Programming manual

S-SDK-MSC15

Document created: 2021-9-9

Kapitel 1

Information

Please read this documentation and the disclaimer carefully before using the software.
By installing and using the software, you explicitly and fully acknowledge and agree to this.
Gigahertz-Optik GmbH reserves the right to make changes to this manual without prior notice.

1.1 Disclaimer

This software was developed with utmost care and thoroughly tested on different computers. No errors were noted for the approved product versions. However, it cannot be guaranteed that the software will work perfectly on all types of computers. Completely error-free software is not possible with the current technology level.
Gigahertz-Optik GmbH is not liable if the software does not perfectly fulfill your desired purpose or if it is incompatible with other software on your computer. You are therefore solely responsible for the choice, installation and use as well as for the intended results.
With the exception of damages caused deliberately, Gigahertz-Optik GmbH is not liable for any damages caused by the use or inability to use the software. This also exclusively applies for loss of business profits, business interruptions, loss of business information or any other economic losses, even if Gigahertz-Optik GmbH had been previously advised of the possibility of such damage. The enclosed documentation/help of the software is with no claim of accuracy or completeness.

1.2 Warranty

Gigahertz-Optik GmbH guarantees the delivery of all functions listed in the product description. Any available delivery media are free of any material defects.
We have taken all the necessary and possible steps that are required to keep this software free of viruses, spyware, the so-called "back door entrancesâ€ť or other harmful code. We do not collect any information about you or your data. We will not deliberately limit you from using the functions of this software or access to your data. This agreement supersedes any non-contractual assurances that we may have explained to you. Any modification to this agreement must be confirmed in writing by both parties.

1.3 License

A license for the full version allows you to use the product on only one workstation. Each concurrent use on another workstation requires an additional product license. The distribution of the product and documentation is prohibited. You are authorized to make a copy of this product for your backup purposes. You may pass on your own software that you have developed using this development package together with the required DLLs for this development package to third parties.

1.4 Overview

This development package provides you with all the tools required (no compiler or integrated software development environments) to directly control a MSC15 and CSS device type series measurement device from Gigahertz-Optik using C/C++. This is primarily with regards to the communication and control libraries for your MSC15 and CSS device type.
In order to use these libraries, you need a programming environment such as Microsoft Visual Studio, Embarcadero C++ builder, etc.

1.5 Contact information of Gigahertz-Optik

Gigahertz Optik GmbH	Gigahertz-Optik Inc
An der Kälberweide 12 D-82299 Türkenfeld Germany Tel.: +49 8193 93700-0 Fax: +49 8193 93700-50 Email: info@gigahertz-optik.de Homepage: http://www.gigahertz-optik.de	110 Haverhill Road Amesbury MA 01913 USA Tel: + 978 462 1818 Fax: + 978 462 3677 Email: b.angelo@gigahertz-optik.com Homepage: https://www.gigahertz-optik.com

1.6 System Requirements

To use the S-SDK MSC15 and CSS device type you have to consider the following points:

- Minimum disk space – approx. 10MB
- Operation system: MS Windows XP, MS Windows 7 (32bit/64bit), MS Windows 10 (32bit/64bit)
- C/C++ development environment such as MS Visual Studio, Embarcadero C++ Builder, etc. when programming with C/C++
- free USB port

1.7 Installation

Follow the steps below to install the MSC15 and CSS device type-SDK from the product CD:

- Read this documentation before you begin the installation

- Close all other applications before installing
- Insert the CD in your CD drive or unpack the supplied ZIP file.
- Copy the Gigahertz-Optik folder from the CD or ZIP file to a location of your choice. If you have already got other development packages from Gigahertz-Optik installed, it is recommended to use the same installation path in order to avoid possible conflicts.
- Add the folder "install dir/Gigahertz-Optik/runtime"t' to your system path. "install dir" hereby corresponds to the base path you added in step 4 above. If you have already got other development packages from Gigahertz-Optik installed, step 5 might not be necessary.

1.8 System preparation

Connect the MSC15 and CSS device type to your computer. The required drivers are standard windows drivers and will be installed automatically.
The SDK is password protected! Each programm created with this SDK has to call the function setPassword() before any other function can be called.

Kapitel 2

Programming example

Example - How to import DLL in your application

Note

The method descriptions (Module) provide examples of how to use the SDK methods.

2.1 C++

As we don't deliver import libraries for different development environments you have to use run-time dynamic linking to be able to use all methods provided by dll.
Following is an C++ example of how to import and use methods from DLL. The example does not include all available methods. The use of the handles is encapsulated in the class. The example searches and initializes a MSC15 and CSS device type, peforms a measurement and then records the results in the console.
At the end, all MSC15 and CSS device type-resources are released again.

2.1.1 MSC15Example.cpp

```
#include "MSC15Import.h
#include <iostream>
int main(int argc, char* argv[])
{
    MSC15Import msc15;
    //search for a MSC15 device
    //first you have to replace the right password in the msc15Import.cpp
    int error = msc15.init("MSC15_0
);
    if (error == 0)
    {
        //set measurement mode and start a new measurement
        error = msc15.measure();
        //if no error occured read the integral values
        if (error == 0)
        {
            double value;
            msc15.getPhotopic(&value);
            std::cout « Ë(V):
 « value « std::endl;
            msc15.getCCT(&value);
            std::cout « "CCT:
 « value « std::endl;
        }
        else
        {
            std::cout « ërror occured:
 « error « std::endl;
        }
        msc15.close();
```

```
    }
    else
    {
        std::cout « ërror occured:
 « error « std::endl;
    }
    system("PAUSE
");
}
```

2.1.2 MSC15Import.cpp

```
#include "MSC15Import.h
MSC15Import::MSC15Import()
{
    hDLLGOMSC15 = NULL;
    handle = -1;
}
MSC15Import:: MSC15Import()
{
}
int __stdcall MSC15Import::init(char* deviceName)
{
    int l_rc = 0;
    if (handle > 0)
        close();
    if (getProcAddresses(&hDLLGOMSC15, "GOMDMSC15.dll
", 6,
        &GOMDMSC15_setPassword, "GOMDMSC15_setPassword
",
        &GOMDMSC15_getHandle, "GOMDMSC15_getHandle
",
        &GOMDMSC15_releaseHandle, "GOMDMSC15_releaseHandle
",
        &GOMDMSC15_measure, "GOMDMSC15_measure
",
        &GOMDMSC15_getCCT, "GOMDMSC15_getCCT
",
        &GOMDMSC15_getPhotopic, "GOMDMSC15_getPhotopic
        ))
    {
        try {
            l_rc = GOMDMSC15_setPassword("passw
"); //replace passw with the right password
            if (l_rc == 0)
                l_rc = GOMDMSC15_getHandle(deviceName, &handle);
            if (handle > 0)
                std::cout « Ïnitialisation sucessfull
 « std::endl;
        }
        catch (...) {
            l_rc = -1;
        }
    }
    else {
        l_rc = -1;
    }
    return l_rc;
}
int __stdcall MSC15Import::measure()
{
    int l_rc = GOMDMSC15_measure(handle);
    return l_rc;
}
int __stdcall MSC15Import::getPhotopic(double* value)
{
    int l_rc = GOMDMSC15_getPhotopic(handle, value);
    return l_rc;
}
int __stdcall MSC15Import::getCCT(double* value)
{
    int l_rc = GOMDMSC15_getCCT(handle, value);
    return l_rc;
}
int __stdcall MSC15Import::close()
{
    int l_rc = GOMDMSC15_releaseHandle(handle);
    handle = -1;
    return l_rc;
}
bool __stdcall MSC15Import::getProcAddresses(HINSTANCE *p_hLibrary,
    const char* p_dllName, INT p_count, ...)
{
    va_list l_va;
```

```
    va_start(l_va, p_count);
    if ((*p_hLibrary = LoadLibrary(p_dllName)) != NULL)
    {
        FARPROC* l_procFunction = NULL;
        char* l_funcName = NULL;
        int l_idxCount = 0;
        while (l_idxCount < p_count)
        {
            l_procFunction = va_arg(l_va, FARPROC*);
            l_funcName = va_arg(l_va, LPSTR);
            if ((*l_procFunction =
                GetProcAddress(*p_hLibrary, l_funcName)) == NULL)
            {
                l_procFunction = NULL;
                return FALSE;
            }
            l_idxCount++;
        }
    }
    else
    {
        va_end(l_va);
        return false;
    }
    va_end(l_va);
    return true;
}
```

