SIEMENS

Programming Guideline for S7-1200/S7-1500

STEP 7 (TIA Portal) and STEP 7 Safety in TIA Portal

http://www.siemens.com/simatic-programming-guideline

Warranty and Liability

Note

The Application Examples are not binding and do not claim to be complete with regard to configuration, equipment or any contingencies. The Application Examples do not represent customer-specific solutions. They are only intended to provide support for typical applications. You are responsible for the correct operation of the described products. These Application Examples do not relieve you of the responsibility of safely and professionally using, installing, operating and servicing equipment. When using these Application Examples, you recognize that we cannot be made liable for any damage/claims beyond the liability clause described. We reserve the right to make changes to these Application Examples at any time and without prior notice. If there are any deviations between the recommendations provided in this Application Example and other Siemens publications – e.g. catalogs – the contents of the other documents have priority.

We do not accept any liability for the information contained in this document.

Any claims against us – based on whatever legal reason – resulting from the use of the examples, information, programs, engineering and performance data etc., described in this application example will be excluded. Such an exclusion will not apply in the case of mandatory liability, e.g. under the German Product Liability Act ("Produkthaftungsgesetz"), in case of intent, gross negligence, or injury of life, body or health, guarantee for the quality of a product, fraudulent concealment of a deficiency or breach of a condition which goes to the root of the contract ("wesentliche Vertragspflichten"). The compensation for damages due to a breach of a fundamental contractual obligation is, however, limited to the foreseeable damage, typical for the type of contract, except in the event of intent or gross negligence or injury to life, body or health. The above provisions do not imply a change of the burden of proof to your detriment.

Any form of duplication or distribution of these Application Examples or excerpts hereof is prohibited without the expressed consent of Siemens Industry Sector.

Security information

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, solutions, machines, equipment and/or networks. They are important components in a holistic industrial security concept. With this in mind, Siemens' products and solutions undergo continuous development. Siemens recommends strongly that you regularly check for product updates.

For the secure operation of Siemens products and solutions, it is necessary to take suitable preventive action (e.g. cell protection concept) and integrate each component into a holistic, state-of-the-art industrial security concept. Third-party products that may be in use should also be considered. For more information about industrial security, visit http://www.siemens.com/industrialsecurity.

To stay informed about product updates as they occur, sign up for a product-specific newsletter. For more information, visit http://support.automation.siemens.com.

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
  - Data types and instructions
- Recommendation on general programming
  - Operating system and user program
  - Storage concept
  - Symbolic addressing
  - Libraries
- Recommendations on hardware-independent programming
- Recommendations on STEP 7 Safety in TIA Portal
- Overview of the most important recommendations

Advantages and benefits

Numerous advantages arise by applying these recommendations and tips:

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

Additional Information

When programming SIMATIC controllers, the task of the programmer is to create as clear and readable a user program as possible. Each user uses its own strategy, for example, how to name tags or blocks or the way of commenting. The different philosophies of the programmers create very different user programs that can only be interpreted by the respective programmer.

The programming styleguide offers you coordinated set of rules for consistent programming. These specifications for example describe a uniform assignment of tags and block names up to clear programming in SCL.

You can user these rules and recommendations at will; they serve as a suggestion (not a standard in programming) for consistent programming.

Hinweis	You will find the programming styleguide for S7-1200 and S7-1500 in the following entry:
	https://support.industry.siemens.com/cs/ww/en/view/81318674

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.

Note	Information on migrating S7-300/S7-400 to S7-1500 is available in the following entry: https://support.industry.siemens.com/cs/ww/en/view/109478811

2.2 Terms

General terms using TIA Portal

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

Figure 2-1: New terms in the TIA Portal

STEP 7 V5.x   STEP 7 (TIA Portal)

Symbol table → PLC tags
UDT → PLC data types
VAT table → Watch table
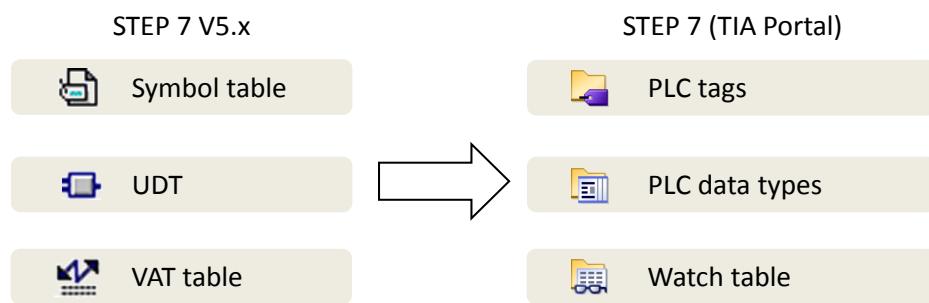
Terms for tags and parameters

When it is about tags, functions, and function blocks, many terms are repeatedly used differently or even incorrectly. The following figure is to clarify these terms.

Figure 2-2: Terms associated with tags and parameters
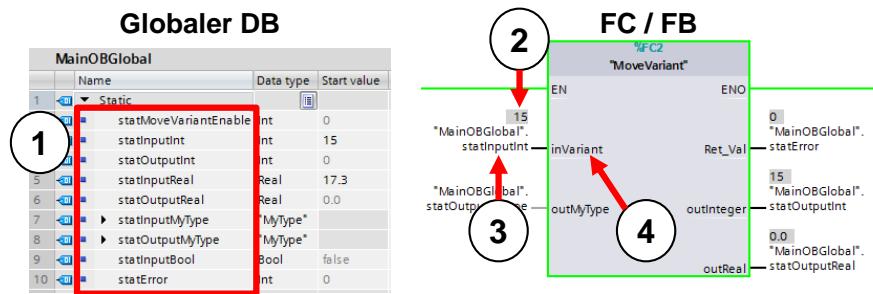
Table 2-1: Terms associated with tags and parameters

	Term	Description
1.	Tag	Tags are labeled by a name/identifier and assign an address in the memory of the controller. Tags are always defined with a certain data type (Bool, Integer, etc.): • PLC tags • Single tags in data blocks • Complete data blocks
2.	Tag value	Tag values are values stored in a tag (e.g., 15 as value of an Integer tag).
3.	Actual parameter	Actual parameters are tags interconnected at the interfaces of instructions, functions, and function blocks.
4.	Formal parameter (transfer parameter, block parameter)	Formal parameters are the interface parameters of instructions, functions, and function blocks (Input, Output, InOut, and Ret_Val).

2.2 Terms

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)? https://support.industry.siemens.com/cs/ww/en/view/56314851
	What prerequisites have to be fulfilled in order to migrate a STEP 7 V5.x project into STEP 7 Professional (TIA Portal)? https://support.industry.siemens.com/cs/ww/en/view/62100731
	PLC migration for S7-1500 with STEP 7 (TIA Portal) https://support.industry.siemens.com/cs/ww/en/view/67858106
	How do you efficiently and effectively program for the S7-1200/S7-1500 in STEP 7 (TIA Portal)? https://support.industry.siemens.com/cs/ww/en/view/67582299
	Why is it not possible to mix register passing and explicit parameter transfer with the S7-1500 in STEP 7 (TIA Portal)? Among others, the migration of STL programs to S7-1500 is described in this entry. https://support.industry.siemens.com/cs/ww/en/view/67655405

2.3 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.

Table 2-2: Programming languages

Programming language	S7-1200	S7-1500
Ladder (LAD)	✓	✓
Function block diagram (FBD)	✓	✓
Structured control language (SCL)	✓	✓
Graph	✗	✓
Statement list (STL)	✗	✓

Note	You will find further information in the following entries:
	SIMATIC S7-1200 / S7-1500 Comparison list for programming languages https://support.industry.siemens.com/cs/ww/en/view/86630375
	What has to be observed when migrating a S7-SCL program in STEP 7 (TIA Portal)? https://support.industry.siemens.com/cs/ww/en/view/59784005
	What instructions cannot be used in STEP 7 (TIA Portal) in an SCL program? https://support.industry.siemens.com/cs/ww/en/view/58002709
	How can the constants be defined under STEP 7 (TIA Portal) in a S7-SCL program? https://support.industry.siemens.com/cs/ww/en/view/52258437

2.4 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.

2.5 Block creation

Figure 2-3: Machine code generation with S7-300/400/WinAC and S7-1200/1500
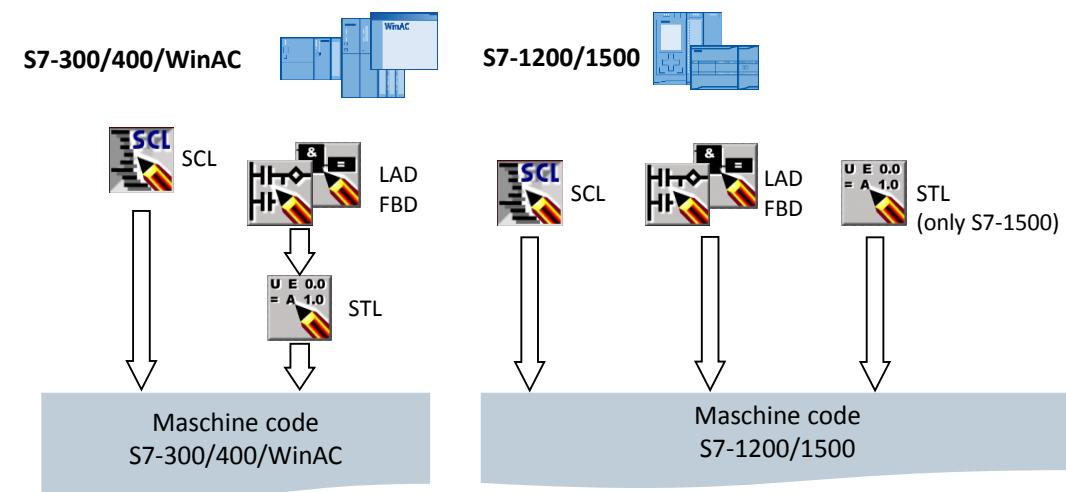
- For S7-300/400/WinAC controllers LAD and FBD programs are first of all compiled in STL before the machine code is created.
- For S7-1200/1500 controllers all programming languages are directly compiled into machine code.

2.5 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 2-4: "Add new block" dialog
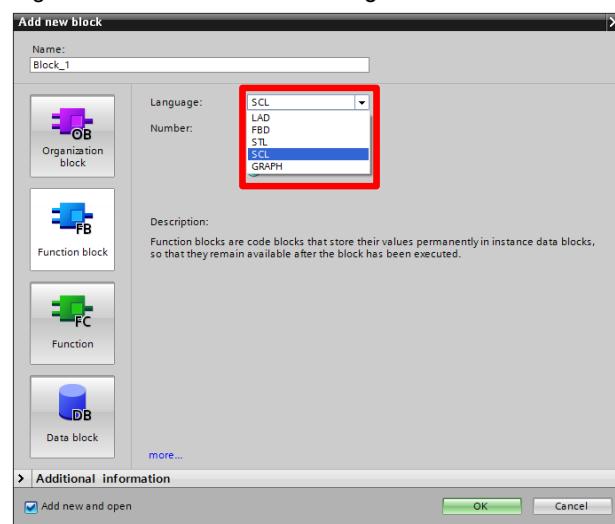
2.6 Optimized blocks

S7-1200/1500 controllers have optimized data storage. In optimized blocks, all tags are automatically sorted by their data type. The sorting ensures that data gaps between the tags are minimized and that the tags 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)

2.6.1 S7-1200: Setup of optimized blocks

Figure 2-5: Optimized block of S7-1200
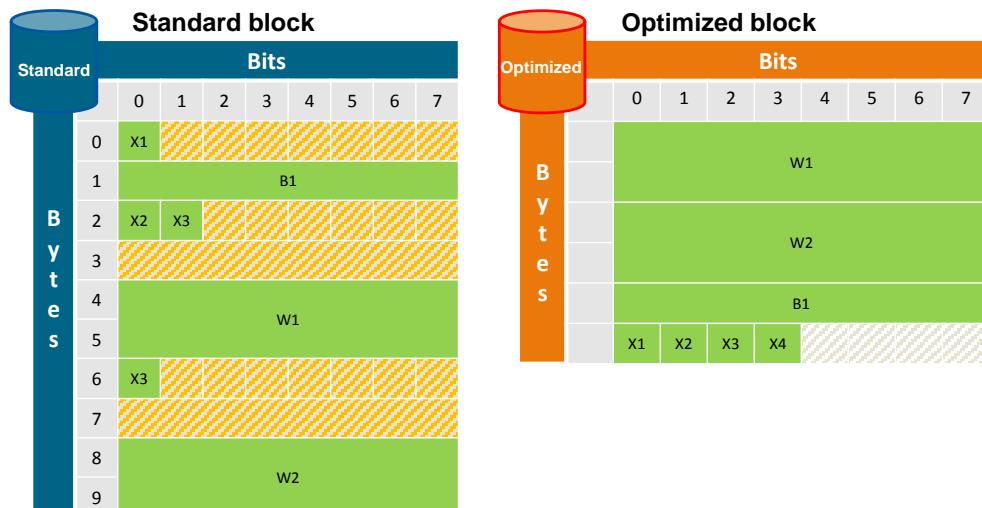
Properties

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

- Only the symbolic access exists for optimized blocks.

2.6.2   S7-1500: Setup of optimized blocks

Figure 2-6: Optimized block of S7-1500
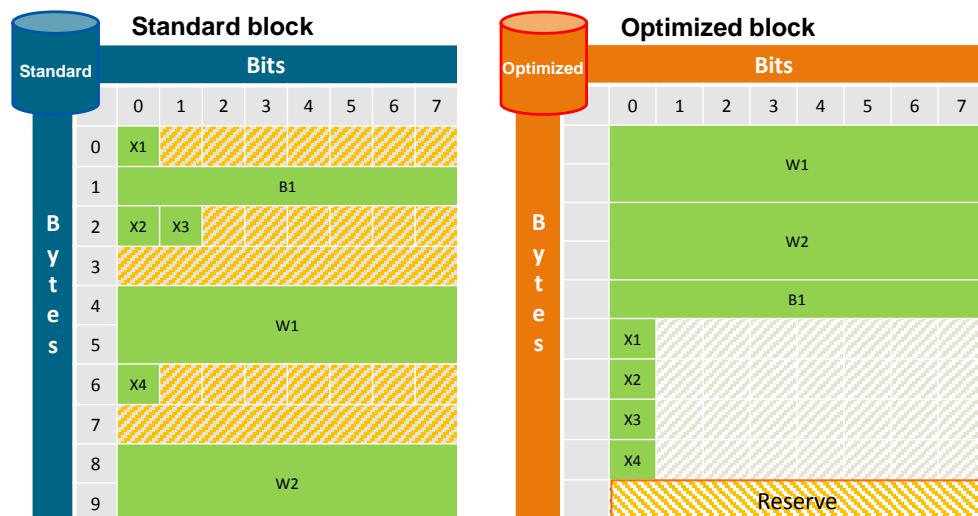
Figure 2-7: Memory space assignment in optimized blocks
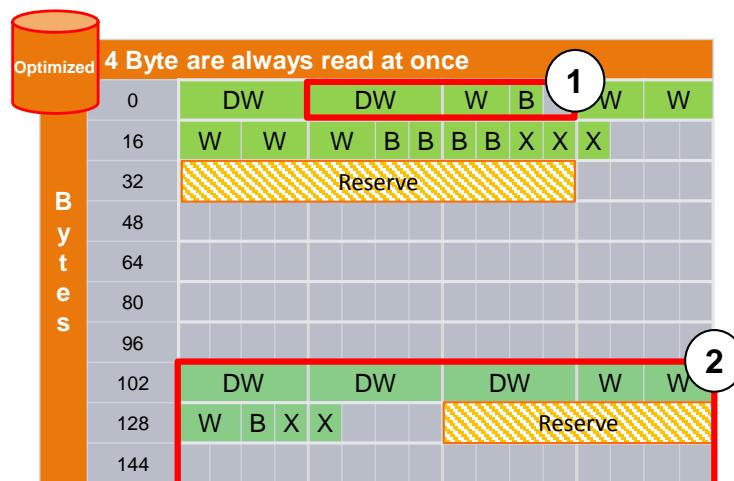
1. Structures are stored separately and can thus be copied as one block.
2. Retentive data are stored in a separate area and can be copied as one block. In the event of a power failure, these data are stored CPU-internally. "MRES" resets these data to the start values stored in the load memory.

Properties

- No data gaps are formed since larger tags 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 tags are stored in a way so that the processor of the S7-1500 can directly read or write all tags with just one machine command).

- Boolean tags are stored as byte for faster access. The controller therefore does not have to mask the access.

2.6 Optimized blocks

- 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.

2.6.3 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 2-8: Data access of a S7-1500 controller
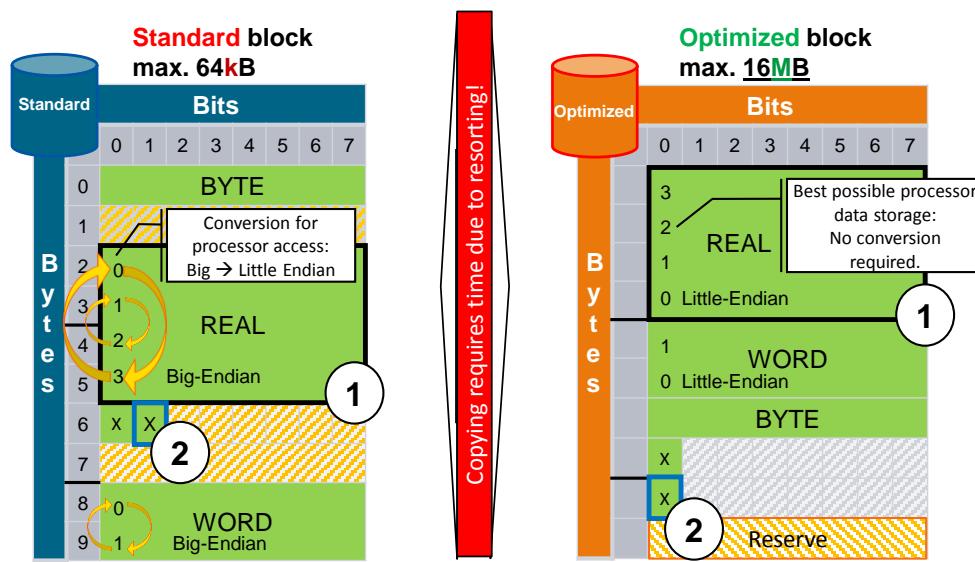
Table 2-3: Data access of a S7-1500 controller

	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). In addition the bytes have to be changed.	The controller stores the tags, access optimized. An access is performed with 32 bit (REAL). A changing of the bytes is not necessary.
2.	The complete byte is read and masked per bit access. The complete byte is blocked for any other access.	Each bit is assigned a byte. 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.

