MCBCOM User Manual

Copyright © 2018 NXP Semiconductors. All Rights Reserved.

Table of contents

PC Master Communication Library

Communication Library

This document describes the services of the communication library used by the master software. The document also contains the complete specification of the communication protocol used to exchange data between a master application and the application running on the embedded target.

The Communication Library provides the code needed for a PC application to connect and control embedded applications using the communication protocol (which is also described herein). Not only is the library an important part of the FreeMASTER application but it may also be used by user's own PC applications to minimize the effort needed to write the communication layer.

The compiled library is called *mcbcom.dll* on Windows and *mcbcom.so* on Linux systems.

The header file *mcbcom.h* contains all the definitions needed to use the communication library and/or implement direct protocol layer access functions.

This manual is divided into two main chapters:

- Protocol Specification
- Communication Library Reference Manual

See also: protocol specification updates in versions 2, 3 and 3.1.

What's new ?

The Protocol v2

The 2nd version of the communication protocol accommodates the needs of microcontrollers with small RAM capabilities. To minimize the receive/transmit buffer size, the MCB_CMD_GETINFOBRIEF command (alternate to MCB_CMD_GETINFO) and several other commands have been added.

The following table includes a complete list of commands added in Protocol Version 2:

MCB_CMD_GETINFOBRIEF	0xc8	Retrieves a subset of board information structure (fast)
MCB_CMD_WRITEVAR8	0xe3	Writes BYTE variable (fast)
MCB_CMD_WRITEVAR16	0xe4	Writes WORD variable (fast)
MCB_CMD_WRITEVAR32	0xf0	Writes DWORD variable (fast)
MCB_CMD_WRITEVAR8MASK	0xe5	Writes specified bits in BYTE variable (fast)
MCB_CMD_WRITEVAR16MASK	0xf1	Writes specified bits in WORD variable (fast)

Two new configuration bits are also defined in this version (MCB_CFGFLAG_NOFASTREAD and MCB_CFGFLAG_NOFASTWRITE). See MCB_CMD_GETINFO or MCB_CMD_GETINFOBRIEF for closer description.

The Protocol v3

The 3rd version of the communication protocol adds the support for devices with 32 bit-wide address bus.

The following table includes a complete list of commands added in Protocol Version 3:

MCB_CMD_READVAR8EX	0xe0	Read BYTE variable
MCB_CMD_READVAR16EX	0xe1	Read WORD variable
MCB_CMD_READVAR32EX	0xe2	Read DWORD variable
MCB_CMD_READMEMEX	0x04	Read a block of memory
MCB_CMD_WRITEMEMEX	0x05	Write a block of memory
MCB_CMD_WRITEMEMMASKEX	0x06	Write a block of memory with bit mask
MCB_CMD_SETUPSCOPEEX	0x0a	Setup the oscilloscope
MCB_CMD_SETUPRECEX	0x0b	Setup the recorder
MCB_CMD_GETRECBUFFEX	0xc9	Get the recorder data

One new configuration bit is also defined in this version (MCB_CFGFLAG_EXACCESSONLY). See MCB_CMD_GETINFO or MCB_CMD_GETINFOBRIEF for closer description.

The Protocol v3.1

This version extends the protocol by adding commands to support Target-side Addressing (TSA) feature. The TSA feature is described in more details in the User Guide of FreeMASTER Serial Communication Driver v1.8.1 and later.

The following table includes a complete list of commands added in Protocol Version 3.1

MCB_CMD_GETTSAINFO	0x11	Get TSA table info
MCB_CMD_GETTSAINFOEX	0x12	Get TSA table info for 32-bit addressing
MCB_CMD_GETSTRLEN	0xD4	Get length of zero-terminated string
MCB_CMD_GETSTRLENEX	0xE6	Get length of zero-terminated string for 32-bit addressing

---

See also: command codes table, status codes table

Protocol Specification

Communication Protocol Reference

The RS-232 protocol is a set of simple binary structures and conventions enabling a data/code exchange between a personal computer (PC) and a target controller board. It uses raw 8 bits, no parity, serial transfer at a standard speed (9600 kbps by default).

The communication model is based on a master-slave relationship, where the PC sends a message with a command and its arguments, and the target responds immediately (within a specified time) with the operation status code and return data. The target never initiates communication; its responses are specified and always of a fixed (known) length (This is true on a logical level. On a link level, there is a replication of special start-of-message bytes, details of which follow).

Command Message

A command message is always sent from the PC to the target.

In the fast command message format, command numbers must be ≥ 0xc0; see the Command Codes Table for details.

start-of-message (1 BYTE)	command (1 BYTE)	data part (known length)	checksum (1 BYTE)

In the standard command message format, command numbers must be < 0xc0 and are shown in the Command Codes Table.

start-of-message (1 BYTE)	command (1 BYTE)	data length (1BYTE)	data part (variable length)	checksum (1 BYTE)

start of message (SOM)

The special character defined as ASCII '+' code (0x2b); in mcbcom.h, this is defined as start-of-block: MCB_SOB

command

A one byte command code (see command codes table)

data length

The length of data part

data part

Variable length data

checksum

Two's complement checksum; computed by taking the two's complement of the sum of all bytes of a message after the SOM.

The following code example calculates checksum for a message in standard format before it is transmitted:

```
typedef unsigned char BYTE;

struct {
 BYTE cmd;
 BYTE len;
 BYTE data[N];
 BYTE _space_for_checksum;
} message;

// prepare message
// ...

// calculate checksum
BYTE *p = &message;
BYTE sum = *p++;  // cmd field
for(int i=0; i<=message.len; i++)
 sum += *p++; // add len and data
// store checksum after last valid data byte
*p = 0x100 - sum;
// transmit SOM byte
// ....

// transmit message and checksum (replicating each occurence of SOM byte)
// ....
```

The start-of-message (SOM) character receives special treatment from the link protocol layer. When received, it should reset the receiver's state machine and initialize it for reception of new message. Since the data being transferred across an RS-232 line is in binary format, a byte with value equal to SOM may be contained in the message body, which could cause an undesirable reinitialization of the receiver. This is why each occurrence of SOM byte in length, data or checksum part of a message is signaled by duplicating this byte. On the other side, the receiver resets its state machine only when the SOM byte it receives is followed by a non-SOM byte. If the receiver receives two consecutive SOM bytes, it merges them to a single one.

The command is any enumerated value from the header file. If the special (fast) commands carry either no fixed length data or short fixed-length data, the data part length is encoded directly into the value.

Fast command code:

1	1	L	L	C	C	C	C

1-bits
The fast command values are identified by two most significant bits set to one (command values are >= 0xc0)

L-bits
Indicates the length of data part in 2-bytes increment (0, 2, 4, and 6 bytes can be specified)

C-bits
Specifies the fast command code

Response Message

A response message is always sent from the target to the PC.

The format of response messages is shown in the following table:

start-of-message (1 BYTE)	status code (1 BYTE)	data part (known length)	checksum (1 BYTE)

start of message
The special character MCB_SOB defined as ASCII '+' code

status code
The one byte operation status code (see status codes table)

data part
Variable length data; the length depends on the status code value. An error response message (status MSB set) carries no data; a success response message carries known data to the PC (the data length is predetermined)

checksum
Two's complement checksum; computed by taking the two's complement of the sum of all bytes of a message after the SOM

See also: command codes table, status codes table

Command Codes Table

The following table lists the protocol commands used to communicate with the master software. The "EX" command variants are defined in the protocol version 3 to support devices with 32-bit address space.

Fast Commands

MCB_CMD_GETINFO	0xc0	Retrieve board information structure
MCB_CMD_GETINFOBRIEF	0xc8	Retrieve a subset of board information structure
MCB_CMD_STARTREC	0xc1	Start data recorder
MCB_CMD_STOPREC	0xc2	Stop data recorder
MCB_CMD_GETRECSTS	0xc3	Get the recorder status
MCB_CMD_GETRECBUFF(EX)	0xc4 (0xc9)	Get the recorder data
MCB_CMD_READSCOPE	0xc5	Get the scope data
MCB_CMD_GETAPPCMDSTS	0xc6	Get the application command status
MCB_CMD_READVAR8(EX)	0xd0 (0xe0)	Read BYTE variable
MCB_CMD_READVAR16(EX)	0xd1 (0xe1)	Read WORD variable
MCB_CMD_READVAR32(EX)	0xd2 (0xe2)	Read DWORD variable

MCB_CMD_WRITEVAR8	0xe3	Write BYTE variable
MCB_CMD_WRITEVAR16	0xe4	Write WORD variable
MCB_CMD_WRITEVAR32	0xf0	Write DWORD variable
MCB_CMD_WRITEVAR8MASK	0xe5	Write specified bits in BYTE variable
MCB_CMD_WRITEVAR16MASK	0xf1	Write specified bits in WORD variable
MCB_CMD_GETSTRLEN(EX)	0xd4 (0xe6)	Get string length (used for TSA table read)

Standard Commands

MCB_CMD_READMEM(EX)	0x01 (0x04)	Read a block of memory
MCB_CMD_WRITEMEM(EX)	0x02 (0x05)	Write a block of memory
MCB_CMD_WRITEMEMMASK(EX)	0x03 (0x06)	Write a block of memory with bit mask
MCB_CMD_SETUPSCOPE(EX)	0x08 (0x0a)	Setup the oscilloscope
MCB_CMD_SETUPREC(EX)	0x09 (0x0b)	Setup the recorder
MCB_CMD_SENDAPPCMD	0x10	Send the application command
MCB_CMD_GETTSAINFO(EX)	0x11 (0x12)	Get address and size of selected TSA table

---

See also: status codes table

Status Codes Table

Success Codes

Success codes are designated by the Most Significant Bit (MSB) set to zero.

