Software for Xemo - Controllers

Xemo DLL
User Manual

Functions library (DLL) for programming Xemo controllers with MotionBasic functionality under Windows - Introduction, Language reference, Examples -

systec Industrial Systems GmbH

Systec Industrial Systems GmbH
Nottulner Landweg 90
48161 Muenster - Germany

Telephone	+49-(0)2534-8001-70
	+49-(0)700-SYSTEC-DE
Telefax	+49-(0)2534-8001-77
Email	info@systec.de
Internet	www.systec.de

Doc. no. 591.11-10.7
Version: 10 2019
Translation of the original manual

Table of contents

1 Introduction

In addition to the development environment IDE for offline programming, MotionBasic also features a Win32-DLL and a Win64-DLL for the PC programmer who wishes to program his/her own application-specific user interface in connection with a Systec Xemo controller. For this reason the DLL is called Xemo-DLL. A DLL (Dynamic Link Library) is a construction of the Microsoft company to encapsulate functionalities in its operating system Windows and in that way make them available for other programs.

The Xemo-DLL provides all commands from the controller as library functions (procedures) for the target languages C/C++, PASCAL, BASIC. The controller can thus be connected to a PC user interface as an intelligent front-end.
The DLL performs the initialization of the serial interface (RS 232 or USB), converts the commands into the correct transmission format, and transmits the data to the controller. Error recovery and diagnostic routines remain active.

1.1 Installation

Xemo-DLL

The Xemo-DLL is a non-administered Windows DLL and is compatible with the following operating systems:

Windows 95 / 98
Windows NT 4.0, 2000, XP
Windows 7 (32 and 64 bit)
Windows 8 (32- und 64-Bit)

Declaration files

In addition to the actual Xemo DLL, declaration files for various development environments are included. The following development environments are supported:

Microsoft Visual C++
Microsoft Visual Basic (VBA, Version 6)
Microsoft Visual Basic – NET
Microsoft C# - NET
Borland C++ Builder

Delivery contents

The following files are contained in the Xemo DLL:

XemoDLL.dll	the 32 bit Windows DLL
Xemo64.dll	the 64 bit Windows DLL

Definition files for Microsoft Visual C++ / Borland C++ Builder

Xemodll.h	Functions prototypes

mbconst.h	MotionBasic constants
XemoDLL.lib	Functions library (Microsoft Visual C++), 32 bit
Xemo64.lib	Functions library (Microsoft Visual C++), 64 bit

The function library for Borland C++ Builder can be generated through the Xemo DLL.dll with the Borland Tool IMPLIB.

Remark

Please retain the naming of the files for the 64-bit programming, Xemo64.dll, and Xemo64.lib for a smooth process.

Definition files for Microsoft Visual Basic

Xemodll.bas	Function declaration for Microsoft Visual Basic
mbconst.bas	MotionBasic constants

.NET

Xemodll.vb	Function declaration for Microsoft Visual Basic
mbconst.vb	MotionBasic constants (rename mbconst.bas)

Installation

The file XemoDLL.dll or Xemo64.dll must be copied either to the working directory of your application or to the Windows system directory. (The file will be sought first in the current directory.)

If you're programming with C++, the header file Xemo DLL.h, and possibly the file mbconst.h, must be included in the source code files, and the library file Xemo DLL.lib or Xemo64.lib copied into the project.

If you're programming with Visual Basic, the file Xemo DLL.bas, and possibly the file MbConst.bas, must be added to the project.

When using VB:NET, you must rename MbConst.bas into MbConst.vb and, together with XemoDll.vb, add it to your project.

Remark

Xemos with Firmware 847 (all current models) use a new USB-ID beginning with version 5.00. For these you need Xemo-DLL version 2.50 or higher.

1.2 Important symbols in this manual

Remark

Please read passages, which are marked with this symbol, definitely. Get important information about dealing with these instructions and conditions or limits for the use of the Xemo DLL.

Tip

Learn addition facts and practical tips in sections, which are marked with this symbol.

[SYSTECxxx]

The literature abbreviation [SYSTECxxx] refers you to other manuals by Systec. See the bibliography in Chap. 6.

2 Syntax

2.1 A few notes on nomenclature

Comment With regard to programming languages, program libraries, and controllers, terms such as Functions, Procedures, Macros, Subroutines as well as Commands, Instructions and Statements appear over and over again. The terms Function, Procedure, Macro and Subroutine generally refer to parts of a program within a program which perform certain tasks. When such a task (subroutine) is activated, the terms Statement, Instruction or Command are used.

Basic In the programming language Basic, as in MotionBasic, one distinguishes between subprocedures and function procedures. A 'subprocedure' is a part of a program which executes a certain task, often on the basis of a certain set of parameters. A 'function procedure' defines a part of a program which calculates a value and returns the value as a function. Further information on this subject can be found in the MotionBasic manual **[SYSTEC717]**.

Pascal Some Basic dialects use the term "procedure" as a general designation for both variants (sub and function). In contrast, the programming languages Pascal and Visual Basic use the term "procedure" only to designate the variant without a return value.

C/C++ In the programming language CC++, only the term "functions" is used, regardless of whether or not a value is returned.

The DLL As regards the DLL, in this manual only the term "function" is used. Statements understood by the controller are referred to as 'Commands'.

2.2 MotionBasic commands

Functions of the DLL which correspond to MotionBasic commands of the Systec controller begin with the prefix "MB_", followed by the name of the function as described in the MotionBasic manual. The prefix and the first letter of the MotionBasic command are always written in upper-case. Within MotionBasic itself, commands are not case-sensitive (i.e. one does not distinguish between upper- and lower-case letters). In the case of most Windows programming languages, however, syntax is case-sensitive. In case of doubt, the correct notation can be taken from the corresponding declaration file.

Example

```
MB_Amove (0,1000)
```

2.3 Parameters

Within MotionBasic there are commands with a variable number of parameters, commands with so-called "named" parameters, as well as implicit commands which have different effects depending on notation.

In contrast, the Xemo DLL allows only explicit functions with unnamed parameters and a constant number of parameters.

So, for instance, in the MotionBasic programming manual **[SYSTEC717]**, the implicit statements SET /GET – used for writing and reading system parameters – are described. In MotionBasic, depending on notation, the system parameters, the axis parameters or the I/O parameters (I/O= input or output) are accessed. The Xemo DLL, however, allows only explicit statements. Consequently, the DLL has additional functions for programming axis parameters. These functions do not appear in the MotionBasic language definition.

Examples

```
MB_SET (_FIFOMarker, _     'Program system parameter
1000)
MB_ASET (0, _Speed, 2000)  'Program axis parameter
```

2.4 Local functions

All functions which perform a task within the DLL (e.g. the initialization of the serial interface) are marked for easy recognition with the prefix 'ML_'.

Example

```
ML_IniCom (1, 19200)
```

3 Scope of functionality

You can find all functions and procedures provided by the DLL clearly arranged in the reference section of this document. They are also provided in the corresponding declaration file ("*.h" for Visual C++, "*.bas" for Visual Basic).

3.1 A look at the Xemo commands

The extensive MotionBasic programming manual **[SYSTEC717]** provides all commands for motor motion, setting and reading inputs and outputs, programming and reading system parameters, as well as the terminal functions from the DLL.
On the other hand, control instructions (If..Then, While.., etc.), arithmetic functions (Sin, Cos, Abs, etc.), and the use of variables are not supported.

A short overview of all commands is presented in the following table. Specific information regarding the functioning of these commands is found is the reference section of the MotionBasic manual.

	Controller commands	
System parameters	`Get, Aget, IoGet, Set, Aset, IoSet`	Set and read

Inputs and outputs	`In, Out, Rout, Sout, Waitinp,`
	`Inw, Outw, Routw`
		Set and read

CAN-Bus	`SdoRcv, SdoTrm`	Control external devices via CAN-Bus

Control of individual axes	`Jog`
	`Amove, Rmove`
	`Home`
	`Stop`
	`Still`
	`Busy`
		Velocity mode
		Positioning
		Reference run
		Stop axis
		Await standstill
		Query state of an axis

Path control	`Lin, Lin0, Lin1`
	`Circle, Arc, Arcc, Arcw`
		Linear interpolation
		Circular interpolation

Time functions	`Delay, Still`	Delay time

Terminal	`Print, Printxy, Cpos, Ctype`
	`Textattrib, Cls, Cleol`
	`Keystate, Keypressed`
	`Keyread, Keyclear`
	`Keyled`
		Print out text
		Keyboard entry

Subroutines	`Call`	Call up subroutine

System control	`GetState`
	`ResErr`
	`SysCtrl`
	`SetFIFO`
		State query
		Delete error
		Interrupt, quit, reset, restart
		Manipulate online FIFO

3.2 Time functions

When using the MotionBasic time functions Delay, Still and Waitinp, please note that these do not cause any delay within the DLL. These are simply normal FIFO commands and are immediately transferred to the Xemo controller, if there is space in the FIFO. Only if the FIFO should be full are they delayed there until there is space for a new command in the FIFO.

Delay time

Instead, the delay time (Delay) and/or waiting for an event (Still, Waitinp) takes place in the controller's interpreter. Not until the delay time has lapsed will the next commando be loaded from the FIFO and executed.

3.3 Transfer of strings

Some functions of the DLL return a text string. As a rule, the DLL provides two variants for this. The first variant delivers a pointer to the text string in the return value, the other variant expects an argument as a reference to a target string and copies the return string in that target. The first variant cannot be used in Visual Basic applications. Although Visual Basic supports function procedures, this procedure cannot be reflected in a DLL function. For that reason, the second function is intended for Visual Basic applications with which a prepared string variable is completed.

Visual Basic Example GetDllVersion

```
Public Function Xemo_DllVersion() As String
 Dim Version As String * 20
 ML_GetDllVersion (Version , 20)
 Xemo_DllVersion = _
  Left(Version, InStr(Version, Chr$(0)) - 1)
End Function
```

In this example, a Visual Basic function procedure is implemented which delivers the version of the Xemo DLL in the form of a string in the return value. Within the function a string with a certain length is prepared, which is then completed by the DLL function. Subsequent to that, the null sign which the DLL function has added as the string-end sign must be eliminated and the string modified to the proper length.

3.4 Parameter values for the trajectory commands

3.4.1 Bit mask of the relevant coordinates

In the case of some trajectory commands, i.e. MB_Arc and MB_Lin, you transfer the relevant coordinates, this means the names of the axes, which are assigned to the coordinate system and controlled via trajectory commands, by means of a bit mask.

Example

To assigne the axes 3 to 5 to the coordinate system and to control them via trajectory commands, you transfer the value 56 to the Xemo controller.
You get the value via the bit mask as follows:
3,4,5 = 111000 = 56.

3.4.2 Array of the target coordinates

To set the array of the target coordinates for the example above you can use a three-dimensional array.
The order of the coordinates corresponds to the order of the bit mask.

3.5 FIFO and state query

3.5.1 The online FIFO

FIFO

The controller is outfitted with a FIFO (First-In-First-Out memory) for online commands. After reception, most commands are routed via the serial interface to this online FIFO. The commands are then read within the controller and then run in the order in which they were received. The execution of a command proceeds independently of internally active programs in a separate task. Online commands can thus be executed simultaneously with programs, or tasks, running internally. With the help of the FIFO, long sequences of commands can be transmitted to the controller in a short time. The control computer does not have to wait for the execution of each individual command. Commands are executed in the controller without any “pause for thought” as long as there are commands in the FIFO. That saves unnecessary delays.

