iC-MHM DLL

LIBRARY DESCRIPTION

DESCRIPTION

This documentation describes the functionality of the Dynamic Link Library: **MHM_1SL_interface.dll**. DLLs provide the standard benefits of shared libraries, such as modularity. Modularity allows changes to be made to code and data in a single self-contained DLL shared by several applications without any change to the applications themselves. Another benefit of the modularity is the use of generic interfaces for plug-ins. A single interface may be developed which allows old as well as new modules to be integrated seamlessly at run-time into pre-existing applications, without any modification to the application itself. Figure 1 shows the hierarchy of the layer model and illustrates the interconnectivity of the particular grades.

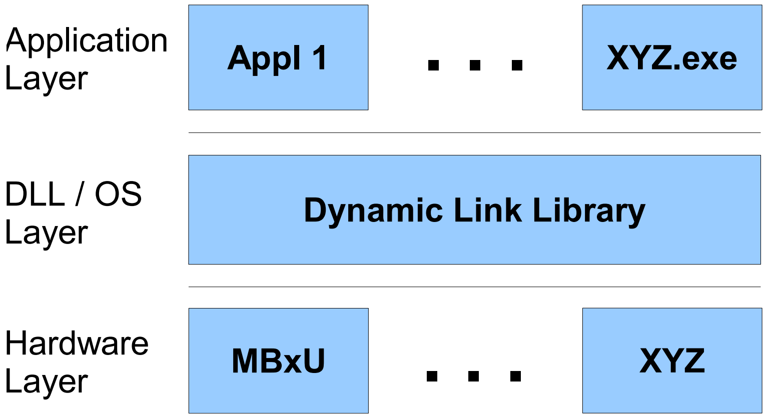

Figure 1: Three layer model

Features

- Reducing evaluation, design-in time and development costs
- Access to the configuration parameters of iC-MHM
- Transfer of predefined configuration HEX files
- Generation of EEPROM HEX files for device configuration
- Reading of iC-MHM sensor data

CONTENTS

GENERAL SETUP 4

MAIN FUNCTIONS 5

MHM_Open . . . 5
MHM_Close . . . 5
MHM_SetInterface . . . 5
MHM_GetInterface . . . 6
MHM_SetConfig . . . 6
MHM_GetConfig . . . 7
MHM_Initialize . . . 7
MHM_SwitchToBiss . . . 7
MHM_SetParam . . . 8
MHM_GetParam . . . 8
MHM_WriteParams . . . 9
MHM_ReadParams . . . 9
MHM_GetCorrectedRegisters . . . 10
MHM_WriteEeprom . . . 10
MHM_ReadEeprom . . . 11
MHM_ReadSens . . . 11
MHM_ReadGainRegister . . . 12
MHM_ReadStatusRegister . . . 12
MHM_WriteCommand . . . 13
MHM_Preset . . . 13
MHM_WritePorts . . . 14
MHM_ReadPorts . . . 14
MHM_SaveParams . . . 14
MHM_LoadParams . . . 15
MHM_GetDLLVersion . . . 15
MHM_GetLastError . . . 15

OPTIONAL FUNCTIONS 17

MHM_SetRegister . . . 17
MHM_GetRegister . . . 17
MHM_WriteRegister . . . 18
MHM_ReadRegister . . . 18
MHM_SaveRegister . . . 19
MHM_LoadRegister . . . 19
MHM_GetInterfaceInfo . . . 20
MHM_SetEdsParam . . . 20
MHM_GetEdsParam . . . 21
MHM_ReadEdsParams . . . 21
MHM_UpdateEDS . . . 21
MHM_SpiActivate . . . 22
MHM_WriteRegister_I2C . . . 22

MHM_WriteRegister_I2C_Ex . . . 23
MHM_ReadRegister_I2C . . . 23
MHM_LoadRegisterEx . . . 24
MHM_SaveRegisterEx . . . 24

CALIBRATION FUNCTIONS 26
MHM_FullCalibration . . . 26
MHM_FullAcquisition . . . 27
MHM_SetCalCfg . . . 28
MHM_GetCalCfg . . . 29
MHM_AdjustGain . . . 29
MHM_ActivateCalCfg . . . 29
MHM_AcquireEx . . . 30
MHM_DeactivateCalCfg . . . 30
MHM_AnalyzeData . . . 31
MHM_Adjust . . . 31
MHM_GetCalParam . . . 32
MHM_SetCalData . . . 32
MHM_GetCalData . . . 33

PARAMETER AND ERROR CODING 34
iC-MHM Parameters . . . 34
DLL Parameters . . . 35
Calibration Parameters (DLL) . . . 37
Error Description . . . 38
EDS Description . . . 39

APPLICATION EXAMPLES 42
Example 1 - Getting Started . . . 42
Example 2 - Automatic Calibration . . . 43
Example 3 - Calibration Step by Step . . . 44
Example 4 - Calibration with external data via BiSS DLL . . . 46

APPENDIX 51

REVISION HISTORY 52

GENERAL SETUP

Supported Programming Environments

This shared library is provided as a 32-bit compilation and can be used in combination with the following programming languages:

- C
- C++
- LabVIEW

Other programming languages are not verified but may be able to use this library.

Operating Systems

This shared library was developed for Microsoft Windows.

USB adapter drivers

To communicate with the evaluation board, USB adapter drivers need to be installed. The minimum required driver versions for this library are:

- MB5U: 6.2.0.0
- MB4U: 6.2.0.0
- MB3U: 2.12.24.0

The driver installation must be completed successfully before connecting the adapter to your PC. If you have not already installed the driver for the USB adapter, run `USB_MB*U_driver_XX.exe` to install the necessary drivers for the MB3U, MB4U or MB5U.

MAIN FUNCTIONS

MHM_Open

Loads the library and returns a handle that will be used for subsequent accesses.

unsigned long **MHM_Open** (unsigned long *pulMHMHandle)

Parameters :

pulMHMHandle	Pointer to the handle.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_Close

Releases the handle and terminates established connections.

unsigned long **MHM_Close** (unsigned long ulMHMHandle)

Parameters :

ulMHMHandle	Handle of the library.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_SetInterface

Connects the interface defined by parameter ulInterface.

unsigned long **MHM_SetInterface** (unsigned long ulMHMHandle,
unsigned long ulInterface)
char *pcInterfaceOption)

Parameters :

`ulMHMHandle` Handle of the library.

`ulInterface` Selection of the interface type.

Value	Name	Description
0	eMHM_NO_INTERFACE	no device
1	eMHM_MB3U_SPI	USB to SPI
2	eMHM_MB3U_BISS	USB to BiSS
3	eMHM_MB4U	USB to BiSS
4	eMHM_MB5U	USB to BiSS

Table 1: Supported devices (MHM_InterfaceEnum)

`pcInterfaceOption` For MB4U/MB5U: Serial number of the adapter (last 4 chars: e.g. "CABC"). Only needed if multiple MB4U/MB5U are connected.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_GetInterface

Returns the selected interface from the DLL.

unsigned long **MHM_GetInterface** (unsigned long `ulMHMHandle`,
unsigned long `*pulInterface`)

Parameters :

`ulMHMHandle` Handle of the library.

`pulInterface` Pointer to the value of the interface type (see table 1, page 6) .

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_SetConfig

Sets the DLL properties.

unsigned long **MHM_SetConfig** (unsigned long `ulMHMHandle`,
unsigned long `ulConfig`,
unsigned long `ulValue`)

Parameters :

`ulMHMHandle` Handle of the library.

`ulConfig` The DLL configuration parameter. A list of configuration parameters can be found in table 10, page 36.

`ulValue` The value of the DLL configuration parameter.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_GetConfig

Gets the DLL properties.

unsigned long **MHM_GetConfig** (unsigned long `ulMHMHandle`,
unsigned long `ulConfig`,
unsigned long `*pulValue`)

Parameters :

`ulMHMHandle`	Handle of the library.
`ulConfig`	The DLL configuration parameter. A list of configuration parameters can be found in table 10, page 36.
`pulValue`	Pointer to the value of the DLL configuration parameter.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_Initialize

Initialize the bus communication.

unsigned long **MHM_Initialize** (unsigned long `ulMHMHandle`)

Parameters :

`ulMHMHandle`	Handle of the library.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_SwitchToBiss

Switches communication to BiSS protocol.

unsigned long **MHM_SwitchToBiss** (unsigned long `ulMHMHandle`,
unsigned long `*pulEnSsi`)

Parameters :

`ulMHMHandle` Handle of the library.
`pulEnSsi` Pointer to the value of ENSSI stored in the EEPROM.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_SetParam

Sets an iC-MHM parameter and recalculates the CRC in the DLL with optional write.

unsigned long **MHM_SetParam** (unsigned long `ulMHMHandle`,
unsigned long `ulParam`,
unsigned long `ulValue`,
unsigned long `ulWriteVerify`)

Parameters :

`ulMHMHandle` Handle of the library.
`ulParam` The iC-MHM configuration parameter. A list of parameters can be found in table 9, page 35.
`ulValue` The value of the iC-MHM configuration parameter.
`ulWriteVerify`

Value	Name	Description
0	eMHM_SETONLY	Sets the parameter to the DLL only
1	eMHM_WRITE	Sets the parameter to the DLL and writes the corresponding register to the iC-MHM immediately.
2	eMHM_VERIFY	Sets the parameter to the DLL, writes the corresponding register to the iC-MHM immediately and reads the register from the iC-MHM for verification. The return value is an error code if verification fails.

Table 2: Write Verification Commands (MHM_WriteVerifyEnum)

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :
Configure parameters with calls of the function **MHM_SetParam**. Then use **MHM_WriteParams** to write all parameters to the iC-MHM. See figure 2.

MHM_GetParam

Gets an iC-MHM parameter from the DLL.

```
unsigned long MHM_GetParam      (unsigned long ulMHMHandle,
                                 unsigned long ulParam,
                                 unsigned long *pulValue)
```

Parameters :

ulMHMHandle	Handle of the library.
ulParam	The iC-MHM configuration parameter. A list of parameters can be found in table 9, page 35.
pulValue	Pointer to the value of the iC-MHM configuration parameter.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :
To get the values saved inside the iC-MHM registers use **MHM_ReadParams** before calling **MHM_GetParam**. See figure 2.

