EAF
Software Development Kit
Revision: 1.7.7
2025.11.18

概述

EAF 可轻松实现即插即用，无需安装任何驱动。ZWO 提供有 ASIAIR 智能天文摄影无线控制器，和 PC 端 ASIStudio，都能直接运行；同时兼容市面上主流的天文摄影软件，支持 ASCOM 与 INDI 平台，与第三方软件无缝衔接。除了手动和手柄对焦，EAF 还搭载了自主研发的智能自动对焦算法，精准灵活，更懂天文爱好者的需求。EAF 设备采用 5V USB 供电，直接连接相机、电脑或 ASIAIR 即可使用，低耗高效，经久耐用，让布线更简洁，体验更纯粹。

本文档旨在指导用户使用 SDK 搭建 EAF 设备使用环境，讲解通过 SDK 实现 EAF 通信的过程。

目录

1 SDK 使用介绍

这个软件开发包（SDK）描述了一组可以用来操作 EAF 电调焦设备的函数，通过 C、C++、C #等开发工具调用，适用于 x86 或 x64 的 Windows，Linux，和 MacOS 操作系统。
头文件：EAF_focuser.h
windows 下的导入库和动态库：EAF_focuser.lib，EAF_focuser.dll
在 Linux 的动态库和静态库：libEAFFocuser.a libEAFFocuser.so libsdbus-c++.so.2 libWrapperSdbus.so
MacOS 下的动态库和静态库：libEAFFocuser.dylib，libEAFFocuser.a
安装方法：
在 Windows 中，下载后解压 zip 文件到任何目录，并添加 DLL 的路径的系统环境变量，有时需要注销并重新登录。还可以将 DLL 置于包含应用程序可执行文件的文件夹中。

前期准备：

USB:

1.一款 EAF 设备
2.PC windows 环境
3.USB 数据线

蓝牙：

1.一款 EAF 设备
2.PC 环境且有蓝牙功能

2 枚举类型和结构体定义

2.1 typedef struct _EAF_INFO

```
{
    int ID; //EAF 设备 ID 值
    char Name[64]; //EAF 设备名称
    int MaxStep;//fixed maximum position //最大步进位置
} EAF_INFO;
//EAF 设备信息结构体
```

2.2 typedef enum _EAF_ERROR_CODE

```
{
    EAF_SUCCESS = 0, //操作成功
    EAF_ERROR_INVALID_INDEX,//非法的序号
    EAF_ERROR_INVALID_ID,//非法 ID 值
    EAF_ERROR_INVALID_VALUE,//无效参数
    EAF_ERROR_REMOVED, //无法找到设备，检测到设备移除
    EAF_ERROR_MOVING,//EAF 设备正在运行
    EAF_ERROR_ERROR_STATE,//EAF 设备状态错误
    EAF_ERROR_GENERAL_ERROR,//其他错误
    EAF_ERROR_NOT_SUPPORTED,//设备不支持
    EAF_ERROR_CLOSED,//设备已关闭
    EAF_ERROR_BATTER_INFO,//电池信息
    EAF_ERROR_INVALID_LENGTH,//不支持数据长度

    // BLE
    EAF_BLE_READ_DATA_FAILED = 50,//蓝牙读数据失败
    EAF_BLE_SEND_DATA_FAILED,//蓝牙发送数据失败
    EAF_BLE_CONNECT_FAILED,//蓝牙连接失败
    EAF_BLE_DISCONNECT, // 蓝牙断连
    EAF_BLE_PAIR_FAILED, // 蓝牙配对失败
    EAF_BLE_CLEAR_PAIR_FAILED, // 蓝牙清除配对失败
    EAF_BLE_PAIRING_TIMEOUT, // 蓝牙配对超时
    EAF_BLE_RECEIVE_TIMEOUT, // 蓝牙接收数据超时
    EAF_BLE_DEVICE_NOT_EXISTS, // 当前设备不支持蓝牙功能
    EAF_BLE_INVALID_CALLBACK,//无效的返回值
    EAF_BLE_NEW_PAIR_REQUEST,    //蓝牙新配对请求
    EAF_BLE_DATA_BUSY,//蓝牙数据繁忙
    EAF_BLE_CHECK_SIZE_FAILED,   // 固件发送长度与接收长度不匹配

    EAF_ERROR_END = -1
}EAF_ERROR_CODE;
EAF 设备错误代码
```