2.1.3 MSC15Import.h

```
#ifndef MSC15ImportH
#define MSC15ImportH
#include <Windows.h>
#include ßtdio.h
#include <iostream>
class MSC15Import
{
public:
    MSC15Import();
    virtual  MSC15Import();
    int __stdcall init(char* deviceName);
    int __stdcall close();
    int __stdcall getPhotopic(double* value);
    int __stdcall getCCT(double* value);
    int __stdcall measure();
private:
    int handle;
    HINSTANCE hDLLGOMSC15;
    bool __stdcall getProcAddresses(HINSTANCE *p_hLibrary, const char* p_dllName, int p_count, ...);
    int(__stdcall *GOMDMSC15_setPassword)(char* value);
    int(__stdcall *GOMDMSC15_getHandle)(char* device, int* handle);
    int(__stdcall *GOMDMSC15_releaseHandle)(int handle);
    int(__stdcall *GOMDMSC15_measure)(int handle);
    int(__stdcall *GOMDMSC15_getCCT)(int handle, double* value);
    int(__stdcall *GOMDMSC15_getPhotopic)(int handle, double* value);
};
#endif
```

2.2 More Examples

Further examples for integrating DLL's, can be found in the installation directory of the SDK.

Kapitel 3

Change History

A list of all modifications of the S-SDK-MSC15 follows:

- **V2016.1**: New: Release of the SDK
- **V2017.1**: New: Support for MSC15-W devices/ New: TM-30-15/ Update: bilirubin and Melanopic to the new standards.
- **V2017.2**: New: Range in pixel-based steps/ Update: adjusted maximum measurement time of the pulse measurement on the sample rate
- **V2018.1**: New: Support for CSS-45
- **V2019.1**: New: Load measurement data from internal logger/ New: Wavelength Correction
- **V2019.2**: Bugfix: color calculation
- **V2019.3**: New: Support for CSS-45-D, Bugfix: RS485 Communication
- **V2019.4**: Update: TM-30-15 to TM-30-18, Bugfix: Functionnames in Interface
- **V2019.5**: New: GOMDMSC15_measureWithManualIntegrationTime(), Bugfix: Error with firmware $>$ 1.45,
- **V2019.6**: Bugfix: GOMDMSC15_getLastIntegrationTime(), New: GOMDMSC15_getTemperature()
- **V2019.7**: Update: Calculation of measurment values will be done in DLL (faster)
- **V2020.1**: Update: Melanopsin after CIE S026
- **V2020.2**: New: Support for MSC15-Bili V01
- **V2020.3**: Update: User WL-Ranges of MSC15-W
- **V2020.4**: Update: Measurement error: no/low signal
- **V2021.1**: Update: set RS485 baudrate of CSS-45/ Bugfix: internal Logger and RTC
- **V2021.2**: New: complementary wavelength/ Bugfix: dominant wavelength and CRI calculation

Kapitel 4

Errors and Warnings

A list of errors and warnings follows:

4.1 Errors

- -500: Communication Error
- -25000: Communication Problems
- -25001: Setup file invalid for device
- -25002: Setup file could not be opened
- -25004: Not a valid handle
- -25005: Communication channel cannot be initialized
- -25006: Too low Firmware Version
- -25007: Problem sending data
- -25008: Problem with data receive
- -25009: device sends an unspecified error
- -25012: Units do not match
- -25013: Error pixel correction eeprom
- -25014: Error main data eeprom
- -25015: Error color data eeprom
- -25016: Error correction factor data eeprom
- -25017: Error dark value eeprom
- -25018: Error calibration factor eeprom
- -25019: Error offset eeprom
- -25020: Too much ambient light
- -25021: measurement aborted
- -25024: Error userdata eeprom

- -25025: The spectral unit reported weak signal
- -25030: Not available
- -25031: Wrong device type, method not valid
- -25032: It was handed over to a wrong password
- -25041: Device Type CSS-D without detector not valid for SDK
- -25100: Value out of range

4.2 Warnings

- 25003: file not found: there were previously saved any default data. Therefore, no default file exists
- 25010: Battery low
- 25022: unstable signal
- 25023: the spectral unit reports an "overload"

Kapitel 5

Module Documentation

5.1 Information

Common information regarding the functions of the GOMDMSC15 DLL.

All the methods described here can be applied to every MSC15 and CSS device type. You can see the supported device variants in the function GOMDMSC15_getMSC15DeviceType(). Certain differences in the application can arise depending on the configuration, calibration and features of your measurement device. For instance, some methods may fail to provide any results for certain device configurations.
Each method provides a return value. Return value “0” means error-free execution of the method. Values less than “0” indicate the occurrence of an error. Values larger than “0” should be regarded as warnings.
A list of all return values is include in the documentation.

5.2 Standard SDK Methods

Functions

- int __stdcall GOMDMSC15_setPassword (char *value)
- int __stdcall GOMDMSC15_getDLLVersion (char *value)
- int __stdcall GOMDMSC15_getHandle (char *deviceName, int *handle)
- int __stdcall GOMDMSC15_releaseHandle (int handle)
- int __stdcall GOMDMSC15_getSerialNumber (int handle, char *value)
- int __stdcall GOMDMSC15_getFirmwareVersion (int handle, double *value)
- int __stdcall GOMDMSC15_isConnected (int handle, bool *value)
- int __stdcall GOMDMSC15_readStatus (int handle, int *status)
- int __stdcall GOMDMSC15_getMSC15DeviceType (int handle, int *type)
- int __stdcall GOMDMSC15_getDetectorType (int handle, int *type)
- int __stdcall GOMDMSC15_getDetectorSerialNumber (int handle, char *value)
- int __stdcall GOMDMSC15_getTemperature (int handle, double *value)

5.2.1 Detailed Description

Methods for handlings the SDK and device.

5.2.2 C++ Example

This example describes the initialization of your device. This is the default structure of initialization:

```
GOMDMSC15_setPassword("Your password
");
int handle;
int l_rc = GOMDMSC15_getHandle("MSC15_0
", &handle);     // initialization of device
if (handle > 0 )
{
    // do something
}
GOMDMSC15_releaseHandle(handle);                        // re-initialization of device
```

5.2.3 Function Documentation

5.2.3.1 GOMDMSC15_setPassword()

```
int __stdcall GOMDMSC15_setPassword (
            char * value )
```

This method must be called before any other to unlock the use of the SDK. The activation takes place on several levels.

- Layer1: Using the SDK in general
- Layer2: All elements of the 1st level plus the storage of calibrations in the user-specific memory area
  The passwords are you ever received from the Gigahertz-Optik GmbH separately.

Parameters

in	*value*	Null-terminated string that contains the password.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.2.3.2 GOMDMSC15_getDLLVersion()

```
int __stdcall GOMDMSC15_getDLLVersion (
             char * value )
```

Returns the version number of this DLL.

