HIOKI

取扱説明書

TM6101 計測ライブラリ

JA

May 2021 Revised edition 3
TM6101A984-03 21-05H

目次

第1章 概要

計測ライブラリソフトウェアは LED 光測定器 TM6101 専用の Windows 用ソフトウェアです。計測ライブラリを利用することで、Windows を搭載する PC で TM6101 を制御するソフトウェアを開発することができます。

動作環境:

対応 OS: Windows 7 (32bit/64bit) 、Windows 8 (32bit/64bit) 、Windows 10 (32bit/64bit)

対応開発環境: Visual Studio 2017、2019 (Visual C++、 Visual Basic 、Visual C#)

注記

対象 OS が問題なく動作する PC を使用してください。また、お客様の使用環境などにより、十分な速度で動作しない場合があります。

計測ライブラリには C 言語のヘッダファイルが含まれます。Visual Basic 等、C 言語以外の開発環境で計測ライブラリを利用する場合は、関数宣言を適宜作成してください。

第2章 使用方法

2.1 インストール

TM6101 本体の取扱書「第 2 章 測定前の準備」にしたがって、ソフトウェアのインストールを実施してください。計測ライブラリを利用するには、ドライバソフトウェアおよび PC アプリケーションソフトウェアのインストールが必須ですので、必ずソフトウェアをインストールしてください。

2.2 使用方法

取扱説明書にしたがってインストールしたフォルダ下に「Library」フォルダができます。「Library」フォルダ内のファイルを、お客様の開発環境の任意の場所へコピーして使用してください。

HiLedMeas.dll	DLL ソフトウェア
HiLedMeas.lib	ライブラリファイル
Tm6101Api.h	ライブラリヘッダファイル

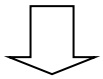
注記

上記ファイル、ドライバソフトウェア、および PC アプリケーションソフトウェアにつきましては、TM6101 を制御する目的でのみ再配布可能です。お客様により開発したソフトウェアを配布する場合に、必要に応じて上記ファイルおよび付属されているインストーラを添付してください。

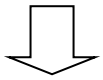
第3章 制御方法

3.1 全体の流れ

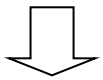
本器のオープン　　オープン関数を使用してデバイス番号を取得します。



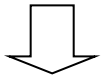
測定条件の設定　　測定条件設定用関数で測定条件を設定します。
各測定条件ごとの関数を使用する方法、および測定条件を一括で設定する関数を使用する方法があります。



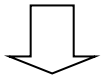
ダーク測定の実行　　ダーク補正を行わない場合、正常な測定値になりませんので、測定を実行する前に必ずダーク測定を実施してください。



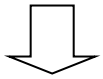
基準値補正の実行　　お客様で用意される基準光源のスペクトルデータや測光値を元に測定器の感度を補正します。必要に応じて基準値補正を実行してください。



測定の実行　　測定対象の光源を測定して、色演算を実行します。



測定結果の取得　　測定結果を取得します。



本器のクローズ　　本器をクローズします。

3.2 本器のオープン

本器を制御するために、オープン用関数を使用して本器をオープンしてください。
オープン用関数が成功すると、戻り値として 1 以上のデバイス番号を取得します。本器のオープン以降はデバイス番号を使用して機器を制御します。
オープン時に本器の測定条件はすべて初期化されます。また、クローズ後に再び同じ機器をオープンした場合、同じデバイス番号を返すとは限りません。また測定条件はすべて初期化されます。

注記

PC の電源を投入後に、本器へ AC アダプタと USB ケーブルをつないだ直後は、本体のパワーインジケータは赤色に点灯します。本器をオープンするとパワーインジケータは緑色に点灯します。
また、本器をクローズすると赤色の点灯になります。
オープンした際に取得するデバイス番号は、同一プロセス内でのみ使用可能です。複数のプロセスから、同時に同じ TM6101 をオープンすることはできません。

オープン用関数:

long TmOpenDevice();	TM6101 をオープンしてデバイス番号を取得します。PC へ複数台の機器が接続されている場合は、特定の機器を指定することができません。
long TmOpenDeviceBySerial(char* pSerial);	製造番号（9 桁の文字列）を指定して、TM6101 をオープンしてデバイス番号を取得します。

3.3 本器のクローズ

本器のすべての制御を終了する場合に、クローズ関数を使用して本器をクローズしてください。クローズ以降は、オープン関数で取得したデバイス番号を使用することはできません。
本器の電源を落とす場合は、本器をクローズ後に実施してください。電源の落とし方については、本体の取扱説明書「1．4　測定の流れ」を参照してください。

注記

本器をクローズすると本体のパワーインジケータは赤色に点灯します。

クローズ用関数:

long TmCloseDevice(long lDeviceId);	指定したデバイス番号の Tm6101 をクローズします。

3.4 測定条件の設定

本器で測定を実行する前に、積分時間、感度、平均化回数などの測定条件を設定します。また、測定条件の構造体を指定して、測定条件を一括で設定することも可能です。
測定条件の取得用関数を使用して、現在設定されている測定条件の取得が可能です。
測定条件の初期化用関数を使用して、オープン時の測定条件へ戻すことができます。

注記
測定条件設定用の関数は、測定待機中以外は使用できません。測定中に関数を使用した場合はエラーを返します。測定待機中かどうかは、後述の TmGetStandbyStatus 関数で知ることができます。

設定する必要がある測定条件は以下の通りです。

通常測定モードの設定

本器では「通常測定モード」、または「AC 点灯測定モード」の 2 通りの測定モードで測定します。オープン時は「通常測定モード」に設定されています。
通常測定モードで測定する場合は、以下の項目を設定してください。

測定モード	「通常測定モード」に設定してください
積分時間	0.1 / 0.5 / 1 / 2 / 4 / 8 / 10 / 16.6 / 20 / 33.3 / 40ms
感度レンジ	High / Low
平均化回数	1～100 回
オートレンジ	OFF / 積分時間オートレンジ / 感度オートレンジ
オートレンジレベル	1～99%

注記
オートレンジを有効にしている場合、下記条件で測定する時はオートレンジ機能は無効になります。
- AC 点灯モード
- 外部トリガ
- ダーク測定

AC 点灯測定モードの設定

AC 点灯測定モードで測定する場合は、以下の項目を設定してください。

測定モード	「AC 点灯測定モード」に設定してください
AC 点灯測定の設定	測定レンジ（レンジ 1～3）
	電源周波数（60 / 50Hz）
	平均化回数（1～100 回）

※AC 点灯測定の設定（測定レンジ、電源周波数、平均化回数）は TmSetAcMode 関数で一括設定します。

外部入出力の設定

トリガタイプ	OFF / 外部トリガ ON（立ち上がり） / 外部トリガ ON（立ち下がり）
トリガディレイ	0 ～ 1,000ms
トリガタイムアウト	10,000 ～ 1,000,000ms
INDEX 出力時間	1 ～ 100ms

注記

外部トリガを使用しない場合、トリガディレイ、トリガタイムアウトは無効になります。

演算関係の設定

基準の光	CIE 昼光 / 黒体放射 / 自動切り替え
	※演色評価数の演算で使用します。
測光距離	0.01 ～ 10.00m
	※光度値の測定で使用します。

3.5 ダーク測定

測定前にダーク測定を実施することにより、ダーク補正された測定結果になります。ダーク補正を行わない場合、正常な測定値になりませんので、測定を実行する前に必ずダーク測定を実施してください。
ダーク測定では平均化回数を設定できます。(平均化回数: 1 ～ 100 回)

注記
ダーク測定を実行する前に、必ず付属の遮光キャップを取りつけてください。

現在の積分時間、感度レンジでダーク測定する方法と、全ての積分時間、感度レンジのダーク測定を実行する方法の、2 通りの方法でダーク測定を実行できます。

現在の積分時間、感度レンジでダーク測定を実行する

ダーク測定終了まで時間がかかりませんが、積分時間、感度レンジ、測定モードを変更したあとは、ダーク測定値がクリアされます。また、オートレンジを使用して測定を実行した場合も、ダーク測定値がクリアされます。

TmExecDarkMeas 関数を使用してダーク測定を実行します。ダーク測定が終了するまで関数から戻りません。

全ての積分時間、感度レンジのダーク測定を実行する

関数を1回呼ぶことで、自動で積分時間、感度レンジを切り替えてダーク測定を実行します。全ての積分時間、感度レンジで測定するため時間がかかります。
積分時間、感度レンジ、測定モードを変更しても再びダーク測定する必要がありません。ダーク測定結果は電源を切るまで有効です。オートレンジを有効にする場合は、すべての積分時間、感度レンジでダーク測定を実行してください。
また、全ダーク補正データの取得関数、全ダーク補正データの設定関数、およびダーク測定を未実施状態にする関数が用意されています。

同期関数、または非同期関数でダーク測定を実行します。

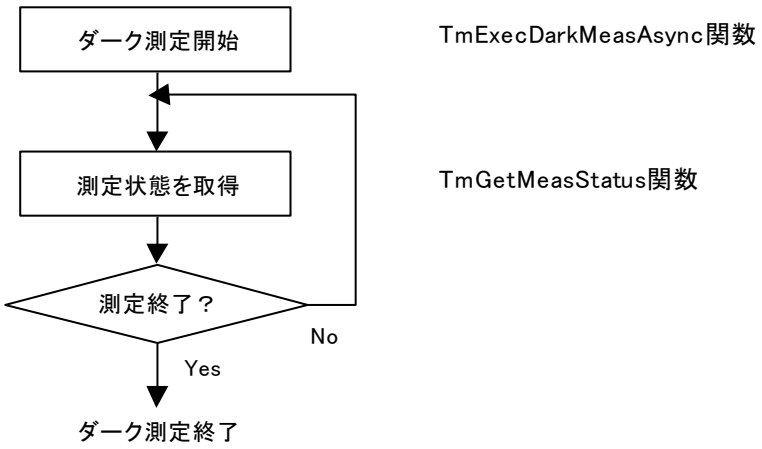
(1)同期関数でダーク測定する

TmExecDarkMeas 関数を使用してダーク測定を実行します。ダーク測定が終了するまで関数から戻りません。

(2)非同期関数でダーク測定する

TmExecDarkMeasAsync 関数を使用してダーク測定を実行します。
関数からすぐに戻りますが、ダーク測定が終了するまで TmGetMeasStatus 関数で測定状態を監視する必要があります。

非同期関数ダーク測定例:



3.6 基準値補正

お客様で用意される基準光源のスペクトルデータや測光値を元に測定器の感度を補正します。必要に応じて基準値補正を実行してください。
基準値補正結果はクローズ関数を使用して本器をクローズするまで有効です。また、基準値補正の実行関数以外に、基準値補正データの取得関数、基準値補正データの設定関数、および基準値補正を未実施状態にする関数が用意されています。

