HIOKI

Instruction Manual

# TM6101 Measuring Library

EN

May 2021 Revised edition 2
TM6101A985-02 21-05H

# Contents

# Chapter 1 Overview

The measuring library consists of Windows software designed for use with the TM6101 LED Optical Meter. It can be used on a computer running Windows to develop software for controlling the TM6101.

Operating environment:

Supported operating systems: Windows 7 (32bit/64bit), Windows 8 (32bit/64bit), Windows 10 (32bit/64bit)

Supported development environments: Visual Studio 2017, 2019 (Visual C++, Visual Basic, Visual C#)

**Note**

Use a computer on which the target operating system operates properly. The software may not operate with sufficient speed in some operating environments.

The measuring library includes C-language header files. When using the library in a development environment other than C, for example with Visual Basic, you will need to create function declarations as necessary.

# Chapter 2 Using the Measuring Library

## 2.1 Installing the Library

Install the software as described in “Chapter 2 Measurement Preparations” of the TM6101 LED Optical Meter Instruction Manual. You will need to install both the driver software and the PC application in order to make use of the measuring library.

## 2.2 Using the Library

A “Library” folder will be created in the folder into which the software was installed as described in the Instruction Manual. Copy the files in the “Library” folder to the location of your choice in your development environment.

| | |
|---|---|
| HiLedMeas.dll | DLL software |
| HiLedMeas.lib | Library file |
| Tm6101Api.h | Library header file |

**Note**

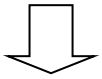
The above files, driver software, and PC application software may only be redistributed for the purpose of controlling the TM6101. When distributing software you have developed, include the above files and the included installer as necessary.

# Chapter 3 Controlling the TM6101

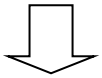
## 3.1 Overall Process

**Open the instrument.**

Use an open function to acquire a device number for the connected instrument.

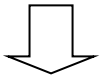


**Set measurement conditions.**



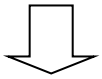
Set measurement conditions with the measurement condition configuration function. You can either use functions that set individual measurement conditions or a function that sets all the measurement conditions at once.

**Perform dark measurement.**



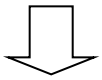
If dark correction if not performed, you will not be able to obtain normal measured values. Be sure to perform dark measurement before making measurements.

**Perform reference value correction.**



In this process, the instrument's sensitivity is corrected based on reference light source spectral data and photometric values that you provide. Perform reference value correction as necessary.

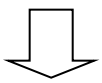
**Perform measurement.**



Measure the target light source and perform color calculations.

**Acquire the measurement results.**

Acquire the measurement results.



**Close the instrument.**

Close the instrument.

## 3.2 Opening the Instrument

In order to control the instrument, it is first necessary to use an open function to open it. When the open function executes successfully, it will return at least one device number. Once the instrument has been opened, the assigned device number is used to control it.

When the instrument is opened, all measurement conditions will be initialized. Additionally, if a given instrument is closed and then reopened, it may be assigned a different device number, and all measurement conditions will be initialized.

**Note**

Immediately after the AC adapter and USB cable are connected to the instrument after turning on the computer's power supply, the main unit's power indicator will turn red. When the instrument is opened, the power indicator will change from red to green, and when the instrument is closed, it will change back to red.

The device number acquired when opening the instrument can only be used within the same process. The same TM6101 cannot be opened at the same time from multiple processes.

Open functions:

| | |
|---|---|
| long TmOpenDevice(); | Opens a TM6101 and acquires a device number. When multiple instruments are connected to the computer, it is not possible to specify a particular device to open. |
| long TmOpenDeviceBySerial(char* pSerial); | Opens the TM6101 specified using a serial number (a 9-digit string) and acquires a device number. |

## 3.3 Closing the Instrument

Use the close function to close the instrument once you have completed all control operations. Once an instrument has been closed, the device number acquired with the open function can no longer be used.

**Note**

When the instrument is closed, its power indicator will change from green to red.

Close function:

| | |
|---|---|
| long TmCloseDevice(long lDeviceId); | Closes the TM6101 with the specified device number. |

## 3.4 Setting Measurement Conditions

Before making measurements with the TM6101, it is necessary to set measurement conditions such as the integration time, sensitivity, and average times. It is also possible to set all the measurement conditions at once by specifying a measurement condition structure.

The current measurement conditions can be acquired using measurement condition acquisition functions.

The instrument can be reverted to the measurement conditions in effect when it was opened using the measurement condition initialization function.

**Note**

Measurement condition configuration functions cannot be used while the instrument is in the measurement standby state. The TmGetStandbyStatus function, described below, can be used to detect whether the instrument is in the measurement standby state.

The following measurement conditions must be set:

### Normal Measurement Mode Settings

The TM6101 can make measurements in either of two measurement modes: normal measurement mode or AC measurement mode. The instrument is set to normal measurement mode when it is opened.

| | |
|---|---|
| Measurement mode | Set to normal measurement mode. |
| Integration time | 0.1 / 0.5 / 1 / 2 / 4 / 8 / 10 / 16.6 / 20 / 33.3 / 40 ms |
| Sensitivity range | High / Low |
| Average times | 1 to 100 |
| Auto-ranging | OFF / Integration time auto-ranging / Sensitivity auto-ranging |
| Auto-ranging level | 1% to 99% |

**Note**

When auto-ranging is enabled, the auto-ranging function will be disabled when making measurements under the following conditions:

- AC measurement mode
- External trigger
- Dark measurement

## AC Measurement Mode Settings

When making measurements in AC measurement mode, configure the following settings:

| | |
|---|---|
| Measurement mode | Set to AC measurement mode. |
| AC drive settings | Measurement range (range 1 to 3) |
| | Power supply frequency (60/50 Hz) |
| | Average times (1 to 100) |

*AC drive settings (measurement range, power supply frequency, and average times) are set together with the TmSetAcMode function.

## External I/O Settings

| | |
|---|---|
| Trigger type | OFF / External trigger ON (rising edge) / External trigger ON (falling edge) |
| Trigger delay | 0 to 1,000 ms |
| Trigger timeout | 10,000 to 1,000,000 ms |
| Index output time | 1 to 100 ms |

**Note**

When not using an external trigger, the trigger delay and trigger timeout settings are disabled.

## Calculation Settings

| | |
|---|---|
| Reference light | CIE daylight / Blackbody radiation / Automatic selection |
| | *Used in calculating the color rendering index. |
| Measuring distance | 0.01 to 10.00 m |
| | *Used in measuring luminous intensity values. |

## 3.5 Dark Measurement

Dark-corrected measurement results are obtained by performing dark measurement before making measurements. If dark correction if not performed, you will not be able to obtain normal measured values. Be sure to perform dark measurement before making measurements. The average times can be set when performing dark measurement (average times: 1 to 100).

**Note**

Be sure to affix the included cap before performing dark measurement.

Either of two methods can be used to perform dark measurement: it can be performed for the current integration time and sensitivity range, or for all integration times and sensitivity ranges.

### Performing Dark Measurement for the Current Integration Time and Sensitivity Range

This approach takes less time to complete, but dark measurement values are cleared whenever the integration time, sensitivity range, or measurement mode is changed. Additionally, dark measurement values are cleared when measurement is performed using auto-ranging.

Dark measurement is performed using the TmExecDarkMeas function. The function does not return until dark measurement is complete.

## Performing Dark Measurement for All Integration Times and Sensitivity Ranges

By calling the function once, dark measurement is performed while automatically switching the integration time and sensitivity range. The operation takes some time to complete since dark measurement is performed for all integration times and sensitivity ranges.

When using this approach, there is no need to repeat dark measurement, even if the integration time, sensitivity range, or measurement mode is changed. Dark measurement results remain valid until the instrument is turned off. When enabling auto-ranging, perform dark measurement for all integration times and sensitivity ranges.

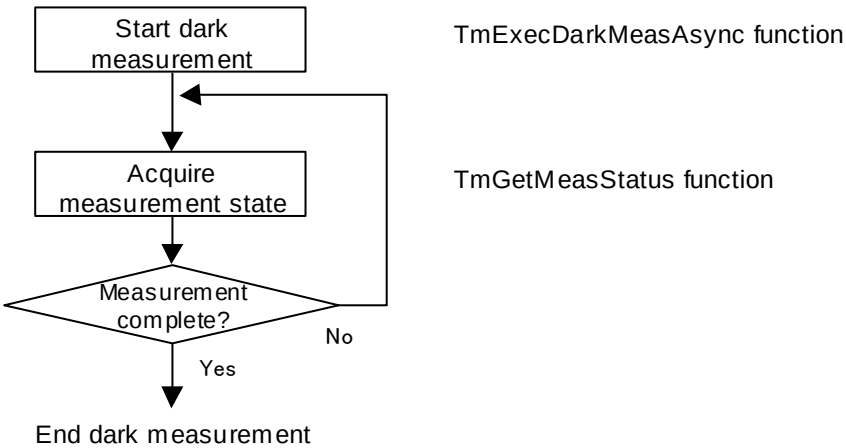
The library provides functions for acquiring all dark correction data, setting all dark correction data, and reverting the instrument to its state before dark measurement was performed.

Dark measurement is performed using either a synchronous or asynchronous function.

(1) Performing dark measurement using the synchronous function
Dark measurement is performed using the TmExecDarkMeas function. The function does not return until dark measurement is complete.

(2) Performing dark measurement using the asynchronous function
Dark measurement is performed using the TmExecDarkMeasAsync function. The function returns immediately, but it is necessary to monitor the measurement status with the TmGetMeasStatus until dark measurement completes.

Example of dark measurement performed using the asynchronous function:



## 3.6 Reference Value Correction

In this process, the instrument's sensitivity is corrected based on reference light source spectral data and photometric values that you provide. Reference value correction should be performed as necessary.

Reference value correction results are valid until the instrument is closed using the close function. In addition to functions for performing reference value correction, the library includes functions for acquiring reference value correction data, setting reference value correction data, and reverting the instrument to its state before reference value correction was performed.

### Performing Chromaticity Correction

1. Measure the spectral characteristics of the light source for which chromaticity correction is to be performed using a standard instrument (spectral-type measuring instrument) and prepare the corresponding measurement results. You will need spectral measurement results at a 5 nm interval from 380 nm to 780 nm.
2. Measure the light source for which chromaticity correction is to be performed with the TM6101. For more information about the measurement procedure, see "3.7 Making Measurements" below.
3. Perform chromaticity correction using one of the chromaticity correction functions. The measurement results from step (2) above (from the preceding step) will be subject to correction.

・When using TmExecChromaticityCorrect, specify the spectral measurement results as the function argument.

・When using TmExecChromaticityCorrectByFile, specify the name of the file in which the spectral measurement results were saved as the function argument. For more information about the file format, see "2.8 Using Correction Functions" in the TM6101 LED Optical Meter Instruction Manual.