Parameters

out	*value*	Null-terminated string; contains by return the version number, minimum size: 10 bytes

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.2.3.3 GOMDMSC15_getHandle()

```
int __stdcall GOMDMSC15_getHandle (
             char * deviceName,
             int * handle )
```

After activation of the SDK must be called next, to initialize the MSC15 principle this method. The parameter "handle" contains a unique sequence number for instantiated instrument that needs to be passed as the first parameter when all other methods.

Parameters

in	*deviceName*	Null-terminated string that identifies the desired device to be initialized. The string is always the following structure: "MSC15_\<serial\>" or "CSS45_\<serial\>". <Serial> is a placeholder for the serial number of the instrument. The string "MSC15_5678" initialized as the MSC15 with the serial number 5678. Another possibility is the transfer of "MSC15_0". Thus, the first Windows registered MSC15 is initialized. The next call to getHandle ("MSC15_0", &handle) returns the next connected MSC15/CSS45, provided that the handle of the first device was not released.
out	*handle*	Pointer to an integer value; This value includes to return a handle> 0 if the initialization was successful, otherwise 0.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.2.3.4 GOMDMSC15_releaseHandle()

```
int __stdcall GOMDMSC15_releaseHandle (
            int handle )
```

This method must be called at the end, to release the resources / memory occupied by MSC15/CSS-45 again.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.2.3.5 GOMDMSC15_getSerialNumber()

```
int __stdcall GOMDMSC15_getSerialNumber (
            int handle,
            char * value )
```

Returns the serial number of the connected device. If you use a CSS-D + CSS-45 see also GOMDMSC15_getDetectorSerialNumber().

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Null-terminated string; contains by return the serial number of the device, minimum size: 10 bytes

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.2.3.6 GOMDMSC15_getFirmwareVersion()

```
int __stdcall GOMDMSC15_getFirmwareVersion (
            int handle,
            double * value )
```

Returns the firmware version of the direct connected device with PC.
Example: CSS-D and CSS-45, returns the FW-version of CSS-D

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Null-terminated string; contains by return, the firmware version, minimum size: 10 bytes

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.2.3.7 GOMDMSC15_isConnected()

```
int __stdcall GOMDMSC15_isConnected (
            int handle,
            bool * value )
```

Gets whether the device is still connected to the current handle.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Boolean; contains, if the device is connected: • true: MSC15 and CSS device type is connected • false: MSC15 and CSS device type is not connected

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.2.3.8 GOMDMSC15_readStatus()

```
int __stdcall GOMDMSC15_readStatus (
            int handle,
            int * status )
```

Gets the current device status of MSC15/CSS-45 (see Errors and Warnings).

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*status*	Integer; at values equal to "0" see return value table (Errors and Warnings).

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.2.3.9 GOMDMSC15_getMSC15DeviceType()

```
int __stdcall GOMDMSC15_getMSC15DeviceType (
             int handle,
             int * type )
```

Returns the type of the conneced MSC15 and CSS device type-device.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*type*	Integer; the device type. • 0: undefined • 1: MSC15 • 2: MSC15-W • 3: LVMH-spectralux100 • 4: CSS-45 • 5: CSS-45-WT • 6: CSS - D • 7: CSS - 45 - HI • 8: MSC15 - Bili

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.2.3.10 GOMDMSC15_getDetectorType()

```
int __stdcall GOMDMSC15_getDetectorType (
             int handle,
             int * type )
```

Note

This function can be used with the following device types: **CSS-D**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Checks if an external detector (e.g. CSS-45, CSS-45-HI, CSS-45-WT, etc.) is connected to the CSS-D and returns the device type of the connected device.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*type*	Integer value; indicates the device type of the connected detector. The number corresponds to the function GOMDMSC15_getMSC15DeviceType(). E.g. type = 7 = CSS-45-HI

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.2.3.11 GOMDMSC15_getDetectorSerialNumber()

```
int __stdcall GOMDMSC15_getDetectorSerialNumber (
             int handle,
             char * value )
```

Note

This function can be used with the following device types: **CSS-D**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Returns the serial number of the connected detector device. Example CSS-D + CSS-45.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Null-terminated string; contains by return the serial number of the detector device, minimum size: 10 bytes

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.2.3.12 GOMDMSC15_getTemperature()

```
int __stdcall GOMDMSC15_getTemperature (
              int handle,
              double * value )
```

Reads the current board temperature of the device. This can be used for relative temperature measurements.
MSC15: Circuit board temperature MSC-15
CSS-45: PCB temperature CSS-45
CSS-D + CSS-45: PCB temperature CSS-45
--> Sensor always reads the temperature of the sensor.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Double value, incluces the temperature.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.3 Methods for Device Settings

Functions

- int __stdcall GOMDMSC15_setDisplayMode (int handle, int mode)
- int __stdcall GOMDMSC15_getDisplayMode (int handle, int *mode)
- int __stdcall GOMDMSC15_getAvailableDisplays (int handle, int *mode)
- int __stdcall GOMDMSC15_setDisplayOrientation (int handle, bool rotated)
- int __stdcall GOMDMSC15_getDisplayOrientation (int handle, bool *rotated)

5.3.1 Detailed Description

Methods to call and change device settings.

5.3.2 Function Documentation

5.3.2.1 GOMDMSC15_setDisplayMode()

```
int __stdcall GOMDMSC15_setDisplayMode (
              int handle,
              int mode )
```

Note

This function can be used with the following device types: **MSC15, MSC15-W, MSC15-Bili, CSS-45, CSS-45-WT, CSS-45-HI, CSS-D + Detector**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

This function activated/deactivates certain displays a the MSC15 and CSS device type-device. CSS-45 has no display, but the display settings are stored in combination with a display unit (CSS-D) in the CSS-45. The CSS-D serves only as a pure display element.

Attention: Not all displays can be activated. To check the available displays for your device call GOMDMSC15_getAvailableDisplays().

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.

Parameters

in	*mode*	Integer value is interpreted bit-wise: • Bit 0: Graphics / E / CT (only MSC15) • Bit 1: CRI (only MSC15) • Bit 2: E / CT (only MSC15) • Bit 3: ES / EP (only MSC15) • Bit 4: Graphics / PAR (only MSC15) • Bit 5: Graphics / bilirubin (outdated) • Bit 6: Melanopic / brightness factor (outdated) • Bit 7: Graphics / E / CT Irradiance (only MSC15) • Bit 8: free • Bit 9: Radiant power for up to 8 wavelength ranges (only MSC15-W) • Bit 10: Irradiance for up to 8 wavelength ranges (only MSC15-W) • Bit 11: CIE1931 color triangle (only MSC15) • Bit 12: Graphics / Radiant Power / Irradiance (only MSC15-W) • Bit 13: Graphics / Irradiance (only MSC15-W) • Bit 14: Meter Only (only MSC15-W) • Bit 15: Graphics / IEC bilirubin (only MSC15) • Bit 16: Graphics / bilirubin AAP (only MSC15) • Bit 17: Melanopic (only MSC15) • Bit 18: TM-30-18 (fw >= 1.48) / TM-30-15 (fw < 1.48)(only MSC15 • Bit 19: Graphics / Bilirubin BLUE LED (only MSC15-Bili) • Bit 20: Graphics / Bilirubin AAP(2004) (only MSC15-Bili) • Bit 21: Bilirubin IEC (only MSC15-Bili) • Bit 22: Bilirubin AAP(2011) (only MSC15-Bili) • Bit 23: Bilirubin BLUE LED (only MSC15-Bili) • Bit 24: Bilirubin AAP(2004) (only MSC15-Bili)

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.3.2.2 GOMDMSC15_getDisplayMode()

```
int __stdcall GOMDMSC15_getDisplayMode (
            int handle,
            int * mode )
```

/note MSC15, MSC15-W, MSC15-Bili, CSS-45, CSS-45-WT, CSS-45-HI, CSS-D + Detector

This method returns the active display function.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*mode*	Pointer to integer value, is interpreted bit-wise, see setDisplayMode().