2.3 typedef struct _EAF_ID

```
{
    unsigned char id[8];//EAF id 值
}EAF_ID;
```

EAF ID 值封装结构体

2.4 typedef struct _EAF_TYPE

```
{
    char type[16]; //EAF 设备类型
}EAF_TYPE;
```

EAF 设备类型结构体

2.5 typedef struct _EAF_BLE_NAME

```
{
    char name[16]; //EAF 蓝牙名称
}EAF_BLE_NAME;
```

EAF 蓝牙名称封装结构体

2.6 typedef struct _EAF_ERROR_MSG

```
{
    char motor_error_code[3];     // 2 characters + 1 terminator  电机错误码
    char battery_error_code[3]; // 2 characters + 1 terminator     电池错误码
}EAF_ERROR_MSG;
```

EAF 错误信息结构体

2.7 typedef struct _EAF_BATTERY_INFO

```
{
    int battery_temp;  电池温度
    int battery_vol;       电池电量
    int battery_charge_curr;  电池充电状态
    int battery_percentage;   电池百分比
    int battery_discharge_curr;    电池未充电
    int battery_health;   电池健康程度
    int battery_charge_vol;
    int battery_num_of_cycles;
}EAF_BATTERY_INFO;
```

EAF 电池状态结构体

2.8 typedef struct _EAF_ALL_INFO

```
{
    int is_run;                          // 电机运行状态
    int backlash_steps;                  // The backlash steps.
    int current_steps;                   //当前步数
```

```
    float temperature;                  //设备温度
    int buzzer_state;                   // 蜂鸣器状态
    int reverse_state;                  // 方向状态
    int handle_pressed;                 // The handle is pressed.
    int handle_connect;                  // 连接状态处理
    int max_steps;                      // 最大步数
    int led_state;                      // Led 状态
    char motor_error_code[2];           // 电机错误码
    char battery_error_code[2];         // 电池错误码
    EAF_BATTERY_INFO battery_info;   // 电池信息
} EAF_ALL_INFO;
```

EAF 信息结构体

2.9 typedef enum _EAF_CONTROL_TYPE

```
{
    CONTROL_BLE_NAME = 0,
    CONTROL_LED,//LED 功能
    CONTROL_BUZZER,//蜂鸣器功能
    CONTROL_EAF_TYPE,
    CONTROL_BAT_INFO,
    CONTROL_ERR_CODE,
    CONTROL_FW_UPDATE_BLE,
    CONTROL_FW_UPDATE_USB,
    CONTROL_MAX_INDEX, // Maximum index.
} EAF_CONTROL_TYPE;
```

EAF 功能项结构体

2.10 typedef struct _EAF_CONTROL_CAPS

```
{
    char name[32];
    char description[128];
    bool isSupported;
    bool isWritable;
    int maxValue;
    int minValue;
    int defaultValue;
    EAF_CONTROL_TYPE controlType;
    char unused[32];
} EAF_CONTROL_CAPS;
```

EAF 控制信息结构体

2.11 typedef struct _BLE_DEVICE_INFO

```
{
    char name[64];
```

```
    char address[64]; // format bluetooth address
    int signalStrength;
    long long int bluetoothAddress;
} BLE_DEVICE_INFO_T;
```

EAF 蓝牙设备信息结构体

2.12 typedef void (*ConnStateCallback)(bool state);

EAF 蓝牙连接状态结构体

2.13 typedef void (*PairStateCallback)(EAF_ERROR_CODE state);

EAF 蓝牙配对状态结构体

3 函数定义

3.1 EAFBLEScan

语法：EAF_ERROR_CODE EAFBLEScan(int DurationMs, BLE_DEVICE_INFO_T* devices, int maxDeviceCount, int* actualDeviceCount)