State byte

The current state of the FIFO can be queried in the state byte with the MB_GetState function. (See also GetState command in the reference section of the MotionBasic programming manual **[SYSTEC717]**). As long as the FIFO state does not show a full FIFO, a complete command can always be transmitted. If, despite a full FIFO, a FIFO command is sent to the controller, the command is ignored, and an error message is generated by the controller.

	FIFO controller commands	
System parameters	`Set, Aset, IoSet`	Assign

Inputs and outputs	`Out, Rout, Sout,`
	`Inw, Outw, Routw`
		Set and read

Control of individual axes	`Jog`
	`Amove, Rmove`
	`Home`
	`Still`
		Velocity mode
		Position
		Reference run
		Await standstill

Trajectory control	`Lin, Lin0, Lin1`
	`Circle, Arc, Arcc, Arcw`
		Linear interpolation
		Circular interpolation

Time functions	`Delay, Still, Waitinp`	Delay time

Direct commands

There are also many commands which are not directed via the online FIFO but instead are executed directly. Among those are commands which directly influence the FIFO itself, as well as the read-out of system parameters. In the reference section of this documentation you will find notes which tell you if a command is a FIFO or a direct (non-FIFO) command.
The following table contains a summary of all direct controller commands.

	Direct controller commands	

System control	`GetState`
	`ResErr`
	`SysCtrl`
	`SetFIFO`
		State query
		Delete error
		Interrupt, quit, reset, restart
		Manipulate online FIFO

Subroutine	`Call`	Call up subroutine
System parameters	`Get, Aget, IoGet`	Read system parameters
Inputs and outputs	`In, Inw, Rout, Routw`	Read inputs and outputs
Control of individual axes	`Busy` `Stop`	Query state of an axis Stop
Terminal	`Print, Printxy, Cpos, Ctype Textattrib, Cls, Cleol,Keyled`	Print out
	`Keystate, Keypressed Keyread, Keyclear`	Keyboard entry
CAN-bus	`SdoRcv, SdoTrm`	Control device via CAN-bus

3.5.2 The state query

The MB_GetState, used to query system state, is of particular importance for communication with the controller via the serial interface. This command returns the state of the Online FIFO, the error state and the state of the axes.
Detailed descriptions of the GetState command and the system parameters _State are found in the reference section of the MotionBasic programming manual [**SYSTEC717**].

Delay time

Before FIFO commands are transmitted, the DLL automatically queries the state of the controller with the MB_GetState command. If the FIFO is full, the transmission is delayed until there is free space in the FIFO. Simultaneously, the error bit is checked and an error recovery is carried out. (See error recovery.)
When the FIFO is full, depending on which command is presently being executed by the controller, there may be a substantial delay before a subsequent command can be transmitted. Although the Xemo DLL is capable of multithreading, this could lead to a blockage of user inputs

in the Windows application, or hinder the cyclical display of, for instance, axis positions. MotionBasic and the Xemo DLL provide a number of solutions to this problem:

1. FIFO state

The application can check the state of the FIFO preceding the transmission of every command. For this purpose, the following DLL function is provided:
ML_FIFOFull Queries FIFO state and returns TRUE as long as the FIFO is full.

2. FIFO memory

If a series of commands is to be transmitted, the application can check the amount of remaining memory in the FIFO with the system parameter _FIFOLeft. This allows you to determine how many commands can be transmitted without a further state query.
MB_Get(_FIFOLeft) Returns the amount of free memory in the FIFO

3. FIFO-Marker

The system state command (MB_GetState) also checks whether or not the FIFO has exceeded a defined limit. , or pointer. This limit is programmed with the system parameter FIFOPointer

4. Multithreading

The Xemo DLL is capable of multithreading (see the Multithread applications chapter in this documentation). As long as a DLL function is waiting for the FIFO, other threads are permitted which can interrupt the presently waiting DLL function, however not with a further FIFO function, as this must likewise wait for the FIFO to free up.
You could, for example, set up two threads; a thread which transfers the continuous positioning data to the controller's FIFO, and another thread which displays the actual positions of the axes in cyclical intervals.

5. Call-back

It is also possible to provide the DLL with a pointer to a function which is then called up (Call-back) whenever the FIFO is full during the execution of a FIFO command and there is therefore time for other tasks. This application-specific call-back function makes it possible, for instance, to update the display of the axis positions, or to react to a cancellation performed by the user. However, this possibility only functions in programming languages which support function pointers.

Example

```
ML_FIFOIdle (MyIdleFunction)
```

3.6 Multithread applications

The Xemo DLL is capable of multithreading. The functions of the Xemo DLL can be simultaneously called up by a number of threads without any further measures being necessary. The Xemo DLL takes care of the

correct sequencing of data, so that the transfer of data to the Xemo controller always occurs in the proper sequence. If, for example, a DLL function is transferring data from another thread and is interrupted by the call-up of an additional DLL function, this new function will be held in line until the initial transfer of data has been completed.
You can, for example, initiate two threads; one which transfers the continuous positioning date to the controller's FIFO, and a second one which displays the actual positions of the axes in cyclical intervals.

Multicontroller

In cases when a number of Xemo controllers are addressed via multiple interfaces, the correct sequence of data is guaranteed in a multithread application (see ML_ComSelect).

Remark

Correct sequencing is only possible in conjunction with MotionBasic functions, as only these identify the start and conclusion of a complete data transfer. In special cases, if elementary communication functions (e.g. ML_Putchar) are used for direct data transfer, the correct data sequence in a multithread application must be guaranteed, if necessary by use of a locking mechanism.

3.7 Error correction

When using the Xemo DLL, you need to distinguish between two different error sources: errors which occur within the DLL, and errors which occur in the controller.

Error sources

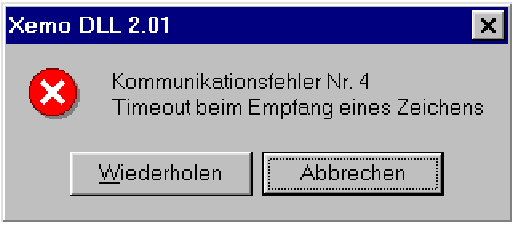
Errors within the DLL arise because of problems with the serial communication and through timeouts. Errors which occur in the controller are recognized by the DLL only when the system state (GetState) is queried. The system state is automatically queried every time a FIFO command is transferred.
An exact description of the controller's error codes can be found in the chapter “Error correction” in the MotionBasic programming manual **[SYSTEC717]**.

3.7.1 Standard error correction

To facilitate especially simple operation of the Xemo DLL, this contains a standard routine for error correction. This standard error correction is pre-configured and does not need to be explicitly initialized. An application which integrates Xemo DLL needs do nothing more than to query the error state at certain intervals with ML_GetErrState. If this has assumed a value of “-1” (ERR_CANCEL), an error has occurred and operation should be interrupted.
In case of an error, an error code is entered into the ErrCode as a static variable and the error state is set at ERR_COM_PENDING or ERR_XEMO_PENDING. After that, the error correction routine is called up which displays the error code and the corresponding error text in a modal notification window.

Communication error

In case of a communication error, for example, the following notification window opens.



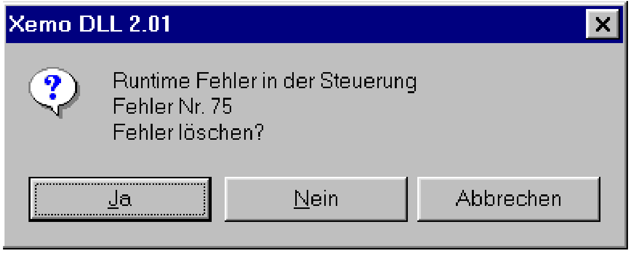
Xemo DLL 2.01

Kommunikationsfehler Nr. 4
Timeout beim Empfang eines Zeichens

If the user clicks on the button “Cancel”, the error state is set to (ERR_CANCEL) to show that the user wishes to break off. As a result, all further commands sent to the Xemo controller by the Xemo DLL are ignored and will not be executed.
In contrast, if the button Repeat is clicked, the error state will merely be erased. The original error code can, however, still be queried with ML_GetErrCode

Xemo runtime error

In case of a runtime error within the Xemo controller, the error code will be displayed in the following modal notification window.



Xemo DLL 2.01

Runtime Fehler in der Steuerung
Fehler Nr. 75
Fehler löschen?

Just as in the case of a communication error (see above), clicking the button “Cancel” will cause the Xemo DLL to ignore all further commands to the Xemo controller and they will not be executed. Further, the error state will be set at (ERR_CANCEL) to show that the user wishes to break off.
In contrast, clicking on the Yes button will delete the error in the Xemo controller. The command MB_ResErr will then be sent to the Xemo controller. The error description will now contain the error code (ERR_XEMO).
If the button No is clicked on, the error in the Xemo controller will not be deleted. Again, the error description will then contain the error code (ERR_XEMO) as well.

Error query

The error code and state of the Xemo DLL can be queried with the following DLL functions:

Example

```
ML_GetErrCode()                 'Error code query
ML_GetErrState()                'Error state query
```

Errors which occur within the Xemo controller might have to be queried specially.

`ML_LastRunErr()`	`Query of the Xemo controller's last runtime error as recorded by the standard error routine.`
`MB_Get (_ErrNo)`	`Query of the runtime error code in the Xemo controller`

Error clearing

The error state can be cleared with

Example

```
ML_SetErrState (NO_ERR)     'DLL error state clearing
```

In addition, the error state is cleared every time the initialization routines ML_IniCom or ML_IniUsb are called up.

`MB_ResErr`	`Clear error state within the Xemo controller`

Remark

As long as errors in the controller have not been cleared with the command MB_ResErr, no further commands from the Online FIFO will be executed. Therefore, it usually makes sense to delete all commands in the FIFO with SetFIFO(_FfClear) before clearing the errors.

Error codes

The individual error codes are described more clearly in the reference section of this manual under ML_GetErrCode. Error state is described more clearly under ML_GetErrState.

Error texts

The error notifications of the standard error correction routine are provided in German and English. If German is set in the system parameters of the operating system, German will be used. In all other cases, it will be English.

3.7.2 Application-specific error correction

Should the standard error correction contained in the Xemo DLL not be adequate for your specific application, there is always the possibility of implementing customized application-specific error correction.

Call-back

Just as when dealing with FIFO commands, an elegant method for correcting errors is use of the call-back procedure. Write a customized central routine for error correction and evaluation and enter a pointer for this routine in the Xemo DLL: The function pointer for such an error correction routine is transferred to the DLL with the function ML_ErrorCallBack

Example

```
ML_ErrorCallBack (MyErrorHandle)
```

If an error occurs within the DLL, a customized error correction routine will be called up instead of the standard one. The appropriate error code is provided as a parameter. After error correction, it is advisable – depending on the programming language used – to restart in the form of an “Exception Handling” at the appropriate location or to exit the error routine with the corresponding return value. Appropriate values for such a return value are

`NO_ERR`	`Delete the error`
`ERR_CANCEL`	`Do not delete the error. Communication with the controller will be interrupted.`

The call-back procedure has the great advantage that it is not necessary to query the error state whenever a DLL function is called up.

Error polling

Every automatic error correction can be prevented by entering the value (NULL) into the parameter “ErrorHandle”.

Example

```
ML_ErrorCallBack(0)           'No error correction
```

In this case, it is advisable to have the application software query the error state after every command sent to the Xemo controller

Tip

In Chapter 5, Application examples, you will find individual examples for C and Visual Basic which also include error correction.