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.3.2.3 GOMDMSC15_getAvailableDisplays()

```
int __stdcall GOMDMSC15_getAvailableDisplays (
            int handle,
            int * mode )
```

Note

This function can be used with the following device types: **MSC15, MSC15-W, MSC15-Bili, CSS-45, CSS-45-WT, CSS-45-HI, CSS-D + Detector**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

This method returns the available display function.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*mode*	Pointer to integer value, is interpreted bit-wise, see setDisplayMode().

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.3.2.4 GOMDMSC15_setDisplayOrientation()

```
int __stdcall GOMDMSC15_setDisplayOrientation (
            int handle,
            bool rotated )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

This function sets the orientation of the device display.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*rotated*	Boolean value: • true: Display is rotated by 180° • false: Display is not rotated by 180°

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.3.2.5 GOMDMSC15_getDisplayOrientation()

```
int __stdcall GOMDMSC15_getDisplayOrientation (
            int handle,
            bool * rotated )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

This function reads back the orientation of the device display.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*rotated*	Pointer to Boolean value, the display is upside down: • true: Display is rotated by 180° • false: Display is not rotated by 180

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.4 Methods for measurement

Functions

- int __stdcall GOMDMSC15_measure (int handle)
- int __stdcall GOMDMSC15_measureWithManualIntegrationTime (int handle, int index)
- int __stdcall GOMDMSC15_measureDarkOffset (int handle)
- int __stdcall GOMDMSC15_setDynamicDarkMode (int handle, int mode)
- int __stdcall GOMDMSC15_isOffsetInvalid (int handle, bool *value)
- int __stdcall GOMDMSC15_getOffsetTime (int handle, int *timeInMs)
- int __stdcall GOMDMSC15_measurePulse (int handle, double pulseTime, double samplingRate)
- int __stdcall GOMDMSC15_dequeuePulse (int handle, double *timer, double *value)
- int __stdcall GOMDMSC15_abortPulse (int handle)

5.4.1 Detailed Description

Methods to prepare and perform a measurement.

5.4.2 C++ Example

This example shows a measurment with the MSC15 and CSS device type-device.

```
GOMDMSC15_getHandle(NULL, &handle);     //initialization of device
GOMDMSC15_measure(handle);              //start measure
        //do something
GOMDMSC15_releaseHandle(handle);        //re-initialization of device
```

5.4.3 Function Documentation

5.4.3.1 GOMDMSC15_measure()

```
int __stdcall GOMDMSC15_measure (
              int handle )
```

This method triggers the measurement. The measured values are stored in the device and can be accessed with the functions

- Methods for spectral measurement settings
- Methods for color measurement
- ...

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.4.3.2 GOMDMSC15_measureWithManualIntegrationTime()

```
int __stdcall GOMDMSC15_measureWithManualIntegrationTime (
              int handle,
              int index )
```

This method triggers the measurement with a manual integration time. The measured values are stored in the device and can be accessed with the functions

- Methods for spectral measurement settings
- Methods for color measurement
- ...

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*index*	integer value connected to integration time: • 0: integration time: 0.000012 s • 1: integration time: 0.00002 s • 2: integration time: 0.00004 s • 3: integration time: 0.00008 s • 4: integration time: 0.00016 s • 5: integration time: 0.00032 s • 6: integration time: 0.00064 s • 7: integration time: 0.00125 s • 8: integration time: 0.0025 s • 9: integration time: 0.005 s • 10: integration time: 0.01 s • 11: integration time: 0.02 s • 12: integration time: 0.04 s • 13: integration time: 0.08 s • 14: integration time: 0.16 s • 15: integration time: 0.32 s • 16: integration time: 0.64 s • 17: integration time: 1.28 s • 18: integration time: 2.56 s

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.4.3.3 GOMDMSC15_measureDarkOffset()

```
int __stdcall GOMDMSC15_measureDarkOffset (
             int handle )
```

This method starts the measurement of the dark offset. On a MSC15 device the shutter must be closed and reopened manually after the offset measurement. With the CSS-45 the shutter will be closed and opened automatically. The measured offset is automatically deducted from the (light) measurements.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.4.3.4 GOMDMSC15_setDynamicDarkMode()

```
int __stdcall GOMDMSC15_setDynamicDarkMode (
             int handle,
             int mode )
```

Note

This function can be used with the following device types: **CSS-45, CSS-45-HI, CSS-45-WT**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

This method is only available for CSS devices.
When activated (mode = 1) after each measurement a dark measurementwith the same integration time will be done and used for calculation.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*mode*	integer value: • 0: dynamic dark mode gets deactivated • 1: dynamic dark mode gets activated

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.4.3.5 GOMDMSC15_isOffsetInvalid()

```
int __stdcall GOMDMSC15_isOffsetInvalid (
            int handle,
            bool * value )
```

Note

This function can be used with the following device types: **CSS-45, CSS-45-HI, CSS-45-WT**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

The method specifies whether the dark current trigger the device is still valid. If not, it must be re-measured by the GOMDMSC15_measureDarkOffset() method.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Boolean value: • true: Dark deduction is invalid • false: Dark deduction is not invalid

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.4.3.6 GOMDMSC15_getOffsetTime()

```
int __stdcall GOMDMSC15_getOffsetTime (
            int handle,
            int * timeInMs )
```

Note

This function can be used with the following device types: **CSS-45, CSS-45-HI, CSS-45-WT**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Specifies the time required to perform an offset measurement.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*timeInMs*	Time for offset measurement in ms.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.4.3.7 GOMDMSC15_measurePulse()

```
int __stdcall GOMDMSC15_measurePulse (
             int handle,
             double pulseTime,
             double samplingRate )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

This method triggers a pulse measurement. A pulse measurement is very fast aud individual measurements, with a pulse profile is recorded. The function returns only after the entire measurement is completed.
With the function dequeuePulse () the measurement times and radiometric values of the individual measurements can be read in a separate thread.
After the pulse measurement, the measured values for the individual measurement with the largest radiometric integral in the SDK are stored and can be queried using the functions in the chapter "Reading out the measured values

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*pulseTime*	The duration of the pulse measurement in seconds
in	*samplingRate*	The sampling rate in Hz determines how quick succession, the individual measurements are to be held. If not enough signal is available, it may be that the adjusted sampling rate is not achieved, since the individual measurements duration is too long.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.4.3.8 GOMDMSC15_dequeuePulse()

```
int __stdcall GOMDMSC15_dequeuePulse (
             int handle,
             double * timer,
             double * value )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

This function reads the measurement time and radiometric values during or after a pulse measurement. It is thread-safe and can be accessed during pulse measurement in a separate thread to read the values of the current pulse measurement.
They should be called in a loop until all measurements have been collected. If the function return value -25030 (Errors), it means that no measurement values exist.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*timer*	Measurement time of individual measurement.
out	*value*	Radiometric value of single measurement.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.4.3.9 GOMDMSC15_abortPulse()

```
int __stdcall GOMDMSC15_abortPulse (
             int handle )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

This function cancels the current pulse measurement. It must be called in a separate thread and causes the GOMDMSC15_measurePulse() function is terminated directly.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.5 Methods for spectral measurement settings

Functions

- int __stdcall GOMDMSC15_getLastIntegrationTime (int handle, double *timeInS)
- int __stdcall GOMDMSC15_getWLBorders (int handle, double *startWL, double *endWL)
- int __stdcall GOMDMSC15_getWLMapping (int handle, double *wavelengths)
- int __stdcall GOMDMSC15_getSpectralDataByPixel (int handle, double *spectrum)
- int __stdcall GOMDMSC15_getSpectralData (int handle, double startWL, double deltaWL, int nrOfSteps, double *spectrum)
- int __stdcall GOMDMSC15_getPeakWL (int handle, double *value)
- int __stdcall GOMDMSC15_getCentreWL (int handle, double *value)
- int __stdcall GOMDMSC15_getCentroidWL (int handle, double *value)
- int __stdcall GOMDMSC15_getFWHM (int handle, double *value)

5.5.1 Detailed Description

Methods to perform a specral measurement.

5.5.2 C++ Example

Reads the spectrum in pixel steps (288) from MSC15-device.