用处：扫描附近蓝牙设备。

描述：如果你使用蓝牙控制 EAF，使用该接口用于扫描周围的蓝牙设备。

int DurationMs：扫描时长（毫秒）。

BLE_DEVICE_INFO_T* devices：用于存储设备信息的结构体数组。

int maxDeviceCount：可保存的最大设备数量。

int* actualDeviceCount：实际扫描到的设备数量。

示例：

```
    BLE_DEVICE_INFO_T info[256];
    EAF_ERROR_CODE err;
    int count;
scan:
    err = EAFBLEScan(3000, info, 100, &count);
    if (err != EAF_SUCCESS) {
        std::cout << "EAF BLE Scan error!" << std::endl;
        return -1;
    }
    else {
        std::cout << "EAF BLE Scan get : " << count << std::endl;
        for (int i = 0; i < count; i++) {
            std::string name = std::string(info[i].name);
            if (!strncmp(name.c_str(), "EAF", 3))
            {
                std::cout << i << "    " << name << std::endl;
            }

            if (name == EAF_AUTOCONNECT)
            {
                findFlag = 1;
                break;
            }
        }
    }
```

注意：

使用蓝牙接口，则不需要调用 EAFGetNum 和 EAFOpen 接口。

3.2 EAFBLEConnect

语法：EAF_ERROR_CODE EAFBLEConnect(const char* pDeviceName, const char* pDeviceAddress, int *ID)

用处：用于连接到 EAF 蓝牙设备。
描述：向设备对应名字的 EAF 设备发送连接请求并获取 ID 值。
const char* pDeviceName：设备名称。
const char* pDeviceAddress：设备地址。
int *ID：返回的连接设备 ID。
示例：

```
EAF_ERROR_CODE err = EAFBLEConnect("EAF Pro_90c92c", NULL, &Id1);
```

3.3 EAFBLERegPairStateCallback

语法：EAF_ERROR_CODE EAFBLERegPairStateCallback(int ID, PairStateCallback callback)
用处：注册 EAF 设备的蓝牙配对状态回调函数。
描述：
回调函数是可选的；如果不设置回调，将不会触发用户自定义操作。
注册前请确保设备 ID 有效且设备已上电。
若想停止接收配对状态更新，可传入 NULL 或使用相应 API 取消注册。

3.4 EAFBLEPair

语法：EAF_ERROR_CODE EAFBLEPair(int ID)
用处：执行与指定 ID 值的蓝牙进行配对操作。

3.5 EAFBLEClearPair

语法：EAF_ERROR_CODE EAFBLEClearPair(int ID)
用处：清除指定 ID 值的 EAF 设备蓝牙配对信息。

3.6 EAFBLEDisconnect

语法：EAF_ERROR_CODE EAFBLEDisconnect(int ID)
用处：断开指定 ID 值的 EAF 设备的蓝牙连接。

3.7 EAFBLERegConnStateCallback

语法：EAF_ERROR_CODE EAFBLERegConnStateCallback(int ID, ConnStateCallback callback)
用处：注册 EAF 设备的蓝牙连接状态回调函数。
描述：
ConnStateCallback callback：连接状态回调。返回值为：
true：已连接
false：未连接

3.8 EAFBLEgetAllInfo

语法：EAF_ERROR_CODE EAFBLEgetAllInfo(int ID, EAF_ALL_INFO *pAllInfo)
用处：通过蓝牙获取指定 ID 值的 EAF 设备的所有信息。
描述：
int ID：设备 ID。
EAF_ALL_INFO *pAllInfo：返回的数据结构。

3.9 EAFGetNum

语法：int EAFGetNum()

用处：获取已连接的 EAF 设备数量。

描述：

调用该函数可刷新设备列表。返回值 1 表示有 1 个 EAF 已连接。

3.10 EAFGetProductIDs

语法：int EAFGetProductIDs(int* pPIDs)

