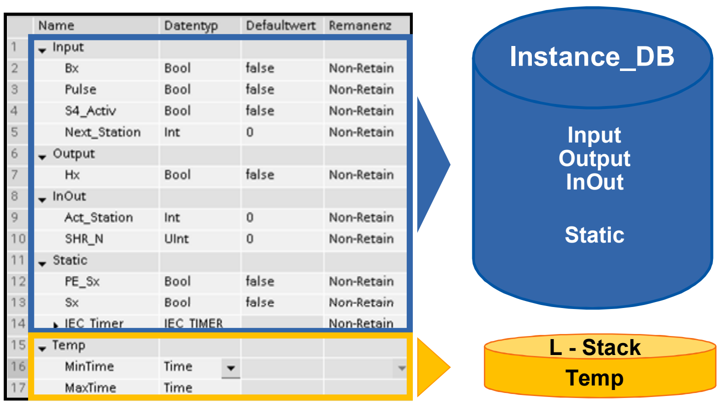
- Use the function blocks in order to create subprograms and structure the user program. A function block can also be called several times in different locations of the user program. This makes programming of frequently recurring program parts easier.

### 3.2.4 Instance data block

The call of a function block is called instance. The data with which the instance is working is saved in an instance DB.

Instance DBs are always created according to the specifications in the FB interface and can therefore not be changed in the instance DB.

Figure 3-8: Structure of the interfaces of an FB



| | Name | Datentyp | Defaultwert | Remanenz |
|---|---|---|---|---|
| 1 | Input | | | |
| 2 | Bx | Bool | false | Non-Retain |
| 3 | Pulse | Bool | false | Non-Retain |
| 4 | S4_Activ | Bool | false | Non-Retain |
| 5 | Next_Station | Int | 0 | Non-Retain |
| 6 | Output | | | |
| 7 | Hx | Bool | false | Non-Retain |
| 8 | InOut | | | |
| 9 | Act_Station | Int | 0 | Non-Retain |
| 10 | SHR_N | UInt | 0 | Non-Retain |
| 11 | Static | | | |
| 12 | PE_Sx | Bool | false | Non-Retain |
| 13 | Sx | Bool | false | Non-Retain |
| 14 | IEC_Timer | IEC_TIMER | | Non-Retain |
| 15 | Temp | | | |
| 16 | MinTime | Time | | |
| 17 | MaxTime | Time | | |

The instance DB consists of a permanent memory with the interfaces input, output, InOut and static. In a volatile memory (L stack) temporary tags are stored. The L stack is always only valid for one cycle. I.e. temporary tags have to be initialized in each cycle.

**Properties**

- Instance DBs are always assigned to a FB.
- Instance DBs do not have to be created manually in the TIA Portal and are created automatically when calling an FB.
- The structure of the instance DB is specified in the appropriate FB and can only be changed there.

**Recommendation**

- Program it in a way so that the data of the instance DB can only be changed by the appropriate FB. This is how you can guarantee that the block can be used universally in all kinds of projects.

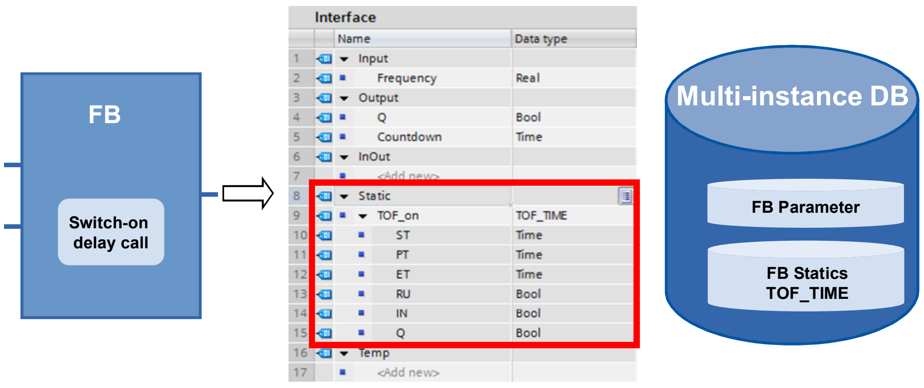
For further information, please refer to chapter 3.4 Block interfaces as data exchange.

### 3.2.5 Multi-instances

With multi-instances called function blocks can store their data in the instance data block of the called function block. I.e. if another function block is called in a function block, it saves its data in the instance DB of the higher-level FBs.

The following figure shows an FB that uses another FB (“IEC Timer”). All data is saved in a multi instance DB.

Figure 3-9: Multi-instances



Interface

| | Name | Data type |
|---|---|---|
| 1 | Input | |
| 2 | Frequency | Real |
| 3 | Output | |
| 4 | Q | Bool |
| 5 | Countdown | Time |
| 6 | InOut | |
| 7 | <Add new> | |
| 8 | Static | |
| 9 | TOF_on | TOF_TIME |
| 10 | ST | Time |
| 11 | PT | Time |
| 12 | ET | Time |
| 13 | RU | Bool |
| 14 | IN | Bool |
| 15 | Q | Bool |
| 16 | Temp | |
| 17 | <Add new> | |

## Advantages

- Reusability
- Multiple calls are possible
- Clearer program with fewer instance DBs
- Simple copying of programs
- Good options for structuring during programming

## Properties

- Multi-instances are memory areas within instance DBs.

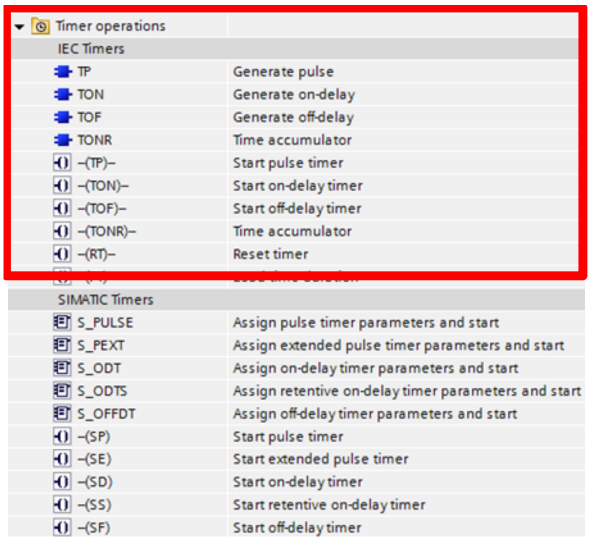
## Recommendation

- Use multi-instances in order to reduce the number of instances. You can therefore create reusable and clear user programs.

## Example

If you require the time and counter function, use the “IEC Timer” blocks and the “IEC Counter” blocks instead of the absolutely addressed SIMATIC Timer. If possible, also always use multi-instances here. This keeps the number of blocks in the user program low.

Figure 3-10: Library of the IEC Timer



| | |
|---|---|
| Timer operations | |
| IEC Timers | |
| TP | Generate pulse |
| TON | Generate on-delay |
| TOF | Generate off-delay |
| TONR | Time accumulator |
| –(TP)– | Start pulse timer |
| –(TON)– | Start on-delay timer |
| –(TOF)– | Start off-delay timer |
| –(TONR)– | Time accumulator |
| –(RT)– | Reset timer |
| SIMATIC Timers | |
| S_PULSE | Assign pulse timer parameters and start |
| S_PEXT | Assign extended pulse timer parameters and start |
| S_ODT | Assign on-delay timer parameters and start |
| S_ODTS | Assign retentive on-delay timer parameters and start |
| S_OFFDT | Assign off-delay timer parameters and start |
| –(SP) | Start pulse timer |
| –(SE) | Start extended pulse timer |
| –(SD) | Start on-delay timer |
| –(SS) | Start retentive on-delay timer |
| –(SF) | Start off-delay timer |

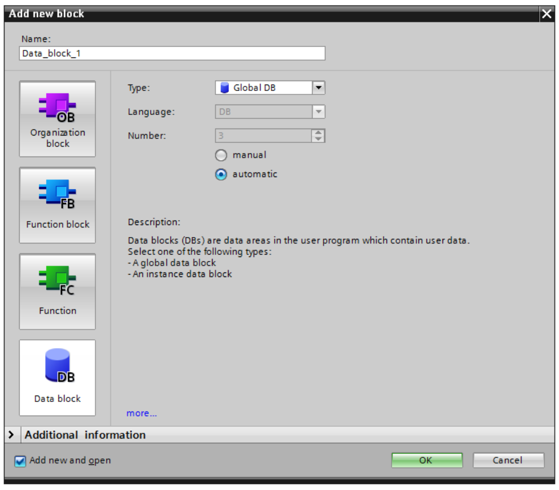
**Note**

You will find further information in the following entry:

How do you declare the timers and counters for the S7-1500 in STEP 7 (TIA Portal) V12?
http://support.automation.siemens.com/WW/view/en/67585220

### 3.2.6 Global data blocks (DB)

Figure 3-11: “Add new block” dialog (DB



Add new block

Name: Data_block_1

Type: Global DB
Language: DB
Number: 3
( ) manual
(●) automatic

Description:
Data blocks (DBs) are data areas in the user program which contain user data.
Select one of the following types:
- A global data block
- An instance data block

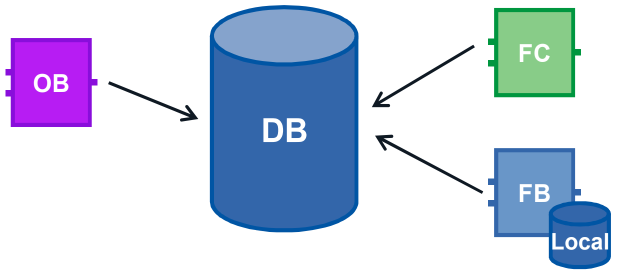
more...

Additional information

[✓] Add new and open

Variable data is located in data blocks that are available to the entire user program.

Figure 3-12: Global DB as central data memory



**Advantages**

- Well structured memory area
- High access speed

**Properties**

- All blocks in the user program can access global DBs.
- The structure of the global DBs can be arbitrarily made up of all data types.
- Global DBs are either created via the program editor or according to a previously created “user-defined PLC data type" (see chapter 3.6.3 STRUCT data type and PLC data types).

**Recommendation**

- Use the global DBs when data is used in different program parts or blocks.
- Use the PLC data types in order to specify a structure for a data block. The PLC data type can be used for any number of DBs. You can easily and conveniently create as many DBs of the same structure and adjust them centrally on the PLC data type (see chapter 3.6.3 STRUCT data type and PLC data types).

**Note**

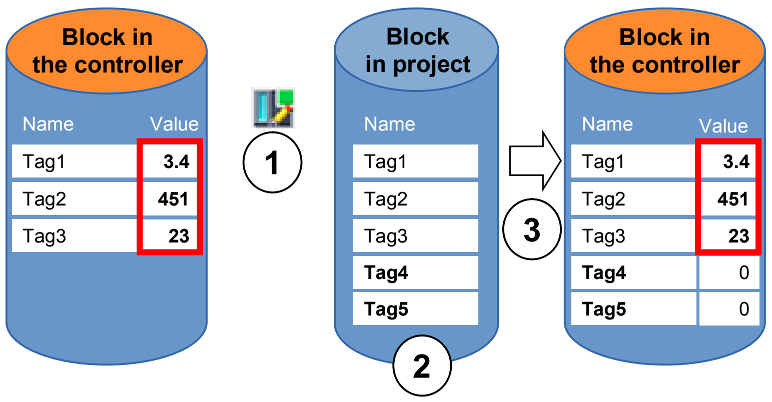
You will find further information in the following entry:

What access types, value columns and operating options are there for the global data blocks in STEP 7 V12?
http://support.automation.siemens.com/WW/view/en/68015631

### 3.2.7 Downloading without reinitialization

In order to change user programs that already run in a controller, S7-1200 (firmware V4.0) and S7-1500 controllers offer the option to expand the interfaces of optimized function or data blocks during operation. You can load the changed blocks without setting the controller to STOP and without influencing the actual values of already loaded tags.

Figure 3-13: Downloading without reinitialization



Execute the following steps whilst the controller is in RUN mode.

1. Enable “Downloading without reinitialization”
2. Insert tags in existing block
3. Load block into controller

**Advantages**

- Reloading of tags without interrupting the running process. The controller stays in “RUN” mode.

