

Lenze

Manual

IEC 61131-3

inside

***Global Drive
PLC Developer Studio***



Global Drive

Function library

LenzeMemDrv.lib

The function library **LenzeMemDrv.lib** can be used for the following Lenze PLCs:

	Type	from hardware version	from software version
9300 Servo PLC	EVS93XX-xI	6A	6.0
9300 Servo PLC	EVS93XX-xT	6A	6.0
Drive PLC	EPL10200	1A	6.0
ECSxA	ECSxAxxx	1C	7.0

Important note :

The software is supplied to the user as described in this document. Any risks resulting from its quality or use remain the responsibility of the user. The user must provide all safety measures protecting against possible maloperation.

We do not take any liability for direct or indirect damage, e.g. profit loss, order loss or any loss regarding business.

© 2002 Lenze Drive Systems GmbH

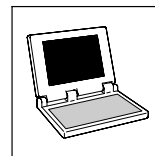
No part of this documentation may be copied or made available to third parties without the explicit written approval of Lenze Drive Systems GmbH.

All information given in this online documentation has been carefully selected and tested for compliance with the hardware and software described. Nevertheless, discrepancies cannot be ruled out. We do not accept any responsibility or liability for any damage that may occur. Required corrections will be included in updates of this documentation.

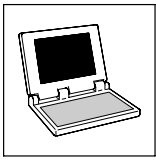
Windows, Windows NT and MS-DOS are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A and/or other countries.

IBM and VGA are registered trademarks of International Business Machines, Inc.

All other product names are trademarks of the corresponding owners.

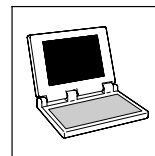


1 Preface and general information	1-1
1.1 About this Manual	1-1
1.1.1 Conventions used in this Manual	1-1
1.1.2 Layout of the descriptions	1-2
1.1.3 Pictographs used in this Manual	1-2
1.1.4 Terminology used	1-2
1.2 Version identifiers of the function library	1-3
2 Introduction	2-1
2.1 General information about the memory management	2-1
2.2 Overview of the memory management functions	2-1
2.2.1 Functions supported by Lenze PLCs	2-2
2.2.2 Backup memory of Lenze PLCs	2-2
3 General memory functions	3-1
3.1 Overview	3-1
3.2 L_MemGetStateDriver	3-2
3.3 L_MemGetStateOfMemory	3-3
4 Functions for RAM memory access	4-1
4.1 Overview	4-1
4.2 L_MemCopyFromRam	4-2
4.3 L_MemCopyToRam	4-4
4.4 L_MemReadDINT	4-6
4.5 L_MemReadDWORD	4-7
4.6 L_MemReadINT	4-8
4.7 L_MemReadWORD	4-9
4.8 L_MemWriteDINT	4-10
4.9 L_MemWriteDWORD	4-11
4.10 L_MemWriteINT	4-12
4.11 L_MemWriteWORD	4-13
5 Functions for FLASH memory access	5-1
5.1 Overview	5-1
5.2 L_MemClearFlash	5-2
5.3 L_MemLoadRamFromFlash	5-3
5.4 L_MemSaveRamToFlash	5-5



Function library LenzeMemDrv.lib

Contents



1 Preface and general information

1.1 About this Manual

This Manual contains information about the **LenzeMemDrv.lib** function library for the **Drive PLC Developer Studio**.

- The **LenzeMemDrv.lib** function library includes functions for read/write access to the additional backup memory (RAM, FLASH) of the PLC.

1.1.1 Conventions used in this Manual

This Manual uses the following conventions to distinguish between different types of information:

Variable identifiers

are written in italics in the explanation:

- “Use *bReset...*”



Tip!

Information about the conventions used for the variables of the Lenze system blocks, function blocks and functions can be found in the appendix of the DDS online documentation “Introduction into IEC61131-3 programming”. The conventions ensure universal and uniform labelling and make reading the PLC program easier.

Lenze functions/function blocks

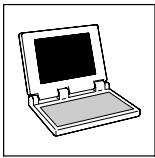
can be recognized by their names. They always begin with an “L_”:

- “Use the function **L_MemCopyToRam...**”

Program listings

are written in “Courier”, keywords are printed in bold:

- “**IF** (ReturnValue < 0) **THEN...**”



Function library LenzeMemDrv.lib

Preface and general information

1.1.2 Layout of the descriptions

All function/function block and system block descriptions contained in this Manual have the same structure:

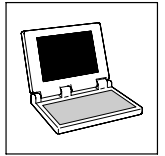
	Function	Function block (FB)/ System block (SB)
①	Headline stating the function and the function identifier	
②	Declaration of the function: - Data type of the return value - Function identifier - List of transfer parameters	-
③	Short description of the most important properties	
④	Function chart including all associated variables - Transfer parameters - Return value	FB/SB chart including all associated variables - Input variables - Output variables
⑤	Table giving information about the transfer parameters: - Identifiers - Data type - Possible settings - Info	Table giving information about the input and output variables: - Identifiers - Data type - Type of variable - Possible settings - Info
⑥	Table giving information about the return value: - Data type of the return value - Possible return values and their meaning: - The return values are listed according to priority in descending order, i.e. the return value with the highest priority is listed first.	-
⑦	Additional information (Notes, tips, application examples, etc.)	

1.1.3 Pictographs used in this Manual

	Use of pictographs	Signal words
Warning of material damage		Stop! Warns of potential damage to material . Possible consequences if disregarded: Damage to the controller/drive system or its environment.
More notes		Tip! Note! Indicates a tip or note.

1.1.4 Terminology used

Term	In the following text used for
DDS	Drive PLC Developer Studio
FB	Function block
GDC	Global Drive Control (parameterization program from Lenze)
Parameter codes	Codes for setting the functionality of a function block
SB	System block



1.2 Version identifiers of the function library

The version of the function library can be found under the global constant `C_w[Function library name]Version`.

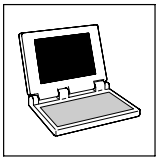
Version identifiers as of PLC software version 7.x:

Constant	Meaning	Example value
<code>C_w[FunctionLibraryName]VersionER</code>	External Release	01
<code>C_w[FunctionLibraryName]VersionEL</code>	External Level	05
<code>C_w[FunctionLibraryName]VersionIR</code>	Internal Release	00
<code>C_w[FunctionLibraryName]VersionBN</code>	Build No.	00

Version: 01 05 00 00

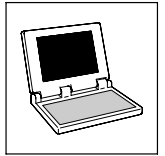
The value of this constant is a hexadecimal code.

- In the example, "01050000" stands for version "1.05".



