LamaPLC: Simatic datatypes

TIA Data type limits

Decimal	Hex	TIA data type	Byte	Description
18,446,744,073,709,551,615	FFFF FFFF FFFF FFFF	LWORD, ULINT	8	The maximum unsigned 64 bit value ($2^{64} - 1$)
9,223,372,036,854,775,807	7FFF FFFF FFFF FFFF	LINT	8	The maximum signed 64 bit value ($2^{63} - 1$)
9,007,199,254,740,992	0020 0000 0000 0000	-	8	The largest consecutive integer in IEEE 754 double precision (2^{53})
4,294,967,295	FFFF FFFF	DWORD, UDINT	4	The maximum unsigned 32 bit value ($2^{32} - 1$)

Decimal	Hex	TIA data type	Byte	Description
2,147,483,647	7FFF FFFF	DINT	4	The maximum signed 32 bit value ($2^{31} - 1$)
16,777,216	0100 0000	-	4	The largest consecutive integer in IEEE 754 single precision (2^{24})
65,535	FFFF	WORD, UINT	2	The maximum unsigned 16 bit value ($2^{16} - 1$)
32,767	7FFF	INT	2	The maximum signed 16 bit value ($2^{15} - 1$)
255	FF	BYTE	1	The maximum unsigned 8 bit value ($2^{8} - 1$)
127	7F	SINT	1	The maximum signed 8 bit value ($2^{7} - 1$)
−128	80	SINT	2	Minimum signed 8 bit value
−32,768	8000	INT	2	Minimum signed 16 bit value
−2,147,483,648	8000 0000	DINT	4	Minimum signed 32 bit value
−9,223,372,036,854,775,808	8000 0000 0000 0000	LINT	8	Minimum signed 64 bit value

TIA Datatypes

List of data types used by Simatic S7. The page contains the more modern TIA variable types as well as the earlier S7-classic types.

There are four data types in: Boolean, Text, Numeric, and Date/Time. Each data type defines the format of information that can be entered into a data field and stored in your database.

Datatyp	Width (bits)	Range of values	Examples	S7-300/400	S7-1200	S7-1500
Binaries						
BOOL (x)	1 (S7-1500 optimized 1 Byte)	FALSE or TRUE BOOL#0 or BOOL#1 BOOL#FALSE oder BOOL#TRUE	TRUE BOOL#1 BOOL#TRUE	X	X	X
BYTE (b)	8	B#16#00 .. B#16#FF 0 .. 255 2#0 .. 2#11111111	15, BYTE#15, B#15	X	X	X
WORD (w)	16	W#16#0000 .. W#16#FFFF 0 .. 65.535 B#(0, 0) .. B#(255, 255)	55555, WORD#55555, W#555555	X	X	X
DWORD (dw)	32	DW#16#0000 0000 .. DW#16#FFFF FFFF 0 .. 4,294,967,295	DW#16#DEAD BEEF B#(111, 222, 255, 200)	X	X	X
LWORD (lw)	64	LW#16#0000 0000 0000 0000 .. LW#16#FFFF FFFF FFFF FFFF 0 .. 18.446.744.073.709.551.615	LW#16#DEAD BEEF DEAD BEEF B#(111, 222, 255, 200, 111, 222, 255, 200)	-	-	X
Datatyp	**Width (bits)**	**Range of values**	**Examples**	**S7-300/400**	**S7-1200**	**S7-1500**
Integers						
SINT (si)	8	-128 .. 127 (hex only positive) 16#0 .. 16#7F	+42, SINT#+42 16#1A, SINT#16#2A	-	X	X
INT (i)	16	-32.768 .. 32.767 (hex only positive) 16#0 .. 16#7FFF	+1234, INT#+3221 16#1ABC	X	X	X
DINT (di)	32	-2.147.483.648 .. +2.147.483.647 (hex only positive) 16#00000000 .. 16#7FFFFFFF	123456, DINT#123.456, 16#1ABC BEEF	X	X	X