**Properties**

- It is assumed that a memory reserve has been defined for block.
- Downloading without reinitialization is only possible for optimized blocks.
- The current values are not initialized and maintain the current value.
- A block with reserve requires more memory space in the controller.
- The memory reserve depends on the work memory of the controller; however it is max. 2 MB.
- By default the memory reserve is set to 100 byte.
- The memory reserve is defined individually for every block.

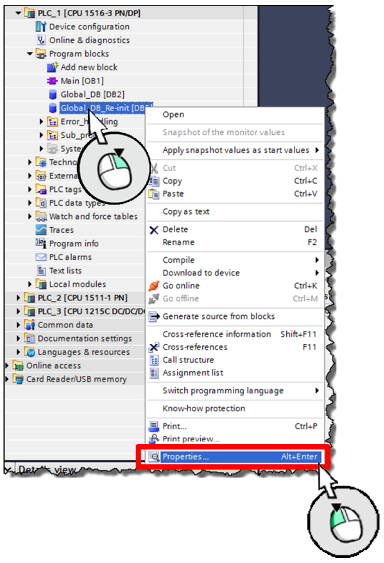
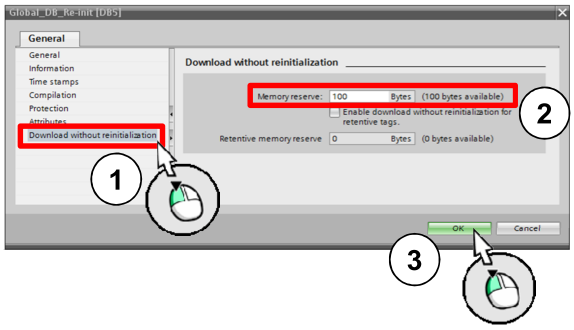
**Recommendation**

- Define a memory reserve for blocks that are to be expanded during commissioning (e.g. test blocks). The commissioning process is therefore not interrupted. The blocks can be variably expanded.

**Example**

The following table describes how you can set the memory reserve for the downloading without reinitializing.

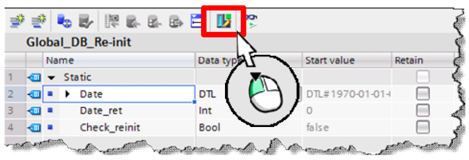
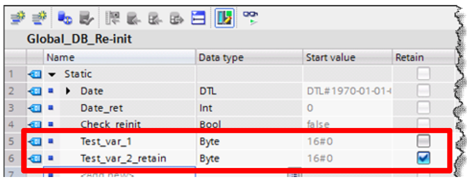
Table 3-2: Setting memory reserve

| Step | Instruction |
|---|---|
| 1. | Right-click any optimized block in the project navigator and select “Properties”.   PLC_1 [CPU 1516-3 PN/DP]<br>Device configuration<br>Online & diagnostics<br>Program blocks<br>Add new block<br>Main [OB1]<br>Global_DB [DB2]<br>Global_DB_Re-init [DB5]<br>PLC tags<br>PLC data types<br>Watch and force tables<br>Traces<br>Program info<br>PLC alarms<br>Text lists<br>Local modules<br>PLC_2 [CPU 1511-1 PN]<br>Common data<br>Documentation settings<br>Languages & resources<br>Online access<br>Card Reader/USB memory<br>Open<br>Snapshot of the monitor values<br>Apply snapshot values as start values<br>Cut Ctrl+X<br>Copy Ctrl+C<br>Paste Ctrl+V<br>Copy as text<br>Delete Del<br>Rename F2<br>Compile<br>Download to device<br>Go online Ctrl+K<br>Go offline Ctrl+M<br>Generate source from blocks<br>Cross-reference information Shift+F11<br>Cross-references F11<br>Call structure<br>Assignment list<br>Switch programming language<br>Know-how protection<br>Print... Ctrl+P<br>Print preview...<br>Properties... Alt+Enter |
| 2. |  Global_DB_Re-init [DB5]<br>General: General, Information, Time stamps, Compilation, Protection, Attributes, Download without reinitialization<br>Download without reinitialization<br>Memory reserve: 100 Bytes (100 bytes available)<br>☐ Enable download without reinitialization for retentive tags.<br>Retentive memory reserve 0 Bytes (0 bytes available)<br> <br><br>1. Click “Download without reinitialization”.<br>2. Enter the desired memory reserve for “Memory reserve”.<br>3. Confirm with "OK". |

In the following example it is displayed how to download without reinitialization.

Table 3-3 Downloading without reinitialization

| Step | Instruction |
|---|---|
| 1. | Requirement: a memory reserve has to be set (see above) |
| 2. | Open, e.g. an optimized global DB. |

| Step | Instruction |
|---|---|
| 3. | Click the “Download without reinitialization” button and confirm the dialog with “OK”  |
| 4. | Add a new tag (retentive tags are also possible).  |
| 5. | Download the block to the controller. |
| 6. | Result:<br>• Actual values of the block remain |

Global_DB_Re-init

| | Name | Data type | Start value | Retain |
|---|---|---|---|---|
| 1 | Static | | | ☐ |
| 2 | Date | DTL | DTL#1970-01-01- | ☐ |
| 3 | Date_ret | Int | 0 | ☐ |
| 4 | Check_reinit | Bool | false | ☐ |

Global_DB_Re-init

| | Name | Data type | Start value | Retain |
|---|---|---|---|---|
| 1 | Static | | | ☐ |
| 2 | Date | DTL | DTL#1970-01-01- | ☐ |
| 3 | Date_ret | Int | 0 | ☐ |
| 4 | Check_reinit | Bool | false | ☐ |
| 5 | Test_var_1 | Byte | 16#0 | ☐ |
| 6 | Test_var_2_retain | Byte | 16#0 | ✓ |

**Note**

Further information can be found in the online help of the TIA Portal under “Loading block extensions without reinitialization”.

### 3.2.8 Reusability of blocks

The block concept offers you a number of options to program in a structured and effective way.

**Advantages**

- Blocks can be used universally in any location of the user program.
- Blocks can be used universally in different projects.
- When every block receives an independent task, a clear and well structured user program is automatically created.
- There are clearly fewer sources of errors and is makes simple error diagnostic possible.

**Recommendation**

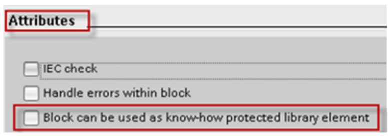
If you want to reuse the block, please note the following recommendations:

- Always look at blocks as encapsulated functions. I.e. each block represents a completed partial task of the entire user program.
- Use several cyclic Main OBs to group the plant parts.
- Always execute a data exchange between the blocks via its interfaces and not via its instances (see chapter 3.4.1 Block interfaces as data exchange).
- Do not use project-specific data and avoid the following block contents:

- Access to global DBs and use of individual instance DBs
- Access to tags
- Access to global constants

- Reusable blocks have the same requirements as know-how-protected blocks in libraries. This is why you have to check the blocks for reusability based on the “Block can be used as know-how protected library element” block property. Compile the block before the check.

Figure 3-14: Block attributes


Attributes

- [ ] IEC check
- [ ] Handle errors within block
- [ ] Block can be used as know-how protected library element

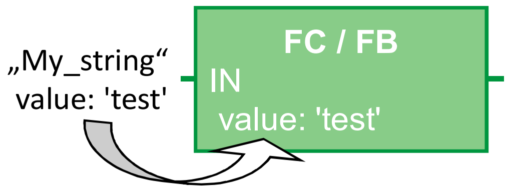
## 3.3 Block interface types

FBs and FCs have three different interface types: In, InOut and Out. Via these interface types the blocks are provided with parameters. The parameters are processed and output again in the block. There are two different options for this parameter transfer.

### 3.3.1 Call-by-value with In interface type

When calling the block, the value of the actual parameter is copied onto the input parameter of the block for the In interface type. For this, additional memory is required.

Figure 3-15: Copying of the value to the input parameter



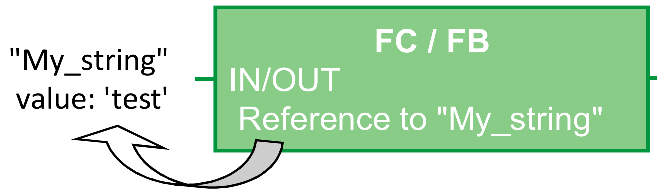
**Properties**

- Each block displays the same behavior with connected parameters
- Values are copied when calling the block

### 3.3.2 Call-by-reference with InOut interface type

When calling the block the address of the actual parameter of the Input parameter is referenced for the InOut interface type. For this, no additional memory is required.

Figure 3-16: Referencing the value (pointer to data storage of the parameter)



**Properties**

- Each block displays the same behavior with connected parameters
- Actual parameters are referenced with the block call

**Recommendation**

- Generally use the InOut interface type for structured tags (e.g. of the ARRAY, STRUCT, STRING, type…) in order to avoid enlarging the required data memory unnecessarily.

## 3.4 Storage concept

For STEP 7 there is generally the difference between the global and local memory area. The global memory area is available for each block in the user program. The local memory area is only available within the respective block.

### 3.4.1 Block interfaces as data exchange

If you are encapsulating the functions and program the data exchange between the blocks only via the interfaces, you will clearly have advantages.

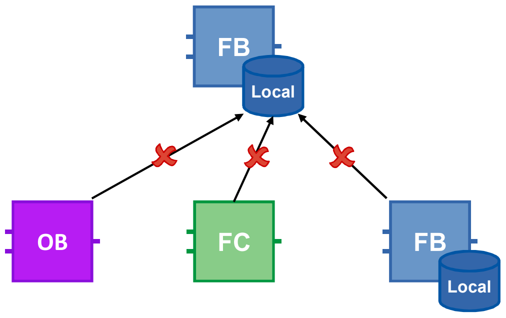
**Advantages**

- Program code is easier to read since there are no hidden cross accesses.
- Program can be made up modularly from ready blocks with partial tasks.
- Program is easy to expand and maintain.

**Recommendation**

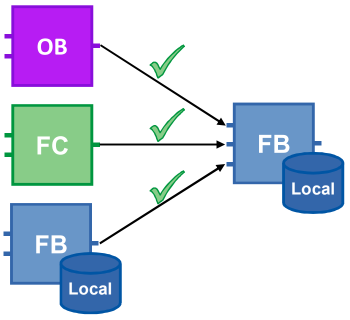
- If possible, only use the tags locally. This is how the blocks can be used universally and modularly.
- Use the data exchange via the block interfaces (In, Out, InOut), when tags are used in several blocks.
- Only use the instance data blocks as local memory for the respective function block. Other blocks must not be written into instance data blocks.

Figure 3-17: Avoiding accesses to instance data blocks



If only the block interfaces are used for the data exchange it can be ensured that all blocks can be used independent from each other.

Figure 3-18: Block interfaces for data exchange



## 3.4.2 Global memory

Memories are called global when they can be accessed from any location of the user program. There are hardware-dependent memories (e.g. bit memory, times, counters, etc.) and global DBs. For hardware-dependent memory areas there is the danger that the program may not be portable to any controller because the areas there may already be used. This is why you should use global DBs instead of hardware-dependent memory areas.

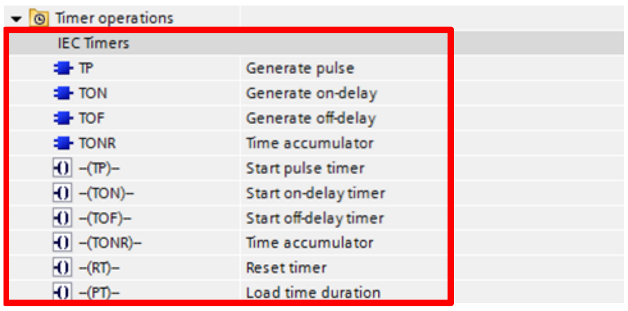
### Advantages

- User programs can be used universally and independent from the hardware.
- The user program can be structured modularly without having to define bit memory address areas for different users.
- Optimized global DBs are clearly more powerful than the bit memory address area that is not optimized for reasons of compatibility.