### Performing Illuminance, Luminous Intensity, and Luminous Flux Correction

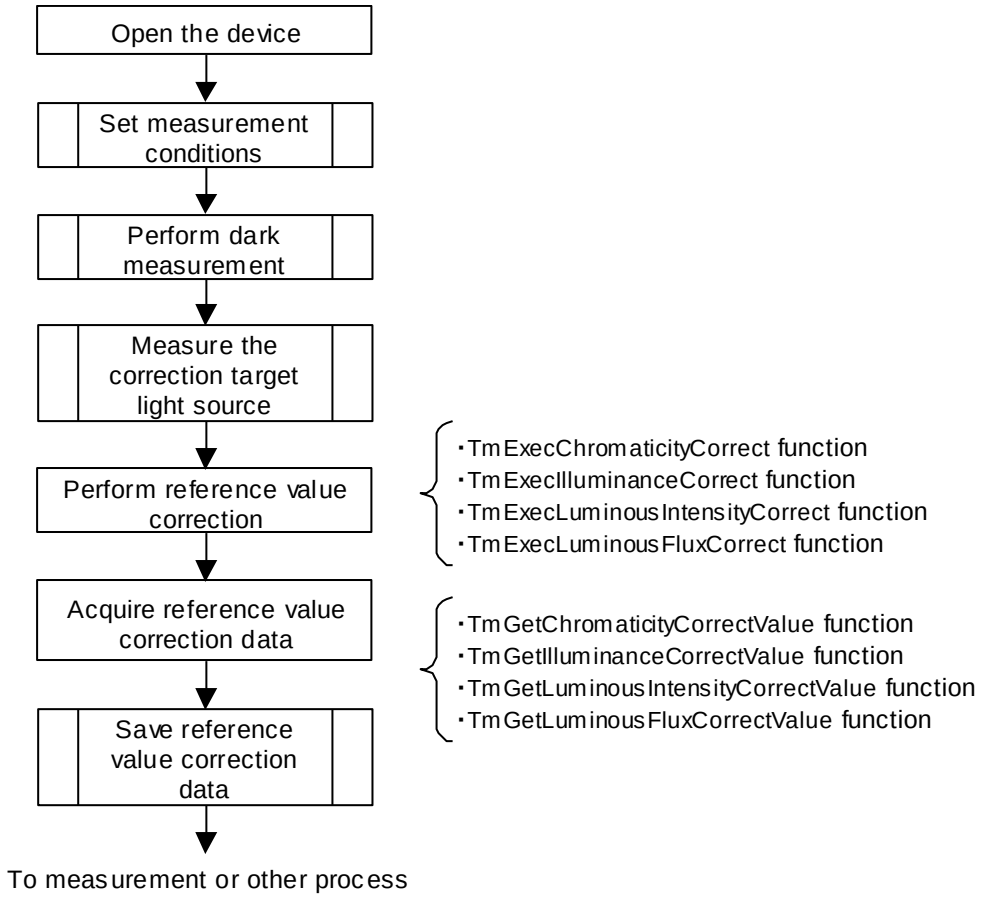
1. Measure the correction target light source with the TM6101. For more information about the measurement procedure, see "3.7 Making Measurements" below.
2. Perform correction using the appropriate correction function, specifying the correction target reference value as the function argument. The measurement results from step (1) above (from the preceding step) will be subject to correction.

・To perform illuminance correction, use TmExecIlluminanceCorrect.

・To perform luminous intensity correction, use TmExecLuminousIntensityCorrect.

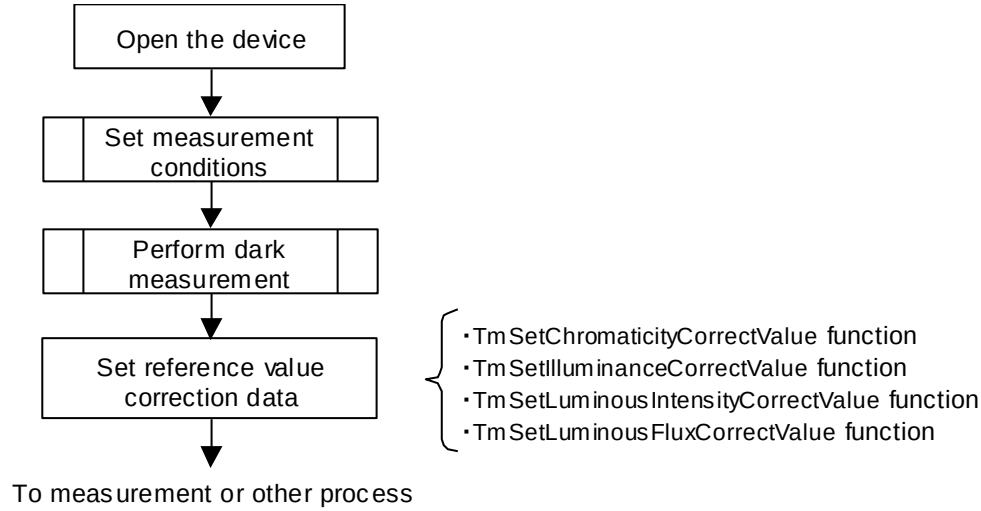
・To perform luminous flux correction, use TmExecLuminousFluxCorrect.

Example of reference value correction:




*Reference value correction data should be acquired and saved as necessary, for example when it will be necessary to restore correction data the next time the instrument is turned on. Processing to save reference value correction data must be implemented by the customer (for example, by saving data to a file, etc.).

Example restoration of reference value correction data:



## 3.7 Making Measurements

Measurement is performed using either a synchronous or asynchronous measurement function. Once measurement is complete, the measurement results can be acquired using a measurement results acquisition function.

**Note**

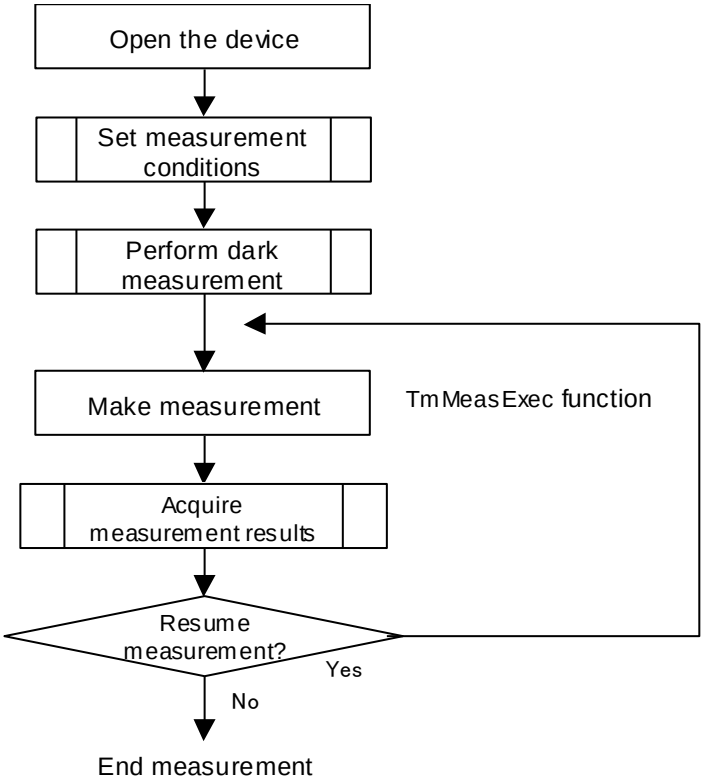
When the external trigger is enabled, external trigger monitoring is started by executing the measurement function. Once measurement using the external trigger is complete, external trigger monitoring is cancelled. To start measurement using the external trigger again, execute the measurement function again.

### Making Measurements Using the Synchronous Function

Measurements are made using the TmMeasExec function. The function does not return until measurement is complete and the instrument is in the measurement standby state. The instrument enters the measurement standby state when external I/O measurement complete output changes to OFF.

When the external trigger is enabled, external trigger monitoring is started by executing the TmMeasExec function. The function does not return until either measurement completes following external trigger input or the timeout time elapses.

Example use of the synchronous function:



## Making Measurements Using the Asynchronous Function

Since the asynchronous function returns immediately when measurement starts, other processing can be performed while the instrument is making measurements.
When the external trigger is enabled, external trigger monitoring is started by executing the TmMeasExec function. While the external trigger is being monitored, the monitoring state can be cancelled (i.e., the instrument can be set to the measurement standby state) by calling the TmCancelMeas function.
Once the TmMeasExecAsync function returns, the measurement status must be monitored with the TmGetMeasStatus and TmGetStandbyStatus functions.
The following procedure is used to make measurements using the asynchronous function:

1. Start measurement with the TmMeasExecAsync function. The function will return as soon as the instrument starts measurement.
   *If auto-ranging is enabled, the function will not return until auto-ranging processing completes.

2. Acquire the measurement status with the TmGetMeasStatus function. The TmGetMeasStatus function should be repeatedly called until measurement completes. If a value of 0xFFFFFFFF is specified as the timeout time argument, the TmGetMeasStatus function will not return until measurement completes. Since measurement processing is performed internally by the TmGetMeasStatus function, be sure to verify the completion of measurement with the TmGetMeasStatus function.

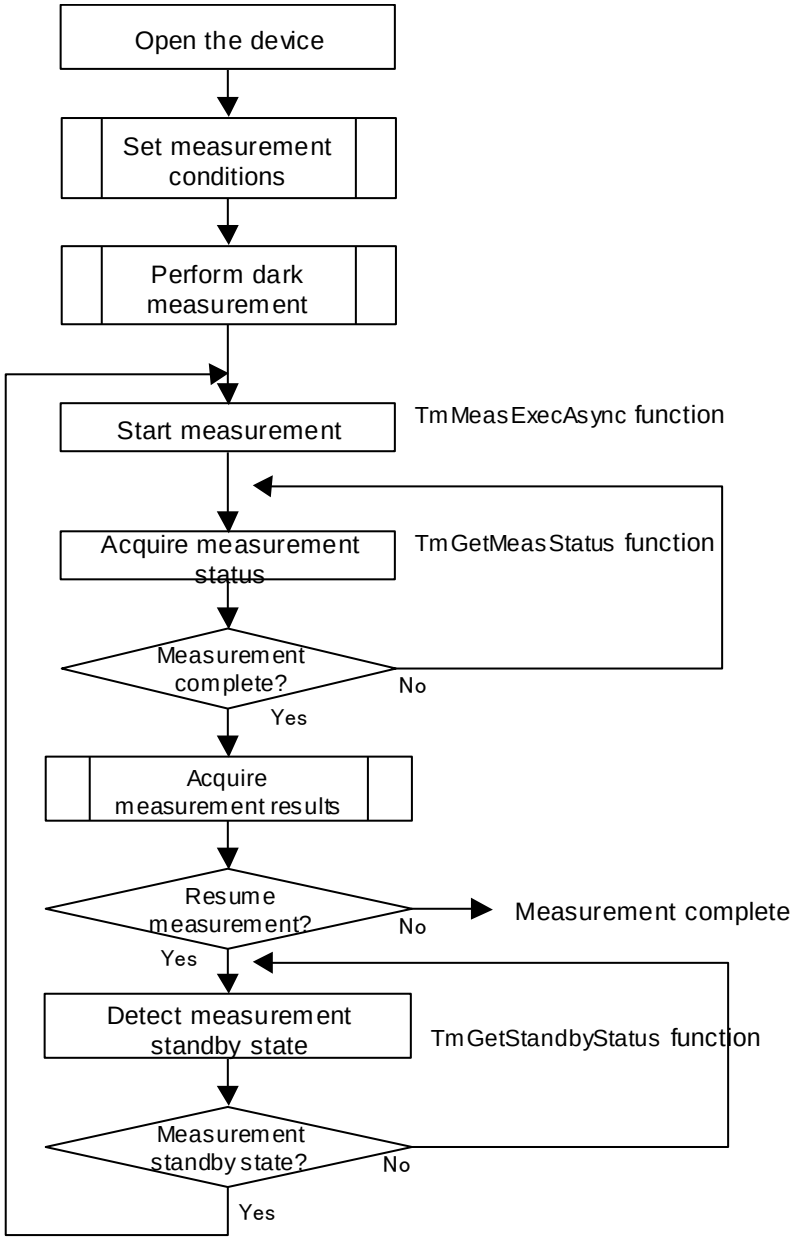
3. Once measurement completes, the measurement results can be acquired using the library's measurement results acquisition functions. Additionally, since measurement by the sensor will have completed by this time, processing such as positioning the next measurement target can be performed.