3.8 Serial communication

The DLL performs the initialization of the communication interface (COM1, COM2 ... or USB), the conversion of the commands into the correct transfer format, and the transmission of data to the controller.

Transfer format

MotionBasic commands are transferred in an efficient binary format via the serial interface RS-232 to the controller. They consist of a command byte and a variable number of data bytes. For ordinary communication, no additional data protection (CRC, check sum (CRC, checksum, etc.) takes place. Further protection by means of a check sum is, however, configurable.

3.8.1 Initializing

Prior to calling up a MotionBasic command, the serial interface must be initialized once with ML_IniCom or ML_IniUsb. This initialization should only be carried out once. With ML_IniCom or ML_IniUsb, the error state is also cleared.
If the serial interface is to be re-programmed, e.g. to another baud rate, you must first run the ML_DeiniCom routine. Moreover, before quitting the program, the interface must be closed via the ML_DeiniCom routine so that it is available for other applications.

Example

```
ML_IniCom (1, 19200)       'Initialize serial inter-
                           'face and delete errors
ML_DeiniCom                'Close serial interface
```

Multicontroller

If more than one Xemo controller is connected to a PC, these can be addressed by opening a number of communications interfaces. Afterward, you can use ML_ComSelect to select the interface which is to be used for communications after that point in time.

3.8.2 Check sum

Transfer of data via the serial interface normally takes place without any additional check-sum protocol. If you desire a higher level of security for such transfers, an additional check-sum protocol can be activated. This additional check-sum protocol reduces the speed of transfer by about 10 – 20%

Example

```
ML_CsumMode (1)            'Activate check-sum
                           'protocol
```

3.8.3 Elementary communication functions

The MotionBasic DLL also contains basic functions for serial communication. These functions serve to send and receive MotionBasic commands and their parameters. The functions are also used by the DLL itself. In addition, they can also be used by the user.
The following table shows the basic functions for the serial interface.

	Serial communications functions	
Send functions	ML_PutChar	Send 1 byte
	ML_PutWord	Send one word (16 bits)
	ML_PutLong	Send one long-word (32 bits)
Receive functions	ML_GetRcvState	Receive state
	ML_GetChar	Receive one byte
	ML_GetWord	Receive 1 word (16 bits)
	ML_GetLong	Receive 1 long-word (32 bits)

3.9 Subroutines

Subroutines (sub procedures) which have been created with the MotionBasic IDE and stored in the controller can also be called up online with the help of the DLL.
In the MotionBasic environment, procedures and functions are called up by their names. Procedures which have been declared with an additional program number can also be called up online by their numbers. Any parameters can be transferred via the global parameter register.

Example

```
Sub @100 Rechteck        'Procedure within the
                         'controller
                         'Accept parameters

Dim sx,sy
 sx = Get (0)
 sy = Get (1)
 Lin _x += sx
 Lin _y += sy
 Lin _X += -sx
 Lin _Y += -sy
end sub
MB_SET (0,100)           'DLL command sequence for
MB_SET (1,200)           'transferring parameter(s)
MB_CALL (100)            'and calling up the sub-
                         'routine
```

Data exchange

A set of parameter registers within the system parameters is provided for external communication with internally-active MotionBasic programs. As is the case with other system parameters, the parameter registers can be programmed (write and read) internally within a program, as well as externally online. An external data exchange is therefore very simple to perform. With these parameter registers, parameters can also be transferred to internal procedures before these are called up.

4 The DLL reference

For better clarity, the reference is divided into functions which are used internally in the DLL ("ML_") from those which affect the controller ("MB_") i.e. put functions of the programming language MotionBasic into effect. This latter group consists of the online functions of Motion-Basic. Marks are also made to indicate whether they can be carried out via the FIFO.
After the grouping by function, the functions are listed in alphabetical order.

4.1 Overview of all functions

	DLL-internal functions		
Initialization	ML_TimeOut	Set timeouts	38
	ML_FIFOIdle	Background routine while waiting for online FIFO	27
	ML_IniCom	Initialize communication (COM, USB)	33
	ML_IniUsb	Initialize communication via USB	35
	ML_IniTCP	Initialize communication via Ethernet	34
	ML_ComSelect	Selection of communication interface	24
	ML_DeIniCom	Close all communication interfaces	25
	ML_CsumMode	Transfer protocol with/without check sum	25
	ML_DllVersion	Return version of Xemo DLL	26
	ML_GetDllVersion	Determine version of Xemo DLL	29
Error correction	ML_ErrorCallBack	Call-Back for general error correction	26
	ML_RunErrCallBack	Call-Back for errors in Xemo controller	38
	ML_GetErrCode	Query error number	30
	ML_GetErrState	Query error state	31
	ML_SetErrState	Set error state	38
	ML_LastRunErr	Query last Xemo controller runtime error	36
	ML_ComErrText	Return error text	24
	ML_GetComErrText	Determine error text	28
Data transfer	ML_FIFOFull	Check if online FIFO is full	27
	ML_PutChar	Send 1 byte via serial interface	36
	ML_PutWord	Send 1 word (16 bits) via serial interface	37
	ML_PutLong	Send 1 long word (32 bits) via serial interface	37
	ML_GetRcvState	Query state of serial interface	32
	ML_GetChar	Read 1 byte from serial interface	28
	ML_GetWord	Read 1 word (16 bits) from serial interface	33
	ML_GetLong	Read 1 long word (32 bits) from serial interface	32

MotionBasic functions

System control	MB_SysCtrl	Break, halt, reset, restart	56
	MB_GetState	Query general state	45
	MB_SetFIFO	Set online FIFO	55
	MB_ResErr	Clear all errors	53
	MB_Call	Call subroutine	43
System parameters	MB_Set	Set system parameters	55
	MB_Seti	Set system parameters	55
	MB_Aset	Set axis parameters	42
	MB_Aseti	Set axis parameters	42
	MB_IoSet	Set I/O parameters	47
	MB_IoSeti	Set I/O parameters	48
	MB_Get	Read system parameters	45
	MB_Aget	Read axis parameters	40
	MB_IoGet	Read I/O parameters	47
Individual axis control	MB_Jog	Velocity mode	48
	MB_Amove	Absolute positioning	40
	MB_Rmove	Relative positioning	53
	MB_Home	Reference run	46
	MB_Stop	Stop	56
	MB_Still	Await standstill	56
	MB_Busy	Query state of one axis	42
Path control	MB_Lin, MB_Lin0	Linear interpolation at rapid traverse velocity	50
	MB_Lin1	Linear interpolation at feed velocity	50
	MB_Circle	Circle with radius, start angle and end angle	43
	MB_Arc	Circular/helical interpolation w radius & target position	41
	MB_Arcc	Circular/helical interpol. w center point & target position	41
	MB_Arcw	Circular/helical interpolation w center point & target position	41
Inputs and outputs	MB_Out	Set outputs	50
	MB_Outi	Set outputs	51
	MB_Sout	Synchronize outputs	55
	MB_Rout	Return output read	53
	MB_In	Read input	46
	MB_Waitinp	Wait for inputs	57
	MB_Inw	Read inputs word by word (16 bit)	46
	MB_Outw	Read outputs word by word (16 bit)	51
	MB_Outwi	Read outputs word by word (16 bit)	52

	MB_Routw	Return outputs read word by word (16 bit)	54
CAN-bus	MB_SdoRcv	Receive SDO	54
	MB_SdoTrm	Send SDO	54
Text production	MB_Print	Print from current cursor position	52
	MB_Printxy	Print from position x,y	52
	MB_Cpos	Position cursor	44
	MB_Ctype	Define cursor	44
	MB_TextAttrib	Set text attributes (normal, blinking)	57
	MB_Cls	Clear screen	44
	MB_Cleol	Clear to end of line	43
Keyboard entry	MB_Keystate	State of a key	49
	MB_Keypressed	State „key pressed“	49
	MB_Keyread	Read key	49
	MB_Keyclear	Clear keyboard buffer	48
Terminal	MB_Keyled	Key LEDs on/off	49
Time functions	MB_Delay	Delay time	44
	MB_Still	Await motor standstill	56
	MB_Waitinp	Wait for input	57

4.2 DLL internal functions

This chapter presents and describes those functions which perform tasks within the DLL. These include the initialization of the serial interface, the transmission and reception of characters via the serial interface, and error correction.

ML_ComErrText

Purpose Return error text

C/C++ `const char * ML_ComErrText (short ErrCode);`
Basic `-- not available --`

Elements	Description
ErrCode	**Error code which was queried with ML_GetErrCode**
Return value	**The error text associated with that error code**

Description The error texts which are used by the standard error routines of the DLL can also be used with this function.

See also ML_GetComErrText, ML_GetErrCode

ML_ComSelect

Purpose Select the communication interfaces

C/C++ `void ML_ComSelect (short ComNo);`
Basic `Sub ML_ComSelect (ByVal ComNo as Integer)`

Elements	Description
ComNo	**Number of the communication interface (0,1,2..)**

Description If a number of Xemo controllers are connected to a PC, these can be addressed by initializing a number of communication interfaces. Subsequent to that, the interface can be selected with ML_ComSelect via which communication from that time on is to occur. The number of the interface chosen in ML_IniUsb must be provided here.

Multithread ML_ComSelect can be used in a multithread application as well. The DLL assures that communication always takes place via the specific interface which ML_ComSelect has selected with its corresponding thread.

See also ML_IniCom, ML_IniUsb

Example

```
ML_IniUsb (0, ”X170001”)     'Initialize the USB inter-
                             'face with the serial num-
                             'ber “X170001“ on channel 0
ML_IniUsb (1, ”X170009”)     'Initialize an additional
                             'USB interface on channel 1
Ml_ComSelect (0)             'Select the USB interface
                             'on channel 0
MB_rmove (0,1000)            'One or more MotionBasic
....                         'commands for this Xemo
                             'controller
Ml_ComSelect (1)             'Select the USB interface
                             'on channel 1
MB_rmove (0,1000             'One or more MotionBasic
...                          'commands for the second
                             'Xemo controller
```

ML_CsumMode

Purpose Transfer protocol with / without check sum

C/C++ `void ML_CsumMode (short Mode);`
Basic `Sub ML_CsumMode (ByVal Mode as Integer)`

Elements	Description
Mode	**Check-sum mode** **Mode = 0 Transfer protocol without check sum** **Mode = 1 Transfer protocol with check sum**

Description Standard data transfer occurs without an additional check-sum protocol. With this function, the check-sum protocol can be activated or deactivated.

See also ML_IniCom

ML_DeIniCom

Purpose Close all communication interfaces

C/C++ `void ML_DeIniCom (void);`
Basic `Sub ML_DeIniCom`

Description Before an application is exited, all communication interfaces should be closed with the routine ML_DeiniCom, so that they are again available for other applications.

See also ML_IniCom, ML_IniUsb

ML_ DllVersion

Purpose Return the Xemo DLL version

C/C++ `const char * ML_DllVersion (void)`
Basic `-- not available –`

Elements	Description
`Return value`	`Pointer to a string with the Xemo DLL version`

Description See ML_GetDllVersion

ML_ErrorCallBack

Purpose Call-back for general error correction

C/C++

```
void ML_ErrorCallBack (short (*ErrorFunc) (short
ErrorCode));
```

Basic

```
Sub ML_ErrorCallBack (ByVal ErrorFuncType as Long)
```

Elements	Description

`ErrorFunc`	`Pointer for one’s own error correction routine`
	`OR`
	`0 = no error correcton routine`
	`1 = standard error correction routine (default)`

`ErrorFuncType`	`0 = no error correction routine`
	`1 = standard error correction routine (default)`

Description With this routine, you can install an application-specific call-back routine which will be called up in case of a DLL error.
With ErrorFunc the type of error correction routine can be programmed. For application-specific error correction routines, a pointer is transferred to the corresponding error correction routine (call-back), if the value is 1, the standard error correction routine of the Xemo DLL is activated, with the value NULL, no error correction routine is active.