### Recommendation

- Do not use any bit memory and use global DBs instead.
- Avoid hardware-dependent memory, such as, for example, clock memory or counter. Use the IEC counter and timer in connection with multi-instances instead (see chapter 3.2.5 Multi-instances). The IEC timers can be found under “Instructions – Basic Instructions – Timer operations”.

Figure 3-19: IEC Timers



| Timer operations | |
|---|---|
| IEC Timers | |
| TP | Generate pulse |
| TON | Generate on-delay |
| TOF | Generate off-delay |
| TONR | Time accumulator |
| –(TP)– | Start pulse timer |
| –(TON)– | Start on-delay timer |
| –(TOF)– | Start off-delay timer |
| –(TONR)– | Time accumulator |
| –(RT)– | Reset timer |
| –(PT)– | Load time duration |

## 3.4.3 Local memory

- Static tags
- Temporary tags

### Recommendation

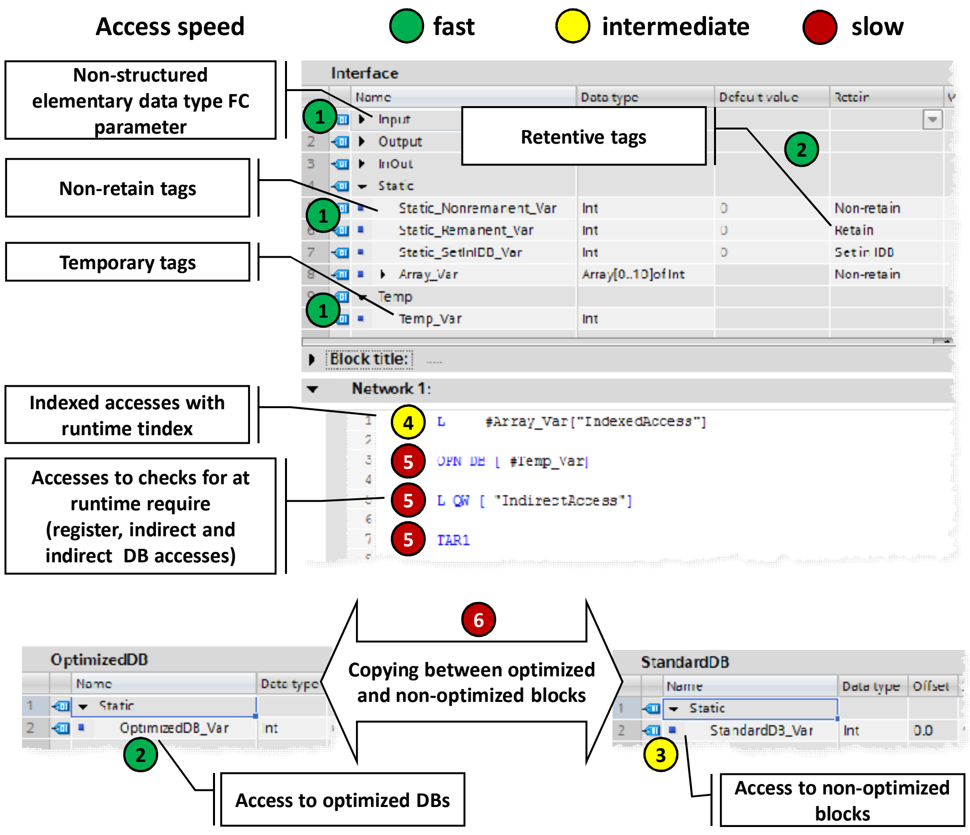
- Use the static tags if the tag values are required for the next cycle.
- Use the temporary tags if tags are only required for the current cycle as intermediate memory. The access time for temporary tags is shorter than for static ones.

**Note** Optimized blocks: Temporary tags are initialized in any block call with the value “0” (S7-1500 und S7-1200 Firmware V4).
Non-optimized blocks: Temporary tags are undefined for each call of the block.

## 3.4.4 Access speed of memory areas

STEP 7 offers different options of memory accesses. For system-related reasons there are faster and slower accesses to different memory areas.

Figure 3-20: Different memory accesses


Access speed
fast
intermediate
slow
Non-structured elementary data type FC parameter
Non-retain tags
Temporary tags
Retentive tags
Indexed accesses with runtime tindex
Accesses to checks for at runtime require (register, indirect and indirect DB accesses)
Copying between optimized and non-optimized blocks
Access to optimized DBs
Access to non-optimized blocks
Interface
Block title:
Network 1:
OptimizedDB
StandardDB
1
2
4
5
6
3

| Name | Data type | Default value | Retain |
|---|---|---|---|
| Input | | | |
| Output | | | |
| InOut | | | |
| Static | | | |
| Static_Nonremanent_Var | Int | 0 | Non-retain |
| Static_Remanent_Var | Int | 0 | Retain |
| Static_SetInIDB_Var | Int | 0 | Set in IDB |
| Array_Var | Array[0..10] of Int | | Non-retain |
| Temp | | | |
| Temp_Var | Int | | |

```
L      #Array_Var["IndexedAccess"]

OPN DB [ #Temp_Var]

L QW [ "IndirectAccess"]

TAR1
```

| Name | Data type |
|---|---|
| Static | |
| OptimizedDB_Var | Int |

| Name | Data type | Offset |
|---|---|---|
| Static | | |
| StandardDB_Var | Int | 0.0 |

**Fastest accesses in the S7-1200/1500 in descending order**

1. Optimized blocks: Temporary tags, parameters of an FC and FB, non-retentive static tags
2. Optimized blocks whose accesses for compiling are known:
   - Retentive FB tags
   - Optimized global DBs
3. Access to non-optimized blocks
4. Indexed accesses with index that was calculated at runtime (e.g. `Motor [i]`)
5. Accesses that require checks at runtime
   - Accesses to DBs that are created at runtime or which were opened indirectly (e.g. OPN DB[i])
   - Register access or indirect memory access
6. Copying of structures between optimized and non-optimized blocks (apart from Array of Bytes)

## 3.5 Retentivity

In the event of a failure of the power supply, the controller copies the retentive data with its buffer energy from the controller’s work memory to a non-volatile memory. After restarting the controller, the program processing is resumed with the retentive data. Depending on the controller, the data volume for retentivity has different sizes.

Table 3-4: Retentive memory for S7-1200/1500

| Controller | Usable retentive memory for bit memory, times, counters, DBs and technology objects |
|---|---|
| CPU 1215C | 10 Kbytes |
| CPU 1215C | 10 Kbytes |
| CPU 1214C | 10 Kbytes |
| CPU 1215C | 10 Kbytes |
| CPU 1511-1 PN | 88 Kbytes |
| CPU 1513-1 PN | 88 Kbytes |
| CPU 1516-3 PN/DP | 472 Kbytes |

Table 3-5: Differences of S7-1200 and S7-1500

| S7-1200 | S7-1500 |
|---|---|
| Retentivity can **only** be set for bit memory | Retentivity can be set for bit memory, times and counters |

### Advantages

- Retentive data maintain their value when the controller goes to STOP and back to RUN or in the event of a power failure and a restart of the controller.

### Properties

For elementary tags of an optimized DB the retentivity can be set separately. Non-optimized data blocks can only be defined completely retentive or non-retentive.

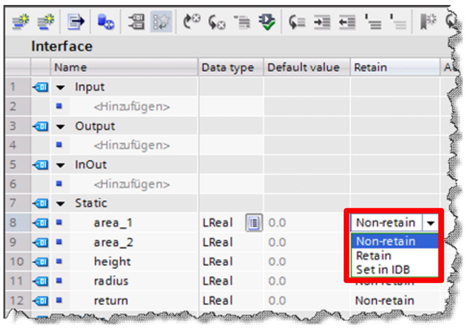
The retentive data can be deleted with the actions "memory reset" or "Reset to factory settings" via:

- Operating switch on the controller (MRES)
- Display of the controller
- Online via STEP 7 (TIA Portal)

### Recommendation

- Avoid the setting “Set in IDB”. Always set the retentive data in the function block and not in the instance data block.
  The “Set in IDB” setting increases the processing time of the program sequence. Always either select “Non-retain” or “Retain” for the interfaces in the FB.

Figure 3-21: Program editor (Functions block interfaces)



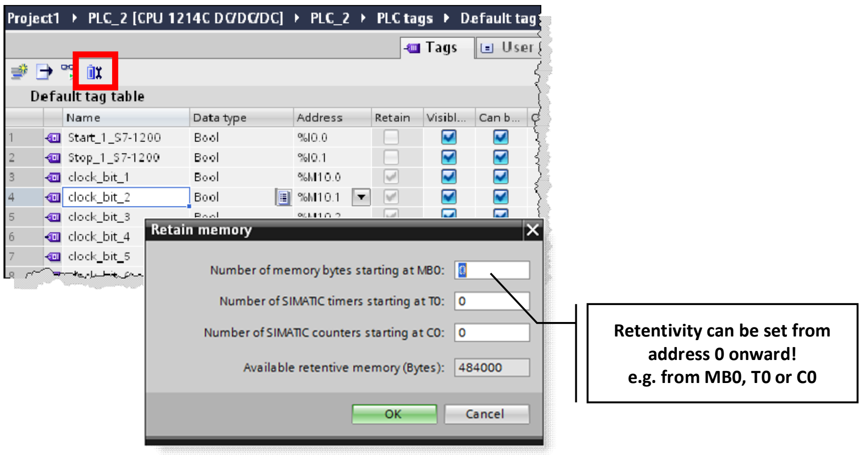
Interface

| | Name | Data type | Default value | Retain |
|---|---|---|---|---|
| 1 | Input | | | |
| 2 | <Hinzufügen> | | | |
| 3 | Output | | | |
| 4 | <Hinzufügen> | | | |
| 5 | InOut | | | |
| 6 | <Hinzufügen> | | | |
| 7 | Static | | | |
| 8 | area_1 | LReal | 0.0 | Non-retain |
| 9 | area_2 | LReal | 0.0 | |
| 10 | height | LReal | 0.0 | |
| 11 | radius | LReal | 0.0 | |
| 12 | return | LReal | 0.0 | Non-retain |

## Example

The setting of the retentive data is performed in the tables of the PLC data types, function blocks and data blocks.

Figure 3-22: Setting of the retentive tags in the table of PLC data types



Project1 ▸ PLC_2 [CPU 1214C DC/DC/DC] ▸ PLC_2 ▸ PLC tags ▸ Default tag
Tags
User

Default tag table

| | Name | Data type | Address | Retain | Visibl... | Can b... |
|---|---|---|---|---|---|---|
| 1 | Start_1_S7-1200 | Bool | %I0.0 | ☐ | ✓ | ✓ |
| 2 | Stop_1_S7-1200 | Bool | %I0.1 | ☐ | ✓ | ✓ |
| 3 | clock_bit_1 | Bool | %M10.0 | ✓ | ✓ | ✓ |
| 4 | clock_bit_2 | Bool | %M10.1 | ✓ | ✓ | ✓ |
| 5 | clock_bit_3 | | | | | |
| 6 | clock_bit_4 | | | | | |
| 7 | clock_bit_5 | | | | | |

Retain memory

Number of memory bytes starting at MB0: 0
Number of SIMATIC timers starting at T0: 0
Number of SIMATIC counters starting at C0: 0
Available retentive memory (Bytes): 484000

Retentivity can be set from address 0 onward!
e.g. from MB0, T0 or C0

## 3.6 Symbolic addressing

### 3.6.1 Symbolic instead of absolute addressing

The TIA Portal is optimized for symbolic programming. This results in many advantages. Due to the symbolic addressing you can program without having to pay attention to the internal data storage. The controller handles where the best possible storage is for the data. You can therefore completely concentrate on the solution for your application task.