色度補正を実行する

1. 色度補正対象の光源の分光特性を標準機（分光タイプ測定器）で測定し、測定結果を用意してください。380 nm から 780 nm まで 5 nm おきの分光測定結果が必要になります。

2. 色度補正対象の光源を本器で測定します。測定手順につきましては、後述の「3. 7 測定の実行」を参照してください。

3. 色度補正の実行関数を使って、色度補正を実行します。上記2. の測定結果（直前の測定結果）が補正の対象になります。

・TmExecChromaticityCorrect を使用する場合は、関数の引数に分光測定結果を指定します。
・TmExecChromaticityCorrectByFile を使用する場合は、関数の引数に分光測定結果を保存したファイル名を指定します。ファイルフォーマットにつきましては、本体の取扱説明書の「2. 8 補正機能を使う」を参照してください。

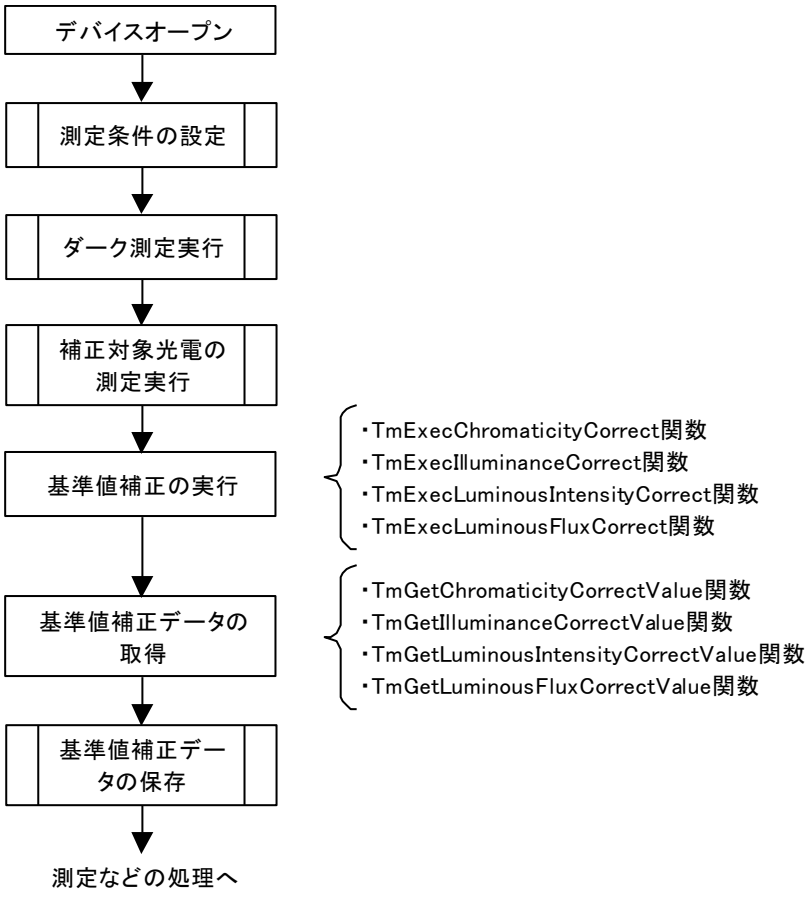
照度補正、光度補正、光束補正を実行する

1. 補正対象の光源を本器で測定します。測定手順につきましては、後述の「3. 7 測定の実行」を参照してください。

2. 各補正の実行関数を使って補正を実行します。関数の引数へ補正対象の基準値を指定します。
上記1. の測定結果（直前の測定結果）が補正の対象になります。

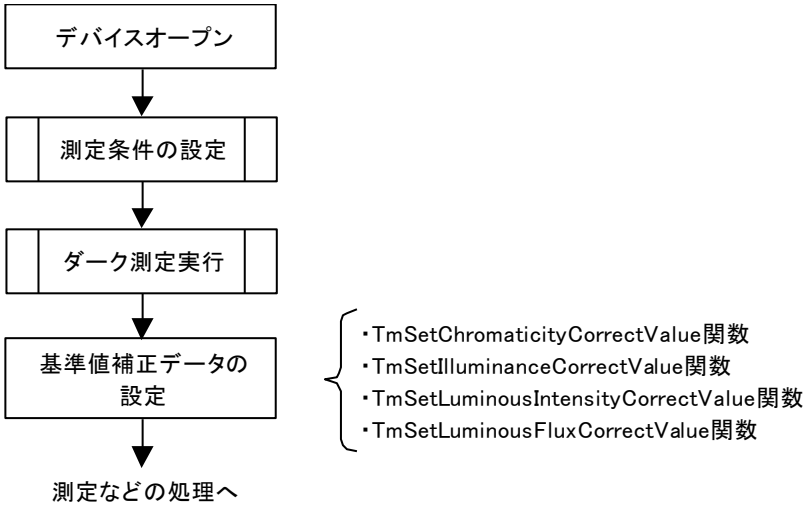
・照度補正を実行する場合は、TmExecIlluminanceCorrect を使用します。
・光度補正を実行する場合は、TmExecLuminousIntensityCorrect を使用します。
・光束補正を実行する場合は、TmExecLuminousFluxCorrect を使用します。

基準値補正の実行例:



※基準値補正データの取得、および基準値補正データの保存は、次回電源投入時に補正データを再設定する必要がある場合などに、必要に応じて実行してください。
また、基準値補正データの保存は、お客様により実装(ファイルへの保存等)していただく必要があります。

基準値補正データの再設定例:



3.7 測定の実行

同期関数、または非同期関数のいずれかの測定用関数を使用して測定を実行します。測定の終了後は、測定結果の取得関数を使用して、測定結果を取得することができます。

注記

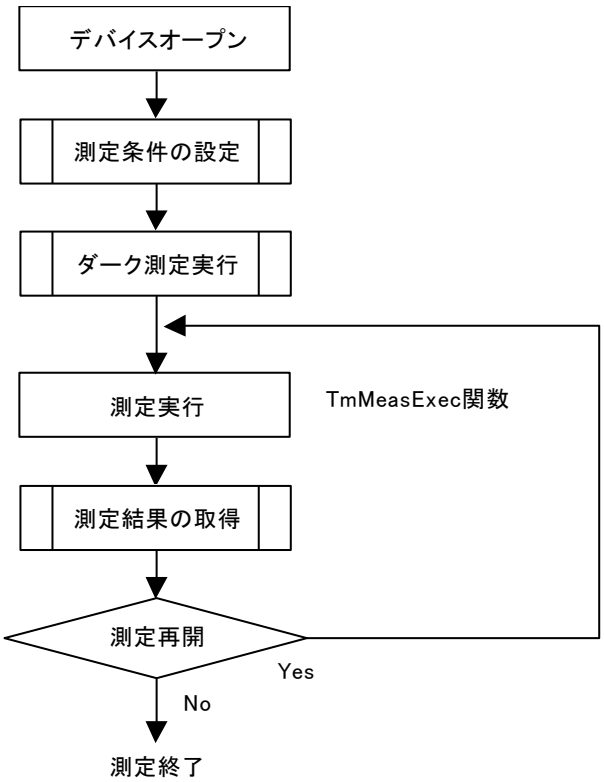
外部トリガを有効にしている場合、測定用関数を実行することで外部トリガ監視を開始します。外部トリガによる測定が終了した後は、外部トリガ監視は解除されます。再び外部トリガによる測定を開始するには、測定用関数を再び実行してください。

同期関数で測定する

TmMeasExec 関数を使用して測定を実行します。測定が終了して測定待機状態になるまで関数から戻りません。外部 I/O の測定終了出力が OFF になると測定待機状態になります。

外部トリガを有効にしている場合、TmMeasExec 関数を実行することで外部トリガ監視を開始します。外部トリガが入力されて測定が終了するか、またはトリガタイムアウト時間が経過するまで関数からは戻りません。

同期関数の使用例:



非同期関数で測定する

非同期関数で測定を開始すると関数からすぐに戻りますので、本器の測定中に他の処理をすることができます。
外部トリガを有効にしている場合、TmMeasExec 関数を実行することで外部トリガ監視を開始します。外部トリガ監視中は TmCancelMeas 関数を呼ぶことで監視状態を解除（測定待機状態）することができます。

TmMeasExecAsync 関数から戻った後は、測定状態を TmGetMeasStatus 関数と TmGetStandbyStatus 関数で監視する必要があります。

非同期関数を使った測定手順は以下のようになります。

1. TmMeasExecAsync 関数で測定を開始します。本器が測定開始すると関数からすぐに戻ります。
   ※オートレンジを有効にしている場合は、オートレンジ処理が終了するまで関数から戻りません。

2. TmGetMeasStatus 関数で測定状態を取得します。測定が終了するまで TmGetMeasStatus 関数を繰り返し呼んでください。引数のタイムアウト時間に 0xFFFFFFFF を指定すると、測定が終了するまで TmGetMeasStatus 関数から戻りません。
   TmGetMeasStatus 関数の内部で測定処理をしていますので、必ず TmGetMeasStatus 関数で測定の終了を確認してください。

