



Buzibachstrasse 31
6023 Rothenburg
Switzerland

Phone +41(0)41 541 50
40
info@zub.ch
www.zub.ch

ZbMoc Library

Software Reference

Version 7.00.102

Address zub machine control AG
Buzibachstrasse 31
CH-6023 Rothenburg
Switzerland
Phone +41-41-541 50 40
Fax +41-41-541 50 49
<http://www.zub.ch>
<http://www.aposs.ch>
info@zub.ch

Copyright © zub machine control AG

zub machine control AG reserves the right to make modifications to the software, documentation, and product as it sees fit without prior or subsequent notice to users.

No part of this publication may be reproduced or distributed in any form or by any means (e.g. photocopy, microfilm, electronic data exchange) or stored in a database or retrieval system, without the prior written permission of zub machine control AG.

While every precaution has been taken in the preparation of this manual, zub machine control AG assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Trademark Many of the designations that are used by manufacturers and sellers to distinguish their products are claimed as trademarks. We point out that such software and hardware designations, brand names, and trademarks used in this publication are protected by law.

In particular, this includes the following:

APOSS is a registered trademark of zub machine control AG.

Microsoft, Windows 7, Windows 10 and Windows 11 are either registered trademarks or trademarks of the Microsoft Corporation in the USA and other countries.

None of the registered trademarks are marked in this publication. This, and the fact that the "R" symbol is missing, does not mean that the trademark is a free trademark.

Table of Contents

1.	Imprint	1
2.	Introduction to ZbMoc	6-7
3.	Function Groups	8-13
4.	ZbMoc Structures	14
4.1.	ZbMocMoconIdentS	14
4.2.	ZbMocMoconInfoS	14-15
4.3.	ZbMocParamS	15-16
4.4.	ZbMocAxisParamS	16-18
5.	ZbMoc Error Codes	19-27
6.	Error Messages	28-34
7.	Function Descriptions	35
7.1.	ZbMocAutoClear	35
7.2.	ZbMocAutoSet	35
7.3.	ZbMocAxisParamRead(DEPRECATED)	35-36
7.4.	ZbMocAxisParamWrite(DEPRECATED)	36
7.5.	ZbMocBlobInfoRequest	36-37
7.6.	ZbMocBlobRead	37
7.7.	ZbMocBlobWrite	37-38
7.8.	ZbMocBreak	38-39
7.9.	ZbMocBreakAll	39
7.10.	ZbMocCanOpenDownloadSDO	39
7.11.	ZbMocCanOpenInhibitPDO	39-40
7.12.	ZbMocCanOpenPollPDO	40
7.13.	ZbMocCanOpenQueuePDO	41
7.14.	ZbMocCanOpenReadSegmentedSDO	41
7.15.	ZbMocCanOpenReadSDO	41-42
7.16.	ZbMocCanOpenStart	42
7.17.	ZbMocCanOpenUploadSDO	42
7.18.	ZbMocCanOpenWritePDO	42-43
7.19.	ZbMocCanOpenWriteSegmentedSDO	43
7.20.	ZbMocCanOpenWriteSDO	43-44
7.21.	ZbMocCanReadRaw	44

7.22. ZbMocCanWriteRaw	44
7.23. ZbMocClose	45
7.24. ZbMocCloseAll	45
7.25. ZbMocClearError	45
7.26. ZbMocCNFRestoreFromFile	45-46
7.27. ZbMocCNFSaveToFile	46
7.28. ZbMocConnect	46
7.29. ZbMocContinue	46-47
7.30. ZbMocDebugPrint	47
7.31. ZbMocDebugPrintw	47
7.32. ZbMocDisconnect	47
7.33. ZbMocExec	47-48
7.34. ZbMocExecTemp	48
7.35. ZbMocGetAxisPosition	48-49
7.36. ZbMocGetError	49
7.37. ZbMocGetIFDriverID	49
7.38. ZbMocGetNumber	49-50
7.39. ZbMocGetZbMocVersion	50
7.40. ZbMocMemoryDump	50
7.41. ZbMocMemoryLock	50-51
7.42. ZbMocMoconBusyWait	51
7.43. ZbMocMoconClearFactory	51
7.44. ZbMocMoconClearParameters	51-52
7.45. ZbMocMoconClearPrograms	52
7.46. ZbMocMoconDeleteEeprom	52
7.47. ZbMocMoconExecutionStatus	52-53
7.48. ZbMocMoconFind	53-54
7.49. ZbMocMoconIdent	54
7.50. ZbMocMoconInfo	54
7.51. ZbMocMoconInfoldx	54-55
7.52. ZbMocMoconNameSet	55
7.53. ZbMocMoconSaveRam	55
7.54. ZbMocMoconSaveRamSection	55-56

7.55. ZbMocMoconSetCanParameters	56-57
7.56. ZbMocMoconState	57
7.57. ZbMocMoconWaitForString	57
7.58. ZbMocOpenCanLpt	57-59
7.59. ZbMocOpenCanUsb	59-60
7.60. ZbMocOpenEtherCat	60-61
7.61. ZbMocOpenSerial	61-62
7.62. ZbMocOpenTcp	62-63
7.63. ZbMocOpenUsb	63-64
7.64. ZbMocOpenV24	64
7.65. ZbMocOpenVlt	64-66
7.66. ZbMocOpenSim	66-67
7.67. ZbMocParamRead	67
7.68. ZbMocParamWrite	67
7.69. ZbMocPollMessage	67-68
7.70. ZbMocProbe	68
7.71. ZbMocProgramDelete	68
7.72. ZbMocProgramList	68-69
7.73. ZbMocProgramLoad	69-70
7.74. ZbMocProgramLoadBinary	70
7.75. ZbMocProgramReadBinary	70-71
7.76. ZbMocProgramSave	71
7.77. ZbMocProgramSaveAs	71-72
7.78. ZbMocReadMemPage	72
7.79. ZbMocReadUserArray	72-73
7.80. ZbMocReadUserVar	73
7.81. ZbMocSDOGetUsage	73-74
7.82. ZbMocSDOGetUsageDetails	74-75
7.83. ZbMocSetCallbackPDO	75
7.84. ZbMocSetCallbackProgress	75-76
7.85. ZbMocSetDebugFile	76
7.86. ZbMocSetDebugFlags	76-77
7.87. ZbMocSetDebugHistory	77

7.88. ZbMocSetDebugImmediate	77
7.89. ZbMocSetDebugLevel	77-78
7.90. ZbMocSetDebugMaxLen	78
7.91. ZbMocSetDebugRetain	78
7.92. ZbMocSourceDeleteFromController	78-79
7.93. ZbMocSourceRestoreFromController	79
7.94. ZbMocSourceSaveInController	79-80
7.95. ZbMocUserParamReadRaw	80
7.96. ZbMocUserParamWriteRaw	80-81
7.97. ZbMocV24ReadRawChar	81
7.98. ZbMocV24ReadRawString	81-82
7.99. ZbMocV24WriteRaw	82
7.100. ZbMocVersionCounts	82-83
7.101. ZbMocWriteUserArray	83
7.102. ZbMocWriteUserVar	83-84
8. Index	85-88

2 Introduction to ZbMoc

What services are provided?

The ZbMoc Library offers a collection of simple to use functions for the

- Configuration
- Control
- Supervising

of zub controllers in a production environment. All of the required functions can be accessed by a Windows application program.

The ZbMoc Library offers a consistent application programming interface (API) for communicating with the different controllers of zub machine control AG. The programming interface is independent of the hardware interfaces and bus systems used by each application and controller.

The slight differences in control and functional range resulting from different hardware and software versions of controllers are handled within the ZbMoc Library.

The library makes it possible to address a controller offering a new interface or bus system, without significant modifications to existing application programs.

The ZbMoc Library is a highly efficient tool which allows the application programmer to focus on the application programming itself rather than working through all details of bus systems, error handling, and so forth.

Developing with the Library

The ZbMoc Library is a Dynamic Link Library (DLL) available for 32-bit Windows (WIN32). It offers full driver support for all supported interfaces. Usage of the DLL requires that the ApossIDE software be installed. Installing the ApossIDE will correctly install and configure the low-level device drivers used by the ZbMoc Library.

Application development depends on the development environment used by your application program. For example, if Microsoft Visual Studio is used, it is sufficient to include the header 'ZbMoc.h' and link the base library (ZbMoc.lib) in order to use all functions of the library. The corresponding 'ZbMoc.dll' must be placed somewhere in the search path which is used by the program.

If the ApossIDE software has been installed, then a small sample program will be installed in the 'Development\ZbMoc' subdirectory under the ApossIDE install directory. This sample program shows ZbMoc Library integration for the Microsoft Visual Studio 2005 and Visual Studio 2010 development environments. Please read the 'ReadMe.txt' file in this directory for details about how to build and execute the sample program.

General usage of the Library

All functions in the library use C calling conventions. Most of the functions will return a signed 16-bit integer error code. This error code will be 0 for success and a negative value for failure. The meanings of the error codes can be found in the 'ZbMocErr.h' header file. Please refer to this file for a current list of possible errors.

Standardized data types defined in the 'ZbTypes.h' header file are used to improve portability of the library. They are:

Signed Integer, 8, 16 and 32 bits long:

- SIGNED8
- SIGNED16

- SIGNED32

Unsigned Integer, 8, 16 and 32 bits long:

- UNSIGNED8
- UNSIGNED16
- UNSIGNED32

Boolean value, 0 or 1 (8 bits long):

- BOOLEAN

Character string using UTF-8 encoding:

- STRUTF8

To communicate with one or more controllers, a 'connection interface' must first be opened by calling one of the 'ZbMocOpen...' functions. These routines will return a 'connection handle' indicating that a connection has been established. This handle is then passed as a parameter, along with the ID of the controller, to the other ZbMoc Library functions. Call one of the 'ZbMocClose...' functions when finished with the Library.

3 Function Groups

Opening and Closing Interfaces

To communicate with a controller, a connection 'interface' must first be opened by calling one of the 'ZbMocOpen...' functions. The 'ZbMocOpen...' functions are responsible for the following internal steps:

1. Availability, compatibility, and fitness-for-use check of the hardware, as well as the required drivers for the selected interface.
2. Setting of all required parameters necessary to open the interface.
3. Scan of all controllers which are connected and match the parameters specified for the interface.
4. Returning an interface handle (0...n) or a negative error code.

The following functions are available for opening the interface:

ZbMocOpenCanLpt	CAN BUS Centronics Adapter
ZbMocOpenCanUsb	CAN USB Stick
ZbMocOpenSerial	V24 Serial Port
ZbMocOpenVlt	VLT Serial Port
ZbMocOpenUsb	USB Port
ZbMocOpenTcp	Ethernet
ZbMocOpenEtherCat	EtherCAT
ZbMocOpenSim	MACS Simulator

All of the 'ZbMocOpen...' functions collect and save information about the connected controllers. The application can retrieve this information by calling either the [ZbMocMoconInfo](#) or [ZbMocMoconInfoIdx](#) function. These functions return the information in a [ZbMocMoconInfoS](#) structure which is defined in 'ZbMoc.h'.

If the application program is being closed, then all open interfaces must first be closed. If an interface must be re-opened with new parameters, then the interface must first be closed before being re-opened. Closing an interface may be done by calling one of the following functions:

ZbMocClose	Close a specified interface.
ZbMocCloseAll	Close all open interfaces.

Other functions related to accessing controllers on an interface (particularly USB interfaces):

ZbMocProbe	Search for controllers on an interface.
ZbMocConnect	Complete a pending connection or re-establish a disconnected connection.
ZbMocDisconnect	Disconnect a single controller without closing an interface.

Controller Information

Information about ZbMoc, open interfaces, and specific controllers can be queried using the following functions:

ZbMocGetZbMocVersion	Return the version string of the active ZbMoc DLL.
ZbMocGetIFDriverID	Get the version string of the low level interface driver.
ZbMocGetNumber	Query the number of open controllers.
ZbMocMoconFind	Find a controller ID based on one of several search criteria.
ZbMocMoconState	Query the connection state of a controller.
ZbMocMoconIdent	Query identification information about a specified controller.
ZbMocMoconInfo	Query general information about a specified controller.
ZbMocMoconInfoIdx	Query general information about a controller identified by index.
ZbMocVersionCounts	Return version-specific controller counts.
ZbMocSDOGetUsage	Query whether or not SDO-based communication may be used.
ZbMocSDOGetUsageDetails	Query whether or not SDO-based communication would be used (if possible).
ZbMocMemoryLock	Query whether or not memory has been locked.

Controller/Interface Status

Information about the status of any open controllers can be queried using the following functions:

ZbMocMoconExecutionStatus	Query the execution state (e.g. Is the controller still executing a program?).
ZbMocMoconBusyWait	Wait until a controller is no longer busy.
ZbMocGetAxisPosition	Query the position of an axis of a controller.
ZbMocGetError	Query the error state of an interface (i.e. not of a controller).
ZbMocClearError	Clear the error state of an interface (i.e. not of a controller).

Controller Configuration

The following functions relate to the configuration of the controller:

ZbMocMoconClearParameters	Reset all parameters of a controller to the factory settings.
ZbMocMoconNameSet	Set the name of a controller.

ZbMocParamRead	Read the global parameters of a controller.
ZbMocParamWrite	Set the global parameters of a controller.
ZbMocAxisParamRead(DEPRECATED)	Read the axis parameters of a controller.
ZbMocAxisParamWrite(DEPRECATED)	Set the axis parameters of a controller.
ZbMocUserParamReadRaw	Read user parameters. User parameters are a set of 100 values intended to hold information used by the application program.
ZbMocUserParamWriteRaw	Write user parameters. User parameters are a set of 100 values intended to hold information used by the application program.
ZbMocReadUserArray	Read DIM Arrays (on-line documentation) .
ZbMocWriteUserArray	Write DIM Arrays (on-line documentation) .
ZbMocMemoryDump	Read an array from a controller and write it into a file.
ZbMocCNFSaveToFile	Read all parameter settings and array definitions from the controller and save them in a configuration file (.zbc) on disk.
ZbMocCNFRestoreFromFile	Read all parameter settings and array definitions from a configuration file (.zbc) and download them to the controller.

The controller's global parameters are read and written using the [ZbMocParams](#) structure. The controller's axis parameters are read and written using the [ZbMocAxisParams](#) structure. Both of these structures are defined in 'ZbMoc.h'.

The number of global and axis parameters implemented in a specific controller depends on the software version of the controller. Use the [ZbMocVersionCounts](#) or [ZbMocMoconInfo](#) function to retrieve the number of parameters supported by a specific controller. As well, some of the parameters have different meanings, depending on the controller chip and software version. For the exact meaning of individual parameters, please see the parameter's documentation.

SDO/PDO Handling

The following functions may be used to handle SDOs and PDOs (if a controller supports them):

ZbMocCanOpenStart	Configure and start a controller as a CAN open slave.
ZbMocCanOpenReadSDO	Read data from an SDO of a controller.
ZbMocCanOpenWriteSDO	Transmit data to an SDO of a controller.
ZbMocCanOpenUploadSDO	Read all subindices of an SDO from a controller.
ZbMocCanOpenDownloadSDO	Write all subindices of an SDO to a controller.
ZbMocCanOpenWritePDO	Transmit a PDO message to a controller.
ZbMocCanOpenInhibitPDO	Set the "inhibit timeout" for the ZbMocCanOpenQueuePDO routine.

ZbMocCanOpenQueuePDO	Transmit a PDO message to a controller.
ZbMocCanOpenPollPDO	Poll for any PDO that has been received.
ZbMocSetCallbackPDO	Defines a pointer to a callback function which will receive PDO messages.

ApossIDE Program Handling

The following functions relate to program handling:

ZbMocProgramList	List all programs permanently stored in the controller.
ZbMocProgramLoad	Download a binary (i.e. compiled) program to a controller using a download program.
ZbMocProgramLoadBinary	Download a binary (i.e. compiled) program to a controller.
ZbMocProgramReadBinary	Upload a binary (i.e. compiled) program from a controller.
ZbMocProgramSave	Store the temporarily downloaded program with a given name. A program number is assigned automatically to it.
ZbMocProgramSaveAs	Store the temporarily downloaded program with a given name and program number.
ZbMocProgramDelete	Delete a program with the given program number.
ZbMocMoconClearPrograms	Delete all programs in a controller.
ZbMocExec	Execute a stored program identified by a program number.
ZbMocExecTemp	Execute the temporary program. It is possible to start the temporary program directly after downloading without first having to save the program.
ZbMocBreak	Stop a running program as well as clear a controller error state. The controller is set to a defined state.
ZbMocBreakAll	Identical to ZbMocBreak , but affects all controllers.
ZbMocContinue	Continue a program that was previously interrupted by a "Break" command.
ZbMocPollMessage	Retrieves the next message generated by a "print" command in the running application program or by a system message generated by

	the controller itself.
ZbMocAutoSet	Set the 'Autostart' flag for a given stored program. The program with the 'Autostart' flag, starts up automatically at power-up of the controller.
ZbMocAutoClear	Erase the 'Autostart' flag. No program is executed automatically at power-up of the controller.
ZbMocSourceSaveInController	Write the source code in the given file into a controller.
ZbMocSourceRestoreFromController	Read the source code of the given program from a controller.
ZbMocSourceDeleteFromController	Delete the source code of the given program from a controller.

Memory Functions

The following memory-related functions are available:

ZbMocMoconClearFactory	Reset a controller to the factory settings.
ZbMocMoconDeleteEeprom	Erase the content of the non-volatile memory (EEPROM) in a controller.
ZbMocMoconSaveRam	Save programs and parameters in non-volatile memory. This is necessary only for controllers without battery backed-up RAM.
ZbMocMoconSaveRamSection	Save the content of the specified RAM section in non-volatile memory.
ZbMocBlobInfoRequest	Return the blob information headers of all saved blobs.
ZbMocBlobRead	Read blob data that was saved in a controller.
ZbMocBlobWrite	Save blob data in a controller.

Other Functions

The following miscellaneous functions are also available:

ZbMocSetCallbackProgress	Defines a callback pointer to a function called to provide progress monitoring.
ZbMocCanReadRaw	Read a single (raw) CAN message of a controller.
ZbMocCanWriteRaw	Transmit a single CAN message to a controller.
ZbMocMoconSetCanParameters	Define the CAN communication parameters of a controller.

ZbMocMoconWaitForString	Wait until the given string is received via serial port of a controller.
ZbMocReadMemPage	Read a memory page out of a controller.
ZbMocReadUserVar	Read a data message from a controller.
ZbMocWriteUserVar	Transmit a data message to a controller.
ZbMocV24ReadRawChar	Read a single character received via the serial interface.
ZbMocV24ReadRawString	Read a string received via the serial interface.
ZbMocV24WriteRaw	Write a string to the serial interface buffer for transmission.

