LamaPLC: Simatic S7 SCL commands: Move/memory operations

BLKMOV

You can use the “Move block” instruction to move the content of a memory area (source area) to another memory area (destination area). The move operation takes place in the direction of ascending addresses. You use ANY pointer to define the source and destination area.

_FB_ **Retval := BLKMOV (**		INT	If there is no error, **Retval** is 0. Otherwise: a summary table of the “retVal” codes can be found here: Simatic S7 error- and statuscodes
	SRCBLK := VARIANT **,**	input	**Source block:** this block will be copied. The feature does not scan data types; only the size of the area should match the size of the target.
	DSTBLK ⇒ VARIANT **,**	output	**Destination block:** the first block is copied here (overwrites everything).
);			

It is important to disable optimization in the DBs used by BLKMOV!
Check: Switch off "optimization" attribute of DBs

BLKMOV call example 2

_FB_ RetVal := **BLKMOV** (SRCBLK := P#M12.0 BYTE 10, DSTBLK ⇒ P#DB1.DBX0.0 BYTE 10);

BLKMOV call example 1

The sample program reads the local time in DT and splits the variable into BYTEs.

The example program below can be downloaded here:

blkmov_example_1.scl

.

The code:

```
// author OB121 / Sandor Vamos
// ob121.com; 2022.04.11.
```

```
// BLKMOV example: read local time and convert byte array
//
// More information:
// https://www.ob121.com/doku.php?id=de:s7:scl_reference#blkmov

// read local time to date_and_time
#state := RD_LOC_T("BLKMOV_DB".datum_dt);

// move date_and_time to byte array
#state := BLKMOV
(SRCBLK := P#db4.dbx0.0 BYTE 8,
 DSTBLK => P#db4.dbx8.0 BYTE 8,
 ENO => ENO);
```

The DB:

Static				
datum_dt	Date_And_Time	0.0	DT#1990-01-01-0	DT#2022-04-11-23:10:45.666
datum	Array[0..7] of Byte	8.0		
datum[0]	Byte	8.0	16#0	16#22
datum[1]	Byte	9.0	16#0	16#04
datum[2]	Byte	10.0	16#0	16#11
datum[3]	Byte	11.0	16#0	16#23
datum[4]	Byte	12.0	16#0	16#10
datum[5]	Byte	13.0	16#0	16#45
datum[6]	Byte	14.0	16#0	16#66
datum[7]	Byte	15.0	16#0	16#62

>> Back to LamaPLC main menu (SCL commands)

MOVE_BLK

Copy arrays in memory blocks.

FC **MOVE_BLK** (IN:= source block; OUT:= target block; COUNT:= number of elements; ENO ⇒ operation enable);

The MOVE_BLK command can be used to move a memory area (IN) to another memory area (OUT) by a specified length (COUNT). The command does not implicitly monitor for irregular addresses, so it is worth monitoring the (ENO) output.

The operation can be performed with the ARRAY types (in which case COUNT determines the number of items). In case of over-addressing, ENO indicates.

For pointer memory area move check: BLKMOV.

Example of moving ARRAY with MOVE_BLK (ENO OK):

```
1  // author Sandor Vamos
2  // obl21.com; 2022.03.07.
3  // MOVE_BLK example with ARRAYS without
4  // ENO error
5  //
6  // More information:
7  // https://www.obl21.com/doku.php?id=de
8  //
9  // source for MOVE_BLK
10 #in_byte_array[1] := 12;
11 #in_byte_array[2] := 24;
12
13 // delete target field
14 FOR #i := 0 TO 10 DO
15     #out_byte_array[#i] := 0;
16 END_FOR;
17
18 // MOVE_BLK
19 MOVE_BLK(IN:=#in_byte_array[1],
20           COUNT:=2,
21           OUT=>#out_byte_array[4],
22           ENO => ENO);
23
24 // check ENO
25 IF ENO THEN
26     #error_code := #no_error;
27 ELSE
28     #error_code := #error;
29 END_IF;
30
31 // check target
32 #temp_byte := #out_byte_array[0];
33 #temp_byte := #out_byte_array[1];
34 #temp_byte := #out_byte_array[2];
35 #temp_byte := #out_byte_array[3];
36 #temp_byte := #out_byte_array[4];
37 #temp_byte := #out_byte_array[5];
38 #temp_byte := #out_byte_array[6];
39
```

