DESCRIPTION

This document describes the functionality of the shared library available to application developers for the iC-MU Series. The shared library provides an intermediate layer between applications, e.g. Graphical User Interfaces (GUI), and the underlying hardware (drivers). This is demonstrated in Figure 1. In addition, the shared library offers the complete functionality for calibrating the products of the iC-MU series.

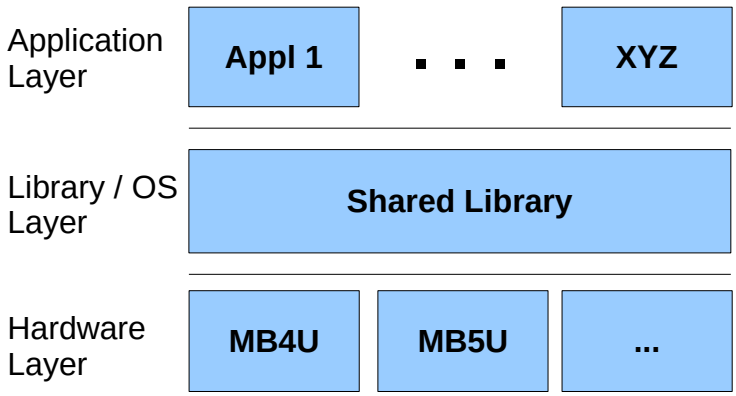


Figure 1: The shared library is the interface between hardware drivers and applications.

Features
This library:

- Reduces costs in evaluation, design-in time, and development
- Provides analog and nonius calibration functions for all iC-MU series products
- Allows access to the chip parameters
- Loads and saves predefined or user-defined Intel HEX configuration files
- Generation of EEPROM Intel HEX files for device configuration
- Reads sensor data, e.g. position data etc.

Supported Programming Environments
This shared library 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 10 and Linux (x86-64).

CONTENTS

GENERAL SETUP

USB adapter drivers
To communicate with the evaluation board, USB adapter drivers need to be installed.

The installation of the drivers is only necessary if a communication via a PC USB adapter is needed, e.g in the **`MU_ReadParams`** function. Other functions, e.g. **`MU_SaveParams`** , do not need a PC USB adapter. If the application uses only functions without communication an installation of the drivers is not needed.

Note: Thoughout this document the terms offline and online are used. This library can be used offline, i.e. only functions are used that do not require communication via a PC USB adapter. When working online also functions that require a PC USB adapter are used.

Windows
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.

Linux-Systems
For the installation of the adapter drivers under Linux follow the instructions in the `drivers/INSTALL_LINUX.md` file.

PACKAGE DESCRIPTION

The iC-MU Series shared library package comprises the following directories:

- `bin`
- `drivers`
- `include`
- `documentation`
- `lv_interface`
- `examples`

The `bin` directory contains the shared libraries for several operating systems (OS) in subdirectories named according to the OS. The subdirectory `windows` includes the Dynamic Link Libraries (.dll) and import libraries (.lib) for the Windows OS.The libraries are provided both as 32 bit and 64 bit compilations. The 32 bit-version is located in the subdirectory `x86`, and the 64 bit-version is located in `x86-64`, respectively. The Shared Object files for the Linux (.so) are provided only as 64 bit-compilation, and can be found in the subdirectory `linux`.

The package includes the latest PC USB-adapter drivers for running 32 and 64 bit applications. The drivers can be found in the directory `drivers`.

The `include` folder contains C/C++ header files for easier integration of the iC-MU Series shared library into customer applications. In `documentation` users can find the release notes and the software manual for the current shared library version. The directory `examples` includes C example programs to demonstrate certain usages of the shared library. The `lv_interface` directory contains all necessary sources for an easy integration of the iC-MU Series shared library into customer LabVIEW applications.

REVISION SUPPORT

Different chip revisions of the iC-MU Series might have different register mappings to the configuration parameters.

Note: Using the wrong mapping might cause functions of the iC-MU Series to work incorrectly.

Therefore, it is necessary to use this library only with supported revisions of the iC-MU Series. For any damage caused by using this library with unsupported or unknown chip revisions, iC-Haus GmbH can not be held responsible.

Supported iC-MU Series chip revisions:

- iC-MU_Y (HARD_REV = 0x05)
- iC-MU_Y1 (HARD_REV = 0x06)
- **iC-MU_Y2 (HARD_REV = 0x07) = latest supported chip revision iC-MU**
- iC-MU150_0 (HARD_REV = 0x10)
- **iC-MU150_1 (HARD_REV = 0x11) = latest supported chip revision iC-MU150**
- **iC-MU200_0 (HARD_REV = 0x20) = latest supported chip revision iC-MU200**

Other chip revisions are divided into two categories. Revisions are unknown if their HARD_REV parameter follows the pattern described above but has a higher value. These could be new revisions of the iC-MU Series which are not yet supported by this library.

Chip revisions are declared as unsupported if they are former chip revisions that are known to have an incompatible register mapping or revisions of the iC-MU Series that are advised not to be used anymore. This library must not be used in conjunction with these products. Please contact iC-Haus GmbH for further information on other versions of this library and libraries that support different products.

GLOSSARY

These key words have specific meanings throughout this document.

Interface	Refers to a MB3U, MB3U-I2C, MB4U or MB5U PC USB adapter that is used to communicate with the chip.
Configuration	A set of hexadecimal values representing all chip specific configuration parameters.
Parameter	1 to 64 specific bits within the chip RAM, EEPROM or Virtual Chip Object assigned to the setting of a single chip function.
Register	Storage of fixed size (e.g. 8 bits) located at a defined address in the chip RAM, EEPROM or in the Virtual Chip Object.
Virtual Chip Object	A virtual copy of the chip.
Online	Using this library to communicate via an iC-Haus PC USB adapter (driver installation required).
Offline	Using this library without any communication via an iC-Haus PC USB adapter (no driver installation required).
Chip Revision	iC-MU Series revision HARD_REV stored in chip RAM at register address 0x74.
Virtual Chip Revision	iC-MU Series revision used for the Virtual Chip Object.

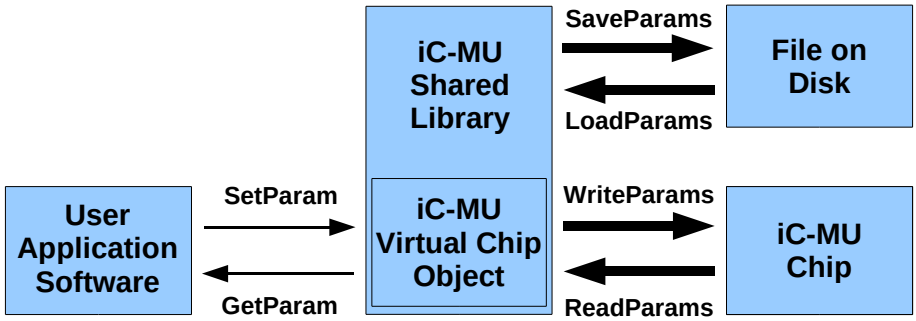


Figure 2: Illustration of the parameter access functions of the iC-MU Series library.

Note: The "SetParam" and "GetParam" functions interact with the Virtual Chip Object.

The actual chip RAM content can be copied to an image of the RAM in the Virtual Chip Object using the "Read-Params" function. Modifications made to the Virtual Chip Object RAM can be transferred to the physical chip RAM using the "WriteParams" function.

The Virtual Chip Object RAM content can be stored as a configuration file on local storage (e.g. PC hard disk) using the "SaveParams" function. A locally stored configuration file can be transferred to the Virtual Chip Object RAM using the "LoadParams" function.

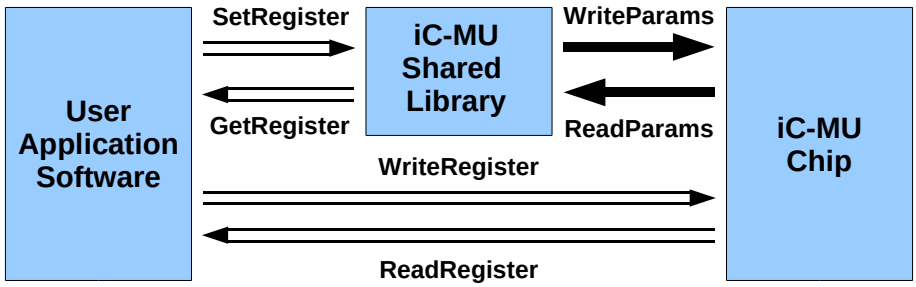


Figure 3: Illustration of the register access functions of iC-MU Series library.

Note: The "SetRegister" and "GetRegister" functions address registers in the Virtual Chip Object.

The "WriteRegister" and "ReadRegister" functions interface directly with the physical chip and need to be used with caution. Any changes made to the physical chip registers using these functions are not necessarily reflected in the Virtual Chip Object.

RECOMMENDED START-UP SEQUENCE

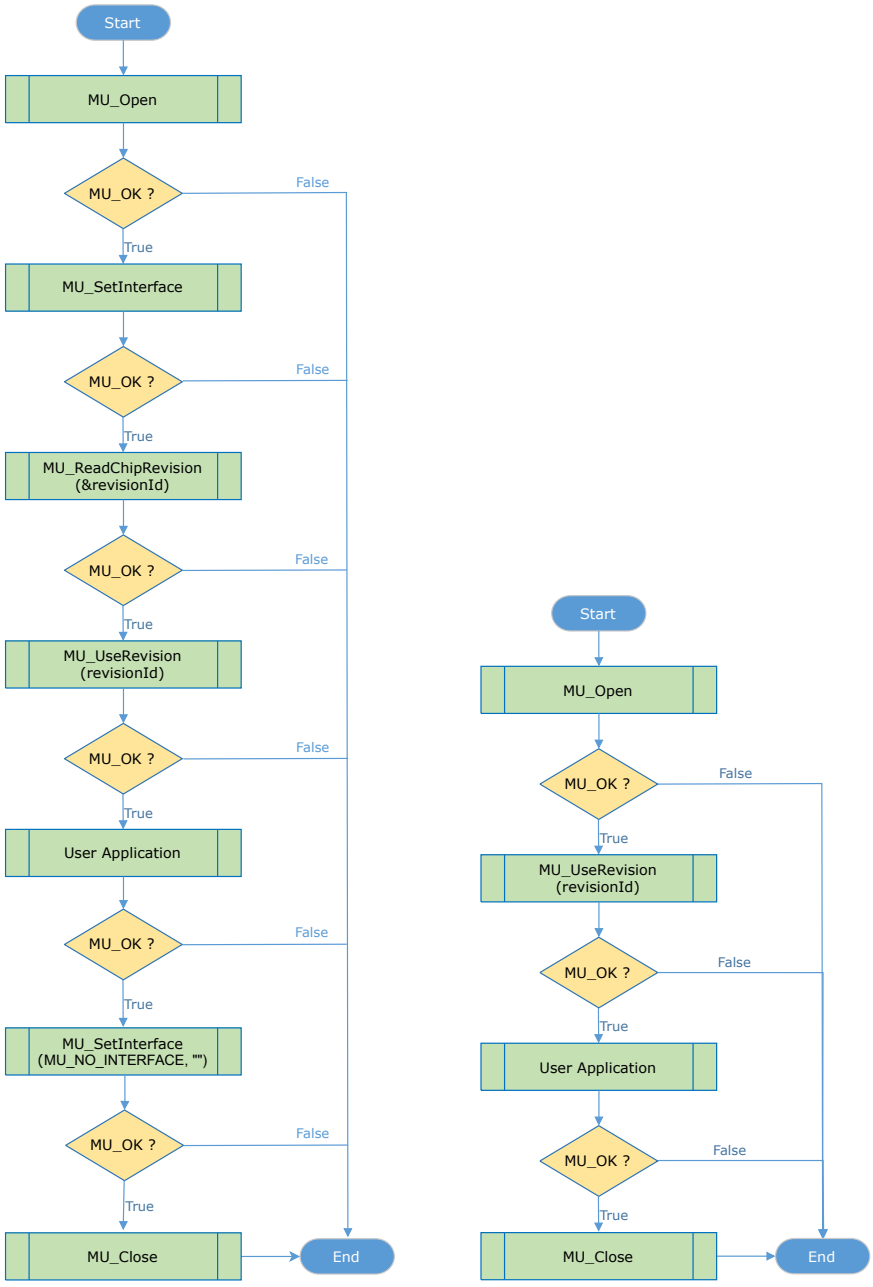
Establish the communication to the iC-MU Series chip following the recommended function call sequence shown in Figure 4 on page 10. All function calls in the library return an error code indicating the success (MU_OK) or failure of the operation (cf. Table 10 on page 76). Every function call in the start-up sequence should be checked for errors. A failure during the start-up sequence indicates problems that need to be fixed before continuing.