MCB_STS_OK	0x00	The operation finished successfully, data returned (if any) are valid
MCB_STS_RECRUN	0x01	Data recorder is running (see data recorder description)
MCB_STS_RECDONE	0x02	Data recorder is stopped (see data recorder description)

Error Codes

Error codes are designated by the MSB set to one.

MCB_STC_INVCMD	0x81	Unknown command code (unsupported operation).
MCB_STC_CMDCSERR	0x82	Command checksum error
MCB_STC_CMDTOOLONG	0x83	Command is too long, the receive buffer is too small to accept it
MCB_STC_RSPBUFFOVF	0x84	The response would not fit into the transmit buffer
MCB_STC_INVBUFF	0x85	Invalid buffer length or operation
MCB_STC_INVSIZE	0x86	Invalid size specified
MCB_STC_SERVBUSY	0x87	Service is busy
MCB_STC_NOTINIT	0x88	Service is not initialized

MCB_STC_EACCESS	0x89	Access to target resource is denied

See also: command codes table

Command Codes

MCB_CMD_GETAPPCMDSTS (0xc6)

This command directs the target to retrieve the application command status. Application commands are described under the MCB_CMD_SENDAPPCMD topic.

Command Data:

This command carries no data

Possible Responses:

MCB_STS_OK (0x00)

A successful operation; the data returned contains a single byte with application command status information.

```
#define MCB_APPCMDRESULT_NOCMD      0xff
#define MCB_APPCMDRESULT_RUNNING    0xfe
typedef struct {
  uint8_t result;     /**< @brief command return value */
} MCB_RESP_GETAPPCMDSTS_t;
```

`result`
The application command status

MCB_STC_INVCMD (0x81)

Application commands not implemented.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command's checksum value was invalid.

See also: command codes table, status codes table

MCB_CMD_GETINFO (0xc0)

This command directs the target to determine the internal board configuration.

Command Data:

This command carries no data.

Possible Responses:

MCB_STS_OK (0x00)

A successful operation; the data returned contains the structure with board configuration information.

```
#define MCB_DESCR_SIZE 25
#define MCB_CFGFLAG_BIGENDIAN     0x01  // board uses bigEndian byte order
#define MCB_CFGFLAG_NOFASTREAD    0x02  // do not try the fast read commands v2+
#define MCB_CFGFLAG_NOFASTWRITE   0x04  // do not try the fast write commands v2+
#define MCB_CFGFLAG_EXACCESSONLY  0x08  // do not try 16-bit address access v3+

typedef struct {
  uint8_t protVer;          // protocol version
  uint8_t cfgFlags;         // MCB_CFGFLAG_ bits
  uint8_t dataBusWdt;       // data bus width (bytes)
  uint8_t globVerMajor;     // board firmware version major number
  uint8_t globVerMinor;     // board firmware version minor number
  uint8_t cmdBuffSize;      // receive/transmit buffer size (without SOB, CMD/STS and CS)
  uint16_t recBuffSize;     // recorder buffer memory
  uint16_t recTimeBase;     // recorder time base (2 MSB exponent 1 = m, 2 = u, 3 = n)
  uint8_t descr[MCB_DESCR_SIZE];  // ANSI string, board description
} MCB_RESP_GETINFO_t;
```

`protVer`

Protocol version number; the valid values are 1, 2 or 3.

`cfgFlags`

Board configuration flags, which can be any combination of flag bits described in following table:

MCB_CFGFLAG_BIGENDIAN	0x01	the board uses the big-endian number format
MCB_CFGFLAG_NOFASTREAD	0x02	disable trying to use fast read commands (**V2+**)
MCB_CFGFLAG_NOFASTWRITE	0x04	disable trying to use fast write and masked-write commands (**V2+**)
MCB_CFGFLAG_EXACCESSONLY	0x08	disable trying to use commands which address the 16-bit address space (**V3+**)

`dataBusWdt`

The width of data word accessible on single address

`globVerMajor, globVerMinor`

Board firmware version

`cmdBuffSize`

The size of the transmit/receive buffer (buffer for receiving commands and transmitting the responses)

`recBuffSize`

The size of the buffer used for the internal recorder

`recTimeBase`

The time base for the internal recorder (the recorder interrupt period). The lower 14 bits of the word are the mantissa and two MSB bits specifically encoded exponent (1 = milli, 2 = micro, 3 = nano).

Example values:
0x4001 ... 1 ms
0x400A ... 10 ms
0x8014 ... 20 us
0xc1F4 ... 500 ns

`descr`

A zero-terminated string with board description

MCB_STC_INVCMD (0x81)

The command was not implemented. Beginning with Protocol V2, the board need not implement this command and the command MCB_CMD_GETINFOBRIEF must be implemented instead.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command's checksum value was invalid.

See also: MCB_CMD_GETINFOBRIEF, command codes table, status codes table

MCB_CMD_GETINFOBRIEF (0xc8) - V2+

This command directs the target to determine the internal board configuration and can be implemented by the board application instead of MCB_CMD_GETINFO to save memory resources.

Command Data:

This command carries no data.

Possible Responses:

MCB_STS_OK (0x00)

A successful operation. The data returned contains the structure with the subset of board configuration information. This structure is in the same format as the full information structure received from MCB_CMD_GETINFO.

```
#define MCB_CFGFLAG_BIGENDIAN     0x01  // board uses bigEndian byte order
#define MCB_CFGFLAG_NOFASTREAD    0x02  // do not try the fast read commands v2+
#define MCB_CFGFLAG_NOFASTWRITE   0x04  // do not try the fast write commands v2+
#define MCB_CFGFLAG_EXACCESSONLY  0x08  // do not try 16-bit address access v3+

typedef struct {
  uint8_t protVer;           // protocol version
  uint8_t cfgFlags;          // MCB_CFGFLAG_ bits
  uint8_t dataBusWdt;        // data bus width (bytes)
  uint8_t globVerMajor;      // board firmware version major number
  uint8_t globVerMinor;      // board firmware version minor number
  uint8_t cmdBuffSize;       // receive/transmit buffer size (without SOB, MD/STS and CS)
} MCB_RESP_GETINFOBRIEF_t;
```

protVer

Protocol version number; the valid value is 2, since this command was first specified in the protocol V2.

cfgFlags

Board configuration flags, this can be any combination of flag bits described in following table:

MCB_CFGFLAG_BIGENDIAN	0x01	*the board uses the big-endian number format*
MCB_CFGFLAG_NOFASTREAD	0x02	*disable trying to use fast read commands (**V2+**)*
MCB_CFGFLAG_NOFASTWRITE	0x04	*disable trying to use fast write and masked-write commands (**V2+**)*
MCB_CFGFLAG_EXACCESSONLY	0x08	*disable trying to use commands which address the 16-bit address space (**V3+**)*

`dataBusWdt`
The width of data word accessible on single address

`globVerMajor, globVerMinor`
Board firmware version

`cmdBuffSize`
The size of the transmit/receive buffer (buffer for receiving commands and transmitting the responses)

MCB_STC_INVCMD (0x81)

The command was not implemented. In this case, the standard MCB_CMD_GETINFO command must be implemented.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command checksum value was invalid.

See also: MCB_CMD_GETINFO, command codes table, status codes table

MCB_CMD_GETRECBUFF (0xc4), MCB_CMD_GETRECBUFFEX (0xc9)

This command gets the recorder data buffer description and can be used when the recorder is stopped to get information on how to read the recorded data values.

The "EX" command variant is defined in the protocol version 3 to support devices with 32-bit address space.

Command Data:

This command carries no data.

Possible Responses:

MCB_STS_OK (0x00)

A successful operation; the data returned contains the structure of recorder buffer information

```
// Response to MCB_CMD_GETRECBUFF command
typedef {
  uint16_t buffAddr;    // cyclic buffer address
  uint16_t startIx;     // first sample index
} MCB_RESP_GETRECBUFF_t;

// Response to MCB_CMD_GETRECBUFFEX command
typedef struct {
  uint32_t buffAddr;    /**< @brief cyclic buffer address */
  uint16_t startIx;     /**< @brief first sample index */
} MCB_RESP_GETRECBUFFEX_t;
```

`buffAddr;`
The physical address of recorder buffer. The buffer is considered to be circular; the length is set by the MCB_CMD_SETUPREC command.

`startIx;`

The index of first variable set in the circular buffer. See MCB_CMD_SETUPREC for a description of the recorder data sets.

MCB_STC_SERVBUSY (0x87)

The recorder is running; it must be in a stopped state before this operation begins. See MCB_STOP_REC and MCB_SETUPREC for the information about stopping the recorder automatically and manually.

MCB_STS_NOTINIT (0x88)

The recorder was not initialized with MCB_CMD_SETUPREC command.

MCB_STC_INVCMD (0x81)

The recorder service is not implemented.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command checksum value was invalid.