### Advantages

- Easier to read programs through symbolic tag names
- Automatic update of tag names at all usage locations in the user program
- Memory storage of the program data does not have to be manually managed (absolute addressing)
- Powerful data access

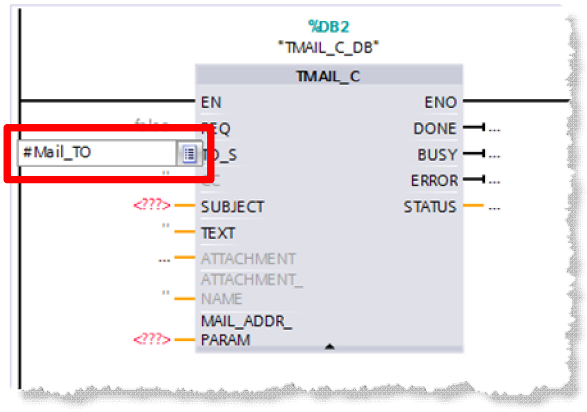
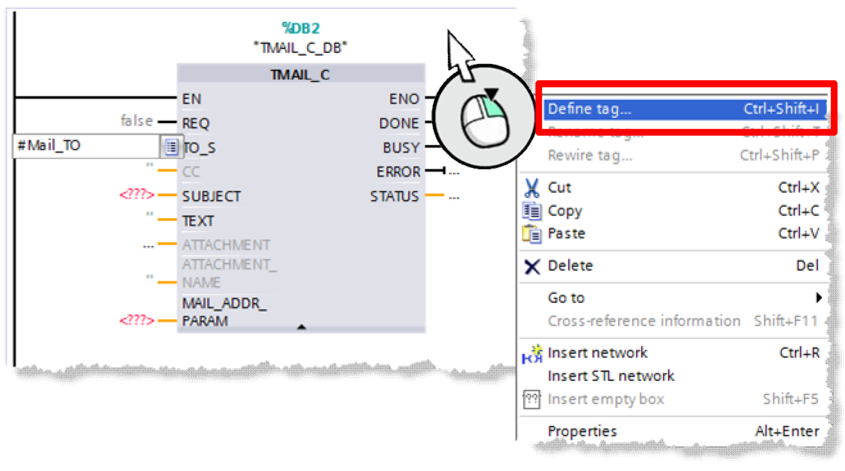
- No manual optimization for performance or program size reasons required
- IntelliSense for fast symbol input
- Fewer program errors due to type safety (validity of data types is checked for all accesses)

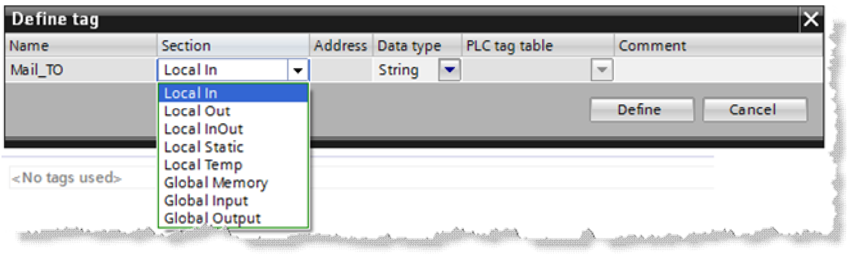
### Recommendation

- “Think” symbolically. Enter the “descriptive” name for each function, tag or data, such as, for example, Pump_boiler_1, heater_room_4, etc. This is how a generated program can easily be read without requiring many comments.
- Give all the tags used a direct symbolic name and define it afterwards with a right-click.

### Example

Table 3-6: Example for creating symbolic tags

| Step | Instruction |
|---|---|
| 1. | Open the program editor and open any block. |
| 2. | Enter a symbolic name directly at the input of an instruction.   %DB2 "TMAIL_C_DB" TMAIL_C EN ENO false REQ DONE #Mail_TO TO_S BUSY ERROR <???> SUBJECT STATUS TEXT ATTACHMENT ATTACHMENT_NAME MAIL_ADDR_PARAM |
| 3. | Right-click next to the block and select “Define tag…” in the context menu.   %DB2 "TMAIL_C_DB" TMAIL_C EN ENO false REQ DONE #Mail_TO TO_S BUSY CC ERROR <???> SUBJECT STATUS TEXT ATTACHMENT ATTACHMENT_NAME MAIL_ADDR_PARAM Define tag... Ctrl+Shift+I Rewire tag... Ctrl+Shift+P Cut Ctrl+X Copy Ctrl+C Paste Ctrl+V Delete Del Go to Cross-reference information Shift+F11 Insert network Ctrl+R Insert STL network Insert empty box Shift+F5 Properties Alt+Enter |

| Step | Instruction |
|---|---|
| 4. | Define the tag.  Define tag<br>Name: Mail_TO<br>Section: Local In<br>Address<br>Data type: String<br>PLC tag table<br>Comment<br>Local In, Local Out, Local InOut, Local Static, Local Temp, Global Memory, Global Input, Global Output<br>Define<br>Cancel<br><No tags used> |

There is an elegant method to save time, if you want to define several tags in a network. Assign all tag names first of all. Then define all tags at the same time with the dialog of step 4.

**Note**

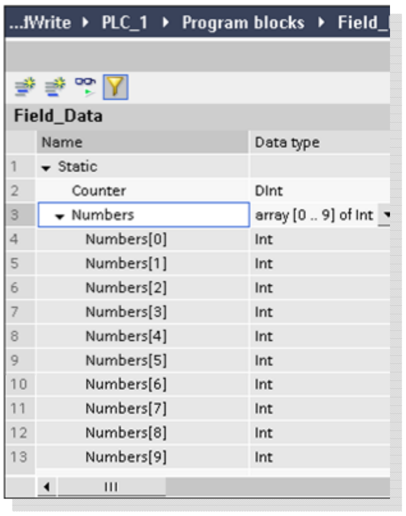
You will find further information in the following entry:

Why is universal definition and utilization of symbols in STEP 7 (TIA Portal) V12 obligatory for the S7-1500?
http://support.automation.siemens.com/WW/view/en/67598995

## 3.6.2 ARRAY data type and indirect field accesses

The ARRAY data type is suitable, for example, for the storage of recipes, material tracking in a queue, cyclic process acquisition, protocols, etc.

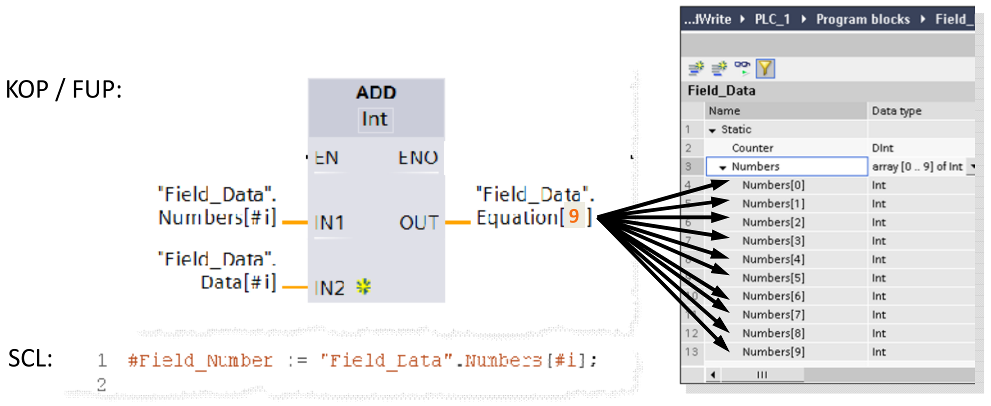
Figure 3-23: ARRAY with 9 elements of the Integer (INT) data type



|  | Name | Data type |
|---|---|---|
| 1 | Static | |
| 2 | Counter | DInt |
| 3 | Numbers | array [0 .. 9] of Int |
| 4 | Numbers[0] | Int |
| 5 | Numbers[1] | Int |
| 6 | Numbers[2] | Int |
| 7 | Numbers[3] | Int |
| 8 | Numbers[4] | Int |
| 9 | Numbers[5] | Int |
| 10 | Numbers[6] | Int |
| 11 | Numbers[7] | Int |
| 12 | Numbers[8] | Int |
| 13 | Numbers[9] | Int |

You can indirectly access individual elements in the ARRAY with a runtime tag (`array ["index"]`).

Figure 3-24: Indirect field access



```
#Field_Number := "Field_Data".Numbers[#i];
```

| | Name | Data type |
|---|---|---|
| 1 | Static | |
| 2 | Counter | DInt |
| 3 | Numbers | array [0 .. 9] of Int |
| 4 | Numbers[0] | Int |
| 5 | Numbers[1] | Int |
| 6 | Numbers[2] | Int |
| 7 | Numbers[3] | Int |
| 8 | Numbers[4] | Int |
| 9 | Numbers[5] | Int |
| 10 | Numbers[6] | Int |
| 11 | Numbers[7] | Int |
| 12 | Numbers[8] | Int |
| 13 | Numbers[9] | Int |

**Advantages**

- Simple access since the data type of the ARRAY elements is irrelevant for the access.
- No complicated pointer creation required
- Fast creation and expansion possible
- Useable in all programming languages

**Properties**

- Structured data type
- Data structure made of fixed number of elements of the same data type
- ARRAYs can be created nested or also multi-dimensional
- Possible indirect access with runtime tag with dynamic index calculation at runtime

**Recommendation**

- Use ARRAY for indexed accesses instead of pointer (e.g. ANY pointer). This makes it easier to read the program since an ARRAY is more meaningful with a symbolic name than a pointer in a memory area.
- As run tag use the INT data type as temporary tag for highest performance.
- Use the “MOVE_BLK” instruction to copy parts of an ARRAY into another one.
- Use the “GET_ERR_ID” instruction to catch access errors within the Array.

**Note**

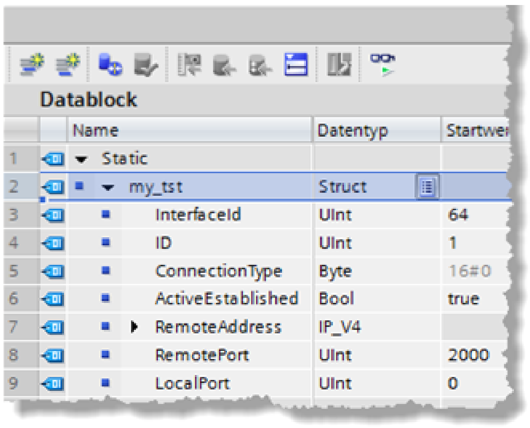
You will find further information in the following entry:

How do you implement an array access with an S7-1500 with variable index?
http://support.automation.siemens.com/WW/view/en/67598676

### 3.6.3 STRUCT data type and PLC data types

The STRUCT data type represents a data structure which is made up of elements of different data types. The declaration of a structure is performed in the respective block.

Figure 3-25: Structure with elements with different data types



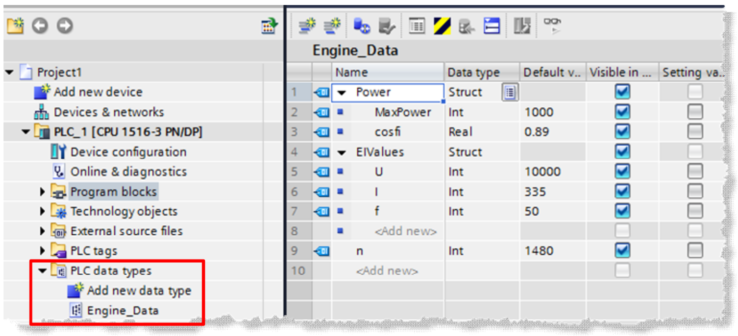
Datablock

| | Name | Datentyp | Startwer |
|---|---|---|---|
| 1 | ▼ Static | | |
| 2 | ▼ my_tst | Struct | |
| 3 | InterfaceId | UInt | 64 |
| 4 | ID | UInt | 1 |
| 5 | ConnectionType | Byte | 16#0 |
| 6 | ActiveEstablished | Bool | true |
| 7 | ▶ RemoteAddress | IP_V4 | |
| 8 | RemotePort | UInt | 2000 |
| 9 | LocalPort | UInt | 0 |

In comparison to structures, PLC data types are defined across the controller in the TIA Portal and can be centrally changed. All usage locations are automatically updated.

PLC data types are declared in the “PLC data types” folder in the project navigation before being used.