At the beginning resources have to be acquired using the **`MU_Open`** function. Also **`MU_Open`** creates a handle that has to be used in subsequent function calls. Next set up the communication using the **`MU_SetInterface`** function.

Some functions provide a check mechanism that compares the chip revision to the virtual chip revision. For proper functionality, both revisions should match. The chip revision is read by calling the **`MU_ReadChipRevision`** function. The chip revision returned is passed to the **`MU_UseRevision`** function to specifiy the revision used for the virtual chip object.

The returned error code will indicate whether this revision is supported (MU_OK) or not. In case of an error, it is strongly recommended to check for new shared library versions and/or chip revisions.

Further information on supported chip revisions can be found in section REVISION SUPPORT.



(a) Online (b) Offline

Figure 4: Recommended start-up sequence using the iC-MU Series library.

FUNCTIONS

MU_Open

`MU_Error` **`MU_Open`** (`MU_Handle` * *`handle`*);

Constructs a new instance of a Virtual Chip Object and obtains a handle for it. The function call acquires all necessary resources and creates the obtained handle. The handle must be used in subsequent function calls that use chip functionalities. To release the handle and close potential connections use **`MU_Close`**.

Parameter	Direction	Description
`handle`	out	Output location for the handle of a Virtual Chip Object (for the type of `MU_Handle` see page 73).

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

See also
`MU_Close`

Example Usage

```
MU_Handle muHandle;
MU_Open(&muHandle);
// ... application code ...
MU_Close(muHandle);
```

MU_Close

`MU_Error` **`MU_Close`** (`MU_Handle` *`handle`*);

Releases the handle, destroys the Virtual Chip Object and its associated resources.

Parameter	Direction	Description
`handle`	in	Handle of a Virtual Chip Object.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

See also
`MU_Open`

MU_SetInterface

```
MU_Error MU_SetInterface(MU_Handle handle,
                         MU_Interface interfaceType,
                         const char * interfaceOption);
```

Sets the interface to a connector type that is specified via the parameter *`interfaceType`* and establishes the connection. To release the interface set *`interfaceType`* to (MU_NO_INTERFACE).

Parameter	Direction	Description
`handle`	in	Handle of a Virtual Chip Object.
`interfaceType`	in	Selection of the interface type, value is equal to the enumeration index of `MU_InterfaceEnum` (see Table 12 on page 77).
`interfaceOption`	in	String containing the last 4 characters of the serial number of the interface to be selected in case more than one MB4U and/or MB5U PC USB adapters are connected. If only one PC USB adapter is connected, an empty string has to be passed. Interface MB3U(MB3U-I2C) does not support this feature, therefore only one MB3U(MB3U-I2C) can be used.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

See also
`MU_GetInterface`

Example Usage

```
MU_Error errorCode = MU_SetInterface(muHandle, MU_NO_INTERFACE, "");
```

```
MU_Error errorCode = MU_SetInterface(muHandle, MU_MB4U, "");
```

```
MU_Error errorCode = MU_SetInterface(muHandle, MU_MB5U, "CABC");
```

MU_GetInterface

```
MU_Error MU_GetInterface(MU_Handle handle,
                         MU_Interface * interfaceType);
```

Get the current (via the function **MU_SetInterface**) selected interface (connector) type.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
interfaceType	out	Output location for the current selected interface type, value is equal to the enumeration index of MU_InterfaceEnum (see Table 12 on page 77).

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_SetInterface

Example Usage

```
MU_Interface interfaceTypeIndex;
MU_GetInterface(muHandle, &interfaceTypeIndex);
MU_InterfaceEnum interfaceType =
    static_cast<MU_InterfaceEnum>(interfaceTypeIndex);
```

MU_SetConfig

```
MU_Error MU_SetConfig(MU_Handle handle,
                      MU_ConfigData configuration,
                      uint32_t value);
```

Sets a configuration property in a specific Virtual Chip Object.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
configuration	in	The configuration property, value is equal to the enumeration index of MU_ConfigDataEnum (see Table 16 on page 82).
value	in	The value of the configuration property.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_GetConfig

Example Usage

```
MU_Error errorCode = MU_SetConfig(muHandle, MU_SLAVE_ID, 0);
```

MU_GetConfig

```
MU_Error MU_GetConfig(MU_Handle handle,
                      MU_ConfigData configuration,
                      uint32_t * value);
```

Gets a configuration property from a specific Virtual Chip Object.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
configuration	in	The configuration property, value is equal to the enumeration index of MU_ConfigDataEnum (see Table 16 on page 82).
value	out	Output location for the value of the configuration property.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_SetConfig

Example Usage

```
uint32_t value;
MU_GetConfig(muHandle, MU_SLAVE_COUNT, &value);
```

MU_SwitchToBiss

```
MU_Error MU_SwitchToBiss(MU_Handle handle,
                         uint32_t * modea,
                         uint32_t * modeb);
```

Switches communication to BiSS protocol if necessary. Only after switching to BiSS the value of address 0x0B is read from the EEPROM and parameters NTOA and MODEB set in the RAM.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
modea	out	Output location for the value of the MODEA parameter. If switched, the value has been read from the EEPROM, if not, the value is read from the RAM.
modeb	out	Output location for the value of the MODEB parameter. If switched, the value has been read from the EEPROM, if not, the value is read from the RAM.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_SwitchToBiss_ex

MU_SwitchToBiss_ex

```
MU_Error MU_SwitchToBiss_ex(MU_Handle handle,
                            uint32_t * modea,
                            bool * switched);
```

This function can be used to switch the interface on port A (MODEA) from SSI to BiSS. If the interface is already configured to BiSS, no switchover is made. This can be checked with the function parameter switched. After switching to BiSS all other chip parameters in address 0x0B (NTOA and MODEB) are restored by reading them from the EEPROM and transferring them to the RAM. The interface set at port A before calling this function is stored in the function parameter modea.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
modea	out	Output location for the value of the MODEA parameter, the value has been read from the EEPROM.
switched	out	Output location for a boolean value that is true when the chip interface has been changed.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

MU_EnableGetMT

```
MU_Error MU_EnableGetMT(MU_Handle handle,
                        uint32_t *modea,
                        bool *switched);
```

This function is intended for 3-Track applications (see Figure 5 on page 17). The function writes the value '1' to the parameter GET_MT of the singleturn iC-MU and switches the communication of the multiturn chip device to BiSS protocol (see function **MU_SwitchToBiss_ex**). Thereafter the BiSS IDs are rearranged, hence the singleturn chip device gets the ID '1'. Does not work with SPI.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
modea	out	Output location for the value of the MODEA parameter of the multiturn iC-MU, the value has been read from the EEPROM.
switched	out	Output location for a boolean value that is true when the chip interface has been changed.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_DisableGetMT

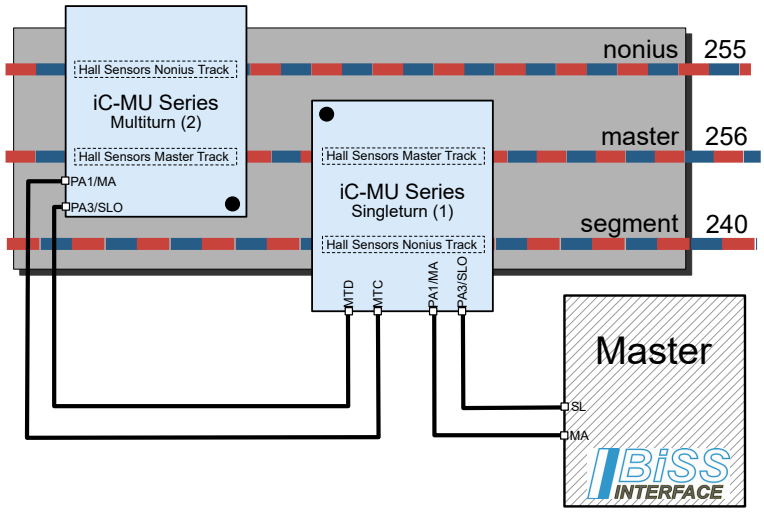


Figure 5: 3-Track application.

MU_DisableGetMT

`MU_Error` **`MU_DisableGetMT`** (`MU_Handle` *`handle`*);

This function is intended for 3-Track applications. The function **`MU_EnableGetMT`** must be called beforehand. Writes the value '0' to the parameter GET_MT of the singleturn iC-MU. Thereafter the BiSS IDs are rearranged, hence the singleturn chip device gets the ID '0'.

Parameter	Direction	Description
`handle`	in	Handle of a Virtual Chip Object.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

See also
`MU_EnableGetMT`

MU_Initialize

`MU_Error` **`MU_Initialize`** (`MU_Handle` *`handle`*);

BiSS: Sends a BiSS-Init and reads a register from the chip RAM to test the communication. SPI: Reads a register from the chip RAM to test the communication.

Parameter	Direction	Description
`handle`	in	Handle of a Virtual Chip Object.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

MU_SetParam

```
MU_Error MU_SetParam(MU_Handle handle,
                     MU_Param parameter,
                     uint32_t valueHigh,
                     uint32_t valueLow,
                     MU_WriteVerify writeVerify);
```

Sets a chip configuration parameter in the Virtual Chip Object.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
parameter	in	The chip configuration parameter, value is equal to the enumeration index of MU_ParamEnum (see Table 13 on page 80).
valueHigh	in	The upper 32 bits of the value of the chip configuration parameter if the parameter value size is greater than 32 bits.
valueLow	in	The lower 32 bits of the value of the chip configuration parameter.
writeVerify	in	Flag used to indicate whether to set the value only in the Virtual Chip Object memory, set the value and write it to the chip RAM, or set the value, write it to the chip RAM, and verify the successful write operation (see Table 15 on page 81).

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_GetParam
MU_WriteParams
MU_ReadParams

Example Usage

```
// Set parameter ENAC to '1' (amplitude control unit activation) in the virtual
    chip object and write the corresponding register to the iC-MU Series
    immediately.
MU_Error errorCode = MU_SetParam(muHandle, MU_ENAC, 0, 1, MU_VERIFY);
```

MU_GetParam

```
MU_Error MU_GetParam(MU_Handle handle,
                     MU_Param parameter,
                     uint32_t * valueHigh,
                     uint32_t * valueLow);
```

Gets an chip configuration parameter from the Virtual Chip Object.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
parameter	in	The chip configuration parameter, value is equal to the enumeration index of MU_ParamEnum (see Table 13 on page 80).
valueHigh	out	Output location for the value of the upper 32 bits of the chip configuration parameter if the parameter value size is greater than 32 bits. In case that the parameter value size is smaller than 32 bit, it is possible to pass a nullptr for *valueHigh*.
valueLow	out	Output location for the value of the lower 32 bits of the chip configuration parameter.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_SetParam
MU_WriteParams
MU_ReadParams

MU_IsParameterAvailableInUsedRevision

```
MU_Error MU_IsParameterAvailableInUsedRevision(MU_Handle handle,
                                               MU_Param parameter,
                                               bool * isInRevision);
```

Checks the availability of a chip configuration parameter of the currently used chip revision.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
parameter	in	The chip configuration parameter, value is equal to the enumeration index of MU_ParamEnum (see Table 13 on page 80).
isInRevision	out	Output location for the information, true means the parameter is available in the currently used chip revision.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

See also
`MU_GetParam`
`MU_UseRevision`

MU_WriteParams

```
MU_Error MU_WriteParams (MU_Handle handle,
                         bool verify,
                         bool * valid);
```

Writes all configuration registers from the Virtual Chip Object to the chip RAM (address ranges: 0x00..0x25, 0x41..0x5F, 0x78..0x7F; parameter MODE_A is written last).