用处：获取 EAF 产品 ID。

描述：

获取每个滤光轮的产品 ID（Product ID）。首次调用时将 pPIDs 设为 0 获取长度，再分配内存。

注意：

此 API 将被弃用，请改用 EAFCheck。

3.11 EAFCheck

语法：int EAFCheck(int iVID, int iPID)

用处：检查设备是否为 EAF。

描述：

int iVID：设备的 VID，厂商 ID，取值为 0x03C3。

int iPID：设备的 PID。

示例：

```
int* pPIDs = new int[numPIDs];
EAF_ERROR_CODE err = EAFGetID(idnum - 1, pPIDs);
ret = EAFCheck(0x03C3, pPIDs[idnum - 1]);
if (ret == 1){
   std::cout << " Confirm that it is an EAF device ! " << std::endl;
}else{
   std::cout << " Device not is EAF! " << std::endl;
}
```

3.12 EAFGetID

语法：EAF_ERROR_CODE EAFGetID(int index, int* ID)

用处：获取某个所连接的 EAF 设备的 ID 值。

描述：

int index：滤镜轮（filter wheel）的索引值，取值范围为 0 到 N - 1，其中 N 由函数 GetNum()返回。

int* ID：指向滤镜轮 ID 的指针。该 ID 是一个唯一的整数，取值范围为 0 到 EAF_ID_MAX - 1。设备打开后，所有的后续操作都基于此 ID 进行，该 ID 在使用过程中不会改变。

返回值：

EAF_ERROR_INVALID_INDEX：表示传入的索引值无效。

EAF_SUCCESS：表示操作成功。

注意：

此 ID 值将在几乎所有的控制和获取信息 API 中使用。

3.13 EAFOpen

语法： EAF_ERROR_CODE EAFOpen(int ID)

用处：打开指定 ID 的 EAF 设备，执行这一步后，你将获得指定 ID 值的 EAF 设备控制权。

3.14 EAFGetProperty

语法：EAF_ERROR_CODE EAFGetProperty(int ID, EAF_INFO *pInfo)

用处：得到指定 ID 值的 EAF 设备信息。

描述：

EAF _INFO *pInfo：指向 EAF 信息结构体的指针

int ID：所控制的 EAF 设备 ID 值。

3.15 EAFGetNumOfControls

语法：EAF_ERROR_CODE EAFGetNumOfControls(int ID, int* piNumberOfControls)

用处：获取 EAF 可用控制项数量。

注意：

EAF 必须已打开。

3.16 EAFGetControlCaps

语法：EAF_ERROR_CODE EAFGetControlCaps(int ID, int iControlIndex, EAF_CONTROL_CAPS *pControlCaps)

用处：获取指定 ID 值的 EAF 设备的控制项属性。

3.17 EAFMove

语法：EAF_ERROR_CODE EAFMove(int ID, int iStep)

用处：将 EAF 调焦器移动到指定绝对位置。

描述：

步进范围为 0～EAF_INFO::MaxStep

3.18 EAFStop

语法：EAF_ERROR_CODE EAFStop(int ID)

用处：让指定 ID 值的 EAF 设备停止转动。

3.19 EAFIsMoving

语法：EAF_ERROR_CODE EAFIsMoving(int ID, bool *pbVal, bool* pbHandControl)

用处：检测调焦器是否在移动。

描述：

同时可判断是否由手动控制移动（手动模式下无法通过 EAFStop 停止）。

3.20 EAFGetPosition

语法：EAF_ERROR_CODE EAFGetPosition(int ID, int* piStep)

用处：获取指定 ID 值的 EAF 设备当前步进位置。

3.21 EAFResetPostion

语法：EAF_ERROR_CODE EAFResetPostion(int ID, int iStep)

用处：重置指定 ID 值的 EAF 设备的当前位置。

3.22 EAFGetTemp

语法：EAF_ERROR_CODE EAFGetTemp(int ID, float* pfTemp)