Datatyp	Width (bits)	Range of values	Examples	S7-300/400	S7-1200	S7-1500
USINT (usi)	8	0 .. 255 16#00 .. 16#FF	42, USINT#42 16#FF	-	X	X
UINT (ui)	16	0 .. 65.535 16#0000 .. 16#FFFF	12.345, UINT#12345 16#BEEF	-	X	X
UDINT (udi)	32	0 .. 4.294.967.295 16#00000000 .. 16#FFFF FFFF	1.234.567.890, UDINT#1234567890	-	X	X
LINT (li)	64	-9.223.372.036.854.775.808 .. +9.223.372.036.854.775.807	+1.234.567.890.123.456.789, LINT#+1.234.567.890.123.456.789	-	-	X
ULINT (uli)	64	0 .. 18.446.744.073.709.551.615	123.456.789.012.345, ULINT#123.456.789.012.345	-	-	X
Datatyp	**Width (bits)**	**Range of values**	**Examples**	**S7-300/400**	**S7-1200**	**S7-1500**
floating point numbers						
REAL (r) →details	32	-3.402823e+38 .. -1.175 495e-38 .. +1.175 495e-38 .. +3.402823e+38	0.0, REAL#0.0 1.0e-13, REAL#1.0e-13	X	X	X
LREAL (lr) →details	64	-1.7976931348623158e+308 .. -2.2250738585072014e-308 .. +2.2250738585072014e-308 .. +1.7976931348623158e+308	0.0, LREAL#0.0	-	X	X
Datatyp	**Width (bits)**	**Range of values**	**Examples**	**S7-300/400**	**S7-1200**	**S7-1500**
Times						
S5TIME (s5t) →details	16	S5T#0H_0M_0S_0MS .. S5T#2H_46M_30S_0MS	S5T#10s, S5TIME#10s	X	-	X
TIME (t)	32	T#-24d20h31m23s648ms .. T#+24d20h31m23s647ms	T#13d14h15m16s630ms, TIME#1d2h3m4s5ms	X	X	X
LTIME (lt)	64	LT#-106751d23h47m16s854ms775us808ns .. LT#+106751d23h47m16s854ms775us807ns	LT#1000d10h15m24s130ms152us15ns, LTIME#200d2h2m1s8ms652us315ns	-	-	X
Timer operations: **IEC timers**, **TON** (Generate on-delay), **TOF** (Generate off-delay), **TP** (Generate pulse), **TONR** (Time accumulator)						
Datatyp	**Width (bits)**	**Range of values**	**Examples**	**S7-300/400**	**S7-1200**	**S7-1500**
Counters						
CHAR	8	ASCII character set	'A', CHAR#'A'	X	X	X
WCHAR (wc) →details	16	Unicode character set	WCHAR#'A'	-	X	X
STRING (s) →details	n+2 (Byte)	0 .. 254 characters (n)	'Name', STRING#'lamaPLC'	X	X	X
WSTRING (ws) →details	n+2 (Word)	0 .. 16382 characters (n)	WSTRING#'lamaPLC'	-	X	X
Counter operations: **CTU** (count up), **CTD** (count down), **CTUD** (count up and down)						
Datatyp	**Width (bits)**	**Range of values**	**Examples**	**S7-300/400**	**S7-1200**	**S7-1500**
Date & time						
DATE (d)	16	D#1990-01-01 .. D#2168-12-31	D#2020-08-14, DATE#2020-08-14	X	X	X
TOD (tod) (TIME_OF_DAY)	32	TOD#00:00:00.000 .. TOD#23:59:59.999	TOD#11:22:33.444, TIME_OF_DAY#11:22:33.444	X	X	X
LTOD (ltod) (LTIME_OF_DAY)	64	LTOD#00:00:00.000000000 .. LTOD#23:59:59.999999999	LTOD#11:22:33.444_555_111, LTIME_OF_DAY#11:22:33.444_555_111	-	-	X
DT (dt) (DATE_AND_TIME)	64	Min.: DT#1990-01-01-0:0:0 Max.: DT#2089-12-31-23:59:59.999	DT#2020-08-14-2:44:33.111, DATE_AND_TIME#2020-08-14-11:22:33.444	X	-	X
LDT (ldt) (L_DATE_AND_TIME)	64	Min.: LDT#1970-01-01-0:0:0.000000000, 16#0 Max.: LDT#2262-04-11-23:47:16.854775807, 16#7FFF_FFFF_FFFF_FFFF	LDT#2020-08-14-1:2:3.4	-	-	X
DTL (dtl)	96	Min.: DTL#1970-01-01-00:00:00.0 Max.: DTL#2554-12-31-23:59:59.999999999	DTL#2020-08-14-10:12:13.23	-	X	X
Datatyp	**Width (bits)**	**Range of values**	**Examples**	**S7-300/400**	**S7-1200**	**S7-1500**
Pointers						
POINTER (p) →details	48		Symbolic: “DB”.“Tag” Absolute: P#10.0 P#DB4.DBX3.2	X	-	X
ANY (any)	80		Symbolic: “DB”.StructVariable.firstComponent Absolut: P#DB11.DBX12.0 INT 3 P#M20.0 BYTE 10	X	-	X