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
verify	in	On true, the function reads the register after write back for verification. On false, it writes the registers without verification.
valid	out	Output location for an array. Provide space for at least 128 boolean values. In case of verification, this array is used to specify which registers are valid. If the written value of a register is equal to the read value, the function writes a true in the array at the position corresponding to the register. If the verification fails, the array contains a false at the position corresponding to the register.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

See also
`MU_SetParam`
`MU_GetParam`
`MU_ReadParams`

MU_ReadParams

```
MU_Error MU_ReadParams (MU_Handle handle);
```

Reads all configuration registers from the chip into the Virtual Chip Object (address ranges: 0x00..0x25, 0x41..0x5F, 0x74, 0x78..0x7F).

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_SetParam
MU_GetParam
MU_WriteParams

MU_ReadStatus

MU_Error **MU_ReadStatus** (MU_Handle *handle*);

Reads the status registers (register address 0x76..0x77) from the chip into the Virtual Chip Object.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_ReadRegister
MU_GetRegister

MU_ReadGain

MU_Error **MU_ReadGain** (MU_Handle *handle*);

Reads the gain registers (register address 0x2B, and 0x2F) from the chip into the Virtual Chip Object.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_ReadRegister
MU_GetRegister

MU_WriteEeprom

```
MU_Error MU_WriteEeprom(MU_Handle handle,
                        MU_E2PArea eepromArea);
```

Transfers the chip parameters from the Virtual Chip Object to the chip RAM. Executes the WRITE_ALL command to store the chip parameters into the EEPROM. The area to be stored is determined ty eepromArea

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
eepromArea	in	Area to be stored in the EEPROM (see MU_E2PAreaEnum Table 17 on page 82).

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

MU_WriteCmdRegister

```
MU_Error MU_WriteCmdRegister(MU_Handle handle,
                             MU_Command command);
```

Writes the command register (addr. 0x75) of the chip.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
command	in	The command to send (see MU_CommandEnum Table 14 on page 81).

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

Note
This function reads the status registers 0x76 and 0x77 after execution of the following commands: WRITE_ALL, WRITE_OFF, SOFT_RESET, SOFT_E2P_PRES, I2C_COM, and SWITCH.

MU_WriteSwitchCommand

```
MU_Error MU_WriteSwitchCommand(MU_Handle handle,
                               uint32_t modeaNew,
                               uint32_t rplNew);
```

Executes the switch command with the specified MODEA and RPL values.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
modeaNew	in	The EEPROM value of MODEA.
rplNew	in	The EEPROM value of RPL.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

MU_ReadSens

```
MU_Error MU_ReadSens(MU_Handle handle,
                     ReadSensStruct * data);
```

Reads the position values from the chip.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
data	out	Output location to the data structure containing the position values (see `ReadSensStruct` Table 4 on page 73).

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

MU_SaveParams

```
MU_Error MU_SaveParams(MU_Handle handle,
                       const char * filepath);
```

Saves chip parameters from the Virtual Chip Object to the specified parameter file. The file format is Intel HEX.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
filepath	in	Null-terminated string for the absolute path of the parameter file.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

See also
`MU_LoadParams`

MU_ReadChipRevisionOfParameterFile

```
MU_Error MU_ReadChipRevisionOfParameterFile(MU_Handle handle,
                                            const char * filepath,
                                            uint8_t * revisionCode);
```

Read the chip revision from the specified file. Use this function before calling **`MU_LoadParams`** to make sure that the version of the parameter file matches the set chip revision (see **`MU_UseRevision`**).

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
filepath	in	Null-terminated string for the absolute path of the file.
revisionCode	out	Revision code of the parameter file. If no revision code set in the file, the value of revisionCode is set to 0.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_LoadParams
MU_UseRevision
MU_ValueOfUsedRevision

MU_LoadParams

```
MU_Error MU_LoadParams(MU_Handle handle,
                       const char * filepath);
```

Loads chip parameters from the specified parameter file to the Virtual Chip Object. The file format is Intel HEX.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
filepath	in	Null-terminated string for the absolute path of the parameter file.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_SaveParams
MU_ReadChipRevisionOfParameterFile

MU_GetLastError

```
MU_Error MU_GetLastError(MU_Handle handle,
                         MU_Error * lastError,
                         MU_ErrorType * errorType,
                         char * errorText );
```

Returns information about the last error that occurred with the given *muHandle*.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
lastError	out	Output location for the error code of the last error that occurred, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76).
errorType	out	Output location for the type of the last error that occurred, the value is equal to the index of the error type enumeration MU_ErrorTypeEnum (see Table 11 on page 77).
errorText	out	Output location to a char array (minimum size: 1024 entries). The error message of the last error that occurred is copied to this location as a null-terminated character string (see Table 10 on page 76).

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

Example Usage

```
MU_Error lastError;
MU_ErrorType errorType;
char errorText[1024];
MU_GetLastError(muHandle, &lastError, &errorType, errorText);
```

MU_GetDLLVersion

```
MU_Error MU_GetDLLVersion(uint32_t * version);
```

Gets the Shared Library Version.

Parameter	Direction	Description
version	out	Output location for the array with two elements that contains the library version.

Return Value
A function call always returns the successful error code 0 (MU_OK).

Note
Index '1' of the Shared Library Version array contains the major version number, index '0' the minor, patch, and build version number (see example usage). The function does not need a handle.

Example Usage

```
// Get the Shared Library Version.
uint32_t muLibVersion[2];
MU_GetDLLVersion(muLibVersion);
const uint32_t MU_3SL_LIB_MAJOR_VERSION = muLibVersion[1];
const uint8_t  MU_3SL_LIB_MINOR_VERSION = muLibVersion[0] & 0xff;
const uint8_t  MU_3SL_LIB_PATCH_VERSION = (muLibVersion[0] >> 8) & 0xff;
const uint16_t MU_3SL_LIB_BUILD_VERSION = (muLibVersion[0] >> 16) & 0xffff;
```

MU_GetVersionString

```
const char* MU_GetVersionString ();
```

Returns the Shared Library Version string. A example return value for the version 3.0.0.0 is: "3.0.0.0".

Return Value
Shared Library Version string.

See also
MU_GetVersionMajor
MU_GetVersionMinor
MU_GetVersionPatch
MU_GetVersionBuild
MU_GetVersionSuffixString

MU_GetVersionMajor

```
uint16_t MU_GetVersionMajor ();
```

Return the library major version.

Return Value
Library major version

See also
MU_GetVersionString

MU_GetVersionMinor

```
uint16_t MU_GetVersionMinor ();
```

Return the library minor version.

Return Value
Library minor version

See also
MU_GetVersionString

MU_GetVersionPatch

```
uint16_t MU_GetVersionPatch ();
```

Return the library patch version.

Return Value
Library patch version

See also
MU_GetVersionString

MU_GetVersionBuild

```
uint16_t MU_GetVersionBuild ();
```

Return the library build version.

Return Value
Library build version

See also
MU_GetVersionString

MU_GetVersionSuffixString

```
const char* MU_GetVersionSuffixString ();
```

Returns the Shared Library Version Suffix string. A example return value for the a beta version is: "-beta".

Return Value
Shared Library Version Suffix string.

See also
MU_GetVersionString
MU_GetVersionMajor
MU_GetVersionMinor
MU_GetVersionPatch
MU_GetVersionBuild

MU_CalcPRES_POS

```
MU_Error MU_CalcPRES_POS (MU_Handle handle,
                          uint32_t valueHigh,
                          uint32_t valueLow,
                          MU_WriteVerify writeVerify );
```

Sets the iC parameter PRES_POS depending on the desired preset position and the parameter OUT_LSB.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
valueHigh	in	The desired preset position value (upper 32 bits).
valueLow	in	The desired preset position value (lower 32 bits).
writeVerify	in	Flag used to indicate whether to set the value only in the Virtual Chip Object memory, set the value and write it to the chip RAM, or set the value, write it to the chip RAM, and verify the successful write operation (see Table 15 on page 81).

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

Note
Use **MU_GetParam** to get the calculated PRES_POS parameter.

MU_GetPRES_POS

```
MU_Error MU_GetPRES_POS(MU_Handle handle,
                        uint32_t * valueHigh,
                        uint32_t * valueLow);
```

Gets the real preset position depending on the parameters PRES_POS and OUT_LSB.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
valueHigh	out	Output location for the preset position value (upper 32 bits).
valueLow	out	Output location for the preset position value (lower 32 bits).

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

MU_UseRevision

```
MU_Error MU_UseRevision(MU_Handle handle,
                        uint8_t revisionId);
```

Configure the library to use the functionality and the register mapping of Chip Revision passed via revisionId. It is mandatory to execute this function to unlock functionality that uses revision-specific features. In case the Chip Revision is not supported by the current Shared Library Version the library will use the latest Chip Revision.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
revisionId	in	Chip Revision (HARD_REV) to be used by the library for subsequent actions.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_ReadChipRevision
MU_ValueOfUsedRevision

Note
Use function **MU_ReadChipRevision** to read the revision of the chip.

MU_ValueOfUsedRevision

```
MU_Error MU_ValueOfUsedRevision(MU_Handle handle,
                                uint8_t * revisionId);
```

Get the Virtual Chip Revision configured by **MU_UseRevision**. If **MU_UseRevision** has not been executed before, revisionId will contain 0.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
revisionId	out	Output location for the Virtual Chip Revision used by the library for subsequent actions.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_UseRevision
MU_ReadChipRevision

MU_ReadChipRevision

```
MU_Error MU_ReadChipRevision(MU_Handle handle,
                             uint8_t * revisionId);
```

Reads the register containing parameter HARD_REV.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
revisionId	out	Output location for the Chip Revision.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also

`MU_UseRevision`
`MU_ValueOfUsedRevision`

OPTIONAL FUNCTIONS

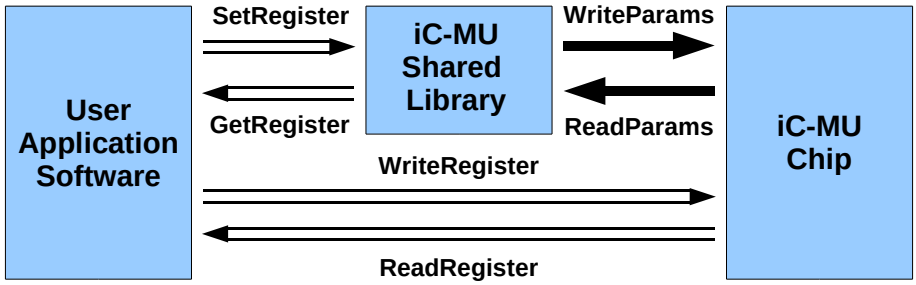


Figure 6: Illustration of the chip parameter and the chip register access functions of the chip library.

MU_SetRegister

```
MU_Error MU_SetRegister(MU_Handle handle,
                        uint32_t address,
                        uint32_t value,
                        MU_WriteVerify writeVerify);
```

Sets a chip register value in the Virtual Chip Object.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
address	in	The address of the register.
value	in	The value of the register.
writeVerify	in	Flag used to indicate whether to set the value only in the Virtual Chip Object memory, set the value and write it to the chip RAM, or set the value, write it to the chip RAM, and verify the successful write operation (see Table 15 on page 81).

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_WriteParams
MU_GetRegister

MU_GetRegister

```
MU_Error MU_GetRegister(MU_Handle handle,
                        uint32_t address,
                        uint32_t * value);
```

Gets a chip register value from the Virtual Chip Object.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
address	in	The address of the register.
value	out	Output location for the value of the register.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_ReadParams
MU_SetRegister

MU_WriteRegister

```
MU_Error MU_WriteRegister(MU_Handle handle,
                          uint32_t address,
                          uint32_t * count,
                          uint32_t * values);
```

This function is used to write data to a number of consecutive registers in the on-chip RAM. The given register values are also set in a specific Virtual Chip Object.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
address	in	The address of the first register to start writing to.
count	in/out	Pointer to the number of registers to be written. In case of a failure, the output is the number of the successfully written registers.
values	in	Pointer to an array with the register values to be written.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

Note
The register address is automatically incremented, thus the address of the last register written is: last_address = address + count - 1.