See also: command codes table, status codes table

MCB_CMD_GETRECSTS (0xc3)

This command directs the target to retrieve the recorder status.

Command Data:

This command carries no data.

Possible Responses:

MCB_STS_RECRUN (0x01)

The recorder is running; the response carries no data.

MCB_STS_RECDONE (0x02)

The recorder is stopped; the response carries no data.

MCB_STC_NOTINIT (0x88)

The recorder was not initialized with MCB_CMD_SETUPREC command.

MCB_STC_INVCMD (0x81)

The recorder service is not implemented.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command's checksum value was invalid.

See also: command codes table, status codes table

MCB_CMD_GETTSAINFO (0x11), MCB_CMD_GETTSAINFOEX (0x12)

This command probes the target if a requested TSA table exists. If yes, a response to this command carries an information about TSA table address and size. Otherwise, zero address and zero table size is returned. Protocol master is expected to query all TSA tables sequentially starting with index 0 and repeating the process until the invalid (zeroed) table information is returned by the protocol slave. Master is expected to try both command variants (MCB_CMD_GETTSAINFO and MCB_CMD_GETTSAINFOEX) to find out which TSA table format the target device support. The TSA table entries are organized either as 4x16bits or 4x32bit structures as described below. The two formats are never mixed in one target implementation, so it is only necessary to query both formats once before retrieving all other TSA-related data.

The process of reading out all TSA tables can be summarized as follows:

1. Issue the MCB_CMD_GETTSAINFO command for table index 0.
2. When invalid command error (MCB_STC_INVCMD) is returned, probe the table at index 0 with the MCB_CMD_GETTSAINFOEX command. When MCB_STC_INVCMD is returned again, the target does not support the TSA feature.
3. If a positive status is returned (MCB_STS_OK) but the response data contains zero address and size, you have reached the end of TSA table list. Exit the TSA read-out process.
4. If a valid TSA table information was obtained with non-zero address and size of the table, use the MCB_CMD_READMEM command to read the whole table content.
5. Determine the format of the table entry by testing the **tsaFlags** member of the information record for the MCB_TSAINFOFLAG_32BIT value.
6. Calculate the number of table entries by dividing the table size by 8 or 16 depending on the table entry format. Use 16 when MCB_TSAINFOFLAG_32BIT flag was set.
7. Refer to the TSA format below to find out how to parse the table items. There are two string pointers in each table entry called **name** and **type**.
8. For each table entry, use the MCB_CMD_GETSTRLEN command to obtain length of the **name** and **type** strings.
9. Use the MCB_CMD_READMEM command to read both strings.
10. Repeat steps 8 and 9 for each table entry.
11. Use the MCB_CMD_GETTSAINFO or MCB_CMD_GETTSAINFOEX command to retrieve information about the next TSA table and repeat the process from step 3.

More details about TSA use cases can be found in user documentation.

Command Data:

Two bytes with the index of the target TSA table for which the information should be returned.

Possible Responses:

MCB_STS_OK (0x00)

A successful operation; response describes the requested TSA table:

```
#define MCB_TSAINFOFLAG_VERSION_MASK  0x000f
#define MCB_TSAINFOFLAG_32BIT  0x0100 // address format (16/32 bit)
#define MCB_TSAINFOFLAG_HV2BA  0x0200 // HawkV2 byte-addressing fix
```

```
// Response to MCB_CMD_GETTSAINFO command
typedef struct {
  uint16_t tsaFlags;
  uint16_t tblSize;
  uint16_t tblAddr;
} MCB_RESP_GETTSAINFO_t;

// Response to MCB_CMD_GETTSAINFOEX command
typedef struct {
  uint16_t tsaFlags;
  uint16_t tblSize;
  uint32_t tblAddr;
} MCB_RESP_GETTSAINFOEX_t;
```

`tsaFlags`

Flags describing the TSA table - see MCB_TSAINFOFLAG_xxx values above. The 32BIT flag is set for tables which consist of 4x32bit entries.
The HV2BA is set on DSC56F800E/EX platforms to indicate that the table address is specified in 16bit-addressing mode (and must be doubled when reading the TSA table content).
The low nibble of the flags value indicates the TSA format version. Version 2 is the one supported by FreeMASTER tool and is also documented herein.

`tblSize, tblAddr`

Size and address of the table in bytes. Use this information to read the TSA table content with MCB_CMD_READMEM command.

MCB_STC_INVCMD (0x81)

The TSA feature is not implemented or is only implemented in the other addressing mode. Protocol master is always expected to try both MCB_CMD_GETTSAINFO and MCB_CMD_GETTSAINFOEX command variants to find out which TSA table entry size is supported by protocol slave.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command checksum value is invalid.

TSA Table Entry Format

Each TSA table consists of table entries, and each entry consists of four parameters. Depending on the microcontroller architecture the table parameters can be either 16-bit or 32-bit. Each parameter size can be determined by testing the **tsaFlags** member of the information structure for the MCB_TSAINFOFLAG_32BIT bit.

Each table entry describes one memory or type element. Such an element can be a variable object, structure type or structure member. In FreeMASTER 2.0 and later, the TSA entry can also describe special information like a web-link or a local memory-mapped file.

```
// TSA flags carried in TSA_ENTRY.info
#define FMSTR_TSA_INFO_ENTRYTYPE_MASK 0x0003U
#define FMSTR_TSA_INFO_STRUCT         0x0000U
#define FMSTR_TSA_INFO_RO_VAR         0x0001U
#define FMSTR_TSA_INFO_MEMBER         0x0002U
#define FMSTR_TSA_INFO_RW_VAR         0x0003U

// TSA Table entry structure (16 bit)
typedef struct MCB_TSA_ENTRY16_t
{
  uint16_t  name;
  uint16_t  type;
```

```
  uint16_t  addr;
  uint16_t  info;
} MCB_TSA_ENTRY16_t;

// TSA Table entry structure (32 bit)
typedef struct MCB_TSA_ENTRY32_t
{
  uint32_t  name;
  uint32_t  type;
  uint32_t  addr;
  uint32_t  info;
} MCB_TSA_ENTRY32_t;

#define MCB_TSA_UINT8   ((char)0xE0U)
#define MCB_TSA_UINT16  ((char)0xE1U)
#define MCB_TSA_UINT32  ((char)0xE2U)
#define MCB_TSA_UINT64  ((char)0xE3U)
#define MCB_TSA_SINT8   ((char)0xF0U)
#define MCB_TSA_SINT16  ((char)0xF1U)
#define MCB_TSA_SINT32  ((char)0xF2U)
#define MCB_TSA_SINT64  ((char)0xF3U)
#define MCB_TSA_UFRAC16 ((char)0xE5U)
#define MCB_TSA_UFRAC32 ((char)0xE6U)
#define MCB_TSA_FRAC16  ((char)0xF5U)
#define MCB_TSA_FRAC32  ((char)0xF6U)
#define MCB_TSA_FLOAT   ((char)0xFAU)
#define MCB_TSA_DOUBLE  ((char)0xFBU)
#define MCB_TSA_INVALID ((char)0xFFU)
#define MCB_TSA_SPECIAL ((char)0xECU) // special information (not a symbol)
```

`name`

*The address of the element **name** zero-terminated string. You need to use the MCB_CMD_GETSTRLEN and MCB_CMD_READMEM commands to get the name string value.*

`type`

*The address of the element **type** zero-terminated string. You need to use the MCB_CMD_GETSTRLEN and MCB_CMD_READMEM commands to retrieve the type string value.*
The type string is either one of the pre-defined constants describing base types (e.g. `MCB_TSA_UINT8`) or a name of user-defined type.

`addr`

For entries describing variables or other memory-mapped objects, this parameter contains the object address. For entries describing structure members, this parameter contains the offset of the structure member within the parent structure type.

`info`

Contains two access-rights bits and a size of the memory object (shifted by two bits left). Refer to FMSTR_TSA_INFO_xxx bit mask values described above.

See also: command codes table, status codes table

MCB_CMD_GETSTRLEN (0xD4), MCB_CMD_GETSTRLENEX (0xE6)

This command determines the length of a zero-terminated string at a specified address. Typically it is used when reading strings referred to by the TSA table entries. See MCB_CMD_GETTSAINFO(EX) command description for more details.

Command Data:

The address of the zero-terminated string. The length of the data part is 2 bytes for the "no-EX" commands (16 bit address) and 4 bytes for "EX" commands (32 bit address).

Possible Responses:

MCB_STS_OK (0x00)

A successful operation; the response is the string length (2-byte-long unsigned integer).

MCB_STC_INVCMD (0x81)

The TSA feature is not implemented.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command checksum value is invalid.

See also: command codes table, status codes table

MCB_CMD_READMEM (0x01), MCB_CMD_READMEMEX (0x04)

This command directs the target to read the block of memory. The "EX" command variant is defined in the protocol version 3 to support devices with 32-bit address space.

Command Data:

This command must contain the structure specifying the address and size of the target memory block.