Call-back If the call-back process is to be used, the application-specific error correction routine must have the following form: short MyErrorFunction (short ErrorCode)
The current error code is transferred as a parameter to the function. The new error state must be delivered as the return value. The individual error codes are further described under ML_GetErrCode. The new error state can determine if the error is to be cleared of if, for example,

the process is to be abandoned. Error state is further described under ML_GetErrState.

Remark

> If an application-specific error correction routine is desired, this function should be called up prior to initialization of the serial interface so that possible errors can be dealt with during initialization.

Remark

> The call-back process functions only with those programming languages which support error pointers. In other languages (e.g. Basic), the only possibility which exists is to work with the standard error-correction routine.

See also ML_RunErrCallBack, ML_GetErrCode, ML_GetErrState

Example

```
ML_ErrorCallBack (MyErrorFunc);
```

ML_FIFOFull

Purpose Check if the online FIFO is full

C/C++

```
short ML_FIFOFull (void);
```

Basic

```
Function ML_FIFOFull () as Integer
```

Elements	Description
`Return value`	`State of the online FIFO`

Description If the online FIFO is full, this function returns a value unequal to 1

See also MB_GetState

Example C/C++

```
while (ML_FIFOFull() != 0)
 printf ("Xemo-FIFO voll");
```

ML_FIFOIdle

Purpose Set up a background routine when waiting for the online FIFO

C/C++

```
void ML_FIFOIdle (void (*FIFOIdle) (void));
```

Basic

```
- not supported -
```

Elements	Description
`FIFOIdle`	`Pointer to idle function`

Description With the parameter FIFOIdle, a customized routine can be entered which should always be called up (call-back) when the FIFO is full at the time a FIFO command is issued and, as a result, there is time for other assignments. Such an application-specific call-back function could, for example, update the display of the axes' positions on the monitor or respond to a user cancellation.

Remark

The call-back process functions only with programming languages which support function pointers.

See also ML_ErrorCallBack, ML_RunErrCallBack

ML_GetChar

Purpose Read one byte from the serial interface

C/C++
Basic

```
short ML_GetChar (void);
Function ML_GetChar() as Integer
```

Elements	Description
`Return value`	`The byte read`

Description The Xemo DLL uses this function to read all characters from the serial interface. If no byte is available for reading, a delay takes place until one becomes available.

Timeout If a timeout was programmed with the routine ML_TimeOut at the time of initialization, the timeout will not be exceeded while waiting for a byte. In such a case, the error routine will be called up with the error code ERR_RCV_TIMEOUT.

See also ML_PutChar, ML_PutWord, ML_PutLong, ML_GetRcvState, ML_GetWord, ML_GetLong

ML_GetComErrText

Purpose Determine error text

C/C++

```
void ML_GetComErrText (short ErrCode, char * ErrTxt,
short MaxLen);
```

Basic

```
Sub ML_GetComErrText (ByVal ErrCode As Integer, _
    ByVal ErrTxt As String, ByVal MaxLen As Integer)
```

Elements	Description
`ErrCode`	`Error code which was queried with ML_GetErrCode`
`ErrTxt`	`Pointer to a string in which the error text is to be entered`
`MaxLen`	`Maximum length of the error text`

Description

With this function, error texts which used by the DLL's standard error routines can also be used for customized applications.

See also

ML_ComErrText, ML_GetErrCode

Example Visual Basic

```
'This Visual Basic function provides the error text
'which belongs to the ErrCode
Public Function MB_ComErrText (ErrCode as Integer) As
String
 Dim ErrText As String * 40
 ML_GetComErrText (ErrCode, ErrText, 40)
 MB_ComErrText = Left(Version, InStr(ErrText,
Chr$(0))-1)
End Function
```

ML_ GetDllVersion

Purpose

Determine the Xemo DLL version

C/C++
Basic

```
void ML_GetDllVersion (char * Version, short MaxLen);
Sub ML_GetDllVersion (ByVal Version As String, ByVal
MaxLen As Integer)
```

Elements	Description
Version	Pointer to a string in which the DLL's version string is to be entered
MaxLen	Maximum length of the string

Description

You can determine the Xemo DLL's current version with this function. It is returned as a character string. When the DLL is delivered, its version is entered in whichever file (XemoDLL.h, xemodll.bas and/or xemoll.vb) belongs to the particular DLL version.

C/C++
Basic

```
#define XEMO_DLL_VESRION "2.16"
Public Const XEMO_DLL_VESRION = "2.16"
```

During initialization of the application, this function can be used to check if the loaded DLL corresponds to the version with which the application was generated.

Remark

To return a pointer to the version string, use the function ML_DllVersion. This possibility is only available in those languages which allow pointers.

See also

ML_DllVersion

ML_GetErrCode

Purpose Query error code

C/C++ `short ML_GetErrCode (void);`

Basic `Function ML_GetErrCode () as Integer`

Elements	Description
Return value	**The most recently registered error code**

Description

Each time an error within the Xemo DLL occurs, the corresponding error code is registered. This error code can be queried with the function ML_GetErrCode. The error code is, however, not cleared and can be queried again as often as desired.

Only with setting the state of error with the procedure ML_SetErrState the error code will be reset to 0. Therefore the new value of the error state is of no importance, the error code will be deleted.

Remark

You can query the state of an error, and through that the results of user communication, with the function ML_GetErrState.

The individual error codes are defined as follows:

ErrorCode

1	ERR_XEMO	This error is generated when an error within the controller is detected at the time the system state is being queried. In certain cases, the error code within the controller must be separately queried.
2	ERR_COM_PORT	The serial interface cannot be initialized.
3	ERR_RCV_OVERFLOW	There is an overflow in the serial interface’s receive-data buffer.
4	ERR_RCV_TIMEOUT	The programmed timeout for receiving a character has expired.
5	ERR_FIFO_TIMEOUT	The programmed timeout for the Online FIFO has expired, i.e., the maximum time during which the Online FIFO can still receive a command.
6	ERR_GETSTATE	An error has occurred during a system state query. (Getstate)
7	ERR_RCV_CMD	An error has occurred during a response from the controller.

8	ERR_TRM_TIMEOUT	The TIMEOUT expired while a character was being sent.
9	ERR_CHECKSUM	A check-sum error has occurred during data transfer.
10	ERR_COM_SELECT	An attempt was made to access an invalid or non-initialized communication interface.
11	ERR_MAX_THREADS	A maximum of only five different threads can access the DLL.
12	ERR_GET_THREAD_COM	Function used without opening COM port previously
13		Firmware file has the wrong format (only Xemo controllers with Ethernet)
14	ERR_COM_CANCELED	Communication to the Xemo canceled. The Online FIFO is no longer writable or readable.

See also ML_GetErrState, ML_ErrorCallBack

ML_GetErrState

Purpose Query error state

C/C++ `short ML_GetErrState (void);`

Basic `Function ML_GetErrState () as Integer`

Elements	Description
Return value	**The current error state**

Description

The error state gives information about the state of error management. The error state informs whether an error is present, it is being corrected at this moment, or if it is necessary to cancel. The error state can be cleared with ML_SetErrState. The error state is also cleared during initialization with ML_IniCom or ML_IniUsb.

When the automatic error handling is switched off, the error state will be automatically set to ERR_RETRY (3) with each new error.

As long as the error state is not reset with ML_SetErrState, the value of the error state will not be cleared.

The error state can be as follows:

Error state

ERR_LEFT	**-1**	A controller run-time error has been indicated by the standard error correction routine but not cleared.
NO_ERR	**0**	There are no errors at present.

`ERR_XEMO_PENDING`	`1`	An error has occurred in the Xemo controller, but it has not been corrected.
`ERR_COM_PENDING`	`2`	An error has occurred in serial communication, but it has not been corrected.
`ERR_RETRY`	`3`	Possible return value in an error correction routine. The error should be cleared and a new run started.
`ERR_CANCEL`	`4`	Possible return value in an error correction routine. The error should not be cleared but cancelled instead. No further data will be sent to the Xemo controller.

Remark

When applying the standard error-correction routine, (see ML_ErrorCallBack, only the error state codings listed here with the meanings described are possible. You are completely free to set up your own error correction in accordance with the call-back procedure, but you must provide for the correct error treatment yourself.

See also ML_SetErrState, ML_ErrorCallBack

ML_GetLong

Purpose Read one long-word (32 bits) from the serial interface

C/C++

```
long ML_GetLong (void);
```

Basic

```
Function ML_GetLong() as Long
```

Elements	Description
`Return value`	**`The 32-bit long word read`**

Description With the help of the function ML_Getchar, four bytes are read from the serial interface and combined to form one long-word.

See also ML_PutChar, ML_PutWord, ML_PutLong,
ML_GetRcvState, ML_Getchar, ML_GetWord

ML_GetRcvState

Purpose Query state of the serial interface

C/C++

```
short ML_GetRcvState (void);
```

Basic

```
Function ML_GetRcvState() as Integer
```

Elements	Description
`Return value`	**`State of the serial interface`**

Description

This function provides a nonzero value as soon as a character is in the receiver buffer.

See also

ML_PutChar, ML_PutWord, ML_PutLong,
ML_Getchar, ML_GetWord, ML_GetLong

ML_GetWord — Read one word (16 bits) from the serial interface

C/C++

Basic

```
short ML_GetWord (void);
Function ML_GetWord() as Integer
```

Elements	Description
Return value	**The 16-bit word read**

Description

With the help of the function ML_Getchar, two bytes are read from the serial interface and combined to form one word.

See also

ML_PutChar, ML_PutWord, ML_PutLong,
ML_GetRcvState, ML_Getchar, ML_GetLong

ML_IniCom

Purpose

Initialize communication (COM, USB)

C/C++

Basic

```
short ML_IniCom (short ComNo, long Baud);
Function ML_IniCom (ByVal ComNo as Integer , ByVal
Baud as Integer) As Integer
```

Elements	Description
ComNo	Number of the communication interface (0 = USB, 1 = COM1, 2 = COM2)
Baud	Baud rate, e.g. 9600 for 9.600 bauds
Return value	Error code

Description

Prior to calling up a MotionBasic command, the communication interface must be initialized once with ML_IniCom.
One of the RS232 interfaces (COM1, COM2, …) or the USB interface can be selected. The USB interface is selected with ComNo = 0. In this case, the value for the baud rate is irrelevant. The first available USB interface to which the Xemo controller is connected is opened. If a number of Xemo devices are connected via USB, it cannot be clearly predicted which of the connected devices will be selected. In this case, the appropriate USB interface should be opened with ML_IniUsb.
If the communication interface or parameters (baud rate) are to be changed during the application, all communication interfaces must be closed with ML_DeiniCom prior to a renewed call-up of ML_IniCom.

The interface should also be closed with ML_DeiniCom prior to ending the application, so that they are again available for other applications. In case an error occurs during the initialization, the error routine is called up with the error code ERR_COM_PORT. Further, the error code will be returned in the return value.

Remark

Any previous error state still present will be cleared with ML_IniCom.