See also
`MU_ReadRegister`
`MU_SetRegister`

MU_ReadRegister

```
MU_Error MU_ReadRegister(MU_Handle handle,
                         uint32_t address,
                         uint32_t * count,
                         uint32_t * values);
```

This function is used to read data from a number of consecutive registers in the on-chip RAM. The read register values are also set in a specific Virtual Chip Object.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
address	in	The address of the first register to start reading from.
count	in/out	Pointer to the number of registers to be read. In case of a failure, the output is the number of the successfully read registers.
values	out	Pointer to an array of size corresponding to at least the number of registers to be read.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

Note
The register address is automatically incremented, thus the address of the last register read is: last_address = address + count - 1.

See also
`MU_WriteRegister`
`MU_GetRegister`

MU_WriteRegisterSSI

```
MU_Error MU_WriteRegisterSSI(MU_Handle handle,
                             uint32_t address,
                             uint32_t * count,
                             uint32_t * values);
```

This function is used to write data to a number of consecutive registers in the on-chip RAM. This function allows write access to the IC registers in SSI mode.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
address	in	The address of the first register to start writing to.
count	in/out	Pointer to the number of registers to be written.
values	in	Pointer to an array with the register values to be written.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

See also
`MU_WriteRegister`

MU_SaveRegister

```
MU_Error MU_SaveRegister(MU_Handle handle,
                         const char * filepath,
                         uint32_t address,
                         uint32_t * count,
                         uint32_t * values);
```

Saves data from an array to a file (address range = start address + number of bytes -1). Always start writing at register 0x00. The file format is Intel HEX.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
filepath	in	Null-terminated string for the absolute path of the file.
address	in	The address of the first register to start saving to.
count	in/out	Pointer to the number of registers to be saved.
values	in	Pointer to an array with the register values to be saved.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

Note
If the file already exists, only the data of the specified address range will be replaced.

See also
`MU_LoadRegister`

MU_LoadRegister

```
MU_Error MU_LoadRegister(MU_Handle handle,
                         const char * filepath,
                         uint32_t address,
                         uint32_t * count,
                         uint32_t * values);
```

Loads data from a file to an array (address range = start address + number of bytes -1). The file format is Intel HEX.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
filepath	in	Null-terminated string for the absolute path of the file.
address	in	The address of the first register to start loading from.
count	in/out	Pointer to the number of registers to be loaded.
values	out	Pointer to an array the register values are loaded to.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

See also
`MU_SaveRegister`

MU_SpiActivate

```
MU_Error MU_SpiActivate(MU_Handle handle,
                        uint32_t data);
```

Sends the SPI command ACTIVATE to the chip.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
data	in	Data byte (RACTIVE/PACTIVE).

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

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

MU_GetInterfaceInfo

```
MU_Error MU_GetInterfaceInfo(MU_Handle handle,
                             MU_InterfaceInfo informationType,
                             char * information);
```

Gets additional information about the selected interface.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
informationType	in	Interface info to be retrieved (see MU_InterfaceInfoEnum Table 18 on page 82)
information	out	Pointer to a string the retrieved information is written to.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

MU_WriteRegister_I2C

```
MU_Error MU_WriteRegister_I2C(MU_Handle handle,
                              uint32_t registerAddress,
                              uint32_t registerCount,
                              uint32_t i2cSlaveAddress,
                              const uint32_t * values);
```

Writes data to an I2C device by using chip command I2C_COM. Uses the chip RAM 0x60..0x6F as communication buffer.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
registerAddress	in	The address of the register.
registerCount	in	Number of registers to write.
i2cSlaveAddress	in	I2C slave address (Eeprom = 0x50).
values	in	Pointer to an array containing the values.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

Note
Use function **MU_WriteRegister_I2C_Ex** in combination with **MU_LoadRegisterEx** to write data from an Intel HEX file!

MU_WriteRegister_I2C_Ex

```
MU_Error MU_WriteRegister_I2C_Ex(MU_Handle handle,
                                 uint32_t registerCount,
                                 uint32_t i2cSlaveAddress,
                                 const uint32_t * values,
                                 const bool * valid);
```

Writes an I2C device by using chip command I2C_COM. Uses the chip Ram 0x60..0x6F as communication buffer. Additional to the function **MU_WriteRegister_I2C** only the registers marked by the valid array (address range = 0x00 to registerCount - 1) will be written.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
registerCount	in	Number of registers to write.
i2cSlaveAddress	in	I2C slave address (Eeprom = 0x50).
values	in	Pointer to an array containing the values.
valid	in	Pointer to an validation array. Value '1' data will be written, value '0' no writing.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

Note
Use this function in combination with **MU_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-PVL software.

MU_ReadRegister_I2C

```
MU_Error MU_ReadRegister_I2C(MU_Handle handle,
                             uint32_t registerAddress,
                             uint32_t registerCount,
                             uint32_t i2cSlaveAddress,
                             uint32_t * values);
```

Reads data from the I2C-device registers to an array (address range = start address + number of bytes - 1).

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
registerAddress	in	The address of the register.
registerCount	in	Number of registers to write.
i2cSlaveAddress	in	I2C slave address (Eeprom = 0x50).
values	out	Pointer to an array containing the values.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

Note
Use function **MU_WriteRegister_I2C_Ex** in combination with **MU_LoadRegisterEx** to write data from an Intel HEX file!

MU_SaveRegisterEx

```
MU_Error MU_SaveRegisterEx(MU_Handle handle,
                           const char * filepath,
                           uint32_t registerAddress,
                           uint32_t registerCount,
                           uint32_t * values);
```

Saves data from an array to a file (address range = start address + number of bytes -1). Always start writing at register 0x00. The file format is Intel HEX.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
filepath	in	Null-terminated string for the absolute path of the file.
registerAddress	in	First register address.
registerCount	in	Number of registers to be save.
values	in	Location to an array with the register values to be save.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

Note
If the file already exists, the old file will be replaced.

See also
`MU_LoadRegister`

MU_LoadRegisterEx

```
MU_Error MU_LoadRegisterEx(MU_Handle handle,
                           const char * filepath,
                           uint32_t registerCount,
                           uint32_t * values,
                           bool * valid);
```

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

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
filepath	in	Null-terminated string for the absolute path of the file.
registerCount	in	Number of registers to be load.
values	out	Output location to an array with the minimum size of the registers to be load for the loaded register values.
valid	out	Pointer to an validation array. Value '1' indicates valid data, value '0' no valid data.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

See also
`MU_SaveRegister`

CALIBRATION FUNCTIONS

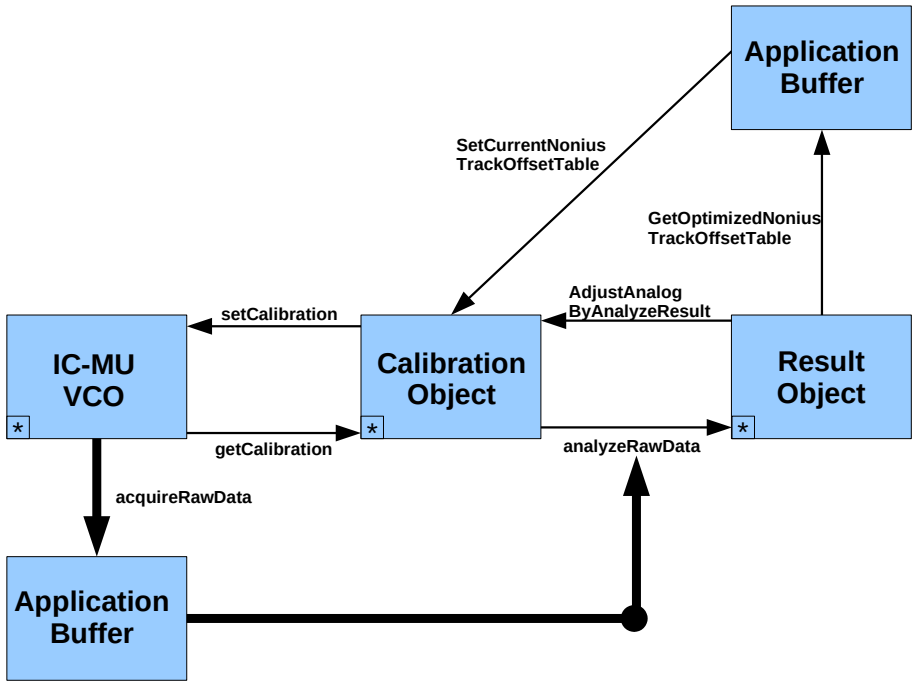


Figure 7: Illustration of the basic calibration functions

MU_acquireRawData

```
MU_Error MU_acquireRawData(MU_Handle handle,
                           uint16_t* masterRawData,
                           uint16_t* noniusRawData,
                           size_t nSamples,
                           uint32_t slaveId,
                           double frameCycleTime_s,
                           double clockFreq_hz);
```

Acquire a specific number of raw data samples via the selected interface.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
masterRawData	out	Pointer to a master track raw data buffer.
noniusRawData	out	Pointer to a nonius track raw data buffer.
nSamples	in	Number of samples to be acquired.
slaveId	in	iC-MU Slave ID.
frameCycleTime_s	in	Frame cycle time in seconds.*
clockFreq_hz	in	Clock frequency in Hz.*

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

Note
* Use **MU_Interface_nextPossibleFrameCycleTime**, **MU_Interface_nearestPossibleFrameCycleTime**, **MU_Interface_nextPossibleClockFreq** and **MU_Interface_nearestPossibleClockFreq** to determine if the used interface supports the clock frequency and frame repetition rate.

See also
MU_Calibration_analyzeRawData
MU_activateCalibrationConfig
MU_deactivateCalibrationConfig

MU_Interface_nextPossibleFrameCycleTime

```
double MU_Interface_nextPossibleFrameCycleTime (MU_Interface interfaceType,
                                                double frameCycleTime_s);
```

Returns the next supported frame cycle time for a given interface.

Parameter	Direction	Description
interfaceType	in	Interface type, value is equal to the enumeration index of MU_InterfaceEnum (see Table 12 on page 77).
frameCycleTime_s	in	Current frame cycle time in seconds.

Return Value
Next possible frame cycle time.

MU_Interface_nearestPossibleFrameCycleTime

```
double MU_Interface_nearestPossibleFrameCycleTime (MU_Interface interfaceType,
                                                   double frameCycleTime_s);
```

Returns the nearest supported frame cycle time for a given interface.

Parameter	Direction	Description
interfaceType	in	Interface type, value is equal to the enumeration index of MU_InterfaceEnum (see Table 12 on page 77).
frameCycleTime_s	in	Current frame cycle time in seconds.

Return Value
Nearest possible frame cycle time.

MU_Interface_nextPossibleClockFreq

```
double MU_Interface_nextPossibleClockFreq(MU_Interface interfaceType,
                                          double clockFreq_hz);
```

Returns the next supported clock frequency for a given interface.

Parameter	Direction	Description
interfaceType	in	Interface type, value is equal to the enumeration index of MU_InterfaceEnum (see Table 12 on page 77).
clockFreq_hz	in	Current clock frequency in Hz.

Return Value
Next possible clock frequency.

MU_Interface_nearestPossibleClockFreq

```
double MU_Interface_nearestPossibleClockFreq(MU_Interface interfaceType,
                                             double clockFreq_hz);
```

Returns the nearest supported clock frequency for a given interface.

Parameter	Direction	Description
interfaceType	in	Interface type, value is equal to the enumeration index of MU_InterfaceEnum (see Table 12 on page 77).
clockFreq_hz	in	Current clock frequency in Hz.

Return Value
Nearest possible clock frequency.

MU_activateCalibrationConfig

`MU_Error` **`MU_activateCalibrationConfig`** (`MU_Handle` *`handle`*);

Configures the iC-MU and BiSS-Master/SPI-Interface for reading the position values via **`MU_acquireRawData`** (see page 43). This function stores a backup of the current configuration, which will be restored by the function **`MU_deactivateCalibrationConfig`** (see page 46). The parameter changes can be seen in table 1 page 46.

Parameter	Direction	Description
`handle`	in	Handle of a Virtual Chip Object.