4. Detect whether the instrument is in the measurement standby state with the TmGetStandbyStatus function. If the instrument is in the measurement standby state, measurement can be started with the TmMeasExecAsync function. If a value of 0xFFFFFFFF is specified as the timeout time argument, the TmGetStandbyStatus function will not return until the instrument enters the measurement standby state.

**Note**

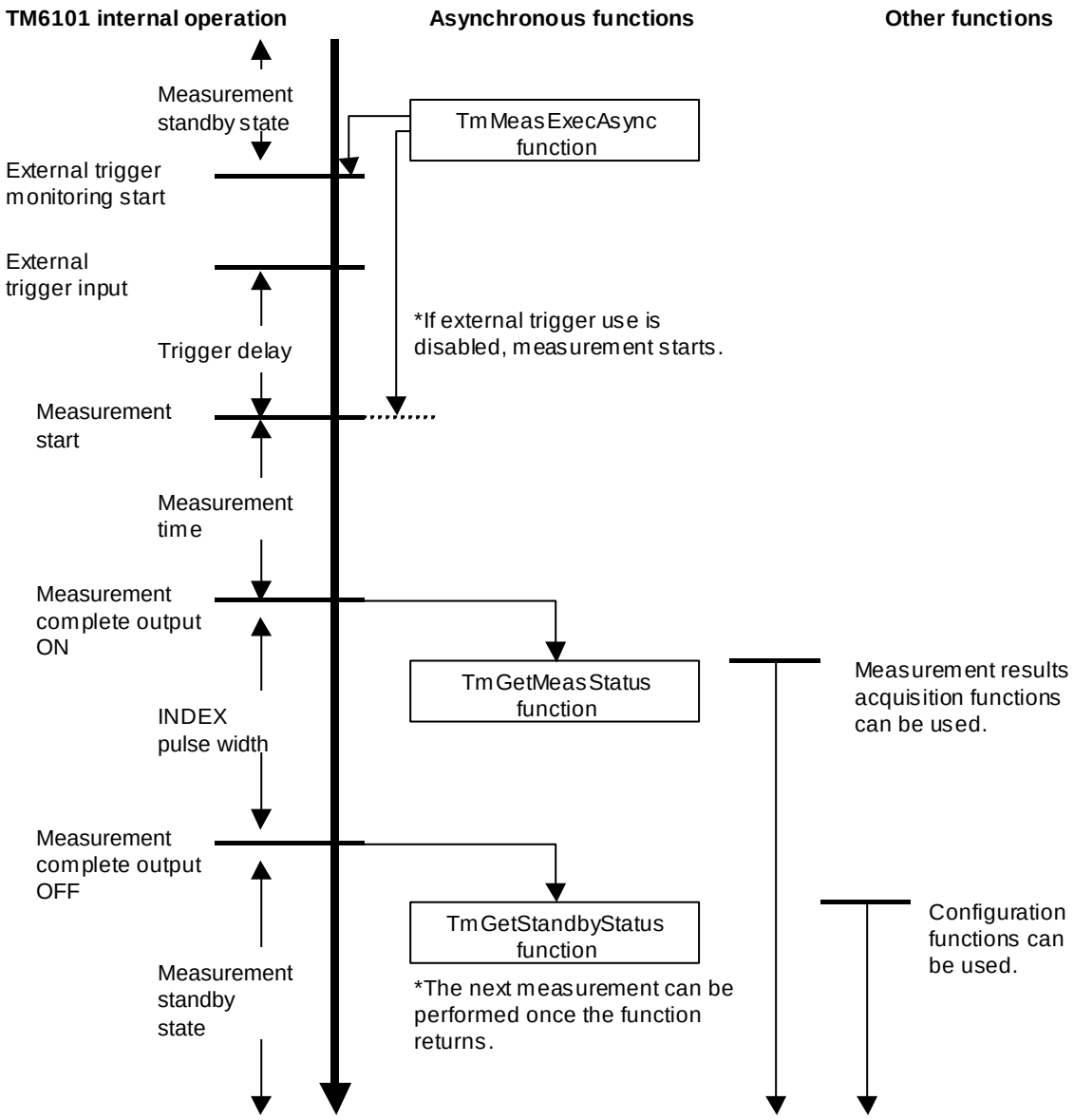
Once measurement starts, measurement conditions cannot be set until measurement completes and the instrument enters the measurement standby state. Do not use the measurement condition configuration functions until you verify that the instrument has entered the measurement standby state.

Example use of asynchronous function:




The following diagram illustrates the timing of instrument and library function operation. For more detailed information about instrument operation timing, see “4.2 Timing Chart” in the TM6101 LED Optical Meter Instruction Manual.

TM6101 operation timing:




* Due to communications time requirements, it takes approximately 2 ms from the time the TmMeasExecAsync function is called until external trigger monitoring starts. Once the TmMeasExecAsync function returns, external trigger monitoring will already have started.

* Similarly, it takes approximately 2 ms from the time measurement completes until the TmGetMeasStatus function returns, and approximately 2 ms from the time measurement complete output turns OFF until the TmGetStandbyStatus function returns (when in the standby state after setting the timeout argument to 0xFFFFFFFF).

* The exact amount of time required for communications varies with factors such as the computer's processing capability and the operating environment being used.

* When making measurements using the TmMeasExec function (a synchronous function), the function returns when measurement complete output changes to OFF.

## 3.8 Acquiring Measurement Results

Once measurement completes, measurement results such as illuminance and chromaticity values should be acquired. When the measurement function terminates normally, measurement results can be acquired. Measurement results can also be acquired together by specifying a measurement results structure.

### Measurement Results That Can Be Acquired

Illuminance value
Luminous intensity value
Luminous flux value
Tristimulus values (XYZ)
Chromaticity values (xy, uv)
Correlated color temperature, Δuv
Special color rendering indexes (R1 to R15)
Average color rendering index (Ra)
Dominant wavelength

**Note**

Measurement results are not finalized until measurement executes successfully. Using a measurement results acquisition function while measurement is still in progress will result in an error. Measurement results can be acquired once the TmMeasExec function (a synchronous function) executes successfully, or once a measurement status of "measurement terminated successfully" is returned by the TmGetMeasStatus function.

Under some circumstances, it may be impossible to calculate measurement results despite measurement having completed successfully (indicated by a return value other than 0). Always verify measurement results acquisition function return values (look for negative chromaticity values, a Δuv value that is greater than or equal to 0.02, etc.). In such cases, the measurement results returned to the function argument will be undefined.

If measurement is canceled with the TmCancelMeas function while the external trigger is being monitored, measurement results acquisition functions will return the results of the previous measurement.

# Chapter 4 Library Function Reference

## 4.1 Connection Functions

### TmOpenDevice

| | |
|---|---|
| **Description** | Opens the TM6101 and acquires a device number, which is subsequently used when performing processing with library functions. |
| **Declaration** | long TmOpenDevice(); |
| **Arguments** | None |
| **Return value** | 1 or greater: Device number 0: Failure |
| **Note** | When multiple instruments are connected to the computer, this function does not allow a specific instrument to be specified. After the instrument has been opened, the indicator on the main unit will change from red to green. |

### TmOpenDeviceBySerial

| | |
|---|---|
| **Description** | Opens the TM6101 with the specified serial number (a 9-digit string) and acquires a device number, which is subsequently used when performing processing with library functions. |
| **Declaration** | long TmOpenDeviceBySerial(char* pSerial); |
| **Arguments** | |
| pSerial | 9-digit serial number string (NULL-terminated) |
| **Return value** | 1 or greater: Device number 0: Failure |
| **Notes** | The serial number is a 9-digit string noted on the main unit or sensor unit. The string is specified as a char (8-bit) array of ASCII characters (NULL-terminated). Do not use 2-byte characters. After the instrument has been opened, the indicator on the main unit will change from red to green. |

**Example use**

```
char szSerial[10] = “100730001”;
long lDeviceId = TmOpenDeviceBySerial(szSerial);
if (lDeviceId <= 0) {
        //Error processing
}

//Configuration, execution processing, etc.

TmCloseDevice(lDeviceId);
```

## TmCloseDevice

| | |
|---|---|
| **Description** | Closes the TM6101. If the instrument is reopened after being closed, all measurement conditions will be initialized. |
| **Declaration** | long TmCloseDevice(long lDeviceId); |
| **Arguments** | |
| lDeviceId | Device number |
| **Return value** | 1: Success 0: Failure |
| **Note** | After the instrument has been closed, the indicator on the main unit will change from green to red. |

## 4.2 Measurement Conditions

### TmSetMeasMode

**Description** Changes the measurement mode. Measurements should usually be made in normal measurement mode.

**Declaration** long TmSetMeasMode(long lDeviceId, char cMeasMode);

**Arguments**

lDeviceId Device number

cMeasMode Measurement mode

0: Normal measurement mode (default) 1: AC measurement mode

**Return value** 1: Success 0: Failure

**Note** When making measurements in AC measurement mode, select AC measurement mode with the TmSetAcMode function after first setting the instrument to AC measurement mode with this function. Since the instrument defaults to normal measurement mode when it is opened, it is not necessary to call this function when using normal measurement mode only.

**Example use**

**・When making measurements in normal measurement mode**

```
long lRet;
lRet = TmSetMeasMode(lDeviceId, 0);          //Normal measurement mode
lRet = TmSetIntegralTime(lDeviceId, 2);      //Integration time: 1 ms
for (char nCh = 0; nCh < 16; nCh++){
    lRet = TmSetSensitivity(lDeviceId, nCh, 0);   //Low sensitivity
}
lRet = TmSetAverageNum(lDeviceId, 1);        //No averaging
```

**・When making measurements in AC measurement mode**

```
long lRet;
lRet = TmSetMeasMode(lDeviceId, 1);          //AC measurement mode
lRet = TmSetAcMode(lDeviceId,  0,  1, 10);   //Range 1, 50 Hz, 10 average times
```

### TmGetMeasMode

**Description** Acquires the current measurement mode.

**Declaration** long TmGetMeasMode(long lDeviceId, char* pcMeasMode);

**Arguments**

lDeviceId Device number

pcMeasMode Returns the current measurement mode.