Debug Functions

The following functions may be useful for developers to debug communication issues related to the use of the ZbMoc Library:

ZbMocSetDebugImmediate	Start a debug log immediately using default level and flag settings.
ZbMocSetDebugFile	Define the log file in which debugging messages are written.
ZbMocSetDebugFlags	Define the debugging flags.
ZbMocSetDebugLevel	Define the debugging level.
ZbMocSetDebugHistory	Set the maximum size of the debug history.
ZbMocSetDebugRetain	Set the number of days to retain debug log files.
ZbMocSetDebugMaxLen	Set the maximum length debug log files.
ZbMocDebugPrint	Insert a string into the debug log file.
ZbMocDebugPrintw	Insert a Unicode string into the debug log file.

4 ZbMoc Structures

4.1 ZbMocMoconIdentS

The ZbMocMoconIdentS structure contains identification information about a single controller. The 'ZbMocOpen...' functions retrieve this information automatically from all connected controllers after the connection has been established.

The ZbMoc library caches this information for each of the controllers connected and returns a copy of it to the application via the [ZbMocMoconIdent](#) function. Note that a call to this function is faster than a call to [ZbMocMoconInfo](#).

```
typedef struct
{
    UNSIGNED16 id;           // Controller ID (same as "can_id" for CAN interfaces,
                           // port number for EtherCAT).
    UNSIGNED16 ctrltype;    // Controller type. ZBMOC_CONTROLLER_UNKNOWN if not
                           // known.
    UNSIGNED32 serial;      // Serial number. 0 if not known.
    UNSIGNED32 firmware;    // Firmware version number (e.g. 4.5.1 is represented
                           // as 40501). 0 if not known.
    STRUTF8 desc[100];      // V24 banner (sign on) string, USB device description,
                           // TCP application title, EtherCAT revision. Empty if
                           // not known or not used.
    SIGNED32 location;      // Device location (USB interfaces). 0 if not known or
                           // not used.
    SIGNED32 pid;           // USB product ID (USB interfaces). 0 if not known or
                           // not used.
    UNSIGNED32 ipaddr;      // IP address (Ethernet interfaces). 0 if not known or
                           // not used.
} ZbMocMoconIdentS;
```

4.2 ZbMocMoconInfoS

The ZbMocMoconInfoS structure contains information about a single controller. The 'ZbMocOpen...' functions retrieve this information automatically from all connected controllers after the connection has been established.

The ZbMoc library caches this information for each of the controllers connected and returns a copy of it to the application via the [ZbMocMoconInfo](#) and [ZbMocMoconInfoldx](#) functions.

```
typedef struct ZbMocMoconInfoS_
{
    UNSIGNED16 state;        // Connection state.
    UNSIGNED16 controller;   // Controller type.
    UNSIGNED16 interf;       // Interface type.
    UNSIGNED16 channelno;    // Channel number.
    UNSIGNED32 full_version;  // Full version number
                           // (e.g. 4.5.1 is represented as 40501).
    UNSIGNED16 major_version; // Major version number.
    UNSIGNED16 minor_version; // Minor version number.
    UNSIGNED16 micro_version; // Micro version number.
    UNSIGNED8 cpu_type;       // CPU type.
    UNSIGNED16 axis_controller_type; // Type of axis controller chip
```



```

// ('L'=National, 'H'=HCTL, 'S'=Software).
UNSIGNED8    number_axis;           // Number of axes.
UNSIGNED8    option_code;           // Option code.
UNSIGNED8    board_revision;        // Board version.
UNSIGNED16   bus_id;                // Controller ID (same as "can_id" for
// CAN interfaces).
UNSIGNED16   can_id;                // CAN ID (0 if not known).
STRUTF8      unit_name[24+1];        // Name of controller (trimmed and
// terminated).
UNSIGNED32   regspeed;              // Regulator speed (8MHz==800).
UNSIGNED8    sdo_compatible;         // Level of firmware SDO compatibility.
UNSIGNED16   globalcnt;             // Number of global parameters.
UNSIGNED16   usercnt;               // Number of user parameters.
UNSIGNED16   axiscnt;               // Number of axis parameters.
UNSIGNED32   arraymax;              // Maximum length of arrays.
double       timebase;              // Length of one controller "tick" in
// milliseconds.
UNSIGNED16   devclass;              // Device class.

UNSIGNED16   pid;                   // USB product ID.
STRUTF8      serial[50];             // Serial number (if known).
STRUTF8      description[100];       // Device description.
STRUTF8      banner[100];            // Banner (sign on)string
// (old V24 controllers only).

UNSIGNED32   number_amp;             // Number of amplifiers.
UNSIGNED32   ipaddr;                // IP address (TCP controller only).

UNSIGNED32   features;              // Controller feature flags.
} ZbMocMoconInfoS;

```

4.3 ZbMocParamS

The ZbMocParamS structure contains the controller's "global" parameters. They can be read with the [ZbMocParamRead](#) function and written with the [ZbMocParamWrite](#) function.

The number of parameters implemented in a specific controller depends on the software version of the controller. See the controller documentation for details. The number of parameters supported by a specific controller, can be found in the [ZbMocMoconInfoS](#) structure.

```

typedef struct ZbMocParamS_
{
    UNSIGNED32 can_nr;                // #100
    UNSIGNED32 can_baud;              // #101
    UNSIGNED32 prgpar;                // #102
    UNSIGNED32 i_prgstart;            // #103
    UNSIGNED32 i_prgchoice;           // #104
    UNSIGNED32 i_break;               // #105
    UNSIGNED32 i_continue;            // #106
    UNSIGNED32 i_errclr;              // #107
    UNSIGNED32 o_error;               // #108
    UNSIGNED32 ainftime;              // #109 (MACS3 >= 6.4.xx only)
    UNSIGNED32 external24v;           // #110 (VLT >= 6.4.xx only)
    UNSIGNED32 rsttermination;        // #111 (VLT >= 6.4.xx and MACS3 >= 6.4.xx only)
    UNSIGNED32 rsbaudrate;            // #112 (VLT >= 6.4.xx and MACS3 >= 6.4.xx only)
    UNSIGNED32 iomode;               // #113 (VLT >= 6.4.xx only)

```

```

    UNSIGNED32 cansynctimer;    // #114 (6.5.16)
    UNSIGNED32 canguardtimer;   // #115 (6.6.18)
    UNSIGNED32 vmencntype;      // #116 (6.6.18)
    UNSIGNED32 vmencres;        // #117 (6.6.18)
    UNSIGNED32 vmencmtyp;       // #118 (6.6.18)
    UNSIGNED32 ipaddressmode;   // #119 (7.0.23)
    UNSIGNED32 ipsubnet;        // #120 (7.0.23)
    UNSIGNED32 reserved1;       // #121 (7.1.32)
    UNSIGNED32 reserved2;       // #122 (7.1.32)
    UNSIGNED32 ipmask;          // #123 (7.1.32)
    UNSIGNED32 ipdefaultgw;     // #124 (7.1.32)
} ZbMocParamS;

```

4.4 ZbMocAxisParamS

The ZbMocAxisParamS structure contains the controller's "axis" parameters. They can be read with the [ZbMocAxisParamRead\(DEPRECATED\)](#) function and written with the [ZbMocAxisParamWrite\(DEPRECATED\)](#) function.

The number of parameters implemented in a specific controller depends on the software version of the controller. See the controller documentation for details. The number of parameters supported by a specific controller, can be found in the [ZbMocMoconInfoS](#) structure. Please note that some of the parameters have different meanings depending on the controller's chip and software version.

It is recommended that the ApossIDE software be used to set axis parameters and permanently save them to controller's memory. The ApossIDE automatically handles all hardware and software controller versions correctly.

```

typedef struct ZbMocAxisParamS_
{
    UNSIGNED32 function_inputs;    // #0
    UNSIGNED32 syncmftime;        // #18
    SIGNED32 sensor_counts;        // #2
    UNSIGNED32 max_velocity;       // #9
    UNSIGNED32 max_acceleration;   // #10
    UNSIGNED32 timer;              // #14
    UNSIGNED32 gain;               // #11
    UNSIGNED32 zero;               // #12
    UNSIGNED32 pole;               // #13
    UNSIGNED32 pullgap;            // #15
    UNSIGNED32 test_window;        // #8
    UNSIGNED32 syncofftime;        // #16
    UNSIGNED32 syncmfpar;          // #17
    UNSIGNED32 home_velocity;      // #7
    UNSIGNED32 home_enforce;       // #3
    UNSIGNED32 negative_limit;     // #4
    UNSIGNED32 positive_limit;     // #5
    UNSIGNED32 integration_limit;  // #21
    UNSIGNED32 use_neg_limit;      // #19
    UNSIGNED32 use_pos_limit;      // #20
    UNSIGNED32 velocity_divisor;   // #22
    UNSIGNED32 user_factor;        // #23
    UNSIGNED32 rpm;                // #1
    UNSIGNED32 mnu_point;          // #6
    UNSIGNED32 test_duration;      // #24
    UNSIGNED32 test_limit;         // #25
    UNSIGNED32 usrteil;            // ???
    UNSIGNED32 encodertyp;         // #27
}

```

```

UNSIGNED32 posdrct;           // #28
UNSIGNED32 posunits;         // #29
UNSIGNED32 mencoder;         // #30
UNSIGNED32 mencodertype;     // #67
UNSIGNED32 ramp_min;         // #31
UNSIGNED32 ramptype;         // #32
UNSIGNED32 dfltvel;          // #33
UNSIGNED32 dfltacc;          // #34
UNSIGNED32 bandwidth;        // #35
UNSIGNED32 ffvel;            // #33
UNSIGNED32 ffacc;            // #37
UNSIGNED32 regwmax;          // #38
UNSIGNED32 regwmin;          // #39
UNSIGNED32 hometype;         // #40
UNSIGNED32 home_ramp;        // #41
UNSIGNED32 originoffs;       // #42
UNSIGNED32 errcond;          // #43
UNSIGNED32 endswmod;         // #44
UNSIGNED32 i_refswitch;      // #45
UNSIGNED32 i_poslimit;       // #46
UNSIGNED32 i_neglimit;       // #47
UNSIGNED32 o_brake;          // #48
UNSIGNED32 syncfactm;        // #49
UNSIGNED32 syncfacts;        // #50
UNSIGNED32 syncctype;        // #51
UNSIGNED32 syncmarkm;        // #52
UNSIGNED32 syncmarks;        // #53
UNSIGNED32 syncposoffs;      // #54
UNSIGNED32 syncaccuracy;     // #55
UNSIGNED32 syncready;        // #56
UNSIGNED32 syncfault;        // #57
UNSIGNED32 syncpulsm;        // #58
UNSIGNED32 syncpulss;        // #59
UNSIGNED32 syncmtypm;        // #60
UNSIGNED32 syncmtyps;        // #61
UNSIGNED32 syncmstart;       // #62
UNSIGNED32 syncvftime;       // #65
UNSIGNED32 syncvelrel;       // #66
UNSIGNED32 revers;           // #63
UNSIGNED32 o_axmove;         // #64
UNSIGNED32 posfact_n;        // #26
UNSIGNED32 syncmwinm;        // #68
UNSIGNED32 syncmwins;        // #69
UNSIGNED32 esccond;          // #70
UNSIGNED32 encoderdatlen;    // #71 (VLT >= 6.4.xx only)
UNSIGNED32 encoderterm;      // #72 (VLT >= 6.4.xx only)
UNSIGNED32 encoderclock;     // #73 (VLT >= 6.4.xx only)
UNSIGNED32 encoderfreq;      // #74 (VLT >= 6.4.xx only)
UNSIGNED32 mencoderdatlen;   // #75 (VLT >= 6.4.xx only)
UNSIGNED32 mencoderterm;     // #76 (VLT >= 6.4.xx only)
UNSIGNED32 mencoderclock;    // #77 (VLT >= 6.4.xx only)
UNSIGNED32 mencoderfreq;     // #78 (VLT >= 6.4.xx only)
UNSIGNED32 statusmonitor;    // #79 (VLT >= 6.4.xx only)
UNSIGNED32 encoderdelay;     // #80 (VLT >= 6.4.xx only)
UNSIGNED32 mencoderdelay;    // #81 (VLT >= 6.4.xx only)
UNSIGNED32 encodermonitoring; // #82 (VLT >= 6.4.xx only)
UNSIGNED32 mencodermonitoring; // #83 (VLT >= 6.4.xx only)
UNSIGNED32 encoderabsturns;   // #84 (VLT >= 6.4.xx only)
UNSIGNED32 mencoderabsturns; // #85 (VLT >= 6.4.xx only)

```

```

UNSIGNED32 encoderabstype; // #86 (VLT >= 6.4.xx only)
UNSIGNED32 mencoderabstype; // #87 (VLT >= 6.4.xx only)
UNSIGNED32 encoderabsres; // #88 (VLT >= 6.4.xx only)
UNSIGNED32 mencoderabsres; // #89 (VLT >= 6.4.xx only)
UNSIGNED32 velkprop; // #90 (6.4.63 zub)
UNSIGNED32 velkint; // #91 (6.4.63 zub)
UNSIGNED32 curkprop; // #92 (6.4.63 zub)
UNSIGNED32 curkint; // #93 (6.4.63 zub)
UNSIGNED32 ampmaxcur; // #94 (6.4.63 zub)
UNSIGNED32 ampcommtyp; // #95 (6.4.63 zub)
UNSIGNED32 amppwmfreq; // #96 (6.4.63 zub)
UNSIGNED32 ampmode; // #97 (6.4.63 zub)
UNSIGNED32 jerkmin; // #98 (6.4.63 zub)
UNSIGNED32 spienncoder; // #99 (6.4.72 MOC305 only)
UNSIGNED32 jerkmin2; // #100 (6.4.72 zub)
UNSIGNED32 jerkmin3; // #101 (6.4.72 zub)
UNSIGNED32 jerkmin4; // #102 (6.4.72 zub)
UNSIGNED32 velkilim; // #103 (6.6.18 zub)
UNSIGNED32 curkilim; // #104 (6.6.18 zub)
UNSIGNED32 kilimtime; // #105 (6.6.18 zub)
UNSIGNED32 syncsftime; // #106 (6.6.18 zub)
UNSIGNED32 enccontrol; // #107 (6.6.18 zub)
UNSIGNED32 menccontrol; // #108 (6.6.18 zub)
UNSIGNED32 syncffvel; // #109 (6.6.24 zub)
UNSIGNED32 syncvflimit; // #110 (6.6.24 zub)
UNSIGNED32 poserrtime; // #111 (6.6.32 zub)
UNSIGNED32 feeddist; // #112 (6.6.51 zub)
UNSIGNED32 feedrev; // #113 (6.6.51 zub)
UNSIGNED32 posencqc; // #114 (6.6.51 zub)
UNSIGNED32 posencrev; // #115 (6.6.51 zub)
UNSIGNED32 homezvel; // #116 (6.6.51 zub)
UNSIGNED32 i2tftime; // #117 (6.7.19 zub)
UNSIGNED32 i2tlimit; // #118 (6.7.19 zub)
UNSIGNED32 curftime; // #119 (6.7.19 zub)
UNSIGNED32 encoderid; // #120 (6.7.40 MOC305 only)
UNSIGNED32 encoderbaud; // #121 (6.7.40 MOC305 only)
UNSIGNED32 encoderparity; // #122 (6.7.40 MOC305 only)
UNSIGNED32 mencoderid; // #123 (6.7.40 MOC305 only)
UNSIGNED32 mencoderbaud; // #124 (6.7.40 MOC305 only)
UNSIGNED32 mencoderparity; // #125 (6.7.40 MOC305 only)
UNSIGNED32 ampencno; // #126 (6.7.43 zub)
UNSIGNED32 ampencres; // #127 (6.7.43 zub)
UNSIGNED32 ampencrpm; // #128 (6.7.43 zub)
UNSIGNED32 mencoderno; // #129 (6.7.44 zub)
UNSIGNED32 pisrc_svirtcount; // #130 (7.4.18 zub)
UNSIGNED32 pisrc_mvirtcount; // #131 (7.4.18 zub)
UNSIGNED32 pisrc_svirtlatch; // #130 (7.4.18 zub)
UNSIGNED32 pisrc_mvirtlatch; // #131 (7.4.18 zub)
UNSIGNED32 kffacc; // #132 (8.0.08 zub)
UNSIGNED32 kffdec; // #133 (8.0.08 zub)
} ZbMocAxisParamS;

```

5 ZbMoc Error Codes

ZbMoc Library error codes are defined in 'ZbMocErr.h'. This is included by 'ZbMoc.h'.

ZbMoc routines will normally return ZBMOC_OK (i.e. 0) if the function succeeded and a non-0 value if the function failed for any reason. However, a few routines (in particular, the "Open" routines and the routines that return counts) will return a non-negative number indicating success.

All errors generated by the ZbMoc Library routines themselves, are negative. All errors (or other status indications) returned by the controller, will be positive. See [Error Messages](#) for the errors returned by the controller. #defines for these errors can be found in 'SysDef.mi' (see **SysDef Defines (on-line documentation)**).

General Errors

Value	#define	Description
0	ZBMOC_OK	Success.
-500	ZBMOC_E_USERABORT	User abort.
-501	ZBMOC_E_NOT_RESPONDING	Lost connection with controller (controller is not responding).
-100	ZBMOC_E_UNAVAILABLE	Operation not available.
-101	ZBMOC_E_ALLBUSY	No new handles are available.
-102	ZBMOC_E_BADHANDLE	Given handle is not valid.
-103	ZBMOC_E_BADINDEX	Given index is not valid.
-104	ZBMOC_E_CANONLY	Function is for CAN connections only.
-105	ZBMOC_E_PARAMETERS	Illegal parameters.
-106	ZBMOC_E_PARAMETER_READOUT	Error in reading parameters.
-107	ZBMOC_E_IS_BUSY	Controller is busy.
-108	ZBMOC_E_FUNCTION_UNAVAILABLE	Function not available.
-109	ZBMOC_E_BADID	ID does not exist.
-110	ZBMOC_E_NOCONTROLLER	No controller found.
-111	ZBMOC_E_COMPRESSION_ERROR	Compression error.
-112	ZBMOC_E_BADBLOB	Problem with Blob (e.g. Blob number is not valid).
-113	ZBMOC_E_PDO_TOO_LONG	PDO length is too long for this interface type.
-114	ZBMOC_E_BADARRAYNO	Array does not exist.
-115	ZBMOC_E_BADARRAYSIZE	Array size mismatch.
-116	ZBMOC_E_BREAK_REQUIRED	A 'Break' command is required for execution of the requested function.