Figure 3-26: PLC data types



- Project1
  - Add new device
  - Devices & networks
  - PLC_1 [CPU 1516-3 PN/DP]
    - Device configuration
    - Online & diagnostics
    - Program blocks
    - Technology objects
    - External source files
    - PLC tags
    - PLC data types
      - Add new data type
      - Engine_Data

Engine_Data

| | Name | Data type | Default v.. | Visible in ... | Setting va.. |
|---|---|---|---|---|---|
| 1 | ▼ Power | Struct | | ✓ | |
| 2 | MaxPower | Int | 1000 | ✓ | ☐ |
| 3 | cosfi | Real | 0.89 | ✓ | ☐ |
| 4 | ▼ ElValues | Struct | | ✓ | |
| 5 | U | Int | 10000 | ✓ | ☐ |
| 6 | I | Int | 335 | ✓ | ☐ |
| 7 | f | Int | 50 | ✓ | ☐ |
| 8 | <Add new> | | | | |
| 9 | n | Int | 1480 | ✓ | ☐ |
| 10 | <Add new> | | | | |

## Advantages

- A change in a PLC data type is automatically updated in all usage locations in the user program.

## Recommendation

- Use the PLC data types to summarize several associated data, such as, e.g. frames or motor data (setpoint, speed, rotational direction, temperature, etc.)
- Always use PLC data types instead of structures for the multiple uses in the user program.
- Use the PLC data types for structuring into data blocks.

**Note**

You will find further information in the following entries:

How do you initialize structures into optimized memory areas for the S7-1500 STEP 7 (TIA Portal)?
http://support.automation.siemens.com/WW/view/en/78678761

How do you create a PLC data type for an S7-1500 controller?
http://support.automation.siemens.com/WW/view/en/67599090

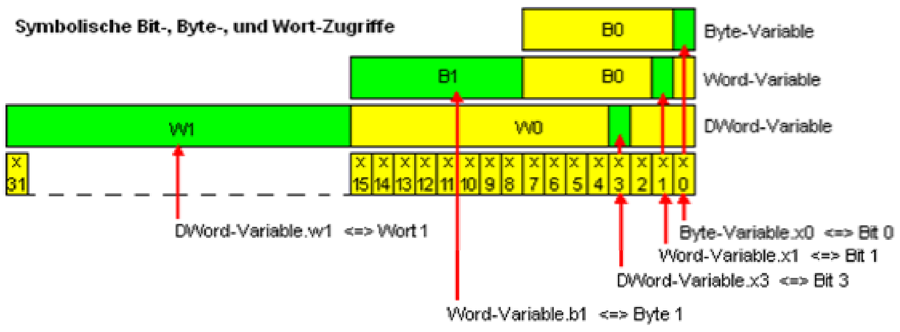
In STEP 7 (TIA Portal) V12, how do you apply your own data types (UDT)?
http://support.automation.siemens.com/WW/view/en/67582844

Why should whole structures instead of many single components be transferred for the S7-1500 when a block is called?
http://support.automation.siemens.com/WW/view/en/67585079

## 3.6.4 Slice access

For S7-1200/1500 controllers, you can access the memory area of tags of the Byte, Word, DWord or LWord data type. The division of a memory area (e.g. byte or word) into a smaller memory area (e.g. Bool) is also called slice. In the figure below displays the symbolic bit, byte and word accesses to the operands.

Figure 3-27: Slice access


Symbolische Bit-, Byte-, und Wort-Zugriffe
B0
Byte-Variable
B1
B0
Word-Variable
W1
W0
DWord-Variable
x 31
x 15 x 14 x 13 x 12 x 11 x 10 x 9 x 8 x 7 x 6 x 5 x 4 x 3 x 2 x 1 x 0
DWord-Variable.w1 <=> Wort 1
Byte-Variable.x0 <=> Bit 0
Word-Variable.x1 <=> Bit 1
DWord-Variable.x3 <=> Bit 3
Word-Variable.b1 <=> Byte 1

### Advantages

- High programming efficiency
- No additional definition in the tag declaration required
- Simple access (e.g. control bits)

### Recommendation

- Use the slice access rather than AT construct via accessing certain data areas in operands.

**Note**

You will find further information in the following entry:

How in STEP 7 (TIA Portal) can you access the unstructured data types bit-by-bit, byte-by-byte or word-by-word and symbolically?
http://support.automation.siemens.com/WW/view/en/57374718

## 3.7 Libraries

With the TIA Portal you can create independent libraries from different project elements that can be easily reused.

**Advantages**

- Simple storage for the data configured in the TIA Portal:
  - Complete devices (controller, HMI, drive, etc.)
  - Controller programs, blocks, tags, monitoring tables
  - HMI image, HMI tags, scripts, etc.
- Cross-project exchange via libraries
- Central update function of library elements
- Versioning library elements
- Fewer error sources when using control blocks through system-supported consideration of dependencies

**Recommendations**

- Create the master copies for easy reusability of blocks, hardware configurations, HMI images, etc.
- Create the types for the system-supported reusability of library elements:
  - Versioning of blocks
  - Central update function of all usage locations
- Use the global library for the exchange with other users or as central storage for the simultaneous use of several users.

### 3.7.1 Types of libraries and library elements

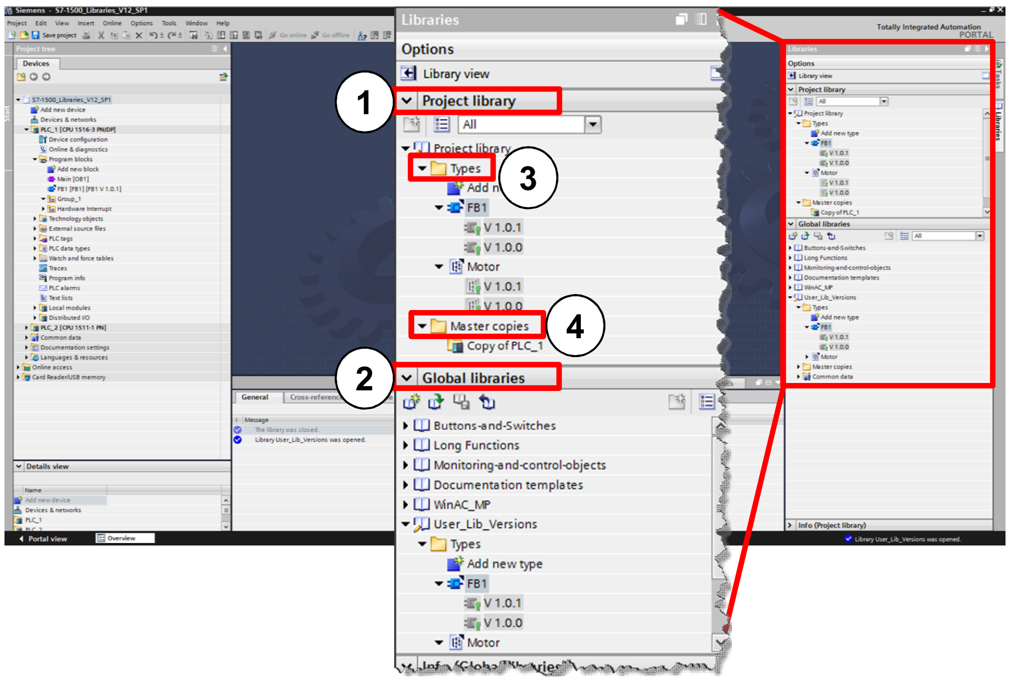
Generally there are two different types of libraries:

- "Project library"
- "Global library".

The content consists of two storage types each:

- "Types"
- "Master Copies"

Figure 3-28: Libraries in the TIA Portal



- (1) "Project library"
  - Integrated in the project and managed with the project
  - Allows the reusability within the project
- (2) "Global library"
  - Independent library
  - Use within several projects possible

A library includes two different types of storage of library elements:

- (3) "Master copies"
  - Copy of configuration elements in the library (e.g. blocks, hardware, PLC tag tables, etc.)
  - Copies are not connected with the elements in the project.
  - Master copies can also be made up several configuration elements.
- (4) "Types"
  - Types are connected with your usage locations in the project. When types are changed, the usage locations in the project can be updated automatically.
  - Supported types are controller blocks (FCs, FBs), PLC data types, HMI images, HMI faceplates, HMI UDT, scripts).
  - Subordinate elements are automatically typified.
  - Types are versioned: Changes can be made by creating a newer version.
  - There can only be one version of a used type within a controller.

### 3.7.2 Type concept

The type concept allows the creation of standardized automation functions that you can use in several plants or machines. The type concept supports you with versioning and updating functions.

You can use types from the library in the user program. This offers the following advantages:

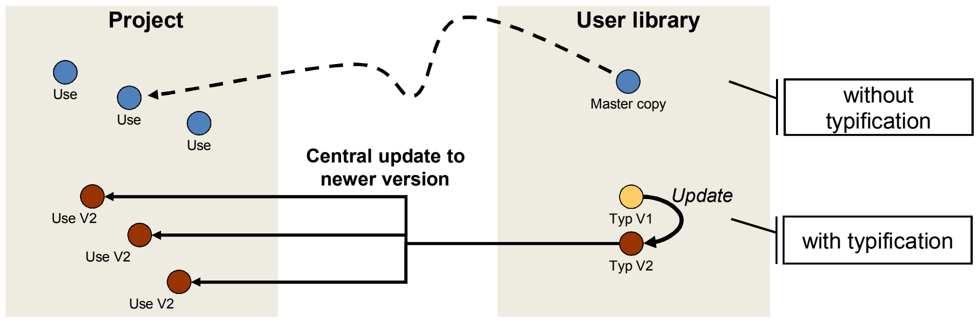
#### Advantages

- Central update of all usage locations in the project
- Unwanted modifications of usage locations of types are not possible.
- The system guarantees that types always remain consistent by hindering unwanted delete operations.
- If a type is deleted, all usage locations in the user program are deleted.

#### Properties

By using types you can make the changes centrally and update them in the complete project.

Figure 3-29: Typifying with user libraries



- Types are always marked in the project for better identification

### 3.7.3 Differences for typifiable objects for controller and HMI

There are system-related differences between the typifiable objects for controllers and HMI:

Table 3-7: Differences of types for controller and HMI

| Controller | HMI |
|---|---|
| Subordinate control elements are typified. | Subordinate HMI elements are **not** typified. |
| Subordinate control elements are instanced. | Subordinate HMI elements are **not** instanced. |
| Control elements are edited in a **test environment**. | HMI images and HMI scripts are edited in a test environment. Faceplates and HMI - UDTs are directly edited in the library **without test environment**. |

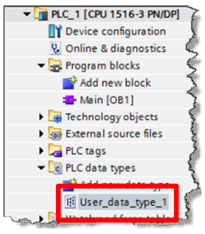
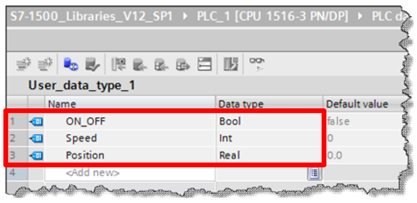
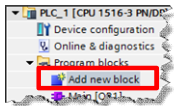
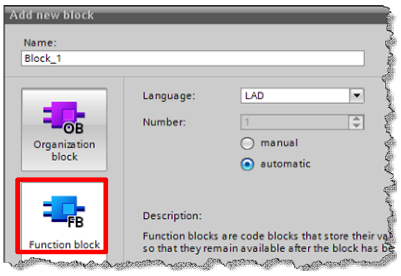
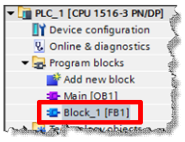
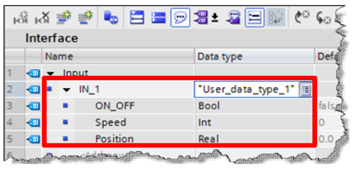
Further information on the handling of libraries can be found in the following example.

## 3.7.4 Versioning of a block

### Example

The following example shows you how the basic functions of the libraries are used with types.