MHM_WriteParams

Writes parameters from the DLL to the iC-MHM registers (Bank0; Addr. 0x00.. 0x13).

```
unsigned long MHM_WriteParams   (unsigned long ulMHMHandle,
                                 unsigned long ulVerify,
                                 unsigned long *pulValid)
```

Parameters :

ulMHMHandle	Handle of the library.
ulVerify	Set to ’1’ to read back all registers for verification. Set to ’0’ for write without verification.
pulValid	Pointer to an array with the size of the count of registers (0x00.. 0x3B = 60). In case of verification, this array is used to specify which registers are valid. If the written value is equal to the read value, the function writes ’1’ in the array at the position corresponding to the register. If the verification fails the array contains a ’0’ at this position.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_ReadParams

Reads registers (SPI and BiSS(Bank0):Addr. 0x00.. 0x13 and 0x73; BiSS only: 0x41.. 0x43, 0x48.. 0x4E, 0x78.. 0x7F) from the iC-MHM to the DLL.

```
unsigned long MHM_ReadParams    (unsigned long ulMHMHandle)
```

Parameters :

`ulMHMHandle`	Handle of the library.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

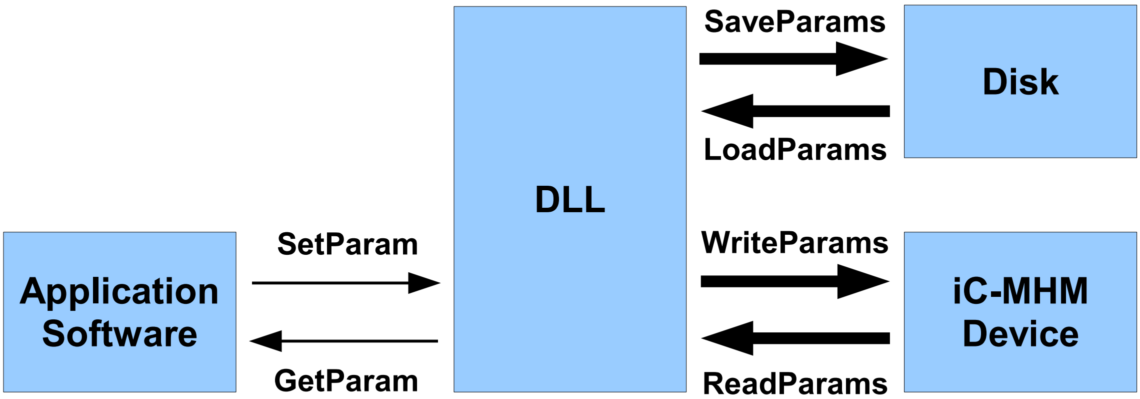

Figure 2: Illustration of the iC-MHM DLL main functions

MHM_GetCorrectedRegisters

In case of an **eMHM_CORRECTED_INVALID_VALUE** error, this function returns an array with the corrected registers.

unsigned long
MHM_GetCorrectedRegisters (unsigned long `ulMHMHandle`,
unsigned long `*pulCorrected`)

Parameters :

`ulMHMHandle`	Handle of the library.
`pulCorrected`	Pointer to an array with a size of the count of registers (0x00.. 0x7F = 128). This array is used to specify which registers have been corrected. If the read value has been corrected, this function writes a ’1’ in the array at the position corresponding to the register. If no correction has been done the array contains a ’0’ at this position.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code.(see table 15, page 39).

MHM_WriteEeprom

Writes all parameters (selected area) from the DLL to the EEPROM.

unsigned long **MHM_WriteEeprom** (unsigned long `ulMHMHandle`,
unsigned long `ulEepromArea`)

Parameters :

`ulMHMHandle` Handle of the library.
`ulEepromArea` Area to be stored in the EEPROM

Value	Name	Description
0	eMHM_E2P_CFG_MHM	iC-MHM only (Bank1; Adr. 0x10..0x23)
1	eMHM_E2P_CFG_MT	Multiturn only (Bank1; Adr. 0x00..0x0F)
2	eMHM_E2P_EDS	EDS only
3	eMHM_E2P_ALL	All
4	eMHM_E2P_CFG_MHM_NO_OFFS	iC-MHM configuration without offsets (Bank1; Adr. 0x10..0x1C)

Table 3: EEPROM area (MHM_E2PAreaEnum)

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_ReadEeprom

Reads all parameters (selected area) from the EEPROM to the DLL.

unsigned long **MHM_ReadEeprom** (unsigned long `ulMHMHandle`,
unsigned long `ulEepromArea`)

Parameters :

`ulMHMHandle` Handle of the library.
`ulEepromArea` Area to be stored in the EEPROM (see table 3)

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_ReadSens

Reads the position value, error bit, warning bit,sign of life counter, status-, port- and gain register from the iC-MHM.

unsigned long **MHM_ReadSens** (unsigned long `ulMHMHandle`,
MHM_ReadSensStruct `*pSensorDataStruct`)

Parameters :

`ulMHMHandle`	Handle of the library.
`pSensorDataStruct`	Pointer to the data structure containing the position values.

Element	Description
ulHighData	Raw data (63..32)
ulLowData	Raw data (31..0)
ulDataLength	Data length
ulMultiturn	Multiturn position value
ulSingleturn	Singleturn position value
ulError	Error Bit
ulWarning	Warning Bit
ulLifeCounter	Sign of life Counter Value
ulStatusRegister	Register Content Adr. 0x70
ulPortRegister	Register Content Adr. 0x71
ulGainRegister	Register Content Adr. 0x72

Table 4: Definition of data type (MHM_ReadSensStruct)

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :
The function **MHM_ReadSens** only works properly if the communication to the iC-MHM is configured. Use the function **MHM_ReadParams** for synchronization of the iC RAM and DLL content to ensure a proper configuration. To enable/disable cyclic read (BiSS mode only) for status-, port- and/or gain register use the function **MHM_SetConfig**.

MHM_ReadGainRegister

Reads the gain register (Addr. 0x72) from the iC-MHM.

unsigned long **MHM_ReadErrorRegister** (unsigned long `ulMHMHandle`,
unsigned long `*pulGain`)

Parameters :

`ulMHMHandle`	Handle of the library.
`pulGain`	Pointer to the value.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :
Use **MHM_GetParam** subsequently to obtain the gain parameters.

MHM_ReadStatusRegister

Reads the status register (Addr. 0x70) from the iC-MHM to the DLL.

unsigned long **MHM_ReadStatusRegister** (unsigned long `ulMHMHandle`)

Parameters :

`ulMHMHandle`	Handle of the library.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :
Use **MHM_GetParam** subsequently to obtain the status parameters.

MHM_WriteCommand

Writes the command register (Addr. 0x74) of iC-MHM.

unsigned long **MHM_WriteCommand** (unsigned long `ulMHMHandle`,
unsigned long `ulCommand`)

Parameters :

`ulMHMHandle`	Handle of the library.
`ulCommand`	The value of the iC-MHM command register.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_Preset

Executes the iC-MHM preset command. Afterwards reads the offset registers of the iC-MHM (Adr. 0x0D..0x13) to the DLL (no reading possible in SSI mode!).

unsigned long **MHM_Preset** (unsigned long `ulMHMHandle`)

Parameters :

`ulMHMHandle`	Handle of the library.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_WritePorts

Writes lower nibble of register(Adr. 0x75) from the iC-MHM.

unsigned long **MHM_WritePorts** (unsigned long `ulMHMHandle`,
unsigned long `ulPorts`)

Parameters :

`ulMHMHandle`	Handle of the library.
`pulPorts`	Value of the ports.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_ReadPorts

Reads lower nibble of register(Adr. 0x71) from the iC-MHM.

unsigned long **MHM_ReadPorts** (unsigned long `ulMHMHandle`,
unsigned long `*pulPorts`)

Parameters :

`ulMHMHandle`	Handle of the library.
`pulPorts`	Pointer to the value.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_SaveParams

Saves iC-MHM parameters from the DLL to a specified file. The file format is Intel HEX

unsigned long **MHM_SaveParams** (unsigned long `ulMHMHandle`,
char `*pcFilename`)

Parameters :

`ulMHMHandle`	Handle of the library.
`pcFilename`	Absolute path of the file.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_LoadParams

Loads iC-MHM parameters from the specified file to the DLL. The file format is Intel HEX

```
unsigned long MHM_LoadParams    (unsigned long ulMHMHandle,
                                 char *pcFilename)
```

Parameters :

`ulMHMHandle`	Handle of the library.
`pcFilename`	Absolute path of the file.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_GetDLLVersion

Gets the version number of the DLL.

```
unsigned long MHM_GetDLLVersion (unsigned long *pulDllVersion)
```

Parameters :

`*pulDllVersion`	Pointer to an array with two elements that contains the DLL version.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :
Index ’1’ of the version number comprises the major version number, Index ’0’ the minor version number. The function does not need a handle.

MHM_GetLastError

Returns details about the last error occurred.

unsigned long
MHM_GetLastError (unsigned long ulMHMHandle,
unsigned long *pulLastError,
unsigned long *pulErrorType,
char *pcErrorText)

Parameters :

ulMHMHandle Handle of the library.

pulLastError Pointer to the error code. (see table 15, page 39)

pulErrorType

Value	Name
0	eMHM_NONE
1	eMHM_HINT
2	eMHM_WARNING
3	eMHM_PROGRAMMING_ERROR
4	eMHM_OPERATING_ERROR
5	eMHM_COMMUNICATION_ERROR

Table 5: MHM_ErrorTypeEnum

pcErrorText Pointer to string with the detailed error text.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

OPTIONAL FUNCTIONS

MHM_SetRegister

Sets an iC-MHM register and recalculates the CRC in the DLL with optional write.

unsigned long **MHM_SetRegister** (unsigned long `ulMHMHandle`,
unsigned long `ulAdr`,
unsigned long `ulValue`,
unsigned long `ulWriteVerify`)

Parameters :

`ulMHMHandle`	Handle of the library.
`ulAdr`	The address of the register.
`ulValue`	The value of the register.

`ulWriteVerify`

Value	Name	Description
0	eMHM_SETONLY	Sets the parameter to the DLL only
1	eMHM_WRITE	Sets the parameter to the DLL and writes the corresponding register to the iC-MHM immediately.
2	eMHM_VERIFY	Sets the parameter to the DLL, writes the corresponding register to the iC-MHM immediately and reads the register from the iC-MHM for verification. The return value is an error code if verification fails.

Table 6: Write Verification Commands (MHM_WriteVerifyEnum)

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :
This function writes a whole register instead of single parameters. Use function **MHM_WriteParams** to write these changes to the iC-MHM RAM.

MHM_GetRegister

Gets an iC-MHM register from the DLL.

unsigned long **MHM_GetRegister** (unsigned long `ulMHMHandle`,
unsigned long `ulAdr`,
unsigned long `*pulValue`)

Parameters :

`ulMHMHandle`	Handle of the library.
`ulAdr`	The address of the register.
`pulValue`	Pointer to the value of the register.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :
To get the register value saved in the iC-MHM RAM, use **MHM_ReadParams** before calling **MHM_GetRegister**.