See also

ML_IniUsb, ML_DeIniCom, ML_ComSelect

Example

```
'The serial interface COM1 will be opened with 9,600
'baud.
ML_IniCom (COM1, 9600)

'The first available USB interface to which a Xemo
'controller is connected will be opened.
ML_IniCom (0,0)
```

ML_IniTCP

Purpose

Initialize communication via Ethernet

C/C++
Basic

```
short ML_IniTCP ( const char * IpAddr, long port);
Function ML_IniTCP (ByVal port as Integer , ByVal
IpAddr As String) As Integer
```

Elemente	Beschreibung
`IpAddr`	`String that contains an IP address, either in point notation "192.168.1.204" Or an alias like „www.systec.de/XemoCtrl“`
`port`	`TCP port address, must be set for Xemo on 502.`
`Return value`	Error code

Description

As an alternative to the traditional functions ML_IniCom and ML_IniUsb, an Ethernet interface associated with the IP address to communicate with the Xemo control can be opened with the function ML_IniTCP. If no Xemo controller with the specified IP address is attached, the error routine will be called up with the error code ERR_COM_PORT and the error code will be returned as the return value.
With the port parameter, you specify the TCP port address. Only the value of 502 is allowed.

Remark

If you want to operate several Xemo controls over Ethernet via one Ethernet interface, they must have different IP addresses. Delivered all Xemo controls have the same IP address. You can change the IP address in case of need via the program XemoUpdate.exe. More information is described in the instruction manual of the MotionBasic IDE **[SYS-TEC875]**.

Tip

Any previous error state still present will be cleared with ML_IniTCP.

See also

ML_IniCom, ML_DeIniCom, ML_ComSelect, ML_IniUsb

Example

```
'The Ethernet interface of the Xemo controller with IP
'address "192.168.1.204" is opened.
IniTCP („192.168.1.204“,502)
```

ML_IniUsb

Purpose

Initialize communications via USB

C/C++

```
short ML_IniUsb (short ComNo, const char * SerialNo);
```

Basic

```
Function ML_IniUsb (ByVal ComNo as Integer , ByVal
SerialNo As String) As Integer
```

Elements	Description
ComNo	Any number (between 0 and 9) for the later selection of the communication interface.
SerialNo	Text string with the USB series number of the connected Xemo controller.
Return value	Error code

Description

As an alternative to the standard function ML_IniCom, the function ML_IniUsb can be used in combination with a USB interface's device series number to initialize that device for communication with the Xemo controller. If no Xemo controller is connected to that series number, the error routine will be called up with the error code ERR_COM_PORT and the error code will be returned as the return value.

With the parameter ComNo, an arbitrary number between 0 and 9 will be provided for the later selection of a specific communication interface. This is necessary when a number of Xemo controllers are to be addressed via a number of interfaces. See also ML_ComSelect.

Xemo USB driver

In order to open a USB interface, you must first install the Xemo USB driver. The Xemo USB driver installation is done automatically with the installation of the MotionBasic IDE.

Tip

Since each Xemo has its own serial number, several Xemo controllers can be operated together via USB through one USB interface.
You can find the serial number of controlling Xemo on the label; for Xemo controllers with Ethernet port you can read the serial number also via MB_Get command (parameter 1003 _SerialNo).

Tip

Any previous error state still present will be cleared with ML_IniUsb.

See also

ML_IniCom, ML_DeIniCom, ML_ComSelect

Example

```
'The USB interface with the series number „12345678“
'connected to a Xemo controller will be initialized.
ML_IniUsb (0, ”12345678”)
```

ML_LastRunErr

Purpose

Query the last Xemo controller runtime error

C/C++

```
short ML_LastRunErr (void);
```

Basic

```
Function ML_LastRunErr () as Integer
```

Elements	Description
Return value	**The most recent Xemo controller runtime error code entered by the standard error routine.**

Description

For every FIFO command and in each case of a MB_GetState call-up to query the Xemo controller system state, the DLL also queries the error bit and, in case of an error, calls up the error correction. If the standard error correction for runtime errors (see also ML_RunErrCallBack) is active, the error is read out of the controller by means of MB_Get (_ErrNo) and displayed in a notification window.
The error which is read out of the controller is stored and can be read out by ML_LastRunErr at any time.

See also

ML_RunErrCallBack, ML_ErrorCallBack, ML_GetErrCode, ML_GetErr-State

ML_PutChar

Purpose

Send one byte via the serial interface

C/C++
Basic

```
void ML_Putchar (short Chr);
Sub ML_Putchar (ByVal Chr as Integer)
```

Elements	Description
`Chr`	`The byte to be sent`

Description

The Xemo DLL uses this function to transfer one byte via the serial interface.

See also

ML_PutChar, ML_PutWord, ML_PutLong,
ML_GetRcvState, ML_Getchar, ML_GetWord, ML_GetLong

ML_PutLong

Purpose

Send one long-word (32 bits) via the serial interface.

C/C++
Basic

```
void ML_PutLong (long Lword);
Sub ML_PutLong (ByVal Lword as long)
```

Elements	Description
`Lword`	`The 32-bit long word to be sent`

Description

A long-word parameter is divided into four bytes and sent in the right sequence sent by ML_PutChar via the serial interface.

See also

ML_PutChar, ML_PutWord,
ML_GetRcvState, ML_Getchar, ML_GetWord, ML_GetLong

ML_PutWord

Purpose

Send one word (16 bits) via the serial interface

C/C++
Basic

```
void ML_PutWord (short Word);
Sub ML_PutWord (ByVal Word as Integer)
```

Elements	Description
`Word`	`The 16-bit word to be sent`

Description

A word parameter is divided into two bytes and sent in the proper sequence by ML_PutChar via the serial interface.

See also

ML_PutChar, ML_PutLong,
ML_GetRcvState, ML_Getchar, ML_GetWord, ML_GetLong

ML_RunErrCallBack

Purpose Call-back for correction of Xemo controller errors

C/C++
Basic

```
void ML_RunErrCallBack (short (*ErrorFunc) (void));
Sub ML_RunErrCallBack (ByVal ErrorFuncType as Long)
```

Elements	Description
`ErrorFunc`	Pointer for one's own error correction routine OR 0 = no error correction routine **`1 = standard error correction routine (default)`**
`ErrorFuncType`	0 = no error correction routine **`1 = standard error correction routine (default)`**

Description Whenever the function MB_GetState is called up to query the system state of the Xemo controller, the error bit is also queried by the DLL and, in case of an error, the error correction is called up.
For correcting errors which occur in the Xemo controller, a separate error correction routine can be installed. You can find additional information on this under ML_ErrorCallBack.

See also ML_ErrorCallBack, ML_GetErrCode, ML_GetErrState

ML_SetErrState

Purpose Clear or re-set the error state

C/C++
Basic

```
void ML_SetErrState (short State);
Sub ML_SetErrState (ByVal State as Integer)
```

Elements	Description
`State`	**`The new error state`**

Description With this function, the error state can be re-set. Appropriate values fort he new state are:

NO_ERR	Clear the error
ERR_CANCEL	Do not clear the error. Communication with the controller will be cancelled.

See also ML_GetErrState

ML_TimeOut

Purpose (Re-)Set timeouts new

C/C++

```
void ML_TimeOut (double ComTimeout, double
FIFOTimeout);
```

Basic

```
Sub ML_TimeOut (ByVal ComTimeout as Double, ByVal
FIFOTimeout as Double)
```

Elements	Description
`ComTimeout`	`Timeout for receiving and sending characters`
`FIFOTimeout`	`Timeout for the online FIFO`

Description

Timeout times (in seconds) are (re-) programmed
ComTimeout sets the maximum time in seconds for receiving and transmitting a character. If the time is exceeded, the error routine is called up with the error code ERR_RCV_TIMEOUT or ERR_TRM_TIMEOUT.

FIFOTimeout provides the maximum time in seconds at the end of which the Online FIFO can again receive a command. If this time is exceeded, the error routine is called up with the error code ERR_FIFO_TIMEOUT.
If no timeout is desired, the corresponding parameter should be set at 0

Default

ComTimeout is preset at 1 second. The FIFOTimeout is set at null, i.e. no FIFO timeout is active.

Remark

It is normally not necessary to program timeouts; they are only necessary when particular situations are to be dealt with.

See also

ML_FIFOIdle, ML_ErrorCallBack

Example

```
ML_TimeOut (1, 5);
```

4.3 The Xemo DLL functions

This chapter described those functions which correspond to a MotionBasic command of the Systec controller. Among those are the commands for moving motors, for setting and reading inputs and outputs, for programming and return of system parameters as well as the terminal functions and communication via the CAN interfaces.
The DLL function creates the correct transmission format and attends to the transfer of data to the controller. Functions expecting a return value wait for the answer from the controller.
It is noted in each command whether it is carried out via the FIFO (see also online FIFO).

Remark

Except for the function MB_GetState, only the format of the DLL functions is described. The extensive description of all MotionBasic commands can be found in the reference section of the MotionBasic programming manual **[SYSTEC717]**.

MB_AGet

Purpose Read an axis parameter

C/C++

```
long MB_AGet (short Axis, short Parameter);
```

Basic

```
Function MB_AGet (ByVal Axis as Integer, ByVal
Parameter as Integer) as Long
```

Elements	Description
Axis	Number of the specific axis
Parameter	Number of the axis parameter
Return value	Value of the axis parameter

FIFO ☐

MB_Amove

Purpose Absolute positioning of axis

C/C++

```
void MB_Amove (short axis, long target position);
```

Basic

```
Sub MB_Amove (ByVal axis as Integer, ByVal target
position as Long)
```

Elements	Description
Axis	Number of the specific axis
Target position	Position to which the axis should run

FIFO X

MB_Arc

Purpose Circular interpolation with radius and target position

C/C++
```
void MB_Arc (unsigned C_Mask, long Radius, long
Coordinate[]);
```

Basic
```
Sub MB_Arc (ByVal C_Mask as Integer, ByVal Radius as
Long, ByRef Coordinate() as Long)
```

Elements	Description
C_Mask	Bit mask of the relevant coordinates Bit mask of the relevant coordinates
Radius	Radius of the arc
Coordinate	Pointer to the array of the target coordinates

FIFO	X

MB_Arcc

Purpose Circular interpolation with center point and target position counter-clockwise

C/C++
```
void MB_Arcc (unsigned C_Mask, long Mx, long My, long
Coordinate[]);
```

Basic
```
Sub MB_Arcc (ByVal C_Mask as Integer, ByVal Mx as
Long, ByVal My as Long, ByRef Coordinate() as Long)
```

Elements	Description
C_Mask	Bit mask of the relevant coordinates
Mx	X-Coordinate of the center point
My	Y-Coordinate of the center point
Coordinate	Pointer to the array of the target coordinates

FIFO	X

MB_Arcw

Purpose Circular interpolation with center point and target position clockwise.