Table 3-8: Creating a type

| Step | Instruction |
|---|---|
| 1. | Create a new PLC data type with “Add new data type” and create some tags. Later on this is the subordinate type.<br> PLC_1 [CPU 1516-3 PN/DP]<br>Device configuration<br>Online & diagnostics<br>Program blocks<br>Add new block<br>Main [OB1]<br>Technology objects<br>External source files<br>PLC tags<br>PLC data types<br>User_data_type_1<br>S7-1500_Libraries_V12_SP1 ▸ PLC_1 [CPU 1516-3 PN/DP] ▸ PLC d<br>User_data_type_1<br>Name / Data type / Default value<br>1 ON_OFF Bool false<br>2 Speed Int 0<br>3 Position Real 0.0<br>4 &lt;Add new&gt; |
| 2. | Create a new function block with “Add new Block”. This is the higher-level type.<br> PLC_1 [CPU 1516-3 PN/DP<br>Device configuration<br>Online & diagnostics<br>Program blocks<br>Add new block<br>Main [OB1]<br>Add new block<br>Name: Block_1<br>Language: LAD<br>Number: 1<br>manual [✓ automatic]<br>Description:<br>Function blocks are code blocks that store their va<br>so that they remain available after the block has be<br> |
| 3. | Define an input tag of the data type you have created. The PLC data type is therefore subordinate to the function block.<br> PLC_1 [CPU 1516-3 PN/DP]<br>Device configuration<br>Online & diagnostics<br>Program blocks<br>Add new block<br>Main [OB1]<br>Block_1 [FB1]<br>Interface<br>Name / Data type / Defa<br>1 Input<br>2 IN_1 "User_data_type_1"<br>3 ON_OFF Bool fals<br>4 Speed Int 0<br>5 Position Real 0.0 |

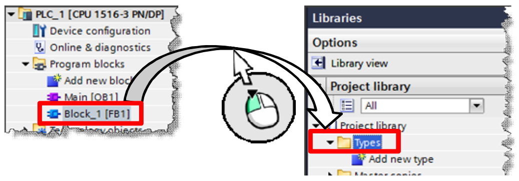
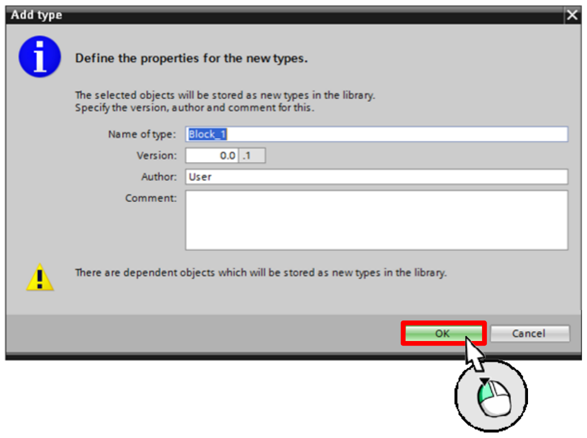
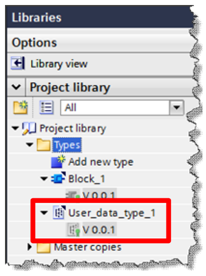
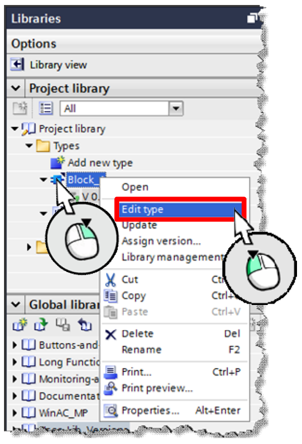
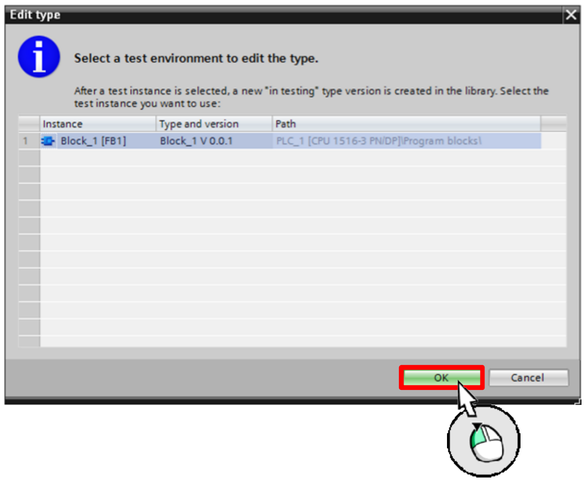
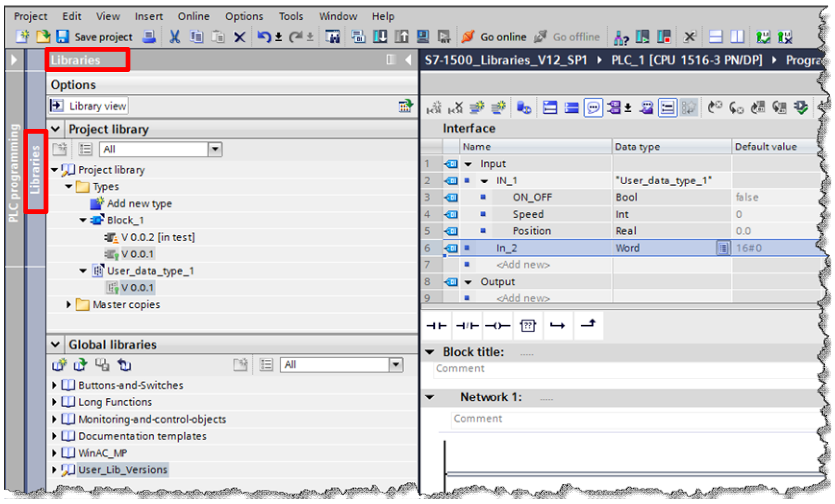
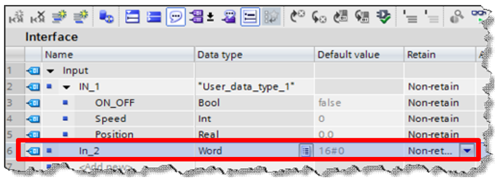
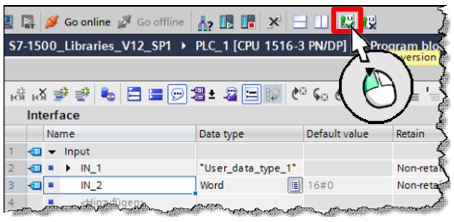
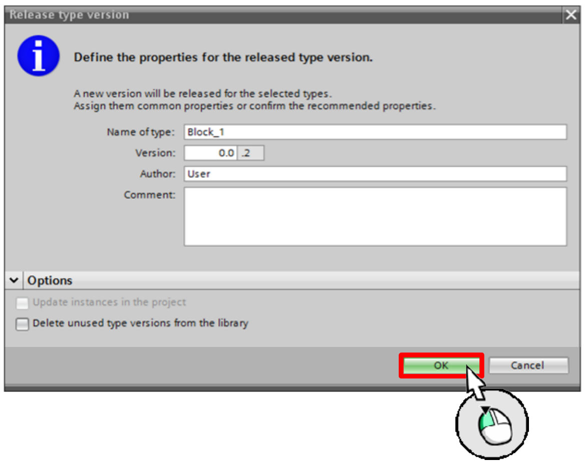
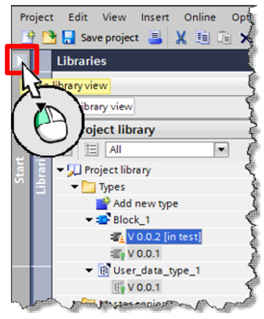
| Step | Instruction |
|---|---|
| 4. | Drag the function block via drag & drop into the “Types” folder in the project library.<br><br> PLC_1 [CPU 1516-3 PN/DP]<br>Device configuration<br>Online & diagnostics<br>Program blocks<br>Add new block<br>Main [OB1]<br>Block_1 [FB1]<br>Libraries<br>Options<br>Library view<br>Project library<br>All<br>Project library<br>Types<br>Add new type |
| 5. | Optionally assign: Type name, version, author and comment and confirm the dialog with “OK”.<br><br>Add type<br>**Define the properties for the new types.**<br>The selected objects will be stored as new types in the library.<br>Specify the version, author and comment for this.<br>Name of type: Block_1<br>Version: 0.0 .1<br>Author: User<br>Comment:<br>There are dependent objects which will be stored as new types in the library.<br>OK Cancel |
| 6. - | The subordinate PLC data type is automatically also stored in the library.<br><br> Libraries<br>Options<br>Library view<br>Project library<br>All<br>Project library<br>Types<br>Add new type<br>Block_1<br>V 0.0.1<br>User_data_type_1<br>V 0.0.1<br>Master copies |

Table 3-9: Changing a type

| Step | Instruction |
|---|---|
| 1. | Right-click the block in the “Project library” and select “Edit type”<br><br> Libraries<br>Options<br>Library view<br>Project library<br>All<br>Project library<br>Types<br>Add new type<br>Block_<br>V 0.<br>Global librar<br>Buttons-and<br>Long Functic<br>Monitoring-a<br>Documentat<br>WinAC_MP<br>Open<br>Edit type<br>Update<br>Assign version...<br>Library management<br>Cut<br>Copy Ctrl+<br>Paste Ctrl+V<br>Delete Del<br>Rename F2<br>Print... Ctrl+P<br>Print preview...<br>Properties... Alt+Enter |
| 2. | Select which controller is to be used as test environment and confirm the dialog with “OK”.<br><br>Edit type<br>**Select a test environment to edit the type.**<br>After a test instance is selected, a new "in testing" type version is created in the library. Select the test instance you want to use:<br>Instance \| Type and version \| Path<br>1 Block_1 [FB1] \| Block_1 V 0.0.1 \| PLC_1 [CPU 1516-3 PN/DP]\Program blocks\ <br> <br><br>If several controllers in the project use the selected block, a controller has to be selected as test environment. |

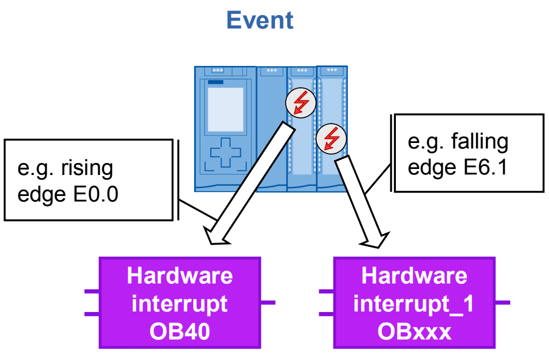
| Step | Instruction |
|---|---|
| 3. | The library view opens. A new version of the block has been created and is now marked with “in test”.   Project Edit View Insert Online Options Tools Window Help<br>Save project<br>Go online<br>Go offline<br>Libraries<br>Options<br>Library view<br>Project library<br>All<br>Project library<br>Types<br>Add new type<br>Block_1<br>V 0.0.2 [in test]<br>V 0.0.1<br>User_data_type_1<br>V 0.0.1<br>Master copies<br>Global libraries<br>Buttons-and-Switches<br>Long Functions<br>Monitoring-and-control-objects<br>Documentation templates<br>WinAC_MP<br>User_Lib_Versions<br>PLC programming<br>S7-1500_Libraries_V12_SP1 ▸ PLC_1 [CPU 1516-3 PN/DP] ▸ Progra<br>Interface<br>Block title:<br>Comment<br>Network 1:<br>Comment <br>1 Input<br>2 IN_1 "User_data_type_1"<br>3 ON_OFF Bool false<br>4 Speed Int 0<br>5 Position Real 0.0<br>6 In_2 Word 16#0<br>7 &lt;Add new&gt;<br>8 Output<br>9 &lt;Add new&gt; |
| 4. | Add another input tag.  <br>Name / Data type / Default value / Retain<br>1 Input<br>2 IN_1 "User_data_type_1" Non-retain<br>3 ON_OFF Bool false Non-retain<br>4 Speed Int 0 Non-retain<br>5 Position Real 0.0 Non-retain<br>6 In_2 Word 16#0 Non-ret...<br><br>In this place you have the option to test the change on the block by loading the project onto a controller. When you have finished testing the block, continue with the following steps. |
| 5. | Click the “Release version” button.   Go online<br>Go offline<br>S7-1500_Libraries_V12_SP1 ▸ PLC_1 [CPU 1516-3 PN/DP]<br>Program blo<br>version<br>Interface <br>Name / Data type / Default value / Retain<br>1 Input<br>2 IN_1 "User_data_type_1" Non-reta<br>3 IN_2 Word 16#0 Non-reta<br>4 &lt;Hinzufügen&gt; |

| Step | Instruction |
|---|---|
| 6. | A dialog box opens. Here you can store a version-related comment. Confirm the dialog with “OK”.<br><br>Release type version<br>Define the properties for the released type version.<br>A new version will be released for the selected types.<br>Assign them common properties or confirm the recommended properties.<br>Name of type: Block_1<br>Version: 0.0 .2<br>Author: User<br>Comment:<br>Options<br>Update instances in the project<br>Delete unused type versions from the library<br> <br>If there are several usage locations of the block in different controllers of the project, you can update them all at the same time: “Update instances in the project”.<br>If older versions of the element are no longer required you can delete them by clicking “Delete unused type versions from library” |
| 7. | Close the library view with “Close library view”<br><br> Project Edit View Insert Online Opt<br>Save project<br>Libraries<br>library view<br>library view<br>roject library<br>All<br>Start<br>Librari<br>Project library<br>Types<br>Add new type<br>Block_1<br>V 0.0.2 [in test]<br>V 0.0.1<br>User_data_type_1<br>V 0.0.1 |