Return Value
On success, returns 0 (`MU_OK`).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

See also
`MU_deactivateCalibrationConfig`

Chip Parameter	Value	Description
eMU150_ENAC	0x01	Enable Amplitude Control
eMU150_OUT_LSB	0x00	LSB = 0
eMU150_OUT_MSB	0x0E	MSB = 13 + 14 = 27
eMU150_OUT_ZERO	0x04 0x00	4 additional zeros to get to 32 Bit for SPI. For BiSS no additional zeros
eMU150_MODE_ST	0x02	Raw data nonius and master track
eMU150_TEST	0x00	deactivate all testmodes
eMU150_MPC		MPC == 12 not allowed for raw data
Config Parameter	**Value**	**Description**
eMU150_READ_GAIN_ENABLE	0x00	Deactivate READ GAIN
eMU150_READ_STATUS_ENABLE	0x00	Deactivate READ STATUS

Table 1: iC-MU calibration configuration

MU_deactivateCalibrationConfig

`MU_Error` **`MU_deactivateCalibrationConfig`** (`MU_Handle` *`handle`*);

Restores the configuration saved in **`MU_activateCalibrationConfig`** (see page 46).

Parameter	Direction	Description
`handle`	in	Handle of a Virtual Chip Object.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

See also
`MU_activateCalibrationConfig`

MU_getCalibration

```
MU_Calibration* MU_getCalibration(MU_Handle handle);
```

Creates a new instance of a calibration object initialized with the Virtual Chip Object properties.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.

Return Value
Pointer to a new calibration object.

MU_setCalibration

```
MU_Error MU_setCalibration(MU_Handle handle,
                           const MU_Calibration* calibration);
```

Updates a Virtual Chip Object with the properties from a calibration object.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
calibration	in	Pointer to the MU_Calibration object.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

MU_createCalibration

```
MU_Calibration* MU_createCalibration(uint8_t revisionCode);
```

Creates a new default instance of a calibration object. The properties have to be set manually.

Parameter	Direction	Description
revisionCode	in	iC-MU Series chip revision (HARD_REV).

Return Value
A pointer to a new calibration object.

MU_Calibration_delete

```
void MU_Calibration_delete(MU_Calibration* calibration);
```

Deletes a calibration object.

Parameter	Direction	Description
calibration	in	Pointer to a MU_Calibration object.

See also
MU_createCalibration
MU_getCalibration

MU_Calibration_preconfigureNumberOfMasterPeriods

```
bool MU_Calibration_preconfigureNumberOfMasterPeriods(MU_Calibration* calibration,
                                                      unsigned int nMasterPeriods);
```

Configures the number of master periods in the calibration object.

Parameter	Direction	Description
calibration	in	Pointer to a MU_Calibration object.
nMasterPeriods	in	Number of master periods.

Return Value
true = configuration succeeded, false = configuration failed.

See also
MU_createCalibration
MU_getCalibration

MU_Calibration_setCurrentAnalogTrackAdjustments

```
bool MU_Calibration_setCurrentAnalogTrackAdjustments(
    MU_Calibration* calibration,
    const MU_Calibration_AnalogTrackAdjustments* masterAdjustments,
    const MU_Calibration_AnalogTrackAdjustments* noniusAdjustments);
```

Set analog parameters in a calibration object.

Parameter	Direction	Description
calibration	in	Pointer to a MU_Calibration object.
masterAdjustments	in	Pointer to an MU_Calibration_AnalogTrackAdjustments object with the master track parameters.
noniusAdjustments	in	Pointer to an MU_Calibration_AnalogTrackAdjustments object with the nonius track parameters.

Return Value
`true` = configuration succeeded, `false` = configuration failed.

See also
MU_createCalibration
MU_getCalibration

MU_Calibration_setCurrentNoniusTrackOffsetTable

```
bool MU_Calibration_setCurrentNoniusTrackOffsetTable(
    MU_Calibration* calibration,
    const MU_Calibration_NoniusTrackOffsetTable* noniusTrackOffsetTable);
```

Sets the nonius offset table in a calibration object.

Parameter	Direction	Description
calibration	in	Pointer to a MU_Calibration object.
noniusTrackOffsetTab	in	Pointer to an MU_Calibration_NoniusTrackOffsetTable object.

Return Value
`true` = configuration succeeded, `false` = configuration failed.

See also
MU_createCalibration
MU_getCalibration

MU_Calibration_getAnalogMasterTrackAdjustments

```
void MU_Calibration_getAnalogMasterTrackAdjustments(
    const MU_Calibration* calibration,
    MU_Calibration_AnalogTrackAdjustments* dest);
```

Returns the analog master track parameters from a calibration object.

Parameter	Direction	Description
calibration	in	Pointer to a MU_Calibration object.
dest	out	Output location.

See also
MU_createCalibration
MU_getCalibration

MU_Calibration_getAnalogNoniusTrackAdjustments

```
void MU_Calibration_getAnalogNoniusTrackAdjustments(
    const MU_Calibration* calibration,
    MU_Calibration_AnalogTrackAdjustments* dest);
```

Returns the analog nonius track parameters from a calibration object.

Parameter	Direction	Description
calibration	in	Pointer to a MU_Calibration object.
dest	out	Output location.

See also
MU_createCalibration
MU_getCalibration

MU_Calibration_getNoniusTrackOffsetTable

```
void MU_Calibration_getNoniusTrackOffsetTable(
    const MU_Calibration* calibration,
    MU_Calibration_NoniusTrackOffsetTable* dest);
```

Returns the nonius track offset table from a calibration object.

Parameter	Direction	Description
calibration	in	Pointer to a MU_Calibration object.
dest	out	Output location.

See also
MU_createCalibration
MU_getCalibration

Calibration Result

MU_Calibration_analyzeRawData

```
MU_CalibrationAnalyzeResult* MU_Calibration_analyzeRawData(
    const MU_Calibration* calibration,
    const uint16_t* masterTrackRawData,
    const uint16_t* noniusTrackRawData,
    size_t nRawDataSamples);
```

Analyzes raw data samples and returns a pointer to a MU_CalibrationAnalyzeResult object.

Parameter	Direction	Description
calibration	in	Pointer to a MU_Calibration object.
masterTrackRawData	in	Raw data samples of the master track.
noniusTrackRawData	in	Raw data samples of the nonius track.
nRawDataSamples	in	Number of raw data samples.

Return Value
Pointer to a MU_CalibrationAnalyzeResult object. In case of an error returns `NULL`.

See also
MU_createCalibration

MU_CalibrationAnalyzeResult_delete

```
void MU_CalibrationAnalyzeResult_delete(MU_CalibrationAnalyzeResult* analyzeResult);
```

Deletes a MU_CalibrationAnalyzeResult object.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.

See also
MU_Calibration_analyzeRawData

MU_Calibration_getAnalyzeResultLog

```
size_t MU_Calibration_getAnalyzeResultLog(
    const MU_CalibrationAnalyzeResult* analyzeResult,
    char* logDest,
    size_t logDestSize,
    unsigned int logLevel);
```

Get a custom analyze result log message.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.
logDest	out	Pointer to a c string buffer for the log message.
logDestSize	in	Maximum size of the logDest buffer.
logLevel	in	Set of MU_CALIBRATION_ANALYZE_RESULT_LOG_* defines concatenated with a logical OR.

Define	Value
MU_CALIBRATION_ANALYZE_RESULT_LOG_ERRORS	1
MU_CALIBRATION_ANALYZE_RESULT_LOG_WARNINGS	2
MU_CALIBRATION_ANALYZE_RESULT_LOG_NOTE	4
MU_CALIBRATION_ANALYZE_RESULT_LOG_ALL	7

Return Value
Number of characters that will be written to logDest. Or in case logDest is NULL,the length of the log message (required array size - 1 for logDest)

See also
MU_Calibration_analyzeRawData

MU_Calibration_numberOfCalculatedMasterPeriods

```
int MU_Calibration_numberOfCalculatedMasterPeriods(
    const MU_CalibrationAnalyzeResult* analyzeResult);
```

Returns the number of calculated master periods.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.

Return Value
Number of calculated master periods.

See also
MU_Calibration_analyzeRawData

MU_Calibration_numberOfRevolutions

```
double MU_Calibration_numberOfRevolutions(
    const MU_CalibrationAnalyzeResult* analyzeResult);
```

Returns the number of acquired revolutions.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.

Return Value
Number of revolutions.

See also
MU_Calibration_analyzeRawData

MU_Calibration_numberOfAcquiredMasterPeriods

```
double MU_Calibration_numberOfAcquiredMasterPeriods(
    const MU_CalibrationAnalyzeResult* analyzeResult);
```

Returns the number of acquired master periods.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.

Return Value
Number of acquired master periods.

See also
`MU_Calibration_analyzeRawData`

MU_Calibration_minimalNumberOfSamplesPerMasterPeriod

```
double MU_Calibration_minimalNumberOfSamplesPerMasterPeriod(
    const MU_CalibrationAnalyzeResult* analyzeResult);
```

Returns the minimal acquired number of samples per master period.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.

Return Value
Minimal number of samples per master period.

See also
`MU_Calibration_analyzeRawData`

Analog

MU_Calibration_adjustAnalogByAnalyzeResult

```
bool MU_Calibration_adjustAnalogByAnalyzeResult(
    MU_Calibration* calibration,
    const MU_CalibrationAnalyzeResult* analyzeResult);
```

Adjusts the analog parameters in the calibration object calculated by the **`MU_Calibration_analyzeRawData`** function.

Parameter	Direction	Description
calibration	in	Pointer to a MU_Calibration object.
analyzeResult	in	Pointer to an MU_CalibrationAnalyzeResult object.

Return Value
`true`: succeeded, `false`: failed.

See also
MU_Calibration_analyzeRawData
MU_setCalibration

MU_Calibration_isAnalogAnalyzeResultAdjustable

```
bool MU_Calibration_isAnalogAnalyzeResultAdjustable(
    const MU_Calibration* calibration,
    const MU_CalibrationAnalyzeResult* analyzeResult,
    char* adjustmentMessage,
    size_t adjustmentMessageBufferSize,
    unsigned int adjustmentMessageLogLevel);
```

Returns whether an analog analysis result is adjustable. A adjustment message with a configurable output depth is an optional output.

Parameter	Direction	Description
calibration	in	Pointer to a MU_Calibration object.
analyzeResult	in	Pointer to an MU_CalibrationAnalyzeResult object.
adjustmentMessage	out	Pointer to a c string buffer destination for the adjustment message.
adjustmentMessageBufferSize	in	Maximum size of the adjustmentMessage buffer.
adjustmentMessageLogLevel	in	Set of MU_ADJUSTMENT_MESSAGE_LOG_* defines concatenated with a logical OR.

Define	Value
MU_ADJUSTMENT_MESSAGE_LOG_ERRORS	1
MU_ADJUSTMENT_MESSAGE_LOG_WARNINGS	2
MU_ADJUSTMENT_MESSAGE_LOG_NOTE	4
MU_ADJUSTMENT_MESSAGE_LOG_ALL	7

Return Value
`true` = adjustable, `false` = not adjustable

See also
MU_createCalibration
MU_getCalibration
MU_Calibration_analyzeRawData

MU_Calibration_isAnalogAnalysesValid

```
bool MU_Calibration_isAnalogAnalysesValid(
    const MU_CalibrationAnalyzeResult* analyzeResult);
```

Returns whether an analog analysis is valid.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.

Return Value
`true` = valid, `false` = invalid.

See also
MU_Calibration_analyzeRawData

MU_Calibration_getRelativeMasterTrackAdjustments

```
void MU_Calibration_getRelativeMasterTrackAdjustments(
    const MU_CalibrationAnalyzeResult* analyzeResult,
    MU_Calibration_RelativeAnalogTrackAdjustments* dest);
```

Returns the calculated relative master track adjustments.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.
dest	out	Output location.

See also
MU_Calibration_analyzeRawData

MU_Calibration_getRelativeNoniusTrackAdjustments

```
void MU_Calibration_getRelativeNoniusTrackAdjustments(
    const MU_CalibrationAnalyzeResult* analyzeResult,
    MU_Calibration_RelativeAnalogTrackAdjustments* dest);
```

Returns the calculated relative nonius track adjustments.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.
dest	out	Output location.

See also
MU_Calibration_analyzeRawData

Nonius

MU_Calibration_getOptimizedNoniusTrackOffsetTable

```
void MU_Calibration_getOptimizedNoniusTrackOffsetTable(
    const MU_CalibrationAnalyzeResult* analyzeResult,
    MU_Calibration_NoniusTrackOffsetTable* dest);
```

Returns the optimized nonius track offset table.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.
dest	out	Output location.