The caller must be aware that the size of the target transmit buffer is limited and cannot accommodate large data blocks. The target data bus width must also be considered when splitting the memory block request into multiple MCB_CMD_READMEM commands. When the "size" bytes are read from the address A0, the next read should access the address A1 = A0 + size/dataBusWdt. The transmit buffer size and data bus width can be determined using the MCB_CMD_GETINFO command.

```
// Data passed to MCB_CMD_GETAPPCMDDATA command
typedef struct {
  uint8_t size;          // required size
  uint16_t addr;         // memory block address
} MCB_DATA_READMEM_t;

// Data passed to MCB_CMD_READMEMEX command
typedef struct {
  uint8_t size;          // required size
  uint32_t addr;         // memory block address
} MCB_DATA_READMEMEX_t;
```

`size`

The size of the required memory block

`addr`

The address of the required memory block

Possible Responses:

MCB_STS_OK (0x00)

A successful operation; the data returned contains the block of the requested memory

MCB_STC_INVCMD (0x81)

The requested command is not supported. The device does not need to support neither of these commands and the client should be prepared to use one of the MCB_CMD_READVARx commands to read the contents of the requested memory location.

MCB_STS_RSPBUFFOVF (0x84)

A buffer overflow has occurred in the response transmit buffer because the required memory block is too long.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command checksum value was invalid.

See also: MCB_CMD_READVARx(EX), MCB_CMD_WRITEMEM, command codes table, status codes table

MCB_CMD_READSCOPE (0xc5)

This command tells the target to read the oscilloscope variables.

Command Data:

This command carries no data.

Possible Responses:

MCB_STS_OK (0x00)

A successful operation; the data returned contains the values of all valid oscilloscope variables. The total data length is a sum of "varDef[i].size" members of MCB_CMD_SETUPSCOPE command used for the oscilloscope initialization.

MCB_STS_NOTINIT (0x88)

The oscilloscope was not initialized with MCB_CMD_SETUPSCOPE command.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command checksum value is invalid.

See also: command codes table, status codes table

MCB_CMD_READVAR8 (0xd0), MCB_CMD_READVAR8EX (0xe0)

This command directs the target to read the 1-byte-long variable.

MCB_CMD_READVAR16 (0xd1), MCB_CMD_READVAR16EX (0xe1)

This command directs the target to read the 2-byte-long variable.

MCB_CMD_READVAR32 (0xd2), MCB_CMD_READVAR32EX (0xe2)

This command directs the target to read the 4-byte-long variable.

Command Data:

The address of the memory location to read. The length of the data part is 2 bytes for the "no-EX" commands (16-bit address) and 4 bytes for "EX" commands (32-bit address).

Possible Responses:

MCB_STS_OK (0x00)

A successful operation; the data returned contains the variable value. Based on which command it is, the data part length is 1, 2 or 4 bytes.

MCB_STC_INVCMD (0x81)

The requested command is not supported. The device may support neither of these commands and the client should be prepared to use the MCB_CMD_READMEM or MCB_CMD_READMEMEX commands to read the contents of the requested memory location.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command checksum value is invalid.

See also: MCB_CMD_READMEM(EX), MCB_CMD_WRITEVARx, command codes table, status codes table

MCB_CMD_SENDAPPCMD (0x10)

This command directs the target to send the application command, which serves as the transport protocol to the target application. Command data is delivered to the target application and the application is notified.

Command Data:

This command must include specific data known to the board application.

Possible Responses:

MCB_STS_OK (0x00)

A successful operation; the application command was sent to the board application and the status of the call can be determined using MCB_CMD_GETAPPCMDSTS command.

MCB_STC_INVCMD (0x81)

Application commands not implemented.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command checksum value is invalid.

See also: command codes table, status codes table

MCB_CMD_SETUPREC (0x09), MCB_CMD_SETUPRECEX (0x0b)

This command directs the target to set up the data recorder and start it. When the data access speed requirements exceed the serial line's capability (even the oscilloscope feature), the variable values must be recorded into an internal controller memory. The MCB_CMD_SETUPREC command changes the internal recorder settings.

When running, the recorder periodically copies the selected memory locations into the internal circular buffer. In a single action, it transfers one set of defined memory locations (variables). When it reaches the end of the buffer, the recorder wraps the pointer back to its beginning. When the trigger occurs (see below), the *postTrigger* variable sets are recorded and the recorder is stopped. If the value of *postTrigger* is less than the total buffer length (counting the variable sets), sets that existed *before* the trigger condition occurred remain in the buffer.

The "EX" command variant is defined in the protocol version 3 to support devices with 32-bit address space.

Command Data:

This command must carry the recorder setup structure and the recorder variables' definition in the same form as the MCB_CMD_SETUPSCOPE command.

```
// Data passed to MCB_CMD_SETUPREC command
typedef struct {
  uint8_t   trgMode;      // trigger mode (0 = disabled, 1 = _/ , 2 = \_)
  uint16_t  totalSmps;    // buffer size in samples
  uint16_t  postTrigger;  // number of samples after trigger to save
  uint16_t  timeDiv;      // time base unit multiplier (0 = fastest)
  uint16_t  trgVarAddr;   // trigger variable address
  uint8_t   trgVarSize;   // trigger variable/threshold size (1,2,4)
  uint8_t   trgVarSigned; // trigger compare mode (0 = unsigned, 1 = signed)
  union {
    uint8_t   b;                // trgVarSize == 1
    uint16_t  w;                // trgVarSize == 2
    uint32_t  dw;               // trgVarSize == 4
  } trgThreshold;               // trigger comparing threshold
  MCB_DATA_SETUPSCOPE_t vars;   // recorded variables
} MCB_DATA_SETUPREC_t;

// Data passed to MCB_CMD_SETUPRECEX command
typedef struct {
```

```
  uint8_t   trgMode;      // trigger mode (0 = disabled, 1 = _/ , 2 = \_)
  uint16_t  totalSmps;    // buffer size in samples
  uint16_t  postTrigger;  // number of samples after trigger to save
  uint16_t  timeDiv;      // time base unit multiplier (0 = fastest)
  uint32_t  trgVarAddr;   // trigger variable address
  uint8_t   trgVarSize;   // trigger variable/threshold size (1,2,4)
  uint8_t   trgVarSigned; // trigger compare mode (0 = unsigned, 1 = signed)
  union {
    uint8_t   b;                // trgVarSize == 1
    uint16_t  w;                // trgVarSize == 2
    uint32_t  dw;               // trgVarSize == 4
  } trgThreshold;               // trigger comparing threshold
  MCB_DATA_SETUPSCOPEEX_t vars; // recorded variables
} MCB_DATA_SETUPRECEX_t;
```

`trgMode`

Trigger mode, when 0, the trigger feature is disabled and the recorder must be stopped manually using MCB_CMD_STOPREC command. When this value is non-zero, it determines the threshold crossing slope for the trigger condition.

`totalSmps`

The recorder buffer length (in variable sets count)

`postTrigger`

The number of sets, which are recorded after the trigger occurs.

`timeDiv`

This variable specifies the frequency at which samples are recorded:
0 = record samples every interrupt
1 = record samples every second interrupt
2 = record samples every third interrupt
etc...

`trgVarAddr`

The address of trigger variable

`trgVarSize`

The size of trigger variable and threshold which can be 1, 2 or 4 bytes

`trgVarSigned`

The trigger compare mode. When nonzero, the trigger comparator treats the trigger variable and the threshold value as signed values

`trgThreshold`

The threshold value; the trgVarSize determines what bytes of this field are valid

`vars`

The recorder variables' definitions, in exactly the same format as the MCB_CMD_SETUPSCOPE command.

Possible Responses:

MCB_STS_OK (0x00)

A successful operation; no data is returned.

MCB_STC_INVCMD (0x81)

The recorder service is not implemented.

MCB_STC_INVBUFF (0x85)

The recorder buffer could not be initialized because the *totalSmps* argument is too large.

MCB_STC_INVSIZE (0x86)

The recorder variable sizes must be a multiple of data bus width. See MCB_CMD_GETINFO command.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command checksum value is invalid.

See also: command codes table, status codes table

MCB_CMD_SETUPSCOPE (0x08), MCB_CMD_SETUPSCOPEEX (0x0a)

This command directs the target to set up the oscilloscope variables. For the fastest possible read access to the selected memory locations, up to eight locations can be selected as the "oscilloscope variables" and their values can be obtained immediately using the special fast command MCB_CMD_READSCOPE.

The "EX" command variant is defined in the protocol version 3 to support devices with 32-bit address space.

Command Data:

This command must include the number of scope variables followed by the appropriate array of definition structures.

```
// Scope setup variable structure
typedef struct {
  uint8_t size;       // variable size
  uint16_t addr;      // variable address
} MCB_DATA_SETUPSCOPE_VARDEF_t;

// Scope setup variable structure extended
typedef struct {
  uint8_t size;       // variable size
  uint32_t addr;      // variable address
} MCB_DATA_SETUPSCOPEEX_VARDEF_t;

// Data passed to MCB_CMD_SETUPSCOPE command
typedef struct {
  uint8_t varCnt;     // number of valid var definitions
  MCB_DATA_SETUPSCOPE_VARDEF_t varDef[1];
} MCB_DATA_SETUPSCOPE_t;

// Data passed to MCB_CMD_SETUPSCOPEEX command
typedef struct {
  uint8_t varCnt;     // number of valid var definitions
  MCB_DATA_SETUPSCOPEEX_VARDEF_t varDef[1];
} MCB_DATA_SETUPSCOPEEX_t;
```

`varCnt`
: *The number of variable definition structures which follow*

`size`
: *The size of the memory location, which must be a multiple of the target data bus width*

`addr`
: *The address of the memory location*

Possible Responses:

MCB_STS_OK (0x00)

A successful operation; no data is returned.