用处：获取当前指定 ID 值的 EAF 设备当前温度。

3.23 EAFSetBeep

语法：EAF_ERROR_CODE EAFSetBeep(int ID, bool bVal)

用处：设置指定 ID 值 EAF 设备的蜂鸣器状态。

描述：

若开启，调焦器开始移动时会发出提示音。

3.24 EAFGetBeep

语法：EAF_ERROR_CODE EAFGetBeep(int ID, bool* pbVal)

用处：获取指定 ID 值 EAF 设备的蜂鸣器状态。

3.25 EAFSetMaxStep

语法：EAF_ERROR_CODE EAFSetMaxStep(int ID, int iVal)

用处：设置指定 ID 值的 EAF 设备的最大步进位置。

3.26 EAFGetMaxStep

语法：EAF_ERROR_CODE EAFGetMaxStep(int ID, int* piVal)

用处：获取指定 ID 值的 EAF 设备的最大步进位置。

3.27 EAFStepRange

语法：EAF_ERROR_CODE EAFStepRange(int ID, int* piVal)

用处：获取指定 ID 值的 EAF 设备的步进范围。

3.28 EAFSetReverse

语法：EAF_ERROR_CODE EAFSetReverse(int ID, bool bVal)

用处：设置指定 ID 值的 EAF 设备转动方向。

3.29 EAFGetReverse

语法：EAF_ERROR_CODE EAFGetReverse(int ID, bool* pbVal)

用处：获取指定 ID 值的 EAF 设备转动方向。

3.30 EAFSetBacklash

语法：EAF_ERROR_CODE EAFSetBacklash(int ID, int iVal)

用处：设置 EAF 调焦器的反向间隙补偿值(0～255)。

3.31 EAFGetBacklash

语法：EAF_ERROR_CODE EAFGetBacklash(int ID, int* piVal)

用处：获取 EAF 调焦器的反向间隙补偿值(0～255)。

3.32 EAFClose

语法：EAF_ERROR_CODE EAFClose(int ID)

用处：关闭指定 ID 值的 EAF 设备。

3.33 EAFGetSDKVersion

语法：char* EAFGetSDKVersion()

用处：获取 SDK 版本号字符串，例如 "1, 4, 0"。

3.34 EAFGetFirmwareVersion

语法：EAF_ERROR_CODE EAFGetFirmwareVersion(int ID, unsigned char *major, unsigned char *minor, unsigned char *build)

用处：获取指定 ID 值的 EAF 设备的固件版本。

3.35 EAFGetSerialNumber

语法：EAF_ERROR_CODE EAFGetSerialNumber(int ID, EAF_SN* pSN)

用处：获取指定 ID 值的 EAF 设备的序列号。

描述：

EAF_SN* pSN：指向序列号结构体的指针。

若固件不支持，则返回 EAF_ERROR_NOT_SUPPORTED。

3.36 EAFGetBatteryInfo

语法：EAF_ERROR_CODE EAFGetBatteryInfo(int ID, EAF_BATTERY_INFO *pBatteryInfo)

用处：获取电池电量与电池温度。

描述：

当温度异常时返回 EAF_ERROR_BATTER_INFO，且温度为 -1。

3.37 EAFSetID

语法：EAF_ERROR_CODE EAFSetID(int ID, EAF_ID alias)

用处：为指定 ID 的 EAF 设备设置别名。

3.38 EAFGetType

语法：EAF_ERROR_CODE EAFGetType(int ID, EAF_TYPE* pEAFType)

用处：获取 指定 ID 值的 EAF 设备的型号类型。

3.39 EAFGetBLEName

语法：EAF_ERROR_CODE EAFGetBLEName(int ID, EAF_BLE_NAME* pEAFBLEName)

用处：获取指定 ID 值的 EAF 设备的蓝牙名称。

3.40 EAFSetBLEName

语法：EAF_ERROR_CODE EAFSetBLEName(int ID, EAF_BLE_NAME EAFBLEName)