MHM_WriteRegister

Writes a specified number of registers from the DLL to the iC-MHM registers.

```
unsigned long MHM_WriteRegister   (unsigned long ulMHMHandle,
                                   unsigned long ulAdr,
                                   unsigned long *pulNumberOfBytes,
                                   unsigned long *pulData)
```

Parameters :

`ulMHMHandle`	Handle of the library.
`ulAdr`	The address of the register.
`pulNumberOfBytes`	Pointer to the number of registers. After execution this parameter contains the number of successfully written bytes.
`pulData`	Pointer to an array containing the values.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_ReadRegister

Reads a specified number of registers from the iC-MHM registers to the DLL.

```
unsigned long MHM_ReadRegister    (unsigned long ulMHMHandle,
                                   unsigned long ulAdr,
                                   unsigned long *pulNumberOfBytes,
                                   unsigned long *pulData)
```

Parameters :

`ulMHMHandle`	Handle of the library.
`ulAdr`	The address of the register.
`pulNumberOfBytes`	Pointer to the number of registers. After execution this parameter contains the number of successfully read bytes.
`pulData`	Pointer to an array containing the values.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_SaveRegister

Saves the specified address range to an Intel HEX file.

unsigned long **MHM_SaveRegister** (unsigned long `ulMHMHandle`,
char `*pcFilename`,
unsigned long `ulAdr`,
unsigned long `*pulNumberOfBytes`,
unsigned long `*pulData`)

Parameters :

`ulMHMHandle`	Handle of the library.
`pcFilename`	Absolute path of the file.
`ulAdr`	The start address.
`pulNumberOfBytes`	Pointer to the number of registers.
`pulData`	Pointer to an array containing the values.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_LoadRegister

Loads the specified address range from an Intel HEX file.

unsigned long **MHM_LoadRegister** (unsigned long `ulMHMHandle`,
char `*pcFilename`,
unsigned long `ulAdr`,
unsigned long `ulNumberOfBytes`,
unsigned long `*pulData`)

Parameters :

`ulMHMHandle`	Handle of the library.
`pcFilename`	Absolute path of the file.
`ulAdr`	The start address.
`ulNumberOfBytes`	Pointer to the number of registers.
`pulData`	Pointer to an array containing the values.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

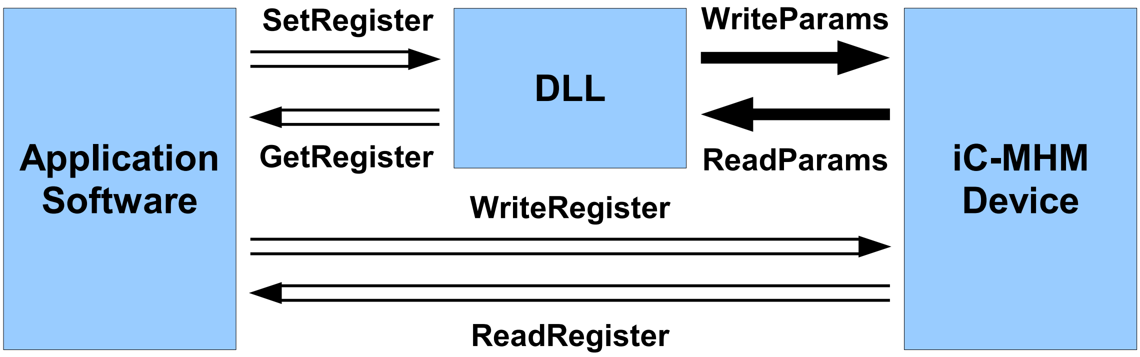

Figure 3: Illustration of the optional functions of the iC-MHM DLL.

MHM_GetInterfaceInfo

Gets additional information about the selected interface.

unsigned long
MHM_GetInterfaceInfo (unsigned long `ulMHMHandle`,
unsigned long `ulIndex`,
char `*pcInterfaceInfo`)

Parameters :

`ulMHMHandle` Handle of the device.

`ulIndex`

Value	Name	Description
0	eMHM_SERIAL_NUMBER	Gets the serial number (MB4U/MB5U only)

Table 7: Interface Info Index (MHM_InterfaceInfoEnum)

`pcInterfaceInfo` Interface information string.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_SetEdsParam

Sets an EDS parameter in the DLL.

unsigned long **MHM_SetEdsParam** (unsigned long `ulMHMHandle`,
unsigned long `ulParam`,
unsigned long `ulValue`)

Parameters :

`ulMHMHandle` Handle of the library.

`ulParam` The EDS configuration parameter. A list of parameters can be found in table 16, page 41.

`ulValue` The value of the EDS configuration parameter.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_GetEdsParam

Gets an EDS parameter from the DLL.

unsigned long **MHM_GetEdsParam** (unsigned long `ulMHMHandle`,
unsigned long `ulParam`,
unsigned long `*pulValue`)

Parameters :

`ulMHMHandle`	Handle of the library.
`ulParam`	The EDS configuration parameter. A list of parameters can be found in table 16, page 41.
`pulValue`	Pointer to the value of the EDS configuration parameter.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_ReadEdsParams

Reads EDS registers from the iC-MHM to the DLL.

unsigned long
MHM_ReadEdsParams (unsigned long `ulMHMHandle`)

Parameters :

`ulMHMHandle`	Handle of the library.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_UpdateEDS

Adjusts EDS parameters EDS_DLEN1, EDS_POLY1, EDS_CPOLY, EDS_CSTART, EDS_MT_LEN, EDS_CO_LEN, EDS_FI_LEN, EDS_MT_CNT, EDS_SIP_CNT, EDS_SIP_RES according to the iC parameter settings.

unsigned long
MHM_UpdateEDS (unsigned long `ulMHMHandle`)

Parameters :

`ulMHMHandle`	Handle of the library.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_SpiActivate

Sends the SPI command ACTIVATE to the iC-MHM.

unsigned long **MHM_SpiActivate** (unsigned long `ulMHMHandle`,
unsigned long `ulData`)

Parameters :

`ulMHMHandle`	Handle of the library.
`ulData`	Data byte (RACTIVE/PACTIVE).

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :

See the iC-MHM specification, chapter **SPI interface: general description** for more details.

MHM_WriteRegister_I2C

Writes an I2C device.

unsigned long
MHM_WriteRegister_I2C (unsigned long `ulMHMHandle`,
unsigned long `ulAdr`,
unsigned long `ulNumberOfBytes`,
unsigned long `ulI2CSlaveAdr`,
unsigned long `*pulData`)

Parameters :

`ulMHMHandle`	Handle of the library.
`ulAdr`	The address of the register.
`ulNumberOfBytes`	Number of registers to write.
`ulI2CSlaveAdr`	I2C slave address (valid range from addr. 0x40 to 0x7F).
`pulData`	Pointer to an array containing the values.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :
Register addresses from 0x40 up to 0x7F are not accessible via the I2C slave address 0x50!
Use function MHM_WriteRegister_I2C_Ex in combination with MHM_LoadRegisterEx to write data from an Intel HEX file!

MHM_WriteRegister_I2C_Ex

Writes an I2C device. In addition to the function **MHM_WriteRegister_I2C** only the registers marked by the valid array (address range = [0x00, ulArraySize - 1]) will be written.

```
unsigned long
MHM_WriteRegister_I2C_Ex     (unsigned long ulMHMHandle,
                              unsigned long ulArraySize,
                              unsigned long ulI2CSlaveAdr,
                              unsigned long *pulData,
                              unsigned long *pulValid)
```

Parameters :

ulMHMHandle	Handle of the library.
ulArraySize	Size of the data array to write.
ulI2CSlaveAdr	I2C slave address (valid range from 0x40 to 0x7F).
pulData	Pointer to an array containing the values.
pulValid	Pointer to an validation array. Value ’1’ data will be written, value ’0’ no writing.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :
Register addresses from 0x40 up to 0x7F are not accessible via the I2C slave address 0x50!
Use this function in combination with MHM_LoadRegisterEx to write data from an Intel HEX file! With this solution it is possible to configure an external I2C device. As an example one can use a configuration file generated by the iC-PV software.

MHM_ReadRegister_I2C

Reads data from the I2C device registers to an array (address range = [ulAdr, ulAdr + ulNumberOfBytes - 1]).

```
unsigned long
MHM_ReadRegister_I2C         (unsigned long ulMHMHandle,
                              unsigned long ulAdr,
                              unsigned long ulNumberOfBytes,
                              unsigned long ulI2CSlaveAdr,
                              unsigned long *pulData)
```

Parameters :

`ulMHMHandle`	Handle of the library.
`ulAdr`	The address of the register.
`ulNumberOfBytes`	Number of registers to read.
`ulI2CSlaveAdr`	I2C slave address (valid range from 0x40 to 0x7F).
`pulData`	Pointer to an array containing the values.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :
Register addresses from 0x40 up to 0x7F are not accessible via the I2C slave address 0x50!

MHM_LoadRegisterEx

Loads data from a file to an array (address range = [0x00, ulArraySize - 1]). The file format is Intel HEX. The existence of registers will be indicated by the validation array.

unsigned long
MHM_LoadRegisterEx (unsigned long `ulMHMHandle`,
char `*pcFilename`,
unsigned long `ulArraySize`,
unsigned long `*pulData`,
unsigned long `*pulValid`)

Parameters :

`ulMHMHandle`	Handle of the library.
`pcFilename`	Absolute path of the file.
`ulArraySize`	Size of the data array to write.
`pulData`	Pointer to an array containing the values.
`pulValid`	Pointer to an validation array. Value '1' indicates valid data, value '0' no valid data.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :
Use this function in combination with MHM_WriteRegister_I2C_Ex to write data from an Intel HEX file!

MHM_SaveRegisterEx

Saves data from an array to a file (address range = [ulAdr, ulAdr + pulNumberOfBytes -1]). The file format is Intel HEX.

```
unsigned long
MHM_SaveRegisterEx          (unsigned long ulMHMHandle,
                             char *pcFilename,
                             unsigned long ulAdr,
                             unsigned long ulNumberOfBytes,
                             unsigned long *pulData)
```

Parameters :

`ulMHMHandle`	Handle of the library.
`pcFilename`	Absolute path of the file.
`ulAdr`	Address of the first register.
`ulNumberOfBytes`	Number of registers to save.
`pulData`	Pointer to an array containing the values.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :
If the file already exists, the old file will be replaced.

MHM_FullCalibration

Measures data from the iC-MHM and calibrates the iC parameters VOSS, VOSC, GCC and HARMCAL. For calibration a constant RPM and a full revolution measured is needed. MHM_FullCalibration does not work with MB3U-BiSS. A MB3U-SPI, MB4U or MB5U is required. A MB4U or MB5U is highly recommended! During the calibration some iC-MHM parameters will be temporarily changed (see table 8, page 30). Figure 4 shows the procedure of this function.

```
unsigned long MHM_FullCalibration (unsigned long ulMHMHandle,
                                   unsigned long ulCycles)
```

Parameters :

`ulMHMHandle`	Handle of the library.
`ulCycles`	Number of Samples.

Return Value :

eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :

Use function **MHM_GetCalData** to get the raw data, which is used for the calibration (see page 33). Use function **MHM_GetCalParam** (see page 32) to get the relative changes of the calibration parameters.