See also
MU_Calibration_analyzeRawData

MU_Calibration_isNoniusAnalysesValid

```
bool MU_Calibration_isNoniusAnalysesValid(
    const MU_CalibrationAnalyzeResult* analyzeResult);
```

Returns whether an nonius analysis is valid.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.

Return Value
true = valid, false = invalid.

See also
MU_Calibration_analyzeRawData

MU_Calibration_numberOfNoniusCurveSamples

```
size_t MU_Calibration_numberOfNoniusCurveSamples(
    const MU_CalibrationAnalyzeResult* analyzeResult);
```

Returns the number of calculated nonius curve samples.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.

Return Value
Number of nonius curve samples.

See also
MU_Calibration_analyzeRawData

MU_Calibration_noniusPhaseError

```
const long* MU_Calibration_noniusPhaseError(
    const MU_CalibrationAnalyzeResult* analyzeResult);
```

Returns a pointer to an array filled with nonius phase error data. The size of the array can be calculated with **MU_Calibration_numberOfNoniusCurveSamples**

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.

Return Value
Array of nonius phase error data.

See also
MU_Calibration_analyzeRawData

MU_Calibration_noniusTrackOffsetCurve

```
const long* MU_Calibration_noniusTrackOffsetCurve(
    const MU_CalibrationAnalyzeResult* analyzeResult);
```

Returns a pointer to an array filled with nonius track offset curve data. The size of the array can be calculated with **MU_Calibration_numberOfNoniusCurveSamples**

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.

Return Value
Array of nonius track offset curve data.

See also
MU_Calibration_analyzeRawData

MU_Calibration_noniusPhaseMargin

```
const long* MU_Calibration_noniusPhaseMargin(
    const MU_CalibrationAnalyzeResult* analyzeResult);
```

Returns a pointer to an array filled with nonius phase margin data. The size of the array can be calculated with **MU_Calibration_numberOfNoniusCurveSamples**.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.

Return Value
Array of nonius phase margin data.

See also
MU_Calibration_analyzeRawData

MU_Calibration_calculateNoniusPosition

```
size_t MU_Calibration_calculateNoniusPosition(
    const MU_CalibrationAnalyzeResult* analyzeResult,
    double* dest,
    size_t destBufferSize,
    MU_Calibration_Unit unit,
    bool continuous);
```

Calculates the positions for the X-axis of the nonius curves (PhaseError, TrackOffset, PhaseMargin).

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.
dest	out	Output location.
destBufferSize	in	Maximum size of the dest buffer.
unit	in	Unit of the position data, see enumeration `MU_UnitEnum` (Table 19 on page 82).
continuous	in	`true` = Map the calculated Nonius curve to a continuous angle range. For multiple rotations or absolute measurement lengths the curves are appended to each other. `false` = Map the calculated Nonius curve to 0° - 360° according to the singleturn position. For multiple rotations or absolute measurement lengths the Nonius curves are superimposed.

Return Value
Number of values that will be written to dest. Or in case dest is NULL, the required buffer size for dest.

See also
MU_Calibration_analyzeRawData

MU_Calibration_noniusPhaseMarginMax

```
long MU_Calibration_noniusPhaseMarginMax(
    const MU_CalibrationAnalyzeResult* analyzeResult);
```

Returns the maximum nonius phase margin.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.

Return Value
Maximum nonius phase margin.

See also
MU_Calibration_analyzeRawData

MU_Calibration_noniusPhaseMarginMin

```
long MU_Calibration_noniusPhaseMarginMin(
    const MU_CalibrationAnalyzeResult* analyzeResult);
```

Returns the minimum nonius phase margin.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.

Return Value
Minimum nonius phase margin.

See also
MU_Calibration_analyzeRawData

MU_Calibration_noniusPhaseRangeLimit

```
long MU_Calibration_noniusPhaseRangeLimit(
    const MU_CalibrationAnalyzeResult* analyzeResult);
```

Returns the nonius phase range limit.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.

Return Value
Nonius phase range limit.

See also
MU_Calibration_analyzeRawData

MU_Calibration_noniusUpperPhaseMargin

```
double MU_Calibration_noniusUpperPhaseMargin(
    const MU_CalibrationAnalyzeResult* analyzeResult);
```

Returns the upper nonius phase margin.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.

Return Value
Upper nonius phase margin.

See also
MU_Calibration_analyzeRawData

MU_Calibration_noniusLowerPhaseMargin

```
double MU_Calibration_noniusLowerPhaseMargin(
    const MU_CalibrationAnalyzeResult* analyzeResult);
```

Returns the lower nonius phase margin.

Parameter	Direction	Description
analyzeResult	in	Pointer to a MU_CalibrationAnalyzeResult object.

Return Value
Lower nonius phase margin.

See also
MU_Calibration_analyzeRawData

MU_getNoniusTrackOffsetTableParameters

```
void MU_getNoniusTrackOffsetTableParameters(
    const MU_Calibration_NoniusTrackOffsetTable* noniusTrackOffsetTable,
    MU_NoniusTrackOffsetTableParameters* dest);
```

Converts nonius track offset table values to parameter values.

Parameter	Direction	Description
noniusTrackOffsetTab	in	Pointer to an MU_Calibration_NoniusTrackOffsetTable object.
dest	out	Output location.

MU_getNoniusTrackOffsetTableByParameters

```
void MU_getNoniusTrackOffsetTableByParameters(
    const MU_NoniusTrackOffsetTableParameters* trackOffsetTableParameters,
    MU_Calibration_NoniusTrackOffsetTable* dest);
```

Convert nonius track offset parameter values to table values.

Parameter	Direction	Description
trackOffsetTableParam	in	Pointer to an MU_NoniusTrackOffsetTableParameters object.
dest	out	Output location.

Multiturn Calibration

MU_acquireMtSyncData

```
MU_Error MU_acquireMtSyncData(MU_Handle handle,
                              MU_MtSyncData* syncData,
                              size_t nSamples);
```

Acquires singleturn, multiturn and synchronization bits via the selected interface.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
syncData	out	Pointer to an MU_MtSyncData object
nSamples	in	Number of samples to be acquire.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_activateMtSyncConfig

MU_activateMtSyncConfig

MU_Error **MU_activateMtSyncConfig** (MU_Handle *handle*);

Configures the iC-MU according to the present multiturn configuration. This is needed to read the synchronization bits of the multiturn together with the singleturn and multiturn data. The iC-MU will be set to SSI and the SSI-frequency to 142.9 kHz. The parameter changes can be seen in table 2 page 64.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

Chip Parameter	Value	Description
eMU150_OUT_MSB		depending on ST, MT and Sync configuration
eMU150_OUT_ZERO	0x00	No additional zeros needed for SSI.
eMU150_TEST	0x00	deactivate all test modes
eMU150_MODE_ST	0x00	ST mode to absolute
eMU150_MODE_MT	0x00	Deactivate Multiturn
eMU150_CHK_MT	0x00	MT counter verification deactivated
eMU150_GSSI	0x00	SSI data output format binary
eMU150_RSSI	0x01	Enable SSI Ring mode
eMU150_MODEA	0x05	Activate SSI with error low (MODEA 0x05-0x07 mandatory for RSSI == 1)
eMU150_GET_MT	0x01	Activate direct communication to MT
Config Parameter	**Value**	**Description**
eMU150_READ_GAIN_ENABLE	0x00	Deactivate READ GAIN
eMU150_READ_STATUS_ENABLE	0x00	Deactivate READ STATUS
eMU150_FREQ_AGS	0x91	AGS set to 562.5 us
eMU150_FREQ_SSI	0x16	142.9 kHz, 156kHz not available in MBxU settings

Table 2: iC-MU Mt sync configuration

MU_deactivateMtSyncConfig

```
MU_Error MU_deactivateMtSyncConfig(MU_Handle handle);
```

Restores the iC-MU configuration stored in **MU_activateMtSyncConfig**.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

MU_acquire3TrackMtSyncData

```
MU_Error MU_acquire3TrackMtSyncData(MU_Handle handle,
                                    MU_MtSyncData* syncData,
                                    size_t nSamples);
```

Acquires singleturn, multiturn and synchronization bits via the selected interface.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
syncData	out	Pointer to an MU_MtSyncData object
nSamples	in	Number of samples to be acquire.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_activate3TrackMtSyncConfig
MU_deactivate3TrackMtSyncConfig

MU_activate3TrackMtSyncConfig

MU_Error **MU_activate3TrackMtSyncConfig** (MU_Handle *handle*);

Configures the iC-MU according to the present multiturn configuration. This is needed to read the synchronization bits of the multiturn together with the singleturn and multiturn data. The clock frequency is set to 142.9 kHz. The parameter changes can be seen in table 3 page 66.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration MU_ErrorEnum (see Table 10 on page 76). More information about the error are available with the function **MU_GetLastError** (see page 27).

See also
MU_acquire3TrackMtSyncData
MU_deactivate3TrackMtSyncConfig

Chip Parameter	Value	Description
eMU150_OUT_MSB		depending on ST, MT and Sync configuration
eMU150_OUT_ZERO	0x00	No additional zeros needed for SSI.
eMU150_TEST	0x00	deactivate all test modes
eMU150_MODE_ST	0x00	ST mode to absolute
eMU150_MODE_MT	0x00	Deactivate Multiturn
eMU150_CHK_MT	0x00	MT counter verification deactivated
eMU150_GSSI	0x00	SSI data output format binary
eMU150_RSSI	0x01	Enable SSI Ring mode
eMU150_GET_MT	0x01	Activate direct communication to MT
Config Parameter	**Value**	**Description**
eMU150_READ_GAIN_ENABLE	0x00	Deactivate READ GAIN
eMU150_READ_STATUS_ENABLE	0x00	Deactivate READ STATUS
eMU150_FREQ_AGS	0x91	AGS set to 562.5 us
eMU150_FREQ_SSI	0x16	142.9 kHz, 156kHz not available in MBxU settings

Table 3: iC-MU 3-Track Mt sync configuration

MU_deactivate3TrackMtSyncConfig

`MU_Error` **`MU_deactivate3TrackMtSyncConfig`** (`MU_Handle` *`handle`*);

Restores the iC-MU configuration stored in **`MU_activate3TrackMtSyncConfig`**.

Parameter	Direction	Description
`handle`	in	Handle of a Virtual Chip Object.

Return Value
On success, returns 0 (`MU_OK`).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

See also
`MU_acquire3TrackMtSyncData`
`MU_activate3TrackMtSyncConfig`

MU_getMtSync

`MU_MtSync*` **`MU_getMtSync`** (`MU_Handle` *`handle`*);

Creates a new instance of a multiturn synchronization object initialized with the Virtual Chip Object properties.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.

Return Value
Pointer to a MU_MtSync object.
In case of an fatal error returns `NULL`.

MU_createMtSync

```
MU_MtSync* MU_createMtSync(const MU_Calibration* muCalibration,
                           size_t nMtSyncBits,
                           bool mtMovementIsReverse);
```

Creates a new default instance of a multiturn synchronization object. The properties have to be set manually.

Parameter	Direction	Description
muCalibration	in	Pointer to a MU_Calibration object.
nMtSyncBits	in	Number of multiturn synchronization bits.
mtMovementIsReverse	in	Multiturn movement direction is reverse to the singleturn direction.

Return Value
Pointer to a MU_MtSync object.
In case of an error returns `NULL`.

MU_MtSync_delete

```
void MU_MtSync_delete(MU_MtSync* mtSync);
```

Deletes a multiturn synchronization object.

Parameter	Direction	Description
mtSync	in	Pointer to a MU_MtSync object.