Datatyp	Width (bits)	Range of values	Examples	S7-300/400	S7-1200	S7-1500
VARIANT (var)	0		Symbolic: “Data_TIA_Portal”. StructVariable.firstComponent Absolute: %MW10 P#DB10.DBX10.0 INT 12	-	X	X
BLOCK_FB	0		-	X	-	X
BLOCK_FC	0		-	X	-	X
BLOCK_DB	0		-	X	-	-
BLOCK_SDB	0		-	X	-	-
VOID	0		-	X	X	X
PLC_DATA_TYPE	0		-	X	X	X

TIA Coding of data types

The following table lists the coding of data types for the ANY pointer:

Hexadecimal code	Data type	Description
B#16#00	NIL	Null pointer
B#16#01	BOOL	Bits
B#16#02	BYTE	bytes, 8 bits
B#16#03	CHAR	8-bit characters
B#16#04	WORD	16-bit words
B#16#05	INT	16-bit integers
B#16#06	DWORD	32-bit words
B#16#07	DINT	32-bit integers
B#16#08	REAL	32-bit floating-point numbers
B#16#0B	TIME	Time duration
B#16#0C	S5TIME	Time duration
B#16#09	DATE	Date
B#16#0A	TOD	Date and time
B#16#0E	DT	Date and time
B#16#13	STRING	Character string
B#16#17	BLOCK_FB	Function block
B#16#18	BLOCK_FC	Function
B#16#19	BLOCK_DB	Data block
B#16#1A	BLOCK_SDB	System data block
B#16#1C	COUNTER	Counter
B#16#1D	TIMER	Timer

TIA Coding of the memory area

The following table lists the coding of the memory areas for the ANY pointer:

Hexadecimal code	Area	Description
B#16#80	P	I/O
B#16#81	I	Memory area of inputs
B#16#82	Q	Memory area of outputs

Hexadecimal code	Area	Description
B#16#83	M	Memory area of bit memory
B#16#84	DBX	Data block
B#16#85	DIX	Instance data block
B#16#86	L	Local data
B#16#87	V	Previous local data

Simatic classic datatypes (S7-300 / S7-400)