Value	#define	Description
-117	ZBMOC_E_CONTROLLER_MISSING	Not all control units respond to a request.
-118	ZBMOC_E_RXBUFFER_TOO_SMALL	Size of provided buffer is too small.
-119	ZBMOC_E_FILEACCESS	File access error, e.g. file not found.
-120	ZBMOC_E_FILECONTENT	Content of file not valid.
-121	ZBMOC_E_LOWMEMORY	Running out of memory.
-122	ZBMOC_E_ARRAYTOOBIG	Array size is too big.
-123	ZBMOC_E_BADPOINTER	Bad pointer (e.g. NULL pointer).
-124	ZBMOC_E_BADPARAMNUM	Bad parameter number.
-125	ZBMOC_E_PROGRAMTOOBIG	Program is too large to download or too large to upload.
-126	ZBMOC_E_BLOCKED	Controller is blocked by a previously executing command.
-127	ZBMOC_E_NO_PROGRAM_SLOT	There is no available program slot to save another program.
-129	ZBMOC_E_PENDINGCONNECTION	Connection (probably USB) is pending.
-130	ZBMOC_E_NAME_TOO_LONG	Name is too long.
-131	ZBMOC_E_CANNOT_PACK	String contains characters that cannot be packed.
-132	ZBMOC_E_CRC_MISMATCH	CRC values did not match.
-133	ZBMOC_E_ALREADYOPEN	Interface is already open.
-134	ZBMOC_E_OPENERERROR	Error trying to open interface (see log file).
-135	ZBMOC_E_CNFINCOMPLETE	Configuration data is incomplete.
-136	ZBMOC_E_TIME_FAILURE	Could not read PC time.

CAN-specific error messages concerning CAN bus problems

Value	#define	Description
-202	ZBMOC_E_CAN_TIMEOUT	Access timeout to CAN.
-203	ZBMOC_E_CAN_NO_RESET	No reset.
-204	ZBMOC_E_CAN_NO_HALT	No halt.
-205	ZBMOC_E_CAN_NO_START	No start.
-206	ZBMOC_E_CAN_NO_HARDWARE	No CAN hardware found.
-207	ZBMOC_E_CAN_NO_MEMORY	Not enough memory.

Value	#define	Description
-208	ZBMOC_E_CAN_NO_USER	No user found.
-209	ZBMOC_E_CAN_NO_CPU_ACCESS	No access to CPU.
-210	ZBMOC_E_CAN_NO_POLLSLAVES	No pollslaves.
-211	ZBMOC_E_CAN_TOO_MANY_SLAVES	Too many slaves.
-212	ZBMOC_E_CAN_NO_SLAVE	No slaves found.
-213	ZBMOC_E_CAN_NO_INIT	Not initialized.
-214	ZBMOC_E_CAN_SLAVES_EXIST	Slave exists.
-215	ZBMOC_E_CAN_NO_POLLING	Polling is not active.
-216	ZBMOC_E_CAN_POLLING_IS_RUN	Polling is active.

VLT-specific error messages

Value	#define
-217	ZBMOC_E_VLT_SCAN_IN_PROGRESS
-218	ZBMOC_E_VLT_SCAN_ERROR
-219	ZBMOC_E_VLT_SCAN_NO_SLAVES
-221	ZBMOC_E_VLT_ILLEGAL_ID
-222	ZBMOC_E_VLT_OPEN
-223	ZBMOC_E_VLT_TRANSMIT
-224	ZBMOC_E_VLT_FRAMETYPE
-225	ZBMOC_E_VLT_MESSAGE_TYPE
-226	ZBMOC_E_VLT_STATUS_READ_ERROR
-227	ZBMOC_E_VLT_RXBCC
-228	ZBMOC_E_VLT_NORXBUFFER
-229	ZBMOC_E_VLT_INTERNAL
-230	ZBMOC_E_VLT_RXBADLENGTH
-231	ZBMOC_E_VLT_UNKNOWN
-232	ZBMOC_E_VLT_TIMEOUT
-233	ZBMOC_E_VLT_LOSTCONNECTION

COM port specific error messages

Value	#define
-234	ZBMOC_E_COM_ILLEGALPORT
-235	ZBMOC_E_COM_OPENERROR

Value	#define
-236	ZBMOC_E_COM_TXERROR
-237	ZBMOC_E_COM_TIMEOUT
-238	ZBMOC_E_COM_ERROR

CANopen error messages

Value	#define
-239	ZBMOC_E_CAN_COMERROR
-240	ZBMOC_E_CAN_TIMER_UNAVAILABLE
-241	ZBMOC_E_CAN_STATUS_TIMEOUT
-242	ZBMOC_E_COM_XON_ERROR
-243	ZBMOC_E_COM_CLOSED
-244	ZBMOC_E_COM_COMMAND
-245	ZBMOC_E_COM_PROTOCOL_LEN
-246	ZBMOC_E_COM_GLOBAL_TIMEOUT
-247	ZBMOC_E_COM_COMMAND_TIMEOUT
-249	ZBMOC_E_NULL_POINTER
-250	ZBMOC_E_SLAVE_NOT_FOUND
-251	ZBMOC_E_NO_PARAMETERS
-252	ZBMOC_MESSAGE_READ
-253	ZBMOC_E_COM_FRAME_FORMAT_ERROR
-254	ZBMOC_E_COM_FRAME_TEMP_OVERFLOW
-255	ZBMOC_E_ETHERCAT_ERROR
-256	ZBMOC_E_CAN_NOT_ZUB_DEVICE
-257	ZBMOC_E_CAN_LISTEN_ONLY
-261	ZBMOC_E_USB_CONNECT_FAILED

TCP error messages

Value	#define	Description
-281	ZBMOC_E_TCP_INVALID_ADDRESS	Invalid TCP/IP address.
-282	ZBMOC_E_TCP_SOCKET_ERROR	A TCP/IP socket error has occurred.
-283	ZBMOC_E_TCP_CONNECTION_LOST	Connection dropped by Server.
-284	ZBMOC_E_TCP_NOT_CONNECTED	Connection was previously dropped.
-285	ZBMOC_E_TCP_BAD_RESP_LENGTH	Invalid response length received.

Value	#define	Description
		Connection dropped.
-286	ZBMOC_E_TCP_BAD_COMMAND	Invalid command received by Server.

CANopen error messages

Value	#define
-301	ZBMOC_E_CANOPEN_ServerAckTimeout
-302	ZBMOC_E_CANOPEN_ResponseNotExpected
-303	ZBMOC_E_CANOPEN_TxTimeout
-304	ZBMOC_E_CANOPEN_ArraySizeMismatch
-305	ZBMOC_E_CANOPEN_UploadLastSegment

CANopen error messages for "ServerAbortDomainTransfer" errors

Value	#define	Description
-350	ZBMOC_E_ABORT_Unknown	Unknown abort code.
-351	ZBMOC_E_ABORT_ToggleBitNotAlternated	Toggle bit not alternated.
-352	ZBMOC_E_ABORT_SdoProtocolTimedOut	SDO protocol timed out.
-353	ZBMOC_E_ABORT_CommandSpecifierNotValid	Client/server command specifier not valid or unknown.
-354	ZBMOC_E_ABORT_InvalidBlockSize	Invalid block size (block mode only).
-355	ZBMOC_E_ABORT_InvalidSequenceNumber	Invalid sequence number (block mode only).
-356	ZBMOC_E_ABORT_CrcError	CRC error (block mode only).

Value	#define	Description
-357	ZBMOC_E_ABORT_OutOfMemory	Out of memory.
-358	ZBMOC_E_ABORT_UnsupportedAccessToAnObject	Unsupported access to an object.
-359	ZBMOC_E_ABORT_AttemptToReadWriteOnlyObject	Attempt to read a write only object.
-360	ZBMOC_E_ABORT_AttemptToWriteReadOnlyObject	Attempt to write a read only object.
-361	ZBMOC_E_ABORT_ObjectDoesNotExist	Object does not exist in the object dictionary.
-362	ZBMOC_E_ABORT_ObjectCannotBeMappedToThePDO	Object cannot be mapped to the PDO.
-363	ZBMOC_E_ABORT_MappedObjectsExceedPDOlength	The number and length of the objects to be mapped would exceed PDO length.
-364	ZBMOC_E_ABORT_GeneralParameterIncompatibility	General parameter incompatibility.
-365	ZBMOC_E_ABORT_GeneralInternalIncompatibility	General internal incompatibility in the device.
-366	ZBMOC_E_ABORT_AccessFailedDueToAnHardwareError	Access failed due to a

Value	#define	Description
		hardware error.
-367	ZBMOC_E_ABORT_DataTypeConflict	
-368	ZBMOC_E_ABORT_LengthOfServiceParameterDoesNotMatch	Data type does not match, length of service parameter does not match.
-369	ZBMOC_E_ABORT_LengthOfServiceParameterTooHigh	Data type does not match, length of service parameter too high.
-370	ZBMOC_E_ABORT_LengthOfServiceParameterTooLow	Data type does not match, length of service parameter too low.
-371	ZBMOC_E_ABORT_ObjectAttributeInconsistent	
-372	ZBMOC_E_ABORT_SubIndexDoesNotExist	Sub-index does not exist.
-373	ZBMOC_E_ABORT_ParamValueRangeExceeded	Value range of parameter exceeded (only for write access).
-374	ZBMOC_E_ABORT_ParamValueWrittenTooHigh	Value of parameter written too high.

Value	#define	Description
-375	ZBMOC_E_ABORT_ParamValueWrittenTooLow	Value of parameter written too low.
-376	ZBMOC_E_ABORT_MaxValueIsLessThanMinValue	Maximum value is less than minimum value.
-377	ZBMOC_E_ABORT_GeneralError	General error.
-378	ZBMOC_E_ABORT_DataCannotBeTransferredOrStored	Data cannot be transferred or stored to the application.
-379	ZBMOC_E_ABORT_DataCannotBeTransferredOrStoredLocalControl	Data cannot be transferred or stored to the application because of local control.
-380	ZBMOC_E_ABORT_DataCannotBeTransferredOrStoredDeviceState	Data cannot be transferred or stored to the application because of the present device state.
-381	ZBMOC_E_ABORT_ObjectDictionaryGenerationFails	Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is

Value	#define	Description
		generated from file and generation fails because of an file error).

6 Error Messages

All of the #define keywords listed below, can be found in the **SysDef.mh ('SysDef Defines' in the on-line documentation)** include file and may be used by developers in application programs if this file is included.

Error #	ApossIDE error message	#define
2 ('Error 2' in the on-line documentation)	There are no more CAN objects available (CANINI).	F_CANMEM
3 ('Error 3' in the on-line documentation)	Axis not in system.	F_NOAXES
5 ('Error 5' in the on-line documentation)	Error not cleared.	F_ERR
6 ('Error 6' in the on-line documentation)	Failed to move to HOME position.	F_NOMNU
7 ('Error 7' in the on-line documentation)	Home velocity 0.	F_MNU
8 ('Error 8' in the on-line documentation)	Position error.	F_POSERR
9 ('Error 9' in the on-line documentation)	Index pulse (encoder) not found.	F_NIO
10 ('Error 10' in the on-line documentation)	Unknown command.	F_UNBEK
11 ('Error 11' in the on-line documentation)	Software end limit activated.	F_GRENZE
12 ('Error 12' in the on-line documentation)	Illegal parameter number.	F_PARNUM
14 ('Error 14' in the on-line documentation)	Too many nested loops.	F_NOCHZ
15 ('Error 15' in the on-line documentation)	INLONG command got an illegal string.	F_NUMFORMAT

16 ('Error 16' in the on-line documentation)	Parameters in memory are corrupted.	F_CRCPAR
17 ('Error 17' in the on-line documentation)	Programs in memory are corrupted.	F_CRCPRG
18 ('Error 18' in the on-line documentation)	Reset by CPU.	F_WDT
19 ('Error 19' in the on-line documentation)	User abort.	F_ABBRUCH
20 ('Error 20' in the on-line documentation)	FC communication error.	F_VLTCOM
21 ('Error 21' in the on-line documentation)	Number of SDO channels exceeded.	F_SDOCHN
22 ('Error 22' in the on-line documentation)	Feature protection error.	F_FEATUREPROT
23	Communication with ARM lost.	F_ARMCOM
25 ('Error 25' in the on-line documentation)	Limit switch activated.	F_ENDSCHALT
26 ('Error 26' in the on-line documentation)	SDO access error in SYSVAR or LINK command.	F_ILLGLSDO
27	Trial to execute a cleared or empty program.	F_NOPROG
39 ('Error 39' in the on-line documentation)	Wrong or no FPGA firmware loaded.	F_FPGA
40	Amplifier error.	F_AMP
42 ('Error 42' in the on-line documentation)	EtherCAT Master Error.	F_ECAT_MASTER
49 ('Error 49' in the on-line documentation)	Too many interrupt functions.	F_NOINTLEFT
50 ('Error 50' in the on-line documentation)	No amplifier power.	F_NO_24VOLT

the on-line documentation)		(deprecated)
50 ('Error 50' in the on-line documentation)	Minimum voltage of the amplifier has been undershot.	F_UMIN
51 ('Error 51' in the on-line documentation)	Too many nested function/ interrupts calls.	F_STACK
52 ('Error 52' in the on-line documentation)	Too many return() commands.	F_RETURN
56 ('Error 56' in the on-line documentation)	A floating point function was called with an invalid argument.	F_MATHERR
57 ('Error 57' in the on-line documentation)	State machine error.	F_STATEMACHINE
60 ('Error 60' in the on-line documentation)	Interrupt happened, but interrupt address is no longer valid.	F_INTPTR
62 ('Error 62' in the on-line documentation)	Error in verifying.	F_MEEP
63	Path control only: Path error.	F_PATHERR
64	Path control only: Time out in V24 path control.	F_TIMEOUT
65	Path control only.	F_PATHINT
70 ('Error 70' in the on-line documentation)	Too many DIM arrays defined.	F_DIM
71 ('Error 71' in the on-line documentation)	Invalid array index.	F_ARRBDS
73 ('Error 73' in the on-line documentation)	Array number does not exist.	F_LTARRAY
74 ('Error 74' in the on-line documentation)	Array is empty.	F_NOARRAY
75 ('Error 75' in the on-line documentation)	No more memory space for the new array defined by DIM.	F_NOSPACE

documentation)		
76 ('Error 76' in the on-line documentation)	Array size does not correspond to the size of the existing array.	F_ARRSIZERR
77 ('Error 77' in the on-line documentation)	Maximum temperature of the amplifier has been exceeded.	F_TEMP
78 ('Error 78' in the on-line documentation)	Maximum voltage of the amplifier has been exceeded.	F_UMAX
79 ('Error 79' in the on-line documentation)	Timeout while waiting for index.	F_TNDX
83 ('Error 83' in the on-line documentation)	Internal command error (illegal parameter or format or range)	F_CMDERR
84 ('Error 84' in the on-line documentation)	Too many TIME or PERIOD interrupts.	F_TIM
87 ('Error 87' in the on-line documentation)	Not enough memory for variables.	F_NOVARMEM
88 ('Error 88' in the on-line documentation)	CAN guarding error.	F_CANGUARD
89 ('Error 89' in the on-line documentation)	CAN send or receive error.	F_CANIO
90 ('Error 90' in the on-line documentation)	Memory locked.	F_MEMLOCK
91 ('Error 91' in the on-line documentation)	Illegal curve array in SETCURVE.	F_CURARR
92 ('Error 92' in the on-line documentation)	Encoder error.	F_ENCERR
93 ('Error 93' in the on-line documentation)	Stack overflow: Too many local variables or nested function calls.	F_DYNSTACK
94 ('Error 94' in the on-line documentation)	Out of dynamic memory.	F_DYNMEM

the on-line documentation)		
95 ('Error 95' in the on-line documentation)	Too many test indices in data logging command.	F_OPALINDX
96 ('Error 96' in the on-line documentation)	Code is too old for the current firmware.	F_ILLGLCODE
97 ('Error 97' in the on-line documentation)	Internal overcurrent detection of power stage.	F_IMAX
98 ('Error 98' in the on-line documentation)	Wrong direction after limit switch tripped and error reset.	F_LIMIT_VIOLATION
99 ('Error 99' in the on-line documentation)	I ² T Limit exceeded.	F_I2TLIMIT
100 ('Error 100' in the on-line documentation)	Communication with amplifier interrupted.	F_AMPCOM
101 ('Error 101' in the on-line documentation)	Illegal access to amplifier parameter.	F_AMPPARAM
102 ('Error 102' in the on-line documentation)	Command is not supported	F_NOTSUPPORTED
103 ('Error 103' in the on-line documentation)	MemoryDump (on-line documentation) error.	F_MEMDUMP
104	Divide by 0.	F_DIVIDEBY0
105 ('Error 105' in the on-line documentation)	EtherCAT Slave Error.	F_ECAT_SLAVE
106 ('Error 106' in the on-line documentation)	Profile Generator Error.	F_PROFGEN
107 ('Error 107' in the on-line documentation)	DIM array is too short.	F_DIMTOOSHORT
108 ('Error 108' in the on-line documentation)	Internal firmware error.	F_INTERNALERROR

documentation)		
109 ('Error 109' in the on-line documentation)	Option boot failure.	F_OPTIONBOOT
110 ('Error 110' in the on-line documentation)	Wrong command usage.	F_COMMAND_USAGE
111 ('Error 111' in the on-line documentation)	Illegal write access to process data.	F_PROCESSDATAUSAGE
112 ('Error 112' in the on-line documentation)	The size of string argument too long.	F_ARGTOOLONG
113 ('Error 113' in the on-line documentation)	Authentication failed.	F_AUTHFAILED
114 ('Error 114' in the on-line documentation)	Amplifier not ready.	F_AMP_NOTREADY
115 ('Error 115' in the on-line documentation)	Logic Supply Voltage too low.	F_LOGIC_UMIN
116	ZFC file update failed.	F_ZFCUPDATE
117 ('Error 117' in the on-line documentation)	Hardware Failure.	F_HWFAIL
118 ('Error 118' in the on-line documentation)	Amplifier's initialization order violated.	F_AMP_INIT_ORDER
119 ('Error 119' in the on-line documentation)	Hall sensor problem.	F_HALL
120 ('Error 120' in the on-line documentation)	Hall angle detection error.	F_HALL_ANGLE
121 ('Error 121' in the on-line documentation)	STO error.	F_STO
122 ('Error 122' in the on-line documentation)	Logic Supply Voltage too high.	F_LOGIC_UMAX

123 ('Error 123' in the on-line documentation)	PROFINET error.	F_PROFINET
xx ('Error xx' in the on-line documentation)	Internal error ##.	

7 Function Descriptions

7.1 ZbMocAutoClear

SIGNED16 ZbMocAutoClear(UNSIGNED16 h, UNSIGNED16 id);

Summary Clear the "Autostart" flag of the controller with the given ID. When the "Autostart" flag is clear, no program is started automatically after power-on.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.

Return Value [ZbMoc Error Codes](#)

Remarks If the "Autostart" flag was already clear when this function is called, then the function will do nothing. It is not an error to clear the flag when it is already clear.