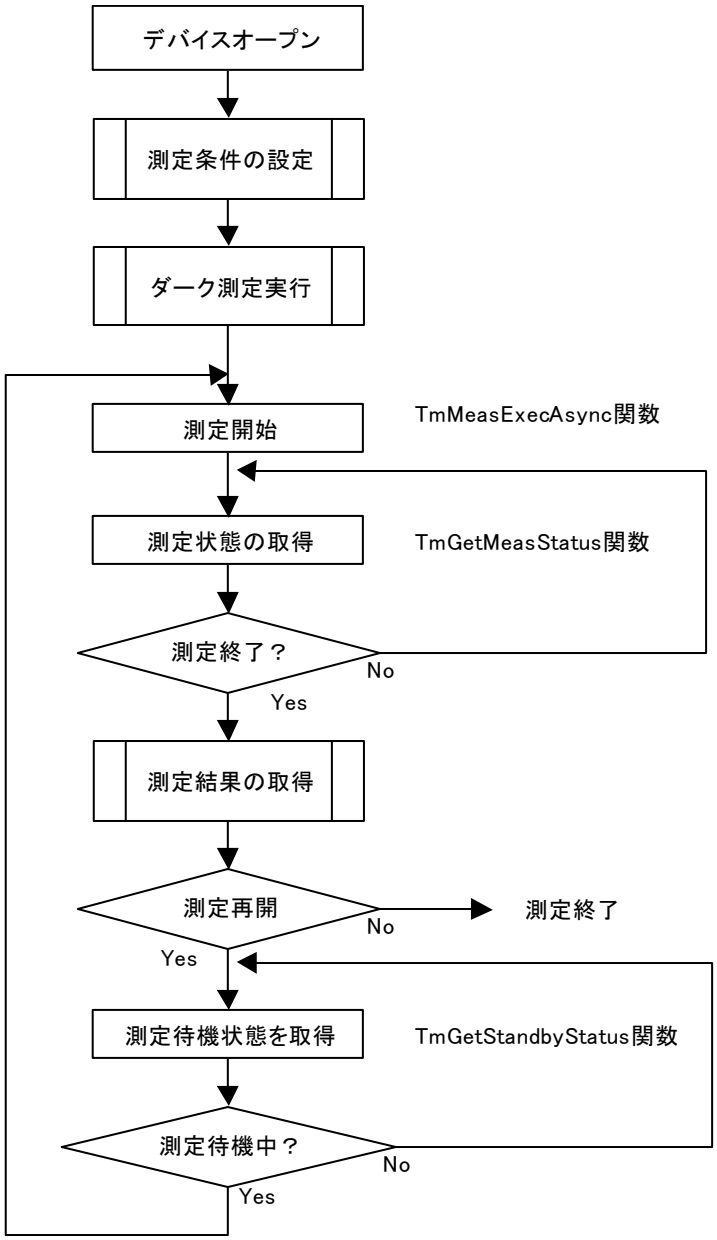
3. 測定の終了後は、測定結果取得用の関数を使用して、測定結果を取得することができます。
   また、センサによる測定は終了してますので、次の測定対象をセットする等の処理が可能です。

4. TmGetStandbyStatus 関数で測定待機状態かどうか取得します。測定待機状態になりましたら、TmMeasExecAsync 関数で測定を開始することができます。引数のタイムアウト時間に 0xFFFFFFFF を指定すると、測定待機状態になるまで TmGetStandbyStatus 関数から戻りません。

注記

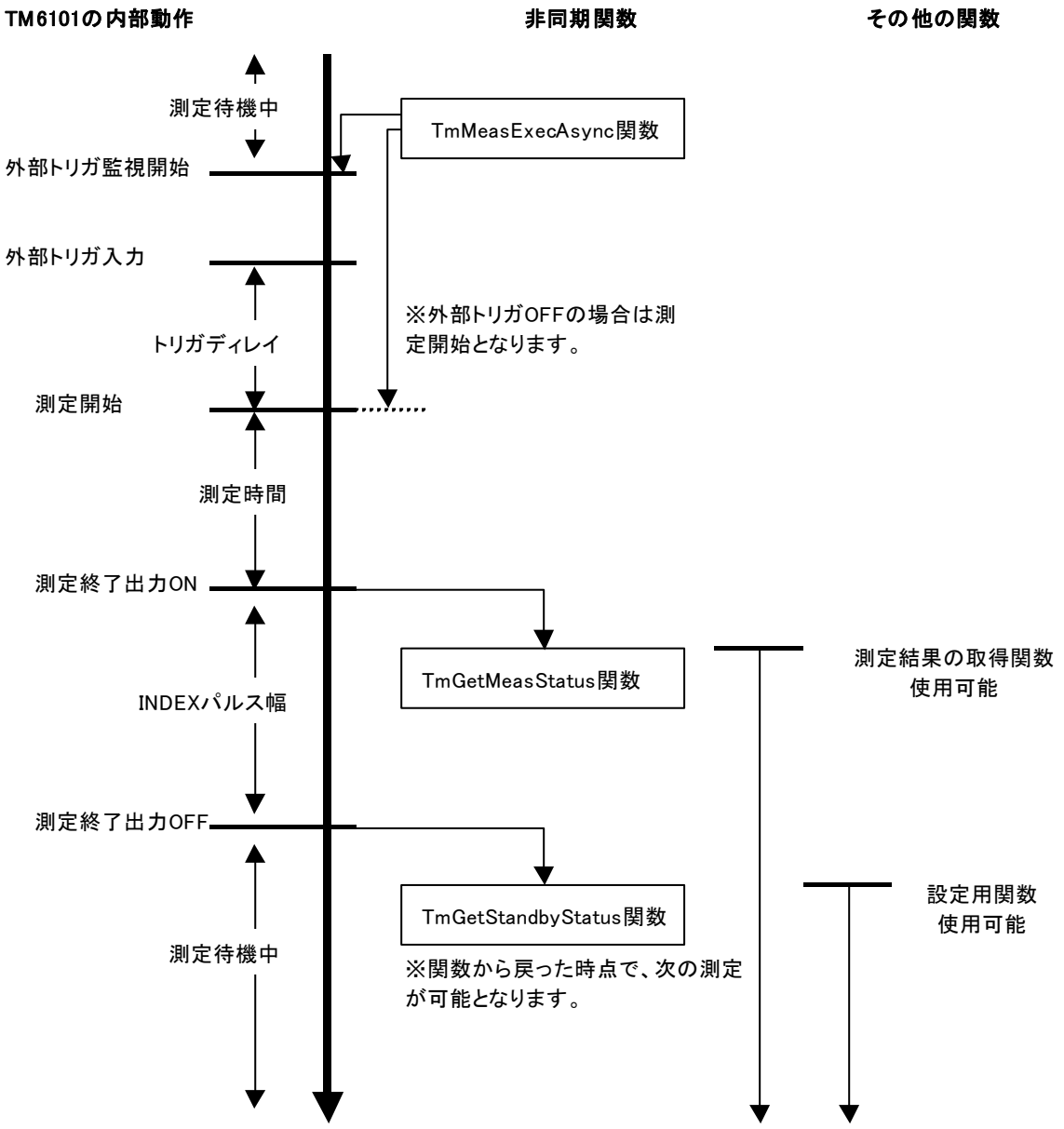
測定開始後は、測定が終了して測定待機状態になるまで、測定条件の設定はできません。測定待機状態になったことを確認するまで、測定条件の設定用関数を使用しないでください。

非同期関数の使用例:



次の図は、本器の動作とライブラリ関数のタイミングについて示しています。本器の動作タイミング詳細につきましては、本体取扱説明書「4．2 タイミングチャート」を参照してください。

TM6101 の動作タイミング:



※通信時間があるため、TmMeasExecAsync 関数を呼んだ直後から外部トリガ監視開始まで約 2ms かかります。また、TmMeasExecAsync 関数から戻った時点では、すでに外部トリガ監視を開始しています。

※上記と同様に、測定終了後に TmGetMeasStatus 関数から戻るまで約 2ms かかります。また、測定終了出力 OFF 後に TmGetStandbyStatus 関数から戻るまで約 2ms かかります。(引数のタイムアウトに 0xFFFFFFFF をして待機している場合)

※通信時間は PC の処理能力や使用環境などにより変化します。

※TmMeasExec 関数(同期関数)で測定を実行した場合は、測定終了出力が OFF になった時点で関数から戻ります

3.8 測定結果の取得

測定終了後は、照度値や色度値などの測定結果を取得します。測定実行関数が正常終了した場合に、測定結果を取得可能です。また、測定結果の構造体を指定して、測定結果を一括で取得することも可能です。

取得可能な測定結果

照度値
光度値
光束値
三刺激値(XYZ)
色度値(xy、uv)
相関色温度、Δuv
特殊演色評価数(R1～R15)
平均演色評価数(Ra)
ドミナント波長

注記

測定結果は、測定が成功するまで確定されません。測定動作中に測定結果の取得用関数を使用した場合はエラーとなります。TmMeasExec 関数(同期関数)が成功した場合、また TmGetMeasStatus 関数(非同期関数)で測定状況を取得して「測定成功終了」が返った場合に、測定結果を取得可能です。

測定が成功(戻り値:0 以外)でも演算不可の場合がありますので、必ず測定結果の取得関数の戻り値を確認してください(照度値がマイナス、Δuv が 0.02 以上など)。その場合は、関数の引数へ戻された測定結果は不定となります。

外部トリガでトリガ監視中に TmCancelMeas 関数で測定を中断した場合、測定結果の取得用関数は前回測定した結果を返します。

第4章 ライブラリ関数リファレンス

4.1 接続用関数

TmOpenDevice

説明 TM6101 をオープンしてデバイス番号を取得します。以後、ライブラリ関数で処理を実行する場合は、取得したデバイス番号を使用します。

宣言 long TmOpenDevice();

引数 なし

戻り値 1 以上： デバイス番号　　0：失敗

注記 PC へ複数台の機器が接続されている場合は、特定の機器を指定することができません。オープン後は、本体のインジケータが赤から緑へ変わります。

TmOpenDeviceBySerial

説明 製造番号（9 桁の文字列）を指定を指定して TM6101 をオープンして、デバイス番号を取得します。以後、ライブラリ関数で処理を実行する場合は、取得したデバイス番号を使用します。

宣言 long TmOpenDeviceBySerial(char* pSerial);

引数

pSerial　　9 桁の製造番号文字列(NULL 終端)

戻り値 1 以上： デバイス番号　　0：失敗

注記 製造番号は本体またはセンサユニットに記されている9桁の文字列です。文字列は char 型（8bit）の配列にアスキー文字列で指定します（NULL 終端）。2 バイト文字は使用しないでください。オープン後は、本体のインジケータが赤から緑へ変わります。

使用例

```
char szSerial[10] = “100730001”;
long lDeviceId = TmOpenDeviceBySerial(szSerial);
if (lDeviceId <= 0) {
        //エラー処理
}

//設定、実行処理等

TmCloseDevice(lDeviceId);
```

TmCloseDevice

説明　TM6101 をクローズします。クローズ後に再びオープンした場合、測定条件はすべて初期化されます。

宣言　long TmCloseDevice(long lDeviceId);

引数

lDeviceId　デバイス番号

戻り値　1：成功　0：失敗

注記　クローズ後は本体のインジケータが緑から赤へ変わります。

4.2 測定条件

TmSetMeasMode

説明　測定モードを変更します。通常は「通常測定モード」で測定してください。

宣言　long TmSetMeasMode(long lDeviceId, char cMeasMode);

引数

lDeviceId	デバイス番号
cMeasMode	測定モード　0:通常測定モード(デフォルト)　1:AC 点灯測定モード

戻り値　1: 成功　0:失敗

注記　AC 点灯測定モードで測定する場合は、本関数で「AC 点灯測定モード」へ変更してから、TmSetAcMode 関数で AC 点灯測定モードの設定をしてください。
オープン時点では通常測定モードですので、通常測定モードのみで測定する場合は呼ぶ必要はありません。