2.6 Optimized blocks

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/assigning of data between optimized and non-optimized blocks. The required conversion between source and destination format requires high processing time.
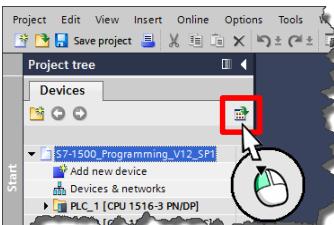
Example: Setting optimized block access

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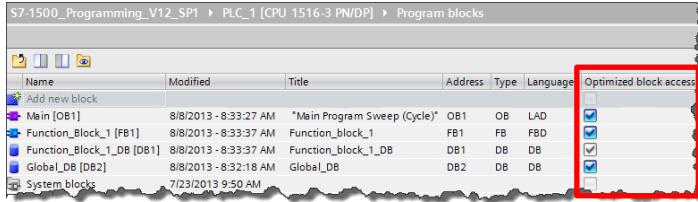
The block access is not reset automatically when a block is migrated from a S7-300/400 controller to a S7-1200/1500. You can change the block access later on to "optimized block access". You need to recompile the program after changing the block access. If you change the FBs to "optimized block access",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

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

Table 2-4: Setting optimized block access

Step	Instruction
1.	Click the "Maximizes/minimizes the Overview" button in the project navigation.
2.	Navigate to "Program blocks".

2.6 Optimized 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. 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.

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 2-9: Optimized data block (without offset)

Figure 2-10: Non-optimized data blocks (with offset)

Table 2-5: Difference: optimized and non-optimized data block

Optimized data block	Non-optimized data block
Optimized data blocks are addressed symbolic. **No** "offset" is displayed.	At non-optimized blocks an "offset" is displayed and can be used for addressing.
In optimized blocks **every** tag can be declared with "Retain".	In non-optimized blocks only **all or no** tags can be declared with "Retain".

2.6 Optimized blocks

The retentivity of tags of a global DB is defined directly in the global DB. The default setting is non-retentive.

The retentivity of tags of one instance is defined in the function block (not in the instance DB).These settings then apply to all instances of this FB.

Access types for optimized and non-optimized blocks

The following table displays all access types to blocks.

Table 2-6: 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 entries:
	What differences should you watch out for between optimized data storage and the standard type of block access in STEP 7 (TIA Portal)? https://support.industry.siemens.com/cs/ww/en/view/67655611
	What properties do you have to pay attention to in STEP 7 (TIA Portal) for the instructions "READ_DBL" and "WRIT_DBL", when you are using DBs with optimized access? https://support.industry.siemens.com/cs/ww/en/view/51434747

2.6.4 Conversion between optimized and non-optimized tags

The general recommendation is to work with optimized tags. However, if you want to keep your previous programming in individual cases, this leads to a mixture of optimized and non-optimized data storage in the program.

The system recognizes the internal storage of each tag, no matter if structured (derived from a user-defined data type) or elementary (INT, LREAL, etc.).

In the case of type-identical allocations between two tags with different storage locations, the system converts automatically. In the case of structured tags, this conversion requires performance and should therefore be avoided, if possible.

2.6.5 Parameter transfer between blocks with optimized access and standard access

If during a block call, structures are transferred to the called block as InOut parameter, these are by default transferred as reference (see chapter 3.3.2 Call-by-reference with InOut interface type).

However, this does not apply if one of the blocks has the property "Optimized access" and the other block the property "Standard access". Principally, all parameters are then transferred as copies (see chapter 3.3.1 Call-by-value with In interface type).

In this case, the called block always works with the copied values. During block processing, these values might be changed and, after processing the block call, be copied back to the original operand.

This may cause problems, if the original operands are changed by asynchronous processes, e.g. by HMI access or alarm OBs. If, after processing the block, the copies are returned to the original operand, the asynchronously performed changes at the original operands are overwritten.

NOTE	You will find further information in the following entry:
	Why is it possible that data of HMI system or web server are overwritten in S7-1500? https://support.industry.siemens.com/cs/ww/en/view/109478253

Recommendation:

- Always set the same access type for both blocks that communicate with each other.

2.6.6 Communication with optimized data

The interface (CPU, CM) transfers the data as they are arranged (no matter if optimized or not).

Figure 2-11: CPU-CPU communication
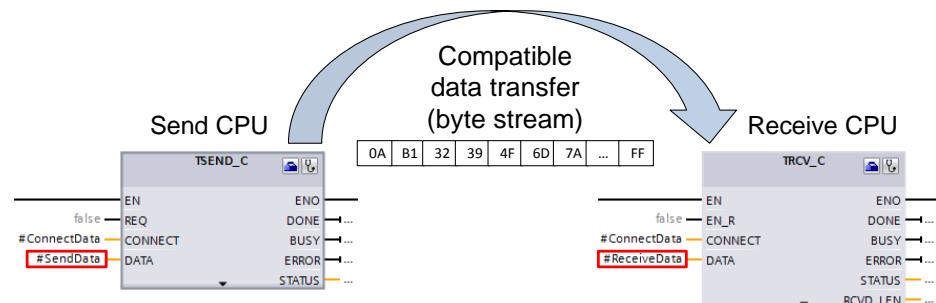
Send data can be:
- optimized
- not optimized
- Tag (any type)
- Buffer (byte array)

Receive data can be:
- optimized
- not optimized
- Tag (any type)
- Buffer (byte array)

Example

- A tag with data type PLC (data record) is to be transferred to a CPU.
- In the send CPU, the tag is interconnected as actual parameter with the communication block (TSEND_C).
- In the receive CPU, the receive data are assigned to a tag of the same type.
- In this case, it is possible to symbolically continue to work directly with the received data.

Note	Any tags or data blocks (derived from PLC data types) can be used as data records.

Note	It is also possible that the send and receive data are not defined identically:

Send data		Receive data
optimized	-->	not optimized
not optimized	-->	optimized

The controller automatically provides for correct data transmission and storage.

2.7 Block properties

2.7.1 Block sizes

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

Table 2-7: 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) **10 MB** (optimized CPU1518)
	Max. number	16.000	65.535	65.535
FC/FB	Max. size	64 kB	64 kB	512 kB **3 MB** (optimized CPU1518)
	Max. number	7.999	65.535	65.535
FC / FB / DB	Max. number	4.096 (CPU319) 6.000 (CPU412)	1.024	10.000 (CPU1518)

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 10 MB) with S7-1500 controllers.

2.7.2 Number of organization blocks (OB)

OBs can be used for creating a hierarchical structure of the user program. Different OBs are available for this purpose.

Table 2-8: Number of organization blocks

Organization block type	S7-1200	S7-1500	Benefit
Cyclic and startup OBs	100	100	Modularization of the user program
Hardware interrupt	50	50	Separator OB possible for each event
Time delay interrupt	4*	20	Modularization of the user program
Cyclic interrupt		20	Modularization of the user program
Time of day	✗	20	Modularization of the user program

* from firmware V4 on 4 delay interrupts and 4 watchdog interrupts each possible.

Recommendation

- Use OBs for structuring the user program hierarchically.
- For further recommendations on using OBs refer please to Chapter 3.2.1 Organization blocks (OB).

2.8 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? https://support.industry.siemens.com/cs/ww/en/view/48711306

2.8.1 Elementary data types

Table 2-9: 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 Trio (10^{18})
LInt*	64 bit	-9,2 Trio .. 9,2 Trio
LWord	64 bit	16#0000 0000 0000 0000 bis 16# FFFF FFFF FFFF FFFF

* only for S7-1500

Table 2-10: Floating-point decimal data types

Type	Size	Value range
Real	32 bit (1 bit signs, 8 bit exponent, 23 bit mantissa), accurate to 7 decimal places	-3.40e+38 .. 3.40e+38
LReal	64 bit (1 bit signs, 11 bit exponent, 52 bit mantissa), accurate to 15 decimal places	-1.79e+308 .. 1.79e+308

Note	The TIA Portal contains the global library "Long Functions" with a great scope of instructions for long data types.
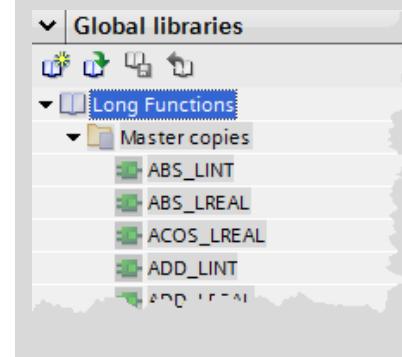
Note	You will find further information in the following entry: Why, in STEP 7 (TIA Portal), is the result of the DInt Addition in SCL not displayed correctly? https://support.industry.siemens.com/cs/ww/en/view/98278626

2.8.2 Date_Time_Long data type

Table 2-11: 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 (e.g. `My_Timestamp.Hour`).

Note	You will find further information in the following entries: In STEP 7 (TIA Portal), how can you input, read out and edit the date and time for the CPU modules of S7-300/S7-400/S7-1200/S7-1500? https://support.industry.siemens.com/cs/ww/en/view/43566349 Which functions are available in STEP 7 V5.5 and in TIA Portal for processing the data types DT and DTL? https://support.industry.siemens.com/cs/ww/en/view/63900229

2.8.3 Further time data types

Table 2-12: Time data types (only S7-1500)

Type	Size	Value range
LTime	64 bit	LT#-106751d23h47m16s854ms775us808ns up to LT#+106751d23h47m16s854ms775us807ns
LTIME_OF_DAY	64 bit	LTOD#00:00:00.000000000 up to LTOD#23:59:59.999999999

2.8 New data types for S7-1200/1500

2.8.4 Unicode data types

Data types WCHAR and WSTRING can be processed using unicode characters.

Table 2-13: Time data types (only S7-1500)

Type	Size	Value range
WCHAR	2 bytes	-
WSTRING	(4 + 2*n) byte	Preset value: 0 ..254 characters Max. value: 0 ..16382

n = length of the character chain

Properties

- Processing characters in Latin Chinese or other languages, for example.
- Line breaks, page feed, tabulator, space character
- Special characters: Dollar sign, quotation marks

Example

- `WCHAR#'a'`
- `WSTRING#'Hello World!'`

2.8.5 VARIANT data type (S7-1500 and S7-1200 from FW4.1)

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. 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 2-12: VARIANT data type as input parameter for the TSEND_C instruction
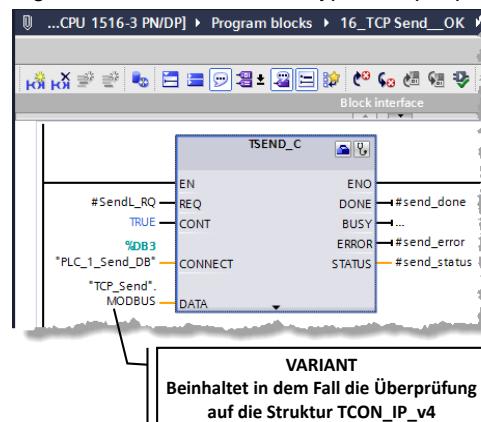
Advantages

- Integrated type test prevents faulty access.
- Due to the symbolic addressing of the variant tags, the code can be read easier.
- Code can be programmed more efficiently and within a shorter time.
- Variant pointers are clearly more intuitive than ANY pointers.
- Variant tags can be used directly using system functions.
- Flexible and performant transfer of differently structured tabs is possible.

Properties

A comparison between ANY and Variant makes the properties apparent.

Table 2-14: Comparison ANY and Variant

ANY	Variant
Requires 10 Kbytes of memory with defined structure	Does not require any memory space for the user
Initialization either via assignment of the data area or by means of filling the ANY structure	Initialization by means of assigning the data area of system instruction
Non-typed – Type of an interconnected structure cannot be read out	Typed – Interconnected type as well as the length for arrays can be determined
Partially typed – Length for arrays can be determined	VARIANT can also be evaluated and created via system instructions

2.8 New data types for S7-1200/1500

Recommendation

- Only use the VARIANT data type for indirect addressing if the data types are not determined until the program is running.

- Check what you have used the ANY pointer for so far. In many cases, a pointer is not required (see table below).

- Only use the VARIANT data type for indirect addressing if the data types are not determined until the program is running.

  - Using data type VARIANT as InOut formula parameter for creating generic blocks which are independent of the data type of the actual parameters (see example in this chapter).

  - Use the VARIANT data type instead of the ANY pointer. Due to the integrated type test, errors are detected early on. Due to the symbolic addressing, the program code can be easily interpreted.

  - Use the Variant instruction, for example, for type recognition (see following example and Chapter 2.9.3 VARIANT instructions)

- Use the indexed ARRAYs instead of ANY pointer to address ARRAY elements (see chapter 3.6.2 ARRAY data type and indirect field accesses).

Table 2-15: Comparison ANY pointer and simplification

What are ANY pointers used for?		Simplification with S7-1200/1500
Programming functions which can process different data types	→	Functions with Variant pointer as InOut parameter for blocks (see following examples)
Processing of arrays • e.g. reading, initializing, copying of elements of the same type	→	Standard array functions • Reading and writing with #myArray[#index] (see chapter 3.6.2 ARRAY data type and indirect field accesses) • Copy with MOVE_BLK (see chapter 2.9.2 MOVE instructions)
Transferring structures and performant processing • e.g. transferring user-defined structure by means of ANY pointers to functions	→	Transferring structures as InOut parameters • see chapter 3.3.2 Call-by-reference with InOut interface type

Example

With data type VARIANT it is possible to recognize data types in the user program and react accordingly. The following code of FC "MoveVariant" shows a possible way of programming.

- The InOut formal parameter "InVar" (data type VARIANT) is used to show a tag independent of the data type.

- The data type of the actual parameter is recognized with the "Type_Of" instruction.

- Using the "MOVE_BLK_VARIANT" instruction, the tag value is copied to the different output formal parameters depending on the data type.

Figure 2-13: Formal parameter of FC "MoveVariant"

		Name	Data type	Default value	Comment
		MoveVariant			
1	▶	Input			
2	▼	Output			
3	■	OutInteger	Int		Integer data
4	■	OutReal	Real		Real data
5	▶	OutMyType	"MyType"		User defined PLC data type
6	▼	InOut			
7	■	InOutVariant	Variant		Variable data input
8	▶	Temp			
9	▼	Constant			
10	■	NO_CORRECT_DATA_TYPE	Word	16#80B4	
11	▼	Return			

```
CASE TypeOf(#InOutVariant) OF // Check datatypes
   Int:  // Move Integer
      #MoveVariant := MOVE_BLK_VARIANT(SRC := #InOutVariant,
                                       COUNT := 1,
                                       SRC_INDEX := 0,
                                       DEST_INDEX := 0,
                                       DEST => #OutInteger);
   Real:  // Move Real
      #MoveVariant := MOVE_BLK_VARIANT(SRC := #InOutVariant,
                                       COUNT := 1,
                                       SRC_INDEX := 0,
                                       DEST_INDEX := 0,
                                       DEST => #OutReal);
   MyType: // Move MyType
      #MoveVariant := MOVE_BLK_VARIANT(SRC := #InOutVariant,
                                       COUNT := 1,
                                       SRC_INDEX := 0,
                                       DEST_INDEX := 0,
                                       DEST => #OutMyType);
   ELSE  // Error, no sufficient data type
      #MoveVariant := WORD_TO_INT(#NO_CORRECT_DATA_TYPE);
      // 80B4: Error code of MOVE_BLK_VARIANT: Data types do
         not correspond
END_CASE;
```

NOTE   If you want to copy values of not structured VARIANT tags, you can also use VariantGet instead of MOVE_BLK_VARIANT (see chapter 2.9.3 VARIANT instructions).

2.9 Instructions

2.9.1 CALCULATE

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 2-14: CALCULATE instruction with formula editor
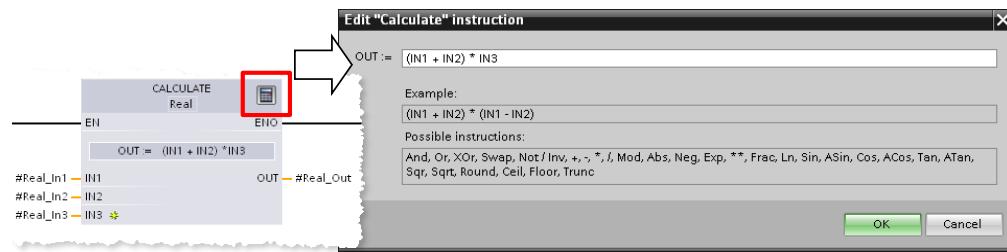
Note	For more information refer to the Online Help of the TIA Portal with the "CALCULATE" instruction.

Advantages

- A mathematical formula only needs one instruction
- Time saving due to simple configuration

Properties

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

Recommendation

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

2.9.2 MOVE instructions

STEP 7 (TIA) provides the following MOVE instructions. The instruction MOVE_BLK_VARIANT for S7-1200/1500 is new.

Table 2-16: Move instructions

Instruction	Typical use	Properties
MOVE	Copy value Copy array	• Copy the content of the parameter at the IN input to the parameter of the OUT output. • The parameters at the input and output must be of the same data type. • The parameters can also be structured tags (PLC data types). • Copy complete arrays and structures.
MOVE_BLK	Copy several areas	• Copy the content of an array to another array. • The source and target array must be of the same data type. • Copy complete arrays and structures. • Copy several array elements with structures as well. In addition, start and number of elements can be assigned.
UMOVE_BLK	Copy array without interruption	• Copies the content of an array consistently without the risk of the OB interrupting the copying process. • The source and target array must be of the same data type.
MOVE_BLK_VARIANT (S7-1500 and S7-1200 from FW4.1)	Copy array	• Copies one or several structured tag(s) (PLC data types). • Recognizes data types at runtime • Supplies detailed error information • Apart from the elementary and structured data types, PLC data types, arrays, and array DBs are also supported.
Serialize (S7-1500 and S7-1200 from FW4.1)	Copy structured data into byte array	• Several data records can be combined into a single byte array and be sent to other devices as a message frame. • Input and output parameters can be transferred as data type Variant.
Deserialize (S7-1500 and S7-1200 from FW4.1)	Copy from a byte array into one/several structure/s	• Application case I-Device: The I-Device receives several data records in the input area which are copied to different structures. • Several data records can be combined into a single byte array. Deserialize enables copying these to different structures.

