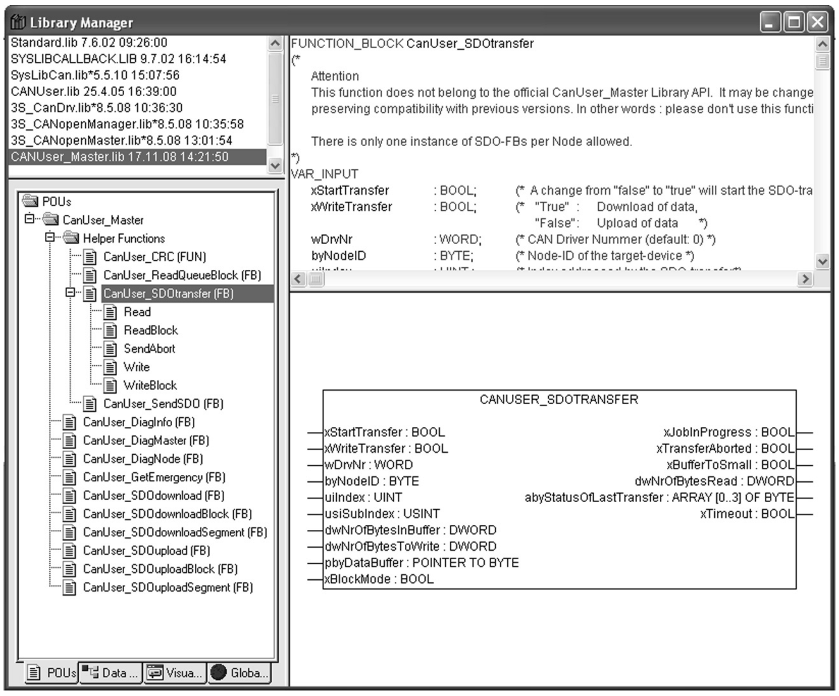
User Manual

10/10 MN05010001Z-EN
replaces 08/04 AWB2786-1554GB

CanUser.lib
CanUser_Master.lib



```
Standard.lib 7.6.02 09:26:00
SYSLIBCALLBACK.LIB 9.7.02 16:14:54
SysLibCan.lib*5.5.10 15:07:56
CANUser.lib 25.4.05 16:39:00
3S_CanDrv.lib*8.5.08 10:36:30
3S_CANopenManager.lib*8.5.08 10:35:58
3S_CANopenMaster.lib*8.5.08 13:01:54
CANUser_Master.lib 17.11.08 14:21:50
```

```
POUs
  CanUser_Master
    Helper Functions
      CanUser_CRC (FUN)
      CanUser_ReadQueueBlock (FB)
      CanUser_SDOtransfer (FB)
        Read
        ReadBlock
        SendAbort
        Write
        WriteBlock
      CanUser_SendSDO (FB)
    CanUser_DiagInfo (FB)
    CanUser_DiagMaster (FB)
    CanUser_DiagNode (FB)
    CanUser_GetEmergency (FB)
    CanUser_SDOdownload (FB)
    CanUser_SDOdownloadBlock (FB)
    CanUser_SDOdownloadSegment (FB)
    CanUser_SDOupload (FB)
    CanUser_SDOuploadBlock (FB)
    CanUser_SDOuploadSegment (FB)
```

```
FUNCTION_BLOCK CanUser_SDOtransfer
(*
   Attention
   This function does not belong to the official CanUser_Master Library API.  It may be change
   preserving compatibility with previous versions. In other words : please don't use this functi

   There is only one instance of SDO-FBs per Node allowed.
*)
VAR_INPUT
   xStartTransfer      : BOOL;      (*  A change from "false" to "true" will start the SDO-tra
   xWriteTransfer      : BOOL;      (*   "True"  :    Download of data,
                                         "False":     Upload of data      *)
   wDrvNr              : WORD;      (* CAN Driver Nummer (default: 0) *)
   byNodeID            : BYTE;      (* Node-ID of the target-device *)
```

```
                          CANUSER_SDOTRANSFER
xStartTransfer : BOOL                          xJobInProgress : BOOL
xWriteTransfer : BOOL                        xTransferAborted : BOOL
wDrvNr : WORD                                  xBufferToSmall : BOOL
byNodeID : BYTE                            dwNrOfBytesRead : DWORD
uiIndex : UINT         abyStatusOfLastTransfer : ARRAY [0..3] OF BYTE
usiSubIndex : USINT                                  xTimeout : BOOL
dwNrOfBytesInBuffer : DWORD
dwNrOfBytesToWrite : DWORD
pbyDataBuffer : POINTER TO BYTE
xBlockMode : BOOL
```

Powering Business Worldwide

Emergency On Call Service
Please call your local representative:
http://www.eaton.com/moeller/aftersales
or
Hotline After Sales Service:
+49 (0) 180 5 223822 (de, en)
AfterSalesEGBonn@eaton.com

Original Operating Instructions
The German-language edition of this document is the original operating manual.

Translation of the original operating manual
All editions of this document other than those in German language are translations of the original German manual.

1st published 2004, edition date 08/04
2nd edition 2010, edition date 10/10

Author: Andreas Lüngen
Production: Thomas Kracht
Translator: David Long

Contents

1. General/overview

1.1 Using the libraries

The CanUser.lib and CanUser_Master.lib libraries provide the user with cross-control access to CAN objects. These include in particular CAN direct functions/function blocks such as direct read and write of CAN telegrams and further CANopen functionality's such as sending and receiving data via SDO functions, or access to diagnostics information from the user program.

The basic functions of the CanUser.lib and the CanUser_Master.lib are available in the following controls:

- XC100
- XC200
- XC600
- XVC100
- HPG200
- HPG300
- XVC600

The respective libraries provide the same interfaces to the outside (to the user side), however the internal structure can be different. Even though a CanUser.lib for a HPG200 has a different internal structure to a CanUser.lib for a XC200 PLC, both libraries provide the same function interfaces on the user side. It is easier to transfer the user code between ports.

In addition to the standard functions, some libraries may contain control-dependant functions. Functions/function blocks of this nature are specially marked.

1.2 Overview

The following functions/function blocks are described in this document:

1.2.1 CanUser.lib

	Function/function block	XC100	XC200	XC600	XVC100	HPG200	HPG300	XVC600
CanUser_DiagInfo	FB	--	--	--	√	√	√	√
CanUser_GetVersions	FB	√	√	√	√	√	√	√
CanUser_ReadImage	FB	√ !	√ !	√ !	√	√	√	√
CanUser_ReadQueue	FB	√ !	√ !	√ !	√	√	√	√
CanUser_Write	F	√	√	√	√	√	√	√

√	=	The function is supported.
√*	=	The function is supported with limitations.
--	=	The function is not supported.
!	=	Refer to section 1.3 for further explanations.

1.2.2 CanUser_Master.lib

	Function/function block	XC100	XC200	XC600	XVC100	HPG200	HPG300	XVC600
CanUser_DiagInfo	FB	√*	√*	--	--	--	--	--
CanUser_DiagMaster	FB	√*	√*	--	√	√	√	√
CanUser_DiagNode	FB	√	√	√ !	√	√	√	√
CanUser_GetEmergency	FB	√	√	√	√	√	√	√
CanUser_SDOdownload	FB	√ !	√ !	√	√	√	√	√
CanUser_SDOdownloadBlock	FB	√ !	√ !	--	-- !	-- !	-- !	-- !
CanUser_SDOdownloadSegment	FB	√ !	√ !	√* !	√ !	√ !	√ !	√ !
CanUser_SDOupload	FB	√ !	√ !	√	√	√	√	√
CanUser_SDOuploadBlock	FB	√ !	√ !	--	-- !	-- !	-- !	-- !
CanUser_SDOuploadSegment	FB	√ !	√ !	√* !	√ !	√ !	√ !	√ !

√	=	The function is supported.
√*	=	The function is supported with limitations.
--	=	The function is not supported.
!	=	Refer to section 1.3 for further explanations.

1.3 Deviations with implementation

In this section the functional differences of individual library functions on the various controls are listed insofar as they affect the user.

1.3.1 XC100

<u>**CanUser.lib - functions**</u>

- If the control configuration of the PLC does not contain a CAN master, the baud rate can be set via the SysCanControl() function from the XC100_SysLibCan library. If a baud rate is not set, the CAN controller operates with the last baud rate setting. This should be avoided.

- After integration of the CanUser.lib library, the following file should be into the user project via the "Project/Import" button: **CanUser.exp**

  The file adds the global constants
  CanUser_DISPATCH_ARRAY_MAX_SIZE : INT := 16; to the user project. These constants <u>limit the number of available instances</u> of the CanUser_ReadImage and CanUser_ReadQueue function blocks.

- The sum of the instances of **CanUser_ReadImage** and **CanUser_ReadQueue** must be less than CanUser_DISPATCH_ARRAY_MAX_SIZE.

  The variable value for CanUser_DISPATCH_ARRAY_MAX_SIZE can be extended if necessary.

- The **CanUser_DiagInfo** function block is implemented for the XC100 control in the CanUser_Master.lib. Not all output-variables are supported by the XC100.

<u>**CanUser_Master.lib - functions**</u>

- When CanUser_Master functions are used, the CAN master control configuration which is used to make baud rate settings must be added.

- After the CanUser_Master.lib library has been integrated, the following file must be integrated into the user project: **CanUserMaster.exp**

  The file adds the following global constants to the user project:

  CanUser_SDO_TRANSFER_POOL_SIZE : INT := 3;

  These constants <u>limit the number of instances</u> of CanUser_SDOxxx functions which can be active <u>simultaneously</u>.