使用例

・通常測定モードで測定する場合

```
long lRet;
lRet = TmSetMeasMode(lDeviceId, 0);                 //通常測定モード
lRet = TmSetIntegralTime(lDeviceId, 2);             //積分時間 1ms
for (char nCh = 0; nCh < 16; nCh++){
    lRet = TmSetSensitivity(lDeviceId, nCh, 0);     //感度 Low
}
lRet = TmSetAverageNum(lDeviceId, 1);               //平均化無し
```

・AC 点灯測定モードで測定する場合

```
long lRet;
lRet = TmSetMeasMode(lDeviceId, 1);                 //AC 点灯測定モード
lRet = TmSetAcMode(lDeviceId,  0,  1, 10);          //レンジ 1、50Hz、平均化 10 回
```

TmGetMeasMode

説明　現在の測定モードを取得します。

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

引数

lDeviceId	デバイス番号
pcMeasMode	現在の測定モードを返します。 0:通常測定モード(デフォルト)　1:AC 点灯測定モード

戻り値　1: 成功　0:失敗

TmSetIntegralTime

説明　積分時間を設定します。通常測定モードで設定可能です。

宣言　long TmSetIntegralTime(long lDeviceId, char cIntTimeIndex);

引数

lDeviceId	デバイス番号
cIntTimeIndex	積分時間 0：0.1ms　1：0.5ms　2：1ms(デフォルト)　3：2ms　4：4ms 5：8ms　6：10ms　7：16.6ms　8：20ms　9：33.3ms　10：40ms

戻り値　1：成功　　0:失敗

注記　現在の測定モードが「AC 点灯測定モード」の場合は、「通常測定モード」へ変更してから本関数を呼んでください。
使用方法につきましては、TmSetMeasMode 関数の使用例を参照してください。

TmGetIntegralTime

説明　現在の積分時間を取得します。

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

引数

lDeviceId	デバイス番号
pcIntTimeIndex	現在の積分時間を返します。 0：0.1ms　1：0.5ms　2：1ms(デフォルト)　3：2ms　4：4ms 5：8ms　6：10ms　7：16.6ms　8：20ms　9：33.3ms　10：40ms

戻り値　1：成功　　0:失敗

注記　積分時間オートレンジで測定すると最適な積分時間に自動設定されますが、この関数を使用することで現在の積分時間設定を取得できます。使用例につきましては、TmSetAutoRange 関数を参照してください。

TmSetSensitivity

説明　感度レンジを設定します。通常測定モードで設定可能です。

宣言　long TmSetSensitivity(long lDeviceId, char nCh, char cSens);

引数

lDeviceId	デバイス番号	
nCh	センサ番号	0 ～15 を指定（センサ 1 ～16）
cSens	感度レンジ	0：感度 High　1：感度 Low

戻り値　1：成功　0:失敗

注記　現在の測定モードが「AC 点灯測定モード」の場合は、「通常測定モード」へ変更してから本関数を呼んでください。

使用方法につきましては、TmSetMeasMode 関数の使用例を参照してください。

本器では光学特性が異なった 16 個の光センサにより測定をしています。各 2 つのセンサのペアごとに感度レンジ設定を持っていますので、対応するセンサの感度レンジは同じ設定になります。例えば、センサ 1 の感度レンジを Low に設定した場合、センサ 2 のセンサ感度も Low になります。（センサ 2 の感度レンジを High に設定した場合、センサ 1 のセンサ感度も High になります。）

センサ 1 センサ 2	センサ 3 センサ 4	センサ 5 センサ 6	センサ 7 センサ 8	センサ 9 センサ 10	センサ 11 センサ 12	センサ 13 センサ 14	センサ 15 センサ 16
High/Low	High/Low	High/Low	High/Low	High/Low	High/Low	High/Low	High/Low

TmGetSensitivity

説明　現在の感度レンジを取得します。

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

引数

lDeviceId	デバイス番号	
nCh	センサ番号	0 ～15 を指定（センサ 1 ～16）
pcSens	現在の感度レンジを返します。	0：感度 High　1：感度 Low

戻り値　1：成功　0:失敗

注記　感度オートレンジで測定すると最適な感度レンジに自動設定されますが、この関数を使用することで現在の感度レンジ設定を取得できます。使用例につきましては、TmSetAutoRange 関数を参照してください。

TmSetAverageNum

説明　平均化回数を設定します。通常測定モードで設定可能です。

宣言　long TmSetAverageNum(long lDeviceId, long lAveNum);

引数

lDeviceId	デバイス番号
lAveNum	平均化回数　1:平均化無し(デフォルト)　2 ～ 100: 平均化回数[回]

戻り値　1: 成功　0:失敗

TmGetAverageNum

説明　現在の平均化回数を取得します。

宣言　long TmGetAverageNum(long lDeviceId);

引数

lDeviceId	デバイス番号

戻り値　平均化回数:　1 ～ 100　(0: 失敗)

注記　AC 点灯測定モードの平均化回数を取得する場合は TmGetAcMode 関数を使用します。

TmSetTrigType

説明　外部トリガを設定します。

宣言　long TmSetTrigType(long lDeviceId, char cTrigType);

引数

lDeviceId	デバイス番号
cTrigType	0:外部トリガ OFF(デフォルト)　1:外部トリガ ON(立ち上がり) 2:外部トリガ ON(立ち下がり)

戻り値　1: 成功　0:失敗

注記　外部トリガ ON に設定した場合は、TmExecMeas 関数を実行することで外部トリガ監視を開始します。

TmGetTrigType

説明　現在の外部トリガ設定を取得します。

宣言　long TmGetTrigType(long lDeviceId, char* pcTrigType);

引数

lDeviceId	デバイス番号
pcTrigType	現在の外部トリガ設定を返します。 0:外部トリガ OFF(デフォルト)　1:外部トリガ ON(立ち上がり) 2:外部トリガ ON(立ち下がり)

戻り値　1: 成功　0:失敗

TmSetTrigDelay

説明　トリガディレイを設定します。

宣言　long TmSetTrigDelay(long lDeviceId, long lDelay);

引数

lDeviceId	デバイス番号
lDelay	トリガディレイ[ms]: 0 ～ 1000（デフォルト: 0ms）

戻り値　1: 成功　0:失敗

注記　外部トリガ ON 時にトリガディレイは有効になります。外部トリガ OFF の時はトリガディレイは働きません。

TmGetTrigDelay

説明　現在のトリガディレイ設定を取得します。

宣言　long TmGetTrigDelay(long lDeviceId, long* plDelay);

引数

lDeviceId	デバイス番号
plDelay	現在のトリガディレイ設定値[ms]を返します。

戻り値　1: 成功　0:失敗

TmSetTrigTimeout

説明　外部トリガタイムアウト時間を設定します。

宣言　long TmSetTrigTimeout(long lDeviceId, long lTimeout);

引数

lDeviceId	デバイス番号
lTimeout	トリガタイムアウト時間[ms]: 10,000 ～ 1,000,000（デフォルト: 100,000ms）

戻り値　1: 成功　0:失敗

注記　測定実行用関数で外部トリガ監視を開始後、トリガタイムアウト時間を経過しても外部トリガが入力されない場合、測定を強制終了します。

TmGetTrigTimeout

説明　現在の外部トリガタイムアウト時間を取得します。

宣言　long TmGetTrigTimeout(long lDeviceId, long* plTimeout);

引数

lDeviceId	デバイス番号
plTimeout	トリガタイムアウト時間[ms]を戻します。

戻り値　1: 成功　0:失敗

TmSetAutoRange

説明　　オートレンジを設定します。

宣言　　long TmSetAutoRange(long lDeviceId, char cAutoRangeType);

引数

lDeviceId　　デバイス番号

cAutoRangeType　　0：OFF（デフォルト）　1：積分時間オートレンジ　　2：感度オートレンジ

戻り値　　1：成功　　0：失敗

注記　　AC 測定モードではオートレンジは使用できません。外部トリガ ON の時はオートレンジは使用できません。

オートレンジで測定すると最適な積分時間、または感度レンジに自動設定されますが、TmGetIntegralTime 関数、TmGetSensitivity 関数を使用することで現在の設定を取得できます。

使用例

```
long lRet;
char cIntegralTime;        //積分時間
char cSens[16]; //感度レンジ

lRet = TmSetAutoRange(lDeviceId,1);                    //積分時間オートレンジ
lRet = TmMeasExec();                                   //積分時間オートレンジで測定実行

lRet = TmGetIntegralTime(lDeviceId, &cIntegralTime);        //積分時間取得

lRet = TmSetAutoRange(lDeviceId,2);                    //感度オートレンジ
lRet = TmMeasExec();                                   //感度オートレンジで測定実行

for (char nCh = 0; nCh < 16; nCh++) {
        lRet = TmGetSensitivity(lDeviceId, nCh, &cSens[nCh]);      //感度レンジ取得
}
```

TmGetAutoRange

説明　オートレンジ設定を取得します。

宣言　long TmGetAutoRange(long lDeviceId, char* pcAutoRangeType);

引数

lDeviceId	デバイス番号
pcAutoRangeType	オートレンジ設定を返します。 0：OFF　1：積分時間オートレンジ　2：感度オートレンジ

戻り値　1：成功　0:失敗

TmSetAutoRangeLevel

説明　オートレンジの検出レベル上下限値を設定します。

宣言　long TmSetAutoRangeLevel(long lDeviceId, char cLevelHigh, char cLevelLow);

引数

lDeviceId	デバイス番号
cLevelHigh	オートレンジの上限値[%]：1～99（デフォルト：90%）
cLevelLow	オートレンジの下限値[%]：1～99 （デフォルト：10%）

戻り値　1：成功　0:失敗

注記　積分時間オートレンジの時は、全センサの検出レベルが、上限値を超えない範囲で下限値以上になるように積分時間を自動調整します。

感度レンジオートレンジの時は、各センサの検出レベルが、上限値を超えない範囲で下限値以上になるように感度（High/Low）を自動調整します。

オートレンジを正常に機能させるためには、上限値はを下限値の 2 倍以上に設定してください。（下限値 30%の場合は、上限値を 60%以上にする）

cLevelHigh, cLevelLow のどちらかへ 0 を指定した場合は、デフォルト（上限値 90%、下限値 10%）に戻ります。

TmGetAutoRangeLevel

説明　オートレンジの検出レベル上下限値を取得します。

宣言　long TmGetAutoRangeLevel(long lDeviceId, char* pcLevelHigh, char* pcLevelLow);