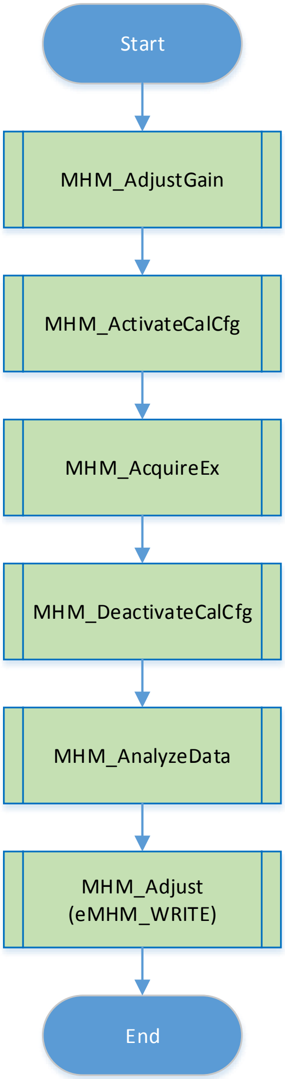

Figure 4: FullCalibration

MHM_FullAcquisition

Measures and analysis data from the iC-MHM. For the acquisition a constant RPM and a full revolution measured is needed. MHM_FullAcquisition does not work with MB3U-BiSS. A MB3U-SPI, MB4U or MB5U is required. A MB4U or MB5U is highly recommended! During the acquisition some iC-MHM parameters will be temporarily changed (see table 8, page 30). Figure 5 shows the procedure of this function.

unsigned long **MHM_FullAcquisition** (unsigned long `ulMHMHandle`,
unsigned long `ulCycles`)

Parameters :

`ulMHMHandle`	Handle of the library.
`ulCycles`	Number of Samples.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :
Use function **MHM_GetCalData** to get the raw data used for the acquisition (see page 33). Use function **MHM_GetCalParam** (see page 32) to get the relative changes of the calibration parameters.

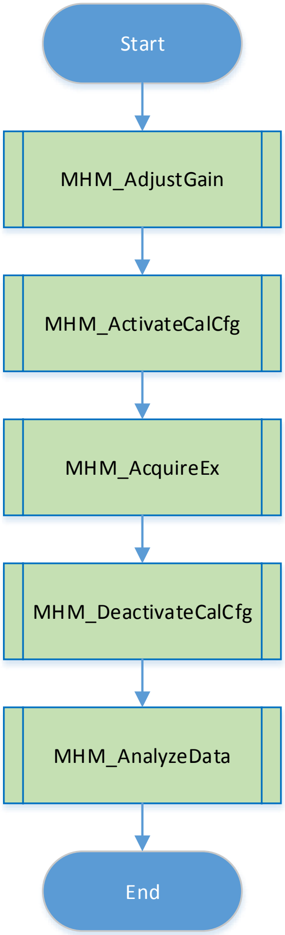

Figure 5: FullAcquisition

MHM_SetCalCfg

Sets the configuration of the calibration functions.

unsigned long **MHM_SetCalCfg** (unsigned long `ulMHMHandle`,
unsigned long `ulConfig`,
long `lValue`)

Parameters :

`ulMHMHandle`	Handle of the library.
`ulConfig`	Calibration configuration enumeration (see table 11, page 37).
`lValue`	Value of the configuration parameter.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_GetCalCfg

Gets the configuration of the calibration functions.

unsigned long **MHM_GetCalCfg** (unsigned long `ulMHMHandle`,
unsigned long `ulConfig`,
long `*plValue`)

Parameters :

`ulMHMHandle`	Handle of the library.
`ulConfig`	Calibration configuration enumeration (see table 11, page 37).
`plValue`	Pointer to the value of the configuration parameter.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_AdjustGain

Enables **eMHM_ENAC**. Reading the current gain and adjusting **GAINR** and **GAINF** .

unsigned long **MHM_AdjustGain** (unsigned long `ulMHMHandle`)

Parameters :

`ulMHMHandle`	Handle of the library.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_ActivateCalCfg

Saves the configuration of the MHM and changes it afterwards. This Configuration is needed for the Calibration. The saved Configuration can be restored with MHM_DeactivateCalCfg(see page 31). The parameter changes can be seen in table 8 page 30.

unsigned long **MHM_ActivateCalCfg** (unsigned long `ulMHMHandle`)

Parameters :

`ulMHMHandle`	Handle of the library.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Parameter	Value	Description
eMHM_ENAC	0x01	Enable amplitude control
eMHM_RESO_ST	0x04	Set ST resolution to 12 Bit
eMHM_RESO_MT	0x00	Deactivate MT during calibration
eMHM_SBL_MTI	0x00	No external multiturn
eMHM_HYS	0x00	No hysteresis
eMHM_TLF	0x07	Converter frequency to 1,75 MHz
eMHM_OFFS_ST	0x00	Deactivate ST offset
eMHM_AVGFILT	0x00	Deactivate filter
eMHM_DIR	0x00	Set rotation to CCW
eMHM_TEST	0x00	Deactivate testmodes
eMHM_DIR_IO2	0x00	Deactivate direction pin
eMHM_NTOA	0x00	Set timeout to adaptive

Table 8: MHM calibration configuration (parameter changes during MHM_ActivateCalCfg)

MHM_AcquireEx

Acquires raw data of the MHM. Only works correct when MHM_ActivateCalCfg (page 29) was executed before. Does not work with MB3U-BiSS. A MB3U-SPI, MB4U or MB5U are required. A MB4U or MB5U are highly recommended!

unsigned long **MHM_AcquireEx** (unsigned long `ulMHMHandle`
unsigned long `ulCycles`

Parameters :

`ulMHMHandle`	Handle of the library.
`ulCycles`	Number of Measurements.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :
Use function MHM_GetCalData to get the raw data of the last acquisition (see page 33).

MHM_DeactivateCalCfg

Restores the configuration saved in MHM_ActivateCalCfg(see page 29) .

unsigned long **MHM_DeactivateCalCfg** (unsigned long `ulMHMHandle`)

Parameters :

`ulMHMHandle`	Handle of the library.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_AnalyzeData

Analyzes the raw data of the last acquisition. DLL calibration parameter eMHM_REF_TYPE (see table 11) decides which kind of filter the analysis uses. MHM_AnalyzeData will only work properly, if the raw data of the iC-MHM was acquired with the configuration shown in table 8.

unsigned long **MHM_AnalyzeData** (unsigned long `ulMHMHandle`)

Parameters :

`ulMHMHandle`	Handle of the library.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :
Use function MHM_GetCalParam to get the relativ changes of the last analysation (see page 32).

MHM_Adjust

Changes the iC-MHM parameters according to the last result of MHM_AnalyzeData (page 31).

unsigned long **MHM_AnalogAdjust** (unsigned long `ulMHMHandle`,
unsigned long `ulWriteVerify`)

Parameters :

`ulMHMHandle`	Handle of the library.
`ulWriteVerify`	Write Verification Command (see table 6, page 17).

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

Hints :
Use function MHM_GetCalParam to get the relative changes which will be used for the adjustment (see page 32).

MHM_GetCalParam

Reads the relativ calibration parameters from the last analysation.

```
unsigned long MHM_GetCalParam    (unsigned long ulMHMHandle,
                                   unsigned long ulParam,
                                   long *plData)
```

Parameters :

`ulMHMHandle`	Handle of the library.
`ulParam`	Calibration parameter enumeration (see table 14, page 37).
`plData`	Pointer to the Value of the choosen parameter.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_SetCalData

Sets the external data for the analog calibration.

```
unsigned long MHM_SetCalData     (unsigned long ulMHMHandle,
                                   unsigned long ulParam,
                                   unsigned long ulNumberOfBytes,
                                   long *plData)
```

Parameters :

`ulMHMHandle`	Handle of the library.
`ulParam`	Calibration data enumeration (see table 13, page 37).
`ulNumberOfBytes`	Number of samples to write.
`plData`	Pointer to the array of the chosen data.

Return Value :
eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

MHM_GetCalData

Gets the acquisition data of the last acquisition from the DLL. The Resolution of one revolution is 12Bit.

```
unsigned long MHM_GetCalData    (unsigned long ulMHMHandle,
                                 unsigned long ulParam,
                                 unsigned long *pulNumberOfBytes,
                                 long *plData)
```

Parameters :

ulMHMHandle	Handle of the library.
ulParam	Calibration data enumeration (see table 13, page 37).
pulNumberOfBytes	Number of samples to read.
plData	Pointer to an array with the selected acquisition data.

Return Value :

eMHM_OK if successful, otherwise the return value is an error code (see table 15, page 39).

PARAMETER AND ERROR CODING

iC-MHM Parameters

Value	Parameter	Description (see latest data sheet)
0	eMHM_AVGFILT	
1	eMHM_TLF	
2	eMHM_DIR	
3	eMHM_HYS	
4	eMHM_RESO_MT	
5	eMHM_RESO_ST	
6	eMHM_CF_MTI	
7	eMHM_SBL_MTI	
8	eMHM_EBL_MTI	
9	eMHM_GET_MTI	
10	eMHM_CFG_IOP	
11	eMHM_MT12	
12	eMHM_RTX_MODE	
13	eMHM_BIN_SSI	
14	eMHM_EXT_SSI	
15	eMHM_ENSSI	
16	eMHM_VOSS	
17	eMHM_VOSC	
18	eMHM_CIBM	
19	eMHM_ENF	
20	eMHM_TEST	
21	eMHM_GAINF	
22	eMHM_GAINR	
23	eMHM_GCC	
24	eMHM_ENAC	
25	eMHM_CRCS	
26	eMHM_ENLC	
27	eMHM_REGPROT	
28	eMHM_INSPROT	
29	eMHM_PRES_IO1	
30	eMHM_DIR_IO2	
31	eMHM_ENCMD3	
32	eMHM_ENCMD2	
33	eMHM_CRC_CFG	
34	eMHM_OFFS_MT	
35	eMHM_OFFS_ST	
36	eMHM_CRC_OFFS	
37	eMHM_PSET_MT	
38	eMHM_PSET_ST	
39	eMHM_CRC_PSET	
40	eMHM_ERR_CFG	
41	eMHM_ERR_OFFS	

Value	Parameter	Description (see latest data sheet)
42	eMHM_ERR_POS	
43	eMHM_ERR_EXT	
44	eMHM_ERR_AMIN	
45	eMHM_ERR_AMAX	
46	eMHM_ERR_MTI	
47	eMHM_ERR_MT	
48	eMHM_GAIN	
49	eMHM_EDSBANK	
50	eMHM_PRO_ID	
51	eMHM_DEV_ID0	
52	eMHM_DEV_ID1	
53	eMHM_DEV_ID2	
54	eMHM_DEV_ID3	
55	eMHM_DEV_ID4	
56	eMHM_DEV_ID5	
57	eMHM_MFG_ID0	
58	eMHM_MFG_ID1	
59	eMHM_HARMCAL	
60	eMHM_DISBISS	
61	eMHM_NTOA	
62	eMHM_ENCMD01	
63	eMHM_CHIP_REL	

Table 9: iC-MHM parameters (MHM_ParamEnum)

DLL Parameters

Value	Parameter	Description
0	eMHM_CHIPVERSION	Reserved
1	eMHM_USEUSB	USB interface available (read only)
2	eMHM_USELPT	LPT interface available (read only)
3	eMHM_MASTERVER	BiSS master version (read only)
4	eMHM_MASTERREV	BiSS master revision (read only)
5	eMHM_SLAVE_ID	BiSS ID of the connected MHM
6	eMHM_SLAVECOUNT	Number of the slave in the BiSS chain
7	eMHM_FREQ_SCD	Sensor data frequency (see table 18)
8	eMHM_FREQ_AGS	AutoGetSens frequency (see table 19)
9	eMHM_FREQ_SPI	SPI frequency (12 MHz / ((MHM_FREQ_SPI+1)*2)
10	eMHM_CLKENI	Enable internal 20 MHz oscillator (enable: 1, disable: 0, default: 1; see BiSS master specification for details)
11	eMHM_BP	BiSS Profile (valid values 1, 3: see EDS specification for details)
12	eMHM_MT_TYPE	Connected multiturn iC (valid values 0: iC-MHM only, 1: iC-PV, 2: iC-MV, 3: iC-PVL)
13	eMHM_RESERVED0	Moved config parameter to eMHM_CalCfgEnum (37).
14	eMHM_RESERVED1	Moved config parameter to eMHM_CalCfgEnum (37).
15	eMHM_RESERVED2	Moved config parameter to eMHM_CalCfgEnum (37).
16	eMHM_RESERVED3	Moved config parameter to eMHM_CalCfgEnum (37).