  CanUser_SDO_TIMEOUT : TIME := t#500 ms;

  These constants define the monitoring time for an SDO transfer. If the CAN slave which has been addressed has not responded by the time the monitoring time has timed out, the SDO transfer is aborted with an error message.

- A maximum of CanUser_SDO_TRANSFER_POOL_SIZE instances from SDO function blocks can be active simultaneously. Every other instance which tries to initiate an SDO transfer will be inhibited.
  The variable value for CanUser_SDO_TRANSFER_POOL_SIZE can be extended if necessary.

1.3.2 XC200

<u>CanUser.lib - functions</u>

- If the control configuration of the PLC does not contain a CAN master, the baud rate can be set via the SysCanControl() function from the XC200_SysLibCan library.

- After integration of the CanUser.lib library, the following file should be into the user project via the "Project/Import" button: **CanUser.exp**
  The file adds the global constants
  CanUser_DISPATCH_ARRAY_MAX_SIZE : INT := 16; to the user project. These constants <u>limit the number of available instances</u> of the CanUser_ReadImage and CanUser_ReadQueue functions.

- The sum of the instances of **CanUser_ReadImage** and **CanUser_ReadQueue** must be less than CanUser_DISPATCH_ARRAY_MAX_SIZE.
  The variable value for CanUser_DISPATCH_ARRAY_MAX_SIZE can be extended if necessary.

- The **CanUser_DiagInfo** function block is implemented for the XC200 control in the CanUser_Master.lib. Not all output-variables are supported by the XC200.

<u>CanUser_Master.lib - functions</u>

- When CanUser_Master function blocks are used, the CAN master control configuration which is used to make baud rate settings must be added.

- After the CanUser_Master.lib library has been integrated, the following file must be integrated into the user project: **CanUserMaster.exp**
  The file adds the following global constants to the user project:
  CanUser_SDO_TRANSFER_POOL_SIZE : INT := 3;
  These constants <u>limit the number of instances</u> of CanUser_SDOxxx functions which can be active <u>simultaneously</u>.

  CanUser_SDO_TIMEOUT : TIME := t#500 ms;
  These constants define the monitoring time for an SDO transfer. If the CAN slave which has been addressed has not responded by the time the monitoring time has timed out, the SDO transfer is aborted with an error message.

- A maximum of CanUser_SDO_TRANSFER_POOL_SIZE instances from SDO functions can be active simultaneously. Every other instance which tries to initiate an SDO transfer will be inhibited.

  The variable value for CanUser_SDO_TRANSFER_POOL_SIZE can be extended if necessary.

1.3.3 XC600

To use the CanUser.lib and CanUser_Master.lib libraries the following firmware-versions are required on the XC600.

XC600:	2.3.2.4-06	
Can-Card:	V01.071	;Build 09.10.03

CanUser.lib - functions

- When CanUser functions are used, the CAN master control configuration which is used to make baud rate settings must be added.

- The **CanUser_DiagInfo()** function block is not supported in the XC600 control.

- If you are operating with the **CanUser_ReadImage()** or with the **CanUser_ReadQueue()** function block, the user-task must add the

  SortReceiveMessage (dwEvent:=0,dwFilter:=0,dwOwner:=0);

  line at the beginning.

  With the SortReceiveMessage() function call, all the telegrams read by CANopen modules are distributed to the individual registered read instances.

- The sum of the instances of **CanUser_ReadImage** and **CanUser_ReadQueue** must be less than CanUser_g_DISPATCH_ARRAY_MAX_SIZE = 64.

- If the CanUser_Write() function is used, the line:

  SendWriteMessage(dwEvent:=0,dwFilter:=0,dwOwner:=0);

  must be added at the end of the user task.

- CanUser_Write(): The Input-Parameter “ucMode” will be ignored.

CanUser_Master.lib - functions

- The **CanUser_DiagMaster()** function block is not supported in the XC600 control.

- The **CanUser_DiagNode()** function block is extended in the XC600 control by the xReady return value when compared to the standard implementation. This value provides information concerning the current status of the function blocks. A new read diagnostics value is only available when xReady is set. From this point, the return values receive the information state of the last read out.

The function block is only available for CAN stations configured in the XC600.
(For diagnostics of CAN stations via a XC600, the XC600_GetDiagByNode() function block from the CanUser.lib of the XC600 can be used. It supplies detailed diagnostics values. However, this function is only available for CAN stations configured in the XC600 and is not supported by other Moeller controls.)

- The **CanUser_GetEmergency()** function block is only available for CAN stations configured in the XC600 and used by the application.

- The **CanUser_SDOdownloadBlock()** function is not supported in the XC600 control.

- The **CanUser_SDOdownloadSegmented()** function can send a maximum of 247 bytes in the XC600 control.

- The **CanUser_SDOuploadBlock()** function is not supported in the XC600 control.

- Data with a maximum length of 247 bytes can be received via the **CanUser_SDOuploadSegmented()** function in the XC600 control.

1.3.4 XVC100

CanUser.lib - functions

No remarks.

CanUser_Master.lib - functions

- The **CanUser_SDOdownloadBlock()** function block starts a segmented SDO download, no block-by-block download.

- The **CanUser_SDOdownloadSegmented()** function block does not have a function.

- The **CanUser_SDOuploadBlock()** function block starts a segmented SDO upload, no block-by-block upload.

- The **CanUser_SDOuploadSegmented()** function block does not have a function.

1.3.5 HPG200

As XVC100.

1.3.6 HPG300

As XVC100.

1.3.7 XVC600

As XVC100.

1.4 Required libraries

The required libraries which are necessary for use of the CanUser.lib or CanUser_Master.lib on the respective controls are described in this section.

These libraries are automatically added to the project by the program system after adding the CanUser.lib or CanUser_Master.lib.

The libraries to be added can be seen in the following listing:

1.4.1 CanUser.lib

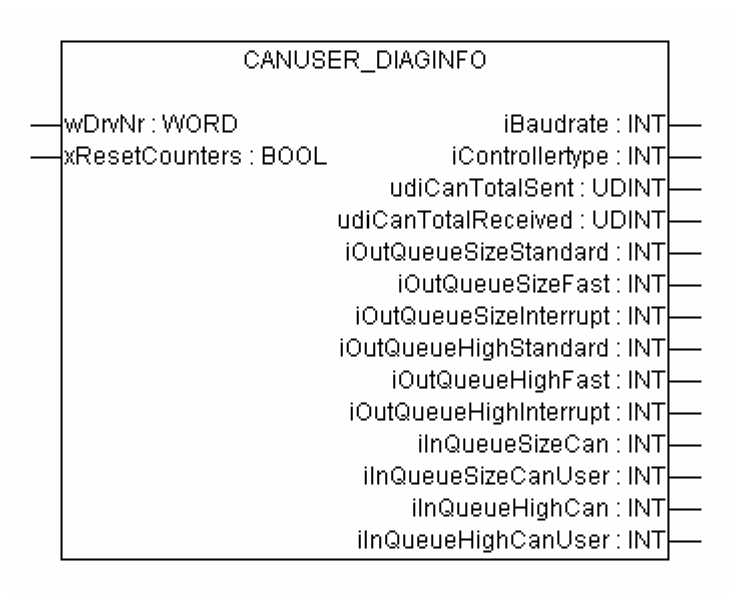
	CanUser.lib	Standard.lib	SysLibCallback.lib	CIF_LIB.lib	XC100_SysLibCan.lib	XC200_SysLibCan.lib	3S_CanDrv.lib	3S_CANopenManager.lib	3S_CANopenMaster.lib
XC100	√	--	--	--	√	--	--	--	--
XC200	√	--	--	--	--	√	--	--	--
XC600	√	--	√	√	--	--	--	--	--
XVC100	√	√	√	--	--	--	√	√	√
HPG200	√	√	√	--	--	--	√	√	√
HPG300	√	√	√	--	--	--	√	√	√
XVC600	√	√	√	--	--	--	√	√	√

1.4.2 CanUser_Master.lib

	CanUser.lib	CanUser_Master.lib	Standard.lib	SysLibCallback.lib	CIF_LIB.lib	XC100_SysLibCan.lib	XC200_SysLibCan.lib	3S_CanDrv.lib	3S_CANopenManager.lib	3S_CANopenMaster.lib
XC100	√	√	√	√	--	√	--	√	√	√
XC200	√	√	√	√	--	--	√	√	√	√
XC600	√	√	--	√	√	--	--	--	--	--
XVC100	√	√	√	√	--	--	--	√	√	√
HPG200	√	√	√	√	--	--	--	√	√	√
HPG300	√	√	√	√	--	--	--	√	√	√
XVC600	√	√	√	√	--	--	--	√	√	√

2 CanUser.lib functions/function blocks

2.1 CanUser_DiagInfo



2.1.1 Description

General information from the CAN driver can be accessed via this function block.

This function block is not contained in the CanUser.lib for the XC100 and XC200 controls, but rather in the CanUser_Master.lib.

This function block is supported by the following controls:

XC100	XC200	XC600	XVC100	HPG200	HPG300	XVC600
√* !	√* !	--	√	√	√	√

√	=	The function block is supported.
√*	=	The function is supported with limitations.
--	=	The function block is not supported.
!	=	Refer to section 1.3 for further explanations.

2.1.2 Inputs of the function blocks

wDrvNr	: CAN driver number. Default setting = “0” XC600→Jumper setting of the CANopen modules to be addressed “0” – “2”
xResetCounters	: TRUE → Resets all counters to “0”. FALSE → Counter values remain unchanged.