Function library LenzeMemDrv.lib

Preface and general information



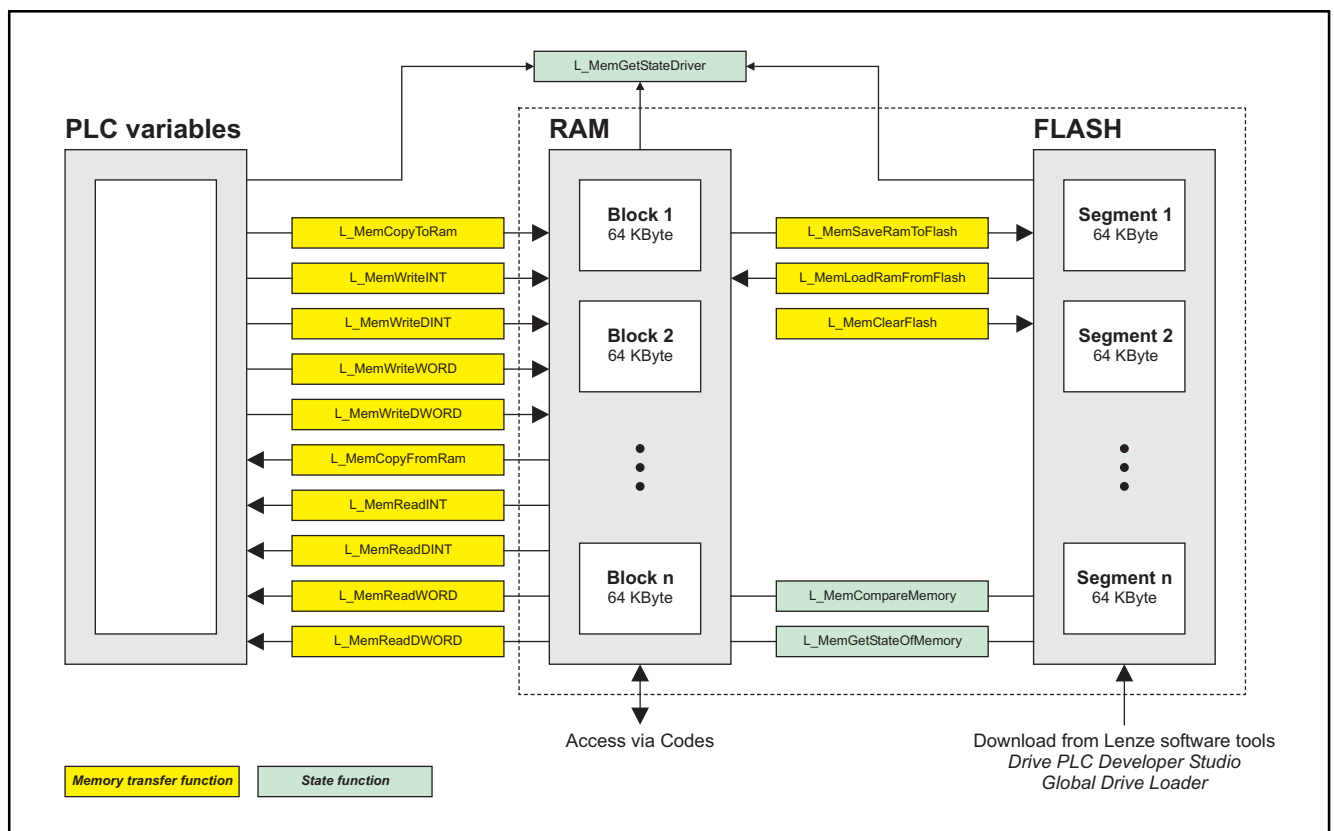
2 Introduction

2.1 General information about the memory management

Lenze PLCs offer the opportunity to copy parts of the variable memory of the PLC (e.g. global variables, arrays, etc.) into a backup memory. In this way, you can, for instance, create recipe memories or measured-value memories.

2.2 Overview of the memory management functions

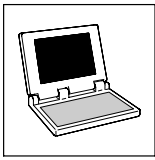
The below figure gives you an overview of the memory management functions available:



Note!

The type and size of the backup memory depends on the Lenze PLC used. This is why you cannot use all functions of the **LenzeMemDrv.lib** function library for all Lenze PLCs.

- In the following sections you can find an overview of the functions supported by the individual Lenze PLCs.



Function library *LenzeMemDrv.lib*

Introduction

Overview of the memory management functions



Tip!

In addition to the functions of the **LenzeMemDrv.lib**, the operating system of the PLC includes codes which allow the following memory-related functions:

- Reading/writing data of a RAM block by means of “autoincrement access” while the PLC program is running (C0504 ... C0508).
- Activating a read/write protection for the FLASH memory (C2006/C2007).

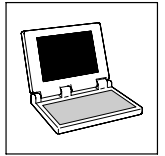
Detailed information can be found in the relevant PLC Manuals.

2.2.1 Functions supported by Lenze PLCs

	9300 Servo PLC	Drive PLC	ECSxA	
General memory functions				
L_MemGetStateDriver	✓	✓	✓	3-2
L_MemGetStateOfMemory	✓	✓	✓	3-3
Functions for RAM memory access				
L_MemCopyFromRam	✓	✓	✓	4-2
L_MemCopyToRam	✓	✓	✓	4-4
L_MemReadDINT	✓	✓	✓	4-6
L_MemReadDWORD	✓	✓	✓	4-7
L_MemReadINT	✓	✓	✓	4-8
L_MemReadWORD	✓	✓	✓	4-9
L_MemWriteDINT	✓	✓	✓	4-10
L_MemWriteDWORD	✓	✓	✓	4-11
L_MemWriteWORD	✓	✓	✓	4-13
L_MemWriteINT	✓	✓	✓	4-12
Functions for FLASH memory access				
L_MemClearFlash	✓			5-2
L_MemLoadRamFromFlash	✓			5-3
L_MemSaveRamToFlash	✓			5-5

2.2.2 Backup memory of Lenze PLCs

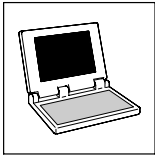
	9300 Servo PLC	Drive PLC	ECSxA
RAM	2 64-kbyte blocks	2 64-kbyte blocks	2 64-kbyte blocks
FLASH	15 64-kbyte segments	-	-



3 General memory functions

3.1 Overview

	9300 Servo PLC	Drive PLC	ECSxA	
L_MemGetStateDriver	✓	✓	✓	 3-2
L_MemGetStateOfMemory	✓	✓	✓	 3-3



Function library *LenzeMemDrv.lib*

General memory functions

L_MemGetStateDriver

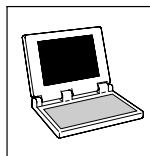
3.2 L_MemGetStateDriver

INT. **L_MemGetStateDriver** (byVoid)

This function shows you the current state of the driver.

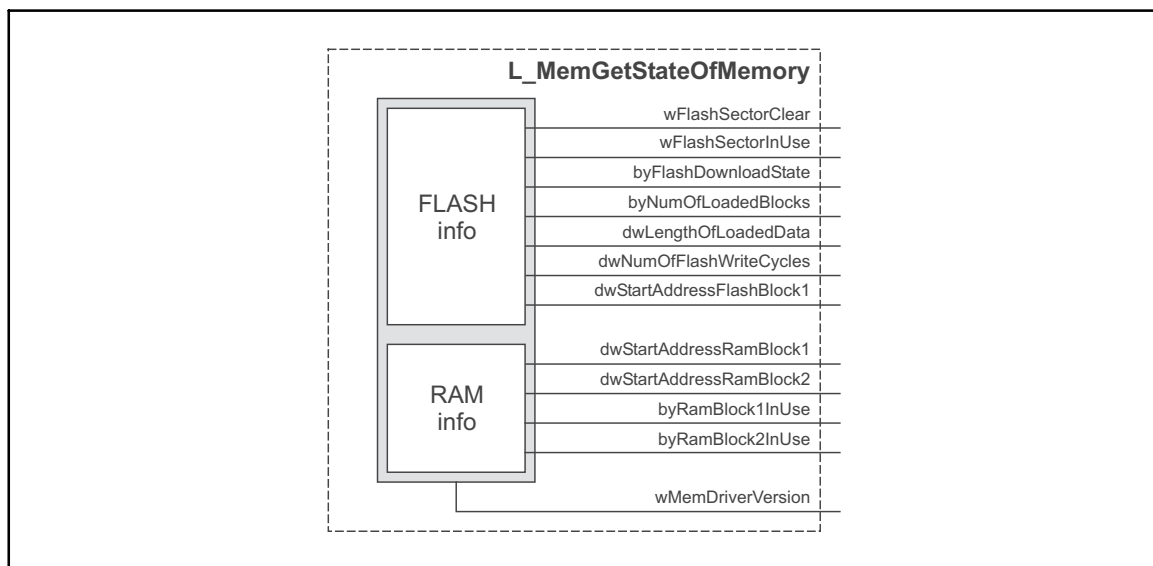
Transfer parameters		
Identifiers	Data type	Info/possible settings
byVoid	Byte	No transfer parameters

Return value: Integer		
Bit	Value	Meaning
0	0	Driver "CopyToRam" inactive
	1	Driver "CopyToRam" active
1	0	Driver "CopyFromRam" inactive
	1	Driver "CopyFromRam" active
2	0	Driver "LoadRamFromFlash" inactive
	1	Driver "LoadRamFromFlash" active
3	0	Driver "SaveRamToFlash" inactive
	1	Driver "SaveRamToFlash" active
4	0	Driver "ClearFlash" inactive
	1	Driver "ClearFlash" active

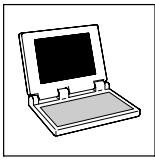


3.3 L_MemGetStateOfMemory

This FB shows you the state of the PLC memory.