0: Normal measurement mode (default) 1: AC measurement mode

**Return value** 1: Success 0: Failure

## TmSetIntegralTime

**Description** Sets the integration time. This parameter can be set when using normal measurement mode.

**Declaration** long TmSetIntegralTime(long lDeviceId, char cIntTimeIndex);

**Arguments**

| | |
|---|---|
| lDeviceId | Device number |
| cIntTimeIndex | Integration time<br>0: 0.1 ms 1: 0.5 ms 2: 1 ms (default) 3: 2 ms 4: 4 ms<br>5: 8 ms 6: 10 ms 7: 16.6 ms 8: 20 ms 9: 33.3 ms 10: 40 ms |

**Return value** 1: Success 0: Failure

**Note** When AC measurement mode is the current measurement mode, switch to normal measurement mode before calling this function. For more information about how to use this function, see the example use provided for the TmSetMeasMode function.

## TmGetIntegralTime

**Description** Acquires the current integration time.

**Declaration** long TmGetIntegralTime(long lDeviceId, char* pcIntTimeIndex);

**Arguments**

| | |
|---|---|
| lDeviceId | Device number |
| pcIntTimeIndex | Returns the current integration time.<br>0: 0.1 ms 1: 0.5 ms 2: 1 ms (default) 3: 2 ms 4: 4 ms<br>5: 8 ms 6: 10 ms 7: 16.6 ms 8: 20 ms 9: 33.3 ms<br>10: 40 ms |

**Return value** 1: Success 0: Failure

**Note** The optimal integration time is set automatically when performing measurements with integration time auto-ranging. This function can be used to acquire the current integration time setting. For an example of how to use this function, see the TmSetAutoRange function.

## TmSetSensitivity

**Description** Sets the sensitivity range. This parameter can be set in normal measurement mode.

**Declaration** long TmSetSensitivity(long lDeviceId, char nCh, char cSens);

**Arguments**

lDeviceId — Device number

nCh — Sensor number

Specify value from 0 to 15, corresponding to sensors 1 through 16.

cSens — Sensitivity range

0: High sensitivity 1: Low sensitivity

**Return value** 1: Success 0: Failure

**Note** When AC measurement mode is the current measurement mode, switch to normal measurement mode before calling this function. For more information about how to use this function, see the example use provided for the TmSetMeasMode function. The instrument makes measurements using 16 optical sensors, each of which has different optical characteristics. Sensitivity range settings are made in pairs, with corresponding sensors sharing the same sensitivity range setting. For example, if the sensitivity range for sensor 1 is set to low, the sensitivity range for sensor 2 will also be set to low. (Conversely, if the sensitivity range for sensor 2 is set to high, the sensitivity range for sensor 1 will also be set to high.)

| Sensor 1<br>Sensor 2 | Sensor 3<br>Sensor 4 | Sensor 5<br>Sensor 6 | Sensor 7<br>Sensor 8 | Sensor 9<br>Sensor 10 | Sensor 11<br>Sensor 12 | Sensor 13<br>Sensor 14 | Sensor 15<br>Sensor 16 |
|---|---|---|---|---|---|---|---|
| High/Low | High/Low | High/Low | High/Low | High/Low | High/Low | High/Low | High/Low |

## TmGetSensitivity

**Description** Acquires the current sensitivity range.

**Declaration** long TmGetSensitivity(long lDeviceId, char nCh, char* pcSens)

**Arguments**

lDeviceId — Device number

nCh — Sensor number

Specify value from 0 to 15, corresponding to sensors 1 through 16.

pcSens — Returns the current sensitivity range.

0: High sensitivity 1: Low sensitivity

**Return value** 1: Success 0: Failure

**Note** The optimal sensitivity range is set automatically when performing measurements with sensitivity range auto-ranging. This function can be used to acquire the current sensitivity range setting. For an example of how to use this function, see the TmSetAutoRange function.

### TmSetAverageNum

| | |
|---|---|
| **Description** | Sets the average times. This parameter can be set in normal measurement mode. |
| **Declaration** | long TmSetAverageNum(long lDeviceId, long lAveNum); |
| **Arguments** | |
| lDeviceId | Device number |
| lAveNum | Average times<br>1: No averaging (default) 2 to 100: Average times |
| **Return value** | 1: Success 0: Failure |

### TmGetAverageNum

| | |
|---|---|
| **Description** | Acquires the current average times. |
| **Declaration** | long TmGetAverageNum(long lDeviceId); |
| **Arguments** | |
| lDeviceId | Device number |
| **Return value** | Average times: 1 to 100 (0: Failure) |
| **Note** | To acquire the average times in AC measurement mode, use the TmGetAcMode function. |

### TmSetTrigType

| | |
|---|---|
| **Description** | Configures the external trigger. |
| **Declaration** | long TmSetTrigType(long lDeviceId, char cTrigType); |
| **Arguments** | |
| lDeviceId | Device number |
| cTrigType | 0: External trigger OFF (default) 1: External trigger ON (rising edge)<br>2: External trigger ON (falling edge) |
| **Return value** | 1: Success 0: Failure |
| **Note** | When the external trigger is set to ON, external trigger monitoring is started by executing the TmExecMeas function. |

### TmGetTrigType

| | |
|---|---|
| **Description** | Acquires the current external trigger setting. |
| **Declaration** | long TmGetTrigType(long lDeviceId, char* pcTrigType); |
| **Arguments** | |
| lDeviceId | Device number |
| pcTrigType | Returns the current external trigger setting.<br>0: External trigger OFF (default) 1: External trigger ON (rising edge)<br>2: External trigger ON (falling edge) |
| **Return value** | 1: Success 0: Failure |

## TmSetTrigDelay

| | |
|---|---|
| **Description** | Sets the trigger delay. |
| **Declaration** | long TmSetTrigDelay(long lDeviceId, long lDelay); |
| **Arguments** | |
| lDeviceId | Device number |
| lDelay | Trigger delay (ms): 0 to 1,000 (default: 0 ms) |
| **Return value** | 1: Success 0: Failure |
| **Note** | The trigger delay setting is valid when the external trigger is ON. The trigger delay does not function when the external trigger is OFF. |

## TmGetTrigDelay

| | |
|---|---|
| **Description** | Acquires the current trigger delay setting. |
| **Declaration** | long TmGetTrigDelay(long lDeviceId, long* plDelay); |
| **Arguments** | |
| lDeviceId | Device number |
| plDelay | Returns the current trigger delay setting (ms). |
| **Return value** | 1: Success 0: Failure |

## TmSetTrigTimeout

| | |
|---|---|
| **Description** | Sets the external trigger timeout time. |
| **Declaration** | long TmSetTrigTimeout(long lDeviceId, long lTimeout); |
| **Arguments** | |
| lDeviceId | Device number |
| lTimeout | Trigger timeout time (ms): 10,000 to 1,000,000 (default: 100,000 ms) |
| **Return value** | 1: Success 0: Failure |
| **Note** | Once external trigger monitoring is started with a measurement function, measurement will be forcibly terminated if no external trigger has been input when the trigger timeout time elapses. |

## TmGetTrigTimeout

| | |
|---|---|
| **Description** | Acquires the current external trigger timeout time. |
| **Declaration** | long TmGetTrigTimeout(long lDeviceId, long* plTimeout); |
| **Arguments** | |
| lDeviceId | Device number |
| plTimeout | Returns the trigger timeout time (ms). |
| **Return value** | 1: Success 0: Failure |

## TmSetAutoRange

**Description** Configures auto-ranging.

**Declaration** long TmSetAutoRange(long lDeviceId, char cAutoRangeType);

**Arguments**

| | |
|---|---|
| lDeviceId | Device number |
| cAutoRangeType | 0: Off (default) |
| | 1: Integration time auto-ranging |
| | 2: Sensitivity auto-ranging |

**Return value** 1: Success 0: Failure

**Note** Auto-ranging cannot be used in AC measurement mode or when the external trigger is enabled. The optimal integration time or sensitivity range is automatically set when making measurements using auto-ranging. The TmGetIntegralTime and TmGetSensitivity functions can be used to acquire the current settings.

**Example use**

```
long lRet;
char cIntegralTime;        //Integration time
char cSens[16];  //Sensitivity range

lRet = TmSetAutoRange(lDeviceId,1);        //Integration time auto-ranging
lRet = TmMeasExec();                       //Make measurement using integration
                                           time auto-ranging

lRet = TmGetIntegralTime(lDeviceId, &cIntegralTime);        //Acquire integration time

lRet = TmSetAutoRange(lDeviceId,2);        //Sensitivity auto-ranging
lRet = TmMeasExec();                       //Make measurement using sensitivity
                                           auto-ranging

for (char nCh = 0; nCh < 16; nCh++) {
        lRet = TmGetSensitivity(lDeviceId, nCh, &cSens[nCh]); //Acquire sensitivity
                                                              range
}
```

## TmGetAutoRange

| | |
|---|---|
| **Description** | Acquires the auto-ranging setting. |
| **Declaration** | long TmGetAutoRange(long lDeviceId, char* pcAutoRangeType); |
| **Arguments** | |
| lDeviceId | Device number |
| pcAutoRangeType | Returns the auto-ranging setting.<br>0: Off (default)<br>1: Integration time auto-ranging<br>2: Sensitivity auto-ranging |
| **Return values** | 1: Success 0: Failure |

## TmSetAutoRangeLevel

| | |
|---|---|
| **Description** | Sets the auto-ranging detection level upper and lower limits. |
| **Declaration** | long TmSetAutoRangeLevel(long lDeviceId, char cLevelHigh, char cLevelLow); |
| **Arguments** | |
| lDeviceId | Device number |
| cLevelHigh | Auto-ranging upper limit (%): 1 to 99 (default: 90%) |
| cLevelLow | Auto-ranging lower limit (%): 1 to 99 (default: 10%) |
| **Return values** | 1: Success 0: Failure |
| **Note** | When using integration time auto-ranging, the integration time is automatically adjusted so that the detection levels of all sensors are greater than or equal to the lower limit while not exceeding the upper limit. When using sensitivity auto-ranging, the sensitivity (high/low) is automatically adjusted so that the detection levels of individual sensors are greater than or equal to the lower limit while not exceeding the upper limit. In order for auto-ranging to function properly, the upper limit should be set to at least twice the lower limit (for example, if the lower limit is 30%, use an upper limit of at least 60%). Setting either cLevelHigh or cLevelLow to 0 causes the settings to revert to their default values (upper limit of 90%, lower limit of 10%). |

## TmGetAutoRangeLevel

| | |
|---|---|
| **Description** | Acquires the auto-ranging detection level upper and lower limits. |
| **Declaration** | long TmGetAutoRangeLevel(long lDeviceId, char* pcLevelHigh, char* pcLevelLow); |
| **Arguments** | |
| lDeviceId | Device number |
| pcLevelHigh | Returns the auto-ranging upper limit (%). |
| pcLevelLow | Returns the auto-ranging lower limit (%). |
| **Return values** | 1: Success 0: Failure |

## TmSetAcMode

| | |
|---|---|
| **Description** | Configures AC measurement mode. |
| **Declaration** | long TmSetAcMode(long lDeviceId, char cAcRange, char cAcPlc, long lAveNum); |
| **Arguments** | |
| lDeviceId | Device number |
| cAcRange | AC measurement mode range<br>0: Range 1 (default) 1: Range 2 2: Range 3 |
| cAcPlc | Power supply frequency 0: 60 Hz 1: 50 Hz (default) |
| lAveNum | AC measurement mode averaging times: 1 to 100 (default: 1) |
| **Return values** | 1: Success 0: Failure |
| **Note** | Use this function to make settings after switching to AC measurement mode with the TmSetMeasMode function. |