## 3.8 Process interrupts

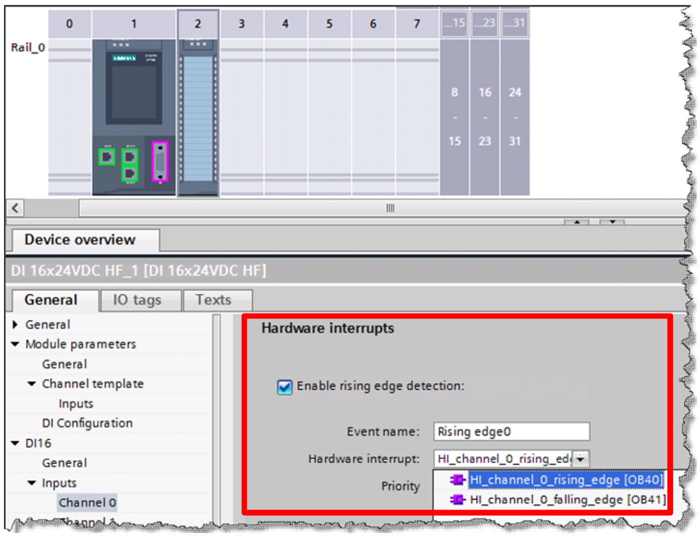
The processing of the user program can be influenced by events such as process interrupts. When you need a fast response of the controller to hardware events (e.g. a rising edge of a channel of a digital input module), configure a process interrupt. For each process interrupt a separate OB can be programmed. This OB is called by the operating system of the controller in the event of a process interrupt. The cycle of the controller is therefore interrupted and continued after processing the process interrupt.

Figure 3-30: Process interrupt is calling OB



In the following figure you can see the configuration of a “hardware interrupt” in the hardware configuration of a digital input module.

Figure 3-31: Configuring hardware interrupt



DI 16x24VDC HF_1 [DI 16x24VDC HF]

Hardware interrupts

[✓] Enable rising edge detection:

Event name: Rising edge0

Hardware interrupt: HI_channel_0_rising_ed

Priority

HI_channel_0_rising_edge [OB40]

HI_channel_0_falling_edge [OB41]

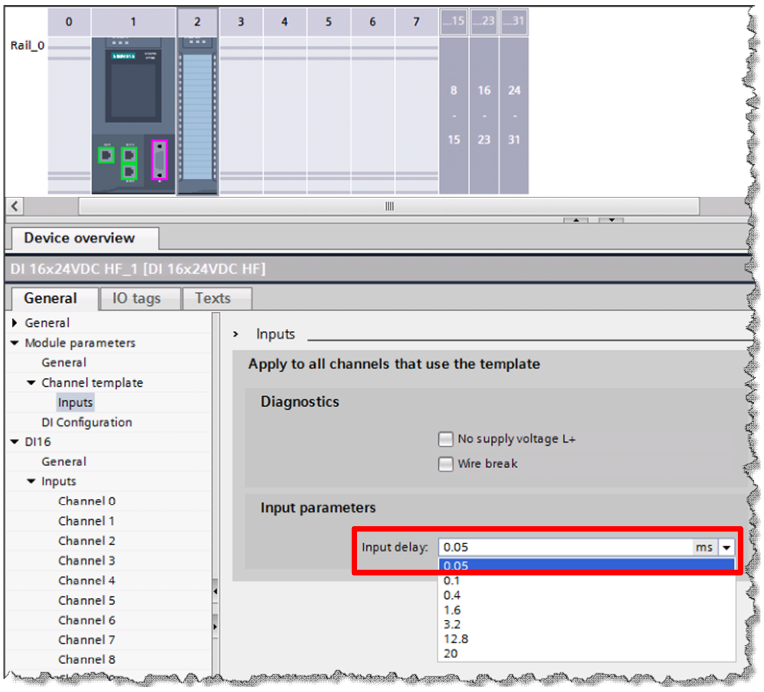
### Advantages

- Fast system response to events (rising, falling edge, etc.)
- Each event can start a separate OB.

### Recommendation

- Use the process interrupts in order to program fast responses to hardware events.
- If the system responses are not fast enough despite programming a process interrupt, you can still accelerate the responses. Set as small an “Input delay” as possible in the module. A response to an event can always only occur if the input delay has lapsed. The input delay is used for filtering the input signal in order to, for example, compensate faults such as contact bounce or chatter.

Figure 3-32: Setting input delay



Input delay: 0.05 ms

## 3.9 Other performance recommendations

Here you can find some general recommendations that enable faster program processing of the controller.

### Recommendation

Note the following recommendations for programming S7-1200/1500 controllers in order to achieve a high performance:

- LAD/FBD: Disable “generate ENO” for blocks. This avoids tests at runtime.
- STL: Do not use registers since address and data registers are only emulated for compatibility reasons by S7-1500.
- Disable the expanded test function after commissioning.

**Note**

You will find further information in the following entry:

How do you disable the ENO enable output of an instruction?
http://support.automation.siemens.com/WW/view/en/67797146

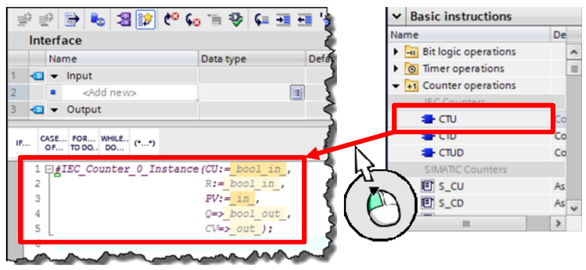
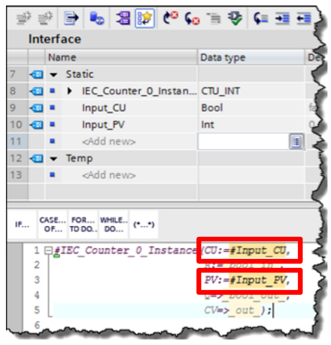
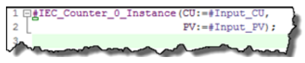
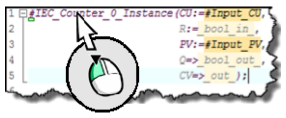
## 3.10 SCL programming language: Tips and tricks

### 3.10.1 Using call templates

Many instructions of the programming languages offer a call template with a list of existing formal parameters.

#### Example

Table 3-10: Easy expanding of the call template

| Step | Instruction |
|---|---|
| 1. | Drag an instruction from the library into the SCL program. The editor shows the complete call template.<br><br>Interface: Name / Data type / Default — 1 Input; 2 &lt;Add new&gt;; 3 Output<br>Basic instructions: Bit logic operations; Timer operations; Counter operations; IEC Counters: CTU, CTD, CTUD; SIMATIC Counters: S_CU, S_CD<br>`#IEC_Counter_0_Instance(CU:=_bool_in_,`<br>`R:=_bool_in_,`<br>`PV:=_in_,`<br>`Q=>_bool_out_,`<br>`CV=>_out_);` |
| 2. | Now fill in the required parameter and finish the entry with the “Return” button.<br><br>Interface: Name / Data type — 7 Static; 8 IEC_Counter_0_Instan… CTU_INT; 9 Input_CU Bool; 10 Input_PV Int; 11 &lt;Add new&gt;; 12 Temp; 13 &lt;Add new&gt;<br>`#IEC_Counter_0_Instance(CU:=#Input_CU,`<br>`R:=_bool_in_,`<br>`PV:=#Input_PV,`<br>`Q=>_bool_out_,`<br>`CV=>_out_);` |
| 3. | The editor automatically reduces the call template.<br><br>`#IEC_Counter_0_Instance(CU:=#Input_CU,`<br>`PV:=#Input_PV);` |
| 4. | If you want to edit the complete call later on again, proceed as follows.<br>Click into the call at any place and then click “**CTRL+SHIFT+SPACE**”. You are now in the Call Template mode. The editor expands the call again. You can navigate with the “TAB” button through the parameters.<br><br>`#IEC_Counter_0_Instance(CU:=#Input_CU,`<br>`R:=_bool_in_,`<br>`PV:=#Input_PV,`<br>`Q=>_bool_out_,`<br>`CV=>_out_);` |
| 5. | Note: In the “Call Template” mode the writing is in italics. |

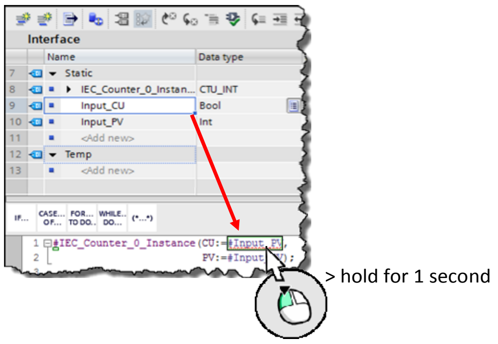
### 3.10.2 What instruction parameters are mandatory?

If you are expanding the call template, the color coding will show you straight away what formal parameters of an instruction are optional and which ones are not. Mandatory parameters are marked dark.

### 3.10.3 Drag & drop with entire tag names

In the SCL editor you can also use drag & drop functions. For tag names you are additionally supported. If you want to replace one tag for another, proceed as follows.

Table 3-11: Drag & drop with tags in SCL

| Step | Instruction |
|---|---|
| 1. | Drag the tag via drag & drop to the tag in the program that is to be replaced.<br>Hold the tag for more than 1 second before releasing it.<br> Interface<br>Name<br>Data type<br>7 Static<br>8 IEC_Counter_0_Instan... CTU_INT<br>9 Input_CU Bool<br>10 Input_PV Int<br>11 &lt;Add new&gt;<br>12 Temp<br>13 &lt;Add new&gt;<br>IF... CASE... OF... FOR... TO DO... WHILE... DO... (*...*)<br>> hold for 1 second<br>`#IEC_Counter_0_Instance(CU:=#Input_PV,`<br>`PV:=#Input_PV);`<br>The complete tag is replaced. |

### 3.10.4 Efficiently inserting CASE instruction

With the CASE instruction in SCL, it will be exactly jumped to the selected CASE block condition. After executing the CASE block the instruction is finished. This allows you, for example, to check frequently required value ranges more specifically and easily.

**Example**

```
CASE #myVar OF
        5:
                FC5(#myParam);
        10,12:
                FC10(#myParam);
        15:
                FC15(#myParam);
        0..20:
                FCGlobal(#myParam);
// FCGlobal is never called for the values 5, 10, 12 or 15!
        ELSE
END_CASE;
```

### 3.10.5 No manipulation of loop counters for FOR loop

FOR loops in SCL are pure counter loops, i.e. the number of iterations is fixed when the loop is entered. In a FOR loop, the loop counter cannot be changed.

With the EXIT instruction a loop can be interrupted at any time.

**Advantages**

- The compiler can optimize the program better, since it does not know the number of iterations.

**Example**

```
FOR #var := #lower TO #upper DO
  #var := #var + 1; // no effect, Compiler -> Warning
END_FOR;
```

### 3.10.6 FOR loop backwards

In SCL you can also increment the index of FOR loops backwards or in another step width. For this, use the optional “BY” key word in the loop head.