MCB_STC_INVCMD (0x81)

Oscilloscope variables are not supported.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command checksum value is invalid.

See also: command codes table, status codes table

MCB_CMD_STARTREC (0xc1)

This command directs the target to start the recorder. When the recorder is running, it can be stopped manually by using the MCB_CMD_STOPREC command. It is also stopped automatically when the trigger condition is satisfied.

Command Data:

This command carries no data.

Possible Responses:

MCB_STS_OK (0x00)

The recorder was started; the response carries no data.

MCB_STS_RECRUN (0x01)

The recorder is already running; the response carries no data.

MCB_STC_NOTINIT (0x88)

An error has occurred; the recorder was not initialized with the MCB_CMD_SETUPREC command.

MCB_STC_INVCMD (0x81)

The recorder service is not implemented.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command checksum value is invalid.

See also: command codes table, status codes table

MCB_CMD_STOPREC (0xc2)

This command directs the target to stop the recorder.

Command Data:

This command carries no data.

Possible Responses:

MCB_STS_OK (0x00)

A successful operation; the recorder was stopped. The response carries no data.

MCB_STS_RECDONE (0x02)

The recorder was already stopped; the response carries no data.

MCB_STC_NOTINIT (0x88)

An error has occurred; the recorder has not been initialized with the MCB_CMD_SETUPREC command.

MCB_STC_INVCMD (0x81)

The recorder service is not implemented.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command checksum value is invalid.

See also: command codes table, status codes table

MCB_CMD_WRITEMEM (0x02), MCB_CMD_WRITEMEMEX (0x05)

This command directs the target to write the block of memory. The "EX" command variant is defined in the protocol version 3 to support devices with 32-bit address space.

Command Data:

The structure specifying the address and size of the memory block, followed by the block of data bytes.

The caller must be aware that the size of the target's receive buffer is limited and cannot write large data blocks. The target data bus width must also be considered. When writing a large memory block, it needs to be split into multiple MCB_CMD_WRITEMEM commands. When the "size" bytes are written to address A0, the next write should access the address A1 = A0 + size/dataBusWdt. The size of the receive buffer and data bus width can be determined using the MCB_CMD_GETINFO command.

```
// Data passed to MCB_CMD_WRITEMEM command
typedef struct {
```

```
  uint8_t size;      // write buffer size
  uint16_t addr;     // dest memory address
  uint8_t buffer[1]; // data block follows
} MCB_DATA_WRITEMEM_t;

// Data passed to MCB_CMD_WRITEMEMEX or MCB_CMD_WRITEMEMMASKEX command
typedef struct {
  uint8_t size;      // write buffer size
  uint32_t addr;     // dest memory address
  uint8_t buffer[1]; // data block follows
} MCB_DATA_WRITEMEMEX_t;
```

`size`

The size of memory block being written

`addr`

The address where the memory block is to be stored

`buffer`

The array of "size" data bytes

Possible Responses:

MCB_STS_OK (0x00)

A successful operation; no data is returned.

MCB_STS_CMDBUFFOVF (0x83)

A buffer overflow occurred in the command receive buffer because the command message is too long.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command checksum value is invalid.

See also: MCB_CMD_WRITEVARx, MCB_CMD_READMEM(EX), command codes table, status codes table

MCB_CMD_WRITEMEMMASK (0x03), MCB_CMD_WRITEMEMMASKEX (0x06)

This command directs the target to write the block of memory with AND mask. Only the data bits which correspond to those set in the mask are written to target memory.

The "EX" command variant is defined in the protocol version 3 to support devices with 32-bit address space.

Command Data:

This command must carry the structure specifying the address and the size of the memory block, followed by the block of data bytes and the block of mask bytes.

The caller must be aware that the size of the target's receive buffer is limited and cannot cope with large data blocks. The target data bus width must also be considered. When

splitting a large memory block, use multiple MCB_CMD_WRITEMEMMASK commands. When the "size" bytes are written to the address A0, the next write should access the address A1 = A0 + size/dataBusWdt. The size of the receive buffer and the data bus width can be determined using the MCB_CMD_GETINFO command.

```
// Data passed to MCB_CMD_WRITEMEM command
typedef struct {
  uint8_t size;      // write buffer size
  uint16_t addr;     // dest memory address
  uint8_t buffer[1]; // data block follows
} MCB_DATA_WRITEMEM_t;

// Data passed to MCB_CMD_WRITEMEMEX or MCB_CMD_WRITEMEMMASKEX command
typedef struct {
  uint8_t size;      // write buffer size
  uint32_t addr;     // dest memory address
  uint8_t buffer[1]; // data block follows
} MCB_DATA_WRITEMEMEX_t;
```

`size`
The size of memory block being written

`addr`
The address where the memory block is to be stored

`buffer`
The array of "size" data bytes, followed by the array of "size" mask bytes

Possible Responses:

MCB_STS_OK (0x00)

A successful operation; no data is returned.

MCB_STS_CMDBUFFOVF (0x83)

A buffer overflow occurred in the command receive buffer because the command message is too long.

MCB_STC_INVCMD (0x81)

This response indicates that masked write is not supported.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command checksum value was invalid.

See also: MCB_CMD_WRITEVARxMASK, command codes table, status codes table

MCB_CMD_WRITEVAR8 (0xe3) - V2+

This is a fast command that directs the target to write a 1-byte-long variable.

MCB_CMD_WRITEVAR16 (0xe4) - V2+

This is a fast command that directs the target to write a 2-byte-long variable.

MCB_CMD_WRITEVAR32 (0xf0) - V2+

This is a fast command that directs the target to write a 4-byte-long variable.

These commands have been specified in Protocol V2 as the fast alternatives to the MCB_CMD_WRITEMEM command.

Note: The data part of the MCB_CMD_WRITEVAR8 command cannot be optimized to 3 bytes (see protocol description) and therefore has the same length as a standard 1-byte-long write using MCB_CMD_WRITEMEM command.

Command Data:

The structure specifying the destination address and the variable value.

```
// Data structure for fast 8 or 16 bit write
typedef struct {
  uint16_t addr;         // memory location address
  union {
    uint8_t val_b;       // 8-bit value for MCB_CMD_WRITEVAR8
    uint16_t val_w;      // 16-bit value for MCB_CMD_WRITEVAR16
  };
} MCB_DATA_WRITEVAR_t;

// Data structure for fast 8 or 16 bit write
typedef struct {
  uint16_t addr;         // memory location address
  uint32_t val;          // 32-bit value
} MCB_DATA_WRITEVAR32_t;
```

`addr`
The address of the variable's memory location

`val_b`
The value of the 1-byte-long variable for the MCB_CMD_WRITEVAR8 command

`val_w`
The value of the 1-byte-long variable for the MCB_CMD_WRITEVAR16 command

`val`
The value of the 4-byte-long variable for the MCB_CMD_WRITEVAR32 command

Possible Responses:

MCB_STS_OK (0x00)

A successful operation; no data is returned.

MCB_STC_INVCMD (0x81)

The command is not implemented.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command checksum value is invalid.

See also: MCB_CMD_WRITEMEM, MCB_CMD_READVARx, command codes table, status

codes table

MCB_CMD_WRITEVAR8MASK (0xe5) - V2+

This is a fast command directing the target to write the 1-byte-long variable with AND mask.

MCB_CMD_WRITEVAR16MASK (0xf1) - V2+

This is a fast command that directs the target to write the 2-byte-long variable with AND mask.

These commands have been specified in Protocol V2 as the fast alternatives to the MCB_CMD_WRITEMEMMASK command.

Command Data:

This command must include the structure specifying the destination address, the variable value and the mask.

```
// Data structure for fast 8 bit masked write
typedef struct {
  uint16_t addr;     // memory location address
  uint8_t val;       // variable value
  uint8_t mask;      // mask value
} MCB_DATA_WRITEVAR8MASK_t;

// Data structure for fast 16 bit masked write
typedef struct {
  uint16_t addr;     // memory location address
  uint16_t val;      // variable value
  uint16_t mask;     // mask value
} MCB_DATA_WRITEVAR16MASK_t;
```

`addr`
: *The address of the variable's memory location*

`val`
: *The variable value*

`mask`
: *The mask value*

Possible Responses:

MCB_STS_OK (0x00)

A successful operation; no data is returned.

MCB_STC_INVCMD (0x81)

The command is not implemented.

MCB_STC_CMDCSERR (0x82)

A communication error has occurred; the command checksum value is invalid.

See also: MCB_CMD_WRITEMEMMASK, command codes table, status codes table

Library Reference

Communication Library Reference

The communication library contains functions and data structures, serial-line handlers and synchronization methods that provide an easy-to-use communication protocol implementation.

RS232 Handler

The library handles all serial communication. It supplies only the communication speed used to access the board and the COM port number when the library is initialized.

Synchronization

All critical library functions are thread-safe and can be called from any thread without explicit synchronization. Calls from one thread are blocked until another thread finishes its board access (i.e., sending a command, waiting for a response or receiving a response).