2.9 Instructions

Figure 2-15: Serialize and deserialize (S7-1500 and S7-1200 from FW4.1)
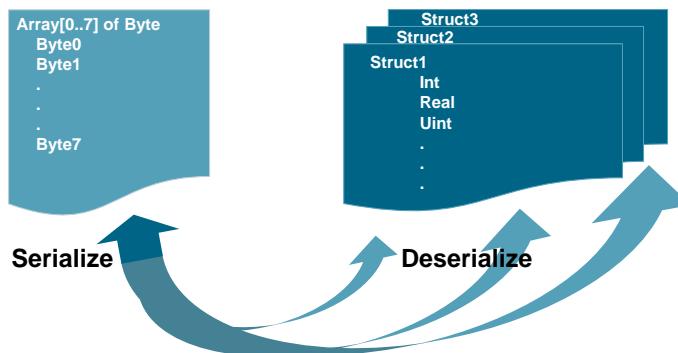
Recommendation

- Generally, you need to distinguish between MOVE, MOVE_BLK and MOVE_BLK_VARIANT
  - Use the MOVE instruction to copy complete structures.
  - Use the MOVE_BLK instruction to copy parts of an ARRAY of a known data type.
  - Only use the MOVE_BLK_VARIANT instruction if you wish to copy parts of ARRAYs with data types which are only known during program run-time.

Note	UMOVE_BLK: The copy process cannot be interrupted by another activity of the operating system. Therefore, the alarm reaction times of the CPU might increase during processing of the instruction "Copy array without interruption".
	For the complete description of the MOVE instructions, please refer to the TIA Portal Online Help.

Note	You will find further information in the following entry:
	In STEP 7 (TIA Portal) how do you copy memory areas and structured data from one data block to another? https://support.industry.siemens.com/cs/ww/en/view/42603881

2.9.3 VARIANT instructions (S7-1500 and S7-1200 from FW4.1)

Table 2-17: Move instructions

Instruction	Typical use	Properties
MOVE operations		
VariantGet	Read value	This instruction enables you to read the value of a tag pointing to a VARIANT.
VariantPut	Write value	This instruction enables you to write the value of a tag pointing to a VARIANT.
List		
CountOfElements	Counting elements	With this instruction you poll the number of ARRAY elements of a tag pointing to a VARIANT.
Compare operations		
TypeOf() (only SCL)	Determining the data type	Use this instruction to poll the data type of a tag pointing to a VARIANT.
TypeOfElements() (only SCL)	Determining the array data type	Use this instruction to poll the data type of the ARRAY elements of a tag pointing to a VARIANT.
Conversion operations		
VARIANT_TO_DB_ANY (only SCL)	Determining the data block number	With this instruction you poll the data block number of an instance data block of a PLC data type, system data type, or array DB.
DB_ANY_TO_VARIANT (only SCL)	Creates a Variant tag from a data block.	With this instruction you create a Variant tag from an instance data block of a PLC data type, system data type, or array DB.

Note	For more VARIANT instructions, please refer to the online help of the TIA Portal.

2.9.4 RUNTIME

Using the "RUNTIME" instruction you measure the runtime of the complete program, single blocks or the command sequences. You can call this instruction in SCL (S7-1200/S7-1500) and in STL (S7-1500).

Note	You will find further information in the following entry: With S7-1200/S7-1500, how do you measure the time of a program section or the complete program cycle at runtime? https://support.industry.siemens.com/cs/ww/en/view/87668055

2.10 Symbolic and comments

2.10.1 Programming editor

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. Network title comments are visible even if networks are collapsed.

- Design the program code in a way so that colleagues can understand the program straight away.

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

Example

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

Figure 2-16: Commenting in the user program (LAD)
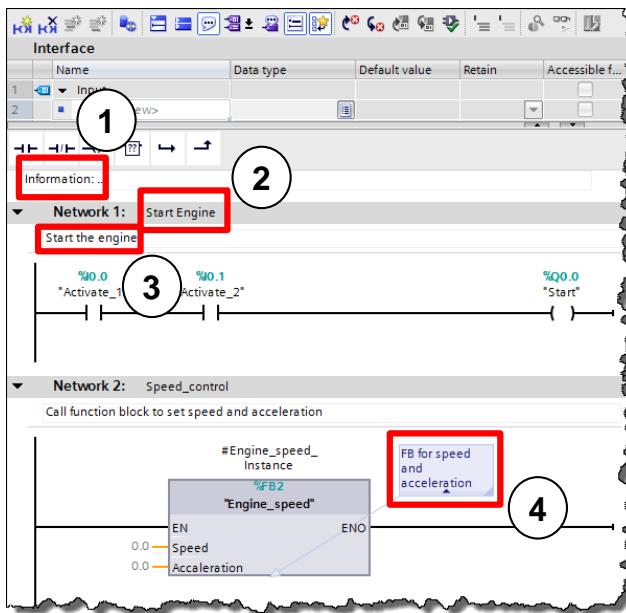
The following comments are possible:
1. Block comment
2. Network title comment
3. Network comment
4. Comment on instructions, blocks and functions (open, close, etc.)

2.10 Symbolic and comments

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
```

Note	For further information, refer to the following entry:
	In STEP 7 (TIA Portal), why are the display texts, titles and comments no longer displayed after opening the project in the block editor? https://support.industry.siemens.com/cs/ww/en/view/41995518

2.10.2 Comment lines in watch table

Advantages

- For better structuring it is possible to create comment lines in the watch table.

Recommendation

- Always use comment lines and sub-divide your watch table.
- Please also comment on the individual tags.

Example

Figure 2-17: Watch table with comment lines

2.11 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 the modules easily during programming.

Example

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

Figure 2-18: "System constants" in the user program
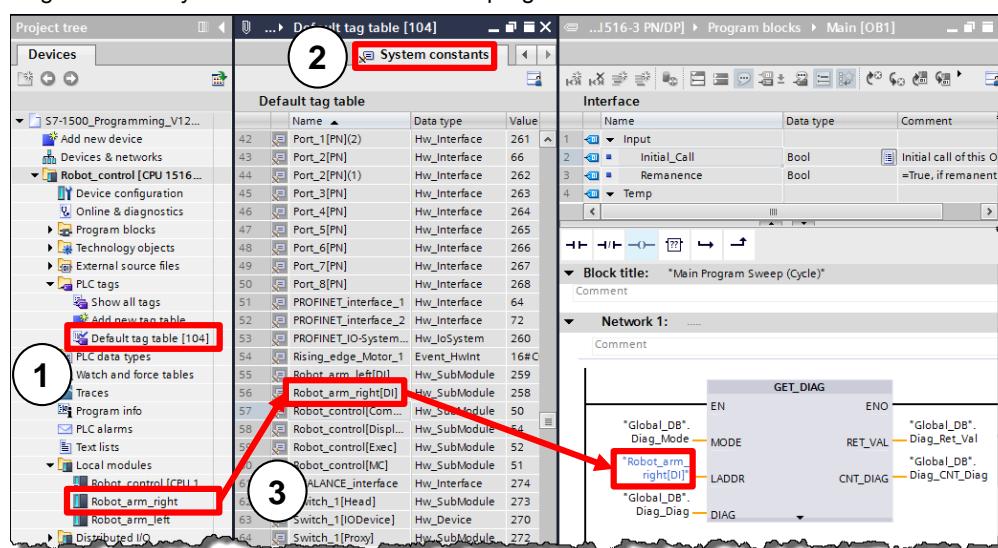
1. System constants of a controller can be found in the "PLC tags – Default tag table" folder.

2. The system constants are in a separate tab 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 tab.
   In the user program "Robot_arm_left" is interconnected with the "GET_DIAG" diagnostic block.

2.12 User constants

Note	Open the "Device configuration" to quickly find the system constant for each device.
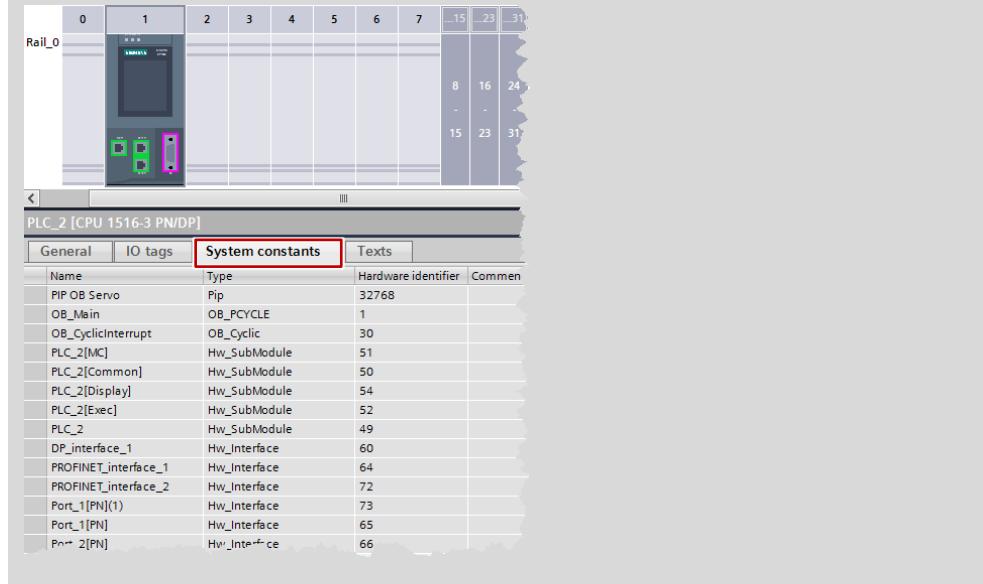
Note	You will find further information in the following entry: What meaning do the system constants have for the S7-1200/1500 in STEP 7 (TIA Portal)? https://support.industry.siemens.com/cs/ww/en/view/78782835

2.12 User constants

Using user constants, constant values can be saved. Generally, there are local constants for OBs, FCs and FBs and global constants for the entire user program in a controller.

Advantages

- User constants can be used for changing constant values globally or locally for all usage locations.
- With user constants, the program can be made more readable.

Properties

- Local user constants are defined in the block interface.
- Global user constants are defined at "PLC tags".
- The user program only enables read access to the user constants.
- For know-how protected blocks the user constants are not visible.

Recommendation

- Use the user constants for improved readability of the program and central changeability of …
    - error codes,
    - CASE instructions,
    - conversion factors,
    - natural constants ...

Example

Figure 2-19: Local user constant of a block for CASE instructions

Figure 2-20: Global user constant of a controller

Note	Another application case of constants is available at the following FAQ:
	How can you convert the unit of a tag in STEP 7 (TIA Portal)? https://support.industry.siemens.com/cs/ww/en/view/61928891

2.13 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.

2.13 Internal reference ID for controller and HMI tags

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

Figure 2-21: 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
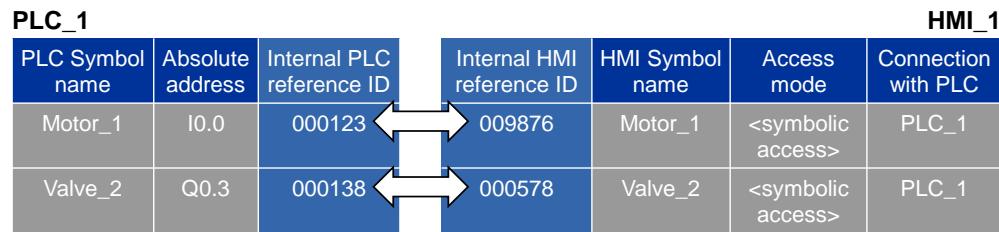
| Note	The ID will be changed if … • name is changed. • type is changed. • tag is deleted.

Advantages

- You can rewire tags without changing internal relations. The communication between controller, HMI and drive also remains unchanged.
- 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 2-22: Changing address or adding row).

Figure 2-22: Changing address or adding row
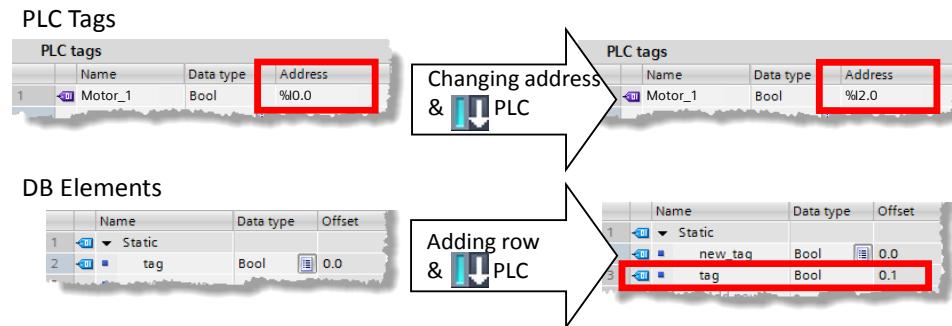
2.14 STOP mode in the event of errors

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 2-18: 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
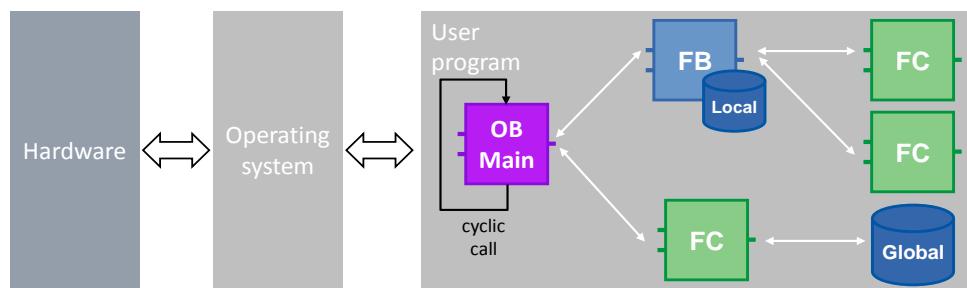
For SIMATIC controllers the user program is always executed cyclically. The "Main" cycle OB already exists in the "Program blocks" folder after a controller was created in STEP 7. The block is processed by the controller and recalled in an infinite loop.

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. Error states in a plant can be more easily detected, analyzed and removed. 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 sub 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 "third party companies".

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

Table 3-1: Programming languages

Programming language	S7-1200	S7-1500
Ladder (LAD)	✓	✓
Function block diagram (FBD)	✓	✓
Structured control language (SCL)	✓	✓
Graph	✗	✓
Statement list (STL)	✗	✓

3.2.1 Organization blocks (OB)

Figure 3-2: "Add new block" dialog (OB)
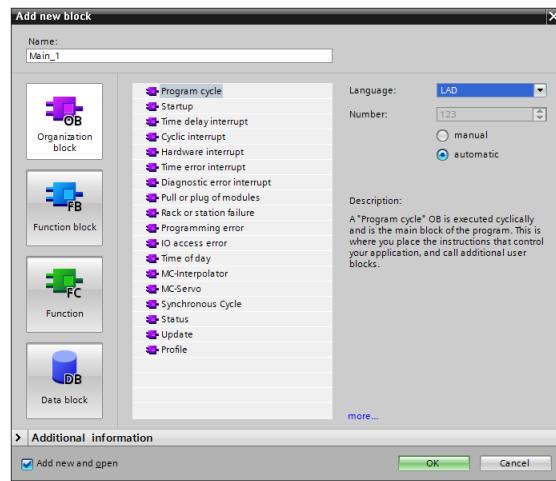
OBs are the interface between the operating system and the user program. They are called by the operating system and control, e.g. the following processes:

- 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.

3.2 Program blocks

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
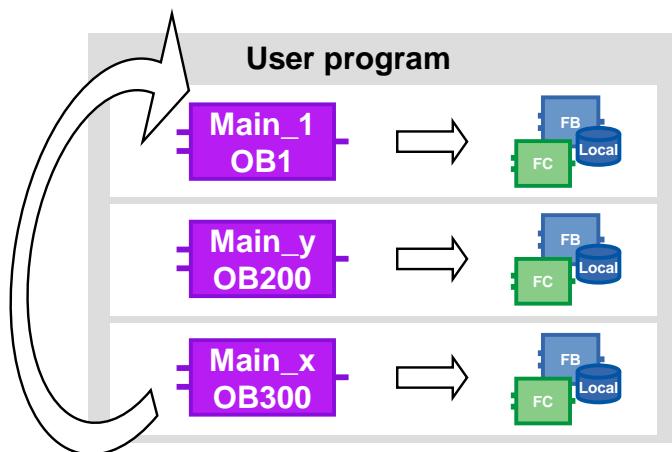
Recommendation

- 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.

3.2 Program blocks

Figure 3-4: Storing program parts in order in the project library
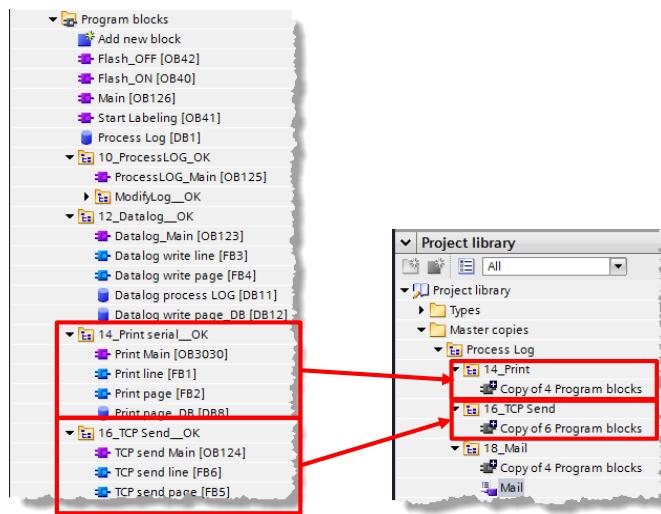
For further information, please refer to chapter 3.7 Libraries.

Note	You will find further information in the following entry: Which organization blocks can be used in STEP 7 (TIA Portal)? https://support.industry.siemens.com/cs/ww/en/view/40654862

3.2.2 Functions (FC)

Figure 3-5: "Add new block" dialog (FC)
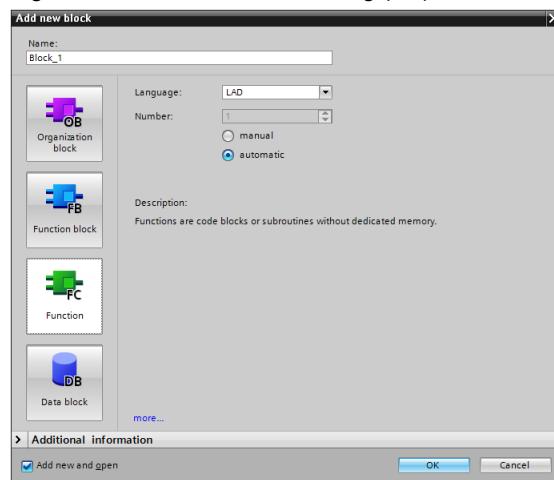
FCs are blocks without cyclic data storages. This is why the values of block parameters cannot be saved until the next call and has to be provided with actual parameters when called.

Properties

- FCs are blocks without cyclic data storages.
- Temporary tags are undefined when called in non-optimized blocks. In optimized blocks, the values are always preset to the default value (S7-1500 and S7-1200 Firmware V4). Thus, the resulting behavior is not accidental but reproducible.
- 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);
  ```
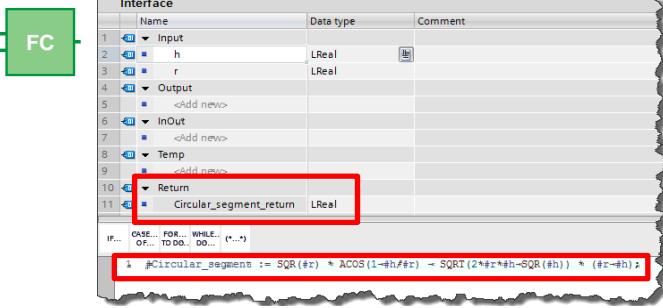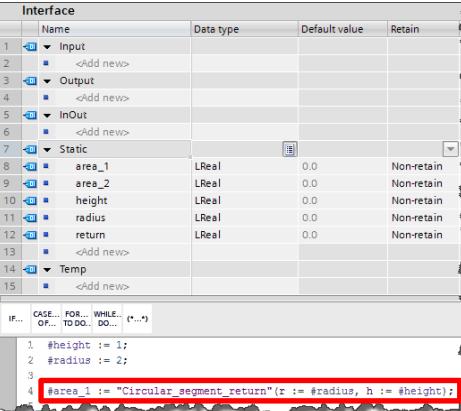
3.2 Program blocks

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 3-2: Reusing function value

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

Note You will find further information in the following entry:

What is the maximum number of parameters you are allowed to define in STEP 7 (TIA Portal) for a function in the S7-1200/S7-1500 CPU?
https://support.industry.siemens.com/cs/ww/en/view/99412890

3.2.3 Function blocks (FB)

Figure 3-6: "Add new block" dialog (FB)
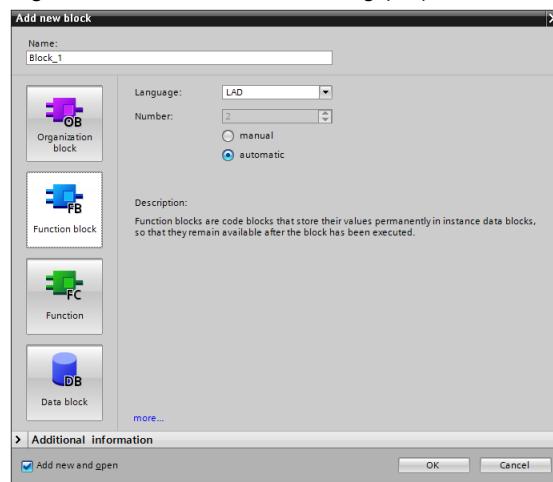
FBs are blocks with cyclic data storage, in which values are permanently stored. The cyclic data storage is realized in an instance DB.