2.1.3 Outputs of the function blocks

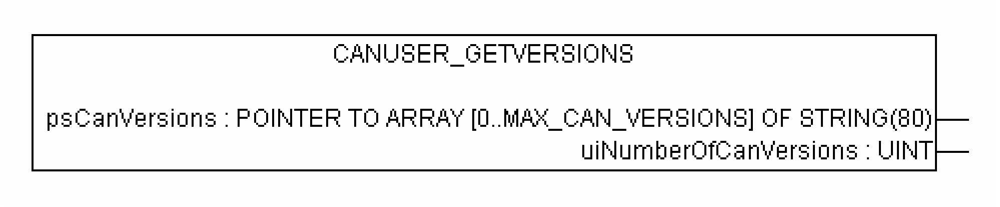
iBaudrate	: CAN baud rate in kBit/s. -1 := Display of this value is not supported by the control.
iControllertype	: CAN controller: 0 → Phillips SJA1000 1 → Intel 82527 2 → Infineon 80C164 integrated CAN controller 3 → Infineon 82C900 -1 → Display of this value is not supported by the control.
udiCanTotalSent	: Total number of all CAN telegrams sent
udiCanTotalReceive	: Total number of all CAN telegrams received
iOutQueueSizeStandard	: Size of the send FIFOs (standard priority.)
iOutQueueSizeFast	: Size of the send FIFOs (high priority.)
iOutQueueSizeInterrupt	: Size of the send FIFOs (interrupt priority.)
iOutQueueHighStandard	: Maximum occupancy of the send FIFOs (standard priority.)
iOutQueueHighFast	: Maximum occupancy of the send FIFOs (high priority.)
iOutQueueHighInterrupt	: Maximum occupancy of the send FIFOs (interrupt priority.)
iInQueueSizeCan	: Size of the CANopen receive FIFOs.
iInQueueSizeCanUser	: Size of the CAN direct receive FIFOs.
iInQueueHighCan	: Maximum occupancy of the CANopen receive FIFOs.
iInQueueHighCanUser	: Maximum occupancy of the CAN direct receive FIFOs.

2.1.4 Example

```
PROGRAM PLC_PRG
VAR
	CAN_DiagInfo : CanUser_DiagInfo;
	iBaudrate: INT;
	iControllertype: INT;
	udiCanTotalSent: UDINT;
	udiCanTotalReceived: UDINT;
	iOutQueueSizeStandard: INT;
	iOutQueueSizeFast: INT;
	iOutQueueSizeInterrupt: INT;
	iOutQueueHighStandard: INT;
	iOutQueueHighFast: INT;
	iOutQueueHighInterrupt: INT;
	iInQueueSizeCan: INT;
	iInQueueSizeCanUser: INT;
	iInQueueHighCan: INT;
	iInQueueHighCanUser: INT;
END_VAR
```

```
CAN_DiagInfo(
        wDrvNr:=0 ,
        xResetCounters:= FALSE,
        iBaudrate=> iBaudrate,
        iControllertype=> iControllertype,
        udiCanTotalSent=>udiCanTotalSent ,
        udiCanTotalReceived=> udiCanTotalReceived,
        iOutQueueSizeStandard=> iOutQueueSizeStandard,
        iOutQueueSizeFast=> iOutQueueSizeFast,
        iOutQueueSizeInterrupt=>iOutQueueSizeInterrupt ,
        iOutQueueHighStandard=>iOutQueueHighStandard ,
        iOutQueueHighFast=>iOutQueueHighFast ,
        iOutQueueHighInterrupt=>iOutQueueHighInterrupt ,
        iInQueueSizeCan=>iInQueueSizeCan ,
        iInQueueSizeCanUser=>iInQueueSizeCanUser ,
        iInQueueHighCan=> iInQueueHighCan,
        iInQueueHighCanUser=> iInQueueHighCanUser);
```

2.2 CanUser_GetVersions


CANUSER_GETVERSIONS
psCanVersions : POINTER TO ARRAY [0..MAX_CAN_VERSIONS] OF STRING(80)
uiNumberOfCanVersions : UINT

2.2.1 Description

A pointer to an array of version strings can be requested via instances of this function block. The version strings contain details concerning the loaded version of the CanUser libraries as well as details of the version of the CAN driver in some controls.

This function block is supported by the following controls:

XC100	XC200	XC600	XVC100	HPG200	HPG300	XVC600
√	√	√	√	√	√	√

√	=	The function block is supported.
√*	=	The function is supported with limitations.
--	=	The function block is not supported.
!	=	Refer to section 1.3 for further explanations.

2.2.2 Inputs of the function blocks

None.

2.2.3 Outputs of the function blocks

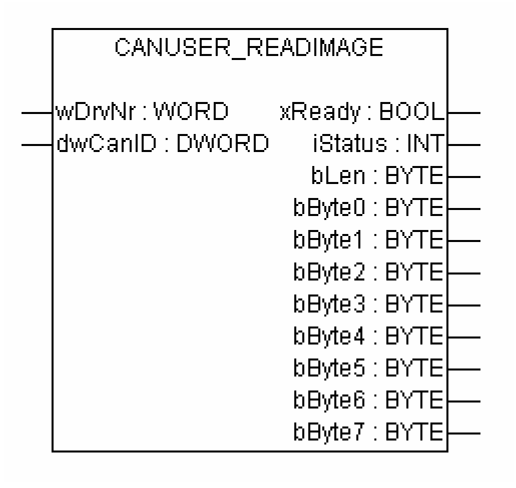
psCanVersions	: Version information of the various library versions is output.
uiNumberOfCanVersions	: Number of the returned string.

2.2.4 Example

```
VAR
        CU_GetVersions              : CanUser_GetVersions;
        psCanVersionStrings         : POINTER TO ARRAY[0..MAX_CAN_VERSIONS] OF STRING;
        uiNrOfCanVersionStrings: UINT;
END_VAR

CU_GetVersions(         psCanVersions=>psCanVersionStrings ,
                        uiNumberOfCanVersions=>uiNrOfCanVersionStrings
                );
```

2.3 CanUser_ReadImage



2.3.1 Description

Telegrams can be read via instances of the CanUser_ReadImage function block.
The last CAN telegram received is saved each time in the instance and can be read out from there.
The COB-ID to be read is assigned in the initialisation of the respective instance; an amendment of the COB-ID during operation is not permitted and generates an error message.
→ A separate instance of the CanUser_ReadImage function block is to be created for every COB-ID to be read.

This function block is supported by the following controls:

XC100	XC200	XC600	XVC100	HPG200	HPG300	XVC600
√ !	√ !	√ !	√	√	√	√

√	=	The function block is supported.
√*	=	The function is supported with limitations.
--	=	The function block is not supported.
!	=	Refer to section 1.3 for further explanations.

2.3.2 Inputs of the function blocks

wDrvNr	: CAN driver number. Default setting = “0” XC600→setting of the CANopen modules to be addressed “0”–“2”
dwCanID	: CAN object (COB-ID) which is to be read. (11 Bit identifier)

2.3.3 Outputs of the function blocks

xReady	: FALSE → function in preparation, return values invalid. TRUE → function completed, return values are valid.
iStatus	: Current status of the function block.
	0 → No CAN telegram available, receive queue is empty.
	1 → CAN telegram read and available.
	3 → RTR telegram read and available. (Only for XC600)
	-1 → Non permissible amendment of the COB-ID or driver number.
	-2 → Invalid COB-ID (valid are only non-referenced COB-IDs in the range from “0” to “16#7FF”.
	-3 → CAN driver not yet initialised.
	-4 → No memory slot available for creating the instance of the function block. Global variables “CanUser_DISPATCH_ARRAY_MAX_SIZE” selected is too small.
	-7x → Internal fault of the CAN driver.
bLen	: Number of data bytes in the read CAN telegram.
bByte0-bByte7	: Data bytes of the read CAN telegram.

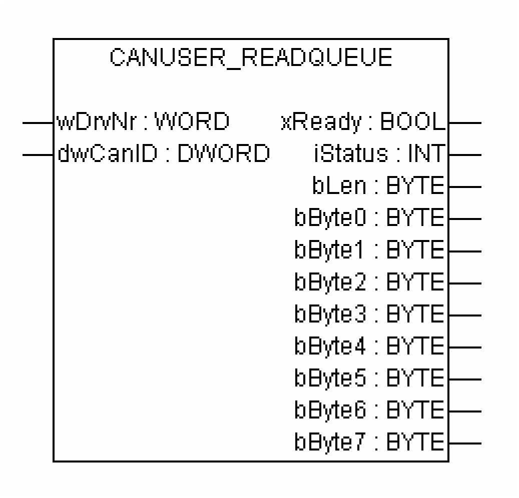
2.3.4 Example

```
VAR CONSTANT
        w_DrvNr         : WORD;
        (*      CAN driver number (default: 0 // XC600: 0-2 depends on jumper setting on the Hilscher board   )

        *)
        dw_CanID        : DWORD;
        (*      CAN Object - ID (COB-ID) --> May not be amended during operation  *)
END_VAR
VAR
        CU_ReadImage    : CanUser_ReadImage;
        x_Ready         : BOOL;
        i_Status        : INT;
        b_Len           : BYTE;
        b_Byte0         : BYTE;
        b_Byte1         : BYTE;
        b_Byte2         : BYTE;
        b_Byte3         : BYTE;
        b_Byte4         : BYTE;
        b_Byte5         : BYTE;
        b_Byte6         : BYTE;
        b_Byte7         : BYTE;
END_VAR
```

```
CU_ReadImage(  wDrvNr:= w_DrvNr ,
               dwCanID:= dw_CanID,
               xReady=>x_Ready ,
               iStatus=>i_Status ,
               bLen=> b_Len,
               bByte0=>b_Byte0 ,
               bByte1=>b_Byte1 ,
               bByte2=>b_Byte2 ,
               bByte3=>b_Byte3 ,
               bByte4=>b_Byte4 ,
               bByte5=>b_Byte5 ,
               bByte6=>b_Byte6 ,
               bByte7=>b_Byte7  );
IF x_Ready THEN
     (*Processing of the read image module complete.*)
     IF i_Status =0 THEN
          (* No CAN telegram available... *)
          ;
     ELSIF i_Status = 1 THEN
          (* CAN telegram received...*)
          ;
     ELSIF i_Status < 0 THEN
          (* Evaluate error message...*)
          ;
     END_IF
ELSE
     (* Processing of the read image module active, return values are invalid.*)
     ;
END_IF
```

2.4 CanUser_ReadQueue



2.4.1 Description

Telegrams can be read via instances of the CanUser_ReadImage function block.