引数

lDeviceId	デバイス番号
pcLevelHigh	オートレンジの上限値[%]を返します。
pcLevelLow	オートレンジの下限値[%]を返します。

戻り値　1：成功　0:失敗

TmSetAcMode

説明　AC 点灯測定モードの設定をします。

宣言　long TmSetAcMode(long lDeviceId, char cAcRange, char cAcPlc, long lAveNum);

引数

lDeviceId	デバイス番号
cAcRange	AC 点灯測定モードレンジ 0：レンジ 1（デフォルト）　1：レンジ 2　2：レンジ 3
cAcPlc	電源周波数　0：60Hz　1：50Hz（デフォルト）
lAveNum	AC 点灯測定モードの平均化回数[回]：1 ～ 100（デフォルト：1 回）

戻り値　1：成功　0:失敗

注記　TmSetMeasMode 関数で AC 点灯測定モードへ変更してから、本関数で設定してください。

使用方法につきましては、TmSetMeasMode 関数の使用例を参照してください。

TmGetAcMode

説明　AC 点灯測定モード設定を取得します。

宣言　long TmGetAcMode(long lDeviceId, char* pcAcRange, char* pcAcPlc,
long* plAveNum);

引数

lDeviceId	デバイス番号
pcAcRange	AC 点灯測定モードレンジを返します。 0:レンジ 1　1:レンジ 2　2:レンジ 3
pcAcPlc	電源周波数を返します。　0：60Hz　1：50Hz
plAveNum	AC 点灯測定モードの平均化回数を返します。　1 ～ 100 回

戻り値　1：成功　0:失敗

TmSetRefIlluminant

説明　演色評価数演算で使用する基準の光を設定します。

宣言　long TmSetRefIlluminant(long lDeviceId, char cType);

引数

lDeviceId	デバイス番号
cType	0：CIE 昼光　1：黒体放射 2：自動切り替え（黒体放射 ＜5000K≦CIE 昼光）（デフォルト）

戻り値　1：成功　0:失敗

TmGetRefIlluminant

説明　演色評価数演算で使用する基準の光を取得します。

宣言　long TmGetRefIlluminant(long lDeviceId, char* pcSet);

引数

lDeviceId	デバイス番号
pcSet	基準の光を返します。 0: CIE 昼光　1: 黒体放射　2: 自動切り替え

戻り値　1: 成功　0:失敗

注記

TmSetLightDistance

説明　光度計算用で使用する測光距離を設定します。

宣言　long TmSetLightDistance(long lDeviceId, double dDistance);

引数

lDeviceId	デバイス番号
dDistance	測光距離[m]: 0.01～10.00 （デフォルト: 0.01[m]）

戻り値　1: 成功　0:失敗

TmGetLightDistance

説明　光度計算用で使用する測光距離を取得します。

宣言　long TmGetLightDistance(long lDeviceId, double* pdDistance);

引数

lDeviceId	デバイス番号
pdDistance	測光距離[m]を返します。

戻り値　1: 成功　0:失敗

TmSetExtIoIndexOutpTime

説明　外部 IO の測定終了出力（INDEX 出力）の ON 時間を設定します。

宣言　long TmSetExtIoIndexOutpTime(long lDeviceId, DWORD dwTimeMsec);

引数

lDeviceId	デバイス番号
dwTimeMsec	ON 時間[ms]: 1～100（デフォルト: 1ms）

戻り値　1: 成功　0:失敗

注記　ダーク測定では測定終了出力は無効です。

TmGetExtIoIndexOutpTime

説明　外部 IO の測定終了出力（INDEX 出力）の ON 時間を取得します。

宣言　long TmGetExtIoIndexOutpTime(long lDeviceId, DWORD* pdwTimeMsec)

引数

lDeviceId	デバイス番号
pdwTimeMsec	ON 時間[ms]を返します。

戻り値　1：成功　0：失敗

TmSetMeasSettingAll

説明　測定条件を一括設定します。

宣言　long TmSetMeasSettingAll(long lDeviceId, TM_MEAS_SET stMeasSet);

引数

lDeviceId	デバイス番号
stMeasSet	測定条件構造体を指定します。全てのメンバ変数へ設定値を指定します。

戻り値　1：成功　0：失敗

注記　TM_MEAS_SET 構造体に全測定条件を指定してください。前回の TM6101 との接続時に TmGetMeasSettingAll で取得した設定値を、今回接続時に再設定する場合などに利用します。

使用例

```
TM_MEAS_SET   stMeasSet;                  //測定条件構造体
stMeasSet.dwMeasMode = 0;                 //通常測定モード
stMeasSet. dwIntTime = 2;                 //積分時間 1ms
  //以降、全てのメンバ変数を設定する
        ・
        ・
TmSetMeasSettingAll(lDeviceId,   stMeasSet);        //測定条件を一括設定
```

TmGetMeasSettingAll

説明　現在の測定条件を一括取得します。

宣言　long TmGetMeasSettingAll(long lDeviceId, TM_MEAS_SET* pstMeasSet);

引数

lDeviceId	デバイス番号
pstMeasSet	測定条件構造体を返します。測定条件構造体をへのポインタを指定します。

戻り値　1：成功　0：失敗

使用例

```
TM_MEAS_SET   stMeasSet;                        //測定条件構造体
TmSetMeasSettingAll(lDeviceId,   &stMeasSet);   //測定条件を一括取得
```

TmInitializeMeasSettings

説明　測定条件を初期化します。

宣言　long TmInitializeMeasSettings(long lDeviceId);

引数

lDeviceId　デバイス番号

戻り値　1：成功　0：失敗

注記　デフォルト値は各関数の説明を参照してください。測定実行中は初期化できません。

4.3 測定実行

TmMeasExec

説明　測定を実行します。外部トリガ ON の場合は、外部トリガ監視を開始します。測定が終了するまで、またはトリガタイムアウト時間が経過するまで関数から戻りません。

宣言　long TmMeasExec(long lDeviceId);

引数

lDeviceId　デバイス番号

戻り値　1：成功　0：失敗

注記　測定終了後は測定結果の取得が可能です。

使用例

・10 回測定して終了する

```
long lDeviceId = TmOpenDevice ();              //TM6101 デバイスオープン
if (lDeviceId <= 0) {
        //エラー処理
}

long lRet;
double x, y;

lRet = TmSetAutoRange(lDeviceId,1);                    //積分時間オートレンジ

        //その他の測定条件はデフォルトで測定する

lRet = TmExecDarkMeas(lDeviceId, 10, 1);    //全レンジダーク測定実行（平均化 10 回）

for (long nNum = 0; nNum < 10; nNum++) {
    lRet = TmMeasExec(lDeviceId);                      //積分時間オートレンジで測定実行
    lRet = TmGetChromaticityValue_xy(lDeviceId, &x,&y);     //色度値 xy 取得
}
    //測定結果の表示等の処理
        ・
        ・

TmCloseDevice(lDeviceId);              //クローズ
```

TmMeasExecAsync

説明　測定を開始します。外部トリガ ON の場合は、外部トリガ監視を開始します。
本関数からはすぐに戻ります。

宣言　long TmMeasExecAsync(long lDeviceId);

引数

lDeviceId　デバイス番号

戻り値　1：成功　0：失敗

注記　測定結果を取得する前に必ず TmGetMeasStatus 関数で測定の終了を確認してください。測定終了後は測定結果の取得が可能です。
外部トリガ監視を終了する場合は、TmCancelMeas を呼んでください。外部トリガ監視中に TmCancelMeas 関数で測定を中断した場合、測定結果の取得用関数は前回測定した結果を返します。
また、次の測定を開始する前に TmGetStandbyStatus 関数で測定待機状態になっていることを確認して、TmMeasExecAsync で測定を開始してください。

使用例

・非同期関数で 10 回測定する

```
long lRet;
double x, y;                    //色度値
DWORD dwStatus;                 //計測状況

    //測定条件はデフォルトで測定する

for (long nNum = 0; nNum < 10; nNum++) {        //10 回測定を実行する
    lRet = TmMeasExecAsync (lDeviceId);                 // 非同期で測定開始

    do {
        lRet = TmGetMeasStatus(lDeviceId,  &dwStatus, 0);  //測定状況を取得
    } while (dwStatus == 0);            //測定中の場合は繰り返し測定状況を取得する

    //測定結果の取得
    lRet = TmGetChromaticityValue_xy(lDeviceId, &x,&y);

    do {
        lRet = TmGetStandbyStatus(lDeviceId,  &dwStatus, 0);        //待機状態を取得
    } while (dwStatus == 0);            //待機中になるまで繰り返し状態を取得する
}
```

TmGetMeasStatus

説明　現在の測定状況を取得します。

宣言　long TmGetMeasStatus(long lDeviceId, DWORD* pdwStatus,
DWORD dwMilliseconds);

引数

lDeviceId	デバイス番号
pdwStatus	測定状態を返します。 0: 測定中、またはトリガ監視中　1: 測定正常終了 2: 測定異常終了
dwMilliseconds	タイムアウト時間[ms]: 0 ～ 0xFFFFFFFF タイムアウト時間が経過するか、または測定が終了すると関数から戻ります。0 を指定すると即座に関数から戻ります。 0xFFFFFFFF を指定すると測定が終了するまで関数から戻りません。

戻り値　1: 成功　0:失敗

注記　TmMeasExecAsync、TmExecDarkMeasAsync で測定を開始した場合、**戻り値** 1 または 2 が返るまで本関数を実行してください。
測定を強制終了、またはトリガ監視を終了する場合は、TmCancelMeas を呼んでください。TmCancelMeas 関数で測定を強制終了した場合は、測定状況は 2(測定異常終了)が戻ります。また、外部トリガ監視を終了した場合は、測定状況は1(測定正常終了)が戻ります。
使用方法につきましては、TmMeasExecAsync 関数を参照してください。

TmGetStandbyStatus

説明　測定待機状態を取得します。

宣言　long TmGetStandbyStatus(long lDeviceId, DWORD* pdwStatus,
DWORD dwMilliseconds);

引数

lDeviceId	デバイス番号
pdwStatus	測定待機状態を返します。　0: 測定処理中　1: 待機中
dwMilliseconds	タイムアウト時間[ms]: 0 ～ 0xFFFFFFFF タイムアウト時間が経過するか、または測定待機状態になると関数から戻ります。0 を指定すると即座に関数から戻ります。 0xFFFFFFFF を指定すると測定待機状態になるまで関数から戻りません。

戻り値　1: 成功　0:失敗

注記　測定待機状態が 1(待機中)が返った場合は、TmMeasExecAsync 関数、TmDarkMeasExecAsync 関数で測定を開始できます。
使用方法につきましては、TmMeasExecAsync 関数を参照してください。

TmCancelMeas

説明　非同期で計測中の場合、計測処理を キャンセルします。

宣言　long TmCancelMeas(long lDeviceId);