Functions Reference

HRESULT_t Return Values

Return values of type HRESULT_t follow the same model as protocol status bytes except that HRESULT_t return values are 32 bits long. The Most Significant Bit (MSB), when zero, signals the success of the operation. When this bit is set, the operation has failed. The lower word of the value can be examined for details about success or error. When the higher byte of this word is zero, the lower byte is a copy of the single-byte status code returned in the board response message. For a detailed description of the function return codes, see the return codes table.

Startup / Cleanup Functions

```
HRESULT_t McbOpenCom(HMCBCOM_t* apCom, const char* acpDevName,
        unsigned int aBaudRate, int64_t aTimeOut = 0);
void McbCloseCom(HMCBCOM_t aCom);

HRESULT_t McbDetect(HMCBCOM_t aCom, MCB_RESP_GETINFO_t* apInfo);
HRESULT_t McbLinSyncAndDetect(HMCBCOM_t aCom, MCB_RESP_GETINFO_t* apInfo);
HRESULT_t McbGetInfo(HMCBCOM_t aCom, MCB_RESP_GETINFO_t* apInfo);
```

Board Memory Access Functions

```
HRESULT_t McbReadDataMem(HMCBCOM_t aCom, void* apDest, uint32_t aAddr, uint16_t aSize);
HRESULT_t McbWriteDataMem(HMCBCOM_t aCom, const void* apcSrc, uint32_t aAddr, uint16_t aSize);
HRESULT_t McbWriteDataMemMask(HMCBCOM_t aCom, const void* apcSrc,
        const void* aMask, uint32_t aAddr, uint16_t aSize);
HRESULT_t McbReadVar8(HMCBCOM_t aCom, void* apDest, uint32_t aAddr);
HRESULT_t McbReadVar16(HMCBCOM_t aCom, void* apDest, uint32_t aAddr);
HRESULT_t McbReadVar32(HMCBCOM_t aCom, void* apDest, uint32_t aAddr);
HRESULT_t McbReadVar(HMCBCOM_t aCom, void* apDest, uint32_t aAddr, uint16_t aSize);
HRESULT_t McbWriteVar(HMCBCOM_t aCom, const void* apcSrc, uint32_t aAddr, uint16_t aSize);
HRESULT_t McbWriteVarBits(HMCBCOM_t aCom, const void* apcSrc, const void* apcMask,
        uint32_t aAddr, uint16_t aSize);
```

Oscilloscope Functions

```
HRESULT_t McbSetupScope(HMCBCOM_t aCom, uint32_t aVarsCnt, MCB_SCOPE_t * apPs);
HRESULT_t McbReadScope(HMCBCOM_t aCom);
```

Recorder Functions

```
HRESULT_t McbSetupRec(HMCBCOM_t aCom, uint32_t aVarsCnt, MCB_SCOPE_t* apScope, MCB_RECORDER_t* apRec);
HRESULT_t McbStartRec(HMCBCOM_t aCom);
HRESULT_t McbStopRec(HMCBCOM_t aCom);
HRESULT_t McbGetRecStatus(HMCBCOM_t aCom);
HRESULT_t McbReadRec(HMCBCOM_t aCom);
```

Application Commands Functions

```
HRESULT_t McbSendAppCmd(HMCBCOM_t aCom, uint8_t aCode, uint32_t aArgSize, const void* apcArgBuff);
HRESULT_t McbGetAppCmdStatus(HMCBCOM_t aCom, uint8_t* apCmdStatus);
HRESULT_t McbGetAppCmdData(HMCBCOM_t aCom, void* aDest, uint16_t aSize, uint16_t aOffset);
HRESULT_t McbCancelAppCmd(HMCBCOM_t aCom);
```

Board Byte Ordering Functions

```
HRESULT_t McbMtoH(HMCBCOM_t aCom, void* apBuff, uint16_t aSize);
HRESULT_t McbHtoM(HMCBCOM_t aCom, void* apBuff, uint16_t aSize);
```

Raw Packets Send and Receive Functions (advanced use)

```
HRESULT_t McbSendRecvPacket(HMCBCOM_t aCom, uint8_t aCmd, const void* apcCmdBuff,
        uint16_t aCmdSize, void* apRespBuff, uint16_t aRespSize, uint8_t aFeatureId);
```

See also: command codes table

Function Return Codes

Standard HRESULT_t Return Values

Success Codes

MCB_S_OK	0x00	Global success code; this value is returned after a successful operation
MCB_S_RECRUN	0x01	For the recorder-related functions; this is a copy of MCB_STC_RECRUN board status code
MCB_S_RECDONE	0x02	For the recorder-related functions; this is a copy of MCB_STC_RECDONE board status code

Error Codes

MCB_E_INVCMD	0x80000081	Error bit extension for MCB_STC_INVCMD board status code
MCB_E_CMDCSERR	0x80000082	Error bit extension for MCB_STC_CMDCSERR board status code
MCB_E_CMDTOOLONG	0x80000083	Error bit extension for MCB_STC_CMDTOOLONG board status code
MCB_E_RSPBUFFOVF	0x80000084	Error bit extension for MCB_STC_RSPBUFFOVF board status code
MCB_E_INVBUFF	0x80000085	Error bit extension for MCB_STC_INVBUFF board status code
MCB_E_INVSIZE	0x80000086	Error bit extension for MCB_STC_INVSIZE board status code
MCB_E_SERVBUSY	0x80000087	Error bit extension for MCB_STC_SERVBUSY board status code
MCB_E_NOTINIT	0x80000088	Error bit extension for MCB_STC_NOTINIT board status code
MCB_E_CANTWRITE	0x80000100	COM port write error

MCB_E_RESPTMOUT	0x80000101	No response form the board (timeout)
MCB_E_RESPCSERR	0x80000102	Response checksum error
MCB_E_CANTDETECT	0x80000103	Board cannot be detected
MCB_E_BUFFSMALL	0x80000104	Operation buffer (destination memory buffer) is too small
MCB_E_INVPOINTER	0x80000105	Pointer is invalid
MCB_E_LIMITEXCDED	0x80000106	Invalid count of entities specified
MCB_E_A32NOTSUPP	0x80000107	32-bit addresses not supported
MCB_E_D32NOTSUPP	0x80000108	32-bit data not supported
MCB_E_ERRSIZE	0x80000109	Invalid size specified
MCB_E_ERRSIZEBUS	0x8000010A	The width of the board bus disallows the operation
MCB_E_ERRDATA	0x8000010B	Wrong data
MCB_E_RESPWRONG	0x8000010C	Wrong (unexpected) response
MCB_E_NOTIMPL	0x8000FFFE	Service not implemented in library
MCB_E_UNEXPECTED	0x8000FFFF	Unspecified global error

See also: status codes table, library reference

Library Functions

McbCloseCom

This function closes the board communication port and frees any resources that were allocated using the board.

```
void McbCloseCom(
    HMCBCOM_t aCom    // board handle
);
```

Parameters:

aCom
[in] A handle obtained using McbOpenCom call.

Return Values:

This function returns no value.

See also : McbOpenCom, library reference

McbDetect

This function identifies the connected board using the MCB_CMD_GETINFO or the MCB_CMD_GETINFOBRIEF command.

```
HRESULT_t McbDetect(
    HMCBCOM_t aCom,                // board handle
    MCB_RESP_GETINFO_t* apInfo     // destination buffer
```

```
);
```

Parameters:

aCom
[in] A handle obtained using McbOpenCom call.
apInfo
[out] A pointer to the buffer which should receive the board detection results. See the MCB_CMD_GETINFO or MCB_CMD_GETINFOBRIEF commands for a description of the information structure.

Return Values:

This function returns a HRESULT_t status code. Use MCB_FAILED or MCB_SUCCEEDED macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : MCB_CMD_GETINFO, MCB_CMD_GETINFOBRIEF, McbCloseCom, McbGetInfo, library reference

McbGetAppCmdStatus

This function retrieves the target-application command status using the MCB_CMD_GETAPPCMDSTS command.

```
#define MCB_APPCMDRESULT_NOCMD      0xff
#define MCB_APPCMDRESULT_RUNNING    0xfe
HRESULT_t McbGetAppCmdStatus(
    HMCBCOM_t aCom,                 // board handle
    uint8_t* apCmdStatus            // pointer to the destination buffer
);
```

Parameters:

aCom
[in] A handle obtained using the McbOpenCom call.
apCmdStatus
[out] A pointer to the BYTE buffer which receives the status of the current application command process. That is a specific return value from the target board application or special status bytes indicating that the command is being processed or that no command has been received yet.

Return Values:

This function returns a HRESULT_t status code. Use MCB_FAILED or MCB_SUCCEEDED macros from *mcberr.h* to find out whether the function succeeded.

See also : McbSendAppCmd, library reference

McbGetInfo

This function is similar to the McbDetect function, but this call returns immediately with the information cached from the last call to *McbDetect*. This function calls *McbDetect* if no

information is available in the cache.

```
HRESULT_t McbGetInfo(
    HMCBCOM_t aCom,               // board handle
    MCB_RESP_GETINFO_t* apInfo    // destination buffer
);
```

Parameters:

aCom
[in] The handle obtained using McbOpenCom call.
apInfo
[out] The pointer to the buffer that's going to receive the board info. See the MCB_CMD_GETINFO or MCB_CMD_GETINFOBRIEF commands for a more detailed description of the MCB_RESP_GETINFO_t structure.