The incoming CAN telegrams are intermediately saved in the internal FIFO queue and can be read from there. Generally, the CanUser_ReadImage must be accessed until the response appears that there are no further telegrams saved.

The COB-ID to be read is assigned in the initialisation of the respective instance; an amendment of the COB-ID during operation is not permitted and generates an error message. → A separate instance of the CanUser_ReadQueue function block is to be created for every COB-ID to be read.

This function block is supported by the following controls:

XC100	XC200	XC600	XVC100	HPG200	HPG300	XVC600
√ !	√ !	√ !	√	√	√	√

√	=	The function block is supported.
√*	=	The function is supported with limitations.
--	=	The function block is not supported.
!	=	Refer to section 1.3 for further explanations.

2.4.2 Inputs of the function blocks

wDrvNr	: CAN driver number. Default setting = “0” XC600→setting of the CANopen modules to be addressed “0”–“2”
dwCanID	: CAN object (COB-ID) which is to be read. (11 Bit identifier)

2.4.3 Outputs of the function blocks

xReady	: FALSE → function in preparation, return values invalid. TRUE → function completed, return values are valid.
iStatus	: Current status of the function block. 0 → No CAN telegram available, receive queue is empty. 1 → CAN telegram read and available. 3 → RTR telegram read and available. (Only for XC600) 5/7→ Buffer overrun: at least one CAN telegram has been overwritten. The telegram buffer contains 5: a CAN telegram/ 7: an RTR telegram. -1 → Non permissible amendment of the COB-ID or driver number. -2 → Invalid COB-ID (valid are only non-referenced COB-IDs in the range from “0” to “16#7FF”. -3 → CAN driver not yet initialised. -4 → No memory slot available for creating the instance of the function block. Global variables “CanUser_DISPATCH_ARRAY_MAX_SIZE” selected is too small. -7x → Internal fault of the CAN driver.
bLen	: Number of data bytes in the read CAN telegram.
bByte0-bByte7	: Data bytes of the read CAN telegram.

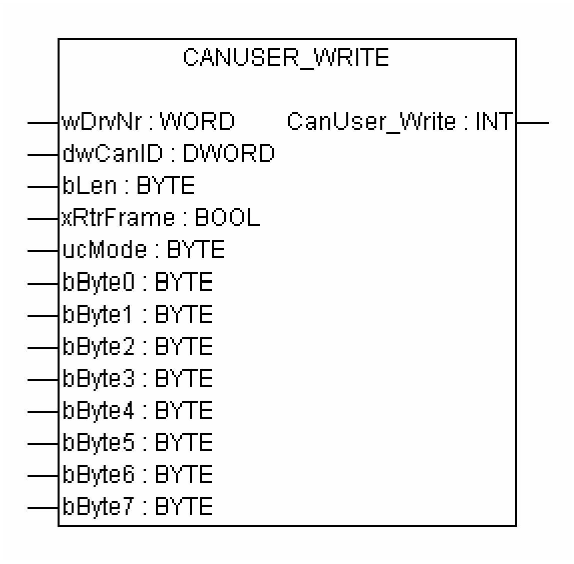
2.4.4 Example

```
VAR CONSTANT
        w_DrvNr         : WORD;
        (*      CAN driver number (default: 0 // XC600: 0-2 depends on jumper setting on the Hilscher board
        )
        *)
        dw_CanID        : DWORD;
        (* CAN Object - ID (COB-ID) --> May not be amended during operation  *)
END_VAR
```

```
VAR
        CU_ReadQueue  : CanUser_ReadQueue;
        x_Ready               : BOOL;
        i_Status              : INT;
        b_Len                 : BYTE;
        b_Byte0               : BYTE;
        b_Byte1               : BYTE;
        b_Byte2               : BYTE;
        b_Byte3               : BYTE;
        b_Byte4               : BYTE;
        b_Byte5               : BYTE;
        b_Byte6               : BYTE;
        b_Byte7               : BYTE;
        i_ReadCob: INT;
END_VAR

i_ReadCob := 1;
WHILE i_ReadCob = 1 DO
        (* Read out of the receive queue, as long as this returns a valid CAN telegram.*)
        CU_ReadQueue( wDrvNr:= w_DrvNr ,
                        dwCanID:= dw_CanID,
                        xReady=>x_Ready ,
                        iStatus=>i_Status ,
                        bLen=> b_Len,
                        bByte0=>b_Byte0 ,
                        bByte1=>b_Byte1 ,
                        bByte2=>b_Byte2 ,
                        bByte3=>b_Byte3 ,
                        bByte4=>b_Byte4 ,
                        bByte5=>b_Byte5 ,
                        bByte6=>b_Byte6 ,
                        bByte7=>b_Byte7 );
        IF x_Ready THEN
                (*Processing of the read image module complete.*)
                i_ReadCob := i_Status;
                IF i_Status =0 THEN
                        (* No CAN telegram available... *)
                        ;
                ELSIF i_Status = 1 THEN
                        (* CAN telegram received...*)
                        ;
                ELSIF i_Status < 0 THEN
                        (* Evaluate error message...*)
                        ;
                END_IF
        ELSE
                (* Processing of the read image module active, return values are invalid.*)
                ;
        END_IF
END_WHILE
```

2.5 CanUser_Write



2.5.1 Description

The CanUser_Write function makes it possible to place CAN telegrams on the bus.

The return value of the function should always be evaluated to ensure that the sent telegram has been accepted into the send queue of the CAN driver.

This function is supported by the following controls:

XC100	XC200	XC600	XVC100	HPG200	HPG300	XVC600
√	√	√	√	√	√	√

√	=	The function is supported.
√*	=	The function is supported with limitations.
--	=	The function is not supported.
!	=	Refer to section 1.3 for further explanations.

2.5.2 Input parameters

wDrvNr : CAN driver number. Default setting = "0"
XC600→setting of the CANopen modules to be addressed "0"–"2"
dwCanID : CAN object (COB-ID) which is to be read. (11 Bit identifier)
bLen : Number of data bytes in the CAN telegram to be written.
xRtrFrame : TRUE → The CAN telegram to be sent is sent as an RTR telegram. (RTR = Remote Transmit Request: request for a data telegram with the respective identifier.)
FALSE→ The CAN telegram to be sent is sent as a data telegram.
ucMode : Setting the queue which is to be used to send the telegram.
This parameter is used in the following controls: HPG200;HPG300;XVC600.
In other controls the parameter is ignored.
The CANopen driver uses the fast queue (high priority.)
0 → STANDARD_QUEUE (standard priority).
1 → FAST_QUEUE (high priority).
2 → INTERRUPT_QUEUE (interrupt - priority).
3 → IMAGE (standard priority. Only the COB identifier is placed in the queue to be sent. The associated data is only attached prior to being sent.)
bByte0-bByte7: Data bytes of the CAN telegram to be written.

2.5.3 Return value

CanUser_Write : 1 → OK.
-1 → Faulty data length.
-2 → Faulty COB-ID (valid IDs "0" to "16#7FF").
-3 → Faulty CAN driver number.
-4 → Faulty setting of the transmission queue (ucMode).
-5 → Send queue full
-7 → Function not supported by existing run time system.
-7x→ Internal fault.