引数

lDeviceId	デバイス番号

戻り値　1: 成功　0:失敗

注記　TmMeasExecAsync 関数、または TmExecDarkMeasAsync 関数で測定を開始した場合、測定処理をキャンセルします。外部トリガ監視中は、外部トリガ監視を終了します。
この関数を呼んだ後、測定処理が終了するまで TmGetMeasStatus で測定終了になるまで監視してください。
通常はこの関数を使用せずに、TmGetMeasStatus 関数で測定状態を取得し、測定終了まで待ってください。

TmExecDarkMeas

説明　ダーク測定を実行します。ダーク測定が終了するまで関数から戻りません。

宣言　long TmExecDarkMeas(long lDeviceId, unsigned int nAveNum,
char nAllRange);

引数

lDeviceId	デバイス番号
nAveNum	平均化回数[回]:　1～100
nAllRange	0: 現在の積分時間、感度レンジのダーク測定を実行
	1: 全ての積分時間、感度レンジ、AC 点灯測定レンジのダーク測定を実行

戻り値　1: 成功　0:失敗

注記　ダーク測定では外部トリガ ON に設定されていても無効になります。また、ダーク測定終了後は測定終了出力(INDEX 出力)はありません。
TmOpenDevice 関数等でオープン後にダーク測定を未実行の場合は、測定結果には出荷時のダーク値が適用されます。毎接続ごとにダーク測定を実行することを推奨します。
現在の積分時間、感度レンジのダーク測定を実行した場合は、積分時間、感度レンジ、測定モードを変更した場合は、ダーク測定が未実行状態になります。
使用方法につきましては、TmMeasExec 関数を参照してください。

TmExecDarkMeasAsync

説明　全積分時間、感度レンジ、AC 点灯測定レンジのダーク測定を実行します。非同期で実行しますので本関数からはすぐに戻ります。

宣言　long TmExecDarkMeasAsync(long lDeviceId, unsigned int nAveNum);

引数

lDeviceId	デバイス番号
nAveNum	平均化回数[回]:　1～100

戻り値　1: 成功　　0:失敗

注記　必ず、TmGetMeasStatus 関数でダーク測定の終了を確認してください。
測定を強制終了する場合は、TmCancelMeas を呼んでください。
ダーク測定では外部トリガ ON に設定されていても無効になります。また、ダーク測定終了後は測定終了出力(INDEX 出力)はありません。
TmOpenDevice 関数等でオープン後にダーク測定を未実行の場合は、測定結果には出荷時のダーク値が適用されます。毎接続ごとにダーク測定を実行することを推奨します。

使用例

```
long lRet;
DWORD dwStatus;          //計測状況

lRet = TmExecDarkMeasAsync (lDeviceId,   10);        // 非同期で測定開始(平均化 10 回)

do {
    lRet = TmGetMeasStatus(lDeviceId,   &dwStatus, 0);        //測定状況を取得
} while (dwStatus == 0);    //測定中の場合は繰り返し測定状況を取得する

do {
    lRet = TmGetStandbyStatus(lDeviceId,   &dwStatus, 0);    //待機状態を取得
} while (dwStatus == 0);    //待機中になるまで繰り返し状態を取得する

////   全レンジダーク測定処理の終了  ////
```

TmGetDarkAll

説明　通常測定モードの全ての積分時間、感度レンジ、AC 点灯測定レンジのダーク値を取得します。

宣言　long TmGetDarkAll(long lDeviceId, DWORD dwDarkDataAll[ ]);

引数

lDeviceId	デバイス番号
dwDarkDataAll[ ]	ダーク値を格納する配列を指定します。 DWORD dwDarkDataAll[448]（DWORD 型 × 448)を指定してください。

戻り値　1：成功　0：失敗

注記　TmExecDarkMeas 関数、TmExecDarkMeasAsync 関数で全積分時間、感度レンジのダーク測定を実行後に取得可能です。

使用例

```
dwDarkDataAll[ 448];                    //ダーク結果格納用の配列
long lRet = TmGetDarkAll(lDeviceId,  dwDarkDataAll);      //ダーク測定結果取得
```

TmSetDarkAll

説明　全積分時間、感度レンジ、AC 点灯測定レンジのダーク値を設定します。

宣言　long TmSetDarkAll(long lDeviceId, DWORD dwDarkDataAll[ ]);

引数

lDeviceId	デバイス番号
dwDarkDataAll[ ]	ダーク値を格納してある配列を指定します。 DWORD dwDarkDataAll[448]（DWORD 型 × 448)を指定してください。

戻り値　1：成功　0：失敗

注記　前回の TM6101 との接続時に TmGetDarkAll で取得したダーク値を再設定する場合などに使用します。

TmResetDark

説明　ダーク測定結果をクリアして、ダーク測定を未実施状態にします。

宣言　long TmResetDark(long lDeviceId);

引数

lDeviceId	デバイス番号

戻り値　1：成功　0：失敗

注記　測定結果には出荷時のダーク値が適用されます。

4.4 基準値補正

TmExecChromaticityCorrect

説明　色度補正を実行します。色度補正対象の光源の分光特性データを配列で指定してください。

宣言　long TmExecChromaticityCorrect(long lDeviceId, double dSpectramData[ ]);

引数

lDeviceId	デバイス番号
dSpectramData [ ]	分光特性データを格納した配列を指定します。 380nm～780nm の 5nm おきのデータ(81 個)です。 double dData[81](double 型 × 81)を指定してください。

戻り値　1: 成功　0:失敗

注記　直前の測定結果が色度補正に反映されます。色度補正を実行する前に、色度補正対象の光源の測定をしてください。

使用例

```
long lRet;
double dSpectramData [ 81];                    //色度補正対象の光源の分光特性データ

for (int nDataNo = 0; nDataNo < 81; nDataNo++)
        dSpectramData [ 81] = ・・・・;        //分光データを登録しておく

lRet = TmSetAutoRange(lDeviceId,1);            //積分時間オートレンジ
lRet = TmSetAverageNum(lDeviceId,5);           //平均化 5 回

        //その他の測定条件はデフォルトで測定する

lRet = TmExecDarkMeas(lDeviceId, 10, 1);    //全レンジダーク測定実行(平均化 10 回)

//補正対象の光源の測定
lRet = TmMeasExec(lDeviceId);                  //積分時間オートレンジで測定実行

//色度補正を実行
lRet = TmExecChromaticityCorrect(lDeviceId,   dSpectramData);

    //以降は、測定処理等を実行
```

TmExecChromaticityCorrectByFile

説明　色度補正を実行します。色度補正対象の光源の分光特性データが保存された CSV 形式ファイルを指定してください。

宣言　long TmExecChromaticityCorrectByFile(long lDeviceId, char* cFilePath);

引数

lDeviceId	デバイス番号
cFilePath	分光特性データを保存してある CSV ファイルのフルパス名を指定してください。380nm～780nm（5nm おき）の、波長と分光特性データ（81 個）が記録されたファイルです。

戻り値　1：成功　0：失敗

注記　ファイル形式については、本器の取扱説明書を参照してください。
直前の測定結果が色度補正に反映されます。色度補正を実行する前に、色度補正対象の光源の測定をしてください。

TmGetChromaticityCorrectValue

説明　色度補正値を取得します。

宣言　long TmGetChromaticityCorrectValue(long lDeviceId, double dData[ ]);

引数

lDeviceId	デバイス番号
dData[ ]	色度補正値を格納する配列（double * 16）を指定してください。 double dData[16]（double 型 × 16）を指定してください。

戻り値　1：成功　0：失敗（または、補正未実行）

TmSetChromaticityCorrectValue

説明　色度補正値を設定します。

宣言　long TmSetChromaticityCorrectValue(long lDeviceId, double dData[ ]);

引数

lDeviceId	デバイス番号
dData[ ]	色度補正値を格納した配列（double * 16）を指定してください。 double dData[16]（double 型 × 16）を指定してください。

戻り値　1：成功　0：失敗

注記　前回の TM6101 との接続時に TmSetChromaticityCorrectValue 関数で取得した補正値を再設定する場合などに使用します。

TmResetChromaticityCorrect

説明　色度補正を実行済みの場合、色度補正値をクリアして色度補正未実施状態にします。

宣言　long TmResetChromaticityCorrect(long lDeviceId);

引数

lDeviceId	デバイス番号

戻り値　1：成功　0：失敗

TmExecIlluminanceCorrect

説明　照度補正を実行します。補正対象の光源の照度値を指定してください。

宣言　long TmExecIlluminanceCorrect(long lDeviceId, double dIlluminance);

引数

lDeviceId	デバイス番号
dIlluminance	補正対象の照度値[lx]

戻り値　1：成功　0：失敗

注記　直前の測定結果が照度補正に反映されます。照度補正を実行する前に、照度補正対象の光源の測定をしてください。

使用例

```
long lRet;

lRet = TmSetAutoRange(lDeviceId,1);                    //積分時間オートレンジ
lRet = TmSetAverageNum(lDeviceId,5);                   //平均化 5 回

        //その他の測定条件はデフォルトで測定する

lRet = TmExecDarkMeas(lDeviceId, 10, 1);    //全レンジダーク測定実行（平均化 10 回）

//補正対象の光源の測定
lRet = TmMeasExec(lDeviceId);                          //積分時間オートレンジで測定実行

//照度補正を実行
lRet = TmExecIlluminanceCorrect (lDeviceId,   1000);          //1000lx で補正

//以降は、測定処理等を実行
```

TmGetIlluminanceCorrectValue

説明　照度補正値を取得します。

宣言　long TmGetIlluminanceCorrectValue(long lDeviceId, double* pdData);

引数

lDeviceId	デバイス番号
pdData	照度補正値を返します。

戻り値　1：成功　0:失敗（または、補正未実行）

TmSetIlluminanceCorrectValue

説明　照度補正値を設定します。

宣言　long TmSetIlluminanceCorrectValue(long lDeviceId, double dData);

引数

lDeviceId	デバイス番号
dData	照度補正値

戻り値　1：成功　0:失敗

注記　前回の TM6101 との接続時に TmGetIlluminanceCorrectValue で取得した補正値を再設定する場合などに使用します。

TmResetIlluminanceCorrect

説明　照度補正を実行済みの場合、照度補正値をクリアして補正未実施状態にします。

宣言　long TmResetIlluminanceCorrect(long lDeviceId);

引数

lDeviceId	デバイス番号

戻り値　1：成功　0:失敗

TmExecLuminousFluxCorrect

説明　光束補正を実行します。補正対象の光源の光束値を指定してください。

宣言　long TmExecLuminousFluxCorrect(long lDeviceId, double dLuminousFlux);