Portability Firmware \geq 2.7

7.2 ZbMocAutoSet

SIGNED16 ZbMocAutoSet(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED8 prgnr);

Summary Set the "Autostart" flag of the given controller for the program defined by the given program number.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
prgnr = Program number.

Return Value [ZbMoc Error Codes](#)

Remarks If the "Autostart" flag was previously set for another program, then this flag is removed. Only one program per controller can be tagged with an "Autostart" flag. The program with the "Autostart" flag starts automatically after power-on or reset of the controller.

Portability Firmware \geq 2.7

7.3 ZbMocAxisParamRead(DEPRECATED)

**SIGNED16 ZbMocAxisParamRead(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED8 axnr,
ZbMocAxisParamS *para);**

Summary Read the axis parameters from the selected controller and axis number. The parameters are decoded and copied into the [ZbMocAxisParamS](#) structure. **Deprecated from firmware 9.00.00**

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.

axnr = Axis number (1...n) of the controller.

para = Pointer to a structure taking the read out axis parameters.

Return Value [ZbMoc Error Codes](#)

Remarks If the controller supports multiple axes, then each axis will maintain its own set of axis parameters.

This function reads all axis parameters supported by the selected controller. All parameters in [ZbMocAxisParamS](#) that are not supported by the controller, are set to zero.

If any conversion is necessary to get compatible results corresponding to the version of the controller, then this is done internally.

Portability 9.00 > Firmware ≥ 2.7

7.4 ZbMocAxisParamWrite(DEPRECATED)

```
SIGNED16 ZbMocAxisParamWrite(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED8 axnr,  
                             ZbMocAxisParamS *para);
```

Summary Write the axis parameters of the given axis number to the selected controller. The parameters are taken from the [ZbMocAxisParamS](#) structure, preprocessed if necessary, and saved in the controller. **Deprecated from firmware 9.00.00**

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
axnr = Axis number (1...n) of the controller.
para = Pointer to a structure holding the parameters.

Return Value [ZbMoc Error Codes](#)

Remarks If the controller supports multiple axes, then each axis will maintain its own set of axis parameters.

This function sets all axis parameters supported by the selected controller. All parameters in [ZbMocAxisParamS](#) that are not supported by the controller, are ignored.

If any conversion is necessary to get compatible results corresponding to the version of the controller, this is done internally before writing the settings.

Portability 9.00 > Firmware ≥ 2.7

7.5 ZbMocBlobInfoRequest

```
SIGNED16 ZbMocBlobInfoRequest(UNSIGNED16 h, UNSIGNED16 id, ZbMocBlobInfoS *blobinfos,  
                              UNSIGNED16 *numentries, UNSIGNED16 maxentries);
```

Summary Return the blob information headers of all saved blobs.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.

blobinfos = Pointer to an array of structures taking the blob information.
 numentries = Pointer to a variable taking the actual number of blob information headers returned.
 maxentries = Maximum number of blob information structures in the array.

Return Value [ZbMoc Error Codes](#)

Remarks Any kind of binary data can be stored in the controller in "blobs" (i.e. binary large objects). Each blob is identified by a unique blob number which can be found in the blob information header returned by this function. The blob number is used by [ZbMocBlobRead](#) to access this stored data.

If more than "maxentries" blobs have been saved in the controller, then only the first "maxentries" headers are returned. Note that "numentries" is the number of headers returned and NOT the number of headers that are stored on the controller.

This function is not available for serial (V24) communication below controller version 6.0.1.

Portability Firmware ≥ 5.17

7.6 ZbMocBlobRead

```
SIGNED16 ZbMocBlobRead(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 blobnum,
                      UNSIGNED8 *blobdata, UNSIGNED32 *blobdatalen,
                      UNSIGNED32 buflen);
```

Summary Read blob data that was saved in the selected controller.

Parameter h = Handle of the interface.
 id = Identification number (ID) of the controller.
 blobnum = Identification number of the blob.
 blobdata = Pointer to an array taking the received blob data.
 blobdatalen = Pointer to a variable receiving the length of the blob in bytes or the required length if the provided buffer was too small.
 buflen = Size of the array provided for taking the blob data.

Return Value [ZbMoc Error Codes](#)

Remarks The parameter "blobnum" is used as an identification number of the blob (i.e. binary large object) holding the data. This number is part of the structure that is returned by the [ZbMocBlobInfoRequest](#) function.

The provided buffer must be at least 8 bytes larger than the actual blob size. If the provided buffer is too small, then the ZBMOC_E_RXBUFFER_TOO_SMALL error code is returned. The actual blob size is given in "blobdatalen".

This function is not available for serial (V24) communication below controller version 6.0.1.

Portability Firmware ≥ 5.17

7.7 ZbMocBlobWrite

```
SIGNED16 ZbMocBlobWrite(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 blobnum,  
                        UNSIGNED8 *blobdata, UNSIGNED32 blobdatalen);
```

Summary Save blob data in the selected controller.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
blobnum = Identification number of the blob.
blobdata = Pointer to an array holding the blob data to be written.
blobdatalen = Length of the blob data (in bytes).

Return Value [ZbMoc Error Codes](#)

Remarks The parameter "blobnum" is used as an identification number of the blob (i.e. binary large object) holding the data. This number is part of the structure that is returned by the [ZbMocBlobInfoRequest](#) function.

The application program uses the 'ZbMocBlob...' functions to manage the storage of source code in the controller (i.e. source code is stored as blobs). The ApossIDE associates source code with the corresponding saved binary code by using the program number as the blob number. If program source is being saved on the controller (either by the ApossIDE or by the [ZbMocSourceSaveInController](#) function), then the user application must avoid using blob numbers in the range 0 to MOC_MAXPROGCNT-1 (see MOC_MAXPROGCNT in 'ZbMoc.h' for more information).

Blob numbers in the range 4096 to 65535 are reserved and should not be used.

This function is not available for serial (V24) communication below controller version 6.0.1.

Portability Firmware ≥ 5.17

7.8 ZbMocBreak

```
SIGNED16 ZbMocBreak(UNSIGNED16 h, UNSIGNED16 id);
```

Summary Transmit a "Break" command to the selected controller.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.

Return Value [ZbMoc Error Codes](#)

Remarks This initiates the following actions:

- Stops program execution (and also autostart).
- Stops all moving axes.
- Clears an error state if the controller is in an error state.

Depending on the type of interface and controller, it might be necessary to call this function repeatedly to stop a program which has the "Autostart" flag set. This is necessary since the first "Break" command results in a restart. A "Break" command acting on an "Autostart" program results in error state 19.

Portability Firmware ≥ 2.7

7.9 ZbMocBreakAll

SIGNED16 ZbMocBreakAll(UNSIGNED16 h);

Summary Transmit a "Break" command to ALL controllers connected to the selected interface. From the point of view of a single controller, this function is identical to [ZbMocBreak](#).

Parameter h = Handle of the interface.

Return Value [ZbMoc Error Codes](#)

Remarks Controllers connected on a CAN bus or other multi-point interfaces are only affected if their ID's are within the range defined in the 'ZbMocOpen...' function. Other controllers which may be connected to the same bus are not affected. If any controller is blocked by a previous request, then ZBMOC_E_BLOCKED is returned and no break command is sent to any controller. Otherwise, an attempt is made to break all controllers even if some attempts fail. If any attempts fail, then the first non-ZBMOC_OK return code is returned and subsequent non-ZBMOC_OK return codes are discarded.

Portability Firmware ≥ 2.7

7.10 ZbMocCanOpenDownloadSDO

SIGNED16 ZbMocCanOpenDownloadSDO(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 index, SIGNED32 *value, SIGNED32 cnt);

Summary Write all subindices of an SDO to the selected controller. The controller must be configured as a CAN open slave.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
index = Index of the SDO.
value = Pointer to a buffer containing the SDO data.
cnt = Number of subindices to be written.

Return Value [ZbMoc Error Codes](#)

Remarks "cnt" may be less than the actual number of subindices in the SDO. If the SDO contains subindices after "cnt", then they are not modified. Subindex 0 is expected to be the highest subindex number and will not be written by this routine.

Portability Firmware ≥ 5.21

7.11 ZbMocCanOpenInhibitPDO

SIGNED16 ZbMocCanOpenInhibitPDO(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED32 timeout);

SIGNED16 ZbMocCanOpenSlaveInhibitPDO(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED32 timeout,

```
void (*callback)(UNSIGNED16 h, UNSIGNED16 id));
```

Summary Set the "inhibit timeout" for the [ZbMocCanOpenQueuePDO](#) routine.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
timeout = Timeout in milliseconds.

Return Value [ZbMoc Error Codes](#)

Remarks PDO's queued with the [ZbMocCanOpenQueuePDO](#) routine will only be transmitted once the "inhibit timeout" expires.

ZbMocCanOpenSlaveInhibitPDO is an alternate version of this routine that includes a callback. If this version is used and a callback routine is specified, then it is called after each PDO is sent.

Portability Firmware ≥ 5.21

7.12 ZbMocCanOpenPollPDO

```
SIGNED16 ZbMocCanOpenPollPDO(UNSIGNED16 *h, UNSIGNED16 *id, double *time,  
                             SIGNED32 *pdo, SIGNED32 *len, UNSIGNED8 *dat,  
                             SIGNED32 buflen);
```

Summary Poll for any PDO that has been received.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
time = Time at which the PDO was received (in seconds since ZbMoc was started).
pdo = Type of PDO: 1 - PDO1, 2 - PDO2, etc.
len = Length of the PDO data (in bytes).
dat = Pointer to array of bytes which is to receive the PDO data.
buflen = Length of the "dat" buffer (in bytes).

Return Value 0 - PDO queue is empty. No PDO is returned.
>0 - Number of PDOs in queue (before poll). The oldest PDO in the queue is returned.
<0 - [ZbMoc Error Codes](#)

Remarks If a PDO callback routine has NOT been defined (i.e. by calling [ZbMocSetCallbackPDO](#)), then incoming PDO messages are queued by ZbMoc. Messages in the queue may be retrieved by calling this routine. The return code will be the number of PDO messages in the queue when the routine is called. So 0 will be returned if the queue is empty. If the queue is not empty, then the oldest PDO message will be returned and that message will be removed from the queue. ZbMoc will queue up to 1000 PDO messages. After that, the oldest PDO message is discarded whenever a new PDO message is received. Note that all of the parameters (except "buflen") point to variables that will receive information about the PDO. "len" will be the actual number of data bytes received in the PDO and the PDO data will be copied into "dat". The smaller of "len" and "buflen" bytes will be copied.

Portability Firmware ≥ 5.21

7.13 ZbMocCanOpenQueuePDO

```
SIGNED16 ZbMocCanOpenQueuePDO(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 type,  
                               UNSIGNED8 *inarray, UNSIGNED8 len);
```

Summary Transmit a PDO message to the selected controller. The controller must be configured as a CAN open slave. The message is only transmitted when the "inhibit timeout" expires.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
type = Type of PDO: 1 - PDO1, 2 - PDO2, etc.
inarray = Pointer to array of bytes which is to be transmitted.
len = Size of the array (in bytes).

Return Value [ZbMoc Error Codes](#)

Remarks The maximum length of a PDO message depends on the type of interface being used. A PDO message queued by this routine will only be sent when the "inhibit timeout" expires. This timeout can be set by the [ZbMocCanOpenInhibitPDO](#) routine. Only one PDO message is queued; if a second message is queued before the timeout expires, then the previous message is overwritten.

Portability Firmware ≥ 5.21

7.14 ZbMocCanOpenReadSegmentedSDO

```
SIGNED16 ZbMocCanOpenReadSegmentedSDO(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 index,  
                                         UNSIGNED8 subindex, UNSIGNED8 *value, SIGNED32 *size);
```

Summary Read data using segmented transfer from an SDO of the selected controller. The controller must be configured as a CAN open slave.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
index = Index of the SDO.
subindex = Subindex of the SDO.
value = Pointer to the received SDO data.
size = Pointer to a variable to receive the size of data

Return Value [ZbMoc Error Codes](#)

Portability Firmware ≥ 5.21

7.15 ZbMocCanOpenReadSDO

```
SIGNED16 ZbMocCanOpenReadSDO(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 index,  
                              UNSIGNED8 subindex, SIGNED32 *value);
```

Summary Read data from an SDO of the selected controller. The controller must be configured as

a CAN open slave.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
index = Index of the SDO.
subindex = Subindex of the SDO.
value = Pointer to the received SDO data.

Return Value [ZbMoc Error Codes](#)

Remarks The size of the "value" array (i.e. "len") will typically be 4 bytes. If the actual number of bytes received for the SDO (i.e. "len_rx_data") is larger than "len", then the additional received bytes are ignored.

Portability Firmware ≥ 5.21

7.16 ZbMocCanOpenStart

```
SIGNED16 ZbMocCanOpenStart(UNSIGNED16 h, UNSIGNED16 id);
```

Summary Configure and start the selected controller as a CAN open slave.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.

Return Value [ZbMoc Error Codes](#)

Remarks This routine will configure the controller as a CANopen slave. It must be called before the other SDO/PDO routines can be called.

Portability Firmware ≥ 5.21

7.17 ZbMocCanOpenUploadSDO

```
SIGNED16 ZbMocCanOpenUploadSDO(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 index,  
                                SIGNED32 *value, SIGNED32 *cnt);
```

Summary Read all subindices of an SDO from the selected controller. The controller must be configured as a CAN open slave.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
index = Index of the SDO.
value = Pointer to a buffer to receive the SDO data.
cnt = Pointer to a variable to receive the number of subindices that were received.

Return Value [ZbMoc Error Codes](#)

Remarks The size of the "value" array MUST be 255 elements long. This is the maximum number of subindices that may be returned for any SDO (i.e. subindex 0-254).

Portability Firmware ≥ 5.21

7.18 ZbMocCanOpenWritePDO

```
SIGNED16 ZbMocCanOpenWritePDO(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 type,  
                               UNSIGNED8 *value, UNSIGNED8 len);
```

Summary Transmit a PDO message to the selected controller. The controller must be configured as a CAN open slave.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
type = Type of PDO: 1 - PDO1, 2 - PDO2, etc.
value = Pointer to array of bytes which is to be transmitted.
len = Size of the array (in bytes).

Return Value [ZbMoc Error Codes](#)

Remarks The maximum length of a PDO message depends on the type of interface being used.

Portability Firmware ≥ 5.21

7.19 ZbMocCanOpenWriteSegmentedSDO

```
SIGNED16 ZbMocCanOpenWriteSegmentedSDO(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 index,  
                                         UNSIGNED8 subindex, UNSIGNED8* value, SIGNED32 size);
```

Summary Transmit data using segmented transfer to an SDO of the selected controller. The controller must be configured as a CAN open slave.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
index = Index of the SDO.
subindex = Subindex of the SDO.
value = Value to be transmitted to the SDO.
size = The size of data to be written

Return Value [ZbMoc Error Codes](#)

Portability Firmware ≥ 5.21

7.20 ZbMocCanOpenWriteSDO

```
SIGNED16 ZbMocCanOpenWriteSDO(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 index,  
                               UNSIGNED8 subindex, SIGNED32 value);
```

Summary Transmit data to an SDO of the selected controller. The controller must be configured as a CAN open slave.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
index = Index of the SDO.
subindex = Subindex of the SDO.

value = Value to be transmitted to the SDO.

Return Value [ZbMoc Error Codes](#)

Remarks The size of the "value" array (i.e. "len") must be 4 bytes. Other lengths are not supported.

Portability Firmware ≥ 5.21

7.21 ZbMocCanReadRaw

```
SIGNED16 ZbMocCanReadRaw(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED8 *msg,  
                          UNSIGNED32 timeout);
```

Summary Read a single (raw) CAN message of the selected controller.

Parameter h = Handle of the interface.

id = Identification number (ID) of the controller.

msg = Pointer to a data section of 8 bytes length taking the received CAN message data.

timeout = Maximum period of time [ms] for receiving the message. 0 = Do not wait (i.e. a message must present immediately). If there is no message received within the defined time slot, then the function returns an error code.

Return Value ZBMOC_OK - Message was received and is copied to the data buffer.

ZBMOC_E_CAN_TIMEOUT - No message received within time slot.

[ZbMoc Error Codes](#)

Remarks This function is only available for CAN bus based communication.

The user data section of the CAN message has a length of 8 bytes. The data is copied into the "msg" buffer exactly the way it was received.

Portability Firmware ≥ 2.7

7.22 ZbMocCanWriteRaw

```
SIGNED16 ZbMocCanWriteRaw(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED8 *msg);
```

Summary Transmit a single CAN message to the selected controller.

Parameter h = Handle of the interface.

id = Identification number (ID) of the controller.

msg = Pointer to a data section of 8 bytes length holding the user data of the CAN message to send.

Return Value [ZbMoc Error Codes](#)

Remarks This function is only available for CAN bus based communication.

The user data section of the CAN message has a length of 8 bytes. The data is transmitted exactly the way it is given in the "msg" data buffer.

Portability Firmware ≥ 2.7

7.23 ZbMocClose

SIGNED16 ZbMocClose(UNSIGNED16 h);

Summary Close the given interface.

Parameter h = Handle of the interface.

Return Value [ZbMoc Error Codes](#)

Remarks This function should be called before the application program exits.

7.24 ZbMocCloseAll

SIGNED16 ZbMocCloseAll(void);

Summary Close all open interfaces.

Return Value [ZbMoc Error Codes](#)

Remarks This function can be used instead of [ZbMocClose](#).

7.25 ZbMocClearError

SIGNED16 ZbMocClearError(UNSIGNED16 h, UNSIGNED16 id);

Summary Clear an error state in the communication layer of the given interface.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.

Return Value [ZbMoc Error Codes](#)

Remarks If an error state was present, then the communication layer is reset (i.e. the transmit and receive buffers of the interface are cleared).

Portability Firmware \geq 2.7

7.26 ZbMocCNFRestoreFromFile

**SIGNED16 ZbMocCNFRestoreFromFile(UNSIGNED16 h, UNSIGNED16 id,
const STRUTF8 *filename, BOOLEAN prmf1g);**

Summary Read a configuration (*.zbc/*.cnf) file and download the data (parameters and arrays) to the selected controller.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
filename = Pointer to a string holding the filename and path of the configuration file to be read. This should include the file extension.
prmf1g = Flag indicating which parameters to restore. If this is FALSE, then only user

parameters and DIM arrays are restored. Otherwise, all parameters (i.e. global parameters, axis parameters, etc.) are restored.

Return Value [ZbMoc Error Codes](#)

Remarks With zub controllers that support SDO-based communication, this function can be executed regardless of the controller's execution state. However, saving large arrays (i.e. totalling more than 1000 elements) can influence the speed of program execution significantly because there is no ApossIDE command, interrupt, or error handling processed during the final persistent saving of the arrays. It is recommended that this routine not be used when a user program is executing.