Figure 3-7: Calling a function block
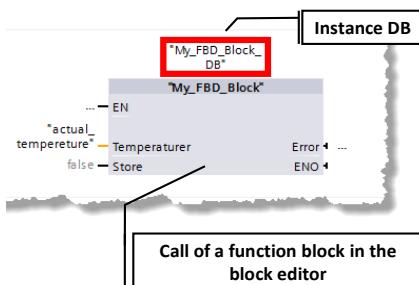
Properties

- FBs are blocks with cyclic data storage.
- Temporary tags are undefined when called in non-optimized blocks. In optimized blocks, the values are always preset to the default value (S7-1500 and S7-1200 Firmware V4). Thus, the resulting behavior is not accidental but reproducible.
- 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.
- If function blocks are applied multiply in the user program, use separate instances, preferably multi-instances.

3.2.4 Instances

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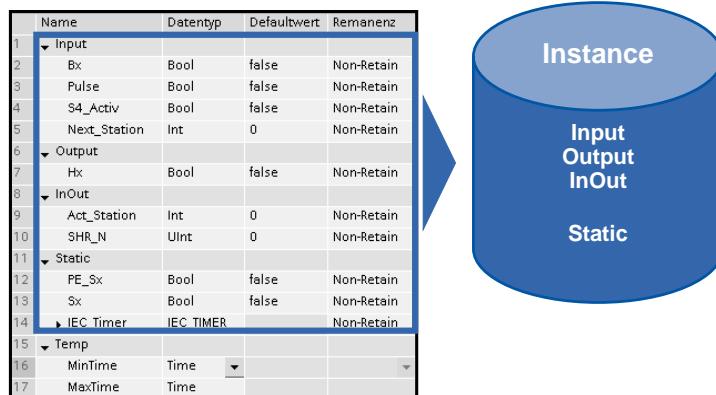
The instance DB consists of a permanent memory with the interfaces input, output, InOut and static. In a volatile memory (L stack) temporary tags are stored. The L stack is always only valid for the current process. I.e. temporary tags have to be initialized in each cycle.

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 functionality of the called block is thus maintained even if it is transferred.

The following figure shows an FB that uses another FB ("IEC Timer"). All data is saved in a multi instance DB. It is thus possible to create a block with an independent time behavior, for example, a clock generator.

3.2 Program blocks

Figure 3-9: Multi-instances
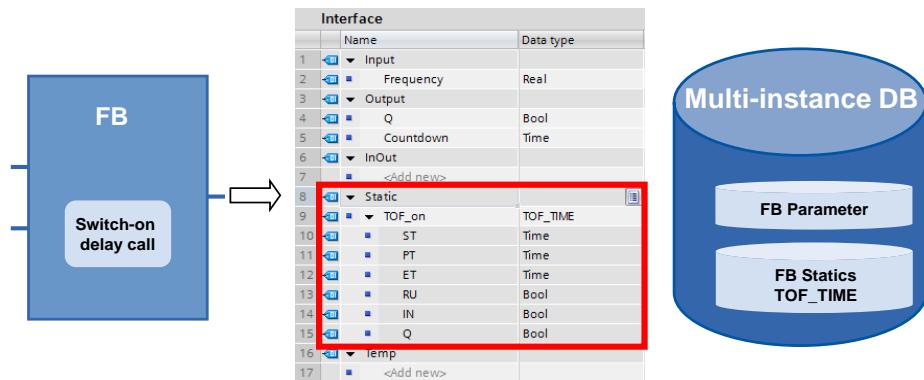
Advantages

- 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 instance DBs.
- create reusable and clear user programs.
- program local functions e.g. timer, counter, edge detection.

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.

3.2 Program blocks

Figure 3-10: Library of the IEC Timer
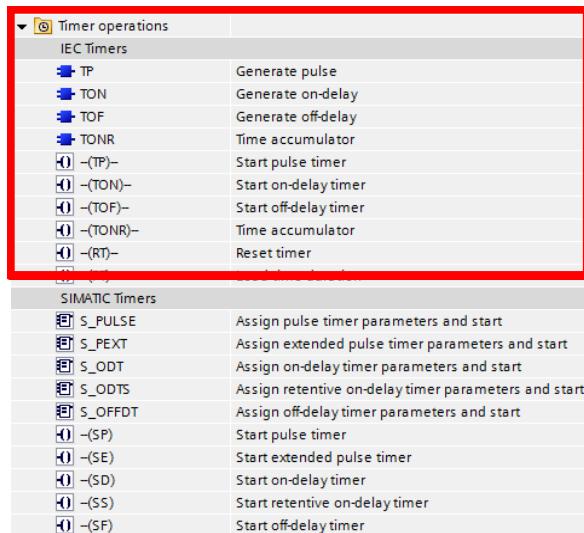
| Note	You will find further information in the following entry:
	How do you declare the timers and counters for the S7-1500 in STEP 7 (TIA Portal)? https://support.industry.siemens.com/cs/ww/en/view/67585220

3.2.6 Global data blocks (DB)

Figure 3-11: "Add new block" dialog (DB
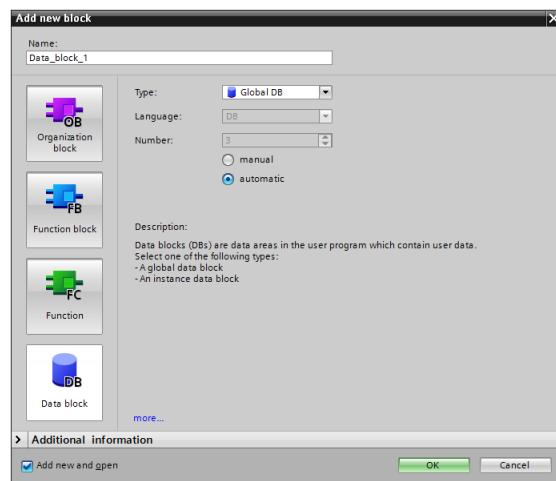
Variable data is located in data blocks that are available to the entire user program.

3.2 Program blocks

Figure 3-12: Global DB as central data memory
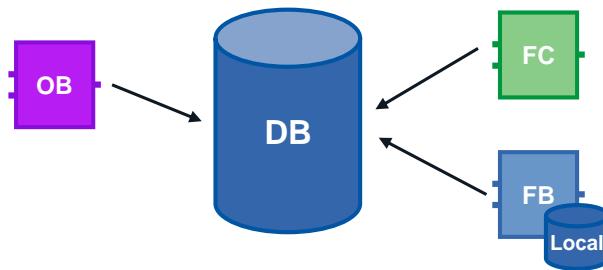
Advantages

- 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 0
- STRUCT data type and PLC data types).

Recommendation

- Use the global DBs when data is used in different program parts or blocks.

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? https://support.industry.siemens.com/cs/ww/en/view/68015630

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.

3.2 Program blocks

Figure 3-13: Downloading without reinitialization
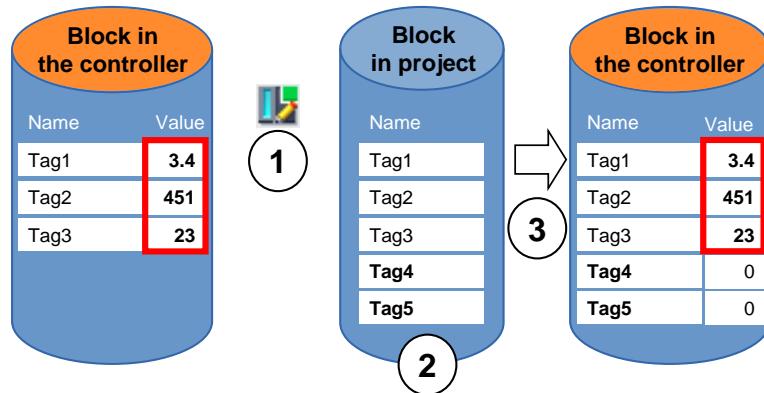
Execute the following steps whilst the controller is in RUN mode.

1. Enable "Downloading without reinitialization"
2. Insert new defined tags in existing block
3. Load block into controller

Advantages

- Reloading of new defined tags without interrupting the running process. The controller stays in "RUN" mode.

Properties

- Downloading without reinitialization is only possible for optimized blocks.
- New defined tags will be initialized. The remaining tags keep their current values.
- 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.
- It is assumed that a memory reserve has been defined for block.
- By default the memory reserve is set to 100 byte.
- The memory reserve is defined individually for every block.
- The blocks can be variably expanded.

Recommendation

- Define a memory reserve for blocks that are to be expanded during commissioning (e.g. test blocks). The commissioning process is not interrupted by download of new defined tags. The current values of already existing tags are kept.
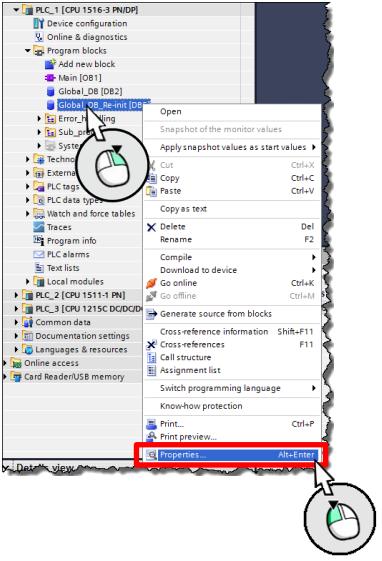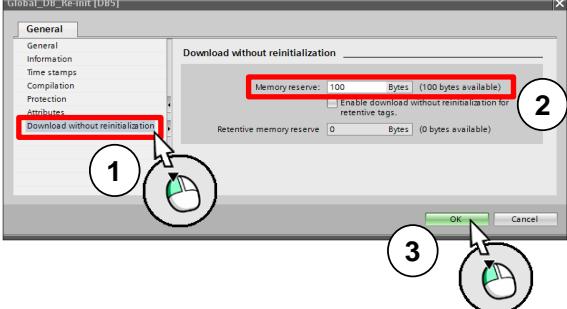
Example: Setting memory reserve in block

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

Table 3-3: Setting memory reserve

| Step	Instruction
1.	Right-click any optimized block in the project navigator and select "Properties".
2.	 1. Click "Download without reinitialization". 2. Enter the desired memory reserve for "Memory reserve". 3. Confirm with "OK".
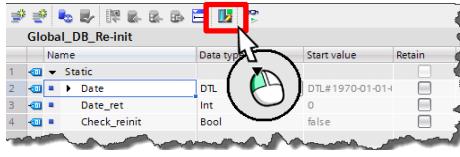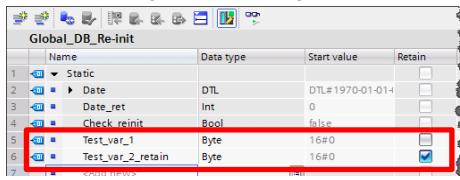
Note	You can also set a default value for the size of the memory reserve for new blocks in the TIA portal. In the menu bar, navigate to "Options – Settings" and then to "PLC programming – General – Download without reinitialization".

Example: Downloading without reinitialization

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

Table 3-4   Downloading without reinitialization

Step	Instruction
1.	Requirement: a memory reserve has to be set (see above)
2.	Open, e.g. an optimized global DB.
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: • Actual values of the block remain

Note	Further information can be found in the online help of the TIA Portal under "Loading block extensions without reinitialization".
	You will find further information in the following entry:
	What options does the S7-1500 provide for downloading data in RUN? https://support.industry.siemens.com/cs/ww/en/view/68015630

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
- 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 within 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
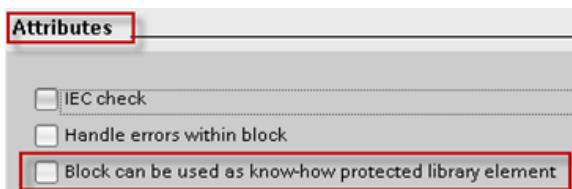
3.2.9 Auto numbering of blocks

For internal processing, required block numbers are automatically assigned by the system (setting in the block properties).

Figure 3-15: Auto numbering of blocks
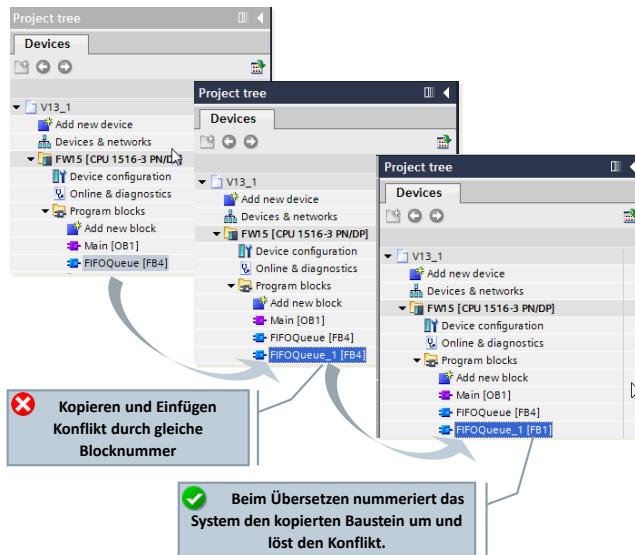
Advantages

- Conflicting block numbers, e.g. as a result of copying, automatically deletes the TIA Portal during compilation.

Recommendation

- Keep the current setting "automatic" unchanged.

Figure 3-16: Setting in the block
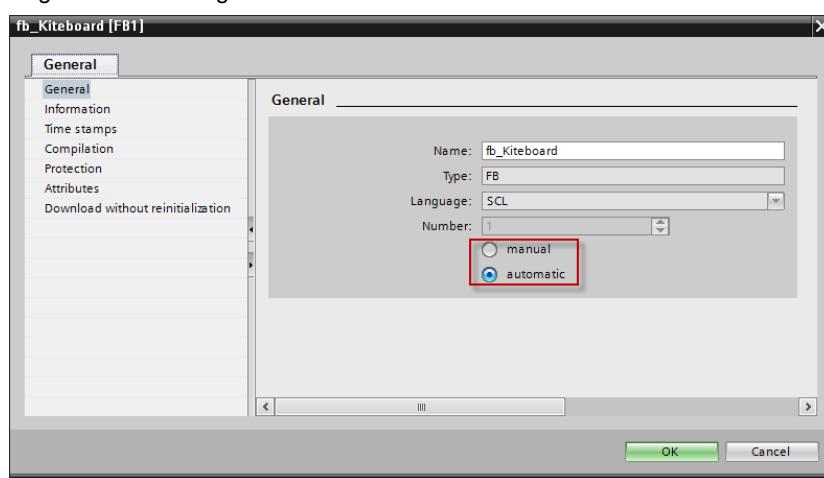
3.3 Block interface types

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. InOut parameters are used to transfer data to the called block as well as to return results. 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 allocated in the called block.

Figure 3-17: Copying of the value to the input parameter
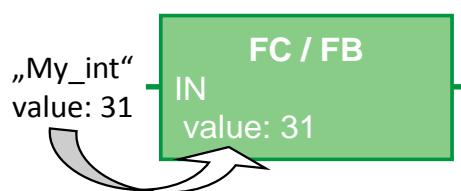
Properties

- 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 3-18: Referencing the value (pointer to data storage of the parameter)
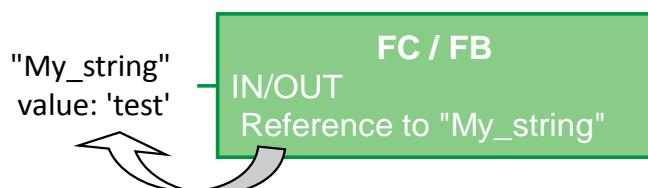
Properties

- 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.

3.3.3 Overview for transfer of parameters

The following table summarizes how S7-1200/1500 block parameters are transferred by means of elementary or structured data type.

Table 3-5: Overview for transfer of parameters

Block type / formal parameter		Elementary data type	Structured data type
FC	Input	copy	**reference**
	Output	copy	**reference**
	InOut	copy	**reference**
FB	Input	copy	copy
	Output	copy	copy
	InOut	copy	**reference**

Note	If, when calling the block, optimized data is transferred to a block with property "standard access", this data is principally transferred as a copy. If the block contains many structured parameters, this may quickly cause the temporary memory area (local data stack) of the block to overflow.
	This can be prevented by creating the same access type for both blocks (see chapter 2.6.5 Parameter transfer between blocks with optimized access and standard access).