```
int handle;
GOMDMSC15_getHandle(NULL, &handle);             //initialization of device
double spectrum[288];                           //Array for 288 pixel
GOMDMSC15_getSpectralData(handle, &spectrum);   //call specrum
        //Do something with data
GOMDMSC15_releaseHandle(handle);                //re-initialization of device
```

5.5.3 Function Documentation

5.5.3.1 GOMDMSC15_getLastIntegrationTime()

```
int __stdcall GOMDMSC15_getLastIntegrationTime (
            int handle,
            double * timeInS )
```

This function returns the measurement time (integration time) of the last measurement. Last measurement can be, among other things, its internal measurement of the device during initialization, GOMDMSC15_measureDarkOffset(), GOMDMSC15_measure(), ... I.e. it is advisable to call this method directly after the measurement has been performed.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*timeInS*	Pointer to Double value; integration time in seconds

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.5.3.2 GOMDMSC15_getWLBorders()

```
int __stdcall GOMDMSC15_getWLBorders (
             int handle,
             double * startWL,
             double * endWL )
```

This function retrieves the wavelength limit of the device.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*startWL*	Pointer to Double value; wavelength start
out	*endWL*	Pointer to Double value; wavelength end

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.5.3.3 GOMDMSC15_getWLMapping()

```
int __stdcall GOMDMSC15_getWLMapping (
             int handle,
             double * wavelengths )
```

This function retrieves the wavelength points for the 288 pixels of the device.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*wavelengths*	Pointer to array of doubles; array with the wavelength bases. The array must be pre-initialized with a size of 288th

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.5.3.4 GOMDMSC15_getSpectralDataByPixel()

```
int __stdcall GOMDMSC15_getSpectralDataByPixel (
            int handle,
            double * spectrum )
```

Determines the measured data and the calibration spectrum and outputs in pixel-based steps. GetWLMapping with the wavelengths can be accessed, which belong to the 288 pixels.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*spectrum*	Pointer to the first element of a double array containing the calculated Readings. The size of the array must be set to the 288th.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.5.3.5 GOMDMSC15_getSpectralData()

```
int __stdcall GOMDMSC15_getSpectralData (
            int handle,
            double startWL,
            double deltaWL,
            int nrOfSteps,
            double * spectrum )
```

Determines the measured data and the calibration spectrum and outputs this in the specified range (start wavelength, increment and number of values). The spectrum is here interpolated to the desired range. With getWL↩ Borders the maximum limits can be read on the spectrum.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*startWL*	Start wavelength.
in	*deltaWL*	Increment of wavelength between einzelen values.
in	*nrOfSteps*	Number of output values. Thus, the end wavelength is also set automatically.
in	*spectrum*	Pointer to the first element of a double array containing the calculated Readings. The size of the array is defined with nrOfSteps.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.5.3.6 GOMDMSC15_getPeakWL()

```
int __stdcall GOMDMSC15_getPeakWL (
             int handle,
             double * value )
```

Returns the wavelength at which the spectrum is maximum.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to Double value that contains the wavelength [nm].

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.5.3.7 GOMDMSC15_getCentreWL()

```
int __stdcall GOMDMSC15_getCentreWL (
             int handle,
             double * value )
```

Returns the center wave length.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to Double value that contains the center of the center wavelength in [nm].

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.5.3.8 GOMDMSC15_getCentroidWL()

```
int __stdcall GOMDMSC15_getCentroidWL (
             int handle,
             double * value )
```

Returns the centroid wavelength. The center wavelength is a measure to aspectrumto characterize. It indicates, where is the "center" of the spectrum.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to Double value that contains the centroid wavelength in [nm].

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.5.3.9 GOMDMSC15_getFWHM()

```
int __stdcall GOMDMSC15_getFWHM (
             int handle,
             double * value )
```

Returns the FWHM (full width at FDHM = half maximum) of the measured spectrum.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to Double value that contains half-width in [nm].

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.6 Methods for integral measurement settings

Functions

- int __stdcall GOMDMSC15_getPhotopic (int handle, double ∗value)
- int __stdcall GOMDMSC15_getScotopic (int handle, double ∗value)
- int __stdcall GOMDMSC15_getSPRatio (int handle, double ∗value)
- int __stdcall GOMDMSC15_getRadiometricUnit (int handle, int ∗unit)
- int __stdcall GOMDMSC15_getRadiometricValue (int handle, double startWL, double endWL, double ∗value)
- int __stdcall GOMDMSC15_getPAR (int handle, double ∗value)
- int __stdcall GOMDMSC15_getBilirubinIEC (int handle, double ∗value)
- int __stdcall GOMDMSC15_getBilirubinAAP (int handle, double ∗value)
- int __stdcall GOMDMSC15_getBilirubinAAP2004 (int handle, double ∗value)
- int __stdcall GOMDMSC15_getBilirubinNatus (int handle, double ∗value)
- int __stdcall GOMDMSC15_getMelanopic (int handle, double ∗Ev, double ∗Eemel, double ∗EvmelD65)

5.6.1 Detailed Description

Methods to perform a integral measurement.

5.6.2 C++ Example

Read the value PAR [µmol/m2/s] from the device.

```
int handle;
double value;
GOMDMSC15_getHandle(NULL, &handle);     //initialization of device
GOMDMSC15_measure();                    //start measure
GOMDMSC15_ getPAR(handle, &value);      //call value
        //Do something with data
GOMDMSC15_releaseHandle(handle);        //re-initialization of device
```

5.6.3 Function Documentation

5.6.3.1 GOMDMSC15_getPhotopic()

```
int __stdcall GOMDMSC15_getPhotopic (
            int handle,
            double * value )
```

Returns the value of the last measured photopic spectrum.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to Double value to return contains the calculated photopic value

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.6.3.2 GOMDMSC15_getScotopic()

```
int __stdcall GOMDMSC15_getScotopic (
            int handle,
            double * value )
```

Returns the scotopic value of the last measured spectrum.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to Double value to return contains the calculated photopic value.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.6.3.3 GOMDMSC15_getSPRatio()

```
int __stdcall GOMDMSC15_getSPRatio (
            int handle,
            double * value )
```

Returns the ratio of scotopischem and photopic value.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to Double value to return containing scotopisch / photopic ratio.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.6.3.4 GOMDMSC15_getRadiometricUnit()

```
int __stdcall GOMDMSC15_getRadiometricUnit (
            int handle,
            int * unit )
```

This function shows the unity of the radiometric value.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*unit*	Pointer to an integer value: • 0: 'W' (MSC15-W) • 1: 'W/m2'

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.6.3.5 GOMDMSC15_getRadiometricValue()