Type and description	size in bits	format options	Area and number notation (lower .. higher value)	example in STL
BOOL (Bit)	1	Boolean text	TRUE/FALSE	TRUE
BYTE (Byte)	8	Hexadecimal number	B#16#0 to B#16#FF	L B#16#10 L byte#16#10
WORD (Word)	16	Binary number	2#0 to 2#1111_1111_1111_1111	L 2#0001_0000_0000_0000
Hexadecimal number	W#16#0 to W#16#FFFF	L W#16#1000 L word#16#1000		
BCD	C#0 to C#999	L C#998		
Decimal number unsigned	B#(0,0) to B#(255,255)	L B#(10,20) L byte#(10,20)		
DWORD (Double word)	32	Binary number	2#0 to 2#1111_1111_1111_1111_1111_1111_1111_1111	L 2#1000_0001_0001_1000_1011_1011_0111_1111
Hexadecimal number	W#16#0000_0000 to W#16#FFFF_FFFF	L DW#16#00A2_1234 L dword#16#00A2_1234		
Decimal number unsigned	B#(0,0,0,0) to B#(255,255,255,255)	L B#(1, 14, 100, 120) L byte#(1,14,100,120)		
INT (Integer)	16	Decimal number signed	-32768 to 32767	L 101
DINT (Double integer)	32	Decimal number signed	L#-2147483648 to L#2147483647	L L#101
REAL (Floating-point number)	32	IEEE Floating-point number	Upper limit +/-3.402823e+38 Lower limit +/-1.175495e-38	L 1.234567e+13
S5TIME (SIMATIC time)	16	S7 time in steps of 10ms (default)	S5T#0H_0M_0S_10MS to S5T#2H_46M_30S_0MS and S5T#0H_0M_0S_0MS	L S5T#0H_1M_0S_0MS L S5TIME#0H_1H_1M_0S_0MS
TIME (IEC time)	32	IEC time in steps of 1 ms, integer signed	T#24D_20H_31M_23S_648MS to T#24D_20H_31M_23S_647MS	L T#0D_1H_1M_0S_0MS L TIME#0D_1H_1M_0S_0MS
DATE (IEC date)	16	IEC date in steps of 1 day	D#1990-1-1 to D#2168-12-31	L D#1996-3-15 L DATE#1996-3-15
TIME _OF_DAY (Time)	32	Time in steps of 1 ms	TOD#0:0:0.0 to TOD#23:59:59.999	L TOD#1:10:3.3 L TIME_OF_DAY#1:10:3.3
CHAR (Character)	8	ASCII characters	A', 'B' etc.	L 'E'

Data-types

Pointer ## ANY ## Variant ## ARRAY ## Real ## Char ## WChar ## String ## WString ## S5time ## IEC Timers ##

Pointer

An example above to the pointer: P#DB100.DBX14.0 WORD 4
The following elements make up a pointer:

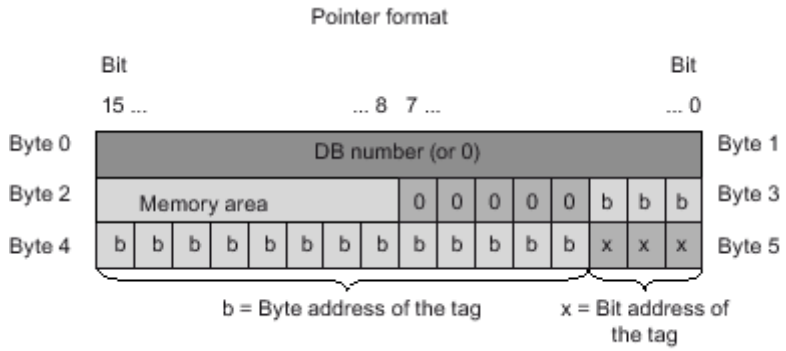
P#	Pointer identifier
DB100	Start Data block / memory area
.DB14.0	Start offset within the data block
WORD 4	Length of the data to be included in the pointer

You cannot use MOVE to access pointer data. This is because the Pointer is considered an **“Any” data type**, which MOVE does not accept. BLKMOV must be used instead, and in order to use that, Optimized Data must be turned off!

A parameter of the type POINTER is a pointer that can point to a specific tag. It occupies 6 bytes (48 bits) in memory and may contain the following tag information:

- DB number, or 0 if the data is not stored in a DB
- Memory area in the CPU
- Tag address

The following figure shows the structure of parameter type POINTER:



Any

An ANY type parameter points to the start of a data area and specifies its length. An ANY pointer occupies 10 bytes of memory and may contain the following information:

Data type:
Data type of the elements of the data area

Repetition factor:
Number of elements of the data area

DB number:
Data block that contains the declaration of data area elements.

Memory area:
Memory area of the CPU that stores the data area elements.

Start address of the data in the format “byte.bit”:
Data area start identified by the ANY pointer.