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

Recommendation

- If possible, only use the local tags. This is how the blocks can be used universally and modularly.
- Use the data exchange via the block interfaces (In, Out, InOut), to ensure the reusability of blocks.
- Only use the instance data blocks as local memory for the respective function block. Other blocks shall not be written into instance data blocks.

3.4 Storage concept

Figure 3-19: Avoiding accesses to instance data blocks
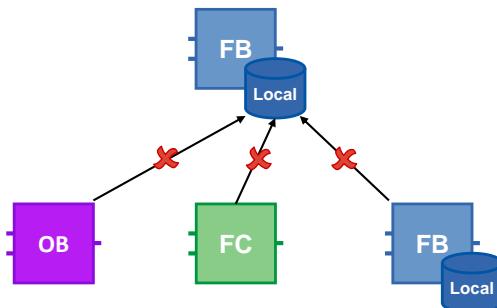
If only the block interfaces are used for the data exchange it can be ensured that all blocks can be used independent from each other.

Figure 3-20: Block interfaces for data exchange
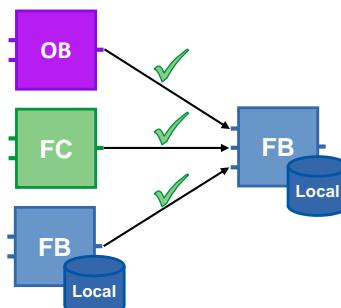

3.4.2 Global memory

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, timers, 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 dividing 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".

3.4 Storage concept

Figure 3-21: IEC Timers
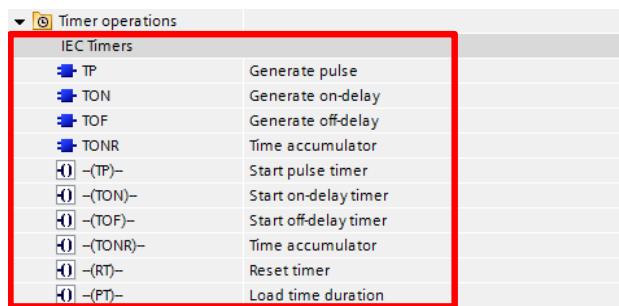
3.4.3 Local memory

- Static tags
- Temporary tags

Recommendation

- Use the static tags for values that are required for the next cycle.
- Use the temporary tags as cache memory in current cycle. 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 "default value" (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 3-22: Different memory accesses
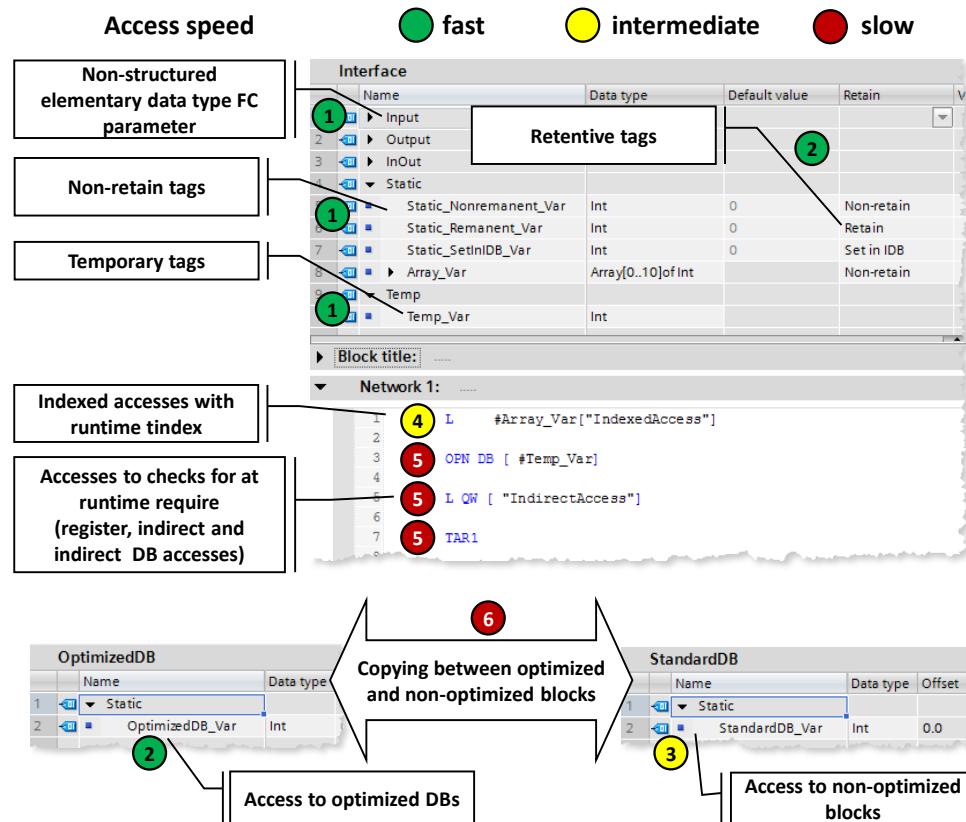
Fastest accesses in the S7-1200/1500 in descending order

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 3-6: Retentive memory for S7-1200/1500

Controller	Usable retentive memory for bit memory, times, counters, DBs and technology objects
CPU 1211C,1212C, 1214C, 1215C, 1217C	10 Kbytes
CPU 1511-1 PN	88 Kbytes
CPU 1513-1 PN	88 Kbytes
CPU 1515-2 PN, 1516-3 PN/DP	472 Kbytes
CPU 1518-4 PN/DP	768 Kbytes

Table 3-7: 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 3-23: Program editor (Functions block interfaces)
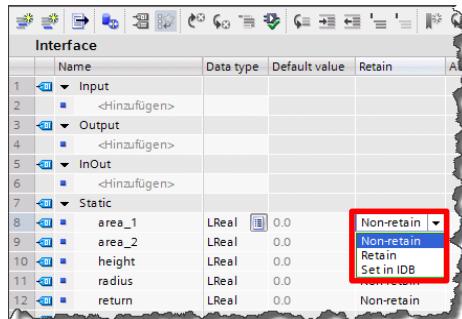
Figure 3-24: Program editor (Data block)
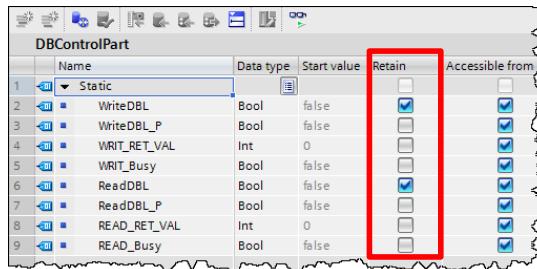
Example: Retentive of PLC tags

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

Figure 3-25: Setting of the retentive tags in the table of PLC tags

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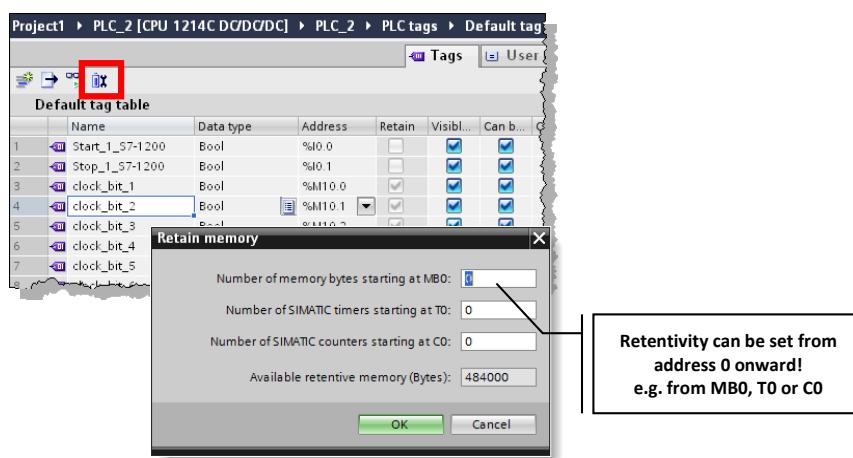
Example: Retentive counter

You can also declare instances of functions (timer, counter, etc.) retentive. As already described in chapter 3.2.5 Multi-instances

.

Figure 3-26: Retentive counter as multi-instance
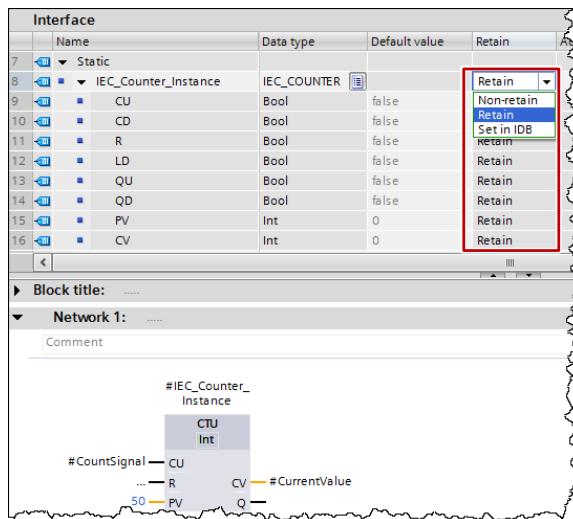
Note If the retentive memory on the PLC is not sufficient, it is possible to store data in the form of data blocks that are only located in the load memory of the PLC. The following entry is described by taking the example of an S7-1200. This programming also works for S7-1500.

You will find further information in the following entry:

In STEP 7 (TIA Portal), how do you configure data blocks with the "Only store in load memory" attribute for an S7-1200?
https://support.industry.siemens.com/cs/ww/en/view/53034113

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
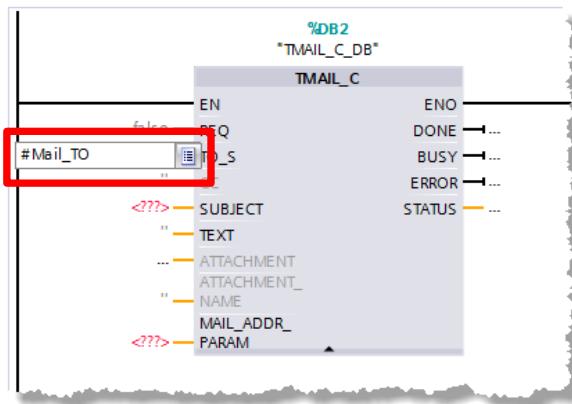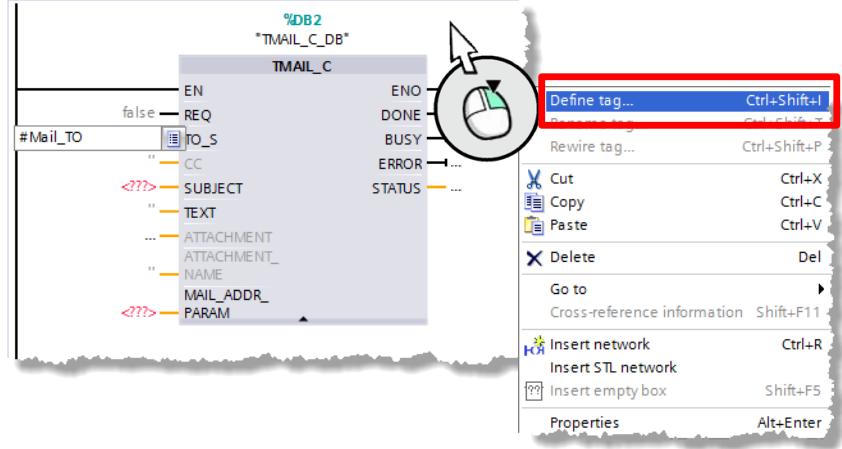
- Easier to read programs through symbolic tag names
- Automatic update of tag names at all usage locations in the user program
- Memory storage of the program data does not have to be manually managed (absolute addressing)
- Powerful data access
- No manual optimization for performance or program size reasons required
- IntelliSense for fast symbol input
- Fewer program errors due to type checking (validity of data types is checked for all accesses)

3.6 Symbolic addressing

Recommendation

- "Don't bother about the organization of the data storage"

- "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.
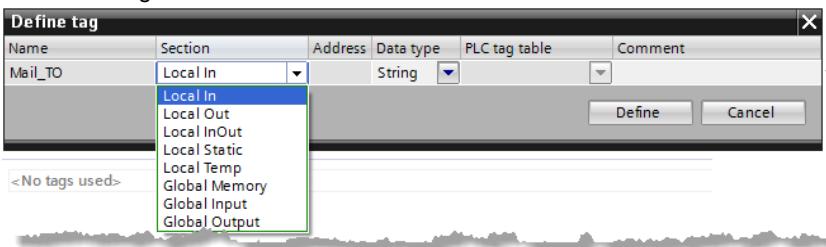
Example

Table 3-8: 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.
3.	Right-click next to the block and select "Define tag…" in the context menu.

Step	Instruction
4.	Define the tag. **Define tag** Name: Mail_TO Section: Local In Address Data type: String PLC tag table Comment Local In Local Out Local InOut Local Static Local Temp Global Memory Global Input Global Output Define Cancel \<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) obligatory for the S7-1500? https://support.industry.siemens.com/cs/ww/en/view/67598995

3.6.2 ARRAY data type and indirect field accesses

The ARRAY data type represents a data structure that consists of several elements of the same data type. The ARRAY data type is suitable, for example, for the storage of recipes, material tracking in a queue, cyclic process acquisition, protocols, etc.

Figure 3-27: ARRAY with 10 elements of the Integer (INT) data type
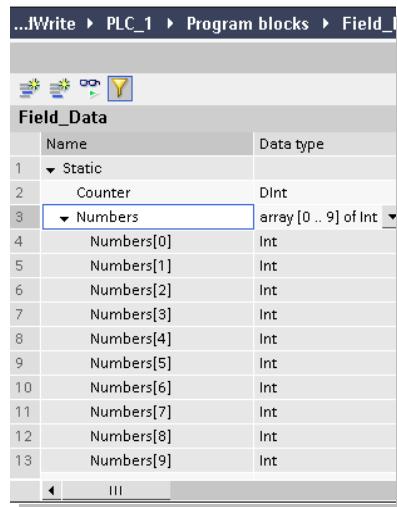
You can indirectly access individual elements in the ARRAY with a runtime tag (`array ["index"]`).

3.6 Symbolic addressing

Figure 3-28: Indirect field access
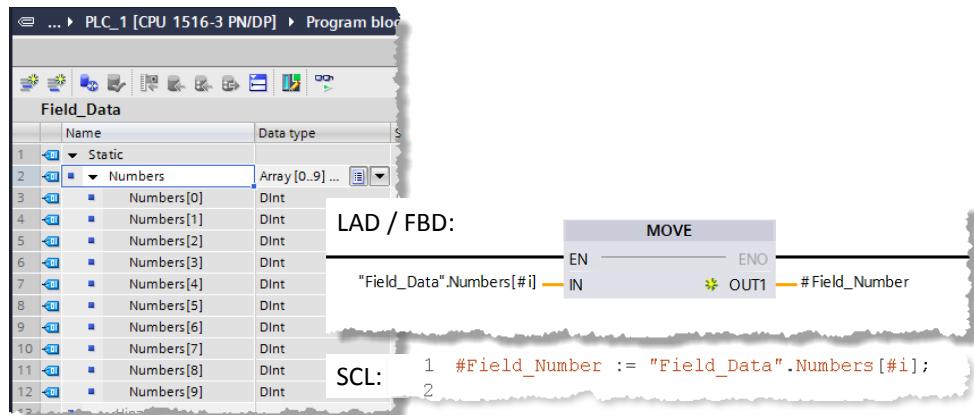
Advantages

- 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 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 DINT 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? https://support.industry.siemens.com/cs/ww/en/view/67598676
	How do you address securely and indirectly in STEP 7 (TIA Portal)? https://support.industry.siemens.com/cs/ww/en/view/97552147
	How, in STEP 7 (TIA Portal), can data be transferred for S7-1500 between two tags of data type "Array of Bool" and "Word"? https://support.industry.siemens.com/cs/ww/en/view/108999241

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 3-29: Structure with elements with different data types
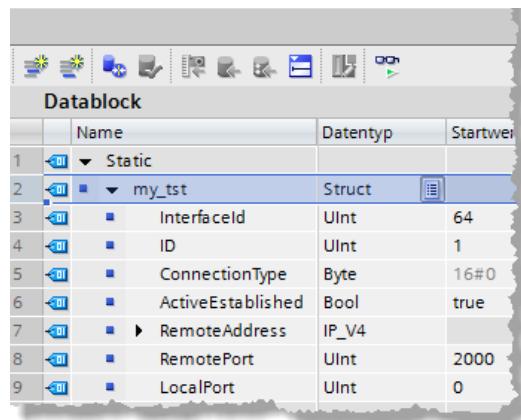
In comparison to structures, PLC data types are defined across the controller in the TIA Portal and can be centrally changed. All usage locations are automatically updated.

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

Figure 3-30: PLC data types

3.6 Symbolic addressing

Advantages

- A change in a PLC data type is automatically updated in all usage locations in the user program.
- Simple data exchange via block interfaces between several blocks

Properties

- PLC data types always end at WORD limits (see the figures below).
- Please consider this system property when ...
  - using different I/O areas (see chapter 3.6.4 Access to I/O areas with PLC data types).
  - using frames with PLC data types for communication.
  - parameter records with PLC data types for I/O.
  - non-optimized blocks absolute addressing.