C/C++
```
void MB_Arcw (unsigned C_Mask, long Mx, long My, long
Coordinate[]);
```

Basic
```
Sub MB_Arcw(ByVal C_Mask as Integer, ByVal Mx as Long,
ByVal My as Long, ByRef Coordinate() as Long)
```

Elements	Description
`C_Mask`	`Bit mask of the relevant coordinates`
`Mx`	`X-Coordinate of the center point`
`My`	`Y-Coordinate of the center point`
`Coordinate`	`Pointer to the array of the target coordinates`

FIFO ☒

MB_Aset

Purpose Set (program) an axis parameter

C/C++

```
void MB_ASet (short axis, short parameter, long
value);
```

Basic

```
Sub MB_ASet (ByVal axis as Integer, ByVa parameter as
Integer, ByVal value as Long)
```

Elements	Description
Axis	Number of the specific axis
Parameter	Number of the parameter
Value	Value of the parameter

FIFO ☒

MB_Aseti

Purpose Set (program) an axis parameter

C/C++

```
void MB_ASet (short axis, short parameter, long
value);
```

Basic

```
Sub MB_ASet (ByVal axis as Integer, ByVa parameter as
Integer, ByVal value as Long)
```

Elements	Description
Axis	Number of the specific axis
Parameter	Number of the parameter
Value	Value of the parameter

FIFO □

MB_Busy

Purpose Query state of an axis

C/C++ `short MB_Busy (short Axis);`

Basic `Function MB_Busy (ByVal Axis as Integer) as Integer`

Elements	Description
`Axis`	`Number of the specific axis`
`Return value`	`State of the specific axis`

FIFO □

MB_Call

Purpose Call up a subroutine

C/C++ `void MB_Call (unsigned ProgNr);`

Basic `Sub MB_Call (ByVal ProgNr as Integer);`

Elements	Description
`ProgNr`	`Number of the subroutine`

FIFO ☒

Remark Only procedures which have been declared with additional program numbers can also be called up online with their numbers. Parameters can be transferred via the global parameter register.

MB_Circle

Purpose Arc with radius, start angle and end angle

C/C++

```
void MB_Circle (long Radius, long start angle, long
end angle);
```

Basic

```
Sub MB_Circle (ByVal Radius as Long, ByVal start angle
as Long, ByVal end angle as Long)
```

Elements	Description
Radius	Radius of the arc
Start angle	Start angle in 1/100 degrees
End angle	End angle in 1/100 degrees

FIFO ☒

MB_Cleol

Purpose Clear to the end of the line

C/C++ `void MB_Cleol (void);`

Basic `Sub MB_Cleol`

FIFO □

MB_Cls

Purpose Clear the monitor screen

C/C++ `void MB_Cls (void);`
Basic `Sub MB_Cls`

FIFO □

MB_Cpos

Purpose Position the cursor

C/C++ `void MB_Cpos (short x, short y);`
Basic `Sub MB_Cpos (ByVal x as Integer, ByVal y as Integer)`

Elements	Description
x	**Column position of the cursor**
y	**Line position of the cursor**

FIFO □

MB_Ctype

Purpose Define the kind of cursor

C/C++ `void MB_Ctype (short TypeNr);`
Basic `Sub MB_Ctype(ByVal TypeNr as Integer)`

Elements	Description
TypeNr	**Kind of cursor**

FIFO □

MB_Delay

Purpose Delay time

C/C++ `void MB_Delay (long delay);`
Basic `Sub MB_Delay (ByVal delay as Long)`

Elements	Description
`Delay`	`Delay time in milliseconds`

FIFO ☒

MB_Get

Purpose Read a system parameter

C/C++ `long MB_Get (short Parameter);`
Basic `Function MB_Get (ByVal Parameter as Integer) as Long`

Elements	Description
`Parameter`	`Number of the parameter`
`Return value`	`Value of the parameter`

FIFO □

MB_GetState

Purpose Query the Xemo state

C/C++ `short MB_GetState (void);`
Basic `Function MB_ GetState () as Integer`

Elements	Description
`Return value`	`System state`

FIFO □

Description The function MB_GetState reads the system parameter _state from the controller. The system parameter contains the most important general state information of the controller. It gives information about the state of the online FIFO, error state, and the state of the axes. Because the system state is important for communication with the controller, this parameter has its own query function to provide a higher level of efficiency.
Before FIFO commands are transferred, the DLL automatically queries the controller’s state with the command MB_GetState. If the FIFO is full, the transfer will be delayed until space has become free again in the FIFO. At the same time, the error bit in the state is evaluated and error correction is carried out. (See the chapter Error Correction in this documentation.)
An exact description of the command GetState and the system parameter _State is to be found in the reference section of the MotionBasic programming manual [SYSTEC717].

Remark

By regularly calling up the function MB_GetState, you can implement an effective automatic error correction, as MB_GetState always concurrently queries the error bit and then calls up the error routine with the error code ERR_XEMO.

See also ML_FIFOFull

MB_Home

Purpose Reference axis

C/C++
```
void MB_Home (short Axis);
```
Basic
```
Sub MB_Home (ByVal Axis as Integer)
```

Elements	Description
`Axis`	`Number of the specific axis`

FIFO ☒

MB_In

Purpose Read inputs

C/C++
```
short MB_In (short Byteno, short Bitno1, short
Bitno2);
```
Basic
```
Function MB_In (ByVal ByteNo as Integer, ByVal BitNo1
as Integer, ByVal BitNo2 as Integer) as Integer
```

Elements	Description
`ByteNo`	`The byte address`
`BitNo1`	`1. Bit number within the byte adddress`
`BitNo2`	`2. Bit number within the byte address`
`Return value`	`State of the input and/or inputs`

FIFO □

MB_Inw

Purpose Read input word by word (16 bits)

C/C++
```
short MB_Inw (short ByteNo);
```
Basic
```
Function MB_Inw (ByVal Byteor as Integer) as Integer
```

Elements	Description
ByteNo	The byte address
Return value	State of the input(s)

FIFO □

MB_IoGet

Purpose Read an I/O parameter

C/C++

```
short MB_IoGet (short ByteNo, short BitNo1, short
BitNo2, short Parameter);
```

Basic

```
Function MB_IoGet (ByVal ByteNo as Integer, ByVal
BitNo1 as Integer, ByVal BitNo2 as Integer, ByVal
Parameter as Integer) As Integer
```

Elements	Description
ByteNo	The byte address
BitNo1	1. Bit number within the byte address
BitNo2	2. Bit number within the byte address
Parameter	Number of the I/O parameter
Return value	Value of the I/O parameter

FIFO □

MB_IoSet

Purpose Set (program) an I/O parameter

C/C++

```
void MB_IoSet (short ByteNo, short BitNo1, short
BitNo2, short Parameter, short Value);
```

Basic

```
Sub MB_IoSet (ByVal ByteNo as Integer, ByVal BitNo1 as
Integer, ByVal BitNo2 as Integer, ByVal Parameter as
Integer, ByVal Value as Long)
```

Elements	Description
ByteNo	The byte address
BitNo1	1. Bit number within the byte address
BitNo2	2. Bit number within the byte address
Parameter	Number of the I/O parameter
Value	New value of the I/O parameter

FIFO ☒

MB_IoSeti

Purpose Set (program) an I/O parameter

C/C++

```
void MB_IoSet (short ByteNo, short BitNo1, short
BitNo2, short Parameter, short Value);
```

Basic

```
Sub MB_IoSet (ByVal ByteNo as Integer, ByVal BitNo1 as
Integer, ByVal BitNo2 as Integer, ByVal Parameter as
Integer, ByVal Value as Long)
```

Elements	Description
ByteNo	The byte address
BitNo1	1. Bit number within the byte address
BitNo2	2. Bit number within the byte address
Parameter	Number of the I/O parameter
Value	New value of the I/O parameter

FIFO □

MB_Jog

Purpose Run axis in velocity mode (continuous run)

C/C++

```
void MB_Jog (short Axis, long velocity);
```

Basic

```
Sub MB_Jog (ByVal Axis as Integer, ByVal velocity as
Long)
```

Elements	Description
Axis	Number of the specific axis
Velocity	New run velocity

FIFO ☒

MB_KeyClear

Purpose Erase keyboard buffer

C/C++

```
void MB_KeyClear (void);
```

Basic

```
Sub MB_KeyClear ()
```

FIFO □

MB_KeyLed

Purpose Turn key LEDs on/off

C/C++ `void MB_KeyLed (short Key, short Onoff);`
Basic `Sub MB_KeyLed (ByVal Key as Integer, Onoff as Integer)`

Elements	Description
Key	**Key code of the key LEDs**
Onoff	**New state of the key LEDs**

FIFO □

MB_KeyPressed

Purpose Check if key is pressed

C/C++ `short MB_KeyPressed (void);`
Basic `Function MB_KeyPressed () as Integer`

Elements	Description
Return value	**Nonzero if a key has been pressed**

FIFO □

MB_KeyRead

Purpose Read key

C/C++ `short MB_KeyRead (void);`
Basic `Function MB_KeyRead () as Integer`

Elements	Description
Return value	**Value of the key read**

FIFO □

MB_KeyState

Purpose Query key state

C/C++ `short MB_KeyState (void);`
Basic `Function MB_KeyState () as Integer`

Elements	Description
`Return value`	`State of the key`

FIFO □

MB_Lin, MB_Lin0

Purpose

Linear interpolation at rapid traverse velocity

C/C++

```
void MB_Lin (unsigned C_Mask, long Coordinate[]);
void MB_Lin0 (unsigned C_Mask, long Coordinate[]);
```

Basic

```
Sub MB_Lin (ByVal C_Mask as Integer, ByRef
Coordinate() as Long)
Sub MB_Lin0 (ByVal C_Mask as Integer, ByRef
Coordinate() as Long)
```

Elements	Description
`C_Mask`	`Bit mask of the relevant coordinates`
`Coordinate`	`Pointer to the array of the target coordinates`

FIFO ☒

MB_Lin1

Purpose

Linear interpolation at feed velocity

C/C++

```
void MB_Lin1 (unsigned C_Mask, long Coordinate[]);
```

Basic

```
Sub MB_Lin1 (ByVal C_Mask as Integer, ByRef
Coordinate() as Long)
```

Elements	Description
`C_Mask`	`Bit mask of the relevant coordinates`
`Coordinate`	`Pointer to the array of the target corrdinates`

FIFO ☒

MB_Out

Purpose

Set outputs

C/C++

```
void MB_Out (short ByteNo, short BitNo1, short BitNo2,
short Value);
```

Basic

```
Sub MB_Out (ByVal ByteNo as Integer, ByVal BitNo1 as
Integer, ByVal BitNo2 as Integer, ByVal Value as
Integer)
```

Elements	Description
`ByteNo`	`The byte address`
`BitNo1`	`1st Bit number within the byte address`
`BitNo2`	`2nd Bit number within the byte address`
`Value`	`New state of the output(s)`

FIFO ☒

Remark

Always give the 2nd bit number, even if you only want to set one output.

MB_Outi

Purpose

Set outputs

C/C++

```
void MB_Out (short ByteNo, short BitNo1, short BitNo2,
short Value);
```

Basic

```
Sub MB_Out (ByVal ByteNo as Integer, ByVal BitNo1 as
Integer, _
    ByVal BitNo2 as Integer, ByVal Value as Integer)
```

Elements	Description
`ByteNo`	`The byte address`
`BitNo1`	`1st Bit number within the byte address`
`BitNo2`	`2nd Bit number within the byte address`
`Value`	`New state of the output(s)`

FIFO □

Remark

Always give the 2nd bit number, even if you only want to set one output.