Outputs ⇨	(Variable type: VAR_OUTPUT)		
Identifiers	Data type	Value	Meaning
wFlashSectorClear	Word	Bit:	
		0	0 Flash memory segment 1 has not been erased.
			1 Flash memory segment 1 has been erased.
		1	0 Flash memory segment 2 has not been erased.
			1 Flash memory segment 2 has been erased.
		...	0 Flash memory segment 3 ... 14 has not been erased.
			1 Flash memory segment 3 ... 14 has been erased.
		14	0 Flash memory segment 15 has not been erased.
wFlashSectorInUse	Word		1 Flash memory segment 15 has been erased.
		15	Is used by the PLC operating system.
		Bit:	
		0	0 Flash memory segment 1 is currently not written.
			1 Flash memory segment 1 is currently written.
		1	0 Flash memory segment 2 is currently not written.
			1 Flash memory segment 2 is currently written.
		...	0 Flash memory segment 3 ... 14 is currently not written.
byFlashDownloadState	Byte		1 Flash memory segment 3 ... 14 is currently written.
		14	0 Flash memory segment 15 is currently not written.
			1 Flash memory segment 15 is currently written.
byNumOfLoadedBlocks	Byte	1 ... 15	Is used by the PLC operating system.
dwLengthOfLoadedData	Double word	1 ... 983040	Number of downloaded Flash memory segments.
dwStartAddressFlashBlock1	Double word		Number of bytes loaded into the Flash memory.
			Start address of Flash memory segment 1



Function library *LenzeMemDrv.lib*

General memory functions

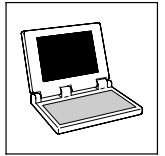
L_MemGetStateOfMemory

Outputs ⇨	(Variable type: VAR_OUTPUT)		
Identifiers	Data type	Value	Meaning
dwStartAddress RamBlock1	Double word		Start address of RAM memory block 1
dwStartAddress RamBlock2	Double word		Start address of RAM memory block 2
byRamBlock1InUse	Byte	0	RAM memory block 1 is currently not written.
		1	RAM memory block 1 is currently written.
byRamBlock2InUse	Byte	0	RAM memory block 2 is currently not written.
		1	RAM memory block 2 is currently written.
wMemDriverVersion	Word		Version of the driver used for memory access The value of this constant is hex-coded. "16#0130" stands, for example, for "Version 1.30".



Tip!

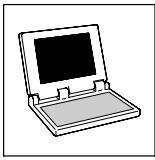
The state of the memory is continuously updated by the operating system. It changes, for instance, when the function **L_MemCopyToRam** is called.



4 Functions for RAM memory access

4.1 Overview

	9300 Servo PLC	Drive PLC	ECSxA	
L_MemCopyFromRam	✓	✓	✓	 4-2
L_MemCopyToRam	✓	✓	✓	 4-4
L_MemReadDINT	✓	✓	✓	 4-6
L_MemReadDWORD	✓	✓	✓	 4-7
L_MemReadINT	✓	✓	✓	 4-8
L_MemReadWORD	✓	✓	✓	 4-9
L_MemWriteDINT	✓	✓	✓	 4-10
L_MemWriteDWORD	✓	✓	✓	 4-11
L_MemWriteWORD	✓	✓	✓	 4-13
L_MemWriteINT	✓	✓	✓	 4-12



Function library *LenzeMemDrv.lib*

Functions for RAM memory access

L_MemCopyFromRam

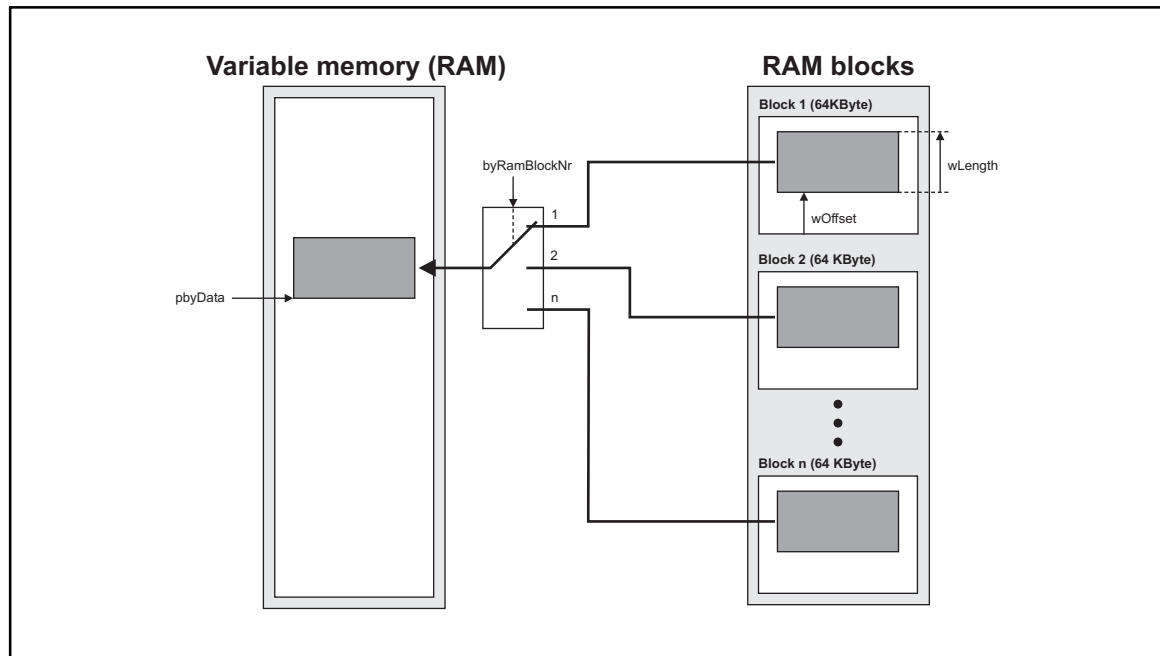
4.2

L_MemCopyFromRam

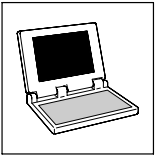
INT. **L_MemCopyFromRam**(byRamBlockNr, wOffset, pbyData, wLength)

This function can be used to copy data from a RAM block of the PLC byte by byte to the variable memory of the PLC.

- The function is processed at the same time as the PLC program in the system task. This is why the processing time depends on the workload of the system.
- You can use the function **L_MemGetStateDriver** to find out if copying has been completed.



Transfer parameters		
Identifiers	Data type	Info/possible settings
byRamBlockNr	Byte	Selecting the RAM memory block including the bytes to be copied
		1 RAM memory 1, size: 64 kbytes
		2 RAM memory 2, size: 64 kbytes
wOffset	Word	Start address in the RAM memory block for the data bytes to be copied
		The start address must not be higher than 65535 - <i>wLength</i>.
pbyData	Pointer to byte	Pointer to the destination address in the variable memory of the PLC where the data bytes are to be copied.
wLength	Word	1 ... 8192 Number of data bytes to be copied



Return value: Integer	
Value	Meaning
0	Copy job has been successfully transferred to the operating system.
-10	Copy job has not been transferred to the operating system since the transfer parameters are faulty: <i>wLength</i> = 0.
-20	Copy job has not been transferred to the operating system since the transfer parameters are faulty: <i>pbyData</i> does not point to the PLC memory area.
-30	Copy job has not been transferred to the operating system since the transfer parameters are faulty: The values for <i>wLength</i> and <i>wOffset</i> do not fit together.
-40	Copy job has not been transferred to the operating system since the copy driver is still processing the previous memory management function.
-50	Copy job has not been transferred to the operating system since the memory is presently accessed (e. g. via codes).
-60	Copy job has not been transferred to the operating system since the transfer parameters are faulty: <i>byRamBlockNr</i> is invalid.
-80	Function is not supported by the operating system.

Example

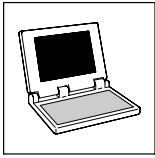
Calling the function in ST:

```
(* copy from RAM if last copy action (L_MemCopyFromRam) *)
(* is finished *)

nMemState := L_MemGetStateDriver(byVoid := 0);
bDriverBusy := INT_TO_BOOL(nMemState AND 16#0002);

IF bDriverBusy = FALSE THEN
    nReturn := L_MemCopyFromRam(byRamBlockNr := 2,
    wOffset := 512,
    pbyData := ADR(g_abyDataBuffer),
    wLength := 256);
END_IF

(* check return value of CopyFromRam function *)
```