Return Values:

This function returns a HRESULT_t status code. Use MCB_FAILED and MCB_SUCCEEDED macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : MCB_CMD_GETINFO, MCB_CMD_GETINFOBRIEF, McbCloseCom, McbDetect, library reference

McbGetRecStatus

This function retrieves the recorder status using the MCB_CMD_GETRECSTS command.

```
HRESULT_t McbGetRecStatus(
    HMCBCOM_t aCom    // the port handle
);
```

Parameters:

aCom
[in] The handle obtained using McbOpenCom call.

Return Values:

This function returns a HRESULT_t status code. Use MCB_FAILED and MCB_SUCCEEDED macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : MCB_CMD_GETRECSTS, McbSetupRec, McbStartRec, McbStopRec, McbReadRec, library reference

McbMtoH, McbHtoM

These functions convert the memory buffer to the proper byte order. To find out whether to swap the byte order, MCB_CFGFLAG_BIGENDIAN flag bit in the board information

structure (see McbDetect and McbGetInfo) is checked. The supplied memory buffer byte order is swapped if the host (PC) and the target board have different endianess.

```
HRESULT_t McbMtoH(
    HMCBCOM_t aCom,       // port handle
    void* apBuff,         // the buffer to convert
    uint16_t aSize        // buffer size
);
HRESULT_t McbHtoM(
    HMCBCOM_t aCom,       // port handle
    void* apBuff,         // the buffer to convert
    uint16_t aSize        // buffer size
);
```

Parameters:

aCom
[in] The handle obtained using McbOpenCom call.
apBuff
[in/out] A pointer to the memory buffer to be converted
aSize
[in] The size of the buffer

Return Values:

This function returns a HRESULT_t status code. Use MCB_FAILED and MCB_SUCCEEDED macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : library reference

McbOpenCom

This function opens the board communication port.

```
HRESULT_t McbOpenCom(
    HMCBCOM_t* apCom,           // a pointer to the variable to store the handle
    const char* acpDevName,     // OS specific name of the device, e.g. “COM3”
    unsigned int aBaudRate,     // the baud rate to use
    int64_t aTimeOut = 0        // timeout period
);
```

Parameters:

apCom
[out] A pointer to a variable which receives the communication handle if the call is successful. This value is required in subsequent calls to any library functions.
acpDevName
[in] The OS specific name of the port.
aBaudRate
[in] The communication speed used to access the board [bauds per second]
aTimeOut
[in] The maximum number of milliseconds to wait before giving up

Return Values:

This function returns a HRESULT_t status code. Use MCB_FAILED and MCB_SUCCEEDED macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : McbCloseCom, McbDetect, library reference

McbReadDataMem

This function reads the block of data and splits it into multiple MCB_CMD_READMEM commands if needed.

```
HRESULT_t McbReadDataMem(
    HMCBCOM_t aCom,     // board handle
    void* apDest,       // the destination buffer
    uint32_t aAddr,     // the target address
    uint16_t aSize      // the size of the memory block
);
```

Parameters:

aCom
[in] A handle obtained using the McbOpenCom call.
apDest
*[out] A pointer to the buffer which should receive the memory block, this buffer must be at least " **size**" bytes long.*
aAddr
[in] The target side address of the memory block
aSize
[in] The size of the memory block

Return Values:

This function returns a HRESULT_t status code. Use MCB_FAILED and MCB_SUCCEEDED macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : MCB_CMD_READMEM , McbReadVar, library reference

McbReadRec

This function reads the recorder memory using the MCB_CMD_GETRECBUFF and MCB_CMD_READMEM commands. It uses the destination buffer pointers and sizes specified at the McbSetupRec call.

```
HRESULT_t McbReadRec(
    HMCBCOM_t aCom          // the port handle
);
```

Parameters:

aCom
[in] The handle obtained using McbOpenCom call.

Return Values:

This function returns a HRESULT_t status code. Use MCB_FAILED and MCB_SUCCEEDED macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : MCB_CMD_GETRECBUFF, McbSetupRec, McbStartRec, McbStopRec, McbGetRecStatus, library reference

McbReadScope

This function reads the set of oscilloscope variables using the fast MCB_CMD_READSCOPE command. It uses the destination buffer pointers specified at the McbSetupScope call.

```
HRESULT_t McbReadScope(
    HMCBCOM_t aCom          // the port handle
);
```

Parameters:

aCom
[in] The handle obtained using McbOpenCom call.

Return Values:

This function returns a HRESULT_t status code. Use MCB_FAILED and MCB_SUCCEEDED macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : MCB_CMD_READSCOPE, McbSetupScope, library reference

McbReadVar

This function reads a variable from the target. This is almost the same operation as McbReadDataMem except that **McbReadVar** is optimized to use the fast MCB_CMD_READVARx commands based on the variable size (Fast reads can be disabled by the target board using configuration flags; see McbDetect or McbGetInfo).

The second important difference is that the correct byte order is taken into account using this function (see McbMtoH or McbHtoM functions).

```
HRESULT_t McbReadVar(
    HMCBCOM_t aCom,         // the port handle
    void* apDest,           // destination buffer
    uint32_t aAddr,         // variable address
    uint16_t aSize          // variable size
);
```

Parameters:

aCom
[in] The handle obtained using McbOpenCom call.
apDest
*[out] The pointer to the buffer to be filled with the variable value, this buffer must be at least " **size**" bytes long.*
aAddr
[in] The target side address of the variable
aSize
[in] The size of the variable

Return Values:

This function returns a HRESULT_t status code. Use MCB_FAILED and MCB_SUCCEEDED macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : MCB_CMD_READVARx , McbReadDataMem, library reference

McbSendAppCmd

This function sends the target-application command and its arguments using the MCB_CMD_SENDAPPCMD command.

```
HRESULT_t McbSendAppCmd(
    HMCBCOM_t aCom,                 // board handle
    uint8_t aCode,                  // application command code
    uint32_t aArgSize,              // arguments size
    const void* apcArgBuff          // arguments buffer
);
```

Parameters:

aCom
[in] The handle obtained using McbOpenCom call.
aCode
[in] A specific command code recognized by the board application
aArgSize
[in] The size of the supplied arguments buffer
apcArgBuff
[in] The arguments recognized by the board application

Return Values:

This function returns a HRESULT_t status code. Use MCB_FAILED and MCB_SUCCEEDED macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : MCB_CMD_SENDAPPCMD, McbGetAppCmdStatus, library reference

McbSendRecvPacket

This function combines the functionality of two other functions, which were made obsolete in the current library version (McbSendPacket and McbRecvResponse).

The function sends a command packet to the board and waits for the response packet. In case of a local connection, the function handles RS232 communication issues and protocol specifics. When communication plug-in module is used to connect to the board, the function call is forwarded to the module.

```
HRESULT_t McbSendRecvPacket(
    HMCBCOM_t aCom,             // the port handle
    uint8_t aCmd,               // command number
    const void* apcCmdBuff,     // command data
    uint16_t aCmdSize,          // command data size
    void* apRespBuff,           // response destination buffer
    uint16_t aRespSize,         // expected response length
    uint8_t aFeatureId          // id of the feature addressed by the command
);
```

Parameters:

aCom
[in] The handle obtained using McbOpenCom call.
aCmd
[in] The command code; see the command codes table for the list of valid command codes.
apcCmdBuff
[in] The pointer to the memory buffer which contains the command data (NULL, if command carries no data)
aCmdSize
[in] The length of the command data
apRespBuff
[out] The pointer to the memory buffer which receives the data part of the response packet (when the response status code is "success").
aRespSize
[in] The expected length of the data part in the response packet. The destination buffer must be at least of this size.
aFeatureId
[in] The board or protocol feature, which is addressed by the command or 0 if the command does not address any special feature. ***Currently not supported by the Freemaster NG Addon.***

Return Values:

This function returns a HRESULT_t status code. Use MCB_FAILED and MCB_SUCCEEDED macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : McbLockFeature, McbUnlockFeature, command codes table, status codes table, library reference

McbSetupRec

This function uses the MCB_CMD_SETUPREC command to set up the recorder.

```
// scope/rec setup
typedef struct MCB_SCOPE_t {
    uint8_t   size;     /**< @brief  variable remote size */
    uint32_t  addr;     /**< @brief  variable remote address */
    void*     destPtr;  /**< @brief  destination buffer */
    uint32_t  destSize; /**< @brief  destination buffer size (robust checking) */
```

```
}
// recorder setup
typedef struct MCB_RECORDER_t {
  uint8_t trgMode;       /**< @brief trigger mode */
  uint16_t totalSmps;    /**< @brief total number of samples required */
  uint16_t preTrigger;   /**< @brief number of samples of pre-trigger */
  uint16_t timeDiv;      /**< @brief time base unit multiplier */
  uint32_t trgVarAddr;   /**< @brief trigger variable address */
  uint8_t  trgVarSize;   /**< @brief trigger variable size (1, 2 or 4) */
  uint8_t  trgVarSigned; /**< @brief signed flag (0 = unsigned, 1 = signed) */
  union {
    uint8_t b;         /**< @brief trgVarSize == 1 */
    uint16_t w;        /**< @brief trgVarSize == 2 */
    uint32_t dw;       /**< @brief trgVarSize == 4 */
  } trgThreshold;      /**< @brief trigger comparing threshold */
} MCB_RECORDER_t;

HRESULT_t McbSetupRec(
    HMCBCOM_t aCom,           // the port handle
    uint32_t aVarsCnt,        // the number of variables used by the recorder
    MCB_SCOPE_t* apScope,     // the variables definition
    MCB_RECORDER_t* apRec     // the recorder settings
);
```

Parameters:

aCom
[in] The handle obtained using McbOpenCom call.
aVarsCnt
[in] The number of variables used in the scope setup; this is also the size of the apScope array
apScope
[in] The array of structures defining the monitored variables.
apRec
[in] The structure defining the specific recorder parameters (see below)

Structures:

MCB_SCOPE_t

size
[in] The size of the variable
addr
[in] The address of the variable
destPtr
*[out] A pointer to the memory which receives the data when the McbReadRec function is called. The buffer should be large enough to hold all the recorder data for the variable (variable size * the number of samples).*
destSize
[in] The size of the buffer. This value is used by the library for precise checking.