The filename may be either UTF-8 or ANSI encoded.

Portability Firmware \geq 2.7

7.27 ZbMocCNFSaveToFile

```
SIGNED16 ZbMocCNFSaveToFile(UNSIGNED16 h, UNSIGNED16 id, const STRUTF8 *filename);
```

Summary Read the parameter settings and array definitions from the given controller and write them into a configuration (*.zbc) file.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
filename = Pointer to a string holding the filename and path of the configuration file to be written. This should include the file extension.

Return Value [ZbMoc Error Codes](#)

Remarks If the given configuration file exists, then it is overwritten!

The filename may be either UTF-8 or ANSI encoded.

Portability Firmware \geq 2.7

7.28 ZbMocConnect

```
SIGNED16 ZbMocConnect(UNSIGNED16 h, UNSIGNED16 id);
```

Summary Complete a pending connection or re-establish a disconnected connection.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.

Return Value [ZbMoc Error Codes](#)

Remarks When a USB interface is opened using [ZbMocOpenUsb](#), the controllers on the interface are not physically opened for communication. Instead, they will be in a "pending" state (see the [ZbMocMoconInfoS](#) structure). The **ZbMocConnect** routine must be called to finalize the connection before communication with the controller can be started.

7.29 ZbMocContinue

SIGNED16 ZbMocContinue(UNSIGNED16 h, UNSIGNED16 id);

Summary Continue a program that was previously interrupted by a "Break" command.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.

Return Value [ZbMoc Error Codes](#)

Remarks

Portability Firmware \geq 2.7

7.30 ZbMocDebugPrint

void ZbMocDebugPrint(long level, const char *str);

Summary Insert a string into the debug log file.

Parameter level = Debug level.
str = Text to be written to the debug file.

Remarks

7.31 ZbMocDebugPrintw

void ZbMocDebugPrintw(long level, const wchar_t *str);

Summary Insert a Unicode string into the debug log file.

Parameter level = Debug level.
str = Text to be written to the debug file.

Remarks

7.32 ZbMocDisconnect

SIGNED16 ZbMocDisconnect(UNSIGNED16 h, UNSIGNED16 id);

Summary Disconnect a controller.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.

Return Value [ZbMoc Error Codes](#)

Remarks This can be used to disconnect a USB controller.

7.33 ZbMocExec

SIGNED16 ZbMocExec(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 prgnr);

Summary Start the program with the given program number on the selected controller.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
prgnr = Program number.

Return Value [ZbMoc Error Codes](#)

Remarks

Portability Firmware \geq 2.7

7.34 ZbMocExecTemp

SIGNED16 ZbMocExecTemp(UNSIGNED16 h, UNSIGNED16 id, BOOLEAN init);

Summary Start the previously downloaded "temporary" program.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
init = TRUE (recommended) if internal variables (e.g. MNU point, etc.) must be initialized before starting.

Return Value [ZbMoc Error Codes](#)

Remarks The temporary program is the program that was downloaded last and is normally not yet stored under a program number.

See also the [ZbMocProgramSave](#) and [ZbMocProgramSaveAs](#) functions.

Portability Firmware \geq 2.7

7.35 ZbMocGetAxisPosition

SIGNED16 ZbMocGetAxisPosition(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED8 axnr, SIGNED32 encoder, SIGNED32 *position);

Summary Read the position of the given axis of the selected controller.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
axnr = Number of the axis: 1...n.
encoder = Encoder to be used: 0 - Slave, 1 - Master.
position = Pointer to a variable taking the returned position value.

Return Value [ZbMoc Error Codes](#)

Remarks The application program calling up this function is responsible for ensuring that the given axis number exists in the selected controller.

Portability Firmware \geq 2.7

7.36 ZbMocGetError

SIGNED16 ZbMocGetError(UNSIGNED16 h);

Summary Return the error state of the communication layer of the given interface.

Parameter h = Handle of the interface.

Return Value [ZbMoc Error Codes](#)

Remarks An error state of the communication layer can be cleared by using the [ZbMocClearError](#) function. This also resets the transmit and receive buffers. Note that this is the PC and communication-based error state; it is NOT the error state of any controller on the interface.

7.37 ZbMocGetIFDriverID

SIGNED16 ZbMocGetIFDriverID(UNSIGNED16 h, STRUTF8 *driverid);

Summary Get the version string of the low level interface driver currently in use (if available).

Parameter h = Handle of the interface.

driverid = Pointer to a string of at least 80 characters taking the identification string of the low level driver.

Return Value [ZbMoc Error Codes](#)

Remarks The given identification string depends on the interface specified by the connection handle. A ZBMOC_E_BADHANDLE error is returned if this handle is invalid.

A maximum of 80 bytes (including the '\0'-terminator) will be returned. The string will always be '\0'-terminated.

7.38 ZbMocGetNumber

UNSIGNED16 ZbMocGetNumber(UNSIGNED16 h);

Summary Return the number of connected controllers on the given interface.

Parameter h = Handle of the interface.

Return Value Number of connected controllers.

Remarks Controllers that have an ID that was not within the ID range defined during execution of the 'ZbMocOpen...' function for the interface, are not included in the returned number. ID's within the range but for which no controller could be found, are also not included in the returned number. However, ID's for controllers that were found but that failed to respond properly (and therefore are not accessible) ARE included in the returned number. Note that the [ZbMocMoconInfo](#) and [ZbMocMoconInfoIdx](#) routines

will return FALSE for these controllers.

For the serial (V24) interface, the number of controllers returned is always 1.

If the interface handle is invalid, then 0 is returned.

7.39 ZbMocGetZbMocVersion

```
void ZbMocGetZbMocVersion(STRUTF8 *version);
```

Summary Return the version string of the active ZbMoc DLL.

Parameter version = Pointer to a character string to receive the version string.

Remarks A maximum of 100 bytes (including the '\0'-terminator) will be returned. The string will always be '\0'-terminated. Although the interface specifies a UTF-8 encoded string, no UTF-8 characters will ever be returned; only characters in the range 0-127 will be returned.

7.40 ZbMocMemoryDump

```
SIGNED16 ZbMocMemoryDump(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 arrnr,  
                          const STRUTF8 *filename);
```

Summary Read the specified array from the given controller and write it into a file. Note that the file will have the same format as files created by the **MemoryDump (on-line documentation)** command.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
arrnr = Number of the array (0...n or 0xFFFF for dynamic memory).
filename = Pointer to a string holding the filename and path of the file to be written.

Return Value [ZbMoc Error Codes](#)

Remarks If the given file exists, then it is overwritten!

The filename may be either UTF-8 or ANSI encoded.

7.41 ZbMocMemoryLock

```
SIGNED16 ZbMocMemoryLock(UNSIGNED16 h, UNSIGNED16 id, BOOLEAN *locked);
```

Summary Test if memory has been locked.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
locked = Pointer to variable taking the result flag.

Return Value [ZbMoc Error Codes](#)

Remarks This function is intended for use only with MCO305 controllers. For all other controllers,

FALSE will be returned.

A program cannot be executing when this routine is called. If a program is executing, then the routine will return a ZBMOC_E_BREAK_REQUIRED error.

If an error return code is returned, then "locked" is NOT returned.

Portability Firmware \geq 5.17

7.42 ZbMocMoconBusyWait

SIGNED16 ZbMocMoconBusyWait(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED32 timeout);

Summary This routine will wait until a controller is no longer busy.

Parameter h = Handle of the interface.
id = Identification number (ID) of the control unit
timeout = Length of time to wait (in ms).

Return Value ZBMOC_OK - Controller is not busy.
ZBMOC_E_IS_BUSY - Controller is still busy after timeout.
[ZbMoc Error Codes](#)

Remarks This routine waits in increments of up to 100ms.

7.43 ZbMocMoconClearFactory

SIGNED16 ZbMocMoconClearFactory(UNSIGNED16 h, UNSIGNED16 id);

Summary Reset the selected controller to the factory settings. Every parameter is overwritten by default values and all programs are erased.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.

Return Value [ZbMoc Error Codes](#)

Remarks It is *not* recommended that this function be used on MACS or VLT/SyncPos controllers! Due to hardware limitations of the MACS and VLT/SyncPos controllers, it is not possible to reset them completely using a software command. If this function is used on one of these controllers, then the controller must be switched off and on again before any other function is executed!

If the complete memory content of a controller is to be reset, then it is recommended that it be done by calling [ZbMocMoconDeleteEeprom](#) and then switching the controller off and on again!

Portability Firmware \geq 2.7

7.44 ZbMocMoconClearParameters

SIGNED16 ZbMocMoconClearParameters(UNSIGNED16 h, UNSIGNED16 id);

Summary Reset all global and axis parameters on the controller to the factory settings.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.

Return Value [ZbMoc Error Codes](#)

Remarks Every parameter on the controller is overwritten by default values.

Portability Firmware ≥ 2.7

7.45 ZbMocMoconClearPrograms

```
SIGNED16 ZbMocMoconClearPrograms(UNSIGNED16 h, UNSIGNED16 id);
```

Summary Erase all programs on the selected controller.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.

Return Value [ZbMoc Error Codes](#)

Remarks

Portability Firmware ≥ 2.7

7.46 ZbMocMoconDeleteEeprom

```
SIGNED16 ZbMocMoconDeleteEeprom(UNSIGNED16 h, UNSIGNED16 id);
```

Summary Erase the content of the non-volatile memory (EEPROM) of the controller.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.

Return Value [ZbMoc Error Codes](#)

Remarks It is recommended that the controller be switched off and on again after calling **ZbMocMoconDeleteEeprom**. This function does not erase the content of the volatile memory (RAM). Switching the controller off and on again, makes sure that the erased content of the EEPROM is copied into the RAM again and both memory units are consistent. Otherwise, there is the risk that the existing content of the RAM is copied again into the non-volatile memory (EEPROM) by accident (e.g. by saving a program).

Portability Firmware ≥ 2.7

7.47 ZbMocMoconExecutionStatus

```
SIGNED16 ZbMocMoconExecutionStatus(UNSIGNED16 h, UNSIGNED16 id,  
                                     BOOLEAN refresh_first, BOOLEAN *is_executing,  
                                     SIGNED16 *errnum, UNSIGNED32 *status);
```

Summary Return the program execution state of the selected controller.

Parameter h = Handle of the interface.
 id = Identification number (ID) of the controller.
 refresh_first = Refresh request:
 - TRUE - Wait until an updated state is available.
 - FALSE - Return buffered state.
 is_executing = Pointer to a variable taking the program execution state:
 - TRUE - Controller is executing a program.
 - FALSE - Controller does not execute a program.
 errnum = Pointer to a variable taking the error code of the controller (or 0 if controller is not in error state).
 status = Pointer to a variable taking the 4 byte status.

Return Value [ZbMoc Error Codes](#)

Remarks ZbMoc caches the current state of the controller. This state is updated periodically during the normal operation of ZbMoc whether or not the application is calling any ZbMoc functions. The frequency of this update depends on several factors including the type of interface, the rate at which print messages are being polled, and which other ZbMoc functions are being called by the application. If an up-to-date status is not critical to the calling application, then "refresh_first" should be set to FALSE. This will return the cached status and avoid putting any additional load of the controller. If an up-to-date status is critical, then "refresh_first" should be set to TRUE. This will make an explicit request to the controller for status information and will not return to the calling application until the controller has replied. Note that in the case of a connection to a Danfoss VLT5000 device, "refresh_first" should always be set to TRUE.

Portability Firmware \geq 2.7: No support of the V24 serial interface.
 Firmware \geq 5.18: Support for all types of interfaces

7.48 ZbMocMoconFind

```
#define ZBMOC_FIND_INDEX 0    // Search by index.
#define ZBMOC_FIND_SERIAL 1   // Search by serial number.
#define ZBMOC_FIND_IPADDR 2   // Search by IP address.
```

```
SIGNED32 ZbMocMoconFind(UNSIGNED16 h, UNSIGNED16 how, UNSIGNED32 value);
```

Summary Find a controller ID based on one of several search criteria.

Parameter h = Handle of the interface.
 how = Search method:
 0 - Search by index.
 1 - Search by serial number.
 2 - Search by IP address.
 value = Search value corresponding to search method.

Return Value \geq 0 - ID of the controller.
 < 0 - [ZbMoc Error Codes](#)

Remarks The returned controller ID can be used with the [ZbMocMoconInfo](#) routine or any of the

other ZbMoc routines. Note that not all search methods can be used with all interface types. For example, searching by IP address will find no controllers on a USB interface.

In certain cases, this routine may find a controller that was not available when the interface was first opened. For example, when a TCP interface is being used and a controller is powered on after the interface is opened, then this routine may find the controller if the search is being made by IP address.

7.49 ZbMocMoconIdent

```
SIGNED16 ZbMocMoconIdent(UNSIGNED16 h, UNSIGNED16 id, ZbMocMoconIdentS *ident);
```

Summary Return identification information about the specified controller.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
ident = [ZbMocMoconIdentS](#) structure that will received a copy of the data.

Return Value [ZbMoc Error Codes](#)

Remarks The returned information is read out of the controller during execution of the 'ZbMocOpen...' function. This routine can be used when the ID's of the controllers are known. If the controller ID's are not known, then [ZbMocMoconInfoIdx](#) can be used. Note that a call to this routine is faster than a call to [ZbMocMoconInfo](#).

7.50 ZbMocMoconInfo

```
SIGNED16 ZbMocMoconInfo(UNSIGNED16 h, UNSIGNED16 id, ZbMocMoconInfoS *info);
```

Summary Query general information about the specified controller.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
info = [ZbMocMoconInfoS](#) structure that will received a copy of the info data.

Return Value [ZbMoc Error Codes](#)

Remarks The returned information is read out of the controller during execution of the 'ZbMocOpen...' function. This routine can be used when the ID's of the controllers are known. If the controller ID's are not known, then [ZbMocMoconInfoIdx](#) can be used.

7.51 ZbMocMoconInfoIdx

```
SIGNED16 ZbMocMoconInfoIdx(UNSIGNED16 h, UNSIGNED16 idx, ZbMocMoconInfoS *info);
```

Summary Return information about the selected controller.

Parameter h = Handle of the interface.
idx = Index of the controller: 0...ZbMocGetNumber()-1
info = Structure that will received a copy of the info data.

Return Value [ZbMoc Error Codes](#)

Remarks The returned information is read out of the controller during execution of the 'ZbMocOpen...' function. This routine can be used when the ID's of the controllers are not known. If the controller ID's are known, then [ZbMocMoconInfo](#) can be used.

7.52 ZbMocMoconNameSet

```
SIGNED16 ZbMocMoconNameSet(UNSIGNED16 h, UNSIGNED16 id, const STRUTF8 *name);
```

Summary Set the ASCII-based project name of the selected controller.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
name = Pointer to a data buffer holding the name.

Return Value [ZbMoc Error Codes](#)

Remarks The name can be a maximum of 10 characters long and should not contain any special (i.e. non-printable) characters. If the name is less than 10 characters long, then it will be padded with space characters. If the name is longer than 10 characters, then it will be truncated. For newer SDO-based controllers, then name cannot be longer than 8 characters.

Portability Firmware ≥ 2.7

7.53 ZbMocMoconSaveRam

```
SIGNED16 ZbMocMoconSaveRam(UNSIGNED16 h, UNSIGNED16 id);
```

Summary Save the content of the RAM in non-volatile memory.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.

Return Value [ZbMoc Error Codes](#)

Remarks The execution of this command may take some time (up to 20 seconds) depending on the type of non-volatile memory used in the controller.

If the controller has battery-buffered RAM, then it is not necessary to execute this function.

Portability Firmware ≥ 2.7

7.54 ZbMocMoconSaveRamSection

```
#define MOC_RAMSECTION_AXISPAR      0    // Save axis parameters.
#define MOC_RAMSECTION_GLOBALPAR    1    // Save global parameters.
#define MOC_RAMSECTION_PROGRAMS     2    // Save programs.
#define MOC_RAMSECTION_ARRAYS       3    // Save arrays.
#define MOC_RAMSECTION_ALL           4    // Save all.
```

```
#define MOC_RAMSECTION_USERPAR      5    // Save user parameters.
#define MOC_RAMSECTION_IAXPAR      6    // Save internal axis parameters.

SIGNED16 ZbMocMoconSaveRamSection(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED8 section);
```

Summary Save the content of the specified RAM section in non-volatile memory.

Parameter h = Handle of the interface.
 id = Identification number (ID) of the controller.
 section = ID of the RAM section to be saved (see above).

Return Value [ZbMoc Error Codes](#)

Remarks This function and the selective saving of a RAM section, is available only for controllers with a firmware version of 6.4.3 or later. For controllers older than this, [ZbMocMoconSaveRam](#) must be used.

Note that saving RAM strongly influences the speed of program execution. There is no ApossIDE command, interrupt, or error handling processed during execution of this function. This means that the program might not react for some seconds depending on the selected RAM section. It is strongly recommended that this function NOT be called for saving programs, arrays, or the complete RAM while the controller is executing a program! Use the [ZbMocBreak](#) function to stop execution before saving RAM!

The execution of this command might take some seconds, depending on the type of non-volatile memory used in the controller and the RAM section selected.

If the controller has battery-buffered RAM, then it is not necessary to execute this function.

Portability Firmware ≥ 6.0.1

7.55 ZbMocMoconSetCanParameters

```
SIGNED16 ZbMocMoconSetCanParameters(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED8 cannr,
                                     UNSIGNED8 baud);
```

Summary Define the CAN communication parameters of the selected controller.

Parameter h = Handle of the interface.
 id = Identification number (ID) of the controller.
 cannr = New CAN ID.
 baud = New CAN baud rate:
no	baud rate
 1 | 10kbps
 2 | 20kbps
 3 | 50kbps
 4 | 100kbps
 5 | 125kbps
 6 | 250kbps
 7 | 500kbps

8 | 1000kbps

Return Value [ZbMoc Error Codes](#)

Remarks Any modifications to the CAN parameters are applied immediately. As a result, it is recommended that this function be used only if a non-CAN (e.g. serial) interface is being used. If a CAN interface is being used, then changing the CAN parameters will result in the connection to the controller being lost. In this case, it will be necessary to close and reopen the ZbMoc connection before the controller can be accessed again.

Portability Firmware \geq 2.7

7.56 ZbMocMoconState

```
SIGNED16 ZbMocMoconState(UNSIGNED16 h, UNSIGNED16 id);
```

Summary Return connection state of the selected controller.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.