2.5.4 Example

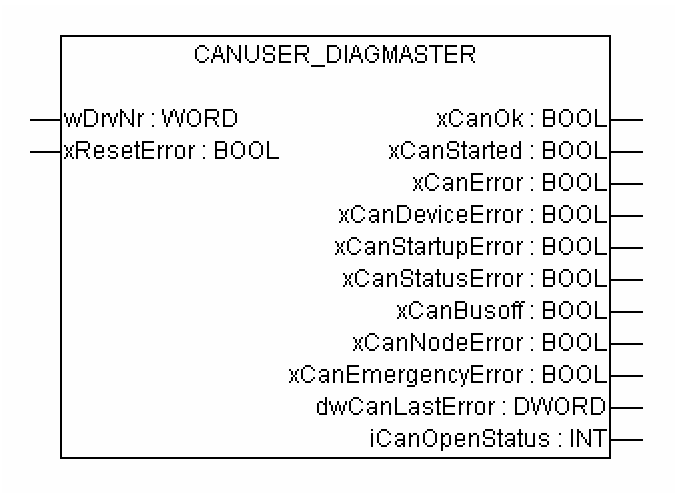
```
VAR CONSTANT
        w_DrvNr:        WORD;
        (*      CAN driver number (default: 0 // XC600: 0-2 depends on jumper setting on the Hilscher
                board)
         *)
        dw_CanID:       DWORD;
        (*      CAN Object - ID (COB-ID) --> May not be amended during operation.        *)
END_VAR


VAR
        CU_ReadQueue  : CanUser_ReadQueue;
        b_Len                   : BYTE;
        xRtrFrame               : BOOL;
        uc_Mode                 : BYTE;
        b_Byte0                 : BYTE;
        b_Byte1                 : BYTE;
        b_Byte2                 : BYTE;
        b_Byte3                 : BYTE;
        b_Byte4                 : BYTE;
        b_Byte5                 : BYTE;
        b_Byte6                 : BYTE;
        b_Byte7                 : BYTE;
END_VAR

CanUser_Write(  wDrvNr:= w_DrvNr ,
                dwCanID:= dw_CanID,
                bLen := b_Len,
                xRtrFrame := xRtrFrame,
                ucMode := uc_Mode,
                bByte0:=b_Byte0 ,
                bByte1:=b_Byte1 ,
                bByte2:=b_Byte2 ,
                bByte3:=b_Byte3 ,
                bByte4:=b_Byte4 ,
                bByte5:= b_Byte5 ,
                bByte6:=b_Byte6 ,
                bByte7:= b_Byte7 );
```

3 CanUser_Master.lib function blocks

3.1 CanUser_DiagMaster



3.1.1 Description

The diagnostics information of a CANopen master can be read via an instance of this function block. The CANopen master module must be integrated into the control configuration of the project for this purpose.

This function block is supported by the following controls:

XC100	XC200	XC600	XVC100	HPG200	HPG300	XVC600
√*	√*	--	√	√	√	√

√	=	The function block is supported.
√*	=	The function is supported with limitations.
--	=	The function block is not supported.
!	=	Refer to section 1.3 for further explanations.

3.1.2 Inputs of the function blocks

wDrvNr	: CAN driver number. Default setting = "0" XC600→Jumper setting of the CANopen modules "0"–"2" to be contacted
xResetError	: TRUE → Reset all diagnostics variables. FALSE → Retain the values of the diagnostics variables.

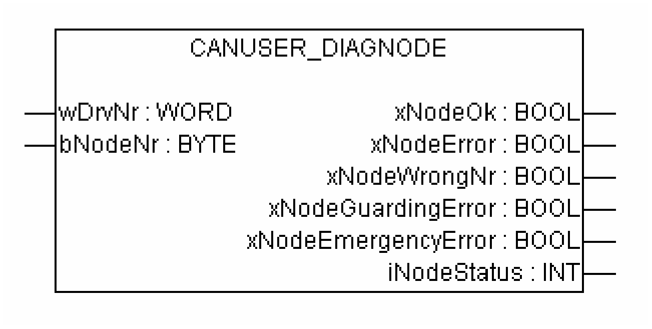
3.1.3 Outputs of the function blocks

xCanOk	:	TRUE → CAN master and CAN station successfully started. (Dependent on the used control the conditions to activate the flag can vary.)
xCanStarted	:	TRUE → CAN master successfully started.
xCanError	:	TRUE → A CAN fault has been detected (xCanStartupError; xCanDeviceError; xCanBusOff; xCanStatusError; xCanLastError <> 0). (Dependent on the used control the conditions to activate the flag can vary.)
xCanDeviceError	:	TRUE → No CAN hardware found.
xCanStartupError	:	TRUE → Fault with the start of the CAN hardware.
xCanStatusError	:	TRUE → No CAN cable connected, or no CAN station exists on the bus.
xCanBusoff	:	TRUE → Communication fault. (e.g.: incorrect baud rate set or bust termination has not been set.)
xCanNodeError	:	TRUE → At least one configured CAN station has not been recognised.
xCanEmergencyError	:	TRUE → An emergency telegram (fault message) exists for at least one CAN station. (Dependent on the used control the conditions to activate the flag can vary.)
dwCanLastError	:	Error code of the last emergency telegram.
iCanOpenStatus	:	Status of the CAN master

3.1.4 Example

```
PROGRAM PLC_PRG
VAR
	Can_DiagMaster : CanUser_DiagMaster;
	wDrvNr: WORD:=0;
	xResetError: BOOL:=FALSE;
	xCanOk: BOOL;
	xCanStarted: BOOL;
	xCanError: BOOL;
	xCanDeviceError: BOOL;
	xCanStartupError: BOOL;
	xCanStatusError: BOOL;
	xCanBusoff: BOOL;
	xCanNodeError: BOOL;
	xCanEmergencyError: BOOL;
	dwCanLastError: DWORD;
	iCanOpenStatus: INT;
END_VAR
Can_DiagMaster(
	wDrvNr:=wDrvNr ,
	xResetError:=xResetError ,
	xCanOk=>xCanOk ,
	xCanStarted=>xCanStarted ,
	xCanError=>xCanError ,
	xCanDeviceError=>xCanDeviceError ,
        xCanStartupError=>xCanStartupError ,
        xCanStatusError=>xCanStatusError ,
        xCanBusoff=> xCanBusoff,
        xCanNodeError=> xCanNodeError,
        xCanEmergencyError=> xCanEmergencyError,
        dwCanLastError=>dwCanLastError ,
        iCanOpenStatus=>iCanOpenStatus );
```

3.2 CanUser_DiagNode



3.2.1 Description

Diagnostics information of the individual CAN stations can be accessed via an instance of this function block.
In order to receive diagnostics information via a CAN station, the addressed CAN station must be integrated into the control configuration.

This function block is supported by the following controls:

XC100	XC200	XC600	XVC100	HPG200	HPG300	XVC600
√	√	√ !	√	√	√	√

√	=	The function block is supported.
√*	=	The function is supported with limitations.
--	=	The function block is not supported.
!	=	Refer to section 1.3 for further explanations.

3.2.2 Inputs of the function blocks

wDrvNr	: CAN driver number. Default setting = “0” XC600→Jumper setting of the CANopen modules “0”–“2” to be contacted
bNodeNr	: Node No. of the CAN station whose diagnostics information is to be read (Node 1 to 127).

3.2.3 Outputs of the function blocks

xReady : (Only for XC600.) Display of the function block status.

TRUE	: Read in preparation.
FALSE	: Read completed.

Independent of function block status, the last read status of the CAN station is read via the return values.

xNodeOk : TRUE → CAN station started.

(Dependent on the used control the conditions to activate the flag can vary.)

xNodeError : TRUE → No fault detected for the CAN station.

xNodeWrongNr : TRUE → No CAN station with the given Node No. exists in the control configuration, or the CAN station identity defined in the control configuration does not correspond with the identity read on the station.

xNodeGuardingError: TRUE → A guarding fault has been registered. The cyclic monitoring of the CAN station has failed.

The "xNodeGuardingError" flag will be reset after a successful recoupling of the station.

xNodeEmergencyError: TRUE → An emergency telegram has been received from this CAN station.

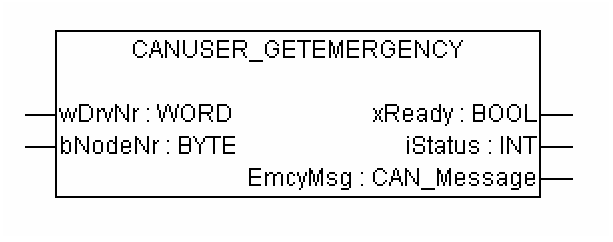
iNodeStatus :Current status of the CAN station.

1-4	= CAN station starts up,
5	= CAN station started.

3.2.4 Example

```
PROGRAM PLC_PRG
VAR
       Can_DiagNode : CanUser_DiagNode;
       wDrvNr: WORD:=0;
       bNodeNr: BYTE :=9;
       xNodeOk: BOOL;
       xNodeError: BOOL;
       xNodeWrongNr: BOOL;
       xNodeGuardingError: BOOL;
       xNodeEmergencyError: BOOL;
       iNodeStatus: INT;
END_VAR
Can_DiagNode(
       wDrvNr:=wDrvNr ,
       bNodeNr:=bNodeNr ,
       xNodeOk=>xNodeOk ,
       xNodeError=> xNodeError,
       xNodeWrongNr=>xNodeWrongNr ,
       xNodeGuardingError=>xNodeGuardingError ,
       xNodeEmergencyError=>xNodeEmergencyError ,
       iNodeStatus=>iNodeStatus );
```

3.3 CanUser_GetEmergency



3.3.1 Description

An emergency telegram of a CAN station can be read out via the instance of this function block. The last read emergency telegram is always returned.

In order to use this function the accessed CAN station must be integrated into the control configuration.

This function should be used in conjunction with the CanUser_DiagNode function.

This function block is supported by the following controls:

XC100	XC200	XC600	XVC100	HPG200	HPG300	XVC600
√	√	√	√	√	√	√

√	=	The function block is supported.
√*	=	The function is supported with limitations.
--	=	The function block is not supported.
!	=	Refer to section 1.3 for further explanations.

3.3.2 Inputs of the function blocks

wDrvNr	: CAN driver number. Default setting = “0” XC600→Jumper setting of the CANopen modules “0”–“2” to be contacted
bNodeNr	: Node No. of the CAN station whose emergency telegram is to be read. (Node 1 to 127.)

3.3.3 Outputs of the function blocks

xReady	: TRUE → Processing complete. Values can be read out. FALSE → Reading of the emergency telegram in preparation. No valid return values.	
iStatus	: 2	(XC600 only): No new emergency telegram received. The last read emergency telegram is returned.
	1	Emergency telegram has been read successfully
	0	Read out of the emergency telegram in preparation.
	-1	Invalid input parameter (wDrvNr or bNodeNr).
	-6	station with the given Node No. exists in the control configuration.
EmcyMsg	: CAN telegram which contains the emergency telegram to be read.	