用处：设置指定 ID 值的 EAF 设备的蓝牙名称。
描述：

名称长度需在 1~6 个字符之间。

修改成功后会断开连接，需要重新扫描连接。

例如：

EAF Pro_57fb5d -> EAF Pro_123456

3.41 EAFGetLedState

语法：EAF_ERROR_CODE EAFGetLedState(int ID, bool *bState)
用处：获取指定 ID 值的 EAF 设备的 LED 灯状态。

3.42 EAFSetLedState

语法：EAF_ERROR_CODE EAFSetLedState(int ID, bool bState)
用处：设置指定 ID 值的 EAF 设备的 LED 灯状态。

3.43 EAFGetErrorCode

语法：EAF_ERROR_CODE EAFGetErrorCode(int ID, EAF_ERROR_MSG* pErrorCode)
用处：获取指定 ID 值的 EAF 设备的错误码。

3.44 EAFSetShippingMode

语法：EAF_ERROR_CODE EAFSetShippingMode(int ID)
用处：设置指定 ID 值的 EAF 设备为运输模式。（目前未使用）
注意：

当前固件不通过 SDK 设置序列号，因此此接口未被使用。

3.45 EAFGetReason

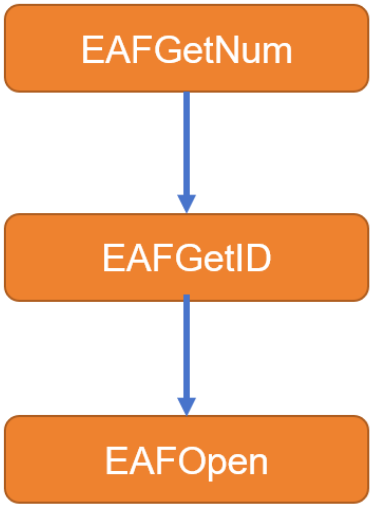
语法：EAF_ERROR_CODE EAFGetReason(int ID, int* pReason)
用处：获取指定 ID 值的 EAF 设备的关机原因。
描述：

0：正常
1：运输模式

4 建议的调用顺序

4.1 与 EAF 设备建立连接

适用于 USB



将 EAF 设备与 PC 电脑通过 USB 进行连接，Pc 上的 USB 驱动会轮询到该 EAF 设备，并根据报告描述符在电脑上生成该设备节点。

获取 EAF 设备个数：

EAF_API　　int EAFGetNum();

我们首先调用此接口，这个接口是第一个需要调用的 API，用于检查所连接的 EAF 设备个数，它的返回值为 EAF 设备个数。

获取所要控制的设备 ID

* EAF_API　　EAF_ERROR_CODE EAFGetID(int index, int* ID);

这个函数用于获取所连接 EAF 的 ID 值，

Index：为 EAFGetNum()获取的 EAF 编号，输入参数为 EAFGetNum() - 1,若您连接了多个，EAF 设备，可用 for 循环来获取每个 EAF 设备的 ID 值。ID 值很重要，我们将会在几乎所有的 API 中使用它（USB 通信模式）。

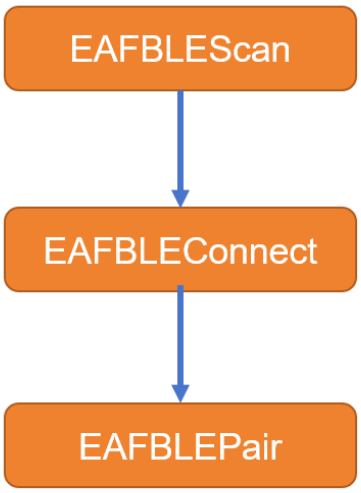
打开设备：

* EAF_API　　EAF_ERROR_CODE EAFOpen(int ID);

这是一个 open 函数，用于打开选择的 EAF 设备用于接下来的通信事件，只有调用这个函数并返回 EAF_SUCCESS 后，才可以控制 EAF 设备。它的参数 ID 就是 GetID 函数获取的 id 值。