Return Value ≥ 0 - Connection state of controller.
 < 0 - [ZbMoc Error Codes](#)

Remarks

7.57 ZbMocMoconWaitForString

```
SIGNED16 ZbMocMoconWaitForString(UNSIGNED16 h, UNSIGNED16 id, char *target,
                                  UNSIGNED32 timeout);
```

Summary Wait until the given string is received via serial port of the selected controller.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
target = Pointer to a character array holding the string to wait for.
timeout = Maximum period of time [ms] for receiving the given string. 0 = Do not wait (i.e. a string must present immediately). If the string is not received within the defined time slot, the function returns an error code.

Return Value [ZbMoc Error Codes](#)

Remarks Strings differing from the defined one are ignored.
This function is only available if serial (V24) communication is used.

Portability Firmware \geq 2.7

7.58 ZbMocOpenCanLpt

```
#define ZBMOC_CAN_SDOALT    0x0001 // Switch to alternate SDO CAN ID (0x680/0x100)
                               // if possible.
```

```
#define ZBMOC_CAN_NONMT      0x0002 // Do not send NMT when interface is opened.
#define ZBMOC_CAN_SDOPROBE  0x0004 // Probe non-responsive ID's for CANopen devices.
#define ZBMOC_CAN_LISTEN    0x0008 // Listen for CAN messages from all ID's.
#define ZBMOC_CAN_SDOFORCE   0x0010 // Force use of alternate SDO CAN ID
                                // (0x680/0x100).
#define ZBMOC_CAN_DEMOTE     0x0020 // Demote device types (to ZBMOC_CAN_SDOPROBE or
                                // ZBMOC_CAN_LISTEN).
```

```
typedef struct
{
```

```
    UNSIGNED16  lptnr;
    UNSIGNED16  irq;
    UNSIGNED16  mode;
    UNSIGNED16  baud;
    UNSIGNED16  scanfrom;
    UNSIGNED16  scantto;
    UNSIGNED32  glbdelay;
    UNSIGNED32  snddelay;
    UNSIGNED32  flags;
} ZbMocOpenCanLptParamS;
```

```
SIGNED16 ZbMocOpenCanLpt(ZbMocOpenCanLptParamS *param);
```

Summary Open the CAN interface using CAN-adapter for PC's auxiliary (centronics) port.

Parameter lptnr = Number of the auxiliary (centronics) port:

0 - LPT1

1 - LPT2

-1 - Check all ports automatically and evaluate corresponding IRQ.

irq = Interrupt used by the auxiliary (centronics) port.

mode = CAN card mode: 0 - SPP, 1 - EPP, 2 - PS2, 3 - Auto

baud = Baud rate of the CAN bus:

1 - 10kbps

2 - 20kbps

3 - 50kbps

4 - 100kbps

5 - 125kbps

6 - 250kbps

7 - 500kbps

8 - 1000kbps

scanfrom = First CAN ID to be checked for the presence of a device.

scantto = Last CAN ID to be checked for the presence of a device.

glbdelay = Delay (in ms) after a global message has been sent before another message may be sent.

snddelay = Delay (in ms) between sent messages for controllers older than firmware version 6.4.0.

flags = Flags:

Name | Bit | Description

-----|-----|-----

ZBMOC_CAN_SDOALT | 0x0001 | Use alternate SDO CAN ID (0x680/0x100).

ZBMOC_CAN_NONMT | 0x0002 | Do not send NMT when interface is opened.

ZBMOC_CAN_SDOPROBE | 0x0004 | Probe non-responsive ID's for CANopen devices.

ZBMOC_CAN_LISTEN | 0x0008 | Listen for CAN messages from all ID's.
 ZBMOC_CAN_SDOFORCE | 0x0010 | Force use of alternate SDO CAN ID (0x680/0x100).
 ZBMOC_CAN_DEMOTE | 0x0020 | Demote device types (to ZBMOC_CAN_SDOPROBE or ZBMOC_CAN_LISTEN).

Return Value ≥ 0 - Handle of the interface connection.
 < 0 - [ZbMoc Error Codes](#)

Remarks It is not possible to simultaneously open more than one CAN-LPT interface.

All controllers (within the defined ID range) are addressed and configured automatically for usage by ZbMoc. The version information and configuration of each controller is read and saved for later access by the application program via the [ZbMocMoconInfo](#) function. Controllers which have an ID outside the defined range are ignored.

If at least one controller is found, then the interface will be opened successfully and the handle of the interface that was established is returned. If no controller at all responds, then a ZBMOC_E_CAN_NO_SLAVE error code is returned and the connection is closed again.

ZbMoc will normally find and connect to only zub controllers. However, if the ZBMOC_CAN_SDOPROBE flag is specified, then ZbMoc will also search for other CANopen-compliant devices on the CAN bus. A subset of ZbMoc functionality (i.e. SDO and PDO usage) will be available for these devices. Also, if the ZBMOC_CAN_LISTEN flag is specified, then ZbMoc will listen for incoming PDO messages from ALL device ID's in the scan range (i.e. including devices that are not CANopen-compliant). In some situations, it may be useful to connect to a zub controller as if it were a simple CANopen device (or even a non-CANopen device). In this case, the ZBMOC_CAN_DEMOTE flag can be specified.

Portability Firmware ≥ 2.7

7.59 ZbMocOpenCanUsb

```
typedef struct
{
    UNSIGNED16  port;
    UNSIGNED16  baud;
    UNSIGNED16  scanfrom;
    UNSIGNED16  scantto;
    UNSIGNED32  flags;
} ZbMocOpenCanUsbParamS;

SIGNED16 ZbMocOpenCanUsb(ZbMocOpenCanUsbParamS *param);
```

Summary Open the CAN-USB interface for controller communication.

Parameter port = Com port: 1 - COM1, 2 - COM2, etc.
 baud = Baud rate of the CAN bus:
 - 1 - 10kbps
 - 2 - 20kbps
 - 3 - 50kbps

- 4 - 100kbps
- 5 - 125kbps
- 6 - 250kbps
- 7 - 500kbps
- 8 - 1000kbps

scanfrom = First CAN ID to be checked for the presence of a device.

scanto = Last CAN ID to be checked for the presence of a device.

flags = Flags (same as ZbMocOpenCanLpt).

Return Value ≥ 0 - Handle of the interface connection.
 < 0 - [ZbMoc Error Codes](#)

Remarks All controllers found on USB connections are configured automatically for usage by ZbMoc. The version information and configuration of each controller is read and saved for later access by the application program via the [ZbMocMoconInfo](#) function. Controllers which have an ID outside the defined range are ignored.

If at least one controller is found, then the interface will be opened successfully and the handle of the interface that was established is returned. If no controller at all responds, then a ZBMOC_E_CAN_NO_SLAVE error code is returned and the connection is closed again.

Portability Firmware ≥ 2.7

7.60 ZbMocOpenEtherCat

```
#define ZBMOC_CAT_SCANALL 0x0001 // Scan all ports.
```

```
typedef struct
{
    UNSIGNED8    netid[6];
    UNSIGNED16   scanfrom;
    UNSIGNED16   scanto;
    UNSIGNED32   flags;
    UNSIGNED32   mskb;
} ZbMocOpenEtherCatParamS;
```

```
SIGNED16 ZbMocOpenEtherCat(ZbMocOpenEtherCatParamS *param);
```

Summary Open the EtherCAT interface for controller communication.

Parameter netid = AMS Net ID of controller on the EtherCAT bus.
 scanfrom = First port to be checked for the presence of a device.
 scanto = Last port to be checked for the presence of a device.
 flags = Flags:
Name	Bit	Meaning
 ZBMOC_CAT_SCANALL | 0x0001 | Scan all ports.
 mskb = Milliseconds/kilobyte timeout value for download/upload.

Return Value ≥ 0 - Handle of the interface connection.
 < 0 - [ZbMoc Error Codes](#)

Remarks All controllers found on EtherCAT connections are configured automatically for usage by ZbMoc. The version information and configuration of each controller is read and saved for later access by the application program via the [ZbMocMoconInfo](#) function. Controllers which have an ID outside the defined range are ignored.

If at least one controller is found, then the interface will be opened successfully and the handle of the interface that was established is returned. If no controller at all responds, then a ZBMOC_E_NOCONTROLLER" error code is returned and the connection is closed again. If the EtherCAT master itself is not turned on, then a ZBMOC_E_COM_OPENERROR error code is returned and the connection is also closed.

Portability All controllers that contain an EtherCAT port.

7.61 ZbMocOpenSerial

```
#define ZBMOC_SER_SXP      0x0001    // Enable switching to SXP.
#define ZBMOC_SER_SD02    0x0002    // Enable SD02 communication.
#define ZBMOC_SER_XON     0x0004    // Enable Xon/Xoff support.
#define ZBMOC_SER_BREAK   0x0008    // Send "break" if no response.
#define ZBMOC_SER_RS485   0x0010    // RS485 connection.
```

```
typedef struct
{
    UNSIGNED16  comnr;
    UNSIGNED32  baud;
    UNSIGNED16  parity;
    UNSIGNED16  retry;
    UNSIGNED32  timeout;
    UNSIGNED32  overfeed;
    UNSIGNED32  flags;
} ZbMocOpenSerialParamS;
```

```
SIGNED16 ZbMocOpenSerial(ZbMocOpenSerialParamS *param, BOOLEAN *prot);
```

Summary Open an enhanced serial interface using COM communication. This will initially create a V24 connection at 9600 baud. If this is successful and the controller can support the X-protocol, then the interface will be automatically switched to an SXP interface at the specified baud rate.

Parameter comnr = Number of the COM port: 0 - COM1, 1 - COM2, etc.
 baud = Baud rate [bps] of the communication: 9600, 115200, etc.
 parity = Parity: 0 = None, 1 = Odd, 2 = Even, 3 = Mark, 4 = Space
 retry = The number of times to attempt to send an X-protocol message before assuming a loss of connection.
 timeout = The timeout (in milliseconds) between X-protocol message send retries.
 overfeed = The timeout (in milliseconds) between the end of a received X-protocol message and the start of the next sent message. This is used to allow a half-duplex RS485 converter time to switch modes.
 flags = Flags:
Name	Bit	Meaning
 ZBMOC_SER_SXP | 0x0001 | Enable switching to SXP.

ZBMOC_SER_SDO2 | 0x0002 | Enable SDO2 communication.
 ZBMOC_SER_XON | 0x0004 | Enable Xon/Xoff support.
 ZBMOC_SER_BREAK | 0x0008 | Send "break" if no response.
 ZBMOC_SER_RS485 | 0x0010 | RS485 connection.
 prot = Final protocol. This will be FALSE if the final connection is a V24 connection and TRUE if the final connection is an X-protocol connection.

Return Value ≥ 0 - Handle of the interface connection.
 < 0 - [ZbMoc Error Codes](#)

Remarks The controller is configured automatically for usage by ZbMoc. The version information and configuration of the controller is read and saved for later access by the application program via the [ZbMocMoconInfo](#) function. Controllers which have an ID outside the defined range are ignored.

If a controller is found, then the interface will be opened successfully and the handle of the interface that was established is returned. If no controller responds, then a ZBMOC_E_NOCONTROLLER error code is returned and the connection is closed again.

Portability Firmware ≥ 2.7

7.62 ZbMocOpenTcp

```
#define ZBMOC_TCP_TCP      0x0001    // Use TCP (rather than UDP).
#define ZBMOC_TCP_START   0x0002    // Reserved.
#define ZBMOC_TCP_SDO2    0x0004    // Enable SDO2 communication.
#define ZBMOC_TCP_SEARCH   0x0008    // Search for all controllers.
```

```
typedef struct
{
    BOOLEAN      local;
    UNSIGNED32    address;
    UNSIGNED32    port;
    UNSIGNED16    retry;
    UNSIGNED32    timeout;
    UNSIGNED32    flags;
} ZbMocOpenTcpParamS;
```

```
SIGNED16 ZbMocOpenTcp(ZbMocOpenTcpParamS *param);
```

Summary Opens the TCP interface for control unit communication.

Parameter local = Reserved. Set to FALSE.
 address = TCP address of host.
 port = TCP port to use.
 retry = The number of times to attempt to send a message before assuming a loss of connection.
 timeout = The timeout (in milliseconds) between message send retries.
 flags = Flags:

Name	Bit	Meaning
ZBMOC_TCP_TCP	0x0001	Use TCP. This must be set.
ZBMOC_TCP_START	0x0002	Reserved.

ZBMOC_TCP_SDO2 | 0x0004 | Enable SDO2 communication.

ZBMOC_TCP_SEARCH | 0x0008 | Search for all controllers.

Return Value ≥ 0 - Handle of the interface connection.

< 0 - [ZbMoc Error Codes](#)

Remarks If the ZBMOC_TCP_SEARCH flag is set, then all control units found on TCP connections are configured automatically for usage by the library. If this flag is not set, then only the specified IP address will be used. Note that ZBMOC_TCP_SEARCH will not find controllers with firmware older than 7.1.16. To connect to older controllers, the IP address must be explicitly specified.

If no controller at all is responding a ZBMOC_E_NOCONTROLLER error code is returned and the connection is closed again. Otherwise, the handle of the connection that was established is returned.

Portability All controllers that contain an Ethernet port.

7.63 ZbMocOpenUsb

```
#define ZBMOC_USB_ALLPID 0x0100 // Allow all known PIDs.
```

```
typedef struct
{
    UNSIGNED32 baud;
    UNSIGNED16 retry;
    UNSIGNED32 timeout;
    UNSIGNED32 flags;
    UNSIGNED32 latency;
} ZbMocOpenUsbParamS;
```

```
SIGNED16 ZbMocOpenUsb(ZbMocOpenUsbParamS *param);
```

Summary Open the USB serial interface for controller communication.

Parameter baud = Baud rate [bps] of the underlying USB chip interface. This should normally be set to 921600.

retry = The number of times to attempt to send an X-protocol message before assuming a loss of connection.

timeout = The timeout (in milliseconds) between X-protocol message send retries.

flags = Flags:

Name | Bit | Meaning

-----|-----|-----

ZBMOC_SER_SDO2 | 0x0002 | Enable SDO2 communication.

ZBMOC_USB_ALLPID | 0x0100 | Allow all known PIDs.

latency = Latency timer (ms). This should be set to 1.

Return Value ≥ 0 - Handle of the interface connection.

< 0 - [ZbMoc Error Codes](#)

Remarks All controllers found on USB connections are configured automatically for usage by ZbMoc. The version information and configuration of each controller is read and saved for later access by the application program via the [ZbMocMoconInfo](#) function.

If at least one controller is found, then the interface will be opened successfully and the handle of the interface that was established is returned. If no controller at all responds, then a ZBMOC_E_VLT_SCAN_NO_SLAVES error code is returned and the connection is closed again.

When a USB interface is opened using **ZbMocOpenUsb**, the controllers on the interface are not physically opened for communication. Instead, they will be in a "pending" state (see the [ZbMocMoconInfoS](#) structure). The [ZbMocConnect](#) routine must be called to finalize the connection before communication with the controller can be started.

Portability All controllers that contain a USB port.

7.64 ZbMocOpenV24

```
typedef struct
{
```

```
    UNSIGNED16  comnr;
    UNSIGNED32  flags;
```

```
} ZbMocOpenV24ParamS;
```

```
SIGNED16 ZbMocOpenV24(ZbMocOpenV24ParamS *param);
```

Summary Open the V24 serial interface for controller communication. This function is obsolete. It should be replaced with "ZbMocOpenSerial()".

Parameter comnr = Number of the COM port: 0 - COM1, 1 - COM2, etc.

flags = Flags:

Name | Bit | Meaning

-----|-----|-----

ZBMOC_SER_SDO2 | 0x0002 | Enable SDO2 communication.

ZBMOC_SER_XON | 0x0004 | Enable Xon/Xoff support.

ZBMOC_SER_BREAK | 0x0008 | Send "break" if no response.

Return Value ≥ 0 - Handle of the interface connection.
 < 0 - [ZbMoc Error Codes](#)

Remarks The V24 settings must be: 9600 baud, 8 data bits, 1 stop bit, no parity.

The controller is configured automatically for usage by ZbMoc. The version information and configuration of the controller is read and saved for later access by the application program via the [ZbMocMoconInfo](#) function.

If a controller is found, then the interface will be opened successfully and the handle of the interface that was established is returned. If no controller responds, then a ZBMOC_E_NOCONTROLLER error code is returned and the connection is closed again.

Portability Firmware ≥ 2.7

7.65 ZbMocOpenVlt

```
#define VLT_STANDARD 0
```

```

#define VLT_TSTDCOM    1
#define VLT_USEDCOM    2
#define VLT_SXP        3
#define VLT_USB        4
#define VLT_TCP        5
#define VLT_MCO        6
#define VLT_SIM        7

typedef struct
{
    UNSIGNED16  comnr;
    UNSIGNED32  baud;
    UNSIGNED16  retry;
    UNSIGNED32  timeout;
    UNSIGNED32  overfeed;
    char        drivername[80];
    UNSIGNED16  mode;
    UNSIGNED16  scanfrom;
    UNSIGNED16  scanto;
    UNSIGNED32  connid;
} ZbMocOpenVltParamS;

SIGNED16 ZbMocOpenVlt(ZbMocOpenVltParamS *param);

```

Summary Open the VLT interface using the RS485 or DCOM communication. This interface can only be used with Danfoss VLT500 and FC300-family devices.

Parameter

- comnr = Number of the COM port: 0 - COM1, 1 - COM2, etc.
- baud = Baud rate [bps] of the VLT/RS485 communication: 9600, 115200, etc.
- retry = The number of times to attempt to send an X-protocol message before assuming a loss of connection.
- timeout = The timeout (in milliseconds) between X-protocol message send retries.
- overfeed = The timeout (in milliseconds) between the end of a received X-protocol message and the start of the next sent message. This is used to allow a half-duplex RS485 converter time to switch modes.
- drivername = Name of the MCT10 communication driver.
- mode = Operating mode of the VLT interface:

no	name	meaning
0	VLT_STANDAR	VLT/RS485 communication
1	VLT_TSTDCOM	(Not supported)
2	VLT_USEDCOM	VLT communication via DCOM
3	VLT_SXP	(Not supported)
4	VLT_USB	(Not supported)
5	VLT_TCP	(Not supported)
6	VLT_MCO	(Not supported)
7	VLT_SIM	(Not supported)

- scanfrom = First VLT ID to be checked for the presence of a device.
- scanto = Last VLT ID to be checked for the presence of a device.
- connid = Connection ID for MTComm.dll (mode = VLT_USEDCOM).

Return Value ≥ 0 - Handle of the interface connection.
 < 0 - [ZbMoc Error Codes](#)

Remarks It is not possible to simultaneously open more than one VLT interface.

The default VLT settings typically in use (for the RS485 interface) are: 9600 baud, 8 data bits, 1 stop bit, even parity.

For VLT_STANDARD mode, the parameters "drivename" and "connid" are ignored. For VLT_USEDCOM mode, the parameters "comnr" and "baud" are ignored and "drivename" and "connid" have MCT10-specific values.