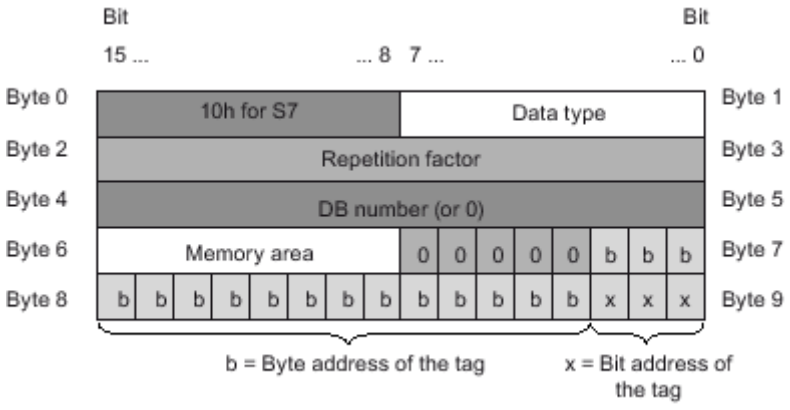
Zero pointer:
Use the zero pointer to indicate a missing value. A missing value may indicate that no value exists, or that the value is not yet known. A zero value represents the absence of a value, but is also a value.

In the programming languages SCL and STL, memory of any kind can be transferred upon a block call if an ANY pointer has been programmed at a block parameter.

The ANY pointer cannot, however, store any information on the structure of the memory. For example, the ANY pointer does not save the information that it points to a tag of the PLC data type. The ANY pointer sees this as an ARRAY of BYTE.

Parameters of the ANY data type can be passed to system function blocks (SFBs) or system functions (SFCs).

Memory area
For an S7-1500 CPU, the ANY pointer can also only point to memory areas with “Standard” access mode.



Variant

A parameter of the VARIANT type is a pointer that can point to tags of different data types other than an instance. The VARIANT pointer can be an object of an elementary data type, such as INT or REAL. It can also be a STRING, DTL, ARRAY of STRUCT, UDT, or ARRAY of UDT. The VARIANT pointer can

recognize structures and point to individual structure components. An operand of data type VARIANT occupies no space in the instance data block or work memory. However, it will occupy memory space on the CPU.

A tag of the VARIANT type is not an object but rather a reference to another object. Individual elements of the VARIANT type can only be declared on formal parameters within the block interface of a function in the VAR_IN, VAR_IN_OUT and VAR_TEMP sections. For this reason, it cannot be declared in a data block or in the static section of the block interface of a function block, for example, because its size is unknown. The size of the referenced objects can change.

You can use VARIANT to generate generic function blocks or functions. When a block is called, you can connect the parameters of the block to tags of any data type. When a block is called, the type information of the tag is transferred in addition to a pointer to the tag. The code of the block can then be executed according to its type in line with the tag transferred during runtime.

If, for example, a block parameter of a function has the VARIANT data type, then a tag of the integer data type can be transferred at one point in the program, and a tag of the PLC data type can be transferred at another point in the program. With the help of the VARIANT instructions, the function is then in a position to react to the situation without errors.

You can only point to a complete data block if it was originally derived from a user-defined data type (UDT).

Array

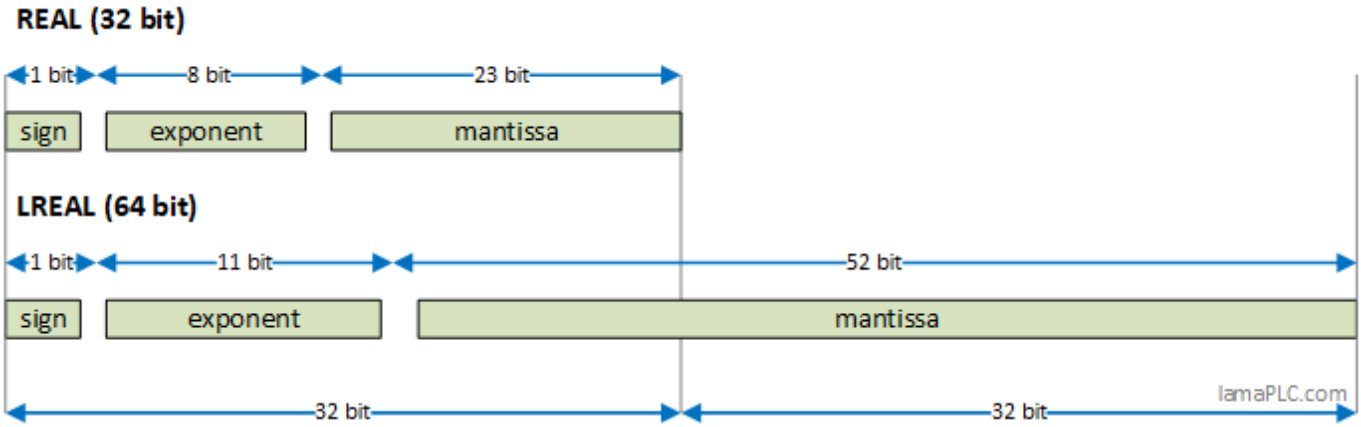
The Array data type represents a data structure that consists of a fixed number of components of the same data type. All data types except Array are permitted.