```
int __stdcall GOMDMSC15_getRadiometricValue (
             int handle,
             double startWL,
             double endWL,
             double * value )
```

This function returns the value of the radiometric integral.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*startWL*	Start value of the range in which the integral is to be calculated.
in	*endWL*	End value of the range in which the integral is to be calculated.
out	*value*	Pointer to Double value: Radiometric integral in the specified wavelength range

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.6.3.6 GOMDMSC15_getPAR()

```
int __stdcall GOMDMSC15_getPAR (
             int handle,
             double * value )
```

This method returns the value of PAR in mol / m2 / s.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to Double value that contains the measured value of PAR.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.6.3.7 GOMDMSC15_getBilirubinIEC()

```
int __stdcall GOMDMSC15_getBilirubinIEC (
             int handle,
             double * value )
```

This method returns the value of bilirubin according to IEC 60601-2-50.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to Double value that contains a value of bilirubin according to IEC 60601-2-50.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.6.3.8 GOMDMSC15_getBilirubinAAP()

```
int __stdcall GOMDMSC15_getBilirubinAAP (
             int handle,
             double * value )
```

This method returns the value of bilirubin by the AAP (American Academy of Pediatrics) of 2011.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to Double value that contains a value of bilirubin by AAP.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.6.3.9 GOMDMSC15_getBilirubinAAP2004()

```
int __stdcall GOMDMSC15_getBilirubinAAP2004 (
            int handle,
            double * value )
```

This method returns the value of bilirubin by the AAP (American Academy of Pediatrics).

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to Double value that contains a value of bilirubin by AAP of 2004.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.6.3.10 GOMDMSC15_getBilirubinNatus()

```
int __stdcall GOMDMSC15_getBilirubinNatus (
            int handle,
            double * value )
```

This method returns the value of bilirubin claimed by Natus of a blue LED.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to Double value that contains a value of bilirubin of a blue LED.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.6.3.11 GOMDMSC15_getMelanopic()

```
int __stdcall GOMDMSC15_getMelanopic (
            int handle,
```

```
            double * Ev,
            double * Eemel,
            double * EvmelD65 )
```

This method returns melanopic values.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*Ev*	Pointer to Double value, illuminance
out	*Eemel*	Pointer to Double value, melanopic irradiance
out	*EvmelD65*	Pointer to Double value, daylight-equivalent(D65) melanopic illuminance

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.7 Methods for color measurement

Functions

- int __stdcall GOMDMSC15_getColor (int handle, double ∗XValue, double ∗YValue, double ∗ZValue)
- int __stdcall GOMDMSC15_getColorCIE1931 (int handle, double ∗xValue, double ∗yValue)
- int __stdcall GOMDMSC15_getColorCIE1976 (int handle, double ∗uValue, double ∗vValue)
- int __stdcall GOMDMSC15_getCRI (int handle, double ∗values)
- int __stdcall GOMDMSC15_getCCT (int handle, double ∗value)
- int __stdcall GOMDMSC15_getDomWL (int handle, double ∗value)
- int __stdcall GOMDMSC15_getCompWL (int handle, double ∗value)
- int __stdcall GOMDMSC15_getDeltauv (int handle, double ∗value)
- int __stdcall GOMDMSC15_getPurity (int handle, double ∗value)
- int __stdcall GOMDMSC15_getTM3018 (int handle, double ∗Rf, double ∗Rg)
- int __stdcall GOMDMSC15_getTM3018RfByHue (int handle, double ∗values)

5.7.1 Detailed Description

Methods to perform a color measurement.

5.7.2 C++ Example

Read u' and v' (CIE1976) from the device.

```
int handle;
GOMDMSC15_getHandle(NULL, &handle);             //initialization of device
GOMDMSC15_measure();                            //start measure
double u, v;
GOMDMSC15_getColorCIE1976(handle, &u, &v);      //call value u, v
        //Do something with data
GOMDMSC15_releaseHandle(handle);                //re-initialization of device
```

5.7.3 Function Documentation

5.7.3.1 GOMDMSC15_getColor()

```
int __stdcall GOMDMSC15_getColor (
            int handle,
            double * XValue,
            double * YValue,
            double * ZValue )
```

This method returns all the calculated color values based on a spectral measurement. The spectral and color calculation must have been activated prior to the measurement.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*XValue*	Pointer to Double value contains "large" X.
out	*YValue*	Pointer to Double value contains "large" Y.
out	*ZValue*	Pointer to Double value contains "large" Z.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.7.3.2 GOMDMSC15_getColorCIE1931()

```
int __stdcall GOMDMSC15_getColorCIE1931 (
            int handle,
            double * xValue,
            double * yValue )
```

This method returns the x and y color coordinates of the CIE1931 color space.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*xValue*	Pointer to Double value contains, x corresponding to the color space CIE1931.
out	*yValue*	Pointer to Double value, y contains the corresponding CIE1931 color space.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.7.3.3 GOMDMSC15_getColorCIE1976()

```
int __stdcall GOMDMSC15_getColorCIE1976 (
            int handle,
            double * uValue,
            double * vValue )
```

This method returns the u 'and v' color coordinates of CIE1976 color space.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*uValue*	Pointer to Double value contains, u 'corresponding to the color space CIE1976
out	*vValue*	Pointer to Double value contains, v 'corresponding to the color space CIE1976

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.7.3.4 GOMDMSC15_getCRI()

```
int __stdcall GOMDMSC15_getCRI (
             int handle,
             double * values )
```

Under color rendering index (English Colour Rendering Index, CRI) is defined as a photometric Size, with the quality of Color rendering from light sources the same correlated color temperaturecan be described. As a reference to evaluate the reproduction quality is used up to a color temperature of 5000 K, the light from ablack body the corresponding color temperatureis discharged. About 5000 K is referenced against a daylight-like spectral distribution. For example, the color reproduction of a household light bulb for the calculation (which is itself a good approximation a black body is) the spectrum of a black body with a temperature of 2700 K is used as a reference for a fluorescent lamp with the light color 865 (865 for a color rendering index of more than 80, 865 for a color temperature of 6500 K), however, the spectrum of daylightilluminantD65. The color rendering index is by definition a specialmetamerism, To calculate the color rendering index are 14 test colors with a standardizedReflectance curvedefined. Mismatch of the spectra between reference and test range is used as a measure for the 14 special color rendering indices. To calculate the general color rendering index Ra, however, only the first eight test colors are used. The 14 test colors are selected by DIN 6169th Here, the color rendering index Ri for color can be found i. A numerical measure of the colors # 1 to # 8 is denoted by Ra. As in the definition of the color rendering index in the 1930s, the reference light sources 100, the then current fluorescent lamps (sort of arbitrarily) were set at 50 and the color rendering index is by no means a percentage value, and negative color rendering indices are possible.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*values*	Pointer to the first element of a double array. This contains the calculated CRI values after return. The size of the array must be initialized by 16. values[0] then contains the Ra value. The entries values[1] - values[15] contain the values R1 - R15.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.7.3.5 GOMDMSC15_getCCT()

```
int __stdcall GOMDMSC15_getCCT (
             int handle,
             double * value )
```

Returns the last measurement, the color temperature (CCT).

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to Double value to return the CCT determined contains value

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.7.3.6 GOMDMSC15_getDomWL()

```
int __stdcall GOMDMSC15_getDomWL (
            int handle,
            double * value )
```

Returns the dominant wavelength of the last measurement.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to Double value to return contains the calculated value of the dominant wavelength

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.7.3.7 GOMDMSC15_getCompWL()

```
int __stdcall GOMDMSC15_getCompWL (
            int handle,
            double * value )
```

Returns the complementary wavelength of the last measurement.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to Double value to return contains the calculated value of the complementary wavelength

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.7.3.8 GOMDMSC15_getDeltauv()