## TmGetAcMode

| | |
|---|---|
| **Description** | Acquires AC measurement mode settings. |
| **Declaration** | long TmGetAcMode(long lDeviceId, char* pcAcRange, char* pcAcPlc, long* plAveNum); |
| **Arguments** | |
| lDeviceId | Device number |
| pcAcRange | Returns the AC measurement mode range.<br>0: Range 1 1: Range 2 2: Range 3 |
| pcAcPlc | Returns the power supply frequency.<br>0: 60 Hz 1: 50 Hz |
| plAveNum | Returns the AC measurement mode averaging times. 1 to 100 |
| **Return value** | 1: Success 0: Failure |

## TmSetRefIlluminant

| | |
|---|---|
| **Description** | Sets the reference light source to use for color rendering index calculations. |
| **Declaration** | long TmSetRefIlluminant(long lDeviceId, char cType); |
| **Arguments** | |
| lDeviceId | Device number |
| cType | 0: CIE daylight 1: Blackbody radiation<br>2: Automatic selection (blackbody radiation < 5,000 K ≤ CIE daylight) (default) |
| **Return value** | 1: Success 0: Failure |

## TmGetRefIlluminant

| | |
|---|---|
| **Description** | Acquires the reference light source to use for color rendering index calculations. |
| **Declaration** | long TmGetRefIlluminant(long lDeviceId, char* pcSet); |
| **Arguments** | |
| lDeviceId | Device number |
| pcSet | Returns the reference light source.<br>0: CIE daylight 1: Blackbody radiation 2: Automatic selection |
| **Return value** | 1: Success 0: Failure |
| **Note** | |

## TmSetLightDistance

| | |
|---|---|
| **Description** | Sets the light measurement distance to use when calculating the luminous intensity. |
| **Declaration** | long TmSetLightDistance(long lDeviceId, double dDistance); |
| **Arguments** | |
| lDeviceId | Device number |
| dDistance | Light measurement distance (m): 0.01 to 10.00 (default: 0.01 [m]) |
| **Return value** | 1: Success 0: Failure |

## TmGetLightDistance

| | |
|---|---|
| **Description** | Acquires the light measurement distance to use when calculating the luminous intensity. |
| **Declaration** | long TmGetLightDistance(long lDeviceId, double* pdDistance); |
| **Arguments** | |
| lDeviceId | Device number |
| pdDistance | Returns the light measurement distance (m). |
| **Return value** | 1: Success 0: Failure |

## TmSetExtIoIndexOutpTime

| | |
|---|---|
| **Description** | Sets the on time for external I/O measurement complete output (index output). |
| **Declaration** | long TmSetExtIoIndexOutpTime(long lDeviceId, DWORD dwTimeMsec); |
| **Arguments** | |
| lDeviceId | Device number |
| dwTimeMsec | On time (ms): 1 to 100 (default: 1 ms) |
| **Return value** | 1: Success 0: Failure |
| **Note** | Measurement complete output is disabled during dark measurement. |

## TmGetExtIoIndexOutpTime

| | |
|---|---|
| **Description** | Returns the on time for external I/O measurement complete output (index output). |
| **Declaration** | long TmGetExtIoIndexOutpTime(long lDeviceId, DWORD* pdwTimeMsec) |
| **Arguments** | |
| lDeviceId | Device number |
| pdwTimeMsec | Returns the on time (ms). |
| **Return value** | 1: Success 0: Failure |

## TmSetMeasSettingAll

| | |
|---|---|
| **Description** | Sets all measurement conditions at once. |
| **Declaration** | long TmSetMeasSettingAll(long lDeviceId, TM_MEAS_SET stMeasSet); |
| **Arguments** | |
| lDeviceId | Device number |
| stMeasSet | Specify a measurement conditions structure with settings for all member variables. |
| **Return value** | 1: Success 0: Failure |
| **Note** | Specify all measurement conditions in the TM_MEAS_SET structure. This functionality is used to reconfigure settings acquired the last time a TM6101 was connected using the TmGetMeasSettingAll function the next time the instrument is connected. |

**Example use**

```
TM_MEAS_SET    stMeasSet;                  //Measurement conditions structure
stMeasSet.dwMeasMode = 0;                  //Normal measurement mode
stMeasSet. dwIntTime = 2;                  //Integration time: 1 ms
  //Continue setting all member variables
        .
        .
TmSetMeasSettingAll(lDeviceId,    stMeasSet);          //Set all measurement conditions
```

## TmGetMeasSettingAll

| | |
|---|---|
| **Description** | Acquires all current measurement settings at once. |
| **Declaration** | long TmGetMeasSettingAll(long lDeviceId, TM_MEAS_SET* pstMeasSet); |
| **Arguments** | |
| lDeviceId | Device number |
| pstMeasSet | Returns a measurement conditions structure. Specify a pointer to a measurement conditions structure. |
| **Return value** | 1: Success 0: Failure |

**Example use**

```
TM_MEAS_SET    stMeasSet;                          //Measurement conditions structure
TmSetMeasSettingAll(lDeviceId,    &stMeasSet);     //Acquire all measurement conditions
                                                      at once
```

## TmInitializeMeasSettings

| | |
|---|---|
| **Description** | Initializes measurement conditions. |
| **Declaration** | long TmInitializeMeasSettings(long lDeviceId); |
| **Arguments** | |
| lDeviceId | Device number |
| **Return value** | 1: Success 0: Failure |
| **Note** | See individual function descriptions for return values. Initialization cannot be performed while measurement is in progress. |

## 4.3 Measurement Execution

### TmMeasExec

| | |
|---|---|
| **Description** | Performs measurement. When the external trigger is enabled, starts monitoring of the external trigger. The function does not return until either measurement completes or the trigger timeout time elapses. |
| **Declaration** | long TmMeasExec(long lDeviceId); |
| **Arguments** | |
| lDeviceId | Device number |
| **Return value** | 1: Success 0: Failure |
| **Note** | Measurement results can be acquired once measurement completes. |

**Example use**

**・Make 10 measurements and terminate:**

```
long lDeviceId = TmOpenDevice ();              //Open the TM6101 device
if (lDeviceId <= 0) {
        //Error processing
}

long lRet;
double x, y;

lRet = TmSetAutoRange(lDeviceId,1);                    //Integration time auto-ranging

        //Use default values for other measurement conditions

lRet = TmExecDarkMeas(lDeviceId, 10, 1);    //Perform dark measurement for all ranges
                                             (averaging times: 10)

for (long nNum = 0; nNum < 10; nNum++) {
    lRet = TmMeasExec(lDeviceId);                    //Perform measurement using
                                                      integration time auto-ranging
    lRet = TmGetChromaticityValue_xy(lDeviceId, &x,&y);      /Acquire xy chromaticity
                                                              values
}
    //Processing to display measurement results, etc.
        .
        .

TmCloseDevice(lDeviceId);              //Close device
```

## TmMeasExecAsync

**Description** Starts measurement. When the external trigger is enabled, starts monitoring of the external trigger. The function returns immediately.

**Declaration** long TmMeasExecAsync(long lDeviceId);

**Arguments**

lDeviceId Device number

**Return value** 1: Success 0: Failure

**Note** Be sure to verify the completion of measurement with the TmGetMeasStatus function before acquiring measurement results. Measurement results can be acquired once measurement completes. Call TmCancelMeas to terminate monitoring of the external trigger. If measurement is canceled with the TmCancelMeas function while monitoring of the external trigger is in progress, the measurement results acquisition function will return results for the previous measurement. Before starting the next measurement, verify that the instrument is in the measurement standby state with the TmGetStandbyStatus function and then start measurement with TmMeasExecAsync.

**Example use**

**・Make 10 measurements with the asynchronous function:**

```
long lRet;
double x, y;                    //Chromaticity values
DWORD dwStatus;                 //Measurement status

    //Make measurements using default measurement conditions

for (long nNum = 0; nNum < 10; nNum++) {        //Make 10 measurements
    lRet = TmMeasExecAsync (lDeviceId);         // Start measurement
                                                 asynchronously

    do {
        lRet = TmGetMeasStatus(lDeviceId,  &dwStatus, 0);  //Acquire measurement
                                                            status
    } while (dwStatus == 0);            //If measurement in progress, acquire measurement
                                         status repeatedly

    //Acquire measurement results
    lRet = TmGetChromaticityValue_xy(lDeviceId, &x,&y);

    do {
        lRet = TmGetStandbyStatus(lDeviceId,  &dwStatus, 0);  //Detect standby state
    } while (dwStatus == 0);            //Acquire status repeatedly until the standby state is
                                         detected
}
```

## TmGetMeasStatus

| | |
|---|---|
| **Description** | Acquires the current measurement status. |
| **Declaration** | long TmGetMeasStatus(long lDeviceId, DWORD* pdwStatus,<br>DWORD dwMilliseconds); |
| **Arguments** | |
| lDeviceId | Device number |
| pdwStatus | Returns the measurement status.<br>0: Measurement or trigger monitoring in progress<br>1: Measurement completed normally<br>2: Measurement terminated with an error |
| dwMilliseconds | Timeout time (ms): 0 to 0xFFFFFFFF<br>The function returns when the timeout time elapses or measurement completes. Specifying 0 causes the function to return immediately. Specifying 0xFFFFFFFFcauses the function to not return until measurement completes. |
| **Return value** | 1: Success 0: Failure |
| **Note** | When starting measurement with TmMeasExecAsync or TmExecDarkMeasAsync, execute this function until it generates a return value of 1 or 2. To forcibly terminate measurement or trigger monitoring, call TmCancelMeas. When measurement is forcibly terminated with the TmCancelMeas function, the measurement status is returned as 2 (measurement terminated with an error). When external trigger monitoring is terminated, the measurement status is returned as 1 (measurement completed normally). For more information about how to use this function, see the TmMeasExecAsync function. |

## TmGetStandbyStatus

| | |
|---|---|
| **Description** | Detects the measurement standby state. |
| **Declaration** | long TmGetStandbyStatus(long lDeviceId, DWORD* pdwStatus,<br>DWORD dwMilliseconds); |
| **Arguments** | |
| lDeviceId | Device number |
| pdwStatus | Returns the measurement standby state.<br>0: Measurement in progress<br>1: Standby |
| dwMilliseconds | Timeout time (ms): 0 to 0xFFFFFFFF<br>The function returns when the timeout time elapses or the instrument changes to the measurement standby state. Specifying 0 causes the function to return immediately. Specifying 0xFFFFFFFFcauses the function to not return until the instrument changes to the measurement standby state. |
| **Return value** | 1: Success 0: Failure |
| **Note** | When the measurement standby state is returned as 1 (standby state), measurement can be started with the TmMeasExecAsync or TmDarkMeasExecAsync function. For more information about how to use this function, see the TmMeasExecAsync function. |

## TmCancelMeas

| | |
|---|---|
| **Description** | Cancels measurement processing when performing measurement asynchronously. |
| **Declaration** | long TmCancelMeas(long lDeviceId); |
| **Arguments** | |
| lDeviceId | Device number |
| **Return value** | 1: Success 0: Failure |
| **Note** | Cancels measurement processing started with the TmMeasExecAsync or TmExecDarkMeasAsync function. When external trigger monitoring is in progress, terminates external trigger monitoring. After calling this function, monitor the measurement status with TmGetMeasStatus until measurement processing completes. This function is not normally used. Instead, acquire the measurement status with the TmGetMeasStatus function and wait for measurement to complete. |