引数

lDeviceId	デバイス番号
dLuminousFlux	補正対象の光束値[lm]

戻り値　1：成功　0:失敗

注記　直前の測定結果が光束補正に反映されます。光束補正を実行する前に、光束補正対象の光源の測定をしてください。
使用方法につきましては、照度補正と同じ流れになりますので、
TmExecIlluminanceCorrect を参照してください。

TmGetLuminousFluxCorrectValue

説明　光束補正値を取得します。

宣言　long TmGetLuminousFluxCorrectValue(long lDeviceId, double* pdData);

引数

lDeviceId	デバイス番号
pdData	光束補正値を返します。

戻り値　1：成功　0:失敗（または、補正未実行）

TmSetLuminousFluxCorrectValue

説明　光束補正値を設定します。

宣言　long TmSetLuminousFluxCorrectValue(long lDeviceId, double dData);

引数

lDeviceId	デバイス番号
dData	光束補正値

戻り値　1：成功　0:失敗

注記　前回の TM6101 との接続時に TmGetLuminousFluxCorrectValue で取得した補正値を再設定する場合などに使用します。

TmResetLuminousFluxCorrect

説明　光束補正を実行済みの場合、光束補正値をクリアして補正未実施状態にします。

宣言　long TmResetLuminousFluxCorrect(long lDeviceId);

引数

lDeviceId	デバイス番号

戻り値　1：成功　0:失敗

TmExecLuminousIntensityCorrect

説明　光度補正を実行します。補正対象の光源の光度値を指定してください。

宣言　long TmExecLuminousIntensityCorrect(long lDeviceId,
double LuminousIntensity);

引数

lDeviceId	デバイス番号
LuminousIntensity	補正対象の光度値[cd]

戻り値　1：成功　0:失敗

注記　直前の測定結果が光度補正に反映されます。あらかじめ、TmSetLightDistance 関数で光源までの距離を設定しておいてください。

光度補正を実行する前に、光度補正対象の光源の測定をしてください。補正方法は照度補正と同じ流れになりますので、TmExecIlluminanceCorrect を参照してください。

TmGetLuminousIntensityCorrectValue

説明　光度補正値を取得します。

宣言　long TmGetLuminousIntensityCorrectValue(long lDeviceId, double* pdData);

引数

lDeviceId	デバイス番号
pdData	光度補正値を返します。

戻り値　1：成功　0:失敗（または、補正未実行）

TmSetLuminousIntensityCorrectValue

説明　光度補正値を設定します。

宣言　long TmSetLuminousIntensityCorrectValue(long lDeviceId, double dData);

引数

lDeviceId	デバイス番号
dData	光度補正値

戻り値　1：成功　0:失敗

注記　前回の TM6101 との接続時に TmGetLuminousIntensityCorrectValue で取得した補正値を再設定する場合などに使用します。

TmResetLuminousIntensityCorrect

説明　光度補正を実行済みの場合、光度補正値をクリアして補正未実施状態にします。

宣言　long TmResetLuminousIntensityCorrect(long lDeviceId);

引数

lDeviceId	デバイス番号

戻り値　1：成功　0:失敗

TmGetUserCorrectData

説明　基準値補正値（色度補正値、照度補正値、光度補正値、光束補正値）を一括取得します。

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

引数

lDeviceId	デバイス番号
pstUserCorrect	補正値を返します。 TM_USER_CORRECT_DATA 構造体を指定してください。

戻り値　1：成功　0：失敗

使用例

```
TM_USER_CORRECT_DATA  stUserCorrect);                    //基準値補正構造体
TmGetUserCorrectData (lDeviceId,  &stUserCorrect);       //補正値を一括取得
```

TmSetUserCorrectData

説明

宣言　long TmSetUserCorrectData(long lDeviceId,
TM_USER_CORRECT_DATA stUserCorrect);

引数

lDeviceId	デバイス番号
stUserCorrect	補正値を格納した TM_USER_CORRECT_DATA 構造体を指定します。

戻り値　1：成功　0：失敗

4.5 測定結果の取得

TmGetIlluminanceValue

説明 照度値を取得します。

宣言 long TmGetIlluminanceValue(long lDeviceId, double* pdData);

引数

lDeviceId	デバイス番号
pdData	照度値[lx]を返します。

戻り値 1：成功　0：失敗

TmGetLuminousIntensityValue

説明 光度値を取得します。

宣言 long TmGetLuminousIntensityValue(long lDeviceId, double* pdData);

引数

lDeviceId	デバイス番号
pdData	光度値[cd]を返します。

戻り値 1：成功　0：失敗

TmGetLuminousFluxValue

説明 光束値を取得します。

宣言 long TmGetLuminousFluxValue(long lDeviceId, double* pdData);

引数

lDeviceId	デバイス番号
pdData	光束値[lm]を返します。

戻り値 1：成功　0：失敗

TmGetTristimulusValues

説明 三刺激値を取得します。

宣言 long TmGetTristimulusValues(long lDeviceId,
double* pX, double* pY, double* pZ);

引数

lDeviceId	デバイス番号
pX, pY, pZ	三刺激値（X 値,Y 値,Z 値）を返します。

戻り値 1：成功　0：失敗

TmGetChromaticityValue_xy

説明　色度値 x,y を取得します。

宣言　long TmGetChromaticityValue_xy(long lDeviceId, double* pX, double* pY);

引数

lDeviceId	デバイス番号
pX, pY	色度値(x,y)を返します。

戻り値　1：成功　0:失敗

TmGetChromaticityValue_uv

説明　色度値 u,v を取得します。

宣言　long TmGetChromaticityValue_uv(long lDeviceId, double* pU, double* pV);

引数

lDeviceId	デバイス番号
pU, pV	色度値(u,v)を返します。

戻り値　1：成功　0:失敗

TmGetCorrelatedColorTemperature

説明　相関色温度および Δuv を取得します。

宣言　long TmGetCorrelatedColorTemperature(long lDeviceId, double* pdTcp, double* pdDUV);

引数

lDeviceId	デバイス番号
pdTcp	相関色温度[K]を返します。
pdDUV	Δuv 値を返します。

戻り値　1：成功　0:失敗

注記　Δuv 値の絶対値が 0.02 を超えた場合は、相関色温度の測定結果はエラーとなります。

TmGetSpecialColorRenderingIndex

説明　特殊色評価数 Ri を取得します。

宣言　long TmGetSpecialColorRenderingIndex(long lDeviceId, char nTestColorNo, double* pdData);

引数

lDeviceId	デバイス番号
nTestColorNo	試験色：0～14（試験色 1～15)を指定します。
pdData	特殊演色評価数 Ri を返します。

戻り値　1：成功　0:失敗

TmGetGeneralColorRenderingIndex

説明　平均色評価数 Ra を取得します。

宣言　long TmGetGeneralColorRenderingIndex(long lDeviceId, double* pdData);

引数

lDeviceId	デバイス番号
pdData	平均演色評価数 Ra 返します。

戻り値　1：成功　0：失敗

TmGetDominantWaveLength

説明　ドミナント波長および刺激純度を取得します。

宣言　long TmGetDominantWaveLength(long lDeviceId, double* pdDomiLen,
double* pdPurity);

引数

lDeviceId	デバイス番号
pdDomiLen	ドミナント波長[nm]を返します。
pdPurity	刺激純度[%]を返します。

戻り値　1：成功　0：失敗

TmGetMeasResultAll

説明　測定結果を一括取得します。

宣言　long TmGetMeasResultAll(long lDeviceId,
TM_MEAS_RESULT* pstMeasResult);

引数

lDeviceId	デバイス番号
pstMeasResult	測定結果を TM_MEAS_RESULT 構造体で返します。

戻り値　1：成功　0：失敗

注記　TM_MEAS_RESULT 構造体には、各測定結果と、各測定結果の有効/無効が格納されます。

使用例

```
TM_MEAS_RESULT　stMeasResult;　　//測定結果構造体

//測定結果一括取得
long lRet =　TmGetMeasResultAll (lDeviceId,　& stMeasResult);
```

TmGetDetectLevel

説明　16 個の各センサの検出レベルを取得します。

宣言　long TmGetDetectLevel(long lDeviceId, char nCh, double* pdLevel);

引数

lDeviceId	デバイス番号
nCh	センサ番号　　0 ～15 を指定（センサ 1 ～16）
pdLevel	検出レベルを返します。
	検出レベル：0.00～1.00 （0.00:アンダーフロー　1.00:オーバーフロー）

戻り値　1：成功　　0:失敗

使用例

4.6 状況取得

TmGetSerialNo

説明 オープン済みの TM6101 の製造番号を取得します。

宣言 long TmGetSerialNo(long lDeviceId, char* pSerial, DWORD nByteSize);

引数

lDeviceId	デバイス番号
pSerial	製造番号文字列を返します。16 バイト以上の char 型配列を指定してください。
nByteSize	pSerial で指定した配列のバイト数

戻り値 取得した製造番号の文字数　0:失敗

注記 文字列は char 型(8bit)の配列にアスキー文字列で返します(NULL 終端)。

TmCheckDevice

説明 オープン済みの TM6101 の状態を取得します。

宣言 long TmCheckDevice(long lDeviceId);

引数

lDeviceId	デバイス番号

戻り値 1: 異常なし　0:異常あり

注記 USB 接続、センサユニットとの接続に異常がある場合は 0(異常あり)が返ります。
TmGetLastError 関数でエラー内容を確認してください。

TmGetLastError

説明　　エラー情報を取得します。

宣言　　long TmGetLastError();

引数　　なし

戻り値　　エラー番号（0：正常）

注記　　ライブラリ関数を使用してエラーが返った場合、エラー内容を取得します。

エラー一覧

エラー番号	内容
1	関数に無効な引数を指定しています。引数を確認してください。
16	ダーク測定を実行していません。ダーク補正値を取得できません。
17	基準値補正を実行していません。基準値補正値を取得できません。
32	センサユニットが接続されていません。センサユニットの接続を確認してください。
33	TM6101 本体とセンサユニットの製造番号が一致しません。TM6101 本体とセンサユニットの製造番号を確認してください。
34	センサユニット接続ケーブルが逆に接続されています。ケーブルの接続方向を確認してください。
35	TM6101 本体、または指定したシリアル番号の TM6101 が PC へ接続されていません。USB 等の接続および、ドライバソフトのインストールが正常に行われているか確認してください。
64	測定がタイムアウトしました。本体およびセンサユニットが正常に接続されているか確認してください。外部トリガ測定の場合は、トリガ入力を確認してください。
65	測定に失敗しました。本体およびセンサユニットが正常に接続されているか確認してください。
66	測定結果の取得関数、基準値補正の実行関数で返ります。色演算が不能の場合、または基準値補正が不能の場合に返ります。
67	測定結果が無効、または測定が未実行です。
68	センサ検出レベルがオーバーフローです。積分時間、感度レンジを適切に設定して測定を再実行してください。
69	センサ検出レベルがアンダーフローです。故障の可能性があります。
70	測定結果が定格（100,000lx）を超えています。故障の原因になりますので、測定を中止してください。
71	相関色温度の Δuv が 0.02 を超えているため、演算結果は無効です。
72	現在測定中、または測定待機状態ではないため、測定を開始できません。
256	その他のエラーです。