```
int __stdcall GOMDMSC15_getDeltauv (
             int handle,
             double * value )
```

Returns the actually existing at the last measurement delta uv value.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to Double value, the delta contains determined to return value uv

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.7.3.9 GOMDMSC15_getPurity()

```
int __stdcall GOMDMSC15_getPurity (
             int handle,
             double * value )
```

Provides color purity Latter measurement.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to Double value that contains the color purity

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.7.3.10 GOMDMSC15_getTM3018()

```
int __stdcall GOMDMSC15_getTM3018 (
             int handle,
             double * Rf,
             double * Rg )
```

TM-30-18 is a color rendering index published by the IES. Unlike the CRI it is based on 99 different test colors in order to obtain an accurate and adapted to all types of light rating system. The main indicators in TM-30-18, the values Rf (Fidelity Index) and Rg (Gamut Index). Rf is basically comparable to the Ra value of the CRI. The Rg value is in addition to the saturation of the colors under the measured light.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*Rf*	Pointer to Double value: Fidelity Index
out	*Rg*	Pointer to Double value: Gammoth Index

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.7.3.11 GOMDMSC15_getTM3018RfByHue()

```
int __stdcall GOMDMSC15_getTM3018RfByHue (
            int handle,
            double * values )
```

When CRI there are the values of R1-R15. In order to give a statement about the color quality for each color when TM-30-15 there is the splitting of the Rf in 16 average color rendering values. The Rf [by Hue].

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*values*	Pointer to the first element of a double array. This contains the calculated Rf values by color. The size of the array must be vorinizialisiert by 16.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.8 Load stored Measurements from Device

Functions

- int __stdcall GOMDMSC15_loggerReadDate (int handle, int index, char ∗date)
- int __stdcall GOMDMSC15_loggerLoadMeasurement (int handle, int index)
- int __stdcall GOMDMSC15_loggerDeleteMeasurement (int handle, int index)
- int __stdcall GOMDMSC15_loggerSetRTC (int handle, char ∗date)
- int __stdcall GOMDMSC15_loggerGetRTC (int handle, char ∗date)

5.8.1 Detailed Description

Methods to read, load and delete measurements from the internal device logger.

Saving of measurement data is only possible in handheld mode.

5.8.2 Function Documentation

5.8.2.1 GOMDMSC15_loggerReadDate()

```
int __stdcall GOMDMSC15_loggerReadDate (
              int handle,
              int index,
              char * date )
```

Note

This function can be used with the following device types: **MSC15, MSC15-Bili, CSS-45, CSS-45-HI, CSS-45-WT, CSS-45 + CSS-D**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

This methods read the date of a measurement saved in the internal data logger.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*index*	index of measurement in internal data logger (0-9)
out	*date*	Null-terminated string; contains by return the date and time(TTMMJJhhmmss)

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.8.2.2 GOMDMSC15_loggerLoadMeasurement()

```
int __stdcall GOMDMSC15_loggerLoadMeasurement (
              int handle,
              int index )
```

Note

This function can be used with the following device types: **MSC15, MSC15-Bili, CSS-45, CSS-45-HI, CSS-45-WT, CSS-45 + CSS-D**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

This methods loads a measurement from the internal data logger. To get the data you can use the common routines for reading measurement data. Actual Measurement values will be overwritten by calling this method.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*index*	index of measurement in internal data logger (0-9)

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.8.2.3 GOMDMSC15_loggerDeleteMeasurement()

```
int __stdcall GOMDMSC15_loggerDeleteMeasurement (
              int handle,
              int index )
```

Note

This function can be used with the following device types: **MSC15, MSC15-Bili, CSS-45, CSS-45-HI, CSS-45-WT, CSS-45 + CSS-D**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

This methods deletes the measurement at given index from the internal data logger.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*index*	index of measurement in internal data logger (0-9)

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.8.2.4 GOMDMSC15_loggerSetRTC()

```
int __stdcall GOMDMSC15_loggerSetRTC (
            int handle,
            char * date )
```

Note

This function can be used with the following device types: **MSC15, MSC15-Bili, CSS-D**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

This methods sets the internal Real Time Clock of the device. This clock is used for the device logger

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*date*	Null-terminated string; containing the date and time(TTMMJJhhmmss)

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.8.2.5 GOMDMSC15_loggerGetRTC()

```
int __stdcall GOMDMSC15_loggerGetRTC (
            int handle,
            char * date )
```

Note

This function can be used with the following device types: **MSC15, MSC15-Bili, CSS-D**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

This method reads the internal Real Time Clock of the device. This clock is used for the device logger.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*date*	Null-terminated string; contains by return the date and time(TTMMJJhhmmss)

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.9 Methods only for MSC15-W-devices

Functions

- int __stdcall GOMDMSC15_setUserWLRangeBorders (int handle, int userRange, double startWL, double endWL)
- int __stdcall GOMDMSC15_setUserWLRangeBoarders (int handle, int userRange, double startWL, double endWL)
- int __stdcall GOMDMSC15_setUserWLRangeBordersToFactory (int handle)
- int __stdcall GOMDMSC15_getUserWLRangeBorders (int handle, int userRange, double *startWL, double *endWL)
- int __stdcall GOMDMSC15_getUserWLRangeBoarders (int handle, int userRange, double *startWL, double *endWL)
- int __stdcall GOMDMSC15_getSelectedUserWLRange (int handle, int *userRange, bool *locked)
- int __stdcall GOMDMSC15_setSelectedUserWLRange (int handle, int userRange, bool locked)
- int __stdcall GOMDMSC15_getUserWLRangeActive (int handle, int userRange, bool *active)
- int __stdcall GOMDMSC15_getUserWLRangeModifiable (int handle, int userRange, bool *modifiable)
- int __stdcall GOMDMSC15_getUserWLRangeRadiometric (int handle, int userRange, double *value)
- int __stdcall GOMDMSC15_setAppertureSize (int handle, int index, double size)
- int __stdcall GOMDMSC15_getAppertureSize (int handle, int index, double *size)
- int __stdcall GOMDMSC15_getAppertureSizeBorders (int handle, double *minSize, double *maxSize)
- int __stdcall GOMDMSC15_setAppertureIndex (int handle, int index)
- int __stdcall GOMDMSC15_getAppertureIndex (int handle, int *index)
- int __stdcall GOMDMSC15_setUserCorrectionFactor (int handle, double value)
- int __stdcall GOMDMSC15_getUserCorrectionFactor (int handle, double *value)
- int __stdcall GOMDMSC15_setUserCorrectionFactorLocked (int handle, bool value)
- int __stdcall GOMDMSC15_getUserCorrectionFactorLocked (int handle, bool *value)

5.9.1 Detailed Description

These methods can only be used in combination with an MSC15-W-device.

5.9.2 Function Documentation

5.9.2.1 GOMDMSC15_setUserWLRangeBorders()

```
int __stdcall GOMDMSC15_setUserWLRangeBorders (
            int handle,
            int userRange,
            double startWL,
            double endWL )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Sets the wavelength limits for user adjustable wavelength ranges.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*userRange*	Integer value; the wavelength range for which the limits are to be set. (range 0-7)
in	*startWL*	Double value; start wavelength for the range
in	*endWL*	Double value; end of the shaft length limit

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.9.2.2 GOMDMSC15_setUserWLRangeBoarders()

```
int __stdcall GOMDMSC15_setUserWLRangeBoarders (
              int handle,
              int userRange,
              double startWL,
              double endWL )
```