Figure 3-31: PLC data types always end at WORD limits
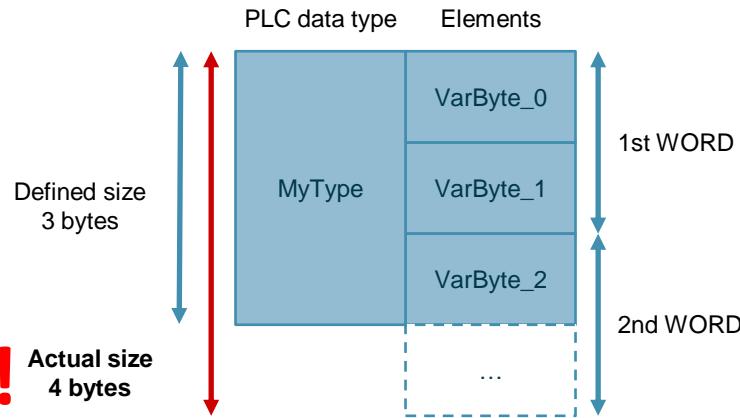
Figure 3-32: PLC data type on I/O area
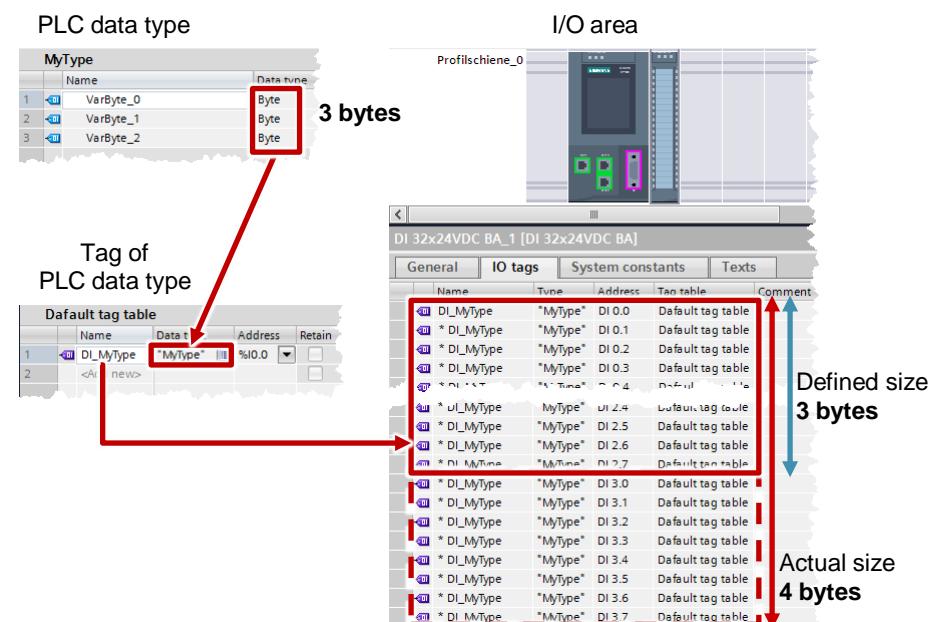
Recommendation

- 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.
- 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.

| 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)? https://support.industry.siemens.com/cs/ww/en/view/78678760
	How do you create a PLC data type for an S7-1500 controller? https://support.industry.siemens.com/cs/ww/en/view/67599090
	In STEP 7 (TIA Portal), how do you apply your own data types (UDT)? https://support.industry.siemens.com/cs/ww/en/view/67582844
	Why should whole structures instead of many single components be transferred for the S7-1500 when a block is called? https://support.industry.siemens.com/cs/ww/de/view/67585079

3.6.4 Access to I/O areas with PLC data types

With S7-1500 controllers, you can create PLC data types and use them for structured and symbolic access to inputs and outputs.

Figure 3-33: Access to I/O areas with PLC data types
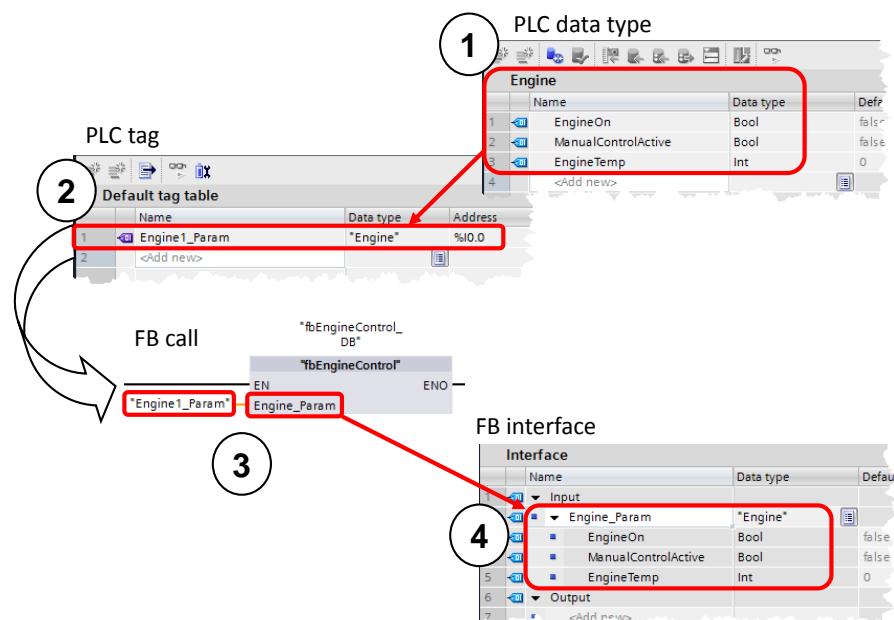
5. PLC data type with all required data

6. PLC tag of the type of the created PLC data type and start address of the I/O data area (%Ix.0 or %Qx.0, e.g., %I0.0, %Q12.0, …)

7. Transfer of the PLC tag as actual parameter to the function block

8. Input of the function block is of the type of the created PLC data type

Advantages

- High programming efficiency
- Easy multiple usability thanks to PLC data types

Recommendation

- Use PLC data types for access to I/O areas, for example, to symbolically receive and send drive telegrams.

Note	Individual elements of a PLC data type of a tag can also be directly accessed in the user program:

3.6 Symbolic addressing
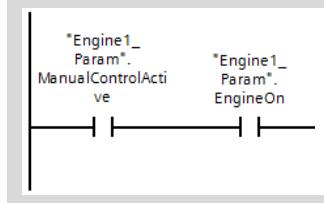
3.6.5 Slice access

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 3-34: Slice access
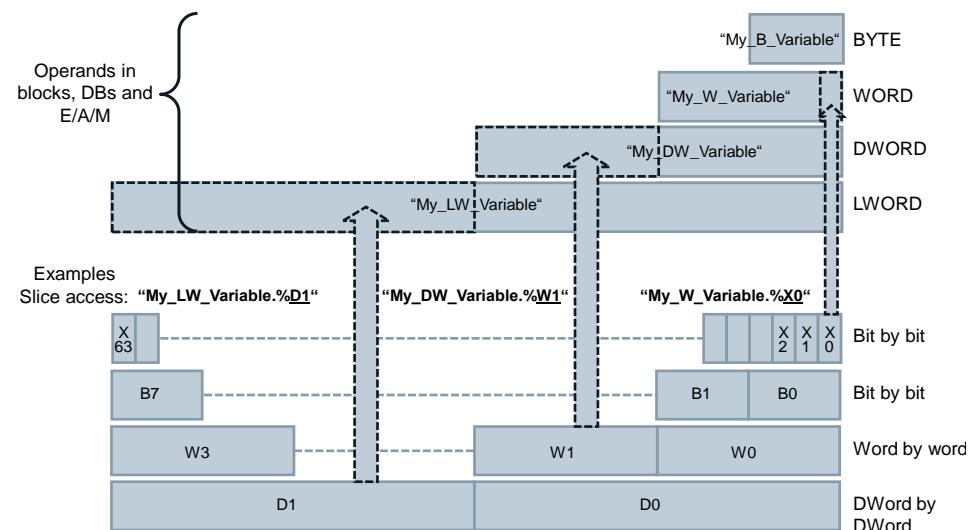
Advantages

- 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? https://support.industry.siemens.com/cs/ww/en/view/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.
- Configure the storage location of your global library so it can automatically be opened when starting the TIA Portal.
  Further information is available at:
  https://support.industry.siemens.com/cs/ww/en/view/100451450

Note	You will find further information in the following entries: Which elements of STEP 7 (TIA Portal) and WinCC (TIA Portal) can you store in a library as Type or as Master Copy? https://support.industry.siemens.com/cs/ww/en/view/109476862 How can you open a global library with write access rights in STEP 7 (TIA Portal)? https://support.industry.siemens.com/cs/ww/en/view/37364723

### 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 3-35: Libraries in the TIA Portal
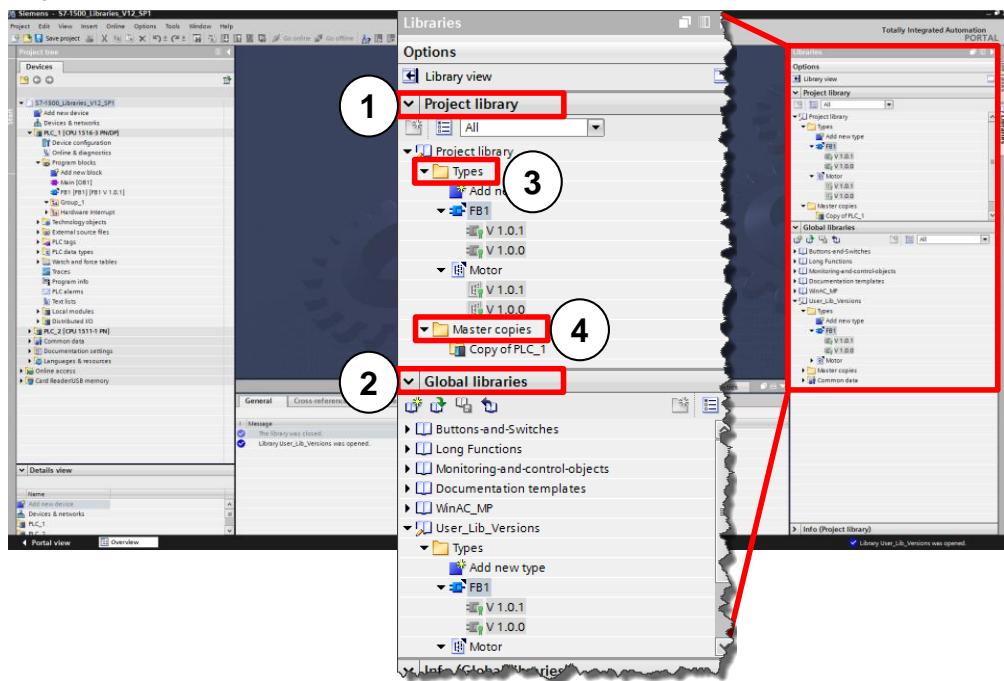
(1) "Project library"

- 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 3-36: Typifying with user libraries
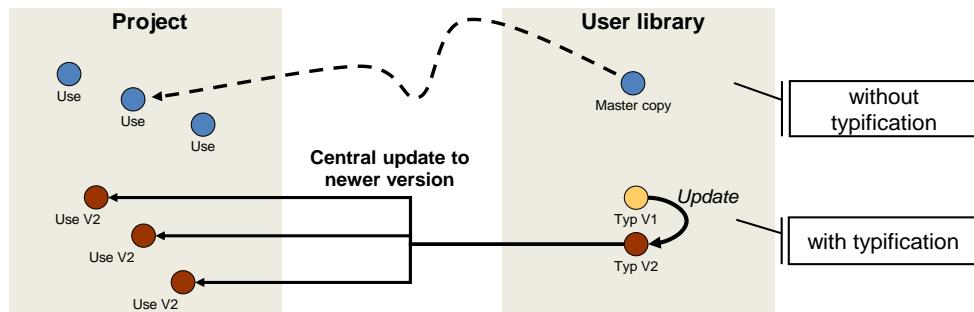
- Types are always marked in the project for better identification

3.7.3 Differences for typifiable objects for CPU and HMI

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

Table 3-9: 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: Creating a type
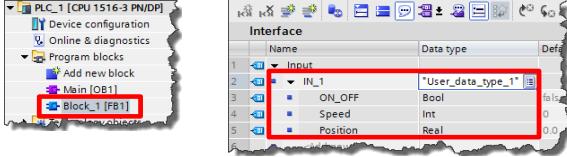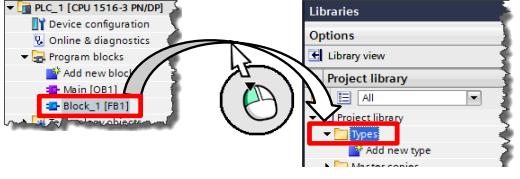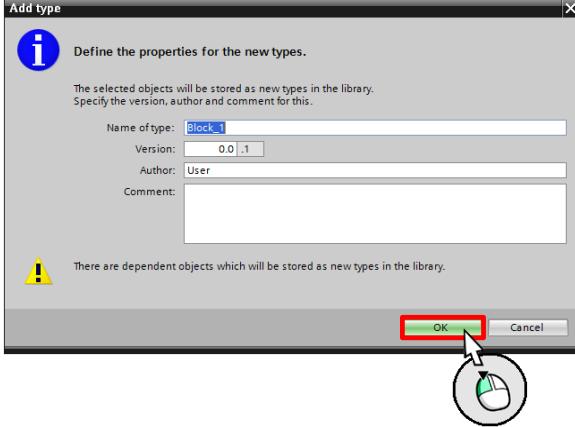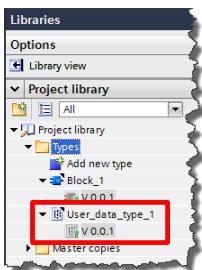
The following example shows you how the basic functions of the libraries are used with types.

Table 3-10: 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.
2.	Create a new function block with "Add new Block". This is the higher-level type.

3.7 Libraries
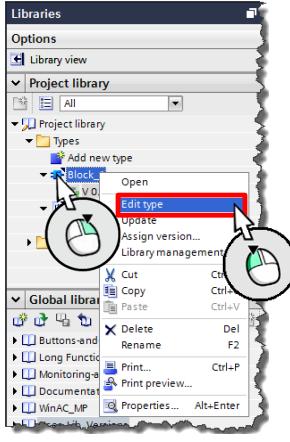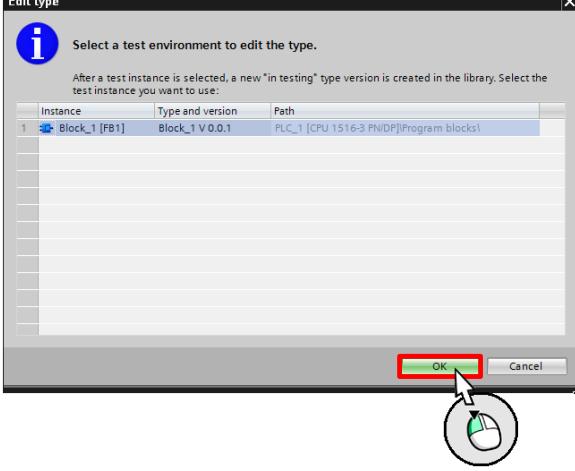
Step	Instruction
3.	Define an input tag of the data type you have created. The PLC data type is therefore subordinate to the function block.
4.	Drag the function block via drag & drop into the "Types" folder in the project library.
5.	Optionally assign: Type name, version, author and comment and confirm the dialog with "OK".
6.	The subordinate PLC data type is automatically also stored in the library.

3.7 Libraries

Example: Changing a type
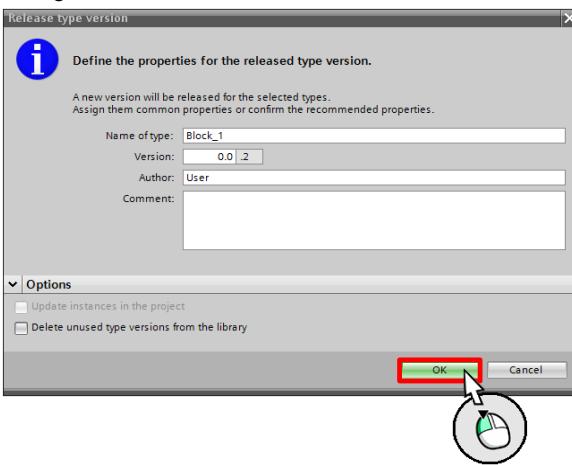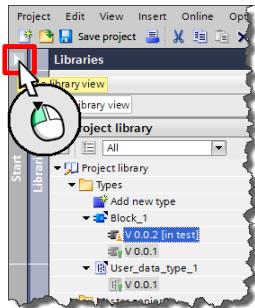
Table 3-11: Changing a type

Step	Instruction
1.	Right-click the block in the "Project library" and select "Edit type"
2.	Select which controller is to be used as test environment and confirm the dialog with "OK". If several controllers in the project use the selected block, a controller has to be selected as test environment.

3.7 Libraries

Step	Instruction
3.	The library view opens. A new version of the block has been created and is now marked with "in test".
4.	Add another input tag. 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.

Step	Instruction
6.	A dialog box opens. Here you can store a version-related comment. Confirm the dialog with "OK". 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". 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"

3.8 Increasing performance with 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 3-37: Process interrupt is calling OB
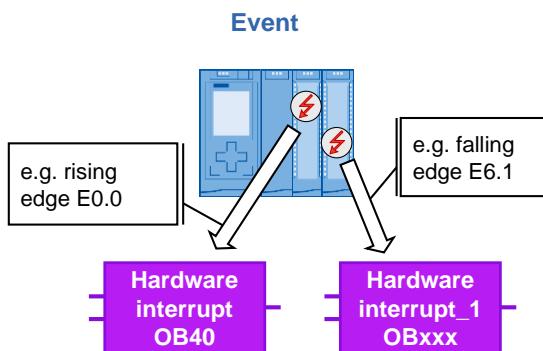
In the following figure you can see the configuration of a "hardware interrupt" in the hardware configuration of a digital input module.

Figure 3-38: Configuring hardware interrupt
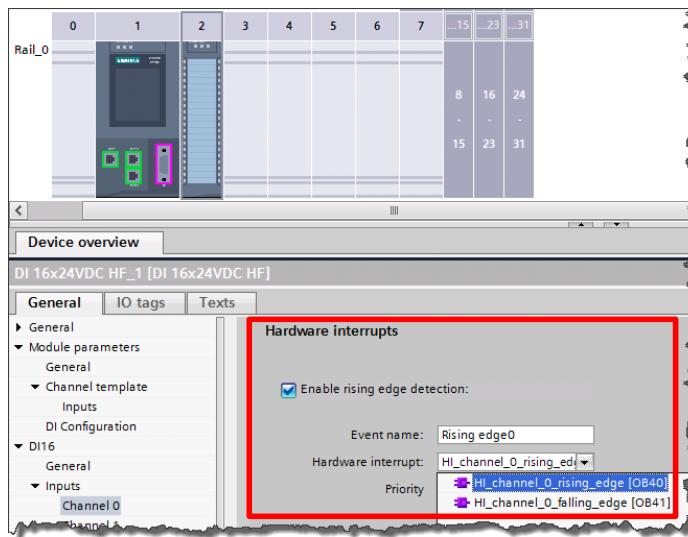
Advantages

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

3.8 Increasing performance with process interrupts

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 3-39: Setting input delay
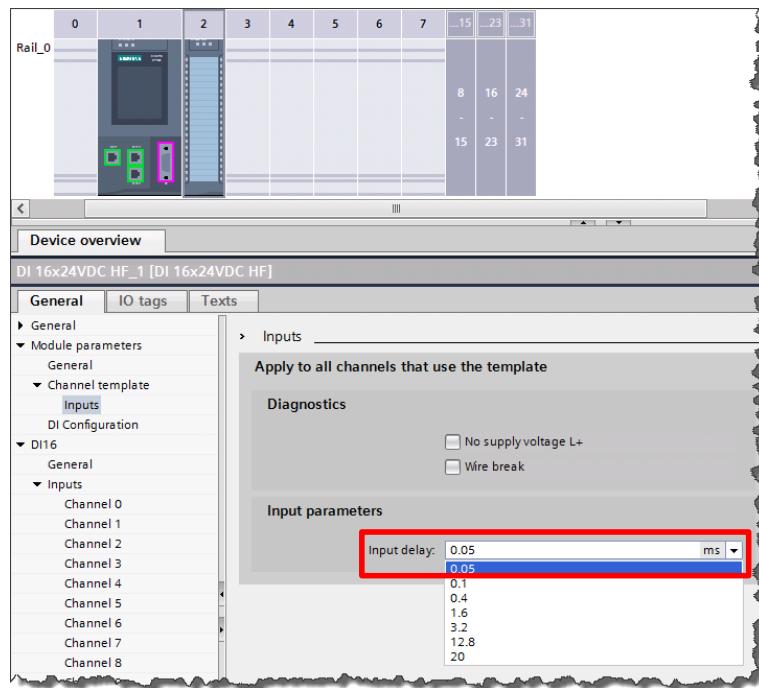
3.9 Additional performance recommendations

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.