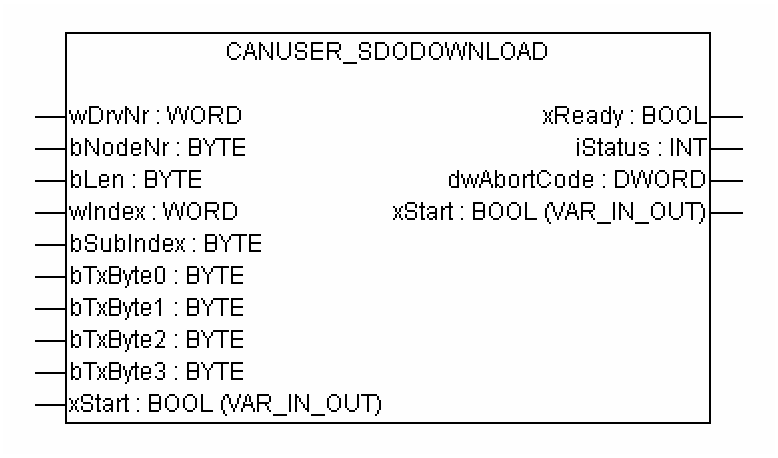
3.3.4 Example

```
PROGRAM PLC_PRG
VAR
        Can_DiagNode : CanUser_DiagNode;
        wDrvNr: WORD:=0;
        bNodeNr: BYTE :=9;
        xNodeOk: BOOL;
        xNodeError: BOOL;
        xNodeWrongNr: BOOL;
        xNodeGuardingError: BOOL;
        xNodeEmergencyError: BOOL;
        iNodeStatus: INT;

        Can_GetEmergency : CanUser_GetEmergency;
        xReady: BOOL;
        iStatus: INT;
        EmcyMsg: CAN_Message;
END_VAR
(* CanUser_DiagNode *)
Can_DiagNode(
        wDrvNr:=wDrvNr ,
        bNodeNr:=bNodeNr ,
        xNodeOk=>xNodeOk ,
        xNodeError=> xNodeError,
        xNodeWrongNr=>xNodeWrongNr ,
        xNodeGuardingError=>xNodeGuardingError ,
        xNodeEmergencyError=>xNodeEmergencyError ,
        iNodeStatus=>iNodeStatus );

(* Emergency-Messages*)
IF xNodeEmergencyError THEN
        Can_GetEmergency(
                wDrvNr:=wDrvNr ,
                bNodeNr:=bNodeNr ,
                xReady=>xReady ,
                iStatus=>iStatus ,
                EmcyMsg=>EmcyMsg );
END_IF
```

3.4 CanUser_SDOdownload



3.4.1 Description

1 to 4 bytes of user data can be sent via an SDO transfer to the defined CAN station via an instance of this function block.
The address of the station is entered via the Node No., the memory slot in the object directory of the CAN station is determined via the Index and Subindex inputs.

This function block is supported by the following controls:

XC100	XC200	XC600	XVC100	HPG200	HPG300	XVC600
√ !	√ !	√	√	√	√	√

√	=	The function block is supported.
√*	=	The function is supported with limitations.
--	=	The function block is not supported.
!	=	Refer to section 1.3 for further explanations.

3.4.2 Inputs of the function blocks

wDrvNr	: CAN driver number. Default setting = “0” XC600→Jumper setting of the CANopen modules “0”–“2” to be contacted
bNodeNr	: Node No. of the CAN station to which the SDO telegram should be sent. (Node 1 to 127.)
bLen	: Number of data bytes which are written to the target address . (Valid values: 1 – 4.)
wIndex	: Index (first part of the address to which the data bytes are to be written .) Values :0-16#FFF.
bSubIndex	: Subindex (Second part of the address to which the data bytes are to be written.) Values :0-16#255.
bTxByte0-bTyByte3	: Data bytes to be sent.
xStart	: FALSE → TRUE: Start of a new SDO transfer

TRUE → FALSE: Stop of the SDO transfer and reset of the module.

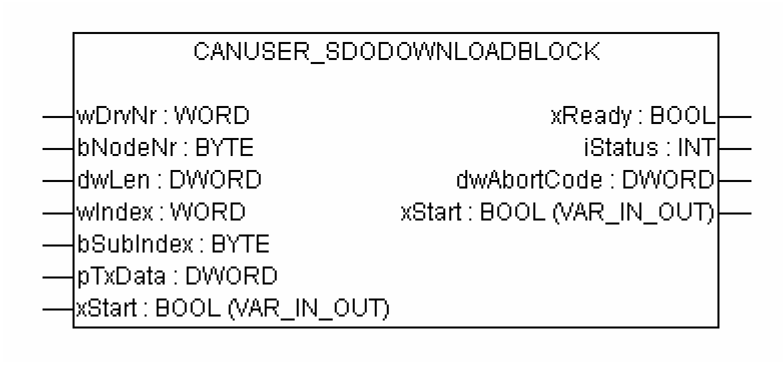
3.4.3 Outputs of the function blocks

xReady	: TRUE : SDO transfer in progress. FALSE : SDO - transfer ended.
iStatus	: Status of the function block 0 : SDO transfer in progress. 1 : SDO transfer successfully ended. -1 : Time monitoring of the SDO transfer timed out. (Default: 500 ms) -2 : Aborting of the SDO transfer. -3 : A further instance of a SDO transfers is operating on the defined CAN station, or all available memory slots for SDO connections are used. -4 : Invalid node no. -5 : Invalid data length entered.
dwAbortCode	: If an abort of the SDO transfer is displayed (iStatus = -2), the respective AbortCode can be read out in this return value. (CIA document DS301)

3.4.4 Example

```
PROGRAM PLC_PRG
VAR
        Can_SDOdownload : CanUser_SDOdownload;
        wDrvNr: WORD := 0;
        bNodeNr: BYTE := 9;
        bLen: BYTE := 3;
        wIndex: WORD :=16#1005;
        bSubIndex: BYTE := 0;
        bTxByte0: BYTE := 11;
        bTxByte1: BYTE := 22;
        bTxByte2: BYTE := 33;
        bTxByte3: BYTE := 0;
        xStart: BOOL;  (* Change xStart from FALSE to TRUE to start the Transfer *)
        xReady: BOOL;
        iStatus: INT;
        dwAbortCode: DWORD;
END_VAR
Can_SDOdownload(
        wDrvNr := wDrvNr,
        bNodeNr := bNodeNr,
        bLen := bLen,
        wIndex := wIndex,
        bSubIndex := bSubIndex,
        bTxByte0 := bTxByte0,
        bTxByte1 := bTxByte1,
        bTxByte2 := bTxByte2,
        bTxByte3 := bTxByte3,
        xStart := xStart,
        xReady => xReady,
        iStatus => iStatus,
        dwAbortCode =>dwAbortCode);
```

3.5 CanUser_SDOdownloadBlock



3.5.1 Description

Larger amounts of data can be sent via a block download to a CAN station via an instance of this function block. The CAN station must support the SDO block download function in order to use this function.

A CRC check is not supported.

The address of the station is entered via the Node No., the memory slot in the object directory of the CAN station is determined via the Index and Subindex inputs.

NOTE:
For the HPG200/HPG300 and XVC100/XVC600 controls, the
CanUser_SDOdownloadSegmented
function block is implemented as
CanUser_SDOdownloadBlock
.

This function block is supported by the following controls:

XC100	XC200	XC600	XVC100	HPG200	HPG300	XVC600
√ !	√ !	--	-- !	-- !	-- !	-- !

√	=	The function block is supported.
√*	=	The function is supported with limitations.
--	=	The function block is not supported.
!	=	Refer to section 1.3 for further explanations.

3.5.2 Inputs of the function blocks

wDrvNr	: CAN driver number. Default setting = “0” XC600→Jumper setting of the CANopen modules “0”–“2” to be contacted
bNodeNr	: Node No. of the CAN station to which the SDO telegram is to be sent. (Node 1 to 127.)
bLen	: Number of data bytes which are written to the target address .
wIndex	: Index (first part of the address to which the data bytes are to be written.) Values :0-16#FFF.
bSubIndex	: Subindex (Second part of the address to which the data bytes are to be written.) Values :0-255.
pTxData	: Pointer to an array of data bytes to be sent.
xStart	: FALSE → TRUE: Start of a new SDO transfer TRUE → FALSE: Stop of the SDO transfer and reset of the module.