Function library *LenzeMemDrv.lib*

Functions for RAM memory access

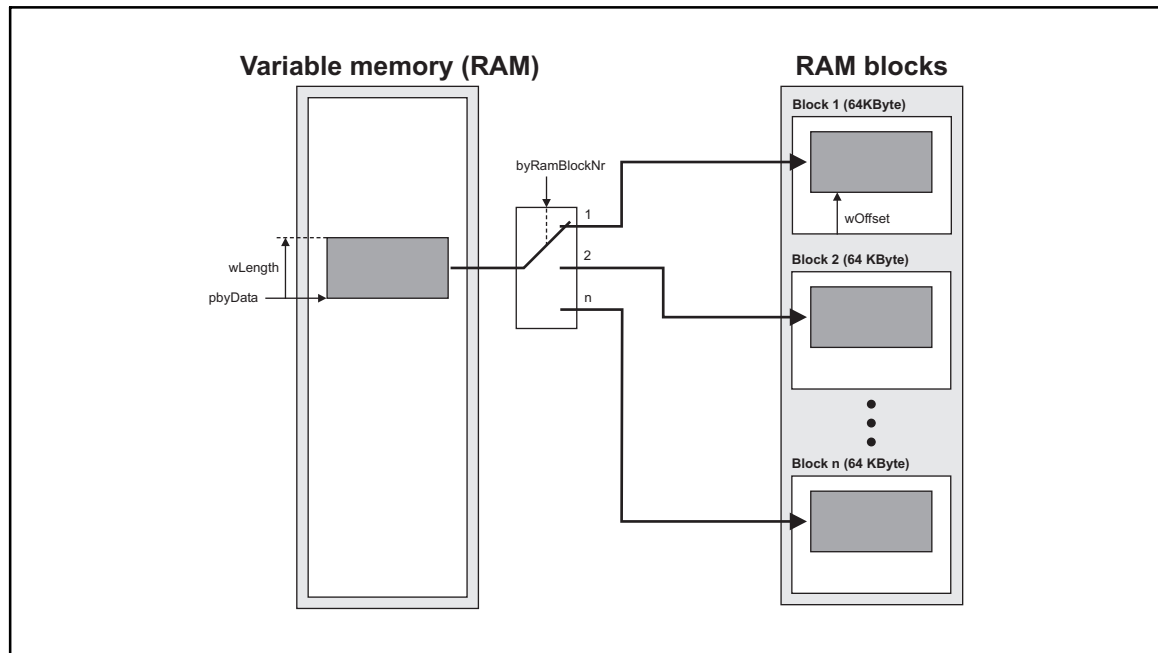
L_MemCopyToRam

4.3 L_MemCopyToRam

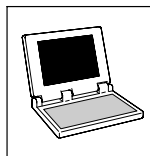
INT. **L_MemCopyToRam**(pbyData, wLength, byRamBlockNr, wOffset)

This function can be used to copy data from the variable memory of the PLC byte by byte to a RAM block of the PLC.

- The function is processed at the same time as the PLC program in the system task. This is why the processing time depends on the workload of the system.
- You can use the function **L_MemGetStateDriver** to find out if copying has been completed.



Transfer parameters		
Identifiers	Data type	Info/possible settings
pbyData	Pointer to byte	Pointer to the start address in the variable memory of the PLC from which on the data bytes to be copied can be found.
wLength	Word	1 ... 8192 Number of data bytes to be copied
byRamBlockNr	Byte	Selecting the RAM memory block to which the data bytes are to be copied
		1 RAM memory 1, size: 64 kbytes
		2 RAM memory 2, size: 64 kbytes
wOffset	Word	Destination address in the selected RAM memory block representing the start address for storing the data bytes to be copied.
		• The destination address must not be higher than 65535 - wLength. 0 ... 65535 Destination address (offset)



Return value: Integer	
Value	Meaning
0	Copy job has been successfully transferred to the operating system.
-10	Copy job has not been transferred to the operating system since the transfer parameters are faulty: <i>wLength</i> = 0.
-20	Copy job has not been transferred to the operating system since the transfer parameters are faulty: <i>pbyData</i> does not point to the PLC memory area.
-30	Copy job has not been transferred to the operating system since the transfer parameters are faulty: The values for <i>wLength</i> and <i>wOffset</i> do not fit together.
-40	Copy job has not been transferred to the operating system since the copy driver is still processing the previous memory management function.
-50	Copy job has not been transferred to the operating system since the memory is presently accessed (e. g. via codes).
-60	Copy job has not been transferred to the operating system since the transfer parameters are faulty: <i>byRamBlockNr</i> is invalid.
-80	Function is not supported by the operating system.

Example

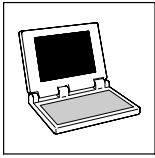
Calling the function in ST:

```
(* copy to RAM if last copy action (L_MemCopyToRam) *)
(* is finished *)

nMemState := L_MemGetStateDriver(byVoid := 0);
bDriverBusy := INT_TO_BOOL(nMemState AND 16#0001);

IF bDriverBusy = FALSE THEN
    nReturn := L_MemCopyToRam(pbyData := ADR(g_abyDataBuffer),
        wLength := 256,
        byRamBlockNr := 2,
        wOffset := 512);
    END_IF

(* check return value of CopyToRam function *)
```



Function library LenzeMemDrv.lib

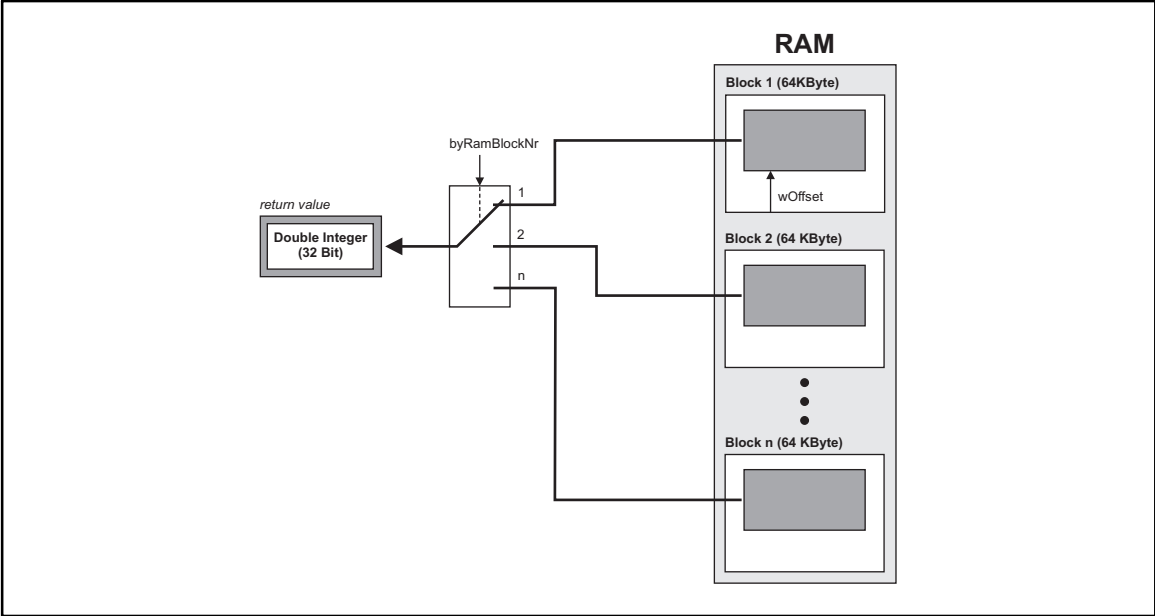
Functions for RAM memory access

L_MemReadDINT

4.4 L_MemReadDINT

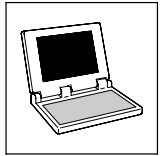
DINT **L_MemReadDINT**(byRamBlockNr, wOffset)

This function can be used to read a 32-bit value of type "double integer" from a RAM block of the PLC.



Transfer parameters		
Identifiers	Data type	Info/possible settings
byRamBlockNr	Byte	Selecting the RAM memory block from which the 32-bit value is to be read.
		1 RAM memory 1, size: 64 kbytes
		2 RAM memory 2, size: 64 kbytes
wOffset	Word	Start address within the selected RAM memory block from which the 32-bit value is to be read.
		0 ... 65532 Start address (offset)

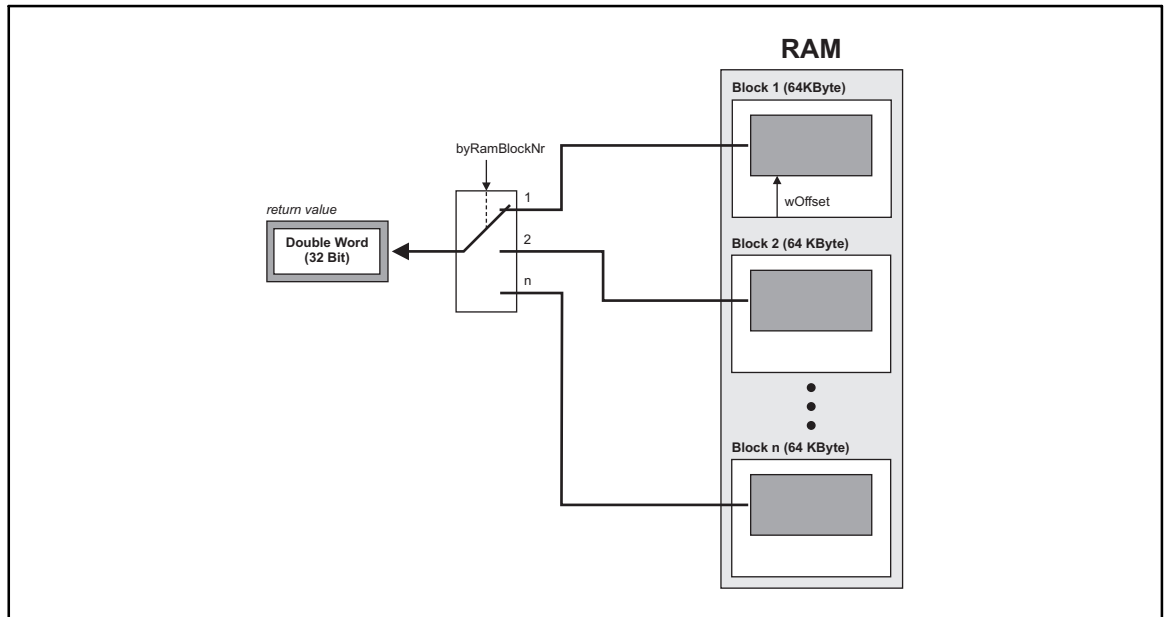
Return value: Double integer	
Value	Meaning
Data	32-bit value read in the RAM memory block <i>byRamBlockNr</i> from position <i>wOffset</i> .
ZERO	Read request has not been transferred to the operating system since the transfer parameters are faulty: <i>byRamBlockNr</i> is invalid.