5.9.2.3 GOMDMSC15_setUserWLRangeBordersToFactory()

```
int __stdcall GOMDMSC15_setUserWLRangeBordersToFactory (
              int handle )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Sets the wavelength limits for user adjustable wavelength ranges back to factory settings.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.9.2.4 GOMDMSC15_getUserWLRangeBorders()

```
int __stdcall GOMDMSC15_getUserWLRangeBorders (
              int handle,
              int userRange,
              double * startWL,
              double * endWL )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Reads the wavelength limits for user adjustable wavelength ranges.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*userRange*	Integer value, the wavelength range for which the limits are to be read (range 0-7)
in	*startWL*	Pointer to double value, start wavelength for the range
in	*endWL*	Pointer to double value, end of the shaft length limit

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.9.2.5 GOMDMSC15_getUserWLRangeBoarders()

```
int __stdcall GOMDMSC15_getUserWLRangeBoarders (
              int handle,
              int userRange,
              double * startWL,
              double * endWL )
```

5.9.2.6 GOMDMSC15_getSelectedUserWLRange()

```
int __stdcall GOMDMSC15_getSelectedUserWLRange (
              int handle,
              int * userRange,
              bool * locked )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Gets the wavelength range from MSC15-device, which is currently used on the device display.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*userRange*	Pointer to integer value, the wavelength range which is currently set. (There are the areas 0-7)
out	*locked*	Pointer to boolean value: • true: Area is locked • false Area is not locked

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.9.2.7 GOMDMSC15_setSelectedUserWLRange()

```
int __stdcall GOMDMSC15_setSelectedUserWLRange (
              int handle,
              int userRange,
              bool locked )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Sets the wavelength range, which is used on the device display.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*userRange*	Integer value; the wavelength range for which the limits are to be read (range 0-7)
in	*locked*	Boolean value; this area is to be locked in the unit. Can not then be changed by the user.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.9.2.8 GOMDMSC15_getUserWLRangeActive()

```
int __stdcall GOMDMSC15_getUserWLRangeActive (
              int handle,
              int userRange,
              bool * active )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Retrieves whether a wavelength range is active, and can thus be selected by the user in the device. This value is set by Gigahertz-Optik when the device is shipped.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*userRange*	Integer value; the wavelength range for which the limits are to be read (range 0-7)
out	*active*	Pointer to boolean value: • true: Range is active • false: Range is not active

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.9.2.9 GOMDMSC15_getUserWLRangeModifiable()

```
int __stdcall GOMDMSC15_getUserWLRangeModifiable (
            int handle,
            int userRange,
            bool * modifiable )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Retrieves whether a wavelength range is modifiable, and can thus be changed by the user in the device. This value is set by Gigahertz-Optik when the device is shipped.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*userRange*	Integer value; the wavelength range for which the limits are to be read (range 0-7)
out	*modifiable*	Pointer to boolean value: • true: Range is modifiable • false: Range is not modifiable

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.9.2.10 GOMDMSC15_getUserWLRangeRadiometric()

```
int __stdcall GOMDMSC15_getUserWLRangeRadiometric (
            int handle,
            int userRange,
            double * value )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Calculates the radiometric value of the spectrum in the wavelength limits that have been set with GOMDMSC15_setUserWLRangeBoarders().

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*userRange*	Integer value; the wavelength range for which the limits are to be read (range 0-7)
out	*value*	Pointer to double value; radiometric value in the range of wavelength limits.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.9.2.11 GOMDMSC15_setAppertureSize()

```
int __stdcall GOMDMSC15_setAppertureSize (
            int handle,
            int index,
            double size )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Sets the aperture diameter for the MSC15-W. Up to 5 diameters are stored in the device. With GOMDMSC15_setAppertureIndex() a aperture can be selected. Caution: The aperture diameter is not automatically charged the SDK functions. If you want to determine the irradiance, you must perform the calculation itself.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*index*	Integer value; the aperture index, which is to be changed. (possible values are 0-4)
in	*size*	Double value; value of the aperture diameter

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.9.2.12 GOMDMSC15_getAppertureSize()

```
int __stdcall GOMDMSC15_getAppertureSize (
            int handle,
            int index,
            double * size )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Reads the aperture diameter for a particular index.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*index*	Integer value, he aperture index, which is to be changed. (possible values are 0-4)
out	*size*	Pointer to double value, value of the aperture diameter.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.9.2.13 GOMDMSC15_getAppertureSizeBorders()

```
int __stdcall GOMDMSC15_getAppertureSizeBorders (
            int handle,
            double * minSize,
            double * maxSize )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Reads the input limits, which can be used for setApperturesize. These have been defined by Gigahertz-Optik.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*minSize*	Pointer to double value, minimum value of the aperture diameter.
out	*maxSize*	Pointer to double value, maximum value of the aperture diameter.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.9.2.14 GOMDMSC15_setAppertureIndex()

```
int __stdcall GOMDMSC15_setAppertureIndex (
              int handle,
              int index )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Defines the aperture diameter (index) to be used in the device for clearing. Caution: The aperture diameter is not automatically charged the SDK functions. If you want to determine the irradiance, you must perform the calculation itself.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*index*	Integer value, the aperture index, which is to be set to active. (possible values 0-4)

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.9.2.15 GOMDMSC15_getAppertureIndex()

```
int __stdcall GOMDMSC15_getAppertureIndex (
              int handle,
              int * index )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Sets the Apperturgröße for MSC15-W. Up to 5 Apperturgrößen be stored in the device. With GOMDMSC15_setAppertureIndex() a Apperur can be selected.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*index*	Integer value, the aperture index, which is actually active. (possible values 0-4)

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.9.2.16 GOMDMSC15_setUserCorrectionFactor()

```
int __stdcall GOMDMSC15_setUserCorrectionFactor (
              int handle,
              double value )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Sets the user settable correction factor. Note: The correction factor is automatically charged to the spectrum and all derived metrics. The accuracy of the values accepts Gigahertz-Optik no guarantee.

Parameters

in	*handle*	Integer value $>$ 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
in	*value*	Double value, the desired correction factor.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.9.2.17 GOMDMSC15_getUserCorrectionFactor()

```
int __stdcall GOMDMSC15_getUserCorrectionFactor (
            int handle,
            double * value )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Reads out the adjusted correction factor. Note: The correction factor is automatically charged to the spectrum and all derived metrics. The accuracy of the values accepts Gigahertz-Optik no guarantee.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to double value, returns the current correction factor.

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.9.2.18 GOMDMSC15_setUserCorrectionFactorLocked()

```
int __stdcall GOMDMSC15_setUserCorrectionFactorLocked (
            int handle,
            bool value )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Locks the currently set correction factor in the device. This cannot be changed then the device display from the user.
Note: The correction factor is automatically charged to the spectrum and all derived metrics. The accuracy of the values accepts Gigahertz-Optik no guarantee.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Boolean value: • true: Value is locked • false: Value can be changed

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.

5.9.2.19 GOMDMSC15_getUserCorrectionFactorLocked()

```
int __stdcall GOMDMSC15_getUserCorrectionFactorLocked (
            int handle,
            bool * value )
```

Note

This function can be used with the following device types: **MSC15-W**
The device type can be determined with the function GOMDMSC15_getMSC15DeviceType().

Determines whether the currently set for the machine correction factor is locked. Note: The correction factor is automatically charged to the spectrum and all derived metrics. The accuracy of the values accepts Gigahertz-Optik no guarantee.

Parameters

in	*handle*	Integer value > 0 for unique identification of the instantiated MSC15 and CSS device type; this value is returned by the getHandle method.
out	*value*	Pointer to boolean value: • true: Value is locked • false: Value can be changed

Returns

Integer values; see return value table (warnings and errors) for values unequal to “0”.