完成如上步骤，就可以通过其他命令接口与 EAF 设备通信。

适用于蓝牙



扫描设备

首先，我们要打开 EAF 硬件设备上的开机按钮，开关调至 ON。在蓝牙通信模式下，我们首先要找到设备。

EAF_API EAF_ERROR_CODE EAFBLEScan(int DurationMs, BLE_DEVICE_INFO_T* devices, int maxDeviceCount, int* actualDeviceCount);

这个函数是蓝牙通信第一个需要调用的 API，它会通过蓝牙模块扫描附近的蓝牙设备，并将设备信息存储到 BLE_DEVICE_INFO_T 结构体中。

DurationMs:一次调用扫描的时间，

BLE_DEVICE_INFO_T:存放设备信息的结构体，

maxDeviceCount:一次扫描最大获取设备数，

actualDeviceCount：实际扫描到的设备数。

```
typedef struct _BLE_DEVICE_INFO {
    char name[64];
    char address[64]; // format bluetooth address
    int signalStrength;
    long long int bluetoothAddress;
} BLE_DEVICE_INFO_T;
```

结构体中存放了扫描到的设备名称等信息。

若通过蓝牙找到了 EAF 设备，接着开始连接它。

EAF_API EAF_ERROR_CODE EAFBLEConnect(const char* pDeviceName, const char* pDeviceAddress, int *ID);

这个函数是与某个蓝牙设备建立通信的 API，

输入参数 pDeviceName 表示经过 EAFBLEScan 函数所扫描到的设备名称，

pDeviceAddress 可默认置为 NULL，

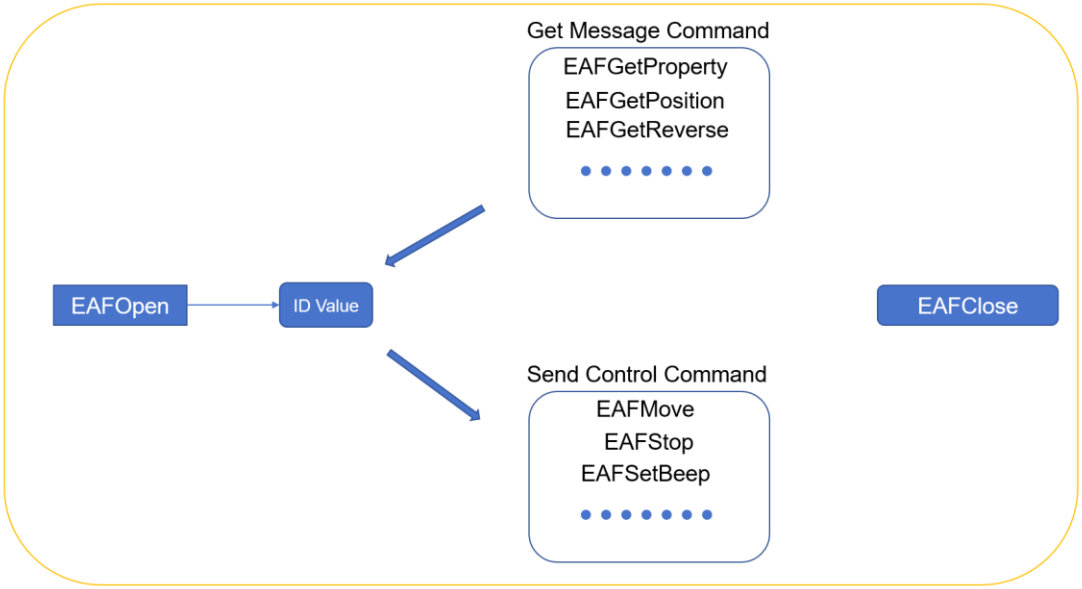
传入 ID 指针用于获取所连接设备的 ID 值，我们将会在几乎所有的 API 中使用它（蓝牙通信模式）。

当与 EAF 设备建立连接之后，需要调用配对接口与设备完成绑定。

EAF_API EAF_ERROR_CODE EAFBLEPair(int ID);