4.5 L_MemReadDWORD

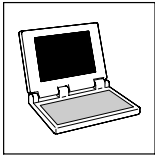
DWORD **L_MemReadDWORD** (byRamBlockNr, wOffset)

This function can be used to read a 32-bit value of type "double word" from a RAM block of the PLC.



Transfer parameters		
Identifiers	Data type	Info/possible settings
byRamBlockNr	Byte	Selecting the RAM memory block from which the 32-bit value is to be read.
		1 RAM memory 1, size: 64 kbytes
		2 RAM memory 2, size: 64 kbytes
wOffset	Word	Start address within the selected RAM memory block from which the 32-bit value is to be read.
		0 ... 65532 Start address (offset)

Return value: Double word	
Value	Meaning
Data	32-bit value read in the RAM memory block <i>byRamBlockNr</i> from position <i>wOffset</i> .
ZERO	Read request has not been transferred to the operating system since the transfer parameters are faulty: <i>byRamBlockNr</i> is invalid.



Function library *LenzeMemDrv.lib*

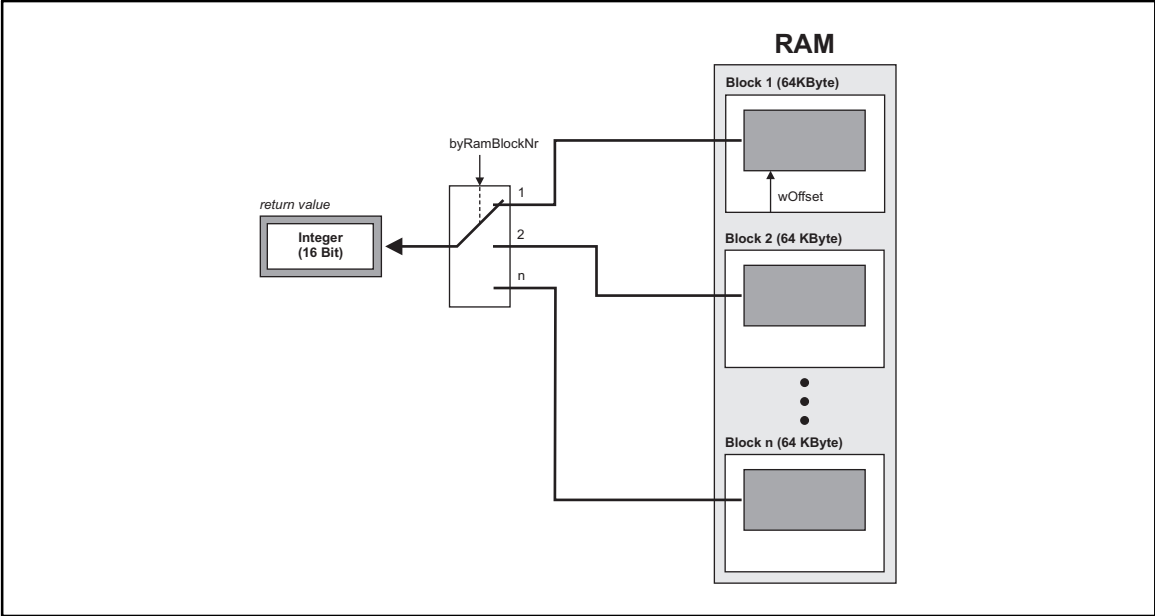
Functions for RAM memory access

L_MemReadINT

4.6 L_MemReadINT

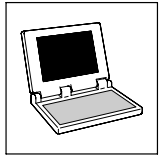
INT. **L_MemReadINT**(byRamBlockNr, wOffset)

This function can be used to read a 16-bit value of type "integer" from a RAM block of the PLC.



Transfer parameters		
Identifiers	Data type	Info/possible settings
byRamBlockNr	Byte	Selecting the RAM memory block from which the 16-bit value is to be read.
		1 RAM memory 1, size: 64 kbytes
		2 RAM memory 2, size: 64 kbytes
wOffset	Word	Start address within the selected RAM memory block from which the 16-bit value is to be read.
		0 ... 65534 Start address (offset)

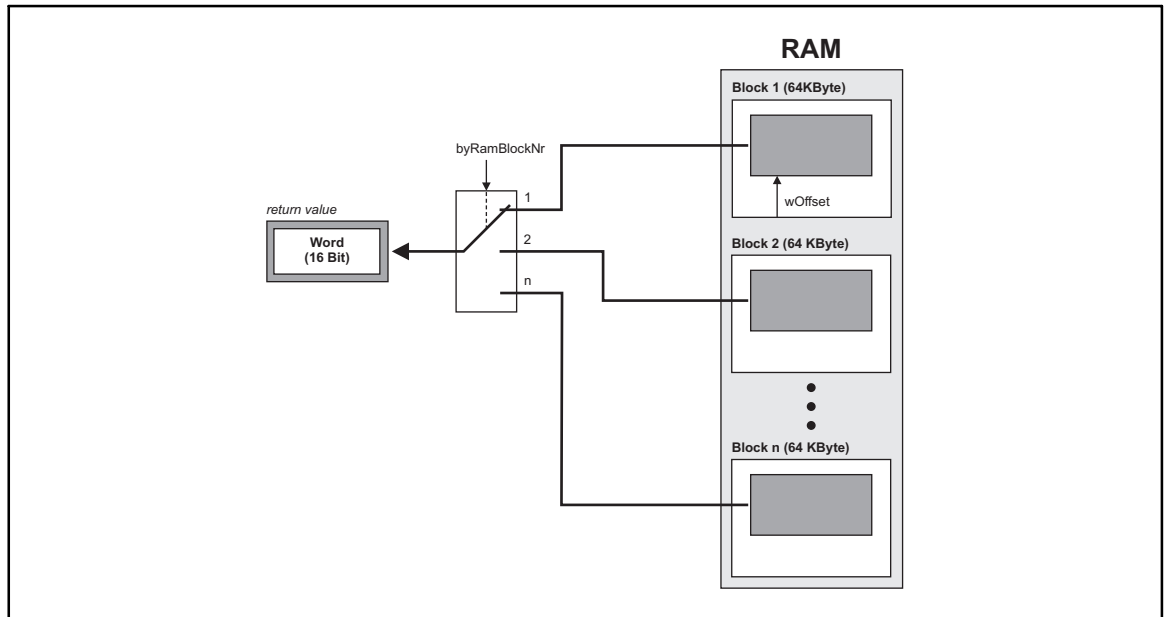
Return value: Integer	
Value	Meaning
Data	16-bit value read in the RAM memory block <i>byRamBlockNr</i> from position <i>wOffset</i> .
ZERO	Read request has not been transferred to the operating system since the transfer parameters are faulty: <i>byRamBlockNr</i> is invalid.



4.7 L_MemReadWORD

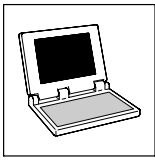
WORD **L_MemReadWORD** (byRamBlockNr, wOffset)

This function can be used to read a 16-bit value of type "word" from a RAM block of the PLC.



Transfer parameters		
Identifiers	Data type	Info/possible settings
byRamBlockNr	Byte	Selecting the RAM memory block from which the 16-bit value is to be read.
		1 RAM memory 1, size: 64 kbytes
		2 RAM memory 2, size: 64 kbytes
wOffset	Word	Start address within the selected RAM memory block from which the 16-bit value is to be read.
		0 ... 65534 Start address (offset)

Return value: Word	
Value	Meaning
Data	16-bit value read in the RAM memory block <i>byRamBlockNr</i> from position <i>wOffset</i> .
ZERO	Read request has not been transferred to the operating system since the transfer parameters are faulty: <i>byRamBlockNr</i> is invalid.