A tag with the Array data type always starts at a WORD limit.

The array components are addressed by means of an index. In the array declaration, the index limits are defined in square brackets after the keyword Array. The low limit must be smaller than or equal to the high limit. An array may contain up to six dimensions, the limits of which can be specified separated by a comma.

Length: Number of components * length of the data type
Format: Array [low limit...high limit] of <data type>
Index limits: Depending on the CPU, the storage capacity of a data block is limited and the number of components of the Array is therefore also limited. However, you may initialize the addressing of the array components at any position within index limits.
Index limits (16-bit limits): [-32768..32767] of <data type>
Index limits (32-bit limits): [-2147483648..2147483647] of <data type>
Data type: Bit strings, integers, floating-point numbers, timers, character strings, structures

Real, LReal IEEE 754



- **Sign** (of Mantissa) : 0: positive, 1: negative
- (Biased) **exponent** : The exponent field needs to represent both positive and negative exponents.
- (Normalised) **mantissa** : Mantissa is part of a number in scientific notation or a floating-point number, consisting of its significant digits.

Char

An operand of data type CHAR has a length of 8 bits and occupies one BYTE in the memory.

The CHAR data type stores a single character in ASCII format. You can find information on the encoding of special characters under “See also > STRING”.

Length (bits): 8
Format: ASCII characters
Value range: ASCII character set
Example of value inputs: 'A', CHAR#'A'

WChar

An operand of data type WCHAR (Wide Characters) has a length of 16 bits and occupies two BYTE in the memory.

The WCHAR data type saves a single character of an expanded character set which is stored in Unicode format. However, only a subset of the entire Unicode range is covered. When a control character is entered, it is represented with a dollar sign.

Length (bits): 16 bit
Format: Unicode
Range of values: $0000 - $D7FF
Example of value input: WCHAR#'a'

String

String types in S7 are not NULL *“terminated”* like C-style strings. They instead have 2 “hidden” characters that precede the string data. The first hidden character is the maximum size of the string, which is 'n' in the example above, and the second hidden character is the actual length of the string (i.e. the number of characters stored).

So the string definition MyStr: STRING[10]:=*“abcdef”* would contain the following ASCII codes:

10, 06, 97, 98, 99, 100, 101, 102

10: maximum (declared) length of the character string
06: the current length of the string
97, 98: “a”, “b”,..

Length (bits): n + 2 (An operand of the STRING data type occupies two bytes more than the specified maximum length in the memory)
Format: ASCII character string incl. special characters
Value range: 0 to 254 characters
Example of value inputs: 'Name', STRING#'NAME'

WString

An operand of data type WSTRING (Wide String) stores several Unicode characters of data type WCHAR in one character string. If you do not specify a length, the character string has a preset length of 254 characters. In a character string, all characters of the Unicode format are permitted. This means you can also use Chinese characters in a character string.

Length (bits): n + 2 (An operand of the WSTRING data type occupies two WORDs more in the memory than the specified maximum length)
Format: Unicode character string;
Range of values: Preset value: 0 to 254 characters, maximal possible value: 0 to 16382
Example of value inputs: WSTRING#'Hello World'

When declaring an operand of data type WSTRING you can define its length using square brackets (for example WSTRING[10]). If you do not specify a length, the length of the WSTRING is set to 254 characters by default. You can declare a length of up to 16382 characters (WSTRING[16382]).