Value	Parameter	Description
17	eMHM_FREQ_SSI	SSI frequency (see table 18)
18	eMHM_READ_STATUS_ENABLE	Enable cyclic read for MHM_ReadSens: status register (0x70). BiSS mode only!
19	eMHM_READ_PORTS_ENABLE	Enable cyclic read for MHM_ReadSens: ports register (0x71). BiSS mode only!
20	eMHM_READ_GAIN_ENABLE	Enable cyclic read for MHM_ReadSens: gain register (0x72). BiSS mode only!
21	eMHM_USB_PERFORMANCE	Indicator value to the current USB Performance. Will be measured once during the MHM_Initialize.

22	eMHM_ENABLE_TTL	This parameter is only available for MB4U / MB5U and depends on MHM_DISABLE_AUTOMATIC_TTL_SWITCH
		MHM_DISABLE_AUTOMATIC_TTL_SWITCH=0: Indicator value if TTL bit is enabled.
		MHM_DISABLE_AUTOMATIC_TTL_SWITCH=1: Set to ’1’ to set TTL mode of the PC adapter, set to ’0’ to disable TTL mode.

23	eMHM_UPDATE_BISSID_ENABLE	Enable automatic BiSS-ID configuration (0x7C..0x7D) according to iC configuration parameters (e.g. RESO_ST).
24	eMHM_DISABLE_AUTOMATIC_TTL_SWITCH	Disable automatic TTL configuration of MB4U / MB5U.

Table 10: DLL parameters (MHM_ConfigDataEnum)

Calibration Parameters (DLL)

Value	Parameter	Description
0	eMHM_CAL_FREQ_SPI	Analog calibration: SPI frequency (12 MHz / ((MHM_FREQ_SPI+1)*2)
1	eMHM_CAL_FREQ_SCD	Analog calibration: sensor data frequency (see table 18)
2	eMHM_CAL_FREQ_AGS	Analog calibration: AutoGetSens frequency (see table 19)
3	eMHM_CAL_REF_TYPE	Modes for calibration reference (see table 12)
4	eMHM_CAL_REF_RES	Resolution of external reference
5	eMHM_CAL_REF_PER	Periods per revolution of external reference
6	eMHM_CAL_QUALITY_CHECK	Analyses the quality of raw analog signals by checking the velocity stability. ('1' = enable (default), '0' = disable).

Table 11: Calibration configuration enumeration (MHM_CalCfgEnum)

Value	Parameter	Description
0	eMHM_CONST_VELOCITY	Constant velocity (recommended)
1	eMHM_FIRST_ORDER	Filter 1st order
2	eMHM_SECOND_ORDER	Filter 2nd order
3	eMHM_MOVING_MEAN	Filter moving mean
4	eMHM_MOVING_ACCEL	Filter for acceleration
5	eMHM_REF_EXTERNAL	External reference, no filter necessary/used

Table 12: Reference configuration for calibration parameters (MHM_ReferenceEnum)

Value	Parameter	Description
0	eMHM_DUTRAW	Raw signals of the DUT (12 Bit and eMHM_DIR == 0)
1	eMHM_DUTCONT	Continuous signals of the DUT
2	eMHM_REFCONT	Continuous signals of the reference(generated)
3	eMHM_DUTERR	Nonlinearity of the DUT
4	eMHM_ADJERR	Adjustable nonlinearity of the DUT
5	eMHM_REFRAW	Raw signals of the Reference (external)

Table 13: Calibration data enumeration (MHM_CalDataEnum)

Value	Parameter	Description
0	eMHM_CAL_SINOFF	Relative correction parameter for sine offset
1	eMHM_CAL_COSOFF	Relative correction parameter for cosine offset
2	eMHM_CAL_GAINCOSCORR	Relative correction parameter for cosine gain correction
3	eMHM_CAL_HARM4	Relative correction parameter for harmonic calibration
4	eMHM_CAL_RPM	Rotation Speed with rotation direction (+ CW, - CCW)

Table 14: Calibration parameter enumeration (MHM_CalParamEnum)

Error Description

Value	Error	Description (Solution)
0	eMHM_OK	Function call successful.
1	eMHM_INVALID_HANDLE	Use the handle returned by the **MHM_Open** function.
2	eMHM_INTERFACEDRIVER_NOT_FOUND	Install FTDI interface drivers.
3	eMHM_INTERFACE_NOT_FOUND	Check wire connections and power supply. The interface driver used may have the wrong version.
4	eMHM_INVALID_INTERFACE	Interface not specified. See Table 1, page 6.
5	eMHM_INVALID_PARAMETER	Parameter not specified. See Table 9, page 35.
6	eMHM_INVALID_ADDRESS	Address out of range.
7	eMHM_INVALID_VALUE	Value out of range.
8	eMHM_USB_ERROR	Check USB to interface connection.
9	eMHM_FILE_NOT_FOUND	Check if file exists.
10	eMHM_INVALID_FILE	Check file format.
11	eMHM_VERIFY_FAILED	Check device connection.
12	eMHM_MASTERCOMM_FAILED	Check device connection of the selected interface.
13	eMHM_BISSCOMM_FAILED	Check adapter to iC-MHM connection. Check power supply.
14	eMHM_SPICOMM_FAILED	Check adapter to iC-MHM connection. Check power supply.
15	eMHM_USB_HIGHSPEED_WARNING	High speed USB device plugged into non high speed USB hub.
16	eMHM_INVALID_BISSMASTER	Connect a BiSS adapter with iC-MB3Z or later.
17	eMHM_NO_INTERFACE_SELECTED	Select an interface before using this function.
18	eMHM_READPARAM_SSI	Read access to registers is not possible in SSI mode (write access is possible).
19	eMHM_SPI_ERROR	Sensor data were invalid on readout.
20	eMHM_SPI_DISMISS	Address refused.
21	eMHM_SPI_FAIL	Data request failed.
22	eMHM_SPI_BUSY_TIMEOUT	Slave is busy with a request.
23	eMHM_FILE_ACCESS_DENIED	Insufficient permission to access file.
24	eMHM_INVALID_CONFIGURATION	iC-MHM signaled a configuration error. Use MHM_WriteParams to write a valid configuration.
25	eMHM_INVALID_EDS	EDS contains invalid values.
26	eMHM_INVALID_EDS_CHECKSUM	EDS checksum error.
27	eMHM_EDS_CORRECTED	Parameters MHM_EDS_DLEN1 / MHM_EDS_CPOLY1 adapted to iC-MHM configuration.
28	eMHM_CORRECTED_INVALID_VALUE	Corrected invalid RAM value according to the datasheet. Call **MHM_GetCorrectedRegisters** to receive an array of affected registers.
29	eMHM_BISS_REGERROR	Invalid address and/or access denied.
30	eMHM_EDS_UNDEFINED	EDS is not used in current EEPROM configuration.
31	eMHM_FILESIZE_WARNING	Unexpected file size detected.
32	eMHM_SLOW_ROTATION	Calibration error: didn't measure at least one complete revolution because of a too slow rotation.
33	eMHM_FAST_ROTATION	Calibration error: too few measurements per revolution because of a too fast rotation.
34	eMHM_GAIN_LIMIT	Calibration error: autogain is too low or too high.
35	eMHM_ACQUISITION_FAILED	Calibration error: a sensor error occurred during the acquisition.
36	eMHM_USB_DATA_LOSS	Reduce AutoGetSens frequency (eMHM_CAL_FREQ_AGS).
37	eMHM_INTERNAL_CALIB_ERROR	Internal error during calibration occurred.

Value	Error	Description (Solution)
38	eMHM_OFFSET_LIMIT	Calibration error: relative change of sine or cosine offset parameter out of range.
39	eMHM_CALIBRATION_FAILED	Calibration error: an error occurred during the calibration procedure.
40	eMHM_I2C_COMM_FAILED	The communication with the chosen I2C device failed.
41	eMHM_GAINCOSCORR_LIMIT	Calibration error: relative change of the parameter GCC out of range.
42	eMHM_HARMCAL_LIMIT	Calibration error: relative change of the parameter HARMCAL is too low or too high.
43	eMHM_BAD_CAL_DATA	Poor quality of acquired raw analog signals. A more constant velocity is needed for calibration.

Table 15: Error description (MHM_ErrorEnum)

EDS Description

Value	Parameter	Description (see latest eds data sheet)
0	eMHM_EDS_EDS_VER	
1	eMHM_EDS_EDS_LEN	
2	eMHM_EDS_USR_STA	
3	eMHM_EDS_USR_END	
4	eMHM_EDS_TMA	
5	eMHM_EDS_TO_MIN	
6	eMHM_EDS_TO_MAX	
7	eMHM_EDS_TOS_MIN	
8	eMHM_EDS_TOS_MAX	
9	eMHM_EDS_TCLK_MIN	
10	eMHM_EDS_TCLK_MAX	
11	eMHM_EDS_TCYC	
12	eMHM_EDS_TBUSY_S	
13	eMHM_EDS_BUSY_S	
14	eMHM_EDS_PON_DLY	
15	eMHM_EDS_DC_NUM	
16	eMHM_EDS_SL_NUM	
17	eMHM_EDS_SL_OFF	
18	eMHM_EDS_BANK1	
19	eMHM_EDS_DLEN1	
20	eMHM_EDS_FORMAT1_STOP	
21	eMHM_EDS_FORMAT1_ALIGN	
22	eMHM_EDS_FORMAT1_TYPE	
23	eMHM_EDS_CPOLY1	
24	eMHM_EDS_BANK2	
25	eMHM_EDS_DLEN2	
26	eMHM_EDS_FORMAT2_STOP	
27	eMHM_EDS_FORMAT2_ALIGN	
28	eMHM_EDS_FORMAT2_TYPE	
29	eMHM_EDS_CPOLY2	