Function library LenzeMemDrv.lib

Functions for RAM memory access

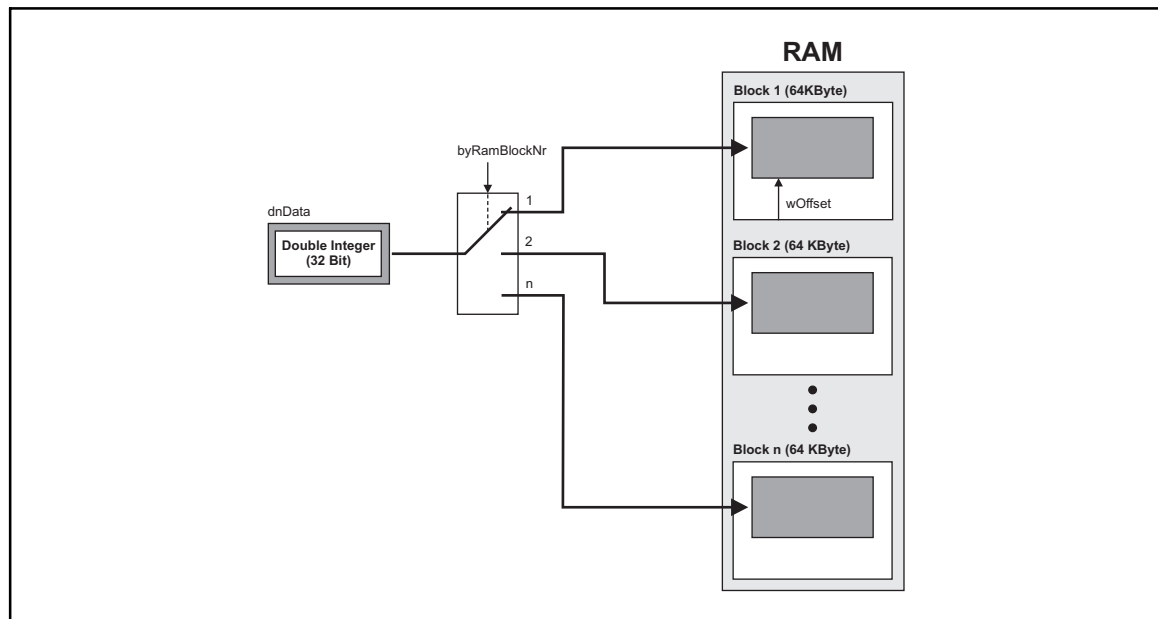
L_MemWriteDINT

4.8

L_MemWriteDINT

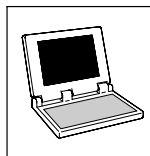
INT. **L_MemWriteDINT** (dnData, byRamBlockNr, wOffset)

This function can be used to write the value of a variable of type "double integer" into a RAM block of the PLC.



Transfer parameters		
Identifiers	Data type	Info/possible settings
dnData	Double integer	Data value to be written (32 bits).
byRamBlockNr	Byte	Selecting the RAM memory block into which <i>dnData</i> is to be written.
		1 RAM memory 1, size: 64 kbytes
		2 RAM memory 2, size: 64 kbytes
wOffset	Word	Destination address in the selected RAM memory block to which <i>dnData</i> is to be written.
		0 ... 65532 Destination address (offset) • Only even values are permissible as offset!

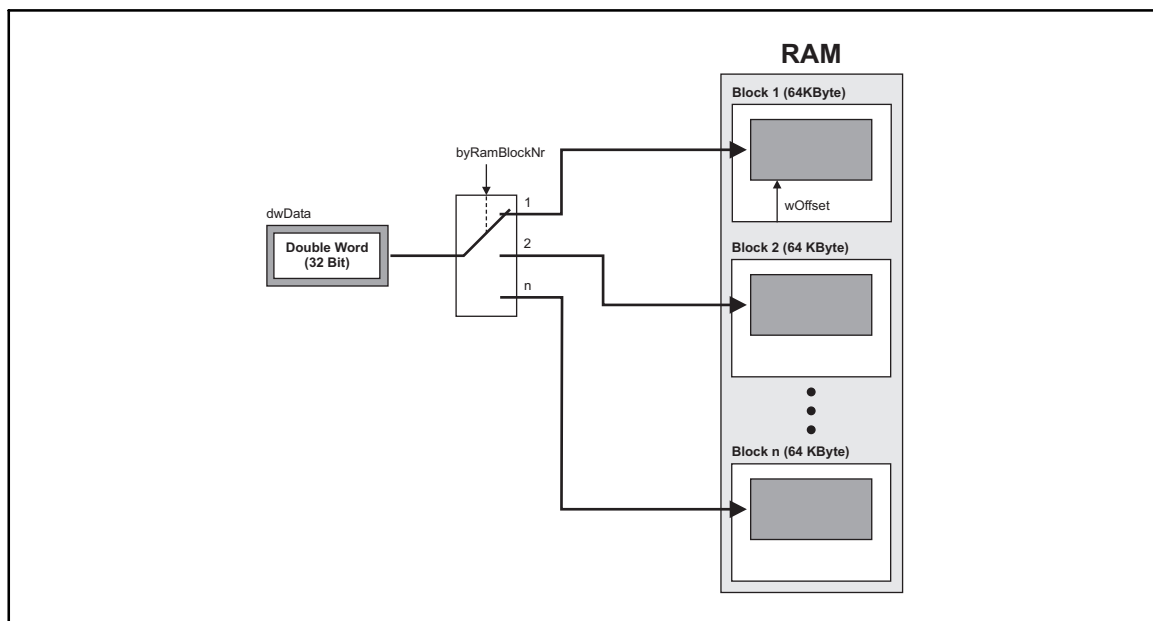
Return value: Integer	
Value	Meaning
0	Write request completed.
-60	Write request has not been transferred to the operating system since the transfer parameters are faulty: <i>byRamBlockNr</i> is invalid.
-35	Copy job has not been transferred to the operating system since the transfer parameters are faulty: <i>wOffset</i> is an odd address or a value beyond the permissible range (0 ... 65532).
-50	Copy job has not been transferred to the operating system since the memory is presently accessed (e. g. via codes).



4.9 L_MemWriteDWORD

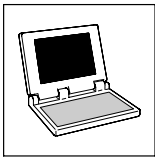
INT. **L_MemWriteDWORD** (dwData, byRamBlockNr, wOffset)

This function can be used to write the value of a variable of type "double word" into a RAM block of the PLC.



Transfer parameters		
Identifiers	Data type	Info/possible settings
dwData	Double word	Data value to be written (32 bits).
byRamBlockNr	Byte	Selecting the RAM memory block into which <i>dwData</i> is to be written.
		1 RAM memory 1, size: 64 kbytes
		2 RAM memory 2, size: 64 kbytes
wOffset	Word	Destination address in the selected RAM memory block to which <i>dwData</i> is to be written.
		0 ... 65532 Destination address (offset) • Only even values are permissible as offset!

Return value: Integer	
Value	Meaning
0	Write request completed.
-60	Write request has not been transferred to the operating system since the transfer parameters are faulty: <i>byRamBlockNr</i> is invalid.
-35	Copy job has not been transferred to the operating system since the transfer parameters are faulty: <i>wOffset</i> is an odd address or a value beyond the permissible range (0 ... 65532).
-50	Copy job has not been transferred to the operating system since the memory is presently accessed (e. g. via codes).