The specification of the characters occurs in single quotes and always with the qualifier WSTRING#.

A character string can also contain special characters. The escape character $ is used to identify control characters, dollar signs and single quotation marks.

Character	Hex	Meaning	Example
$L or $l	000A	Line feed	'$LText', '$000AText'
$N	000A and 000D	Line break The line break occupies 2 characters in the character string.	'$NText', '$000A$000DText'
$P or $p	000C	Page feed	'$PText', '$000CText'

Character	Hex	Meaning	Example
$R or $r	000D	Carriage return (CR)	'$RText','$000DText'
$T or $t	0009	Tab	'$TText', '$0009Text'
$$	0024	Dollar sign	'100$$t', '100$0024t'
$'	0027	Single quotation mark	'$'Text$'','$0027Text$0027

The maximum length of the character string can be specified during the declaration of an operand using square brackets after the keyword WSTRING (for example, WSTRING[4]). If the specification of the maximum length is omitted, the standard length of 254 characters is set for the respective operand.

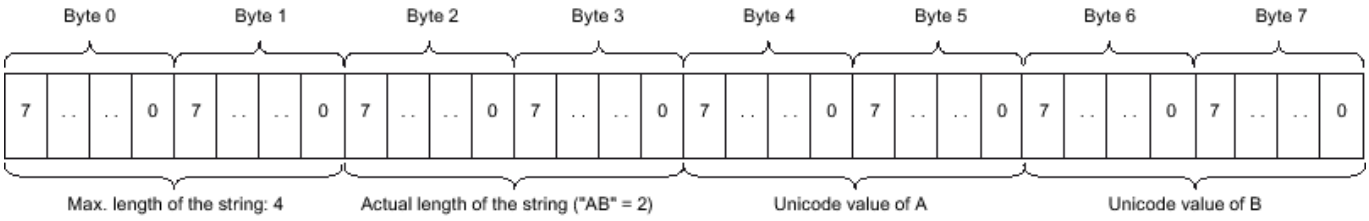
If the actual length of a specified character string is shorter than the declared maximum length, the characters are written to the character string left-justified and the remaining character spaces remain undefined. Only occupied character spaces are considered in the value processing.

Access to block parameters of data type WSTRING
Operands of the data type WSTRING can be transferred as parameters up to the maximum length for blocks with *“optimized”* access.

For function blocks (FB) with *“standard”* access, operands of the data type WSTRING can be declared as parameters in all sections of the block interface except in the section *“InOut”*. For a function (FC) with “standard” access, only operands of the data type STRING can be transferred as parameters.

The function value of an FC in the *“Return”* section and expressions in the SCL programming language are another exception to this rule. In these cases, the WSTRING tag must not be longer than 1022 characters. If you need a WSTRING tag with more than 1022 characters, declare a tag of the data type *“WSTRING”* with the required character length in the section “Temp” and assign the function value to the tag.

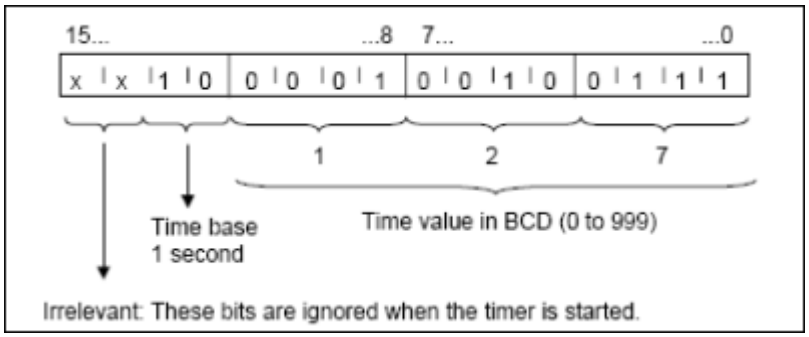
Example
The example below shows the byte sequence if the WSTRING[4] data type is specified with output value 'AB':