MB_Outw

Purpose

Set output word by word (16 bit)

C/C++

```
void MB_Outw (short ByteNo, short Value);
```

Basic

```
Sub MB_Outw (ByVal ByteNo as Integer, ByVal Value as
Integer)
```

Elements	Description
ByteNo	The byte address
Value	New state of the output(s)

FIFO ☒

MB_Outwi

Purpose

Set output word by word (16 bit)

C/C++
Basic

```
void MB_Outw (short ByteNo, short Value);
Sub MB_Outw (ByVal ByteNo as Integer, ByVal Value as
Integer)
```

Elements	Description
ByteNo	The byte address
Value	New state of the output(s)

FIFO □

MB_Print

Purpose

Print out text at the current cursor position

C/C++
Basic

```
void MB_Print (char * Text);
Sub MB_Print (ByVal Text as String)
```

Elements	Description
Text	Text to be printed

FIFO □

MB_Printxy

Purpose

Print out text at location x, y

C/C++
Basic

```
void MB_Printxy (short x, short y, char * Text);
Sub MB_Printxy (ByVal x as Integer, ByVal y as
Integer, ByVal Text as String)
```

Elements	Description
Text	Text to be printed
x	Column location of the text

y	Line location of the text

FIFO □

MB_ResErr

Purpose Erase all controller errors

C/C++
```
void MB_ResErr (void);
```
Basic
```
Sub MB_ResErr
```

FIFO □

MB_Rmove

Purpose Relative axis positioning

C/C++
```
void MB_Rmove (short Axis, long run path);
```
Basic
```
Sub MB_Rmove (ByVal Axis as Integer, ByVal run path as
Long)
```

Elements	Description
Axis	Number of the specific axis
Run path	Path along which the axis shall run

FIFO X

MB_Rout

Purpose Return output(s)

C/C++
```
void MB_Rout (short ByteNo, short BitNo1, short
BitNo2);
```
Basic
```
Sub MB_Rout (ByVal ByteNo as Integer, ByVal BitNo1 as
Integer, _
    ByVal BitNo2 as Integer
```

Elements	Description
ByteNo	The byte address
BitNo1	1. Bit number within the byte address
BitNo2	2. Bit number within the byte address
Return value	State of the output(s)

FIFO □

MB_Routw

Purpose Return outputs word by word (16 bit)

C/C++ `void MB_Routw (short ByteNo);`

Basic `Sub MB_Routw (ByVal ByteNo as Integer)`

Elements	Description
ByteNo	The byte address
Return value	State of the output(s)

FIFO □

MB_SdoRcv

Purpose Receive SDO (service data object) from CANopen device

C/C++

```
long MB_SdoRcv (unsigned short NodeId, unsigned short
Index, short SubIndex);
```

Basic

```
Function MB_SdoRcv (ByVal NodeId as integer, ByVal
Index as integer, ByVal SubIndex as integer) as long
```

Elements	Description
NodeId	Address of the CANopen device
Index	Index of the SDO object
SubIndex	Subindex of the SDO object
Return value	Value of the SDO-object

FIFO □

MB_SdoTrm

Purpose Send SDO (service data object) to CANopen device

C/C++

```
void MB_SdoTrm (unsigned short NodeId, unsigned
short Index, short SubIndex, long Value);
```

Basic

```
Sub MB_SdoTrm (ByVal NodeId as integer, ByVal Index
as integer, ByVal SubIndex as integer, ByVal Value
as long)
```

Elements	Description
NodeId	Address of the CANopen device
Index	Index of the SDO object
SubIndex	Subindex des SDO object
Value	Value of the SDO object

FIFO □

MB_Set

Purpose Set (program) a system parameter

C/C++
Basic

```
void MB_Set (short Parameter, long Value);
Sub MB-Set (ByVal Parameter as integer, ByVal Value as
Long
```

Elements	Description
Parameter	**Number of the parameter**
Value	**Value of the parameter**

FIFO ☒ X

MB_SetFIFO

Set the state of the online FIFO

C/C++
Basic

```
void MB_SetFIFO (short State);
Sub MB_SetFIFO (ByVal State as Integer)
```

Elements	Description
State	**New state of the FIFO**

FIFO □

MB_Seti

Purpose Set (program) a system parameter

C/C++
Basic

```
void MB_Set (short Parameter, long Value);
Sub MB-Set (ByVal Parameter as integer, ByVal Value as
Long
```

Elements	Description
Parameter	**Number of the parameter**
Value	**Value of the parameter**

FIFO □

MB_Sout

Purpose Setting synchronous outputs

C/C++

```
void MB_Sout (short ByteNo, short BitNo1, short
BitNo2, short Value);
```

Basic

```
Sub MB_Sout (ByVal ByteNo as Integer, ByVal BitNo1 as
Integer, ByVal BitNo2 as Integer, ByVal Value as
Integer)
```

Elements	Description
`ByteNo`	`The byte address`
`BitNo1`	`1. Bit number within the byte address`
`BitNo2`	`2. Bit number within the byte address`
`Value`	`New state of the output(s)`

FIFO ☒

MB_Still

Purpose Await standstill of one or all axes

C/C++
Basic

```
void MB_Still (short Axis);
Sub MB_Still (ByVal Axis as Integer)
```

Elements	Description
`Axis`	`Number of the specific axis. “xall“ (= -1) for all axes is possible (see MotionBasic manual)`

FIFO ☒

MB_Stop

Purpose Stop an axis

C/C++
Basic

```
void MB_Stop (short Axis);
Sub MB_Stop (ByVal Axis as Integer)
```

Elements	Description
`Axis`	`Number of the specific axis`

FIFO □

MB_SysCtrl Break, halt, reset, restart

C/C++
Basic

```
void MB_SysCtrl (short Control);
Sub MB_ SysCtrl (ByVal Control as Integer)
```

FIFO □

MB_TextAttrib

Purpose Set a text attribute

C/C++ `void MB_TextAttrib (short Attrib);`

Basic `Sub MB_TextAttrib (ByVal Attrib as Integer)`

Elements	Description
Attrib	Text attribute

FIFO □

MB_Waitinp

Purpose Wait for state at input

C/C++

```
void MB_Waitinp (short ByteNo, short BitNo1, short
BitNo2, short Value);
```

Basic

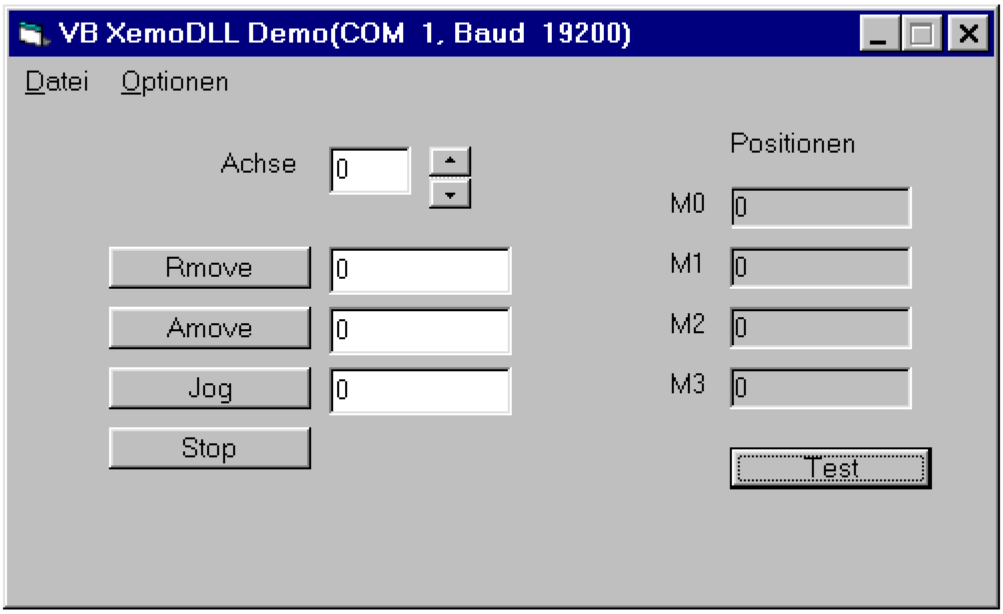
```
Sub MB_Waitinp (ByVal ByteNo as Integer, ByVal BitNo1
as Integer, ByVal BitNo2 as Integer, ByVal Value as
Integer)
```

Elements	Description
ByteNo	The byte address
BitNo1	1. Bit number within the byte address
BitNo2	2. Bit number within the byte address
Value	State of the input(s) being waited for

FIFO ☒

5 Application examples

5.1 Visual Basic sample application



5.1.1 Description

A simple dialog application has been provided in the Visual Basic project VBDemo.vbp, with which you can operate up to four Xemo controller axes. The specific axis can be selected with the scroll bar. The axis thus selected is then steered in relative, absolute or jogging operation with the corresponding buttons. The run-path and/or the velocity can be entered in the input box next to the buttons. Clicking on the stop button halts all axes. At the right side, the positions of all the axes are displayed.

5.1.2 Project modules

The project consists of the following modules:

DemoForm.frm	Visual Basic Form for the application dialog
Formser.frm	Visual Basic Form for the interface dialog
VBdemo.bas	Initialization procedures
XemoDLL.bas	Declaration of the MotionBasic functions
Mbconst.bas	useful MotionBasic constants

5.1.3 Program start

After you start the application in the event procedure Form_Load, the initialization of the Xemo DLL with the function procedure Ini_Xemo is the next step. In case the initialization fails or is aborted by the user, the application itself is aborted.

Form_Load Procedure (DemoForm.frm)

```
Dim ExitEvent As Boolean
Private Sub Form_Load()

 BAUDRATE = GetSetting("XemoDemo", "Comport",_
 "Baudrate", 19200)
 COMNR = GetSetting("XemoDemo", "Comport",_
  "ComNr", 1)
 Caption = "VB XemoDLL Demo" + _
    "(COM " + Str(COMNR) + _
    ", Baud " + Str(BAUDRATE) + ")"
 If Ini_Xemo = False Then
  Unload Me
  Exit Sub
 End If
 ExitEvent = False
 Timer_Refresh.Interval = 100
 Timer_Refresh.Enabled = True

End Sub
```

5.1.4 Initializing

The initializing of the Xemo DLL is somewhat more complex here so that, in case of an error such as a false configuration of the serial interface or if the controller is not connected, the user can either re-configure the interface or abort. Until one or the other has occurred, the initialization procedure remains active.
After initialization of the serial interface (ML_IniCom), the state of the Xemo controller is queried first (MB_Getstate). This query shall verify error-free communication. At the same time, any possible errors in the controller are intercepted. Further action depends on the error state, which is read out with ML_GetErrState. If no error is present, the procedure is ended with “True”. If the user wishes to abort, the process can be exited immediately with “False”. Otherwise, he has the possibility to re-configure the serial interface.

Ini_Xemo Procedure (VBdemo.bas)

```
Function Ini_Xemo() As Boolean

 ML_IniCom COMNR, BAUDRATE

 Do
  MB_GetState

  Dim ErrState As Integer
  ErrState = ML_GetErrState()

 Select Case ErrState
 Case NO_ERR
  Ini_Xemo = True
  Exit Do
 Case ERR_CANCEL
  Ini_Xemo = False
  Exit Function

 Case Else

  If MsgBox("re-configure serial interface?",
vbOKCancel _
    + vbQuestion, "XemoDLL Demo") = vbOK Then

   FormInterface.Show vbModal
  Else
   Ini_Xemo = False
   Exit Function
  End If
 End Select
 Loop

End Function
```

5.1.5 Ending the application

Before the application is ended, the interface must always be closed with the routine ML_DeiniCom, so that it is available for other applications again.