Note	You will find further information in the following entry:
	How do you disable the ENO enable output of an instruction? https://support.industry.siemens.com/cs/ww/en/view/67797146
	How can you improve the performance in STEP 7 (TIA Portal) and in the S7-1200/S7-1500 CPUs? https://support.industry.siemens.com/cs/ww/en/view/37571372
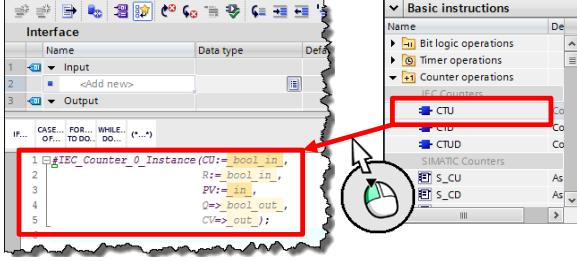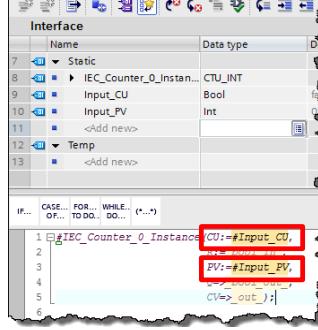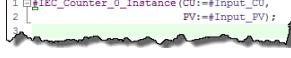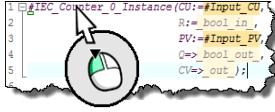
3.10 SCL programming language: Tips and tricks

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 3-12: 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.
2.	Now fill in the required parameter and finish the entry with the "Return" button.
3.	The editor automatically reduces the call template.
4.	If you want to edit the complete call later on again, proceed as follows. 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.
5.	Note: in the "Call Template" mode the writing is in italics.

3.10 SCL programming language: Tips and tricks
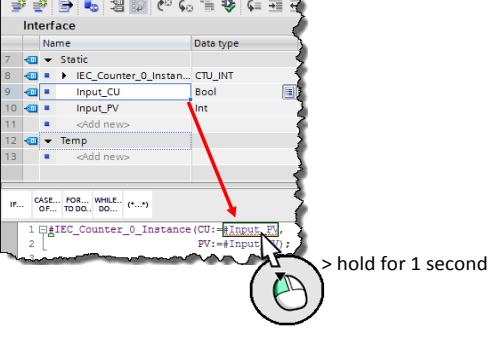
3.10.2 What instruction parameters are mandatory?

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 3-13: 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. Hold the tag for more than 1 second before releasing it. The complete tag is replaced.

3.10.4 Correct application of FOR, REPEAT and WHILE loops

When using loops, there are different versions and application cases. The following examples show the differences.

Properties: FOR loop

The FOR loop runs over a **defined number of cycles**. A start value is assigned to the runtime tag at the beginning. Then, in each loop cycle, the runtime tag is counted up with the specified step width up to the end value.

For performance reasons, the start as well as the end values are calculated once at the beginning. The runtime tag can therefore no longer be influenced in the loop code.

Syntax

```
FOR counter := start_count TO end_count DO
    // Statement section ;
END_FOR;
```

Using the EXIT command, the loop can be interrupted at any time.

Properties: WHILE loop

The WHILE loop can be terminated by means of a cancel condition. The **cancel condition is checked at the beginning** of the loop code. That is, the loop is not executed if the condition is fulfilled immediately. In the loop code, each tag can be adjusted for the next cycle.

Syntax

```
WHILE condition DO
     // Statement section ;
END_WHILE;
```

Properties: REPEAT loop

The REPEAT loop can be terminated by means of a cancel condition. The **cancel condition is checked at the end** of the loop code. That is, the loop is **run at least once**. In the loop code, each tag can be adjusted for the next cycle.

Syntax

```
REPEAT
    // Statement section ;
UNTIL condition
END_REPEAT;
```

Recommendation

- Use FOR loops if the runtime tag is clearly defined.
- Use WHILE or REPEAT loops if a runtime tag needs to be adjusted during loop editing.

3.10 SCL programming language: Tips and tricks

3.10.5 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;
```

Note CASE instructions also work with CHAR, STRING data types, as well as with elements (see example in Chapter 2.8.5 VARIANT data type).

3.10.6 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;
```

3.10.7 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;
```
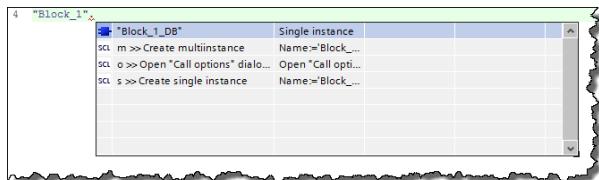
If you are defining "BY" as "-2", as in the example, the counter is lowered by 2 in every iteration. If you omit "BY", the default setting for "BY" is 1

3.10.8 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 3-14: Easy creation of instances

Step	Instruction
1.	Give the block name a: followed by a "." (dot). The automatic compilation now shows you the following.
2.	On the top you can see the already existing instances. In addition, you can directly create a new single instance or multi-instance. Use the shortcuts "s" or "m" to go directly to the respective entries in the automatic compilation window.

3.10.9 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