Value	Parameter	Description (see latest eds data sheet)
30	eMHM_EDS_BANK3	
31	eMHM_EDS_DLEN3	
32	eMHM_EDS_FORMAT3_STOP	
33	eMHM_EDS_FORMAT3_ALIGN	
34	eMHM_EDS_FORMAT3_TYPE	
35	eMHM_EDS_CPOLY3	
36	eMHM_EDS_BANK4	
37	eMHM_EDS_DLEN4	
38	eMHM_EDS_FORMAT4_STOP	
39	eMHM_EDS_FORMAT4_ALIGN	
40	eMHM_EDS_FORMAT4_TYPE	
41	eMHM_EDS_CPOLY4	
42	eMHM_EDS_BANK5	
43	eMHM_EDS_DLEN5	
44	eMHM_EDS_FORMAT5_STOP	
45	eMHM_EDS_FORMAT5_ALIGN	
46	eMHM_EDS_FORMAT5_TYPE	
47	eMHM_EDS_CPOLY5	
48	eMHM_EDS_BANK6	
49	eMHM_EDS_DLEN6	
50	eMHM_EDS_FORMAT6_STOP	
51	eMHM_EDS_FORMAT6_ALIGN	
52	eMHM_EDS_FORMAT6_TYPE	
53	eMHM_EDS_CPOLY6	
54	eMHM_EDS_BANK7	
55	eMHM_EDS_DLEN7	
56	eMHM_EDS_FORMAT7_STOP	
57	eMHM_EDS_FORMAT7_ALIGN	
58	eMHM_EDS_FORMAT7_TYPE	
59	eMHM_EDS_CPOLY7	
60	eMHM_EDS_BANK8	
61	eMHM_EDS_DLEN8	
62	eMHM_EDS_FORMAT8_STOP	
63	eMHM_EDS_FORMAT8_ALIGN	
64	eMHM_EDS_FORMAT8_TYPE	
65	eMHM_EDS_CPOLY8	
66	eMHM_EDS_BC_OFF	
67	eMHM_EDS_BP_VER	
68	eMHM_EDS_BP_LEN	
69	eMHM_EDS_BP_ID	
70	eMHM_EDS_FB1	
71	eMHM_EDS_FB2	
72	eMHM_EDS_PON_PDL	
73	eMHM_EDS_EN_TYP	
74	eMHM_EDS_POS_NUM	
75	eMHM_EDS_MT_LEN	
76	eMHM_EDS_MT_FMT	
77	eMHM_EDS_CO_LEN	
78	eMHM_EDS_CO_FMT	

Value	Parameter	Description (see latest eds data sheet)
79	eMHM_EDS_FI_LEN	
80	eMHM_EDS_FI_FMT	
81	eMHM_EDS_MT_CNT	
82	eMHM_EDS_SIP_CNT	
83	eMHM_EDS_SIP_RES	
84	eMHM_EDS_CPOLY	
85	eMHM_EDS_CSTART	
86	eMHM_EDS_ABS_ACU	
87	eMHM_EDS_REL_ACU	
88	eMHM_EDS_SPD_ACU	
89	eMHM_EDS_HYST	
90	eMHM_EDS_SPD_MAX	
91	eMHM_EDS_ACC_MAX	
92	eMHM_EDS_TMP_MIN	
93	eMHM_EDS_TMP_MAX	
94	eMHM_EDS_VLT_MIN	
95	eMHM_EDS_VLT_MAX	
96	eMHM_EDS_CUR_MAX	

Table 16: EDS parameters (MHM_EdsParamEnum)

Example 1 - Getting Started

Listing 1: Source code example (C++)

```
#include "MHM_1SL_interface.h"
#include <iostream>
using namespace std;
unsigned long ulMHMHandle;

int main()
{
 unsigned long ulaVersion[2];
 unsigned long ulError;

 //Get the version number of the DLL
 MHM_GetDLLVersion(ulaVersion);
 cout << "MHM_1SL_interface.dll␣Version:␣" << ulaVersion[1] << "." << ulaVersion[0] << endl << endl;

 //Load the library and get a handle that will be used for subsequent accesses.
 ulError = MHM_Open(&ulMHMHandle);


 //Set the DLL properties (BiSS Master Clock Frequency for Single Cycle Data)
 //p.e. FREQ_SCD = 2 MHz @ MB4U
 ulError = MHM_SetConfig(ulMHMHandle, eMHM_FREQ_SCD, 4);

 //Connect the interface MB4U
 ulError = MHM_SetInterface(ulMHMHandle, eMHM_MB4U, "");

 if (ulError)
 {
  switch (ulError){
   case eMHM_INTERFACEDRIVER_NOT_FOUND:
    cout << "MHM_SetInterface␣failed␣(eMHM_INTERFACEDRIVER_NOT_FOUND):" << endl
    << "Install␣FTDI␣Interface␣Drivers" << endl; break;
   case eMHM_INTERFACE_NOT_FOUND:
    cout << "MHM_SetInterface␣failed␣(eMHM_INTERFACE_NOT_FOUND):␣" << endl
    << "Check␣wire␣connections␣and␣power␣supply" << endl; break;
   default:
    cout << "MHM_SetInterface␣failed␣(errorcode:␣" << ulError << ")"
    << endl; break;
  }
 }
 else
 {
  cout << "Interface␣MB4U␣opened" << endl << endl;

  ulError = MHM_Initialize(ulMHMHandle);

  //Read registers from the iC-MHM to the DLL
  ulError = MHM_ReadParams(ulMHMHandle);
  if (ulError)
   cout << "Could␣not␣read␣Params!" << endl;
  else
  {

   //then read the position values from the iC-MHM
   MHM_ReadSensStruct ReadSensData;
   ulError = MHM_ReadSens(ulMHMHandle, &ReadSensData);

   //Set some parameters to the DLL and write the corresponding register to the iC-MHM immediately
   ulError = MHM_SetParam(ulMHMHandle, eMHM_OFFS_MT, 1, eMHM_WRITE);
   ulError = MHM_SetParam(ulMHMHandle, eMHM_GAINF, 0, eMHM_WRITE);

   //Get all iC-MHM parameter from the DLL
   unsigned long ulParamValue;
   for (int iCnt=0; iCnt<=62; iCnt++)
   {
    ulError = MHM_GetParam(ulMHMHandle, iCnt, &ulParamValue);
    cout << "Param(" << dec << iCnt << "):␣0x" << hex << ulParamValue << endl;
   }

  }

 }

 //Release the handle and terminate established connections
 MHM_Close(ulMHMHandle);
}
```

Example 2 - Automatic Calibration

This example explains the basic steps needed for calibration for constant rotative applications.

Listing 2: Source code example (C++) for automatic calibration functions

```
#include "MHM_1SL_interface.h"
#include <iostream>
using namespace std;
unsigned long ulMHMHandle;

int main()
{
 unsigned long ulaVersion[2];
 unsigned long ulError;

 //Get the version number of the DLL
 MHM_GetDLLVersion(ulaVersion);
 cout << "MHM_1SL_interface.dll␣Version:␣" << ulaVersion[1] << "." << ulaVersion[0] << endl << endl;

 //Load the library and get a handle that will be used for subsequent accesses.
 ulError = MHM_Open(&ulMHMHandle);

 //Connect the interface MB4U
 ulError = MHM_SetInterface(ulMHMHandle, eMHM_MB4U, "");

 if (ulError)
 {
  switch (ulError){
   case eMHM_INTERFACEDRIVER_NOT_FOUND:
    cout << "MHM_SetInterface␣failed␣(eMHM_INTERFACEDRIVER_NOT_FOUND):" << endl
    << "Install␣FTDI␣Interface␣Drivers" << endl; break;
   case eMHM_INTERFACE_NOT_FOUND:
    cout << "MHM_SetInterface␣failed␣(eMHM_INTERFACE_NOT_FOUND):␣" << endl
    << "Check␣wire␣connections␣and␣power␣supply" << endl; break;
   default:
    cout << "MHM_SetInterface␣failed␣(errorcode:␣" << ulError << ")"
    << endl; break;
  }
 }
 else
 {
  cout << "Interface␣MB4U␣opened" << endl << endl;

  // Initialize
  ulError = MHM_Initialize(ulMHMHandle);

  //afterwards read all registers from the iC-MH to the DLL
  ulError = MHM_ReadParams(ulMHMHandle);
  //Setting Calib parameters
  ulError = MHM_SetCalCfg(ulMHMHandle, eMHM_CAL_REF_TYPE, eMHM_CONST_VELOCITY); //Filter type of the
      analysis, use MHM_ReferenceEnum
  ulError = MHM_SetCalCfg(ulMHMHandle, eMHM_CAL_FREQ_SCD, 0x04); //2Mhz Clock
  ulError = MHM_SetCalCfg(ulMHMHandle, eMHM_CAL_FREQ_AGS, 0x63); //100us Frame repetition rate

  //One full Acquisition
  ulError = MHM_FullAcquisition(ulMHMHandle, 2000);

  long lSin_Off, lCos_Off, lGainC, lRPM, lHarm4;
  if (ulError == eMHM_OK)
  {
   ulError = MHM_GetCalParam(ulMHMHandle, eMHM_CAL_SINOFF, &lSin_Off);
   ulError = MHM_GetCalParam(ulMHMHandle, eMHM_CAL_COSOFF, &lCos_Off);
   ulError = MHM_GetCalParam(ulMHMHandle, eMHM_CAL_GAINCOSCORR, &lGainC);
   ulError = MHM_GetCalParam(ulMHMHandle, eMHM_CAL_HARM4, &lHarm4);
   ulError = MHM_GetCalParam(ulMHMHandle, eMHM_CAL_RPM, &lRPM);
   cout << "First␣Acquisition"<< endl;

   cout << "SinOff:␣"<< double(lSin_Off) << "\t|␣Cos_Off:␣"<< double(lCos_Off) << "\t|␣GainC:␣"<<
       double(lGainC) << "\t|␣Harm4:␣"<<double(lHarm4) << "\t|␣RPM:␣"<<double(lRPM) << endl << endl;
  }

  //full Calibration
  unsigned int uiIteration=4;
  for (unsigned int uiRun =0 ; uiRun <uiIteration; uiRun++){

   ulError = MHM_FullCalibration(ulMHMHandle, 2000);
   if (ulError != eMHM_OK){
    cout << "Calibration␣failed";
    uiRun = uiIteration;
   }
   else
   {
    ulError = MHM_GetCalParam(ulMHMHandle, eMHM_CAL_SINOFF, &lSin_Off);
    ulError = MHM_GetCalParam(ulMHMHandle, eMHM_CAL_COSOFF, &lCos_Off);
    ulError = MHM_GetCalParam(ulMHMHandle, eMHM_CAL_GAINCOSCORR, &lGainC);
    ulError = MHM_GetCalParam(ulMHMHandle, eMHM_CAL_HARM4, &lHarm4);
    ulError = MHM_GetCalParam(ulMHMHandle, eMHM_CAL_RPM, &lRPM);
    cout<<"------------------------------------------------------------------"<< endl;
    cout << "Calibration␣Run␣"<<double (uiRun+1)<< endl;
    cout << "SinOff:␣"<< double(lSin_Off) << "\t|␣Cos_Off:␣"<< double(lCos_Off) << "\t|␣GainC:␣"<<
        double(lGainC) << "\t|␣Harm4:␣"<<double(lHarm4) << "\t|␣RPM:␣"<<double(lRPM) << endl << endl;
    //If the changes are not bigger then 1Digit, the Calibration was succesful and stable
   }

  }
 }

 //Release the handle and terminate established connections
 MHM_Close(ulMHMHandle);
}
```

Example 3 - Calibration Step by Step

This example explains the calibration sequence in more detail.