3.5.3 Outputs of the function blocks

xReady	: TRUE : SDO transfer in progress. FALSE : SDO - transfer ended.
iStatus	: Status of the function block 0 : SDO transfer in progress. 1 : SDO transfer successfully ended. -1 : Time monitoring of the SDO transfer timed out. (Default: 500 ms) -2 : Aborting of the SDO transfer. -3 : A further instance of a SDO transfers is operating on the defined CAN station, or all available memory slots for SDO connections are used. -4 : Invalid node no. -5 : Invalid data length entered. -7 : Function is not supported in the current runtime system.
dwAbortCode	: If an abort of the SDO transfer is displayed (iStatus = -2), the respective AbortCode can be read out in this return value. (CIA document DS301)

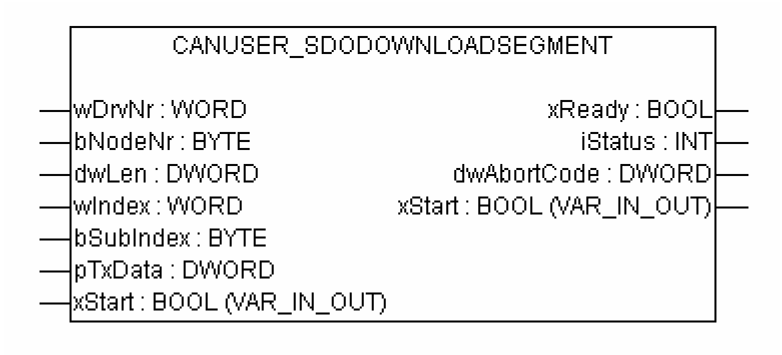
3.5.4 Example

```
PROGRAM PLC_PRG
VAR
       xFirstRun : BOOL := TRUE;
       Can_SDOdownloadBlock : CanUser_SDOdownloadBlock;
       wDrvNr: WORD := 0;
       bNodeNr: BYTE := 9;
       dwLen: DWORD := 852;
       wIndex: WORD :=16#1005;
       bSubIndex: BYTE := 0;
       pTxData: POINTER TO BYTE;
       xStart: BOOL;  (* Change xStart from FALSE to TRUE to start the Transfer *)
       xReady: BOOL;
       iStatus: INT;
       dwAbortCode: DWORD;
       ar_bDataBuffer: ARRAY [0..1000] OF BYTE;
       i: INT;
       bDummy: BYTE := 0;
END_VAR

IF xFirstRun THEN
       xFirstRun := FALSE;
       pTxData := ADR( ar_bDataBuffer[0] );
       FOR i:=0 TO 1000 DO
              bDummy := bDummy +1;
              ar_bDataBuffer[i] := bDummy;
       END_FOR
END_IF

(* CanUser_DiagNode *)
Can_SDOdownloadBlock(
       wDrvNr:=wDrvNr ,
       bNodeNr:=bNodeNr ,
       dwLen:=dwLen,
       wIndex:=wIndex ,
       bSubIndex:= bSubIndex,
       pTxData:=pTxData ,
       xStart:=xStart ,
       xReady=>xReady ,
       iStatus=>iStatus ,
       dwAbortCode=>dwAbortCode );
```

3.6 CanUser_SDOdownloadSegment



3.6.1 Description

Data can be sent via a segmented download to a CAN station via an instance of this function block.

The address of the station is entered via the Node No., the memory slot in the object directory of the CAN station is determined via the Index and Subindex inputs.

<u>**NOTE:**</u>
For the HPG200/HPG300 and XVC100/XVC600 controls, the function block
CanUser_SDOdownloadSegmented
is implemented under the name
CanUser_SDOdownloadBlock
.

This function block is supported by the following controls:

XC100	XC200	XC600	XVC100	HPG200	HPG300	XVC600
√ !	√ !	√* !	√ !	√ !	√ !	√ !

√	=	The function block is supported.
√*	=	The function is supported with limitations.
--	=	The function block is not supported.
!	=	Refer to section 1.3 for further explanations.

3.6.2 Inputs of the function blocks

wDrvNr	: CAN driver number. Default setting = “0” XC600→Jumper setting of the CANopen modules “0”–“2” to be contacted
bNodeNr	: Node No. of the CAN station to which the SDO telegram should be sent. (Node 1 to 127.)
bLen	: Number of data bytes which are written to the target address .
wIndex	: Index (first part of the address to which the data bytes are to be written.) Values :0-16#FFF.
bSubIndex	: Subindex (Second part of the address to which the data bytes are to be written.) Values :0-255.
pTxData	: Pointer to an array of data bytes to be sent.
xStart	: FALSE → TRUE: Start of a new SDO transfer TRUE → FALSE: : Stop of the SDO transfer and reset of the module.

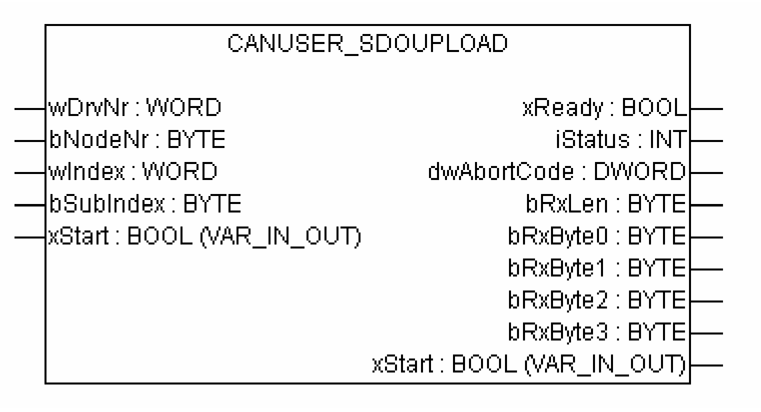
3.6.3 Outputs of the function blocks

xReady	: TRUE : SDO transfer in progress. FALSE : SDO - transfer ended.
iStatus	: Status of the function block 0 : SDO transfer in progress. 1 : SDO transfer successfully ended. -1 : Time monitoring of the SDO transfer timed out. (Default: 500 ms) -2 : Aborting of the SDO transfer. -3 : A further instance of a SDO transfers is operating on the defined CAN station, or all available memory slots for SDO connections are used. -4 : Invalid node no. -5 : Invalid data length entered. -7 : Function is not supported in the current runtime system.
dwAbortCode	: If an abort of the SDO transfer is displayed (iStatus = -2), the respective AbortCode can be read out in this return value. (CIA document DS301)

3.6.4 Example

```
PROGRAM PLC_PRG
VAR
       xFirstRun : BOOL := TRUE;
       Can_SDOdownloadSegment : CanUser_SDOdownloadSegment;
       wDrvNr: WORD := 0;
       bNodeNr: BYTE := 9;
       dwLen: DWORD := 852;
       wIndex: WORD :=16#1005;
       bSubIndex: BYTE := 0;
       pTxData: POINTER TO BYTE;
       xStart: BOOL;  (* Change xStart from FALSE to TRUE to start the Transfer *)
       xReady: BOOL;
       iStatus: INT;
       dwAbortCode: DWORD;
       ar_bDataBuffer: ARRAY [0..1000] OF BYTE;
END_VAR
Can_SDOdownloadSegment(
       wDrvNr:=wDrvNr ,
       bNodeNr:=bNodeNr,
       dwLen:= dwLen,
       wIndex:= wIndex,
       bSubIndex:=bSubIndex ,
       pTxData:=pTxData ,
       xStart:=xStart ,
       xReady=>xReady ,
       iStatus=>iStatus ,
       dwAbortCode=>dwAbortCode );
```

3.7 CanUser_SDOupload



3.7.1 Description

1 to 4 bytes of user data can be read via an SDO transfer from a defined CAN station via an instance of this function block.
The address of the station is entered via the Node No., the memory slot in the object directory of the CAN station is determined via the Index and Subindex inputs.

This function block is supported by the following controls:

XC100	XC200	XC600	XVC100	HPG200	HPG300	XVC600
√ !	√ !	√	√	√	√	√

√	=	The function block is supported.
√*	=	The function is supported with limitations.
--	=	The function block is not supported.
!	=	Refer to section 1.3 for further explanations.

3.7.2 Inputs of the function blocks

wDrvNr	: CAN driver number. Default setting = “0” XC600→Jumper setting of the CANopen modules “0”–“2” to be contacted
bNodeNr	: Node No. of the CAN station to which the SDO telegram should be sent. (Node 1 to 127.)
wIndex	: Index (first part of the address to which the data bytes are to be written.) Values :0-16#FFF.
bSubIndex	: Subindex (Second part of the address to which the data bytes are to be written.) Values :0-16#255.
xStart	: FALSE → TRUE: Start of a new SDO transfer TRUE → FALSE: : Stop of the SDO transfer and reset of the module.

3.7.3 Outputs of the function blocks

xReady : TRUE : SDO transfer in progress.
FALSE : SDO - transfer ended.

iStatus : Status of the function block
0 : SDO transfer in progress.
1 : SDO transfer successfully ended.
-1 : Time monitoring of the SDO transfer timed out.
(Default: 500 ms)
-2 : Aborting of the SDO transfer.
-3 : A further instance of a SDO transfers is operating
on the defined CAN station, or all available memory slots
for SDO connections are used.
-4 : Invalid node no.
-5 : Invalid data length entered.

dwAbortCode : If an abort of the SDO transfer is displayed (iStatus = -2), the respective AbortCode can be read out in this return value. (CIA document DS301)

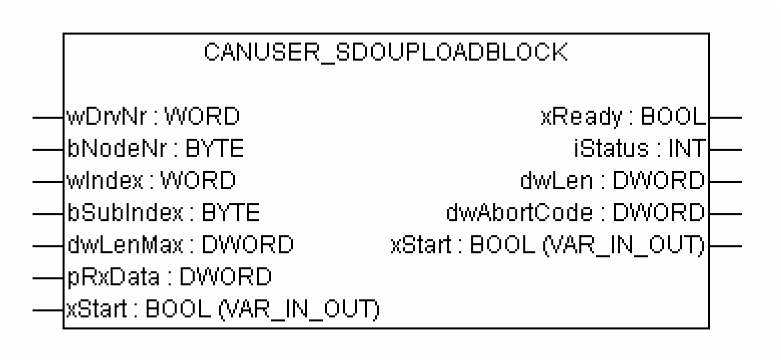
dwLen : Number of data bytes which have been read out of the target address

bRxByte0-bRxByte1: Memory slot for read data bytes.