All controllers found on the VLT network are configured automatically for usage by ZbMoc. The version information and configuration of each controller is read and saved for later access by the application program via the [ZbMocMoconInfo](#) function. Controllers which have an ID outside the defined range are ignored.

If at least one controller is found, then the interface will be opened successfully and the handle of the interface that was established is returned. If no controller at all responds, then a ZBMOC_E_VLT_SCAN_NO_SLAVES error code is returned and the connection is closed again.

Portability Firmware ≥ 2.7

7.66 ZbMocOpenSim

```
#define ZBMOC_SIM_SDO2      0x0002    // Enable SDO2 communication.
#define ZBMOC_SIM_REMOTE    0x0004    // Connect to remote computer (i.e. use UDP
communication).
```

```
typedef struct
{
    UNSIGNED32  flags;
} ZbMocOpenSimParamS;
```

```
SIGNED16 ZbMocOpenSim(ZbMocOpenSimParamS* param);
```

Summary Open the zub Simulator interface for controller communication.

Parameter flags = Flags:

Name | Bit | Meaning

-----|-----|-----

ZBMOC_SIM_SDO2 | 0x0002 | Enable SDO2 communication.

ZBMOC_SIM_REMOTE | 0x0004 | Connect to remote computer.

Return Value ≥ 0 - Handle of the interface connection.
 < 0 - [ZbMoc Error Codes](#)

Remarks The controller is configured automatically for usage by ZbMoc. The version information and configuration of the controller is read and saved for later access by the application program via the [ZbMocMoconInfo](#) function. Controllers which have an ID outside the defined range are ignored.

If a controller is found, then the interface will be opened successfully and the handle of the interface that was established is returned. If no controller responds, then a ZBMOC_E_NOCONTROLLER error code is returned and the connection is closed again.

Portability ApossIDE 7.00.84

7.67 ZbMocParamRead

SIGNED16 ZbMocParamRead(UNSIGNED16 h, UNSIGNED16 id, ZbMocParamS *para);

Summary Read all global parameters from the selected controller. The parameters are decoded and copied into the [ZbMocParamS](#) structure.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
para = Pointer to a structure taking the parameters.

Return Value [ZbMoc Error Codes](#)

Remarks This function reads all global parameters supported by the selected controller. All parameters in [ZbMocParamS](#) that are not supported by the controller, are set to zero.

If any conversion is necessary to get compatible results corresponding to the version of the controller, then this is done internally.

Portability Firmware \geq 2.7

7.68 ZbMocParamWrite

SIGNED16 ZbMocParamWrite(UNSIGNED16 h, UNSIGNED16 id, ZbMocParamS *para);

Summary Write all global parameters to the selected controller. The parameters are taken from the [ZbMocParamS](#) structure, preprocessed if necessary, and saved in the controller.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
para = Pointer to structure holding the parameters to write.

Return Value [ZbMoc Error Codes](#)

Remarks This function sets all global parameters supported by the selected controller. All parameters in [ZbMocParamS](#) that are not supported by the controller, are ignored.

If any conversion is necessary to get compatible results corresponding to the version of the controller, this is done internally before writing the settings.

Portability Firmware \geq 2.7

7.69 ZbMocPollMessage

SIGNED16 ZbMocPollMessage(UNSIGNED16 h, SIGNED16 id, STRUTF8 *message);

Summary Retrieves the next message generated by a "print" command in the running application program or by a system message generated by the selected controller itself.

Parameter h = Handle of the interface.

id = Identification number (ID) of the controller.
 message = Pointer to a buffer taking the message.

Return Value 0 - Message returned.
 1 - No message available.
 <0 - [ZbMoc Error Codes](#)

Remarks If "id" is -1, then all controllers are polled up until a message is found. Subsequent controllers are not polled. If a message is read, then it will be returned in "message". Messages can be up to MOC_PRINTMESSAGEBUFFER_LEN bytes long. Any controller which is blocked by a previous request, will be treated as if no message is read.

UTF-8 encoded strings are returned only if the application program executing in the controller was written to return UTF-8 encoded strings.

Portability Firmware ≥ 2.7

7.70 ZbMocProbe

SIGNED16 ZbMocProbe(UNSIGNED16 h);

Summary Search for controllers on the given interface.

Parameter h = Handle of the interface.

Return Value [ZbMoc Error Codes](#)

Remarks This function will search the interface for controllers. This can be useful for interface types that support controllers that appear and disappear as they are powered up and down. For example, this behaviour is normal for USB and Ethernet devices. Note that calling this routine may change the number of controllers returned by [ZbMocGetNumber](#) and the order of the controllers returned by [ZbMocMoconInfoIdx](#). Controller identification numbers of any open controllers are NOT changed.

7.71 ZbMocProgramDelete

SIGNED16 ZbMocProgramDelete(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 prgnr);

Summary Deletes the program in the program list with the given program number.

Parameter h = Handle of the interface.
 id = Identification number (ID) of the control unit.
 prgnr = Number (0...94) used to register the temporary program in the list.

Return Value [ZbMoc Error Codes](#)

Remarks

Portability Firmware ≥ 2.7

7.72 ZbMocProgramList

```
typedef struct ZbMocProgramS_
{
    UNSIGNED16 index;           // Program number.
    STRUTF8    name[24+1];     // Program name.
    BOOLEAN    autostart;      // Autostart: TRUE/FALSE
} ZbMocProgramS;

SIGNED16 ZbMocProgramList(UNSIGNED16 h, UNSIGNED16 id, ZbMocProgramS *pl,
                          UNSIGNED16 maxentries);
```

Summary Return a list of all program stored in the selected controller.

Parameter h = Handle of the interface.
 id = Identification number (ID) of the controller.
 pl = Pointer to an array of "ZbMocProgramS" structures taking the program numbers, names, and the state of the "Autostart" flag.
 maxentries = Number of elements in the array.

Return Value ≥ 0 - Number of programs returned.
 < 0 - [ZbMoc Error Codes](#)

Remarks The returned list of programs includes all programs which were previously stored using one of the [ZbMocProgramSave](#) or [ZbMocProgramSaveAs](#) functions. Any downloaded "temporary" program, is not included.

If more than "maxentries" programs have been saved in the controller, then only the first "maxentries" programs are returned. Note that the return code is the number of programs returned and NOT the number of programs that are stored on the controller. The calling application must set "maxentries" to at least "MOC_MAXPROGCNT" to ensure that all programs are retrieved (see MOC_MAXPROGCNT in 'ZbMoc.h' for more information).

UTF-8 encoded program names are returned only if program names have been stored using UTF-8 encoded names. Program names will always be null-terminated.

Portability Firmware ≥ 2.7

7.73 ZbMocProgramLoad

```
SIGNED16 ZbMocProgramLoad(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED32 prglen,
                          UNSIGNED8 (*prg)(UNSIGNED32));
```

Summary Download a binary (i.e. compiled) program to the selected controller.

Parameter h = Handle of the interface.
 id = Identification number (ID) of the controller.
 prglen = Length of the binary program in number of bytes.
 prg = Pointer to a function returning one byte of program code.


Return Value [ZbMoc Error Codes](#)

Remarks The application source program (*.m) must be compiled into binary code by the ApossIDE compiler before a download can take place. The binary code must then be

downloaded to the controller. It is not possible to download application source code directly.

This function calls the function "(*prg)()" to get each byte of binary program code (up to "prglen" bytes). The argument to "(*prg)()" is the index of the requested byte. The return value of "(*prg)()" is the corresponding binary program code byte.

The program is stored as a "temporary" program and has no program number or name. A temporary program can be started using [ZbMocExecTemp](#) or it can be saved as a "permanent" program using one of the [ZbMocProgramSave](#) or [ZbMocProgramSaveAs](#) functions. Depending on the firmware version of the controller, it may also be necessary to call the function [ZbMocMoconSaveRam](#) to save the program persistently.

 **ZbMocProgramLoad** is designed to be used in conjunction with the **ZbCompReadcode (on-line documentation)** routine in the **ZbComp Library** ('Introduction to ZbComp' in the on-line documentation).

Portability Firmware \geq 2.7

7.74 ZbMocProgramLoadBinary

```
SIGNED16 ZbMocProgramLoadBinary(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED32 prglen,
                                UNSIGNED8 *prg);
```

Summary Download a binary (i.e. compiled) program to the selected controller.

Parameter h = Handle of the interface.
 id = Identification number (ID) of the controller.
 prglen = Length of the binary program in number of bytes.
 prg = Pointer to the binary program code.

Return Value [ZbMoc Error Codes](#)

Remarks The application source program (*.m) must be compiled into binary code by the ApossIDE compiler before a download can take place. The binary code must then be downloaded to the controller. It is not possible to download application source code directly.

This function is identical to [ZbMocProgramLoad](#) except that it takes a pointer to a buffer containing the entire binary program rather than a pointer to a retrieval function.

The program is stored as a "temporary" program and has no program number or name. A temporary program can be started using [ZbMocExecTemp](#) or it can be saved as a "permanent" program using one of the [ZbMocProgramSave](#) or [ZbMocProgramSaveAs](#) functions. Depending on the firmware version of the controller, it may also be necessary to call the function [ZbMocMoconSaveRam](#) to save the program persistently.

Portability Firmware \geq 2.7

7.75 ZbMocProgramReadBinary

```
SIGNED16 ZbMocProgramReadBinary(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 prgnr,  
                                UNSIGNED32 maxlen, UNSIGNED8 *prg,  
                                UNSIGNED32 *prglen);
```

Summary Reads (i.e. uploads) a binary program from the selected control unit.

Parameter h = Handle of the interface.
id = Identification number (ID) of the control unit.
prgnr = Number (0...94) of program to read.
maxlen = Length of binary program code buffer.
prg = Pointer to the binary program code buffer.
prglen = Length of the binary program in number of bytes (returned).

Return Value [ZbMoc Error Codes](#)

Remarks

7.76 ZbMocProgramSave

```
SIGNED16 ZbMocProgramSave(UNSIGNED16 h, UNSIGNED16 id, const STRUTF8 *name);
```

Summary Save the current "temporary" program using the next available program number.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
name = Pointer to a character array holding the program name.

Return Value ≥ 0 - Program number used to save program.
< 0 - [ZbMoc Error Codes](#)

Remarks A program must be downloaded using either the function [ZbMocProgramLoad](#) or [ZbMocProgramLoadBinary](#) before it can be saved.

The program name may be either UTF-8 encoded or ANSI encoded. However, it may not consist of more than 8 bytes (not including the null-terminator).

The program is saved as a new program using the next available program number. If the program number must be defined manually or an existing program must be overwritten, then the function [ZbMocProgramSaveAs](#) should be used instead.

Depending on the firmware version of the controller, it may also be necessary to call the function [ZbMocMoconSaveRam](#) afterwards to save the program persistently.

Portability Firmware ≥ 2.7

7.77 ZbMocProgramSaveAs

```
SIGNED16 ZbMocProgramSaveAs(UNSIGNED16 h, UNSIGNED16 id, const STRUTF8 *name,  
                             UNSIGNED16 prgnr);
```

Summary Save the current "temporary" program using the given program number.

Parameter h = Handle of the interface.
 id = Identification number (ID) of the controller.
 name = Pointer to a character array holding the program name.
 prgnr = Program number used to save the program.

Return Value [ZbMoc Error Codes](#)

Remarks A program must be downloaded using either the function [ZbMocProgramLoad](#) or [ZbMocProgramLoadBinary](#) before it can be saved.

The program name may be either UTF-8 encoded or ANSI encoded. However, it may not consist of more than 8 bytes (not including the null-terminator). Program numbers can be in the range 0 to MOC_MAXPROGCNT-1. Note that controllers with firmware versions 7.00.00 and later support more program numbers (see MOC_MAXPROGCNT in 'ZbMoc.h' for more information). If a program with the same program number already exists, then the existing program is erased and replaced with the new program.

Depending on the firmware version of the controller, it may also be necessary to call the function [ZbMocMoconSaveRam](#) afterwards to save the program persistently.

Portability Firmware ≥ 2.7

7.78 ZbMocReadMemPage

```
SIGNED16 ZbMocReadMemPage(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED32 pageaddress,
                          UNSIGNED16 *crc, UNSIGNED8 *values);
```

Summary Read a memory page out of the selected controller.

Parameter h = Handle of the interface.
 id = Identification number (ID) of the controller.
 pageaddress = Number of the memory.
 crc = Pointer to a variable taking the CRC.
 values = Pointer to an array taking the received 256 bytes of data.

Return Value [ZbMoc Error Codes](#)

Remarks Each memory page contains 256 bytes.
 This function is only available if serial (V24) communication is used.

Portability MVS Version only

7.79 ZbMocReadUserArray

```
SIGNED16 ZbMocReadUserArray(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 arrnr,
                           UNSIGNED32 mxlen, SIGNED32 *outarray,
                           UNSIGNED32 *arrsize);
```

Summary Read a user-defined array from the selected controller.

Parameter h = Handle of the interface.

id = Identification number (ID) of the controller.
 arrnr = Number of the array (0...n).
 mxlen = Number of values in the "values" array (i.e. NOT the byte length).
 outarray = Pointer to an array taking the received user data.
 arrsize = Pointer to a variable taking the actual number of values in the array on the controller.

Return Value [ZbMoc Error Codes](#)

Remarks Arrays are numbered consecutively, starting with 0.

The number of values copied from the array, is the smaller of either "maxlen" or the actual number of values in the array. However, the actual number of values in the array, is always returned in "arrsize". The application can use this to determine whether or not the entire array has been read. If the array does not exist, then "arrsize" will be set to 0. This can be used to detect the last valid array.

Portability Firmware ≥ 2.7

7.80 ZbMocReadUserVar

```
SIGNED16 ZbMocReadUserVar(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 *varnr,
                          SIGNED32 *val, UNSIGNED32 timeout);
```

Summary Read a data message from the selected controller. A message can be sent from an application program by using the [OUTMSG](#) command. The message always contains one 16 bit data value and one 32 bit data value.

Parameter h = Handle of the interface.
 id = Identification number (ID) of the controller.
 varnr = Pointer to a variable taking the 16 bit value.
 val = Pointer to a variable taking the 32 bit value.
 timeout = Maximum period of time [ms] for receiving the message.
 0 = Do not wait.
 If there is no message received within the defined time slot, the function returns an error code.

Return Value [ZbMoc Error Codes](#)

Remarks If the timeout is set to 0, then a message is expected to be available immediately. If no message is available, then **ZbMocReadUserVar** will wait up to 1 second for a message to become available. If this routine is being used simply to poll whether or not a message is available, then the timeout should be set to 1. The routine will then return after a maximum of 1 ms.

This function is not available for serial (V24) communication!

Portability Firmware ≥ 2.7

7.81 ZbMocSDOGetUsage

```
BOOLEAN ZbMocSDOGetUsage(UNSIGNED16 h, UNSIGNED16 id);
```

Summary Return a flag indicating whether or not SDO-based communication MAY be used internally for ZbMoc function processing.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.

Return Value TRUE - Internal SDO-based communication MAY be used.
FALSE - Internal SDO-based communication CANNOT be used.

Remarks When "FALSE" is returned (i.e. SDO-based communication for ZbMoc function processing is not in use), this is usually due to a controller not supporting the zub SDO object dictionary.

More detailed information (i.e. if SDO-based communication would currently be used and if the controller is capable of it) can be retrieved by the function [ZbMocSDOGetUsageDetails](#).

zub controllers starting with firmware version 6.01.00 support SDO-based communication. SDO-based communication allows many functions (reading and writing parameter values in particular) to be executed by the controller regardless of the controller's execution state. Prior to version 6.01.00, many ZbMoc functions would return a ZBMOC_E_IS_BUSY return code if the controller was already busy executing an application program.

7.82 ZbMocSDOGetUsageDetails

```
BOOLEAN ZbMocSDOGetUsageDetails(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED8 *level,  
                                BOOLEAN *sdosupport);
```

Summary Return a flag indicating whether or not SDO-based communication would currently be used (if possible) for ZbMoc function processing of the given controller.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
level = Pointer to a variable that will return the level of SDO support provided by the controller.
sdosupport = Pointer to a variable that will return a flag indicating whether or not SDO-based communication is supported by the controller.

Return Value TRUE - Internal SDO-based communication would currently be used (if possible).
FALSE - Internal SDO-based communication would not currently be used.

Remarks The return value depends on the following factors:

- SDO-based communication compatibility of the controller.
- Execution status of the controller.

Note that this is a temporary snapshot. Whether or not SDO-based communication is actually used or not for processing of the next ZbMoc function call, depends on the function call and the execution state of the controller at the time of the next function

call.

zub controllers starting with firmware version 6.01.00 support SDO-based communication. SDO-based communication allows many functions (reading and writing parameter values in particular) to be executed by the controller regardless of the controller's execution state. Prior to version 6.01.00, many ZbMoc functions would return a ZBMOC_E_IS_BUSY return code if the controller was already busy executing an application program.

7.83 ZbMocSetCallbackPDO

```
void ZbMocSetCallbackPDO(void (*cb)(UNSIGNED16 h, UNSIGNED16 id, double time,
                                     SIGNED32 pdo, SIGNED32 len, UNSIGNED8 *dat));
```

Summary Defines a pointer to a callback function which will receive PDO messages.

Parameter cb = Pointer to a callback function.

Remarks This function registers a callback function for receiving PDO messages. The ZbMoc library calls this callback function whenever a PDO message is received. See the description of [ZbMocCanOpenPollPDO](#) for an explanation of the parameters to the callback function.

Portability Firmware ≥ 5.21

7.84 ZbMocSetCallbackProgress

```
void ZbMocSetCallbackProgress(UNSIGNED32 (*cb)(UNSIGNED32 data1, UNSIGNED32 data2,
                                                UNSIGNED32 code, UNSIGNED32 value),
                              UNSIGNED32 data1, UNSIGNED32 data2);
```

Summary Defines a callback pointer to a function called to provide progress monitoring.

Parameter cb = Pointer to callback function.

data1 = Arbitrary user-defined value that will be passed to the callback function.

data2 = Arbitrary user-defined value that will be passed to the callback function.

Remarks The callback function will have the following arguments:

data1 - An arbitrary value that will be passed back to the caller.

data2 - An arbitrary value that will be passed back to the caller.

code - A code indicating the type of call. This may be any of the following values:

0 - Progress value.

1 - Wait call.

2 - Enable any "Cancel" button. The "Cancel" button should be disabled by default.

3 - Set the "high" value (i.e. the total).

4 - Show the progress dialog. The progress dialog should be hidden by default.

value - A value depending on "code" as follows:

0 - The amount of progress completed. This will go from 0 up to "high".

- 1 - The value can be ignored.
- 2 - The value can be ignored.
- 3 - The "high" value.
- 4 - The value can be ignored.

The callback function is expected to return the following return codes:

- 0 - Continue.
- 1 - Abort the process.