Function library *LenzeMemDrv.lib*

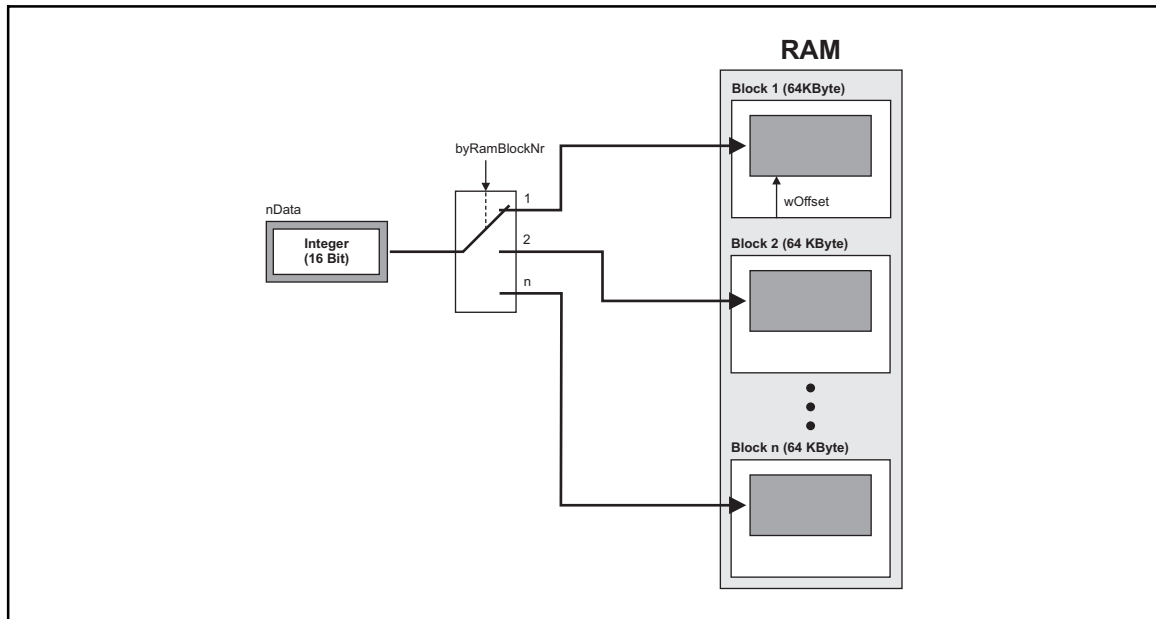
Functions for RAM memory access

L_MemWriteINT

4.10 L_MemWriteINT

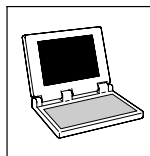
INT. **L_MemWriteINT**(nData, byRamBlockNr, wOffset)

This function can be used to write the value of a variable of type "integer" into a RAM block of the PLC.



Transfer parameters		
Identifiers	Data type	Info/possible settings
nData	Integer	Data value to be written (16 bits).
byRamBlockNr	Byte	Selecting the RAM memory block into which <i>nData</i> is to be written.
		1 RAM memory 1, size: 64 kbytes
		2 RAM memory 2, size: 64 kbytes
wOffset	Word	Destination address in the selected RAM memory block to which <i>nData</i> is to be written.
		0 ... 65534 Destination address (offset) • Only even values are permissible as offset!

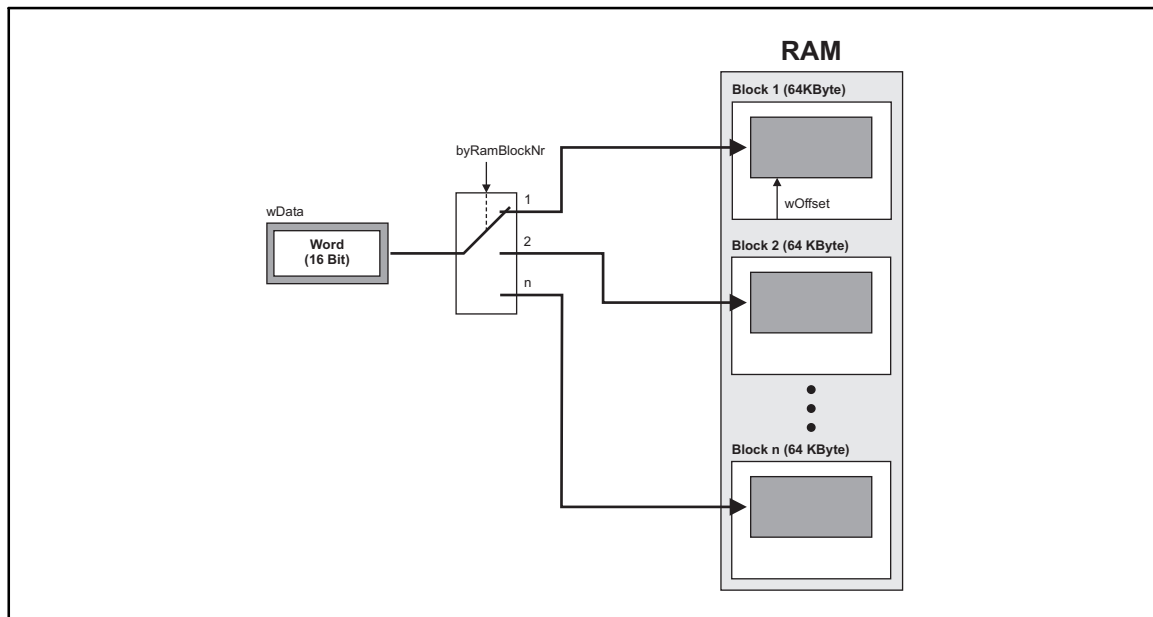
Return value: Integer	
Value	Meaning
0	Write request completed.
-60	Write request has not been transferred to the operating system since the transfer parameters are faulty: <i>byRamBlockNr</i> is invalid.
-35	Copy job has not been transferred to the operating system since the transfer parameters are faulty: <i>wOffset</i> is an odd address or a value beyond the permissible range (0 ... 65534).
-50	Copy job has not been transferred to the operating system since the memory is presently accessed (e. g. via codes).



4.11 L_MemWriteWORD

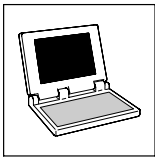
INT. **L_MemWriteWORD**(wData, byRamBlockNr, wOffset)

This function can be used to write the value of a variable of type "word" into a RAM block of the PLC.



Transfer parameters			
Identifiers	Data type	Info/possible settings	
wData	Word	Data value to be written (16 bits).	
byRamBlockNr	Byte	Selecting the RAM memory block into which <i>wData</i> is to be written.	
		1	RAM memory 1, size: 64 kbytes
		2	RAM memory 2, size: 64 kbytes
wOffset	Word	Destination address in the selected RAM memory block to which <i>wData</i> is to be written.	
		0 ... 65534	Destination address (offset) • Only even values are permissible as offset!

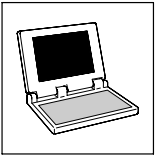
Return value: Integer	
Value	Meaning
0	Write request completed.
-60	Write request has not been transferred to the operating system since the transfer parameters are faulty: <i>byRamBlockNr</i> is invalid.
-35	Copy job has not been transferred to the operating system since the transfer parameters are faulty: <i>wOffset</i> is an odd address or a value beyond the permissible range (0 ... 65534).
-50	Copy job has not been transferred to the operating system since the memory is presently accessed (e. g. via codes).



Function library LenzeMemDrv.lib

Functions for RAM memory access

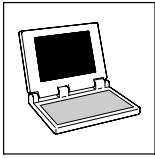
L_MemWriteWORD



5 Functions for FLASH memory access

5.1 Overview

	9300 Servo PLC	Drive PLC	ECSxA	
L_MemClearFlash	✓			 5-2
L_MemLoadRamFromFlash	✓			 5-3
L_MemSaveRamToFlash	✓			 5-5



Function library *LenzeMemDrv.lib*

Functions for RAM memory access

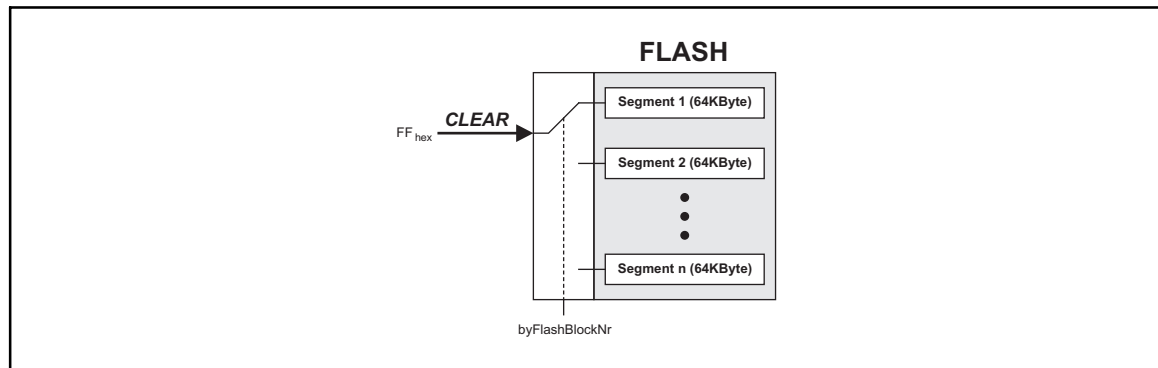
L_MemClearFlash

5.2 L_MemClearFlash

INT. **L_MemClearFlash**(byFlashBlockNr)

This function can be used to erase a memory segment from the Flash memory of the PLC.