4.7 構造体

測定条件構造体

```
typedef struct tagTmMeasSet
{
    DWORD   dwMeasMode;          // 測定モード　0:通常測定モード　1:AC 点灯測定モード
    DWORD   dwIntTime;           //積分時間　　0 - 10 を指定(0.1ms - 40ms に対応)
    DWORD   dwAmpSens[16];       //感度レンジ　 0:High　1:Low
    DWORD   dwAveNum;            //平均回数
    DWORD   dwPlc;               //電源周波数　 0:60Hz　1:50Hz
    DWORD   dwAcRange;           //AC 点灯測定モード測定レンジ　0 - 2 レンジ 1 - 3 に対応)
    DWORD   dwAcAveNum;          // AC 点灯測定モード平均回数
    DWORD   dwExtTrig;           //外部トリガ　　0: OFF　1: RisingEdge　2: FallingEdge
    DWORD   dwTrigDelay;         //トリガディレイ[msec]
    DWORD   dwTrigTimeout;       //外部トリガタイムアウト[msec]
    DWORD   dwIndexOutpTime;     //外部 IO INDEX 出力の ON 時間[msec]
    DWORD   dwAutoRange;         //オートレンジ　0:オートレンジ OFF　1:積分時間オート
                                 //              2:アンプ感度オート(通常測定モードのみ)
    DWORD   dwAutoLevelHigh;     //オートレンジ検出レベル上限値[%]
    DWORD   dwAutoLevelLow;      //オートレンジ検出レベル下限値[%]
    DWORD   dwStdIllumSel;       //基準の光　0:CIE 昼光　1:黒体放射　2:自動切り替え
    double  dLightDistance;      //測光距離 [m]
} TM_MEAS_SET, *PTM_MEAS_SET;
```

基準値補正値構造体

```
typedef struct tagTmUserCorrectData
{
    DWORD   dwChromaticityCorrectEnable;      //色度補正値有効/無効　0: 無効　1:有効
    double  dChromaticityGain[16];            //色度補正値
    DWORD   dwIlluminanceCorrectEnable;       //照度補正値有効/無効　0: 無効　1:有効
    double  dIlluminanceGain;                 //照度補正値
    DWORD   dwLuminousFluxCorrectEnable;      //光束補正値有効/無効　 0: 無効　1:有効
    double  dLuminousFluxGain;                //光束補正値
    DWORD   dwLuminousIntensityCorrectEnable; //光度補正値有効/無効　0: 無効　1:有効
    double  dLuminousIntensityGain;           //光度補正値
} TM_USER_CORRECT_DATA, *PTM_USER_CORRECT_DATA;
```

測定結果構造体

```
typedef struct tagTmMeasResult
{
    DWORD   dwIlluminanceEnable;        //照度値有効/無効    0:無効 1:有効
    double  dIlluminance;               //照度値[lx]
    DWORD   dwLuminousFluxEnable;       //光束値有効/無効    0:無効 1:有効
    double  dLuminousFlux;              //光束値[lm]
    DWORD   dwLuminousIntensityEnable;  //光度値有効/無効    0:無効 1:有効
    double  dLuminousIntensity;         //光度値[cd]
    DWORD   dwChromaticityEnable;       //三刺激値、色度値有効/無効    0:無効 1:有効
    double  dX;                         //三刺激値 X
    double  dY;                         //三刺激値 Y
    double  dZ;                         //三刺激値 Z
    double  dChromticity_x;             //色度値 x
    double  dChromticity_y;             //色度値 y
    double  dChromticity_u;             //色度値 u
    double  dChromticity_v;             //色度値 v
    DWORD   dwColorTempEnable;          //相関色温度、Δuv 有効/無効    0:無効 1:有効
    double  dTcp;                       //相関色温度[K]
    double  dDeltaUV;                   //Δuv
    DWORD   dwColorRenderingEnable;     //演色性有効/無効        0:無効 1:有効
    double  dRi[TEST_COLOR_NUM];        //特殊演色評価数 R1～R15
    double  dRa;                        //平均演色評価数 Ra
    DWORD   dwDominantEnable;           //ドミナント波長、刺激純度有効/無効
                                        //              0:無効 1:有効
    double  dDominant;                  //ドミナント波長[nm]
    double  dPurity;                    //刺激純度
} TM_MEAS_RESULT, *PTM_MEAS_RESULT;
```

注記

本ライブラリで使用している構造体は 8 バイト境界で配置されています。お客様の開発環境に合わせて、構造体メンバ配置を調整してください。

使用許諾契約書

重要　以下の契約書を慎重にお読みください。本使用許諾契約書（以下、本契約書とする）は、本ソフトウェア製品に関してお客様（個人または法人）と日置電機株式会社（以下、弊社とする）との間に締結される法的な契約書で、本ソフトウェア製品は、コンピュータソフトウェアおよびそれに関連した媒体、ならびに印刷物（取扱説明書などの文書）が含まれ、製品によっては電子文書が含まれます。
本ソフトウェア製品をインストール、複製、または使用することによって、お客様は本契約書の条項に拘束されることに承諾されたものとします。

本ソフトウェア製品は、著作権法および国際著作権条約をはじめ、その他の無体財産権に関する法律ならびに条約によって保護されています。本ソフトウェア製品は許諾されるもので、販売されるものではありません。

1. ライセンスの許諾　本契約書はお客様に以下の権利を許諾します。

お客様は、本ソフトウェア製品のコピー 1 部を特定の 1 台のコンピュータ上にインストールして使用することができます。

2. その他の権利および制限の説明

-1. リバースエンジニアリング、逆コンパイル、逆アセンブルの制限
お客様は、本ソフトウェア製品をリバースエンジニアリング、逆コンパイル、または逆アセンブルすることはできません。

-2. 構成部分の分離
本ソフトウェア製品は 1 つの製品として許諾されています。その構成部分を分離して複数のコンピュータで使用することはできません。

-3. 貸与
お客様は、本ソフトウェア製品を貸与またはリースすることはできません。

-4. ソフトウェアの譲渡
お客様は、本契約書に基づいてお客様のすべての権利を恒久的に譲渡することができます。ただしその場合、複製物を保持することはできず、ソフトウェア製品の一切（すべての構成部分、媒体、取扱説明書などの文書、および本契約書）を譲渡し、かつ受取人が本契約書の条項に同意することを条件とします。

-5. 解除
お客様が本契約書の条項および条件に違反した場合、弊社は、他の権利を害することなく本契約を解除することができます。そのような場合、お客様は本ソフトウェア製品の複製物およびその構成部分をすべて破棄しなければなりません。

3. 著作権

本ソフトウェア製品、付属の取扱説明書などの文書、および本ソフトウェア製品の複製物についての権原および著作権は、弊社またはその供給者が有するもので、本ソフトウェア製品は著作権法および国際条約の規定によって保護されています。したがって、お客様は本ソフトウェア製品を他の著作物と同様に扱わなければなりません。ただし、お客様はオリジナルを保存する以外の目的で使用しないことを厳守する限り、次の (1)(2) のいずれかを行うことができます。

(1) 本ソフトウェア製品コピーを 1 部のみ作成すること。

(2) 本ソフトウェア製品を 1 台のコンピュータ上へインストールすること。
お客様は、本ソフトウェア製品付属の取扱説明書など文書を複製することはできません。

4. デュアルメディアソフトウェア

お客様は、複数種類の媒体によって本ソフトウェア製品を受け取ることがあります。受け取る媒体の種類やサイズにかかわらず、お客様は、特定の１台のコンピュータに適する媒体を１つだけ使用することができ、別のコンピュータ上で残りの媒体を使用またはインストールすることはできません。また、本ソフトウェア製品の、上記に規定された恒久的な譲渡の場合を除いては、残りの媒体を別のユーザに貸与、リースあるいは譲渡することはできません。

5. 保証の範囲

-1. 弊社は、本ソフトウェア製品の仕様を予告なしに変更することがあります。

-2. 弊社は、本ソフトウェアが付属の取扱説明書に従って実質的に動作しない場合または本ソフトウェアの媒体または取扱説明書に物理的な瑕疵がある場合に、お買い上げ後１年間に限り、弊社の判断に基づき、交換または修補のいずれかにより対応するものとします。

-3. 上記 -2. 項の事態が火災、地震、第三者による行為その他の事故、お客様の故意もしくは過失、誤用その他異常な条件下での使用において生じるなど弊社の責に帰さない理由により生じた場合、弊社は保証の責任を負わないものとします。なお、以下に定める場合も保証の対象とはなりません。

(1) お客様によるお買い上げ後の輸送、移動、落下、その他の衝撃による故障

(2) 改造、不当な修理、その他の取り扱いが適切でなかったことによる故障

-4. 交換または修補後の製品の保証期間は、元の保証期間の残存期間の満了日または交換・修補された製品の引き渡し後６か月間の満了日のいずれか遅く到来する日までとします。

-5. 法律上の請求の原因の種類を問わず、いかなる場合においても、弊社およびその供給者は、この製品の使用または使用不能から生ずる本保証規定に規定されていないいかなる他の損害（事業利益の損失、事業の中断、事業情報の損失またはその他の金銭的損害を含むがこれらに限定されない）に関して、一切責任を負わないものとします。たとえ、弊社がかかる損害の可能性について知らされていた場合でも同様です。いかなる場合においても、弊社の責任は、欠陥のないソフトウェア製品と交換することをもって保証限度とします。

HIOKI

国内拠点

www.hioki.co.jp/

本社 〒386-1192 長野県上田市小泉 81

製品のお問い合わせ

0120-72-0560

9:00 ～ 12:00, 13:00 ～ 17:00
土・日・祝日を除く

TEL 0268-28-0560　FAX 0268-28-0569　info@hioki.co.jp

修理・校正のお問い合わせ

ご依頼はお買上店（代理店）または最寄りの営業拠点まで
お問い合わせはサービス窓口まで
TEL 0268-28-1688　cs-info@hioki.co.jp

2103 JA

編集・発行　日置電機株式会社

Printed in Japan