## TmExecDarkMeas

| | |
|---|---|
| **Description** | Performs dark measurement. The function does not return until dark measurement completes. |
| **Declaration** | long TmExecDarkMeas(long lDeviceId, unsigned int nAveNum, |
| char nAllRange); | |
| **Arguments** | |
| lDeviceId | Device number |
| nAveNum | Averaging times: 1 to 100 |
| nAllRange | 0: Performs dark measurement for the current integration time and sensitivity range.<br>1: Performs dark measurement for all integration times, sensitivity ranges, and AC measurement ranges. |
| **Return value** | 1: Success 0: Failure |
| **Note** | Dark measurement is disabled while the external trigger is enabled. Additionally, measurement complete output (index output) is not generated after dark measurement completes. Until dark measurement is performed after opening an instrument with TmOpenDevice or a similar function, the default dark values (the values set at the time of shipment) are applied to measurement results. It is recommended to perform dark measurement every time an instrument is connected. If dark measurement is performed for the current integration time and sensitivity range, it will need to be repeated if the integration time, sensitivity range, or measurement mode is changed. For more information about how to use this function, see the TmMeasExec function. |

## TmExecDarkMeasAsync

| | |
|---|---|
| **Description** | Performs dark measurement for all integration times, sensitivity ranges, and AC measurement ranges. Processing is performed asynchronously, so the function returns immediately. |
| **Declaration** | long TmExecDarkMeasAsync(long lDeviceId, unsigned int nAveNum); |
| **Arguments** | |
| lDeviceId | Device number |
| nAveNum | Averaging times: 1 to 100 |
| **Return value** | 1: Success 0: Failure |
| **Note** | Be sure to verify that dark measurement has completed with the TmGetMeasStatus function. Dark measurement is disabled while the external trigger is enabled. Additionally, measurement complete output (index output) is not generated after dark measurement completes. Until dark measurement is performed after opening an instrument with TmOpenDevice or a similar function, the default dark values (the values set at the time of shipment) are applied to measurement results. It is recommended to perform dark measurement every time an instrument is connected. |

**Example use**

```
long lRet;
DWORD dwStatus;          //Measurement status

lRet = TmExecDarkMeasAsync (lDeviceId,  10);  // Start asynchronous
                                                measurement (averaging times: 10)

do {
    lRet = TmGetMeasStatus(lDeviceId,  &dwStatus, 0);  //Acquire measurement
                                                          status
} while (dwStatus == 0);   //Repeatedly acquire the measurement status while
measurement is in progress

do {
    lRet = TmGetStandbyStatus(lDeviceId,  &dwStatus, 0);  //Detect the standby
                                                             state
} while (dwStatus == 0);   //Repeatedly acquire the status until the instrument changes
                             to the standby state

//// End of dark measurement processing for all ranges ////
```

## TmGetDarkAll

| | |
|---|---|
| **Description** | Acquires dark values for all normal measurement mode integration times, sensitivity ranges, and AC measurement ranges. |
| **Declaration** | long TmGetDarkAll(long lDeviceId, DWORD dwDarkDataAll[ ]); |
| **Arguments** | |
| lDeviceId | Device number |
| dwDarkDataAll[ ] | Specify an array for storing the dark values.<br>Specify DWORD dwDarkDataAll[448]（DWORD×448). |
| **Return value** | 1: Success 0: Failure |
| **Note** | Dark values can be acquired after performing dark measurement for all integration times and sensitivity ranges with the TmExecDarkMeas or TmExecDarkMeasAsync function. |
| **Example use** | |

```
dwDarkDataAll[ 448];                    //Array for storing dark results
long lRet = TmGetDarkAll(lDeviceId,  dwDarkDataAll);    //Acquire dark measurement
                                                          results
```

## TmSetDarkAll

| | |
|---|---|
| **Description** | Sets dark values for all integration times, sensitivity ranges, and AC measurement ranges. |
| **Declaration** | long TmSetDarkAll(long lDeviceId, DWORD dwDarkDataAll[ ]); |
| **Arguments** | |
| lDeviceId | Device number |
| dwDarkDataAll[ ] | Specify an array storing the desired dark values.<br>Specify DWORD dwDarkDataAll[448]（DWORD×448). |
| **Return value** | 1: Success 0: Failure |
| **Note** | This function is used to reconfigure the TM6101 with the dark values acquired using TmGetDarkAll the last time the instrument was connected. |

## TmResetDark

| | |
|---|---|
| **Description** | Clears dark measurement results and resets the instrument to its state before dark measurement was performed. |
| **Declaration** | long TmResetDark(long lDeviceId); |
| **Arguments** | |
| lDeviceId | Device number |
| **Return value** | 1: Success 0: Failure |
| **Note** | Executing this function causes the default dark values (the values set at the time of shipment) to be applied to measurement results. |

## 4.4 Reference Value Correction

### TmExecChromaticityCorrect

**Description** Performs chromaticity correction. Specify the spectral characteristics data for the chromaticity correction target light source as an array.

**Declaration** long TmExecChromaticityCorrect(long lDeviceId, double dSpectramData[ ]);

**Arguments**

lDeviceId Device number

dSpectramData [ ] Specify an array storing the spectral characteristics data.
Data is required every 5 nm for 380 nm to 780 nm (81 data points). Specify double dData[ 81] (double×81).

**Return value** 1: Success 0: Failure

**Note** The latest measurement results are applied to chromaticity correction. Before performing chromaticity correction, measure the chromaticity correction target light source.

**Example use**

```
long lRet;
double dSpectramData [ 81];          //Spectral characteristics data for the chromaticity
                                     correction target light source

for (int nDataNo = 0; nDataNo < 81; nDataNo++)
        dSpectramData [ 81] = ・・・・;      //Register the spectral data

lRet = TmSetAutoRange(lDeviceId,1);          //Integration time auto-ranging
lRet = TmSetAverageNum(lDeviceId,5);         //Averaging times; 5

        // Make measurements using default measurement conditions

lRet = TmExecDarkMeas(lDeviceId, 10, 1);     //Perform dark measurement for all ranges
                                             (averaging times: 10)

//Measure the correction target light source
lRet = TmMeasExec(lDeviceId);                //Perform measurement using integration
                                             time auto-ranging

//Perform chromaticity correction
lRet = TmExecChromaticityCorrect(lDeviceId,  dSpectramData);

    //Perform measurement and other processing
```

## TmExecChromaticityCorrectByFile

| | |
|---|---|
| **Description** | Performs chromaticity correction. Specify a CSV file storing the spectral characteristics data for the chromaticity correction target light source. |
| **Declaration** | long TmExecChromaticityCorrectByFile(long lDeviceId, char* cFilePath); |
| **Arguments** | |
| lDeviceId | Device number |
| cFilePath | Specify the full pathname for a CSV file storing the spectral characteristics data. This file must contain wavelength and spectral characteristics data for 380 nm to 780 nm (every 5 nm) (81 data points). |
| **Return value** | 1: Success 0: Failure |
| **Note** | For more information about the file format, see the instrument's instruction manual. The latest measurement results are applied to chromaticity correction. Before performing chromaticity correction, measure the chromaticity correction target light source. |

## TmGetChromaticityCorrectValue

| | |
|---|---|
| **Description** | Acquires chromaticity correction values. |
| **Declaration** | long TmGetChromaticityCorrectValue(long lDeviceId, double dData[ ]); |
| **Arguments** | |
| lDeviceId | Device number |
| dData[ ] | Specify an array for storing the chromaticity correction values (double×16). Specify double dData[ 16] (double×16). |
| **Return value** | 1: Success 0: Failure (or correction not performed) |

## TmSetChromaticityCorrectValue

| | |
|---|---|
| **Description** | Sets chromaticity correction values. |
| **Declaration** | long TmSetChromaticityCorrectValue(long lDeviceId, double dData[ ]); |
| **Arguments** | |
| lDeviceId | Device number |
| dData[ ] | Specify an array storing the chromaticity correction values (double×16). Specify double dData[ 16] (double×16). |
| **Return value** | 1: Success 0: Failure |
| **Note** | This function is used to reconfigure the TM6101 with correction values acquired using the TmSetChromaticityCorrectValue function the last time the instrument was connected. |

## TmResetChromaticityCorrect

| | |
|---|---|
| **Description** | When chromaticity correction has been performed, clears the chromaticity correction values and reverts the instrument to its state before chromaticity correction was performed. |
| **Declaration** | long TmResetChromaticityCorrect(long lDeviceId); |
| **Arguments** | |
| lDeviceId | Device number |
| **Return value** | 1: Success 0: Failure |

## TmExecIlluminanceCorrect

| | |
|---|---|
| **Description** | Performs illuminance correction. Specify the illuminance value for the correction target light source. |
| **Declaration** | long TmExecIlluminanceCorrect(long lDeviceId, double dIlluminance); |
| **Arguments** | |
| lDeviceId | Device number |
| dIlluminance | Correction target illuminance value (lx) |
| **Return value** | 1: Success 0: Failure |
| **Note** | The latest measurement results are applied to illuminance correction. Before performing illuminance correction, measure the illuminance correction target light source. |

**Example use**

```
long lRet;

lRet = TmSetAutoRange(lDeviceId,1);                 //Integration time auto-ranging
lRet = TmSetAverageNum(lDeviceId,5);                //Averaging times: 5

        // Make measurements using default measurement conditions

lRet = TmExecDarkMeas(lDeviceId, 10, 1);    //Perform dark measurement for all ranges
                                            (averaging times: 10)

//Measure the correction target light source
lRet = TmMeasExec(lDeviceId);               //Perform measurement using integration
                                            time auto-ranging

//Perform illuminance correction
lRet = TmExecIlluminanceCorrect (lDeviceId,   1000);        //Correct using 1,000 lx

// Perform measurement and other processing
```

## TmGetIlluminanceCorrectValue

| | |
|---|---|
| **Description** | Acquires the illuminance correction value. |
| **Declaration** | long TmGetIlluminanceCorrectValue(long lDeviceId, double* pdData); |
| **Arguments** | |
| lDeviceId | Device number |
| pdData | Returns the illuminance correction value. |
| **Return value** | 1: Success 0: Failure (or correction not performed) |

## TmSetIlluminanceCorrectValue

| | |
|---|---|
| **Description** | Sets the illuminance correction value. |
| **Declaration** | long TmSetIlluminanceCorrectValue(long lDeviceId, double dData); |
| **Arguments** | |
| lDeviceId | Device number |
| dData | Illuminance correction value |
| **Return value** | 1: Success 0: Failure |
| **Note** | This function is used to reconfigure the TM6101 with the correction value acquired using TmGetIlluminanceCorrectValue the last time the instrument was connected. |

## TmResetIlluminanceCorrect

| | |
|---|---|
| **Description** | When illuminance correction has been performed, clears the illuminance correction value and reverts the instrument to its state before illuminance correction was performed. |
| **Declaration** | long TmResetIlluminanceCorrect(long lDeviceId); |
| **Arguments** | |
| lDeviceId | Device number |
| **Return value** | 1: Success 0: Failure |