Form_Unload Procedure (DemoForm.frm)

```
Private Sub Form_Unload(Cancel As Integer)

 Timer_Refresh.Enabled = False

 SaveSetting "XemoDemo", "Comport", "Baudrate", _
BAUDRATE
 SaveSetting "XemoDemo", "Comport", "ComNr", COMNR

 ML_DeIniCom
 ExitEvent = True

End Sub
```

5.1.6 Running the axes

For each of the four buttons for steering the axes, there is an event procedure. The procedure for the absolute positioning of an axis looks like this.

Amove Procedure (DemoForm.frm)

```
Private Sub Amove_Click()
Dim Axis
Dim Pos
```

```
 Axis = Val(AxisParameter.Text)
 Pos = Val(AmoveParameter.Text)
 MB_Amove Axis, Pos

End Sub
```

5.1.7 Displaying the positions

So that the current positions of the axes are continuously displayed, a timer is started after initialization (see Timer_Ini procedure) which takes control of the read-out and display of these positions. You should, however, not forget that reading out the positions (MB_AGet) takes a certain amount of time (about 5 ms at 19200) because of the serial interfaces' transmission time. Consequently, good programming will permit other intermediate events by calling up the "DoEvents" procedure. In the worst case (if the timer frequency is set very high and/or the number of queries is quite large), operation of the application could seem very slow and delayed. In this application, the timer is set at 50 ms. However, you should always keep in mind that, when the "DoEvents" procedure is called up, all kinds of possible events can occur, including the "Unload" event. With the global variable "ExitEvent", this event can be intercepted and the timer procedure immediately exited.

Timer_Refresh Procedure (DemoForm.frm)

```
Private Sub Timer_Refresh_Timer()

 If ML_GetErrState() = ERR_CANCEL Then
  Unload Me
  Exit Sub
 End If

Dim x As Integer
For x = 0 To 3
 DoEvents
 If ExitEvent Then Exit Sub
 Position(x).Caption = MB_AGet(x, m_RPos)
Next x

End Sub
```

5.1.8 Aborting in case of error

The Xemo DLL includes a standard routine for error correction. The standard error correction is pre-set and does not need to be initialized explicitly. An application which integrates the Xemo DLL need do nothing else but query the error state with ML_GetErrState at certain intervals. If this has received the value (ERR_CANCEL), some kind of error has occurred and the user should abort.
The timer-refresh procedure takes over querying the error state. If ERR_CANCEL is returned in the error state, the application is ended with "Unload".

5.2 Application example in ANSI –C

5.2.1 Description

Introduction

The file dllappl.c contains a small Windows console application which demonstrates the integration and use of the Xemo DLL in a C- program. In addition, the use of the call-back function for application-specific error correction is made clear.

Function

The application constitutes a small interpreter for MotionBasic commands. The commands are entered by line via the keyboard or read in from a file. After interpretation with the help of the Xemo DLL, the commands are sent to the Xemo controller where they are then executed. Errors occurring within either the DLL or the controller are intercepted and displayed.

If when calling up a program a file name is given as a parameter, that file will be read in; otherwise commands are read in through the keyboard.

Commands

A line-interpreter command begins with a letter followed by one or more numerical parameters which must be separated by a space.

The following commands are provided.

'a':	Absolute positioning of an axis
'r':	Relative positioning of an axis
'v':	Set positioning velocity of an axis
‘j’	Run an axis in the velocity mode
's':	Stop an axis
'q':	Quit the program

Examples of commands

```
a 0 1000          //Axis 0 absolute positioning
r 1 2000          //Axis 1 relative positioning
v 1 800           //Feed for axis 1 = 800
j 4 5000          //Run axis 4 in velocity mode
```

Compiling

The program is written in ANSI C and was compiled as a Windows console application with the Microsoft Visual C++ 6.0 Compiler

Include

At the beginning of the program we find the DLL’s header files, which must always be included together with the system-defined header files.

"Xemodll.h"	Definition of the DLL functions
"mbconst.h"	Constant definitions

Error correction

For testing, the program can be run with an application-specific error routine as well as with the Xemo DLL’s standard error routine. At the beginning of the program there is a “#define” allocation for switching between the two alternatives. It is worth the while to try out both versions.

```
#define APPL_ERROR_FUNC
```

The application-specific error routine (ErrorFunc) first checks to see if it is dealing with a serious communication error or rather a runtime error in the Xemo controller.

Communication error

In case of a communication error, the error code and the error text will be displayed on the monitor. The global variable appl_end signifies that the application should be ended. If the value ERR_CANCEL is returned to the Xemo DLL, it will prevent any further DLL functions from being carried out.
In the main loop, polling appl_end makes certain that the program is ended.

```
while (appl_end == 0)
```

Runtime error

In case of a Xemo controller runtime error, the error code will be provided. In addition, the user has the possibility of erasing the error with the keyboard. Dependent on that, either ERR_RETRY or ERR_CANCEL will be returned.

Initializing

In the main routine, the error correction routine will be communicated to the Xemo DLL after the input stream is opened. Afterwards, the serial interface will be initialized.

```
ML_ErrorCallBack (ErrorFunc);
ML_IniCom (COM_PORT,19200L);
```

Error query

Whenever the various MotionBasic functions are called up within the „case“ query, the controller’s system state will automatically be queried so that, at the same time, any error in the controller is identified. If an error is found, an error function is automatically generated. However, this always occurs before the actual MotionBasic command is transferred to the controller. As a result, the state will again be queried after transfer of the command so that a possible error caused by that command will be identified. This additional query will, however, only be made here if the command was entered via the keyboard. Otherwise, it will be at the end of the program.

5.2.2 Source code listings

5.2.2.1 dllappl.c

```
#define APPL_ERROR_FUNC
/*----------------------------------------------------
 XEMO DLL Version 2.02
 Copyright © 2000, Systec Elektronik und Software GmbH
 Small MotionBasic Interpreter
 Compiled with Microsoft Visual C++ 6.0
---------------------------------------------------*/
```

```
#include <windows.h>

#include <stdio.h>
#include <conio.h>
#include <time.h>

#include "XemoDll.h"
#include "mbconst.h"

#define COM_PORT1           // Number of the serial
                            //interface

int appl_end = 0;           // Variable for program
                            //abort

#ifdef APPL_ERROR_FUNC

//----------------------------------------------------
//   Read and evaluate error from the Xemo controller
//----------------------------------------------------
short Xemo_Runtime_Error ()
{
 long error_code = MB_Get (_ErrNo);
 printf ("\nXemo Runtime Error # %d\n", error_code);

 printf ("Delete error (y/n)? ");
 int key = tolower (getchar());
 printf ("\n");

 if ((key == 'y') || (key == 'y')) {
  MB_ResErr();
  return ERR_RETRY;
 }
 return ERR_CANCEL;
}

//----------------------------------------------------
//   Xemo DLL error correction
//----------------------------------------------------
short ErrorFunc (short ErrCode)
{
 int errState;

 if (ErrCode == ERR_XEMO) {
  errState = Xemo_Runtime_Error();
 } else {
  printf ("\nCommunicationerror no. %d\n",ErrCode);
  printf (ML_ComErrText (ErrCode));
  errState = ERR_CANCEL;
 }

 if (errState == ERR_CANCEL)
  appl_end = 1;
 return (errState);
}

#endif                      // APPL_ERROR_FUNC

//----------------------------------------------------
//   Main program, the interpreter
//----------------------------------------------------
void main (int argc, char * argv[])
{
 int  m;
 FILE *   cmd_file;
 char inp_line[81];
 char token;
 long p[4];


 printf ("\nMini MotionBasic interpreter");
 printf ("  Xemo Dll V. " XEMO_DLL_VERSION "\n");

 cmd_file = stdin;          // default keyboard input

 if (argc > 1) {
  if ((cmd_file = fopen (argv[1],"r")) == NULL)
   cmd_file = stdin;        // input file
 }

#ifdef APPL_ERROR_FUNC
 ML_ErrorCallBack (ErrorFunc);
#endif

 ML_IniCom                  // initialize port
(COM_PORT,19200L);

 printf ("Reset Controller ...");
 MB_SysCtrl(_Reset);        // reset controller
 printf ("\n\n");

 MB_GetState();
 if (appl_end == 0) {
  for (m = 0; m <= 3; ++m)  // ramp values
{

   MB_ASet (m, _Speed, 100000L);
   MB_ASet (m, _Accel, 200000L);
  }

  MB_Printxy (1,1, "DLL-Test");
 }


 while (appl_end == 0) {

  printf ("\r* ");          // input prompt
  inp_line[0] = 0;

  if (fgets (inp_line,80,cmd_file) == NULL)
   break;

  if (inp_line[0] == '\n')
   continue;

  if (cmd_file != stdin)
   printf (inp_line);
  // show file input

  token = inp_line[0];      // read command character

// read parameter
  sscanf (&inp_line[1],"%ld%ld%ld",&p[1],&p[2],&p[3]);

  switch (token) {          // evaluate command
   case 'a':                // absolut position
    MB_Amove ((int)p[1],p[2]);
    break;

   case 'r':                // relative position
    MB_Rmove ((int)p[1],p[2]);
    break;

   case 'v':                // position speed
    MB_ASet ((int)p[1], _Speed, p[2]);
    break;

   case 'j':                // velocity mode
    MB_Jog ((int)p[1],p[2]);
    break;
   case 's':
    MB_Stop ((int)p[1]);
    break;

   case 'l':
    MB_Lin (C_XYZ,p);
    break;

   case 'c':
    MB_Circle (p[1],p[2],p[3]);
    break;

   case 'x':
    MB_Arc (C_XY, p[1],&p[2]);
    break;

   case 'q':
    appl_end = 1;          // terminate program
    break;

   default:  printf ("unknown command\n");
  }

  if (cmd_file == stdin)   // if input from keyboard
   MB_GetState();          // test state for error

#ifndef APPL_ERROR_FUNC
  if (ML_GetErrState() == ERR_CANCEL)
   break;
#endif
 }
 MB_Cls();

 MB_GetState();            // test state for error
 ML_DeIniCom ();           // deinitialize com port

 printf ("\nProgram ended");
 printf ("\nAny key...");
 _getch();
}
```

6 Bibliography

[SYSTEC625] Xemo R/S Equipment Manual, Doc-No. 625-11
© Systec GmbH, Münster, 2019

[SYSTEC717] MotionBasic 6 Programming manual, Doc-No. 717-11
© Systec GmbH, Münster, 2019

[SYSTEC735] Xemo M Equipment Manual, Doc-No. 735-11
© Systec GmbH, Münster, 2017

[SYSTEC764] Xemo B-Panelsteuerung Gerätehandbuch, Doc-No. 764-12
© Systec GmbH, Münster, 2019

[SYSTEC767] LabView-Funktionsbibliothek, Doc-No. 767-22
© Systec GmbH, Münster, 2016

[SYSTEC772] Technologieoptionen, Doc-No. 772-12
© Systec GmbH, Münster, 2017

[SYSTEC775] Xemo!Go User Manual, Doc-No. 775-11
© Systec GmbH, Münster, 2019

[SYSTEC826] Structured troubleshooting, Doc-No. 826-41
© Systec GmbH, Münster 2019

[SYSTEC836] OT300 Gerätehandbuch, Doc-No. 836-12
© Systec GmbH, Münster 2019

[SYSTEC858] Xemo-Step Gerätehandbuch, Doc-No. 858-12
© Systec GmbH, Münster, 2018

[SYSTEC875] MotionBasic 6 IDE User Manual, Doc-No. 875-11
© Systec GmbH, Münster, 2017

You will find these manuals in your manual folder, on your Systec CD or also downloadable on www.systec.de/en/downloads/ .

7 Index