Listing 3: Source code example (C++) for calibration functions step by step

```
#include "MHM_1SL_interface.h"
#include <iostream>
using namespace std;
unsigned long ulMHMHandle;

unsigned long Measure( unsigned long ulPoints );

int main()
{
```

```
unsigned long ulaVersion[2];
unsigned long ulError;

MHM_GetDLLVersion(ulaVersion);
//Get the version number of the DLL
cout << "MHM_1SL_interface.dll␣Version:␣" << ulaVersion[1] << "." << ulaVersion[0] << endl << endl;

//Load the library and get a handle that will be used for subsequent accesses.
ulError = MHM_Open(&ulMHMHandle);

//Connect the interface MB4U, chip interface must be in BiSS Mode.
ulError = MHM_SetInterface(ulMHMHandle, eMHM_MB4U, "");
if (ulError)
 cout << "MHM_SetInterface␣failed␣(errorcode:␣" << ulError << ")" << endl;
else
{
 cout << "Interface␣opened" << endl << endl;
 // Initialize
 ulError = MHM_Initialize(ulMHMHandle);
 //Read Ram of MHM
 ulError = MHM_ReadParams(ulMHMHandle);
 // Set ags time to approx 100us
 ulError = MHM_SetCalCfg(ulMHMHandle, eMHM_CAL_FREQ_AGS, 0x63);
 // Set Frequence to 2 MHz
 ulError = MHM_SetCalCfg(ulMHMHandle, eMHM_CAL_FREQ_SCD, 0x04);
 // Set refeference to Moving Acceleration
 ulError = MHM_SetCalCfg(ulMHMHandle, eMHM_CAL_REF_TYPE, eMHM_CONST_VELOCITY); //Filter type of the
     analysis, use MHM_ReferenceEnum

 //-------------------------------------------------------------
 //----------------ANALOG CALIBRATION Step by Step--------------
 //-------------------------------------------------------------

 // FullMeasurement
 unsigned long ulCycles = 2000;
 long lSinOffset, lCosOffset, lGainCorr, lHarm4, lRPM;
 unsigned long ulErrortemp = eMHM_OK;
 unsigned int uiIterations = 3;

 cout << endl << "----------------------------------------------------------" << endl;
 cout <<   "---------------------Analog␣Calibration-------------------" << endl;
 cout  <<   "----------------------------------------------------------" << endl;

 //Depending on the signal quality, the reference quality and the filter settings, the Calibration
     will need some iterations to find the best Configuration for the MHM
 for (unsigned int uiRun=0; uiRun< uiIterations; uiRun++)
 {

  //activate Acquisition Config
  ulError = MHM_ActivateCalCfg(ulMHMHandle);
  if(ulError == 0){
   //Acquire raw position data
   ulErrortemp = MHM_AcquireEx(ulMHMHandle , ulCycles);

   //restore to original config
   ulError = MHM_DeactivateCalCfg(ulMHMHandle);
  }

  if((ulError != 0)||(ulErrortemp != 0))
   cout<< "␣Acquisition␣Failed␣!"<< endl;
  else
  {
   //Analyze analog signals
   ulError = MHM_AnalyzeData(ulMHMHandle);

   if(ulError)
    cout<< "␣Analog␣analysis␣failed␣!"<< endl;
   else
   {
    //Show Results
    //Get calib parameter
```

```
     MHM_GetCalParam(ulMHMHandle, eMHM_CAL_SINOFF, &lSinOffset);
     MHM_GetCalParam(ulMHMHandle, eMHM_CAL_COSOFF, &lCosOffset);
     MHM_GetCalParam(ulMHMHandle, eMHM_CAL_GAINCOSCORR,  &lGainCorr);
     MHM_GetCalParam(ulMHMHandle, eMHM_CAL_HARM4,  &lHarm4);
     MHM_GetCalParam(ulMHMHandle, eMHM_CAL_RPM,  &lRPM);

     cout << "␣Acquisition␣Run␣" << uiRun + 1 << endl;
     cout << "SinOff:␣\t"<< lSinOffset << endl;
     cout << "CosOff:␣\t"<< lCosOffset << endl;
     cout << "Gain:␣␣␣\t"<< lGainCorr <<  endl;
     cout << "Harm4:␣␣\t"<< lHarm4 <<  endl;
     cout << "RPM:␣\t" << lRPM <<   endl;
     cout << "----------------------------------------------------------" <<endl;
    }
   }

   if (uiRun < uiIterations-1)
   {
    //Adjust MHM
    ulError = MHM_Adjust(ulMHMHandle, eMHM_WRITE);
    cout << endl << "----------------------------------------------------------" <<endl;
    if(ulError)
     cout << "␣Adjustment␣Failed!␣" << endl ;
    else
     cout << "␣MHM␣Adjusted␣" << endl ;
    cout << endl << "----------------------------------------------------------" <<endl;
   }
  }
 }
 MHM_Close(ulMHMHandle);
}
```

Example 4 - Calibration with external data via BiSS DLL

This example explains two ways to calibrate via additional software (here using the BiSS DLL):