The return code is ignored for all codes except code 0. Note that some processes cannot be aborted.

7.85 ZbMocSetDebugFile

```
void ZbMocSetDebugFile(const STRUTF8 *filename);
```

Summary Define the log file (complete name including path), in which debugging messages are written. If this routine has not been called to set a file name, then no debugging messages are written.

Parameter filename = Path and filename of debug log file.

Remarks The given string holding the filename and path information should not exceed 259 characters in length. Names longer than this will be truncated. Strings may be specified using either ANSI or UTF-8 encoding.

Calling this routine effectively restarts the log system. Hence, care must be taken when calling this routine if the "new file" flag (i.e. 0x00000010) has been set using [ZbMocSetDebugFlags](#) since this will cause the previous file to be erased even if the same file name has been specified.

7.86 ZbMocSetDebugFlags

```
void ZbMocSetDebugFlags(UNSIGNED32 flags);
```

Summary Define the debugging flags.

Parameter flags = Debugging flags. This can be a combination of:

Bit	Meaning
----- -----	
0x00000001	Write elapsed seconds since start.
0x00000004	Indent messages.
0x00000008	Flush and close file after every message.
0x00000010	Create a new file, erasing any previous file.
0x00000020	Write timestamps.
0x00000040	Write thread number.

Remarks The default value of the debug flags is 0x00000004 (i.e. indent messages).

If the indent flag is set, then messages will generally be indented more and more, the further execution goes down the routine calling stack.

If the flush flag is set, then the log file is flushed and closed after every message is written. This will ensure that all messages are safely written to the file in the case of a crash and no messages are lost. However, it can significantly increase the execution time of calls.

If the new file flag is set, then any existing log file of the same name will be erased when the first NEW message is written.

7.87 ZbMocSetDebugHistory

```
void ZbMocSetDebugHistory(SIGNED32 max);
```

Summary Set the maximum size of the debug history.

Parameter max = Maximum number of messages to save in history. 0 for none.

Remarks If the history size is set to 0, then all debug messages are written to the log file. If the history size is greater than 0, then all non-error debug messages are saved in the history and are not written to the log file. Only the last "max" messages are saved; older messages will be lost. When an error message is to be written, then all saved messages are written before the error message is written.

7.88 ZbMocSetDebugImmediate

```
void ZbMocSetDebugImmediate(void);
```

Summary Start a ZbMoc debug log immediately using default level and flag settings.

Remarks Calling this routine will start the log system immediately. The log file will be called "ZbMocDebug.dm" and it will be placed in the standard Windows "Application Data" directory. If this routine is used, then the other 'ZbMocSetDebug...' routines should not be called.

7.89 ZbMocSetDebugLevel

```
void ZbMocSetDebugLevel(SIGNED16 level);
```

Summary Define the debugging level.

Parameter level = Debugging level:

Value | Meaning

-----|-----

0 | switch debug messages off.

+ve | the higher the value, the more detailed and the more extensive, are the messages.

-ve | only level 0 messages and messages for the specified level are generated.

Remarks The default value of the debug level is 0 (i.e. no debugging messages are generated).

If the debug level is set to a value higher than 0 (values larger than 9 generally produce no additional messages), debugging messages are written to the log file specified by calling [ZbMocSetDebugFile](#). The log file is an ASCII text format file.

ZbMocSetDebugLevel will write messages to the log file. Hence, if debug flags are to be set (i.e. by [ZbMocSetDebugFlags](#)), then they should be set PRIOR to setting the debug level.

The execution time of a program and library functions will be increased by setting the debug level to a non-0 value. This can be significant for larger debug level values.

7.90 ZbMocSetDebugMaxLen

```
void ZbMocSetDebugMaxLen(SIGNED32 maxlen);
```

Summary Set the maximum length debug log files.

Parameter maxlen = Maximum length in MB. 0 for none.

Remarks If the log file exceeds this length, then it is automatically closed and a new log file is opened. If this is set to ≤ 0 , then the log file will have no maximum length. The maximum length cannot be set larger than 2047MB. Larger numbers will be treated as 2047MB.

7.91 ZbMocSetDebugRetain

```
void ZbMocSetDebugRetain(SIGNED32 days);
```

Summary Set the number of days to retain debug log files.

Parameter days = Number of days to save files. -1 for none.

Remarks If this is set to -1, then the log file will have the name specified by the [ZbMocSetDebugFile](#) routine. If set to 0 or more, then the log file names will have the data and a sequence number appended to them. Files that match this naming format and that are older than the specified number of days, will be automatically deleted when [ZbMocSetDebugFile](#) is called. Note that a value of 0 means "Keep files from today".

7.92 ZbMocSourceDeleteFromController

```
SIGNED16 ZbMocSourceDeleteFromController(UNSIGNED16 h, UNSIGNED16 id,  
                                         UNSIGNED16 prognum);
```

Summary Delete the source code (if any) of the given program from the selected controller.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.

prognum = Program number of source to be deleted.

Return Value [ZbMoc Error Codes](#)

Remarks

Portability Firmware \geq 5.17

7.93 ZbMocSourceRestoreFromController

```
SIGNED16 ZbMocSourceRestoreFromController(UNSIGNED16 h, UNSIGNED16 id,
                                           UNSIGNED16 prognum,
                                           const STRUTF8 *filename,
                                           ZbMocSourceInfoS *sourceinfo);
```

Summary Read the source code of the given program from the selected controller.

Parameter h = Handle of the interface.

id = Identification number (ID) of the controller.

prognum = Program number of source to be read.

filename = Pointer to a string holding the filename and path of the file into which the restored source code is to be written.

sourceinfo = Pointer to a structure taking information about the source code (i.e. original file name, date, and time of the save).

Return Value [ZbMoc Error Codes](#)

Remarks If the given file already exists, then it is overwritten! The filename may be either UTF-8 or ANSI encoded.

Source code must previously be saved in the controller using the function [ZbMocSourceSaveInController](#) before it can be restored. It is not possible to restore source code from a binary program.

There is no guarantee that the source code stored in the controller corresponds to the binary program with the same program number. Binary programs are saved using [ZbMocProgramSave](#) or [ZbMocProgramSaveAs](#) and source is save using [ZbMocSourceSaveInController](#). There is no internal link in between the binary code and the source code. It is the responsibility of the application program to make sure that both versions are consistent.

The source data is stored in a compressed data format in the controller. If the decompression fails, then a ZBMOC_E_COMPRESSION_ERROR error code is returned.

Portability Firmware \geq 5.17

7.94 ZbMocSourceSaveInController

```
SIGNED16 ZbMocSourceSaveInController(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 prognum,
                                       const STRUTF8 *filename);
```

Summary Write the source code in the given file into the selected controller.

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
prognum = Program number used to save source. This should normally be the program number of the binary program corresponding to the source being saved.
filename = Pointer to a string holding the filename and path of the source code file to be stored in the controller.

Return Value [ZbMoc Error Codes](#)

Remarks It is not possible to save empty files in the controller (the ZBMOC_E_FILECONTENT error code will be returned).

The file content is saved as compressed data in the controller. If the compression fails, a ZBMOC_E_COMPRESSION_ERROR error code is returned.

There is no internal link between the binary program code and the corresponding source code. The application is responsible for managing the correspondence between binary programs and source code. It is strongly recommended that this be done by using the same by program number for both the binary program and the corresponding source code.

The filename may be either UTF-8 or ANSI encoded.

Portability Firmware ≥ 5.17

7.95 ZbMocUserParamReadRaw

```
SIGNED16 ZbMocUserParamReadRaw(UNSIGNED16 h, UNSIGNED16 id, SIGNED32 *param,  
                                UNSIGNED16 first, UNSIGNED16 last);
```

Summary Read the specified number of user parameters from the selected controller. The parameters are copied into the given array without any modifications (i.e. no conversion or decoding takes place).

Parameter h = Handle of the interface.
id = Identification number (ID) of the controller.
param = Pointer to an array taking the parameter values.
first = Number of the first parameter to read: 0..n-1
last = Number of the last parameter to read: 0..n-1

Return Value [ZbMoc Error Codes](#)

Remarks The calling application must ensure that the requested parameters exist for the connected controller. The [ZbMocVersionCounts](#) routine can be used to determine this.

The array size must be big enough to hold all requested parameters. The array size is NOT checked by this function!

Portability Firmware ≥ 2.7

7.96 ZbMocUserParamWriteRaw

```
SIGNED16 ZbMocUserParamWriteRaw(UNSIGNED16 h, UNSIGNED16 id, SIGNED32 *param,
```

```
UNSIGNED16 first, UNSIGNED16 last);
```

Summary Write the specified number of user parameters to the selected controller. The array values are copied into the user parameters of the controller without any modifications (i.e. no conversion or decoding takes place).

Parameter h = Handle of the interface.
 id = Identification number (ID) of the controller.
 param = Pointer to an array holding the parameter values to write.
 first = Number of the first parameter to write: 0..n-1
 last = Number of the last parameter to write: 0..n-1

Return Value [ZbMoc Error Codes](#)

Remarks The calling application must ensure that the requested parameters exist for the connected controller. The [ZbMocVersionCounts](#) routine can be used to determine this.
 The array size must be big enough to hold all specified parameters. The array size is NOT checked by this function!

Portability Firmware ≥ 2.7

7.97 ZbMocV24ReadRawChar

```
SIGNED16 ZbMocV24ReadRawChar(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED8 *rxd,  
                             UNSIGNED32 timeout);
```

Summary Read a single character received via the serial interface.

Parameter h = Handle of the interface.
 id = Identification number (ID) of the controller.
 rxd = Pointer to a variable taking the received character.
 timeout = Maximum period of time [ms] for receiving a character.
 0 = Do not wait (i.e. a character must present immediately).
 If there is no character received within the defined time slot, the function returns an error code.

Return Value [ZbMoc Error Codes](#)

Remarks This function can be used by applications that receive byte or character based data from the controller via the V24 interface.

This function is only available if serial (V24) communication is used.

Portability Firmware ≥ 2.7

7.98 ZbMocV24ReadRawString

```
SIGNED16 ZbMocV24ReadRawString(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED8 *rxbuf,  
                                UNSIGNED16 rxbuflen, UNSIGNED32 timeout);
```

Summary Read a string received via the serial interface.

Parameter h = Handle of the interface.
 id = Identification number (ID) of the controller.
 rxbuf = Pointer to a data section taking the received string.
 rxbuflen = Length of the data buffer.
 timeout = Maximum period of time [ms] for receiving a string.
 - 0 = Do not wait (i.e. a character must be present immediately).
 - If there is no character received within the defined time slot,
 - the function returns an error code.

Return Value [ZbMoc Error Codes](#)

Remarks This function can be used by applications that receive string data from the controller via the V24 interface. The string will be null terminated.

The string can be sent by the controller using the "print" command. The string is terminated on the serial interface line with carriage return and line feed characters.

This function is only available if serial (V24) communication is used.

Portability Firmware ≥ 2.7

7.99 ZbMocV24WriteRaw

```
SIGNED16 ZbMocV24WriteRaw(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED8 *buf,
                          UNSIGNED16 buflen);
```

Summary Write a string to the serial interface buffer for transmission.

Parameter h = Handle of the interface.
 id = Identification number (ID) of the controller.
 buf = Pointer to a data section holding the characters to transmit.
 buflen = Number of characters to transmit.

Return Value [ZbMoc Error Codes](#)

Remarks This function can be used by applications that want to transmit data, single characters, or strings to the controller (and/or application program) via the V24 interface.

Exactly "buflen" bytes are transmitted. Null characters are processed like any other characters.

This function is only available if serial (V24) communication is used.

Portability Firmware ≥ 2.7

7.100 ZbMocVersionCounts

```
void ZbMocVersionCounts(UNSIGNED32 version, UNSIGNED16 *glbcnt, UNSIGNED16 *usrcnt,
                        UNSIGNED16 *axscnt, UNSIGNED32 *arrmax);
```

Summary Return version-specific controller counts.

Parameter version = Full version of controller (from "ZbMocMoconInfoS" structure).

glbcnt = Number of global parameters. This value is returned.
 usrcnt = Number of user parameters. This value is returned.
 axscnt = Number of axis parameters. This value is returned.
 arrmax = Maximum length of DIM arrays. This value is returned.

Remarks

7.101 ZbMocWriteUserArray

```
SIGNED16 ZbMocWriteUserArray(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 *arrnr,
                             UNSIGNED32 *arrsize, UNSIGNED32 *rsize,
                             SIGNED32 *inarray);
```

Summary Write a user-defined array to the selected controller.

Parameter h = Handle of the interface.
 id = Identification number (ID) of the controller.
 arrnr = Pointer to a variable holding the number of the array (0...n).
 arrsize = Pointer to a variable holding the number of array values to write.
 rsize = Pointer to a variable taking the actual size of the array stored in the controller (if already present).
 inarray = Pointer to an array holding the values to write.

Return Value [ZbMoc Error Codes](#)

Remarks Arrays are numbered consecutively, starting with 0.

It is possible to both write new arrays or overwrite existing arrays. If a new array is written, then its number must be one more than the array number of the last array already defined in the controller (i.e. arrays number must be consecutive). If enough free memory is available in the controller, then the new array will be allocated. If an existing array is overwritten, then size of the array being written must not be larger than the size of the array already existing on the controller (i.e. arrays cannot be "grown"). The size of the array which already exists on the controller is returned in "rsize".

Portability Firmware ≥ 2.7

7.102 ZbMocWriteUserVar

```
SIGNED16 ZbMocWriteUserVar(UNSIGNED16 h, UNSIGNED16 id, UNSIGNED16 varnr,
                             SIGNED32 val);
```

Summary Transmit a data message to the selected controller. This message can be received by an application program using the [INMSG](#) command. The message always contains one 16-bit data value and one 32 bit data value.

Parameter h = Handle of the interface.
 id = Identification number (ID) of the controller.
 varnr = 16 bit value to transmit.
 val = 32 bit value to transmit.

Return Value [ZbMoc Error Codes](#)

Remarks This function can be used by applications that receive byte or character based data from the controller via the V24 interface.

This function is only available if serial (V24) communication is used.

Portability Firmware \geq 2.7

8 Index

Error Messages, 28-34
Function Groups, 8-13
Imprint, 1
Introduction to ZbMoc, 6-7
ZbMoc Cover Page, -1-0
ZbMoc Error Codes, 19-27
ZbMocAutoClear, 35
ZbMocAutoSet, 35
ZbMocAxisParamRead(DEPRECATED), 35-36
ZbMocAxisParamS, 16-18
ZbMocAxisParamWrite(DEPRECATED), 36
ZbMocBlobInfoRequest, 36-37
ZbMocBlobRead, 37
ZbMocBlobWrite, 37-38
ZbMocBreak, 38-39
ZbMocBreakAll, 39
ZbMocCanOpenDownloadSDO, 39
ZbMocCanOpenInhibitPDO, 39-40
ZbMocCanOpenPollPDO, 40
ZbMocCanOpenQueuePDO, 41
ZbMocCanOpenReadSDO, 41-42
ZbMocCanOpenReadSegmentedSDO, 41
ZbMocCanOpenStart, 42
ZbMocCanOpenUploadSDO, 42
ZbMocCanOpenWritePDO, 42-43
ZbMocCanOpenWriteSDO, 43-44
ZbMocCanOpenWriteSegmentedSDO, 43
ZbMocCanReadRaw, 44
ZbMocCanWriteRaw, 44
ZbMocClearError, 45
ZbMocClose, 45
ZbMocCloseAll, 45
ZbMocCNFRestoreFromFile, 45-46
ZbMocCNFSaveToFile, 46
ZbMocConnect, 46

ZbMocContinue, 46-47
ZbMocDebugPrint, 47
ZbMocDebugPrintw, 47
ZbMocDisconnect, 47
ZbMocExec, 47-48
ZbMocExecTemp, 48
ZbMocGetAxisPosition, 48-49
ZbMocGetError, 49
ZbMocGetIFDriverID, 49
ZbMocGetNumber, 49-50
ZbMocGetZbMocVersion, 50
ZbMocMemoryDump, 50
ZbMocMemoryLock, 50-51
ZbMocMoconBusyWait, 51
ZbMocMoconClearFactory, 51
ZbMocMoconClearParameters, 51-52
ZbMocMoconClearPrograms, 52
ZbMocMoconDeleteEeprom, 52
ZbMocMoconExecutionStatus, 52-53
ZbMocMoconFind, 53-54
ZbMocMoconIdent, 54
ZbMocMoconIdentS, 14
ZbMocMoconInfo, 54
ZbMocMoconInfoldx, 54-55
ZbMocMoconInfoS, 14-15
ZbMocMoconNameSet, 55
ZbMocMoconSaveRam, 55
ZbMocMoconSaveRamSection, 55-56
ZbMocMoconSetCanParameters, 56-57
ZbMocMoconState, 57
ZbMocMoconWaitForString, 57
ZbMocOpenCanLpt, 57-59
ZbMocOpenCanUsb, 59-60
ZbMocOpenEtherCat, 60-61
ZbMocOpenSerial, 61-62
ZbMocOpenSim, 66-67

ZbMocOpenTcp, 62-63
ZbMocOpenUsb, 63-64
ZbMocOpenV24, 64
ZbMocOpenVlt, 64-66
ZbMocParamRead, 67
ZbMocParamS, 15-16
ZbMocParamWrite, 67
ZbMocPollMessage, 67-68
ZbMocProbe, 68
ZbMocProgramDelete, 68
ZbMocProgramList, 68-69
ZbMocProgramLoad, 69-70
ZbMocProgramLoadBinary, 70
ZbMocProgramReadBinary, 70-71
ZbMocProgramSave, 71
ZbMocProgramSaveAs, 71-72
ZbMocReadMemPage, 72
ZbMocReadUserArray, 72-73
ZbMocReadUserVar, 73
ZbMocSDOGetUsage, 73-74
ZbMocSDOGetUsageDetails, 74-75
ZbMocSetCallbackPDO, 75
ZbMocSetCallbackProgress, 75-76
ZbMocSetDebugFile, 76
ZbMocSetDebugFlags, 76-77
ZbMocSetDebugHistory, 77
ZbMocSetDebugImmediate, 77
ZbMocSetDebugLevel, 77-78
ZbMocSetDebugMaxLen, 78
ZbMocSetDebugRetain, 78
ZbMocSourceDeleteFromController, 78-79
ZbMocSourceRestoreFromController, 79
ZbMocSourceSaveInController, 79-80
ZbMocUserParamReadRaw, 80
ZbMocUserParamWriteRaw, 80-81
ZbMocV24ReadRawChar, 81
ZbMocV24ReadRawString, 81-82

ZbMocV24WriteRaw, 82

ZbMocVersionCounts, 82-83

ZbMocWriteUserArray, 83

ZbMocWriteUserVar, 83-84