## TmExecLuminousFluxCorrect

| | |
|---|---|
| **Description** | Performs luminous flux correction. Specify the luminous flux value for the correction target light source. |
| **Declaration** | long TmExecLuminousFluxCorrect(long lDeviceId, double dLuminousFlux); |
| **Arguments** | |
| lDeviceId | Device number |
| dLuminousFlux | Correction target luminous flux value (lm) |
| **Return value** | 1: Success 0: Failure |
| **Note** | The latest measurement results are applied to luminous flux correction. Before performing luminous flux correction, measure the luminous flux correction target light source. For more information about how to use this function, see the TmExecIlluminanceCorrect function. The same processing sequence is used as for illuminance correction. |

### TmGetLuminousFluxCorrectValue

| | |
|---|---|
| **Description** | Acquires the luminous flux correction value. |
| **Declaration** | long TmGetLuminousFluxCorrectValue(long lDeviceId, double* pdData); |
| **Arguments** | |
| lDeviceId | Device number |
| pdData | Returns the luminous flux correction value. |
| **Return value** | 1: Success 0: Failure (or correction not performed) |

### TmSetLuminousFluxCorrectValue

| | |
|---|---|
| **Description** | Sets the luminous flux correction value. |
| **Declaration** | long TmSetLuminousFluxCorrectValue(long lDeviceId, double dData); |
| **Arguments** | |
| lDeviceId | Device number |
| dData | Luminous flux correction value |
| **Return value** | 1: Success 0: Failure |
| **Note** | This function is used to reconfigure the TM6101 with the correction value acquired using the TmGetLuminousFluxCorrectValue function the last time the instrument was connected. |

### TmResetLuminousFluxCorrect

| | |
|---|---|
| **Description** | When luminous flux correction has been performed, clears the luminous flux correction value and reverts the instrument to its state before luminous flux correction was performed. |
| **Declaration** | long TmResetLuminousFluxCorrect(long lDeviceId); |
| **Arguments** | |
| lDeviceId | Device number |
| **Return value** | 1: Success 0: Failure |

### TmExecLuminousIntensityCorrect

| | |
|---|---|
| **Description** | Performs luminous intensity correction. Specify the luminous intensity value for the correction target light source. |
| **Declaration** | long TmExecLuminousIntensityCorrect(long lDeviceId, |

double LuminousIntensity);

| | |
|---|---|
| **Arguments** | |
| lDeviceId | Device number |
| LuminousIntensity | Correction target luminous intensity value (cd) |
| **Return value** | 1: Success 0: Failure |
| **Note** | The latest measurement results are applied to luminous intensity correction. Before using this function, set the distance to the light source with the TmSetLightDistance function. Before performing luminous intensity correction, measure the luminous intensity correction light source. For more information about how to use this function, see the TmExecIlluminanceCorrect function. The same processing sequence is used as for illuminance correction. |

## TmGetLuminousIntensityCorrectValue

| | |
|---|---|
| **Description** | Acquires the luminous intensity correction value. |
| **Declaration** | long TmGetLuminousIntensityCorrectValue(long lDeviceId, double* pdData); |
| **Arguments** | |
| lDeviceId | Device number |
| pdData | Returns the luminous intensity correction value. |
| **Return value** | 1: Success 0: Failure (or correction not performed) |

## TmSetLuminousIntensityCorrectValue

| | |
|---|---|
| **Description** | Sets the luminous intensity correction value. |
| **Declaration** | long TmSetLuminousIntensityCorrectValue(long lDeviceId, double dData); |
| **Arguments** | |
| lDeviceId | Device number |
| dData | Luminous intensity correction value |
| **Return value** | 1: Success 0: Failure |
| **Note** | This function is used to reconfigure the TM6101 with the correction value acquired using the TmGetLuminousIntensityCorrectValue function the last time the instrument was connected. |

## TmResetLuminousIntensityCorrect

| | |
|---|---|
| **Description** | When luminous intensity correction has been performed, clears the luminous intensity correction value and reverts the instrument to its state before luminous intensity correction was performed. |
| **Declaration** | long TmResetLuminousIntensityCorrect(long lDeviceId); |
| **Arguments** | |
| lDeviceId | Device number |
| **Return value** | 1: Success 0: Failure |

## TmGetUserCorrectData

**Description** Acquires all reference value correction values (chromaticity correction values, illuminance correction value, luminous intensity correction value, and luminous flux correction value) at once.

**Declaration** long TmGetUserCorrectData(long lDeviceId,
TM_USER_CORRECT_DATA* pstUserCorrect);

**Arguments**

| | |
|---|---|
| lDeviceId | Device number |
| pstUserCorrect | Returns correction values.<br>Specify a TM_USER_CORRECT_DATA structure. |

**Return value** 1: Success 0: Failure

**Example use**

```
TM_USER_CORRECT_DATA  stUserCorrect);          //Reference value correction
                                                structure
TmGetUserCorrectData (lDeviceId,  &stUserCorrect);          //Acquires all correction
                                                values at once.
```

## TmSetUserCorrectData

**Description**

**Declaration** long TmSetUserCorrectData(long lDeviceId,
TM_USER_CORRECT_DATA stUserCorrect);

**Arguments**

| | |
|---|---|
| lDeviceId | Device number |
| stUserCorrect | Specify a TM_USER_CORRECT_DATA structure storing the correction values. |

**Return value** 1: Success 0: Failure

## 4.5 Acquiring Measurement Results

### TmGetIlluminanceValue

| | |
|---|---|
| **Description** | Acquires the illuminance value. |
| **Declaration** | long TmGetIlluminanceValue(long lDeviceId, double* pdData); |
| **Arguments** | |
| lDeviceId | Device number |
| pdData | Returns the illuminance value (lx). |
| **Return value** | 1: Success 0: Failure |

### TmGetLuminousIntensityValue

| | |
|---|---|
| **Description** | Acquires the luminous intensity value. |
| **Declaration** | long TmGetLuminousIntensityValue(long lDeviceId, double* pdData); |
| **Arguments** | |
| lDeviceId | Device number |
| pdData | Returns the luminous intensity value (cd). |
| **Return value** | 1: Success 0: Failure |

### TmGetLuminousFluxValue

| | |
|---|---|
| **Description** | Acquires the luminous flux value. |
| **Declaration** | long TmGetLuminousFluxValue(long lDeviceId, double* pdData); |
| **Arguments** | |
| lDeviceId | Device number |
| pdData | Returns the luminous flux value (lm). |
| **Return value** | 1: Success 0: Failure |

### TmGetTristimulusValues

| | |
|---|---|
| **Description** | Acquires tristimulus values. |
| **Declaration** | long TmGetTristimulusValues(long lDeviceId,<br>double* pX, double* pY, double* pZ); |
| **Arguments** | |
| lDeviceId | Device number |
| pX, pY, pZ | Acquires the tristimulus values (x, y, and z). |
| **Return value** | 1: Success 0: Failure |

### TmGetChromaticityValue_xy

| | |
|---|---|
| **Description** | Acquires the chromaticity values (x and y). |
| **Declaration** | long TmGetChromaticityValue_xy(long lDeviceId, double* pX, double* pY); |
| **Arguments** | |
| lDeviceId | Device number |
| pX, pY | Returns the chromaticity values (x and y). |
| **Return value** | 1: Success 0: Failure |

### TmGetChromaticityValue_uv

| | |
|---|---|
| **Description** | Acquires the chromaticity values (u and v). |
| **Declaration** | long TmGetChromaticityValue_uv(long lDeviceId, double* pU, double* pV); |
| **Arguments** | |
| lDeviceId | Device number |
| pU, pV | Returns the chromaticity values (u and v). |
| **Return value** | 1: Success 0: Failure |

### TmGetCorrelatedColorTemperature

| | |
|---|---|
| **Description** | Acquires the correlated color temperature and ∆uv value. |
| **Declaration** | long TmGetCorrelatedColorTemperature(long lDeviceId, double* pdTcp, double* pdDUV); |
| **Arguments** | |
| lDeviceId | Device number |
| pdTcp | Returns the correlated color temperature (K). |
| pdDUV | Returns the ∆uv value. |
| **Return value** | 1: Success 0: Failure |
| **Note** | If the absolute value of the ∆uv value is greater than 0.02, the correlated color temperature measurement result will indicate an error. |

### TmGetSpecialColorRenderingIndex

| | |
|---|---|
| **Description** | Acquires the special color rendering index Ri. |
| **Declaration** | long TmGetSpecialColorRenderingIndex(long lDeviceId, char nTestColorNo, double* pdData); |
| **Arguments** | |
| lDeviceId | Device number |
| nTestColorNo | Specify the test color (0 to 14, for test colors 1 to 15). |
| pdData | Returns the special color rendering index Ri. |
| **Return value** | 1: Success 0: Failure |

## TmGetGeneralColorRenderingIndex

| | |
|---|---|
| **Description** | Acquires the general color rendering index Ra. |
| **Declaration** | long TmGetGeneralColorRenderingIndex(long lDeviceId, double* pdData); |
| **Arguments** | |
| lDeviceId | Device number |
| pdData | Returns the average color rendering index Ra. |
| **Return value** | 1: Success 0: Failure |

## TmGetDominantWaveLength

| | |
|---|---|
| **Description** | Acquires the dominant wavelength and excitation purity. |
| **Declaration** | long TmGetDominantWaveLength(long lDeviceId, double* pdDomiLen, double* pdPurity); |
| **Arguments** | |
| lDeviceId | Device number |
| pdDomiLen | Returns the dominant wavelength (nm). |
| pdPurity | Returns the excitation purity (%). |
| **Return value** | 1: Success 0: Failure |

## TmGetMeasResultAll

| | |
|---|---|
| **Description** | Acquires all measurement results at once. |
| **Declaration** | long TmGetMeasResultAll(long lDeviceId, TM_MEAS_RESULT* pstMeasResult); |
| **Arguments** | |
| lDeviceId | Device number |
| pstMeasResult | Returns the measurement results in a TM_MEAS_RESULT structure. |
| **Return value** | 1: Success 0: Failure |
| **Note** | Measurement results and an indication of whether each result is valid or invalid are stored in the TM_MEAS_RESULT structure. |

**Example use**

```
TM_MEAS_RESULT  stMeasResult;                    //Measurement results structure

//Acquire all measurement results at once
long lRet =  TmGetMeasResultAll (lDeviceId,  & stMeasResult);
```

## TmGetDetectLevel

| | |
|---|---|
| **Description** | Acquires the detection level for each of the instrument's 16 sensors. |
| **Declaration** | long TmGetDetectLevel(long lDeviceId, char nCh, double* pdLevel); |
| **Arguments** | |
| lDeviceId | Device number |
| nCh | Sensor number (specify 0 to 15, for sensors 1 through 16). |
| pdLevel | Returns the detection level.<br>Detection level: 0.00 to 1.00<br>(0.00: Underflow 1.00: Overflow) |
| **Return value** | 1: Success 0: Failure |
| **Example use** | |

## 4.6 Acquiring the Instrument Status

### TmGetSerialNo

| | |
|---|---|
| **Description** | Acquires the serial number of a previously opened TM6101. |
| **Declaration** | long TmGetSerialNo(long lDeviceId, char* pSerial, DWORD nByteSize); |
| **Arguments** | |
| lDeviceId | Device number |
| pSerial | Returns the serial number as a string. Specify a char array of at least 16 bytes. |
| nByteSize | Number of bytes in the array specified with pSerial |
| **Return value** | Number of characters in the acquired serial number 0: Failure |
| **Note** | Returns an ASCII string (NULL-terminated) to the char array (8-bit). |