1. Calibration: e.g. needed if not acquiring data via the iC-MHM DLL.

2. Calibration with a reference encoder.

Listing 4: Source code example (C++) for calibration functions with external data via BiSS DLL

```
#include "MHM_1SL_interface.h"
#include "BISS1SL_interface.h"
#include <iostream>
#include <math.h>
#include <fstream>
#include <sstream>
#include <iomanip>
#define CYCLECNT 5000
using namespace std;
unsigned long ulMHMHandle;


//Using the BiSS-DLL for acquiring data of an application with a DUT (iC-MHM) and a reference encoder
    (iC-LGC) within a BiSS-Chain.
//Because the iC-MHM cannot be configured by the DLL if you use external Data, the iC-MHM
    configuration has to be changed by the user
//The needed configuration can be found in iC-MHM DLL LIBRARY DESCRIPTION (see description of function
     MHM_ActivateCalCfg)
unsigned long Measure(unsigned long ulNumberOfCycles, unsigned long *pulaDUT, unsigned long *
    pulaReference  )
{
 unsigned long ulaVersion[2];
 unsigned long ulError = 0;
 unsigned long ulBissHandle;
```

```
//Get the version number of the DLL
BISS_GetDLLVersion(ulaVersion);
cout << "BISS1SL_interface.dll␣Version:␣" << ulaVersion[1] << "." << ulaVersion[0] << endl << endl;

//Load the library and get a handle that will be used for subsequent accesses.
ulError = BISS_Open(&ulBissHandle);

//Set the BiSS Master properties
ulError = BISS_SetComParam(ulBissHandle, eBISS_FREQ_SCD, 0x04);  //Single Cycle Data communication
    clockrate 2MHz
ulError = BISS_SetComParam(ulBissHandle, eBISS_FREQ_AGS, 0x63);  //Auto Get Sens 100us

//Connect the interface MB4U
ulError = BISS_SetInterface(ulBissHandle, eBISS_MB4U, "");
//Set protocol to BiSS-C
ulError = BISS_SetComParam(ulBissHandle, eBISS_PROTOCOL, eBISS_PROTOCOL_C);

if (ulError)
{
 switch (ulError){
  case eBISS_INTERFACEDRIVER_NOT_FOUND:
   cout << "BISS_SetInterface␣failed␣(eBISS_INTERFACEDRIVER_NOT_FOUND):" << endl
   << "Install␣FTDI␣Interface␣Drivers" << endl; break;
  case eBISS_INTERFACE_NOT_FOUND:
   cout << "BISS_SetInterface␣failed␣(eBISS_INTERFACE_NOT_FOUND):␣" << endl
   << "Check␣wire␣connections␣and␣power␣supply" << endl; break;
  default:
   cout << "BISS_SetInterface␣failed␣(errorcode:␣" << ulError << ")"
   << endl; break;
 }
}
else
{
 cout << "Interface␣MB4U␣opened" << endl << endl;

 //Define DUT properties
 // BISS Data Channel 1 = iC-MHM. 12 Bit Singleturn + 1 Error + 1 Warning + 6 CRC = 20 Bit (Sign of
     life counter deactivated)
 // BISS Data Channel 2 = iC-LGC (used as reference with 16 Bit). 16 Bit Singleturn + 1 Error + 1
     Warning + 6 CRC = 24 Bit
 unsigned long ulSCDLength[2] = {20,24}; //SCD length of slaves
 //MHM and LGC 0x43 CRC poly inverted
 unsigned long ulCRCpoly[2] = {0x43, 0x43};
 unsigned long ulCRCinvert[2] = {1,1};

 //Set the properties of the connected slaves
 for (int iDataCH = 0; iDataCH < 2; iDataCH++)
 {
  ulError = BISS_SetFrameParam(ulBissHandle, iDataCH, eBISS_SCD_CRC_POLY, ulCRCpoly[iDataCH]);
  ulError = BISS_SetFrameParam(ulBissHandle, iDataCH, eBISS_SCD_CRC_INVERT, ulCRCinvert[iDataCH]);
  ulError = BISS_SetFrameParam(ulBissHandle, iDataCH, eBISS_SCD_LENGTH, ulSCDLength[iDataCH]);
 }
 ulError = BISS_WriteMasterParams(ulBissHandle); //Write Masterconfig to MB4U
 ulError = BISS_InitBissComm(ulBissHandle);

 //Read one SCD frame to check connection
 unsigned long *pulScdH, *pulScdL;
 pulScdH = new unsigned long [ulNumberOfCycles];
 pulScdL = new unsigned long [ulNumberOfCycles];
 unsigned long ulScdValid;
 unsigned long ulStartControlFrame=0;
 unsigned long ulControlFrameEnd;
 ulError = BISS_ReadSCD(ulBissHandle, pulScdH, pulScdL, &ulScdValid, ulStartControlFrame, &
     ulControlFrameEnd);
 for (int iDataCH = 0; iDataCH < 2; iDataCH++)
 {
  cout <<"Data␣Channel:␣" << iDataCH << "␣SCD:␣0x" << hex << pulScdH[iDataCH] << pulScdL[iDataCH] <<
      endl;
 }
 delete[] pulScdH;
```

```
  delete[] pulScdL;

  //Cycleread of the BiSS-Chain

  //Buffer for DUT data
  unsigned long ulDummyHigh[5000];
  unsigned long ulDummyLow[5000];
  unsigned long ulTimeDummy[5000];
  unsigned long ulStatusDummy[5000];

  //Read number of cycles
  ulError = BISS_ReadSCDFrames(ulBissHandle, 5000, ulDummyHigh, ulDummyLow, ulTimeDummy, ulStatusDummy
      );

  if (ulError)
   cout << "Data␣Acquisition␣failed!"<< endl;
  else
  {
   //Get slavedata from last ReadSCDFrames
   for (unsigned long ulFrameIndex = 0; ulFrameIndex < ulNumberOfCycles; ulFrameIndex++)
   {
    ulError = BISS_GetSCDFrame(ulBissHandle, ulFrameIndex, ulDummyHigh, ulDummyLow, ulTimeDummy,
        ulStatusDummy);
    if (ulError)
    {
     //Release the handle and terminate established connections
     BISS_Close(ulBissHandle);
     return ulError;
    }
    //Get data from BiSS frame. Remove Error and Warning. CRC is already tested and removed from BiSS-
        DLL
    pulaDUT[ulFrameIndex] = (ulDummyLow[0] >>2);
    pulaReference[ulFrameIndex] = (ulDummyLow[1] >> 2);
   }
  }
 }

 //Release the handle and terminate established connections
 BISS_Close(ulBissHandle);

 return ulError;
}

//Example for a calibration with external data.
int main()
{
 unsigned long ulError;


 unsigned long ulaDUTRaw[CYCLECNT];
 unsigned long ulaReference[CYCLECNT];

 //Get external data via BiSS-DLL
 ulError = Measure(CYCLECNT, ulaDUTRaw, ulaReference);

 unsigned long ulMHMHandle;
 MHM_Open(&ulMHMHandle);
 //Get the version of the DLL
 unsigned long ulaVersion[2];
 MHM_GetDLLVersion(ulaVersion);
 cout << "MHM1SL_interface.dll␣Version:␣" << ulaVersion[1] << "." << ulaVersion[0] << endl;

 //Set interface is needed, because analog function check for usefull interfaces
 ulError = MHM_SetInterface(ulMHMHandle, eMHM_MB4U, "");

 //Setting analog calib config
 ulError = MHM_SetCalCfg(ulMHMHandle, eMHM_CAL_REF_TYPE, eMHM_REF_EXTERNAL); //Set to external
     reference, use MHM_ReferenceEnum
 ulError = MHM_SetCalCfg(ulMHMHandle, eMHM_CAL_QUALITY_CHECK, 0);   //Deactivate signal quality check
     when using an external reference
 ulError = MHM_SetCalCfg(ulMHMHandle, eMHM_CAL_REF_RES, 65536);    //Set reference resolution to 16Bit
ulError = MHM_SetCalCfg(ulMHMHandle, eMHM_CAL_REF_PER, 1);     //Set reference period to 1 per
     revolution


//Convert measured data from unsigned long to long
long laDUTRaw[CYCLECNT],laRefRaw[CYCLECNT];
for (int i = 0; i< CYCLECNT; i++)
{
 laDUTRaw[i] = ulaDUTRaw[i];
 laRefRaw[i] = ulaReference[i];
}

//Setting measured data
ulError = MHM_SetCalData(ulMHMHandle, eMHM_DUTRAW,  CYCLECNT, laDUTRaw); //Setting MHM (DUT) raw data
ulError = MHM_SetCalData(ulMHMHandle, eMHM_REFRAW,  CYCLECNT, laRefRaw); //Setting LGC (reference)
     raw data

long lSinOffset, lCosOffset, lGainCorr, lHarm4;
//Analyze analog signals
//The analyses will only work properly, if the Raw Data of the iC-MHM was acquired with the correct
//configuration(see description of function MHM_ActivateCalCfg in the iC-MHM DLL LIBRARY DESCRIPTION)
     .
ulError = MHM_AnalyzeData(ulMHMHandle);
if (ulError)
{
 cout <<"Analysis␣failed!" << endl;
}
else
{
 // Show results
 //Get relative calib parameters
 MHM_GetCalParam(ulMHMHandle, eMHM_CAL_SINOFF, &lSinOffset);
 MHM_GetCalParam(ulMHMHandle, eMHM_CAL_COSOFF,  &lCosOffset);
 MHM_GetCalParam(ulMHMHandle, eMHM_CAL_GAINCOSCORR,  &lGainCorr);
 MHM_GetCalParam(ulMHMHandle, eMHM_CAL_HARM4, &lHarm4);

 cout << "␣Analysis␣MHM:␣" << endl;

 cout << "SinOff:␣\t"<< double(lSinOffset) << endl;
 cout << "CosOff:␣\t"<< double(lCosOffset) << endl;
 cout << "Gain:␣␣␣\t"<< double(lGainCorr) << endl;
 cout << "Harm4:␣␣\t"<< double(lHarm4) << endl;
 cout << "----------------------------------------------------------" <<endl;
}

//Adjust the parameters to the MHM-DLL only
ulError = MHM_Adjust(ulMHMHandle, eMHM_SETONLY);
//Save adjusted config if needed. Could be used to load the config to the MHM afterwards.
ulError = MHM_SaveParams(ulMHMHandle, "c:\\temp\\MHM_adjusted_config.cfg");

//Save measured data
long DUTCont[CYCLECNT];
long DUTError[CYCLECNT];
long RefCont[CYCLECNT];
long AdjDUT[CYCLECNT];
unsigned long ulNroBy= CYCLECNT;

ulError = MHM_GetCalData(ulMHMHandle,eMHM_REFCONT, &ulNroBy, RefCont);
ulError = MHM_GetCalData(ulMHMHandle,eMHM_DUTCONT, &ulNroBy, DUTCont);
ulError = MHM_GetCalData(ulMHMHandle,eMHM_DUTERR, &ulNroBy, DUTError);
ulError = MHM_GetCalData(ulMHMHandle,eMHM_ADJERR, &ulNroBy, AdjDUT);

//Exporting measurement data to a log file
ofstream ofsFile;
ostringstream osValue;
ofsFile.open ("c:\\temp\\MHM_ref_meas.txt", ofstream::out);
bool bAccessTest = ofsFile.is_open();
if (!bAccessTest)
  return eMHM_FILE_ACCESS_DENIED;
osValue.fill('0');
osValue.clear();
```

```
 osValue.str("");
 osValue<<"DUTRAW,␣REFRAW,␣REFCONT,␣DUTCONT,␣DUTERROR,␣ADJUSTDUT"<< endl;
 for (int i=0; i<CYCLECNT;i++)
 {
  osValue<<laDUTRaw[i]<<","<<laRefRaw[i]<<",";
  osValue<<RefCont[i]<<","<<DUTCont[i]<<","<<DUTError[i]<<","<<AdjDUT[i]<<endl;
 }
 ofsFile << osValue.str() << endl;
 ofsFile.close();


 MHM_Close(ulMHMHandle);
}
```

APPENDIX

Prefix	Type	Description
ul	unsigned long	Data type size of 32 bit; positive values only
pul	pointer of unsigned long	
c	char	Data type size of 8 bit
pc	pointer of char	String
d	double	Double precision floating point
pd	pointer of double	
e	enum	List of items

Table 17: Data Type Definitions

Value	MHM_FREQ_SCD Description
0x00.. 0x0F	f_{CLK}/ (2 * (MHM_FREQ_SCD(3:0)+1))
0x10	not permissible
0x11.. 0x1F	f_{CLK}/ (20 * (MHM_FREQ_SCD(3:0)+1))
Notes	f_{CLK} = 20 MHz for PC adapters MB3U (MB3U-I2C), MB4U and MB5U

Table 18: Sensor Data Frequency

Value	MHM_FREQ_AGS Description
0x00.. 0x7B	f_{CLK}/ (20 * (MHM_FREQ_AGS(6:0)+1))
0x7C	AGSMIN
0x7D.. 0x7F	AGSINFINITE
0x80.. 0xFF	f_{CLK}/ (625 * (MHM_FREQ_AGS(6:0)+1))
Notes	f_{CLK} = 20 MHz for PC adapters MB3U (MB3U-I2C), MB4U and MB5U

Table 19: AutoGetSens Frequency

REVISION HISTORY

Modifications made to this document are listed below. For modifications made to the corresponding DLL see the latest release notes (included in the DLL zip package).

Rev	Notes	Pages affected	DLL reference
A1	Initial DLL release	all	Version 1.2
A2	Description of function MHM_WriteParams changed	9	Version 1.4
	New function MHM_UpdateEds added	21	
	EDS_ParamEnum modified	41	
A3	New functions MHM_Calibrate, MHM_Acquire	26	Version 1.5
	MHM_GetCalData, MHM_GetCalParam	33	
	MHM_ErrorEnum, MHM_ConfigDataEnum extended	36, 39	
A4	MHM_ErrorEnum extended	39	Version 1.6
A5	MHM_InterfaceEnum extended	6	Version 1.7
	MHM_ConfigDataEnum extended	36	
	Tables MHM_FREQ_SCD and MHM_FREQ_AGS added	51	
A6	New function MHM_GetInterfaceInfo added	20	Version 1.8
	MHM_ConfigDataEnum extended	36	
B1	MHM_ParamEnum, MHM_ErrorEnum extended	35, 39	Version 2.0
	Chapter Revision History revised	52	
B2	Added parameters NTOA and ENROT_P to calibrate functions	26	Version 2.1
	MHM_ConfigDataEnum extended	36	
	Error description MHM_READPARAM_SSI modified	39	
C1	Added new function MHM_GetLastError with new Enum MHM_ErrorTypeEnum	15	Version 3.0
	Replaced MHM_Calibrate with MHM_FullCalibration	26	
	Replaced MHM_Acquire with MHM_FullAcquisition	27	
	Extended MHM_CalDataEnum	37	
	Extended MHM_ErrorEnum	39	
	Added new chapter Calibration Functions with several new functions and enums	26	
	Moved eMHM_REFERENCE, eMHM_CAL_FREQ_SPI, eMHM_CAL_FREQ_SCD and eMHM_CAL_FREQ_AGS to the new MHM_CalCfgEnum	37	
	Moved the application examples to new Chapter Application Examples and added two new Samples	42	
D1	Updated MHM_ParamEnum according to iC-MHM datasheet version C1	35	Version 4.0
	Extended MHM_E2PAreaEnum	11	
	Added new function MHM_Preset	13	
	Added new optional functions MHM_WriteRegister_I2C, MHM_WriteRegister_I2C_Ex, MHM_ReadRegister_I2C, MHM_LoadRegisterEx and MHM_SaveRegisterEx	22 to 24	
	Updated description of MHM_AdjustGain	29	
	Updated description of MHM_AnalyzeData	31	
	Extended MHM_ErrorEnum	39	
	Updated Example 1 - Getting Started	42	
	Updated Example 2 - Automatic Calibration	43	
	Updated Example 4 - Calibration with external data via BiSS DLL	46	
D2	Extended MHM_ErrorEnum	39	Version 4.1
	Extended MHM_ConfigDataEnum	36	

Rev	Notes	Pages affected	DLL reference
D3	Extended MHM_ConfigDataEnum	36	Version 4.2
D4	Minimum required adapter driver versions added	4	Version 4.3