- After erasing the selected memory segment, all data in this segment have the value “FF_{hex}”.
- The function is processed at the same time as the PLC program in the system task. This is why the processing time depends on the workload of the system.
- You can use the function **L_MemGetStateDriver** to find out if erasing has been completed.



Transfer parameters		
Identifiers	Data type	Info/possible settings
byFlashBlockNr	Byte	Selecting the Flash memory segment to be erased.
		1 Flash memory segment 1, size: 64 kbytes
		
		15 Flash memory segment 15, size: 64 kbytes

Return value: Integer	
Value	Meaning
0	Erase request has been successfully transferred to the operating system.
-40	Copy job has not been transferred to the operating system since the copy driver is still processing the previous memory management function.
-50	Copy job has not been transferred to the operating system since the memory is presently accessed (e. g. via codes).
-55	Flash write protection has been activated; access denied.
-65	Erase request has not been transferred to the operating system since the transfer parameters are faulty: <i>byFlashBlockNr</i> is invalid.
-80	Function is not supported by the operating system.

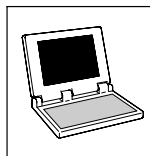
Example

Calling the function in ST:

```
(* clear FLASH segment 2 *)

nReturn := L_MemClearFlash(byFlashBlockNr := 2);

(* check function return value *)
```

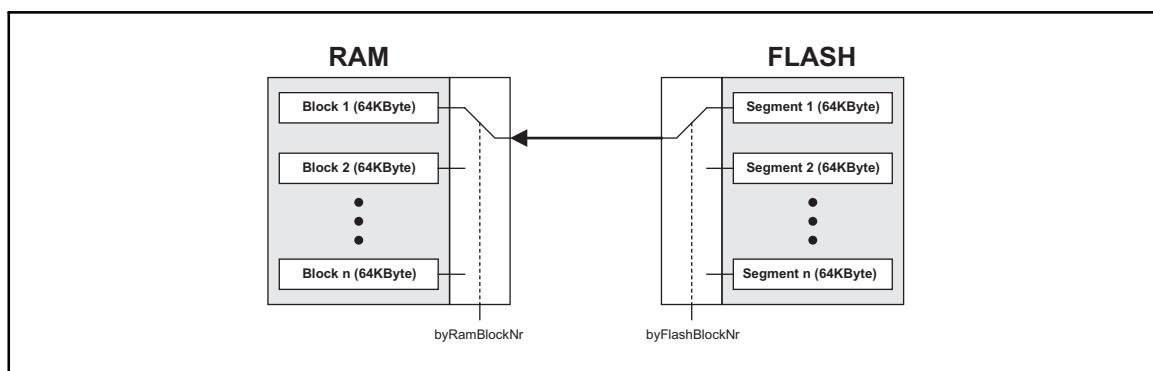


5.3 L_MemLoadRamFromFlash

INT. **L_MemLoadRamFromFlash** (byFlashBlockNr, byRamBlockNr)

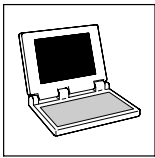
This function can be used to load (permanently saved) data of a whole Flash memory segment of the PLC into a RAM block of the PLC.

- The function is processed at the same time as the PLC program in the system task. This is why the processing time depends on the workload of the system.
- You can use the function **L_MemGetStateDriver** to find out if loading has been completed.



Transfer parameters		
Identifiers	Data type	Info/possible settings
byFlashBlockNr	Byte	Selecting the Flash memory segment including data to be loaded.
		1 Flash memory segment 1, size: 64 kbytes
		
		15 Flash memory segment 15, size: 64 kbytes
byRamBlockNr	Byte	Selecting the RAM memory block into which the data of the selected Flash memory segment are to be loaded.
		1 RAM memory 1, size: 64 kbytes
		2 RAM memory 2, size: 64 kbytes

Return value: Integer	
Value	Meaning
0	Copy job has been successfully transferred to the operating system.
-40	Copy job has not been transferred to the operating system since the copy driver is still processing the previous memory management function.
-50	Copy job has not been transferred to the operating system since the memory is presently accessed (e. g. via codes).
-60	Copy job has not been transferred to the operating system since the transfer parameters are faulty: <i>byRamBlockNr</i> is invalid.
-65	Copy job has not been transferred to the operating system since the transfer parameters are faulty: <i>byFlashBlockNr</i> is invalid.
-80	Function is not supported by the operating system.



Function library LenzeMemDrv.lib

Functions for RAM memory access

L_MemLoadRamFromFlash

Example

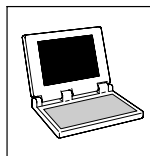
Calling the function in ST:

```
(* load from FLASH if last copy action (L_MemLoadRamFromFlash) *)
(* is finished *)

nMemState := L_MemGetStateDriver(byVoid := 0);
bDriverBusy := INT_TO_BOOL(nMemState AND 16#0004);

IF bDriverBusy = FALSE THEN
  nReturn := L_MemLoadRamFromFlash(byFlashBlockNr := 2
    byRamBlockNr := 1);
END_IF

(* check function return value *)
```

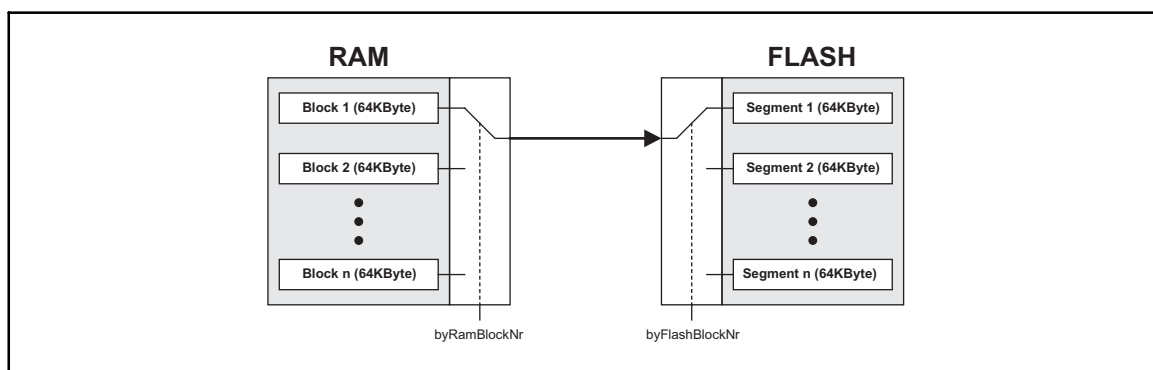


5.4 L_MemSaveRamToFlash

INT. **L_MemSaveRamToFlash** (byRamBlockNr, byFlashBlockNr)

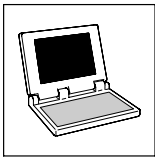
This function can be used to save the data of a whole RAM block of the PLC permanently in a Flash memory segment of the PLC.

- Before saving data, the function erases the selected Flash memory segment (overwrites it with FF hex).
- The function is processed at the same time as the PLC program in the system task. This is why the processing time depends on the workload of the system.
- You can use the function **L_MemGetStateDriver** to find out if data saving has been completed.



Transfer parameters		
Identifiers	Data type	Info/possible settings
byRamBlockNr	Byte	Selecting the RAM memory block to be saved
		1 RAM memory 1, size: 64 kbytes
		2 RAM memory 2, size: 64 kbytes
byFlashBlockNr	Byte	Selecting the Flash memory segment in which the data of the selected RAM block are to be permanently saved.
		1 Flash memory segment 1, size: 64 kbytes
		
		15 Flash memory segment 15, size: 64 kbytes

Return value: Integer	
Value	Meaning
0	Copy job has been successfully transferred to the operating system.
-40	Copy job has not been transferred to the operating system since the copy driver is still processing the previous memory management function.
-50	Copy job has not been transferred to the operating system since the memory is presently accessed (e. g. via codes).
-55	Flash write protection has been activated; access denied.
-60	Copy job has not been transferred to the operating system since the transfer parameters are faulty: <i>byRamBlockNr</i> is invalid.
-65	Copy job has not been transferred to the operating system since the transfer parameters are faulty: <i>byFlashBlockNr</i> is invalid.
-80	Function is not supported by the operating system.



Function library *LenzeMemDrv.lib*

Functions for RAM memory access

L_MemSaveRamToFlash

Example

Calling the function in ST:

```
(* save to FLASH if last copy action (L_MemSaveRamToFlash) *)
(* is finished *)

nMemState := L_MemGetStateDriver(byVoid := 0);
bDriverBusy := INT_TO_BOOL(nMemState AND 16#0008);

IF bDriverBusy = FALSE THEN
  nReturn := L_MemSaveRamToFlash(byRamBlockNr := 1
    byFlashBlockNr := 2);
END_IF

(* check function return value *)
```