	#in_byte_ar...	16#0C
	#in_byte_ar...	16#18
	#i	-
▶	#out_byte_a...	-
	#in_byte_ar...	16#0C
	#out_byte_a...	16#0C
	ENO	TRUE
▶	Result	TRUE
▶	#error_code	1
▶	#error_code	
▶	#temp_byte	16#00
▶	#temp_byte	16#00
▶	#temp_byte	16#00
▶	#temp_byte	16#00
▶	#temp_byte	16#0C
▶	#temp_byte	16#18
▶	#temp_byte	16#00

Example of moving ARRAY with MOVE_BLK (ENO error!):
I overrated the source area with COUNT: = 3.

```
// source for MOVE_BLK
#in_byte_array[1] := 12;
#in_byte_array[2] := 24;

// delete target field
FOR #i := 0 TO 10 DO
    #out_byte_array[#i] := 0;
END_FOR;

// MOVE_BLK
MOVE_BLK(IN:=#in_byte_array[1],
         COUNT:=3,
         OUT=>#out_byte_array[4],
         ENO => ENO);

// check ENO
IF ENO THEN
    #error_code := #no_error;
ELSE
    #error_code := #error;
END_IF;

// check target
#temp_byte := #out_byte_array[0];
#temp_byte := #out_byte_array[1];
#temp_byte := #out_byte_array[2];
#temp_byte := #out_byte_array[3];
#temp_byte := #out_byte_array[4];
#temp_byte := #out_byte_array[5];
#temp_byte := #out_byte_array[6];
```

#in_byte_ar...	16#0C
#in_byte_ar...	16#18
#i	-
#out_byte_a...	-
#in_byte_ar...	16#0C
#out_byte_a...	16#0C
ENO	FALSE
Result	FALSE
ENO	FALSE
#error_code	2
#temp_byte	16#00
#temp_byte	16#00
#temp_byte	16#00
#temp_byte	16#00
#temp_byte	16#0C
#temp_byte	16#18
#temp_byte	16#00

The example MOVE_BLK program below can be downloaded here:

move_blk_example.scl

.

>> Back to LamaPLC main menu (SCL commands)

FILL_BLK / UFILL_BLK

Fill a memory area (ARRAY) (target range) with a value.

UFILL_BLK: In the case of the UFILL_BLK instruction, the difference is that the operation cannot be interrupted, it must be executed in the same call-cycle.

The instruction can only be executed if the source range and the target range have the same data type.

_FB_ FILL_BLK (			
	IN := 5 **,**	**input:** Binary numbers, integers, floating-point numbers, timers, TOD, LTOD, DATE, CHAR, WCHAR	fill a memory area (target range) with the value of the **IN** input.
	COUNT := 4 **,**	**input:** USINT, UINT, UDINT, ULINT	The number of repeated copy operations is specified with the **COUNT** parameter
	OUT ⇒ “db_prog”.intArray[2]	**output:**Binary numbers, integers, floating-point numbers, timers, TOD, LTOD, DATE, CHAR, WCHAR	The target range is filled beginning with the address specified at the **OUT** output. When the instruction is executed, the value at the input **IN** is moved to the target range as often as specified by the value of the **COUNT** parameter.
);			

```
18
19 FILL_BLK(IN:=5,
20          COUNT:=4,
21          OUT=>"db_prog".intArray[2]);
22
23
```

"db_prog".intArray[2]	5

call FILL_BLK: The program had to fill only 4 array elements with “5”.

db_prog

	Name	Data type	Offset	Start value	Monitor value
1	Static				
2	intArray	Array[0..10] ...	0.0		
3	intArray[0]	Int	0.0	0	0
4	intArray[1]	Int	2.0	0	0
5	intArray[2]	Int	4.0	0	5
6	intArray[3]	Int	6.0	0	5
7	intArray[4]	Int	8.0	0	5
8	intArray[5]	Int	10.0	0	5
9	intArray[6]	Int	12.0	0	0
10	intArray[7]	Int	14.0	0	0
11	intArray[8]	Int	16.0	0	0
12	intArray[9]	Int	18.0	0	0
13	intArray[10]	Int	20.0	0	0

the result in DB

>> Back to LamaPLC main menu (SCL commands)

Simatic, SCL, TIA, blkmov, move blk, memory, move

This page has been accessed for: Today: 2, Until now: 306

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

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

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