### TmCheckDevice

| | |
|---|---|
| **Description** | Acquires the status of a previously opened TM6101. |
| **Declaration** | long TmCheckDevice(long lDeviceId); |
| **Arguments** | |
| lDeviceId | Device number |
| **Return value** | 1: No error 0: Error |
| **Note** | This function returns a value of 0 (error) if an error has occurred with either the USB connection or the connection between the main unit and the sensor unit. Verify the nature of the error with the TmGetLastError function. |

## TmGetLastError

**Description** Acquires error information.

**Declaration** long TmGetLastError();

**Arguments** None

**Return value** Error number (0: No error)

**Note** Acquires a description of the error when an error is returned by a library function.

List of errors

| Error no. | Description |
|---|---|
| 1 | An invalid argument was specified for the function. Check the argument. |
| 16 | Dark measurement has not been performed. Dark correction values cannot be acquired. |
| 17 | Reference value correction has not been performed. Reference value correction values cannot be acquired. |
| 32 | The sensor unit is not connected. Check the sensor unit connection. |
| 33 | The serial numbers of the TM6101 main unit and sensor unit do not match. Check the TM6101 main unit and sensor unit serial numbers. |
| 34 | The sensor unit connection cable is connected backwards. Check the direction of the cable connection. |
| 35 | The TM6101 main unit or TM6101 with the specified serial number is not connected to the computer. Check the USB and other connections and verify that the driver software has been properly installed. |
| 64 | Measurement timed out. Verify that the main unit and sensor unit are properly connected. If performing external trigger measurement, check trigger input. |
| 65 | Measurement failed. Verify that the main unit and sensor unit are properly connected. |
| 66 | This error is returned by the measurement results acquisition and reference value correction functions if the instrument is unable to calculate color or perform reference value correction. |
| 67 | The measurement results are invalid, or measurement has not yet been performed. |
| 68 | Sensor detection level overflow. Set the integration time and sensitivity range to appropriate values and repeat measurement. |
| 69 | Sensor detection level underflow. The instrument may be malfunctioning. |
| 70 | The measurement results exceeded the instrument's rating (100,000 lx). Stop measurement as continuing may damage the instrument. |
| 71 | Calculation results are invalid since the $\Delta$uv value for the correlated color temperature exceeded 0.02. |
| 72 | Unable to start measurement since measurement is currently in progress, or the instrument is not in the measurement standby state. |
| 256 | Other error |

## 4.7 Structures

### Measurement conditions structure

```
typedef struct tagTmMeasSet
{
    DWORD   dwMeasMode;         //Measurement mode   0: Normal measurement mode
                                                       1: AC measurement mode
    DWORD   dwIntTime;          //Integration time
                                Specify 0 to 10 (corresponds to 0.1 ms to 40 ms)
    DWORD   dwAmpSens[16];      //Sensitivity range   0: High   1: Low
    DWORD   dwAveNum;           //Averaging times
    DWORD   dwPlc;              //Power supply frequency      0: 60 Hz   1: 50 Hz
    DWORD   dwAcRange;          //AC measurement mode measurement range
                                Specify 0 to 2 (corresponds to ranges 1 through 3)
    DWORD   dwAcAveNum;         //AC measurement mode averaging times
    DWORD   dwExtTrig;          //External trigger   0: Off   1: Rising edge   2: Falling edge
    DWORD   dwTrigDelay;        //Trigger delay (ms)
    DWORD   dwTrigTimeout;      //External trigger timeout (ms)
    DWORD   dwIndexOutpTime;    //External I/O index output on time (ms)
    DWORD   dwAutoRange;        //Auto-ranging        0: Auto-ranging off
                                1: Integration time auto-ranging
                                2: Amp sensitivity auto-ranging (Normal measurement
                                                                 mode only)
    DWORD   dwAutoLevelHigh;    //Auto-ranging detection level upper limit (%)
    DWORD   dwAutoLevelLow;     //Auto-ranging detection level lower limit (%)
    DWORD   dwStdIllumSel;      //Reference light   0: CIE daylight
                                1: Blackbody radiation   2: Automatic selection
    double  dLightDistance;     //Light measurement distance (m)
} TM_MEAS_SET, *PTM_MEAS_SET;
```

## Reference value correction value structure

```
typedef struct tagTmUserCorrectData
{
    DWORD   dwChromaticityCorrectEnable;        //Chromaticity correction enable/disable
                                                0: Disable          1: Enable
    double  dChromaticityGain[16];              //Chromaticity correction values
    DWORD   dwIlluminanceCorrectEnable;         //Illuminance correction enable/disable
                                                0: Disable          1: Enable

    double  dIlluminanceGain;                   //Illuminance correction value
    DWORD   dwLuminousFluxCorrectEnable;        //Luminous flux correction enable/disable
                                                0: Disable          1: Enable
    double  dLuminousFluxGain;                  //Luminous flux correction value
    DWORD   dwLuminousIntensityCorrectEnable;   //Luminous intensity correction enable/disable
                                                0: Disable          1: Enable
    double  dLuminousIntensityGain;             //Luminous intensity correction value
} TM_USER_CORRECT_DATA, *PTM_USER_CORRECT_DATA;
```

## Measurement results structure

```
typedef struct tagTmMeasResult
{
    DWORD   dwIlluminanceEnable;        //Illuminance value valid/invalid
                                        0: Invalid  1: Valid
    double  dIlluminance;               //Illuminance value (lx)
    DWORD   dwLuminousFluxEnable;       //Luminous flux valid/invalid
                                        0: Invalid  1: Valid
    double  dLuminousFlux;              //Luminous flux value (lm)
    DWORD   dwLuminousIntensityEnable;  //Luminous intensity valid/invalid
                                        0: Invalid  1: Valid
    double  dLuminousIntensity;         //Luminous intensity value (cd)
    DWORD   dwChromaticityEnable;       //Tristimulus /chromaticity value valid/invalid
                                        0: Invalid  1: Valid
    double  dX;                         //Tristimulus value X
    double  dY;                         //Tristimulus value Y
    double  dZ;                         //Tristimulus value Z
    double  dChromticity_x;             //Chromaticity value x
    double  dChromticity_y;             //Chromaticity value y
    double  dChromticity_u;             //Chromaticity value u
    double  dChromticity_v;             //Chromaticity value v
    DWORD   dwColorTempEnable;          //Correlated color temperature
                                        Δuv valid/invalid
                                        0: Invalid  1: Valid
    double  dTcp;                       //Correlated color temperature (K)
    double  dDeltaUV;                   //Δuv
    DWORD   dwColorRenderingEnable;     //Color rendering index valid/invalid
                                        0: Invalid  1: Valid
    double  dRi[TEST_COLOR_NUM];        //Special color rendering index R1 to R15
    double  dRa;                        //General color rendering index Ra
    DWORD   dwDominantEnable;           //Dominant wavelength/excitation purity
                                        valid/invalid
                                        0: Invalid  1: Valid
    double  dDominant;                  //Dominant wavelength (nm)
    double  dPurity;                    //Excitation purity
} TM_MEAS_RESULT, *PTM_MEAS_RESULT;
```

**Note**

The structures used by the library are 8-byte aligned. Structure member alignment should be adjusted as needed to accommodate your development environment.

# User's License Agreement

## Important

Please read the following agreement carefully. This user's license agreement (hereafter referred to as Agreement) is a legal contract between the software user (individual or institution) and HIOKI E. E. CORPORATION (hereafter referred to as HIOKI). The term "software" includes any related electronic documentation and computer software and media, as well as any printed matter (such as the Instruction Manual).
By installing, reproducing, or using the software, you, the Licensee, agree to accept the license terms set forth in this Agreement.

---

This software is protected by copyright laws, international copyright agreements, as well as non-corporate laws. The software is a licensed product, and is not sold to the user.

### 1. License

This Agreement grants you, the Licensee, a license to install a single copy of the software on a specified computer system.

### 2. Explanation of other rights and restrictions

-1. Restrictions on reverse engineering, decompiling, and disassembling:

You may not reverse engineer, decompile, or disassemble the software.

-2. Separation of components:

This software is licensed for use as a single product. You may not separate the components for use on multiple computer systems.

-3. Loaning:

You may not loan or lease the software.

-4. Transfer of software:

You may transfer full rights in accordance with this Agreement. However, if you do so, you may not retain any copy of the software, but must transfer the software in its entirety (all components, media, related documentation such as the Instruction Manual, and this Agreement), and must ensure that the receiver of the software agrees with the terms set forth in this Agreement.

-5. Cancellation:

In the event that the terms and conditions set forth in this Agreement are violated, HIOKI retains the right to cancel this Agreement without compromise of any of its other rights. In this event, you must destroy all copies of the software and its components.

### 3. Copyright

The title and copyright rights concerning the software's related documentation, such as the Instruction Manual and copies of the software, are the property of HIOKI and other licensors, and are protected by copyright laws and international agreement regulations. Accordingly, you must treat the software as you would any other copyrighted document. However, you are permitted to make copies as indicated in (A) and (B) below provided such copies are not intended for use other than back-up purposes.

(A) You may make a single copy of the software.

(B) You may install this software on a single computer.

However, you may not reproduce the documentation supplied with the software, such as the Instruction Manual.

## 4. Dual media software

You may receive the same software on more than one type of media. However, regardless of the type and size of media provided, you may only use one media type and only on a single computer. You must not use or install the other media on any other computer. Furthermore, except when transferring the software as stipulated above, you may not loan, lease, or transfer the other media to any other user.

## 5. Warranty

-1. HIOKI reserves the right to make changes to the software specifications without any prior warning.

-2. If the software does not operate in accordance with the supplied Instruction Manual, or the software media or Instruction Manual are damaged in any way, you have one year from the date of purchase to apply for either an exchange or repair at HIOKI's discretion.

-3. In no event will HIOKI be liable for any damages resulting from fire, earthquake, or actions of a third party under the conditions stated in item number 2 above, or for any damage caused as a result of your using the software incorrectly or under unusual circumstances. Further, the warranty is invalid if the following occurs:

(A) Damage incurred through transport, moving, droppage, or any other kind of impact after you purchased the software.

(B) Damage incurred through any form of alteration, unwarranted servicing, or any other type of mistreatment.

-4. In the event that the software is exchanged or repaired, the period of warranty expires on the latest occurring date out of the day stated in the original warranty, and exactly 6 months from the day the exchanged/repaired software is returned to you.

-5. Regardless of the grounds for making a legal claim, HIOKI and its licensors will not be liable for any damage incurred (including, but not limited to: lost profits, suspension of business, loss of data or lost savings) unstated in the warranty terms for the use of this software. This is true even if HIOKI is notified of the possibility of such damages. In any event, HIOKI's liability shall be limited only to replacing defective software with software that is not defective.

HIOKI

www.hioki.com/

All regional contact information

**HIOKI E.E. CORPORATION**
81 Koizumi, Ueda, Nagano 386-1192 Japan

2309 EN

Edited and published by HIOKI E.E. CORPORATION Printed in Japan



**Europe only**

・EU declaration of conformity can be downloaded from our website.
・Contact in Europe: HIOKI EURPOPE GmbH
Helfmann-Park 2, 65760 Eschborn, Germany hioki@hioki.eu