3.7.4 Example

```
PROGRAM PLC_PRG
VAR
        xFirstRun : BOOL := TRUE;
        Can_SDOupload : CanUser_SDOupload;
        wDrvNr: WORD := 0;
        bNodeNr: BYTE := 9;
        wIndex: WORD :=16#1000;
        bSubIndex: BYTE := 0;
        xStart: BOOL;  (* Change xStart from FALSE to TRUE to start the Transfer *)
        xReady: BOOL;
        iStatus: INT;
        dwAbortCode: DWORD;
        bRxLen: BYTE;
        bRxByte0: BYTE;
        bRxByte1: BYTE;
        bRxByte2: BYTE;
        bRxByte3: BYTE;
END_VAR
Can_SDOupload(
        wDrvNr:=wDrvNr ,
        bNodeNr:=bNodeNr ,
        wIndex:=wIndex ,
        bSubIndex:= bSubIndex,
        xStart:=xStart ,
        xReady=>xReady ,
        iStatus=>iStatus ,
        dwAbortCode=>dwAbortCode,
        bRxLen=> bRxLen,
        bRxByte0=>bRxByte0 ,
        bRxByte1=>bRxByte1,
        bRxByte2=>bRxByte2,
        bRxByte3=>bRxByte3 );
```

3.8 CanUser_SDOuploadBlock



3.8.1 Description

Larger amounts of data can be read out via a block download from a CAN station with an instance of this function block. The CAN station must support the SDO block download function in order to use this function.

A CRC check is not supported.

The “CanUser_SEGMENTS_PER_BLOCK” variable can modify the number of data segments up to the next acknowledgement (Default value = 4).

The address of the station is entered via the Node No., the memory slot in the object directory of the CAN station is determined via the Index and Subindex inputs.

NOTE:

For the HPG200/HPG300 and XVC100/XVC600 controls, the

CanUser_SDOuploadBlock

function block is implemented as

CanUser_SDOuploadSegmented

.

This function block is supported by the following controls:

XC100	XC200	XC600	XVC100	HPG200	HPG300	XVC600
√ !	√ !	--	-- !	-- !	-- !	-- !

√	=	The function block is supported.
√*	=	The function is supported with limitations.
--	=	The function block is not supported.
!	=	Refer to section 1.3 for further explanations.

3.8.2 Inputs of the function blocks

wDrvNr	: CAN driver number. Default setting = “0” XC600→Jumper setting of the CANopen modules “0”–“2” to be contacted
bNodeNr	: Node No. of the CAN station to which the SDO telegram should be sent. (Node 1 to 127.)
wIndex	: Index (first part of the address to which the data bytes are to be written.) Values :0-16#FFF.
bSubIndex	: Subindex (Second part of the address to which the data bytes are to be written.) Values :0-255.
dwLenMax	: Maximum number of data bytes which can be written into the transferred data array.
pRxData	: Pointer to an “Array of bytes” into which the read data out bytes can be written.
xStart	: FALSE → TRUE: Start of a new SDO transfer TRUE → FALSE: : Stop of the SDO transfer and reset of the module.

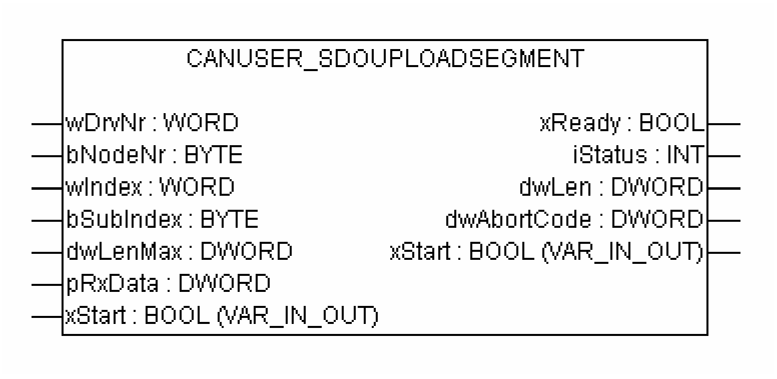
3.8.3 Outputs of the function blocks

xReady	: TRUE : SDO transfer in progress. FALSE : SDO - transfer ended.
iStatus	: Status of the function block 0 : SDO transfer in progress. 1 : SDO transfer successfully ended. -1 : Time monitoring of the SDO transfer timed out. (Default: 500 ms) -2 : Aborting of the SDO transfer. -3 : A further instance of a SDO transfers is operating on the defined CAN station, or all available memory slots for SDO connections are used. -4 : Invalid node no. -5 : Invalid data length entered. -7 : Function is not supported in the current runtime system.
dwLen	: Number of read out data bytes.
dwAbortCode	: If an abort of the SDO transfer is displayed (iStatus = -2), the respective AbortCode can be read out in this return value. (CIA document DS301)

3.8.4 Example

```
PROGRAM PLC_PRG
VAR
       Can_SDOuploadBlock : CanUser_SDOuploadBlock;
       wDrvNr: WORD := 0;
       bNodeNr: BYTE := 9;
       wIndex: WORD :=16#1000;
       bSubIndex: BYTE := 0;
       xStart: BOOL;  (* Change xStart from FALSE to TRUE to start the Transfer *)
       xReady: BOOL;
       iStatus: INT;
       dwAbortCode: DWORD;
       ar_bDataBuffer: ARRAY [0..1000] OF BYTE;
       dwLenMax: DWORD :=1001;
       pRxData: POINTER TO BYTE;
       dwLen: DWORD;
END_VAR
pRxData := ADR(ar_bDataBuffer[0]);
(* CanUser_SDOuploadBlock *)
Can_SDOuploadBlock(
       wDrvNr:=wDrvNr ,
       bNodeNr:=bNodeNr ,
       wIndex:=wIndex,
       bSubIndex:=bSubIndex ,
       dwLenMax:=dwLenMax ,
       pRxData:=pRxData ,
       xStart:=xStart ,
       xReady=>xReady,
       iStatus=> iStatus,
       dwLen=> dwLen,
       dwAbortCode=>dwAbortCode );
```

3.9 CanUser_SDOuploadSegment



3.9.1 Description

Data can be read via a segmented upload from a CAN station via an instance of this function block.

The address of the station is entered via the Node No., the memory slot in the object directory of the CAN station is determined via the Index and Subindex inputs.

NOTE:
For the HPG200/HPG300 and XVC100/XVC600 controls, the function block
CanUser_SDOuploadSegmented
is implemented under the name
CanUser_SDOuploadBlock
.

This function block is supported by the following controls:

XC100	XC200	XC600	XVC100	HPG200	HPG300	XVC600
√ !	√ !	√* !	√ !	√ !	√ !	√ !

√	=	The function block is supported.
√*	=	The function is supported with limitations.
--	=	The function block is not supported.
!	=	Refer to section 1.3 for further explanations.

3.9.2 Inputs of the function blocks

wDrvNr	: CAN driver number. Default setting = “0” XC600→Jumper setting of the CANopen modules “0”–“2” to be contacted
bNodeNr	: Node No. of the CAN station to which the SDO telegram is to be sent. (Node 1 to 127.)
wIndex	: Index (first part of the address to which the data bytes are to be written.) Values :0-16#FFF.
bSubIndex	: Subindex (Second part of the address to which the data bytes are to be written.) Values :0-255.
dwLenMax	: Maximum number of data bytes which can be written into the transferred data array.
pRxData	: Pointer to an “Array of bytes” into which the read data out bytes can be written.
xStart	: FALSE → TRUE: Start of a new SDO transfer TRUE → FALSE: : Stop of the SDO transfer and reset of the module.

3.9.3 Outputs of the function blocks

xReady	: TRUE : SDO transfer in progress. FALSE : SDO - transfer ended.
iStatus	: Status of the function block 0 : SDO transfer in progress. 1 : SDO transfer successfully ended. -1 : Time monitoring of the SDO transfer timed out. (Default: 500 ms) -2 : Aborting of the SDO transfer. -3 : A further instance of a SDO transfers is operating on the defined CAN station, or all available memory slots for SDO connections are used. -4 : Invalid node no. -5 : Invalid data length entered. -7 : Function is not supported in the current runtime system.
dwLen	: Number of read out data bytes.
dwAbortCode	: If an abort of the SDO transfer is displayed (iStatus = -2), the respective AbortCode can be read out in this return value. (CIA document DS301)

3.9.4 Example

```
PROGRAM PLC_PRG
VAR
        Can_SDOuploadSegment : CanUser_SDOuploadSegment;
        wDrvNr: WORD := 0;
        bNodeNr: BYTE := 9;
        wIndex: WORD :=16#1000;
        bSubIndex: BYTE := 0;
        xStart: BOOL;  (* Change xStart from FALSE to TRUE to start the Transfer *)
        xReady: BOOL;
        iStatus: INT;
        dwAbortCode: DWORD;
        ar_bDataBuffer: ARRAY [0..1000] OF BYTE;
        dwLenMax: DWORD :=1001;
        pRxData: POINTER TO BYTE;
        dwLen: DWORD;
END_VAR
pRxData := ADR(ar_bDataBuffer[0]);
(* CanUser_SDOuploadSegmented *)
Can_SDOuploadSegment(
        wDrvNr:=wDrvNr ,
        bNodeNr:=bNodeNr ,
        wIndex:= wIndex,
        bSubIndex:=bSubIndex ,
        dwLenMax:=dwLenMax ,
        pRxData:=pRxData ,
        xStart:=xStart ,
        xReady=>xReady ,
        iStatus=>iStatus ,
        dwLen=>dwLen ,
        dwAbortCode=>dwAbortCode );
```