**Example**

```
FOR #var := #upper TO #lower BY -2 DO


END_FOR;
```

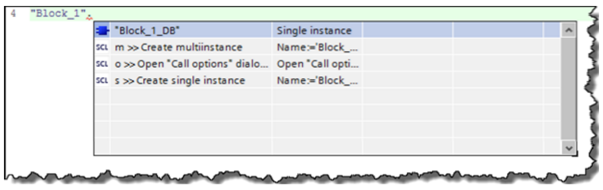
If you are defining “BY” as “-2”, as in the example, the counter is lowered by 2 in every iteration. If you omit “BY”, the default setting for “BY” is 1

### 3.10.7 Simple creating of instances for calls

If you prefer to work with the keyboard, there is a simple possibility to create instances for blocks in SCL.

**Example**

Table 3-12: Easy creation of instances

| Step | Instruction |
|---|---|
| 1. | Give the block name a: followed by a "." (dot). The automatic compilation now shows you the following.   4 "Block_1".<br>"Block_1_DB" Single instance<br>m >> Create multiinstance Name:='Block_...<br>o >> Open "Call options" dialo... Open "Call opti...<br>s >> Create single instance Name:='Block_... |
| 2. | On the top you can see the already existing instances. In addition, you can directly create a new single instance or multi-instance.<br>Use the shortcuts "s" or "m" to go directly to the respective entries in the automatic compilation window. |

### 3.10.8 Handling of time tags

You can calculate the time tags in SCL just as with normal numbers i.e. you do not need to look for additional functions, such as, e.g. T_COMBINE but you can use simple arithmetic. This approach is called “overload of operands”. The SCL compiler automatically uses the suitable functions. You can use a reasonable arithmetic for the time types and can therefore program more efficiently.

**Example**

```
Time_difference := Time stamp_1 – Time stamp_2;
```

The following table summarizes the overloaded operators and which operation is behind it:

Table 3-13: Overloaded operands for SCL

| Overloaded operand | Operation |
|---|---|
| ltime + time | T_ADD LTime |
| ltime + time | T_SUB LTime |
| ltime + lint | T_ADD LTime |
| ltime + lint | T_SUB LTime |
| time + time | T_ADD Time |
| time + time | T_SUB Time |
| time + dint | T_ADD Time |
| time + dint | T_SUB Time |
| ldt + ltime | T_ADD LDT / LTime |
| ldt + ltime | T_ADD LDT / LTime |
| ldt + time | T_ADD LDT / Time |
| ldt + time | T_SUB LDT / Time |
| dtl + ltime | T_ADD DTL / LTime |
| dtl + ltime | T_SUB DTL / LTime |
| dtl + time | T_ADD DTL / Time |
| dtl + time | T_SUB DTL / Time |
| ltod + ltime | T_ADD LTOD / LTime |
| ltod + ltime | T_SUB LTOD / LTime |
| ltod + lint | T_ADD LTOD / LTime |
| ltod + lint | T_SUB LTOD / LTime |
| ltod + time | T_ADD LTOD / Time |
| ltod + time | T_SUB LTOD / Time |
| tod + time | T_ADD TOD / Time |
| tod + time | T_SUB TOD / Time |
| tod + dint | T_ADD TOD / Time |
| tod + dint | T_SUB TOD / Time |
| dt + time | T_ADD DT / Time |
| dt + time | T_SUB DT / Time |
| ldt – ldt | T_DIFF LDT |
| dtl – dtl | T_DIFF DTL |
| dt – dt | T_DIFF DT |
| date – date | T_DIFF DATE |
| ltod – ltod | T_DIFF LTOD |
| date + ltod | T_COMBINE DATE / LTOD |
| date + tod | T_COMBINE DATE / TOD |

# 4 Hardware-Independent Programming

To make sure that a block can be used on all controllers without any further adjustments, it is important not use hardware-dependent functions and properties.

## 4.1 Data types of S7-300/400 and S7-1200/1500

Below is a list of all elementary data types and data groups.

**Recommendation**

- Only use the data types that are supported by the controllers on which the program is to run.

Table 4-1: Elementary data types correspond to standard EN 61131-3

| | Description | S7 - 300/400 | S7-1200 | S7-1500 |
|---|---|---|---|---|
| Bit data types | • BOOL<br>• BYTE<br>• WORD<br>• DWORD | ✓ | ✓ | ✓ |
| | • LWORD | ✗ | ✗ | ✓ |
| Character type | • CHAR (8 bit) | ✓ | ✓ | ✓ |
| Numerical data types | • INT (16 bit)<br>• DINT (32 bit)<br>• REAL (32 bit) | ✓ | ✓ | ✓ |
| | • SINT (8 bit)<br>• USINT (8 bit)<br>• UINT (16 bit)<br>• UDINT (32 bit)<br>• LREAL (64 bit) | ✗ | ✓ | ✓ |
| | • LINT (64 bit)<br>• ULINT (64 bit) | ✗ | ✗ | ✓ |
| Time types | • TIME<br>• DATE<br>• TIME_OF_DAY | ✓ | ✓ | ✓ |
| | • S5TIME | ✓ | ✗ | ✓ |
| | • LTIME<br>• L_TIME_OF_DAY | ✗ | ✗ | ✓ |

Table 4-2: Data groups that are made up of other data types

| | Description | S7 - 300/400 | S7-1200 | S7-1500 |
|---|---|---|---|---|
| Time types | • DT (DATE_AND_TIME) | ✓ | ✗ | ✓ |
| | • DTL | ✗ | ✓ | ✓ |

| | Description | S7 - 300/400 | S7-1200 | S7-1500 |
|---|---|---|---|---|
| | • LDT (L_DATE_AND_TIME) | × | × | ✓ |
| Character type | • STRING | ✓ | ✓ | ✓ |
| Field | • ARRAY | ✓ | ✓ | ✓[1)] |
| Structure | • STRUCT | ✓ | ✓ | ✓ |

[1)] For S7-1500 the ARRAY data type is limited to 64 bit instead of 16 bit

Table 4-3: Parameter types for formal parameters that are transferred between blocks

| | Description | S7 - 300/400 | S7-1200 | S7-1500 |
|---|---|---|---|---|
| Pointer | • POINTER<br>• ANY | ✓ | × | ✓[1)] |
| | • VARIANT | × | ✓ | ✓ |
| Blocks | • TIMER<br>• COUNTER | ✓ | ✓[2)] | ✓ |
| | • BLOCK_FB<br>• BLOCK_FC | ✓ | × | ✓ |
| | • BLOCK_DB<br>• BLOCK_SDB | ✓ | × | × |
| | • VOID | ✓ | ✓ | ✓ |
| PLC data types | • PLC Data Type | ✓ | ✓ | ✓ |

[1)] For optimized accesses, only symbolic addressing is possible

[2)] For S7-1200/1500 the TIMER and COUNTER data type is represented by IEC_TIMER and IEC_Counter.

## 4.2 No bit memory but global data blocks

**Advantages**

- Optimized global DBs are clearly more powerful than the bit memory address area that is not optimized for reasons of compatibility.

**Recommendation**

- The handling with bit memory is problematic, since every controller has a bit memory address area with a different size. Do not use bit memory for the programming but always global data blocks. This is how the program can always be used universally.

## 4.3 Programming of "clock bits"

### Recommendation

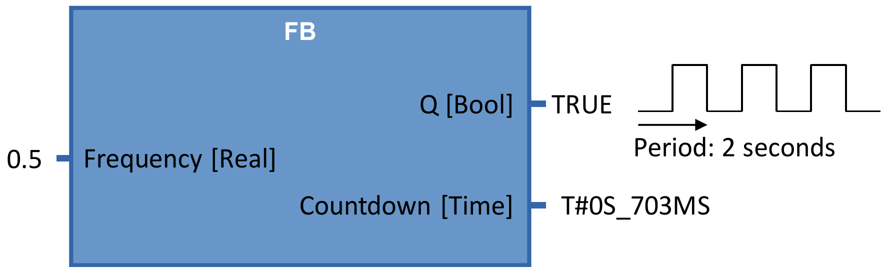
For the programming of clock memory bits, the hardware configuration always has to be correct.

Use a programmed block as clock generator. Below, you can find a programming example for a clock generator in the SCL programming language.

### Example

The programmed block has the following functions. A desired frequency is specified. The “Q” output is a Boolean value that toggles in the desired frequency. The “Countdown” output outputs the remaining time of the current state of “Q”.

If the desired frequency is smaller or equal 0.0, then the output Q = FALSE and Countdown = 0.0.



Here you can find the example code (SCL) for the function block:

```
FUNCTION_BLOCK "Frequency"
{ S7_Optimized_Access := 'TRUE' }
VERSION : 0.1
   VAR_INPUT
      Frequency : Real;    // Input in Hz
   END_VAR

   VAR_OUTPUT
      Q : Bool;
      Countdown : Time;
   END_VAR

   VAR
      TOF_on {OriginalPartName := 'IEC_TIMER'; VersionGUID :=
'b68d17d6-3fcc-4468-818a-b36d847990bb'} : TOF_TIME;
      TOF_off {OriginalPartName := 'IEC_TIMER'; VersionGUID
:= 'b68d17d6-3fcc-4468-818a-b36d847990bb'} : TOF_TIME;
      Frequency_mem : Real;
      Time_base : Time;
   END_VAR
```

```
  IF #Frequency <= 0.0 THEN // If input is <= 0.0 >>
Frequency block is off
    #Q:= FALSE;
    #Countdown:= t#0s;

  ELSE
  IF #Frequency_mem <> #Frequency THEN // apply frequency one
time
    #Time_base:= t#1000s / TRUNC(#Frequency*2000); //
calculate Time
    #Frequency_mem := #Frequency;
  END_IF;

  //TOF_off expired >> Start again
  #TOF_on(IN:= NOT #TOF_off.Q, PT:=#Time_base);
  #Countdown:= #Time_base - #TOF_on.ET;

  #TOF_off(IN:= #TOF_on.Q, PT:=#Time_base);

  IF #Countdown=t#0s THEN
   #Countdown:= #Time_base - #TOF_off.ET;
  END_IF;

  #Q := #TOF_on.Q;

  END_IF;

END_FUNCTION_BLOCK
```

# 5 The Most Important Recommendations

- Use optimized blocks
  - Chapter 2.5 Optimized blocks
- Structuring the program clearly and well
  - Chapter 3.2 Organization blocks (OB)
- Inserting instructions as multi-instance (TON, TOF ..)
  - Chapter 3.2.5 Multi-instances
- Reusable programming of blocks
  - Chapter 3.2.8 Reusability of blocks
- Symbolic programming
  - Chapter 3.6 Symbolic addressing
- When handling data, work with ARRAY
  - Chapter 3.6.2 ARRAY data type and indirect field accesses
- Creating PLC data types
  - Chapter 3.6.3 STRUCT data type and PLC data types
- Using libraries for storing program elements
  - Chapter 3.7 Libraries
- No memory bits but global data blocks
  - Chapter 4.2 No bit memory but global data blocks

# 6 Related Literature

Table 6-1

| | Topic | Title |
|---|---|---|
| \1\ | Siemens Industry Online Support | http://support.automation.siemens.com |
| \2\ | Download page of the entry | http://support.automation.siemens.com/WW/view/en/81318674 |
| \3\ | TIA Portal - An Overview of the Most Important Documents and Links | http://support.automation.siemens.com/WW/view/en/65601780 |
| \4\ | STEP 7 (TIA Portal) manuals | http://support.automation.siemens.com/WW/view/en/29156492/133300 |
| \5\ | S7-1200 Manuals | http://support.automation.siemens.com/WW/view/en/34612486/133300 |
| \6\ | S7-1500 Manuals | http://support.automation.siemens.com/WW/view/en/56926743/133300 |
| \7\ | S7-1200 Getting Started | http://support.automation.siemens.com/WW/view/en/39644875 |
| \8\ | S7-1500 Getting Started | http://support.automation.siemens.com/WW/view/en/78027451 |

# 7 History

Table 7-1

| Version | Date | Modifications |
|---|---|---|
| V1.0 | 09/2013 | First version |
| V1.1 | 10/2013 | Corrections in the following chapters:<br>2.5.3 Best possible data storage in the processor on S7-1500<br>2.11 Internal reference ID for controller and HMI tags<br>3.2.2 Functions (FC)<br>3.2.3 Function blocks (FB)<br>3.4.3 Local memory |