MU_MtSync_calculatePosition

```
size_t MU_MtSync_calculatePosition(
    const MU_MtSync* mtSync,
    double* dest,
    const MU_MtSyncData* syncData,
    size_t nDataSamples,
    uint8_t mtTrackOffset,
    uint64_t resolution,
    MU_Calibration_Unit unit,
    bool continuous);
```

Calculates the positions for the X-axis of the multiturn synchronization (OffsetError)

Parameter	Direction	Description
mtSync	in	Pointer to a MU_MtSync object.
dest	out	Output location.
syncData	out	Pointer to a multiturn synchronization data buffer.
nDataSamples	in	Number of data samples.
mtTrackOffset	in	Multiturn track offset (SPO_MT).
resolution	in	System resolution.
unit	in	Unit of the position data, see enumeration `MU_UnitEnum` (Table 19 on page 82).
continuous	in	`true` = Map the calculated Nonius curve to a continuous angle range. For multiple rotations or absolute measurement lengths the curves are appended to each other. `false` = Map the calculated Nonius curve to 0° - 360° according to the singleturn position. For multiple rotations or absolute measurement lengths the Nonius curves are superimposed.

Return Value
Number of values that will be written to dest. Or in case dest is NULL, the required buffer size for dest.

Multiturn Calibration Result

MU_MtSync_analyzeData

```
MU_MtAnalyzeResult* MU_MtSync_analyzeData(const MU_MtSync* mtSync,
                                          const MU_MtSyncData* syncData,
                                          size_t nDataSamples);
```

Analyzes multiturn synchronization data samples and returns a pointer to a MU_MtAnalyzeResult object.

Parameter	Direction	Description
mtSync	in	Pointer to a MU_MtSync object.
syncData	out	Pointer to a multiturn synchronization data buffer.
nDataSamples	in	Number of data samples.

Return Value
Pointer to a MU_MtAnalyzeResult object.
In case of an error returns `NULL`.

See also
`MU_createMtSync`

MU_MtAnalyzeResult_delete

```
void MU_MtAnalyzeResult_delete(MU_MtAnalyzeResult* analyzeResult);
```

Deletes a MU_MtAnalyzeResult object.

Parameter	Direction	Description
analyzeResult	in	Pointer to an MU_MtAnalyzeResult object.

MU_updateMtSync

```
MU_Error MU_updateMtSync(MU_Handle handle,
                         const MU_MtAnalyzeResult* analyzeResult);
```

Updates SPO_MT in a Virtual Chip Object from a multiturn synchronization object.

Parameter	Direction	Description
handle	in	Handle of a Virtual Chip Object.
analyzeResult	in	Pointer to an MU_MtAnalyzeResult object.

Return Value
On success, returns 0 (MU_OK).
On failure or warning, returns a value >0, the value is equal to the index of the error enumeration `MU_ErrorEnum` (see Table 10 on page 76). More information about the error are available with the function **`MU_GetLastError`** (see page 27).

MU_MtAnalyzeResult_optimalSpoMt

```
uint8_t MU_MtAnalyzeResult_optimalSpoMt(const MU_MtAnalyzeResult* analyzeResult);
```

Returns the calcualted SPO_MT value.

Parameter	Direction	Description
analyzeResult	in	Pointer to an MU_MtAnalyzeResult object.

Return Value
Calculated SPO_MT parameter value.

MU_MtAnalyzeResult_calculateOffsetError

```
size_t MU_MtAnalyzeResult_calculateOffsetError(
    const MU_MtAnalyzeResult* analyzeResult,
    double* dest,
    size_t destBufferSize,
    uint8_t mtTrackOffset);
```

Calculates the multiturn offset error by a given SPO_MT value.

Parameter	Direction	Description
analyzeResult	in	Pointer to an MU_MtAnalyzeResult object.
dest	out	Output location.
destBufferSize	in	Capacity (maximum size) of the dest buffer.
mtTrackOffset	in	Multi turn track offset (SPO_MT).

Return Value
Number of values that will be written to dest. Or in case dest is NULL, the required buffer size for dest.

MU_MtAnalyzeResult_maxOffsetError

```
double MU_MtAnalyzeResult_maxOffsetError(
    const MU_MtAnalyzeResult* analyzeResult,
    uint8_t mtTrackOffset);
```

Returns the maximum multiturn offset error by a given SPO_MT value.

Parameter	Direction	Description
analyzeResult	in	Pointer to an MU_MtAnalyzeResult object.
mtTrackOffset	in	Multi turn track offset (SPO_MT).

Return Value
Maximum offset error.

MU_MtAnalyzeResult_minOffsetError

```
double MU_MtAnalyzeResult_minOffsetError(
    const MU_MtAnalyzeResult* analyzeResult,
    uint8_t mtTrackOffset);
```

Returns the minimum multiturn offset error by a given SPO_MT value.

Parameter	Direction	Description
analyzeResult	in	Pointer to an MU_MtAnalyzeResult object.
mtTrackOffset	in	Multiturn track offset (SPO_MT).

Return Value
Minimum offset error.

MU_MtAnalyzeResult_offsetErrorRangeLimit

```
double MU_MtAnalyzeResult_offsetErrorRangeLimit(const MU_MtAnalyzeResult* analyzeResult);
```

Returns the multiturn offset error limit.

Parameter	Direction	Description
analyzeResult	in	Pointer to an MU_MtAnalyzeResult object.

Return Value
Offset error range limit.

MU_MtAnalyzeResult_upperOffsetErrorMargin

```
double MU_MtAnalyzeResult_upperOffsetErrorMargin(
    const MU_MtAnalyzeResult* analyzeResult,
    uint8_t mtTrackOffset);
```

Returns the upper multiturn offset margin by a given SPO_MT value.

Parameter	Direction	Description
analyzeResult	in	Pointer to an MU_MtAnalyzeResult object.
mtTrackOffset	in	Multi turn track offset (SPO_MT).

Return Value
Upper offset error margin.

MU_MtAnalyzeResult_lowerOffsetErrorMargin

```
double MU_MtAnalyzeResult_lowerOffsetErrorMargin(
    const MU_MtAnalyzeResult* analyzeResult,
    uint8_t mtTrackOffset);
```

Returns the lower multiturn offset margin by a given SPO_MT value.

Parameter	Direction	Description
analyzeResult	in	Pointer to an MU_MtAnalyzeResult object.
mtTrackOffset	in	Multi turn track offset (SPO_MT).

Return Value
Lower offset error margin.

PARAMETER AND ERROR CODING

Handle to Virtual Chip Object

The `MU_Handle` is an opaque pointer to the internal struct `MU_Handle_`.

```
typedef struct MU_Handle_* MU_Handle;
```

Calibration Object

The `MU_Calibration` is an opaque pointer to a internal object.

```
typedef struct MU_Calibration_ MU_Calibration;
```

Calibration Analysis Result Object

The `MU_CalibrationAnalyzeResult` is an opaque pointer to a internal object.

```
typedef struct MU_CalibrationAnalyzeResult_ MU_CalibrationAnalyzeResult;
```

Multi Turn Synchronization Object

The `MU_MtSync` is an opaque pointer to a internal object.

```
typedef struct MU_MtSync_ MU_MtSync;
```

Multi Turn Synchronization Analysis Result Object

The `MU_MtAnalyzeResult` is an opaque pointer to a internal object.

```
typedef struct MU_MtAnalyzeResult_ MU_MtAnalyzeResult;
```

ReadSensStruct

Element	Description
mt	Multiturn value
st	Singleturn value
err	Error bit value
warn	Warning bit value
rawMaster	Raw data of master track
rawNonius	Raw data of nonius track

Table 4: Definition of data type `ReadSensStruct`.

MU_MtSyncData

Element	Description
singleTurnPosition	Singleturn value
internalSingleTurnPosition	Singleturn value used for synchronization
normalizedExternalMultiTurnSyncBits	Sync Bits value from multiturn
normalizedInternalMultiTurnSyncBits	Sync Bits value used for synchronization
externalMultiTurnPosition	Multiturn value used for synchronization
multiTurnPosition	Multiturn value
singleCycleData	Full SCD data

Table 5: Definition of data type `MU_MtSyncData`.

MU_Calibration_RelativeAnalogTrackAdjustments

Element	Description
cosineGain_lsb	Cosine gain in relative parameter values (LSB) changes
sineOffset_lsb	Sine offset in relative parameter values (LSB) changes
cosineOffset_lsb	Cosine offset in relative parameter values (LSB) changes
phase_lsb	Phase in relative parameter values (LSB) changes

Table 6: Definition of data type `MU_Calibration_RelativeAnalogTrackAdjustments`.

MU_Calibration_AnalogTrackAdjustments

Element	Description
cosineGain	GX_{M; N}
sineOffset	VOSS_{M; N}
cosineOffset	VOSC_{M; N}
phase	PH_{M; N}
phaseRange	PHR_{M; N} (with iC-MU{128} not used; 0 only)

Table 7: Definition of data type `MU_Calibration_AnalogTrackAdjustments`.

MU_Calibration_NoniusTrackOffsetTable

Element	Description
spoBase	SPO_BASE value (-8..7)
spoN[16]	SPO_0..15 value (-8..7)

Table 8: Definition of data type `MU_Calibration_NoniusTrackOffsetTable`.

MU_NoniusTrackOffsetTableParameters

Element	Description
spoBase	SPO_BASE parameter value (0..15)
spoN[15]	SPO_{0..15} parameter value (0..15)

Table 9: Definition of data type `MU_NoniusTrackOffsetTableParameters`.

Error Description

Index	Enumerator	Error Message (Solution)
0	MU_OK	Function call successful (no error).
1	MU_INVALID_HANDLE	Use the handle returned by the MU_Open function.
2	MU_INTERFACEDRIVER_NOT_FOUND	Install the interface specific drivers.
3	MU_INTERFACE_NOT_FOUND	Check wire connections and power supply.
4	MU_INVALID_INTERFACE	Interface not specified.
5	MU_NO_INTERFACE_SELECTED	Select an interface before using this function.
6	MU_INVALID_PARAMETER	Parameter not specified.
7	MU_INVALID_ADDRESS	Address out of range.
8	MU_INVALID_VALUE	Value out of range.
9	MU_USB_ERROR	Check the USB connection to the specific interface.
10	MU_FILE_NOT_FOUND	Check if file exists.
11	MU_INVALID_FILE	Check file format.
12	MU_SPI_ERROR	Sensor data was invalid on readout.
13	MU_SPI_DISMISS	Address refused.
14	MU_SPI_FAIL	Data request has failed.
15	MU_SPI_BUSY_TIMEOUT	Slave is busy with a request.
16	MU_VERIFY_FAILED	The verification failed, check the device connection.
17	MU_MASTERCOMM_FAILED	Failed communication to BiSS master, check the selected interface.
18	MU_BISSCOMM_FAILED	BiSS communication failed, check wire connections and power supply.
19	MU_INVALID_BISSMASTER	Connected BiSS-Master not compatible, connect a BiS-S-Adapter with min. iC-MB3.
20	MU_USB_HIGHSPEED_WARNING	High speed USB device plugged into non high speed USB hub.
21	MU_FAST_ROTATION	Rotation speed is too fast for calibration. Reduce rotation speed and/or increase sampling rate.
22	MU_SLOW_ROTATION	Rotation speed is too slow for calibration. Increase rotation speed and/or increase samples to measure.
23	MU_FILE_ACCESS_DENIED	Insufficient permission to access file.
24	MU_READPARAM_SSI	Using MU_ReadParam is not possible in SSI-mode. Use MU_SwitchToBiSS.
25	MU_SSIRING_ERROR	SSI Ring transmission failed (different data words).

