

Baudrate : 38400

parity : n

data bit : 8

stop bit : 1

STX = 0x02

ETX = 0x03

Checksum = ch1 + ch2 (기타 참조)

CR = 0x0d

1. Mode (측정 모드 설정)

-> STX + "Mode" + 0 or 1 or 2 + etx + checksum + CR

- 0 : 컬러측정모드 1 : Flicker(Jeita) 측정 모드 2 : Flicker(Contrast)

2. MeaWBS (컬러 1회 측정)

->STX+"MeaWBS"+ETX+Checksum+CR

<-STX+"OK"+005+Data(Lv,x,y,CCT,duv)+ETX+Checksum+CR

Ex) 반환되는 데이터는 floating hex 값

OK00544433B233E6A803F3E6D220A000000003BE93790::

Lv = 44433B23

x = 3E6A803F

y = 3E6D220A

CCT = 00000000

duv = 3BE93790

3. MeaWBC (컬러 연속 측정)

->STX+"MeaWBC"+ETX+Checksum+CR * 시작

<-STX+"OK"+005+Data(Lv,x,y,CCT,duv)+ETX+Checksum+CR

->STX+"MeaWBC"+ETX+Checksum+CR * 중단

* 한번 연속 측정을 시작하면 중단 할 때까지 결과 값이 나올때마다 반환 합니다.

* MeaWBC 명령을 한번 실행하면 연속 측정을 시작하며 다시 한번 보내면 연속 측정을 중단합니다.

4. Fmess (플리커 1회 측정)

->STX+"Fmess"+ETX+Checksum+CR

<-STX+"Fmes"+"065"+Data(65개)+ETX+Checksum+CR * Mode가 1일 경우

ex) Mode가 1일 때 반환 값(JEITA)

최종 결과는 음수(-)로 나타납니다.

64 Hz 까지 표시가 됩니다.

* 사용자가 렌즈를 완전히 차단한 다음 Dmes를 실행하면 이후의 모든 플리커 측정에 영향을 줍니다.

7. UcalW (2차 교정 데이터 입력)

-> STX+"UcalW"+Data(9)+Checksum+CR

<- STC+"UcalWOK"+Checksum+CR

* 2차 교정을 진행하기 위해서는 별도로 준비된 DLL 파일을 사용해야 합니다.

* DLL 파일 : GETCALI.dll

a. 먼저 2차 교정 데이터를 얻기 위해 아래와 같이 총 24개의 타겟 데이터와 측정 데이터를 준비합니다.

(1) Target Data Lv, x, y (White)

(2) Target Data Lv, x, y (Red)

(3) Target Data Lv, x, y (Green)

(4) Target Data Lv, x, y (Blue)

(5) Measured Data Lv, x, y (White)

(6) Measured Data Lv, x, y (Red)

(7) Measured Data Lv, x, y (Green)

(8) Measured Data Lv, x, y (Blue)

b. 24개의 데이터를 double 형으로 값을 할당한 다음 GETCALI.dll에서 inverse_f 함수에 위의 데이터를 호출한 다음 결과를 반환 받습니다.

* 아래의 샘플 소스 코드 참조

c. 반환 받은 Result(결과값은 float 형) 변수에 있는 결과값 9개를 UcalW 명령어 다음에 입력한 다음 내보냅니다.

* b 과정을 위한 샘플 소스 코드

```
#include "stdafx.h"
#include "stdio.h"
#pragma comment(lib, "GETCALI.lib")
extern "C" __declspec(dllimport) float* inverse_f(double Cal2[24]);
void main()
{
    double Cal2[24]={0.0,};
    float* Result;
    Cal2[0] = 0.6031;      // Target. White Lv
    Cal2[1] = 0.27601073; // Target. White x
    Cal2[2] = 0.28889634; // Target. White y
    Cal2[3] = 1.1737;     // Ref. Red Lv
    Cal2[4] = 0.288796516; // Target. Red x
    Cal2[5] = 0.29040479; // Target. Red y
    Cal2[6] = 41.581;     // Target. Green Lv
    Cal2[7] = 0.2755558;  // Target. Green x
    Cal2[8] = 0.275381621; // Target. Green y
    Cal2[9] = 340.0309;   // Target. Blue Lv
    Cal2[10] = 0.318787141; // Target. Blue x
    Cal2[11] = 0.326429309; // Target. Blue y
    Cal2[12] = 25.0;      // Measured. White Lv
    Cal2[13] = 0.363636364; // Measured. White x
    Cal2[14] = 0.378787879; // Measured. White y
    Cal2[15] = 38.0;      // Measured. Red Lv
    Cal2[16] = 0.37755102; // Measured. Red x
    Cal2[17] = 0.387755102; // Measured. Red y
    Cal2[18] = 1187.0;    // Measured. Green Lv
    Cal2[19] = 0.360128617; // Measured. Green x
    Cal2[20] = 0.381672026; // Measured. Green y
    Cal2[21] = 9935.0;    // Measured. Blue Lv
    Cal2[22] = 0.392190294; // Measured. Blue x
    Cal2[23] = 0.413149249; // Measured. Blue y
    Result = inverse_f(Cal2);
    for(int i=0;i<9;i++)
    {
        printf("%fWn", Result[i]);
    }
}
```

7. SNOK : OK 사운드 (1회 beep)

8. SNNG : NG 사운드 (3회 beep)

9. Comtest (테스트 프로토콜)

->STX+"Comtest"+ETX+Checksum+CR

<-STX+"ComtestOK"+ETX+Checksum+CR

10. 기타

*Checksum = STX ~ ETX까지 더한 값(2바이트의 결과를 1바이트씩 나눠서 2개로 전송)

unsigned char Makechecksum(unsigned char *data, int len)

```
{
    int i;
    unsigned char a = 0;
    for(i=0; i<len; i++)
        a = (char)(a + *(data+i));
    return a;
}
```

aCrc = Makechecksum(전송할 데이터(STX+명령어+ETX), Length);

ch1 = (aCrc & 0xF0) >> 4 | 0x30;

ch2 = (aCrc & 0x0F) | 0x30;

* 모든 명령어는 STX + 명령어 + ETX+ CheckSum(ch1) + CheckSum(ch2) + CR 형태로 출력되어야 합니다.

* 모든 측정 결과값(Color, Flicker 데이터)는 float 형입니다.

결과 값 읽는 방법

// 하나의 결과 Data 중 C1D74E91 라는 결과 값을 읽었을 경우 변환 방법

```
#include
typedef union{
    unsigned long hex;
    float flt;
}
HexFloat;
int main(void){
    HexFloat val;
    val.hex = 0xC1D74E91;
    printf("hex = %x, float =%f\n",val.hex, val.flt);
    // Result is -26.913363
    return 0;
}
```