MCB_RECORDER_t

trgMode
[in] The trigger mode (0 = no trigger, 1 = positive slope , 2 = negative slope)
totalSmps
[in] The number of samples to be recorded for each variable
preTrigger
[in] The number of samples to be saved in the recorder circular buffer from the time before the trigger occurred

`timeDiv`
[in] The time base unit multiplier. When 0, the recording takes place every DSP/MCU timer interrupt, 1 every second, etc...
`trgVarAddr`
[in] The trigger variable address (NULL if no trigger is used)
`trgVarSize`
[in] The size of the trigger variable; only standard sizes of integer variables are valid (1, 2 or 4)
`trgVarSigned`
[in] When `false` (zero), the trigger variable will be treated as unsigned; when `true`, the variable will be treated as signed
`trgThreshold`
[in] The trigger threshold value. Fill in the appropriate member of the union according to the trigger variable size.

Return Values:

This function returns a `HRESULT_t` status code. Use `MCB_FAILED` and `MCB_SUCCEEDED` macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : MCB_CMD_SETUPREC, McbStartRec, McbStopRec, McbGetRecStatus, McbReadRec, library reference

McbSetupScope

This function uses the MCB_CMD_SETUPSCOPE command to set up the oscilloscope variables.

```
// scope/rec setup
typedef struct MCB_SCOPE_t {
    uint8_t   size;     /**< @brief  variable remote size */
    uint32_t  addr;     /**< @brief  variable remote address */
    void*     destPtr;  /**< @brief  destination buffer */
    uint32_t  destSize; /**< @brief  destination buffer size (robust checking) */
}
HRESULT_t McbSetupScope(
    HMCBCOM_t aCom,               // the port handle
    uint32_t aVarsCnt,            // number of variables used in oscilloscope
    MCB_SCOPE_t * apPs            // variables definition
);
```

Parameters:

`aCom`
[in] The handle obtained using McbOpenCom call.
`aVarsCnt`
[in] The number of the variables used in the scope setup; this is also the size of the `apPs` *array*
`apPs`
[in] The array of structures defining the monitored variables

Structures:

MCB_SCOPE_t

`size`
[in] The size of the variable

`addr`
[in] The address of the variable
`destPtr`
*[out] A pointer to the memory which receives the data when the McbReadRec function is called. The buffer should be large enough to hold all the recorder data for the variable (variable size * the number of samples).*
`destSize`
[in] The size of the buffer. This value is used by the library for precise checking.

Return Values:

This function returns a `HRESULT_t` status code. Use `MCB_FAILED` and `MCB_SUCCEEDED` macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : MCB_CMD_SETUPSCOPE, McbReadScope, library reference

McbStartRec

This function starts the recorder using the MCB_CMD_STARTREC command.

```
HRESULT_t McbStartRec(
    HMCBCOM_t aCom                  // the port handle
);
```

Parameters:

`aCom`
[in] The handle obtained using McbOpenCom call.

Return Values:

This function returns a `HRESULT_t` status code. Use `MCB_FAILED` and `MCB_SUCCEEDED` macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : MCB_CMD_STARTREC, McbSetupRec, McbStopRec, McbReadRec, McbGetRecStatus, library reference

McbStopRec

This function stops the recorder using the MCB_CMD_STOPREC command.

```
HRESULT_t McbStopRec(
    HMCBCOM_t aCom                  // the port handle
);
```

Parameters:

`aCom`
[in] The handle obtained using McbOpenCom call.

Return Values:

This function returns a HRESULT_t status code. Use MCB_FAILED and MCB_SUCCEEDED macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : MCB_CMD_STOPREC, McbSetupRec, McbStartRec, McbReadRec, McbGetRecStatus, library reference

McbWriteDataMem

This function writes a block of bytes into the board data memory. It splits the block into multiple MCB_CMD_WRITEMEM commands if needed.

```
HRESULT_t McbWriteDataMem(
    HMCBCOM_t aCom,             // the port handle
    const void* apcSrc,         // source buffer
    uint32_t aAddr,             // target address
    uint16_t aSize              // size of the memory block
);
```

Parameters:

aCom
[in] The handle obtained using McbOpenCom call.
apcSrc
*[in] A pointer to the buffer be written to the board data memory, this buffer is expected to be "**size**" bytes long.*
aAddr
[in] The target board destination address.
aSize
[in] The size of the memory block.

Return Values:

This function returns a HRESULT_t status code. Use MCB_FAILED and MCB_SUCCEEDED macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : MCB_CMD_WRITEMEM , McbWriteVar, library reference

McbWriteDataMemMask

This function writes a block of bytes into the board's data memory using the bit mask. It splits the block into multiple MCB_CMD_WRITEMEMMASK commands if needed.

```
HRESULT_t McbWriteDataMemMask(
    HMCBCOM_t aCom,             // the port handle
    const void* apcSrc,         // source buffer
    const void* aMask,          // mask buffer
    uint32_t aAddr,             // target address
    uint16_t aSize              // size of the memory block
);
```

Parameters:

aCom
[in] The handle obtained using McbOpenCom call.
apcSrc
*[in] A pointer to the buffer to be written to the board data memory; this buffer is expected to be "**size**" bytes long.*
aMask
*[in] A pointer to the buffer which should be used as a bit mask when writing the memory block; this buffer is also expected to be "**size**" bytes long.*
aAddr
[in] The target board destination address
aSize
[in] The size of the memory block (and of the mask block)

Return Values:

This function returns a HRESULT_t status code. Use MCB_FAILED and MCB_SUCCEEDED macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : MCB_CMD_WRITEMEMMASK , McbWriteVarBits, library reference

McbWriteVar

This function is the same as the McbWriteDataMem function except that the correct byte order is taken into account; see McbMtoH or McbHtoM functions.

For 1-byte, 2-byte and 4-byte-long variables, the fast MCB_CMD_WRITEVARx command is used if the board implements the protocol V2 and if the fast writes are not disabled in the information structure (see McbDetect or McbGetInfo).

```
HRESULT_t McbWriteVar(
    HMCBCOM_t aCom,         // the port handle
    const void* apcSrc,     // source buffer
    uint32_t aAddr,         // target address
    uint16_t aSize          // size of the memory block
);
```

Parameters:

aCom
[in] The handle obtained using McbOpenCom call.
apcSrc
*[in] A pointer to the buffer to be written to the board data memory; this buffer is expected to be "**size**" bytes long.*
aAddr
[in] The target board destination address
aSize
[in] The size of the memory block

Return Values:

This function returns a HRESULT_t status code. Use MCB_FAILED and MCB_SUCCEEDED macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : MCB_CMD_WRITEMEM , MCB_CMD_WRITEVARx , MCB_CMD_GETINFO , McbReadVar, library reference

McbWriteVarBits

This function is the same as the McbWriteDataMemMask function, except that the correct byte order is taken into account; see McbMtoH or McbHtoM functions.

For 1-byte-long and 2-byte-long variables, the fast MCB_CMD_WRITEVARxMASK command is used if the board implements the Protocol V2 and if the fast writes are not disabled in the information structure (see McbDetect or McbGetInfo).

```
HRESULT_t McbWriteVarBits(
    HMCBCOM_t aCom,          // the port handle
    const void* apcSrc,      // source buffer
    const void* apcMask,     // mask buffer
    uint32_t aAddr,          // target address
    uint16_t aSize           // size of the memory block (and the mask block)
);
```

Parameters:

aCom
[in] The handle obtained using McbOpenCom call.
apcSrc
*[in] A pointer to the buffer to be written to the board data memory; this buffer is expected to be "**size**" bytes long.*
apcMask
*[in] A pointer to the buffer which should be used as a bit mask when writing the memory block; this buffer is also expected to be "**size**" bytes long.*
aAddr
[in] The target board destination address
aSize
[in] The size of the memory block (and of the mask block)

Return Values:

This function returns a HRESULT_t status code. Use MCB_FAILED and MCB_SUCCEEDED macros from *mcberr.h* to find out whether the function succeeded.

See *mcberr.h* for the list of possible error codes.

See also : MCB_CMD_WRITEMEMMASK , MCB_CMD_WRITEVARxMASK , MCB_CMD_GETINFO , McbReadVar, library reference