S5TIME

- Underscores in time and date are optional
- It is not necessary to specify all time units (for example: T# 5h10s is valid)
- Maximum time value = 9,990 seconds or 2H_46M_30S

S5TIME Struktur



Time base	Binary Code
10 ms	00
100 ms	01
1 s	10
10 s	11

IEC timers (TP, TONR, TON, TOF)

You can find more information about these timer types here: SCL commands (timer, counter)

Indirect addressing in Simatic Classic

Indirect addressing	**Example**
Indirect addressing of a DB	Data type “BLOCK_DB”
Indirect addressing of DB tags	#block.%DBW3 WORD_TO_BLOCK_DB(#myWord).%DBW3 #block.DW(IDX := #myInt) WORD_TO_BLOCK_DB(#myWord).DW(IDX:=#myInt) %DB1.DW(IDX :=#myInt)
Indirect addressing of I/O	QB(IDX :=#myInt):P
Indirect addressing of PLC tags	IX(IDX :=#myInt1,Bit:=#myInt2) QB(IDX :=#myInt) MW(IDX :=#myInt)

Indirect addressing in TIA Portal

In many cases, indirect addressing may be necessary in PLC technology. Its typical application is the preparation of signals for communication and SCADA, as well as the transformation of communication and SCADA signals and the “unpacking” of bits.

For example

- In the case of SCADA transmission, it is easier to use a WORD and transfer 16 bits in it
- Modbus communication is based on WORD registers. These are easy to handle with indirect addressing
- Many measuring units use two of the Modbus WORD registers (DWORD) to transfer one REAL. Structurally, DWORD and REAL are not the same, but the signals can be easily read with indirect addressing

TIA Portal slice access

- from Bit to: BYTE, WORD, DWORD
- from Byte to: WORD, DWORD
- from Word to: DWORD

```
bit := byte.%X1;    // bit 1 from byte
bit := word.%X4;    // bit 4 from word
bit := dword.%X11;  // bit 11 from dword

byte := word.%B1;   // 2.byte from word
byte := dword.%B2;  // 3. byte from dword

word := dword.%W0;  // 1. word from dword
```

The following example is a SPLIT function that splits a WORD Input variable into bits:

```
// FC Input : inWord (Word)
// FC output: 16 variable bit0..bit15 (Bool)
// splitting
#bit0 := #inWord.%X0;
#bit1 := #inWord.%X1;
#bit2 := #inWord.%X2;
#bit3 := #inWord.%X3;
#bit4 := #inWord.%X4;
#bit5 := #inWord.%X5;
#bit6 := #inWord.%X6;
#bit7 := #inWord.%X7;
#bit8 := #inWord.%X8;
#bit9 := #inWord.%X9;
```

```
#bitA := #inWord.%X10;
#bitB := #inWord.%X11;
#bitC := #inWord.%X12;
#bitD := #inWord.%X13;
#bitE := #inWord.%X14;
#bitF := #inWord.%X15;
```

And this is a JOIN function that assembles a WORD from 16 bits:

```
// FC Input: 16 variable bit00..bit15 (Bool)
// FC Output : OUT (Word)
// JOIN
// Assemble bits to a word
#OUT.%X0 := #bit00;
#OUT.%X1 := #bit01;
#OUT.%X2 := #bit02;
#OUT.%X3 := #bit03;
#OUT.%X4 := #bit04;
#OUT.%X5 := #bit05;
#OUT.%X6 := #bit06;
#OUT.%X7 := #bit07;
#OUT.%X8 := #bit08;
#OUT.%X9 := #bit09;
#OUT.%X10 := #bit10;
#OUT.%X11 := #bit11;
#OUT.%X12 := #bit12;
#OUT.%X13 := #bit13;
#OUT.%X14 := #bit14;
#OUT.%X15 := #bit15;
```

simatic, s7, scl, datatype, IEEE 754, string, wstring, S5TIME, indirect addressing, slice access, PLC, TIA

This page has been accessed for: Today: 32, Until now: 401

From:
https://lamaplc.com/ - **lamaPLC**

Permanent link:
https://lamaplc.com/doku.php?id=simatic:typedef

Last update: **2026/04/21 20:46**