Index	Enumerator	Error Message (Solution)
26	MU_SEMI_ROTATION	Calibration successful, but rotation speed is too slow to acquire one complete turn.
27	MU_BISS_REGERROR	Invalid address and/or access denied.
28	MU_INTERNAL_CALIB_ERROR	Reserved.
29	MU_INVALID_CONFIGURATION	iC-MU signaled a configuration error. Use MU_WriteParams to write a valid configuration.
30	MU_CALIBRATION_FAILED	An error occurred during the calibration procedure.
31	MU_ACQUISITION_FAILED	An error occurred during data acquisition.
32	MU_GAIN_LIMIT	The calculated value change of the gain parameter is out of range.
33	MU_OFFSET_LIMIT	The calculated value change of the offset parameter is out of range.
34	MU_PHASE_LIMIT	The calculated value change of the phase parameter is out of range.
35	MU_BAD_CAL_DATA	Poor quality of acquired raw analog signals. A more constant velocity is needed for calibration.
36	MU_I2C_COMM_FAILED	The communication to the chosen I2C device failed.
37	MU_USB_DATA_LOSS	Slow down the frame repetition rate (MU_FREQ_AGS) and/or reduce the system load of your PC.
38	MU_MT_SYNC_FAILED	MT Sync Bits do not match the iC-MU singleturn position. Possible error reasons: direction of rotation (DOR) of the multiturn does not fit the singleturn DOR; the singleturn and/or multiturn had a position error; or the communication was disturbed during acquisition (no CRC during SSI communication).
39	MU_UNKNOWN_ERROR	An unexpected error has occurred.
40	MU_UNKNOWN_REVISION	The chip revision (0xXX) is not known. It is strongly recommended not to use this software version with the connected chip revision.
41	MU_UNSUPPORTED_REVISION	The chip revision (0xXX) is not supported.
42	MU_UNKNOWN_PARAMETER	The parameter (X) is not known.
43	MU_PARAMETER_NOT_IN_REVISION	The chip revision (0xXX) does not know the parameter (X).
44	MU_REVISION_NOT_SET	Chip revision not set.
45	MU_UNSUPPORTED_CHIP	Chip revision unsupported.
46	MU_CONTRADICTORY_REVISIONS	The chip revision (0xXX) used by the software does not match the revision of the chip. The software may work incorrectly.
47	MU_FRAME_RATE_NOT_SUPPORTED	The selected frame rate is not supported by the interface and/or the function.
48	MU_CLOCK_FREQUENCY_NOT_SUPPORTED	The selected clock frequency is not supported by the interface and/or the function.
49	MU_FRAME_CYCLE_TIME_TO_SHORT	The frame cycle time is to short.

Table 10: Error Definitions (`MU_ErrorEnum`)

Error Type Definitions

Index	Error Type Enumerator	Description
0	MU_NONE	Function call successful.
1	MU_HINT	Hint that might improve usage.
2	MU_WARNING	Warning without affecting operation.
3	MU_PROGRAMMING_ERROR	Incorrect access to the chip.
4	MU_OPERATING_ERROR	Incorrect handling.
5	MU_COMMUNICATION_ERROR	Communication with the chip is disturbed.

Table 11: Error Type Definitions (`MU_ErrorTypeEnum`)

Interface Types

Index	Enumerator	Description
0	MU_NO_INTERFACE	no device
1	MU_MB3U_SPI	USB to SPI adapter
2	MU_MB3U_BISS	USB to BiSS adapter
3	MU_MB4U	USB to BiSS adapter
4	MU_MB5U	USB to BiSS adapter

Table 12: Supported interfaces (`MU_InterfaceEnum`)

iC-MU Series Configuration Parameters

A detailed description of each parameter is available in the respective chip data sheet. The parameter name is the last part of the enumeration name after the library prefix ("MU_").

Index	Enumerator	Comment
0	MU_GF_M	
1	MU_GC_M	
2	MU_GX_M	
3	MU_VOSS_M	
4	MU_VOSC_M	
5	MU_PH_M	
6	MU_PHR_M	iC-MU150 and iC-MU200 only
7	MU_CIBM	
8	MU_ENAC	
9	MU_GF_N	
10	MU_GC_N	
11	MU_GX_N	
12	MU_VOSS_N	
13	MU_VOSC_N	

Index	Enumerator	Comment
14	MU_PH_N	
15	MU_PHR_N	iC-MU150 and iC-MU200 only
16	MU_MODEA	
17	MU_NTOA	iC-MU150 and iC-MU200 only
18	MU_MODEB	
19	MU_CFGEW	
20	MU_EMTD	
21	MU_ACRM_RES	
22	MU_NCHK_NON	
23	MU_NCHK_CRC	
24	MU_ACC_STAT	
25	MU_FILT	
26	MU_LIN	
27	MU_ESSI_MT	
28	MU_ROT_MT	
29	MU_MPC	
30	MU_SPO_MT	
31	MU_MODE_MT	
32	MU_SBL_MT	
33	MU_CHK_MT	
34	MU_GET_MT	
35	MU_OUT_MSB	
36	MU_OUT_ZERO	
37	MU_OUT_LSB	
38	MU_MODE_ST	
39	MU_RSSI	
40	MU_GSSI	
41	MU_RESABZ	
42	MU_FRQAB	
43	MU_ENIF_AUTO	
44	MU_SS_AB	
45	MU_ROT_ALL	Name in iC-MU data sheet: ROT
46	MU_INV_Z	
47	MU_INV_B	
48	MU_INV_A	
49	MU_PP60UVW	
50	MU_CHYS_AB	
51	MU_LENZ	
52	MU_PPUVW	
53	MU_RPL	
54	MU_TEST	

Index	Enumerator	Comment
55	MU_ROT_POS	iC-MU150 and iC-MU200 only
56	MU_OFF_ABZ	
57	MU_OFF_POS	
58	MU_OFF_COM	
59	MU_PA0_CONF	
60	MU_AFGAIN_M	
61	MU_ACGAIN_M	
62	MU_AFGAIN_N	
63	MU_ACGAIN_N	
64	MU_EDSBANK	
65	MU_PROFILE_ID	
66	MU_SERIAL	
67	MU_OFF_UVW	
68	MU_PRES_POS	
69	MU_SPO_BASE	
70	MU_SPO_0	
71	MU_SPO_1	
72	MU_SPO_2	
73	MU_SPO_3	
74	MU_SPO_4	
75	MU_SPO_5	
76	MU_SPO_6	
77	MU_SPO_7	
78	MU_SPO_8	
79	MU_SPO_9	
80	MU_SPO_10	
81	MU_SPO_11	
82	MU_SPO_12	
83	MU_SPO_13	
84	MU_SPO_14	
85	MU_SPO_15	
86	MU_RPL_RESET	
87	MU_I2C_DEV_START	
88	MU_I2C_RAM_START	
89	MU_I2C_RAM_END	
90	MU_I2C_DEVID	
91	MU_I2C_RETRY	
92	MU_HARD_REV	
93	MU_DEV_ID0	
94	MU_DEV_ID1	
95	MU_DEV_ID2	

Index	Enumerator	Comment
96	MU_DEV_ID3	
97	MU_DEV_ID4	
98	MU_DEV_ID5	
99	MU_MFG_ID0	
100	MU_MFG_ID1	
101	MU_CRC16	
102	MU_CRC8	
103	MU_AM_MIN	
104	MU_AM_MAX	
105	MU_AN_MIN	
106	MU_AN_MAX	
107	MU_STUP	
108	MU_CMD_EXE	
109	MU_FRQ_CNV	
110	MU_FRQ_ABZ	
111	MU_NON_CTR	
112	MU_MT_CTR	
113	MU_MT_ERR	
114	MU_EPR_ERR	
115	MU_CRC_ERR	

Table 13: iC-MU Series Configuration Parameters (`MU_ParamEnum`)

iC-MU Series Chip Commands

A detailed description of each command is available in the respective chip data sheet. The command name is the last part of the enumeration name after the library prefix ("MU_").

Index	Enumerator	Comment
0	MU_CMD_NO_FUNCTION	
1	MU_CMD_WRITE_ALL	
2	MU_CMD_WRITE_OFF	
3	MU_CMD_ABS_RESET	
4	MU_CMD_NON_VER	
5	MU_CMD_MT_RESET	
6	MU_CMD_MT_VER	
7	MU_CMD_SOFT_RESET	
8	MU_CMD_SOFT_PRES	
9	MU_CMD_SOFT_E2P_PRES	
10	MU_CMD_I2C_COM	
11	MU_CMD_EVENT_COUNT	
12	MU_CMD_SWITCH	
13	MU_CMD_CRC_VER	

Index	Enumerator	Comment
14	MU_CMD_CRC_CALC	
15	MU_CMD_SET_MTC	
16	MU_CMD_RES_MTC	
17	MU_CMD_RESERVED0	
18	MU_CMD_MODEA_SPI	
19	MU_CMD_ROT_POS	
20	MU_CMD_ROT_POS_E2P	

Table 14: Supported chip commands (`MU_CommandEnum`)

Set/Write/Verify Flag

Index	Enumerator	Description
0	MU_SETONLY	Sets the parameter in the Virtual Chip Object only.
1	MU_WRITE	Sets the parameter in the Virtual Chip Object and writes the corresponding register to the chip immediately.
2	MU_VERIFY	Sets the parameter in the Virtual Chip Object, writes the corresponding register to the chip immediately, and reads the register from the chip for verification. If the verification fails, the corresponding function returns the error code MU_VERIFY_FAILED.

Table 15: Set/Write/Verification Flag (`MU_WriteVerifyEnum`)

Config Data Definitions

Value	Name	Description
0	MU_RESERVED0	Reserved
1	MU_FREQ_SPI	SPI frequency = $\frac{12MHz}{(MU\_FREQ\_SPI+1)*2}$
2	MU_MASTERVER	BiSS master version (read only)
3	MU_MASTERREV	BiSS master revision (read only)
4	MU_SLAVE_ID	Slave ID of the connected chip
5	MU_FREQ_SCD	BiSS master parameter FreqSens (see table 21)
6	MU_CLKENI	Enable internal clock of MB3U (MB3U-I2C) interface
7	MU_FREQ_AGS	BiSS master parameter FreqAgs (see table 22)
8	MU_SLAVE_COUNT	Number of connected slaves (read only)
9	MU_START_CONTROL_FRAME	Reserved use READ_STATUS_ENABLE and/or READ_GAIN_ENABLE instead.
10	MU_REG_END	Indicates the completed readout of the status/gain register via MU_START_CONTROL_FRAME. (read only)
11	MU_FREQ_SSI	BiSS master parameter FreqSens in SSI mode (see table 21)
12	MU_READ_STATUS_ENABLE	Enable cyclic read for **`MU_ReadSens`**: status register (0x76.. 0x77).
13	MU_USB_PERFORMANCE	Reserved

Value	Name	Description
14	MU_READ_GAIN_ENABLE	Enable cyclic read for **MU_ReadSens**: gain register (0x2B and 0x2F).
15	MU_BP	Calculates BiSS Profile at address 0x42 and 0x43 (value 0 = disabled / values 1 and 3: see EDS specification for details) accordingto the chip configuration parameters (e.g.MU_OUT_MSB).
16	MU_UPDATE_BISSID_ENABLE	Enables automatic BiSS-ID configuration (0x7B..0x7D) according to the chip configuration parameters (e.g. MU_OUT_MSB).
17	MU_ENABLE_TTL	Switch for adapter communication signals: 0 = differential RS422, 1 = single-ended TTL (MB4U & MB5U only).

Table 16: Config data definitions (`MU_ConfigDataEnum`)

EEPROM area

Index	Enumerator	Description
0	MU_E2P_CFG	Complete chip configuration including the BiSS registers (PROFILE_ID, DEV_ID, MFG_ID)

Table 17: EEPROM area (`MU_E2PAreaEnum`)

Interface Info

Index	Enumerator	Description
0	MU_SERIAL_NUMBER	Gets the serial number (MB4U/MB5U only)
1	MU_DRIVER_VERSION	The driver version.

Table 18: Interface info (`MU_InterfaceInfoEnum`)

Unit of position

Index	Enumerator	Description
0	MU_CALIBRATION_UNIT_RESOLUTION	Digits of resolution.
1	MU_CALIBRATION_UNIT_TURN	Turns (one turn = 1.0)
2	MU_CALIBRATION_UNIT_DEGREE	Degree angle (one turn = 360.0)
3	MU_CALIBRATION_UNIT_RAD	Rad angle (one turn = 2*PI)

Table 19: Unit of position (`MU_UnitEnum`)

Defines for the iC-MU revision ID

Define	Value
MU_REV_NONE	0x00

MU_REV_MU_Y	0x05
MU_REV_MU_Y1	0x06
MU_REV_MU_Y2	0x07
MU_REV_MU150_0	0x10
MU_REV_MU150_1	0x11
MU_REV_MU200_0	0x20

Table 20: Definition of the iC-MU revision IDs.

APPLICATION EXAMPLES

All examples can be found in the shared library package subdirectory **examples**.

APPENDIX

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

Table 21: Sensor Data Frequency

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

Table 22: AutoGetSens Frequency

REVISION HISTORY

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

Rev	Notes	Pages affected	Library reference
B1	Chapter CALIBRATION FUNCTIONS completely revised	43 ff.	Version 3.0.0

Table 23: Revision History