```
timeDifference := timeStamp1 – timeStamp2;
```

3.10 SCL programming language: Tips and tricks

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

Table 3-15: 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 • BYTE • WORD • DWORD	✓	✓	✓
	• LWORD	✗	✗	✓
Character type	• CHAR (8 bit)	✓	✓	✓
Numerical data types	• INT (16 bit) • DINT (32 bit) • REAL (32 bit)	✓	✓	✓
	• SINT (8 bit) • USINT (8 bit) • UINT (16 bit) • UDINT (32 bit) • LREAL (64 bit)	✗	✓	✓
	• LINT (64 bit) • ULINT (64 bit)	✗	✗	✓
Time types	• TIME • DATE • TIME_OF_DAY	✓	✓	✓
	• S5TIME	✓	✗	✓
	• LTIME • L_TIME_OF_DAY	✗	✗	✓

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

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	✗	✓	✓
	• LDT (L_DATE_AND_TIME)	✗	✗	✓
Character type	• STRING	✓	✓	✓
Field	• ARRAY	✓	✓	✓
Structure	• STRUCT	✓	✓	✓

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

	Description	S7 - 300/400	S7-1200	S7-1500
Pointer	• POINTER • ANY	✓	✗	✓[1]
	• VARIANT	✗	✓	✓
Blocks	• TIMER • COUNTER	✓	✓[2]	✓
	• BLOCK_FB • BLOCK_FC	✓	✗	✓
	• BLOCK_DB • 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 (also system and clock memory bits) 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.

	FB	
	Q [Bool] — TRUE	Period: 2 seconds
0.5 — Frequency [Real]		
	Countdown [Time] — T#0S_703MS	
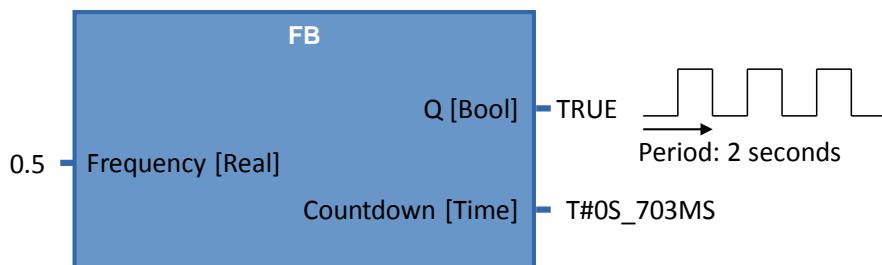
| Note	The complete programming example is available for free download in the following entry:
	https://support.industry.siemens.com/cs/ww/en/view/87507915

5 STEP 7 Safety in the TIA Portal

5.1 Introduction

Fail-safe S7-1200F and S7-1500F CPUs are supported from TIA Portal V13 SP1 onward. In these controllers, standard as well as the fail-safe programming in a device is possible. For programming the fail-safe user programs, the SIMATIC STEP 7 Safety (TIA Portal) option package is used.

Figure 5-1: Standard and safety program
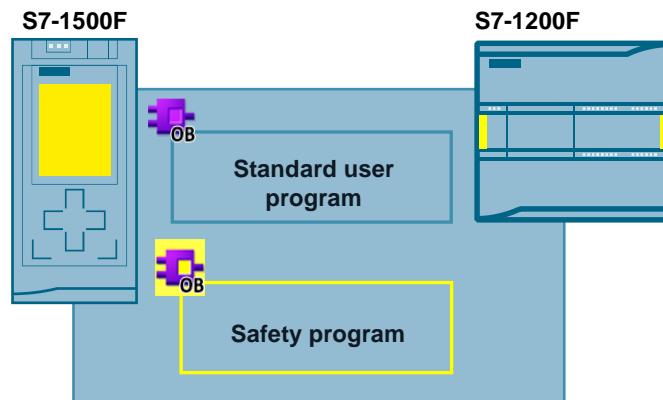
Advantages

- Uniform programming in standard and fail-safe program with an engineering tool: TIA Portal
- Familiar programming in LAD and FBD
- Uniform diagnostics and online functions

Note	Fail-safe does not mean that the program contains no errors. The programmer is responsible for the correct programming logic.
	Fail-safe means that the correct processing of the fail-safe user program in the controller is ensured.

Note	Further information on the topic of safety, as in the safety requirements or the principles of safety programs, for example, are available at:
	TIA Portal - An Overview of the Most Important Documents and Links - Safety https://support.industry.siemens.com/cs/ww/en/view/90939626
	Applications & Tools – Safety Integrated https://support.industry.siemens.com/cs/ww/en/ps/14675/ae
	STEP 7 Safety (TIA Portal) - Manuals https://support.industry.siemens.com/cs/ww/en/ps/14675/man

5.2 Terms

This document consistently uses the terms with the following meaning.

Table 5-1: Safety terms

Term	Description
Standard user program	The standard user program is the program part, which is not connected with F programming.
Safety program (F program, fail-safe user program)	The fail-safe user program is the program part which is processed fail-safe independently of the controller. All fail-safe blocks and instructions are shaded yellow at the software user interface (e. g. in the project navigation) in order to distinguish blocks and instructions of the standard user program. The fail-safe parameters of F-CPUs and F-I/O are shaded yellow in the hardware configuration.

5.3 Components of the safety program

Das safety program always consists of user-generated or system-generated F blocks and the "Safety administration" editor.

Table 5-2: Components of the safety program

Description	Screen

1. "Safety administration" editor
  - Status of the safety program
  - F collective signature
  - Status of the safety operation
  - Creating/organizing F run-time groups
  - Information on the F blocks
  - Information on F-conform PLC data types
  - Defining/changing the access protection
 	
2. User-created F blocks
 	
3. System-generated F-runtime blocks
  - Blocks contain status information on the F run-time group.
 	
4. System-generated F-I/O data blocks
  - Blocks contain tags for evaluating the F modules.
 	
5. "Compiler blocks"
System-generated verification blocks
  - These run in the background of the controller and provide for fail-safe processing of the safety program.
  - These blocks cannot be processed by the user.	

5.4 F runtime group

A safety program is always processed in an F-runtime group with defined cycle. An F run-time group consists of a "Fail-safe organization block" which calls a "Main safety block". All user-generated safety functions are called from the "Main safety block".

Figure 5-2: F-runtime group in the "Safety administration" editor
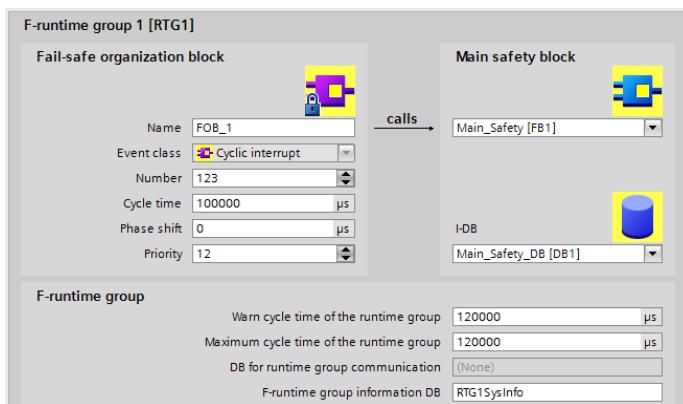
Advantages

- Runtime groups can simply be created and configured in the "Safety Administrator".
- F-blocks in the run-time group are automatically created.

Properties

- A maximum of two F run-time groups can be created.

5.5 F signature

Each F component (station, I/O, blocks) has a unique F signature. Using the F signature it can be quickly detected whether an F device configuration, F blocks or a complete station still corresponds to the original configuration or programming.

Advantages

- Simple and quick comparison of F blocks and F device configurations

Properties

- F parameter signature (without address of F-I/O)…
  - only changed by adjusting the parameters.
  - remains unchanged when changing the PROFIsafe address. However, the F collective signature of the station changes.
- F block signature is only changed when the logic in the F block changes.

5.5 F signature

- F block signature remains unchanged by changing the
  - block number,
  - block interface,
  - block version.

Example

Figure 5-3: Examples of F signatures

1. F collective signature of the station in the "Safety administration" editor
2. F block signatures in the "Safety Administration" editor (can also be read out from the properties of the block)
3. F parameter signature in the "Device view" at "Devices & Networks"

Note	For S7-1500F controllers it is possible to read the F overall signature directly on the installed display or in the integrated web server.

5.6 Assigning the PROFIsafe address at the F-I/O

Each F-I/O device has a PROFIsafe address for identification and communication with F controllers. When assigning the PROFIsafe address, two different configurations are possible.

Table 5-3: Setting the F address

ET 200M / ET 200S (PROFIsafe address type 1)	ET 200MP / ET 200SP (PROFIsafe address type 2)
Assigning the PROFIsafe address directly at the modules via DIL switch In the device configuration of the TIA Portal and in the DIL switch position at the periphery, the PROFIsafe address must be the same.	Assigning the PROFIsafe address exclusively via TIA Portal The configured PROFIsafe address is loaded onto the intelligent coding module of the module.

Advantages

- Replacing an F module is possible without reassigning the PROFIsafe address at ET 200MP and ET 200SP. The intelligent coding module remains in the BaseUnit during module exchange.

- Simple configuration since TIA Portal indicates a faulty assignment of the PROFIsafe address warnings.

- The PROFIsafe address of all F modules can be assigned at the same time within an ET 200SP.

Note	Further information on assigning the PROFIsafe address for the F-I/O is available at: SIMATIC Industrial Software SIMATIC Safety – Configuring and Programming https://support.industry.siemens.com/cs/ww/en/view/54110126

5.7 Evaluation of F-I/O

All of the current states of the respective F-I/O are saved in the F-I/O blocks. In the safety program the states can be evaluated and processed. The following differences exist between S7-1200/1500F and S7-300F/400F.

Table 5-4: Tags in the F-I/O DB with S7-300F/400F and S7-1500F

Tag in F-I/O DB or value status in PAE	F-I/O with S7-300/400F	F-I/O with S7-1200/1500F
ACK_NEC	✓	✓
QBAD	✓	✓
PASS_OUT	✓	✓
QBAD_I_xx *	✓	✗
QBAD_O_xx *	✓	✗
Value status	✗	✓

* QBAD_I_xx and QBAD_O_xx show you the validity of the channel value and correspond to the **inverted** value status at S7-1200/1500F (further information is available in the following chapter).

5.8 Value status (S7-1200F / S7-1500F)

In addition to the diagnostic messages and the status and error display, the F module provides information on the validity of each input and output signal - the value status. The value status is stored in the same way as the input signal in the process image:

The value status informs about the validity of the respective channel value.

- 1: a valid process value is output for the channel.

- 0: a substitute value is output for the channel.

Table 5-5: Differences between Q_BAD (S7-300F/400F) and value status (S7-1200/1500F)

Scenario	QBAD (S7-300F/400F)	Value status (S7-1200/1500F)
Valid values at the F-I/O (no error)	FALSE	TRUE
Channel error occurs	TRUE	FALSE
Channel error going (ACK_REQ)	TRUE	FALSE
Acknowledgement of the failure (ACK_REI)	FALSE	TRUE

Properties

- The value status is entered into the process image of the inputs and outputs.

- Channel value and value status of an F-I/O must only be accessed from the same F run-time group.

Recommendation

- For improved readability you assign the ending "_VS", e.g. "Tag_In_1_VS" as the symbolic name for the value status.

Example

Position of the value status bits in the process image using the example of an F-DI 8x24VDC HF module.

Table 5-6: Value status bits in the process image using the example of an F-DI 8x24VDC HF

Byte in the F-CPU	Assigned bits in the F-CPU							
	7	6	5	4	3	2	1	0
x + 0	DI_7	DI_6	DI_5	DI_4	DI_3	DI_2	DI_1	DI_0
x +1	Value status for DI_7	Value status for DI_6	Value status for DI_5	Value status for DI_4	Value status for DI_3	Value status for DI_2	Value status for DI_1	Value status for DI_0

x = module start address

5.9 Data types

Note	More information about the value status of all ET 200SP modules is available at:
	Failsafe CPUs - Manuals https://support.industry.siemens.com/cs/ww/en/ps/13719/man
	Failsafe I/O modules - Manuals https://support.industry.siemens.com/cs/ww/en/ps/14059/man

5.9 Data types

There is an unrestricted scope of data types for the safety programs of the S7-1200/1500F.

Table: 5-7: Integer data types

Type	Size	Value range
BOOL	1 bit	0 .. 1
INT	16 Bit	-32.768 .. 32.767
WORD	16 Bit	-32.768 .. 65.535
DINT	32 bit	-2.14 .. 2.14 million
TIME	32 bit	T#-24d20h31m23s648ms to T#+24d20h31m23s647ms

5.10 F-conform PLC data type

For safety programs it is also possible to structure data optimal with PLC data types.

Advantages

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

Properties

- F-PLC data types are declared and used in the same way as PLC data types.
- F-PLC data types can use all data types which are allowed in the safety program.
- Nesting of F-PLC data types within other F-PLC data types is not supported.
- In F-PLC data types, standard user programs can be used in the safety program as well as in the standard user program.

5.10 F-conform PLC data type

Recommendation

- For accessing I/O areas you use F-PLC data types (as in Chapter)
- The following rules must be observed here:
  - The structure of the tags of the F-conform PLC data type must match the channel structure of the F-I/O.
  - An F-conform PLC data type for an F-I/O with 8 channels is, for example:
    - 8 BOOL tags (channel value)
    - 16 BOOL tags (channel value + value status)
  - Access to F-I/O is only permitted for activated channels. When configuring a 1oo2 (2v2) evaluation, the higher channel is always deactivated.

Example

Figure 5-4: Access to I/O areas with F-PLC data types
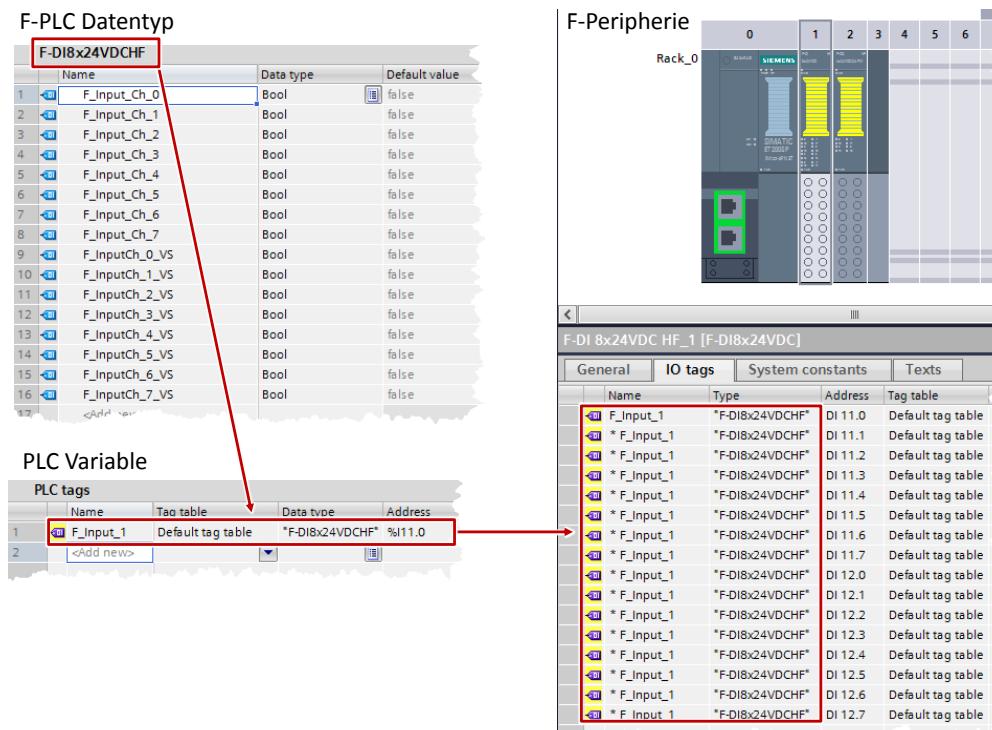
5.11   TRUE / FALSE

If you require "TRUE" and "FALSE" signals in the safety programs, there are two possible cases:

- as actual parameter at blocks
- as assignment to operations

Actual parameter at blocks

For S7-1200/1500F controllers you can use the Boolean constants "FALSE" for 0 and "TRUE" for 1 as actual parameter for supplying formal parameters during block calls in the safety program. Only the keyword "FALSE" or "TRUE" is written to the formal parameter.

Figure 5-5: "TRUE" resp. "FALSE" signals as actual parameter
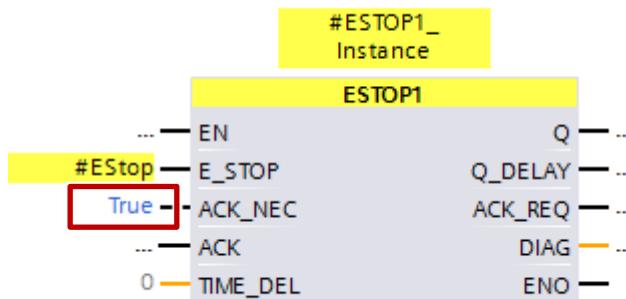
**Assignments to operations**

If you require "TRUE" or "FALSE" signals at operations you can create them as in the figure below.

- Use programming language FBD.
- Create two static or temp tags of datatype bool: "alwaysTrue", "alwaysFalse".
- Create the network 1, like the following figure.
- The tags can be used as "True" and "False"-tag in the entire block.

Figure 5-6: "TRUE" and "FALSE" signals
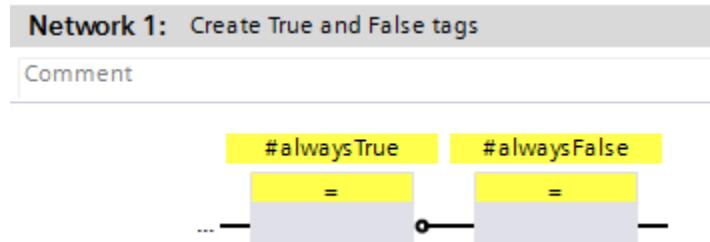
5.12 Optimizing compilation and program runtime

This chapter shows you the various options of reducing the compilation and program runtime.

Depending on the application, it is not always possible to apply all of the suggestions. However, they explain why certain programming methods cause shorter compilation and program runtimes than a standard program.

Apart from blocks programmed by the user, there are also may system-internal blocks which are created automatically. These blocks ensure that the safety program is safe. The compilation and program runtime is increased by the system-internal blocks.

Example

Figure 5-7: User and system-generated F-blocks

5.12 Optimizing compilation and program runtime
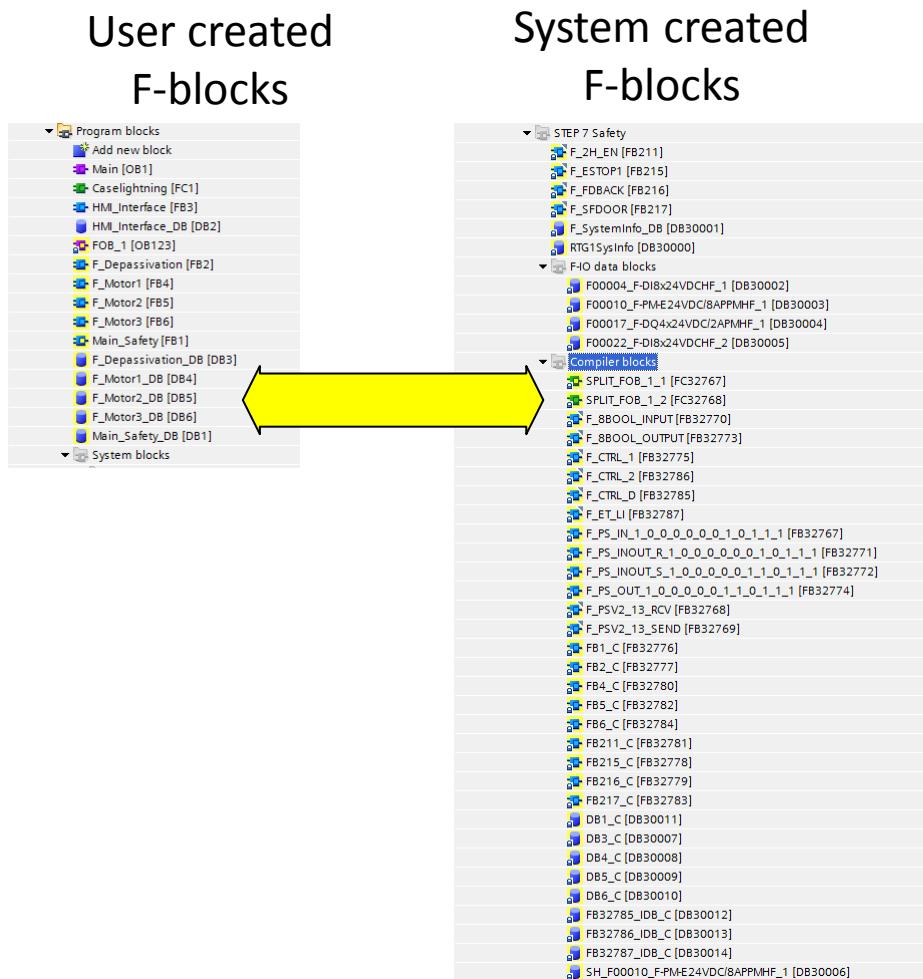
5.12.1 Avoiding time-processing blocks: TP, TON, TOF

Each time-processing block (TP, TON, TOF) requires additional blocks and global data corrections in security code.

Recommendation

Use these blocks as little as possible.

5.12.2 Avoiding deep call hierarchies

Deep call hierarchies enlarge the code of the system-generated F blocks since a larger scope of security functions and checks becomes necessary. If the nesting depth of 8 is exceeded, a warning is issued by the TIA Portal during compilation.

Recommendation

Structure your program so unnecessarily deep call hierarchies are avoided.

5.12.3 Separation of standard and safety program

In complex projects it is often necessary to exchange information between standard and safety program. If the exchange is realized via tags (e.g. flags), changes in the standard program may make a compilation of the safety program necessary. To download the changes, the CPU must then be set to STOP.

Recommendation

Use standard DBs (see chapter 5.13 Data exchange between standard and F-program). Changes in the standard program then do not affect the safety program. The controller does not need to be in STOP mode for loading the standard program.

5.12.4 Use of multi-instances

If an instance DB is called at several locations in the safety program, this DB must be processed more than once during a cycle. This processing requires a higher scope of system-internal F blocks.

Recommendation

If possible, always use multi-instances. This may reduce the scope of the system-internal F blocks.

5.12.5 Avoiding JMP/label structures

If a block call is jumped via JMP/LABEL, it is additionally secured in the system-internal F blocks. In this case, a correction code to the jumped block call must be run. This reduces performance and adds time during compilation.

Recommendation

> Avoid JMP/Label structures as far as possible to reduce the system-internal F blocks.

5.13 Data exchange between standard and F-program

> In some cases it is necessary to exchange data between the safety program and the standard user program. The following recommendations should urgently be noted in order to guarantee data consistency between standard and the safety program.

Recommendations

- No data exchange via flags (see chapter <u>4.2 No bit memory but global data blocks</u>

- Concentrate the access between safety program and the standard user program on two standard DBs.

Figure 5-8: Data exchange between standard safety program
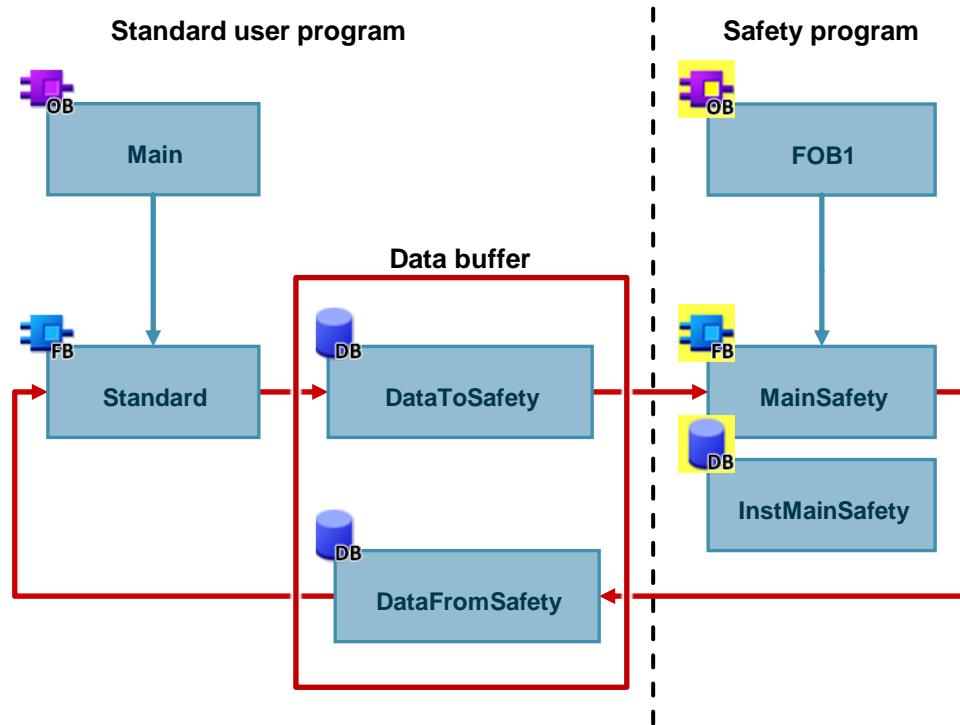
5.14 Testing the safety program

In addition to the always controllable data of a standard user program you can change the following data of a safety program in the deactivated safety mode.

- Process image of F−I/O
- F−DBs (except DB for F−run-time group communication), instance−DBs of F-FBs
- F-I/O DBs

Properties

- Controlling F−I/O is only possible in F−CPU RUN mode.
- From a watch table you can control a maximum of 5 inputs/outputs in a safety program.
- You can use several watch tables.
- The trigger point needs to be set to "permanent" or "once" for "cycle start" or "cycle end".
- Forcing is not possible for the F-I/O.
- Setting stop points in the standard user program will lead to errors in the safety program:
  - Expiring of F cycle time monitoring
  - Error during communication with the F-I/O
  - Error at fail-safe CPU-CPU communication
  - Internal CPU error
- If you still wish to use stop points for testing, you need to deactivate the safety mode beforehand. This leads to the following errors:
  - Error during communication with the F-I/O
  - Error at fail-safe CPU-CPU communication

5.15 STOP mode in the event of F errors

In the following cases, the STOP mode is triggered for F-CPUs:

- In the "System blocks" folder you must not add, change or delete any blocks.

- If the result of an instruction lies outside of the area permitted for the data type (overflow). The cause of the diagnostic event is entered in the diagnostics buffer of the F CPU.

- There must not be any access to instance DBs of F-FBs which are not called in the safety program.

- If the "Maximal cycle time of the F run-time group" is exceeded, the F-CPU goes to STOP. Select the maximal permitted time for "Maximum cycle time der F run-time group" which can elapse between two calls of this F run-time group (maximal 20,000 ms).

- If the F run-time group, from whose DB for F-run-time groups tags shall be read, is processed (main safety block of the F run-time group), the F-CPU goes to STOP.

- Editing the start values in instance DBs of F-FBs is not permitted online and offline and can lead to STOP of the F-CPU.

- The main safety block must not contain any parameters since they cannot be supplied.

- Outputs of F-FCs must always be initialized.

5.16 Migration of safety programs

Information on migrating safety programs is available at:
https://support.industry.siemens.com/cs/ww/en/view/109475826

5.17 General recommendations for safety

Generally, the following recommendations apply for handling STEP 7 Safety and F modules.

- Whenever possible, always use F controllers. A later expansion of safety functions can be realized very simply.

- Always use one password for the safety program to prevent unauthorized changes. The password is set in the "Safety administration" editor.

6 The Most Important Recommendations

- Use optimized blocks
  - Chapter 2.6 Optimized blocks
- Use data type VARIANT instead of ANY
  - Chapter 2.8.5 VARIANT data type
- 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 3.6.4 Access to I/O areas with 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

7 Related Literature

Table 7-1

	Topic	Title
\1\	Siemens Industry Online Support	https://support.industry.siemens.com/cs/start?lc=en-DE
\2\	Download page of the entry	https://support.industry.siemens.com/cs/ww/en/view/81318674
\3\	Programmierstyleguide for S7-1200 und S7-1500	https://support.industry.siemens.com/cs/ww/en/view/81318674
\4\	TIA Portal - An Overview of the Most Important Documents and Links	https://support.industry.siemens.com/cs/ww/en/view/65601780
\5\	STEP 7 (TIA Portal) manuals	https://support.industry.siemens.com/cs/ww/en/ps/14673/man
\6\	S7-1200 Manuals	https://support.industry.siemens.com/cs/ww/en/ps/13683/man
\7\	S7-1500 (F) Manuals	https://support.industry.siemens.com/cs/ww/de/ps/13716/man
\8\	ET 200SP CPU Manuals	https://support.industry.siemens.com/cs/ww/en/ps/13888/man
\9\	S7-1200 Getting Started	https://support.industry.siemens.com/cs/ww/de/view/39644875
\10\	S7-1500 Getting Started	https://support.industry.siemens.com/cs/ww/de/view/78027451
\11\	SIMATIC S7-1200 / S7-1500 Comparison list for programming languages	http://support.automation.siemens.com/WW/view/en/86630375

8 History

Table 8-1

Version	Date	Modifications
V1.0	09/2013	First version
V1.1	10/2013	Corrections in the following chapters: 2.6.3 Best possible data storage in the processor on S7-1500 2.13 Internal reference ID for controller and HMI tags 3.2.2 Functions (FC) 3.2.3 Function blocks (FB) 3.4.3 Local memory
V1.2	03/2014	New chapter: 2.6.4 Conversion between optimized and non-optimized tags 2.6.6 Communication with optimized data 2.9.2 MOVE instructions 2.9.3 VARIANT instructions 3.6.4 Access to I/O areas with PLC data types Extension of following chapter: 2.2 Terms 2.3 Programming languages 2.6 Optimized blocks 2.10 Symbolic and comments 3.2 Program blocks 3.5 Retentivity 4.3 Programming of "clock bits" Several corrections in different chapter
V1.3	09/2014	New chapter: 2.8.4 Unicode data types 2.10.2 Comment lines in watch table 2.12 User constants 3.2.9 Auto numbering of blocks 5 STEP 7 Safety in the TIA Portal Extension of following chapter: 2.7 Block sizes 2.8 New data types for S7-1200/1500 2.9 Instructions 2.10 Symbolic and comments 3.6.3 STRUCT data type and PLC data types 3.7 Libraries Several corrections in different chapter

Version	Date	Modifications
1.4	11/2015	New chapter: 2.6.5 Parameter transfer between blocks with optimized access and standard access 3.3.3 Overview for transfer of parameters 3.10.4 Correct application of FOR, REPEAT and WHILE loops 5.12 Optimizing compilation and program runtime