这个函数是请求与 EAF 设备配对的 API，根据传入的 ID 值与 EAF 设备建立通信。主机会与对应 ID 的 EAF 设备绑定完成。自此可以调用其他命令接口与 EAF 设备通信。

4.2 EAF 设备通信交互



EAF SDK 支持如下交互命令：（适用于蓝牙/USB）

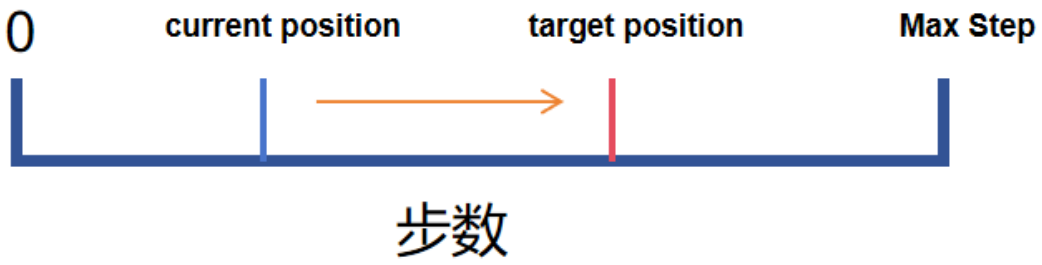
获取数据		控制设备	
获取 ID	EAFGetID	移动位置	EAFMove
获取设备数据	EAFGetProperty	停止转动	EAFStop
获取控制项信息	EAFGetControlCaps	重置位置	EAFResetPostion
获取运动状态	EAFIsMoving	设置蜂鸣器开关	EAFSetBeep
获取当前位置	EAFGetPosition	设置最大步数	EAFSetMaxStep
获取设备温度	EAFGetTemp	设置移动方向	EAFSetReverse
获取蜂鸣器状态	EAFGetBeep	设置回差间隙	EAFSetBacklash
获取最大步数	EAFGetMaxStep	设置蓝牙名称	EAFSetBLEName
获取步进范围	EAFStepRange	设置灯光状态	EAFSetLedState
获取移动方向	EAFGetReverse		
获取回差间隙	EAFGetBacklash		
获取 SDK 版本	EAFGetSDKVersion		
获取固件版本	EAFGetFirmwareVersion		
获取设备序列号	EAFGetSerialNumber		
获取电池信息	EAFGetBatteryInfo		

获取 EAF 类型	EAFGetType		
获取蓝牙名称	EAFGetBLEName		
获取灯光状态	EAFGetLedState		
获取错误信息	EAFGetErrorCode		
获取关闭原因	EAFGetReason		

EAF 设备内部保存了最大可移动步长，因此在控制它移动的时候，我们需要先获取 EAF 设备的当前位置和最大位置，以防控制命令超过最大补偿导致命令失效。

获取位置信息：

EAF_API EAF_ERROR_CODE EAFGetPosition(int ID, int* piStep);
此函数是用于获取 EAF 当前步数，输入参数 ID 为 GetID 函数获取的 id 值，结果会被写入传入的指针 piStep 所指的内存中。
EAF_API EAF_ERROR_CODE EAFSetMaxStep(int ID, int iVal);
此函数是用于获取 EAF 最大可运动步数的，输入参数 ID 为 GetID 函数获取的 id 值，结果会被写入传入的指针 iVal 所指的内存中。
因此得到了 EAF 设备当前位置以及可移动的最大位置。



控制 EAF：

使用如下函数来控制 EAF 设备的转动调焦。
EAF_API EAF_ERROR_CODE EAFMove(int ID, int iStep);
此函数是用于控制 EAF 设备运行的，输入参数 ID 为 GetID 函数获取的 id 值，输入参数 iStep 是运行到 iStep 步数位置。注意输入的 iStep 目标位置不能超过最大位置。
想要让 EAF 停下来，使用如下函数让 EAF 停止运行。
* EAF_API EAF_ERROR_CODE EAFStop(int ID);
此函数是用于让 EAF 设备停止运行，输入参数 ID 为 GetID 函数获取的 id 值。EAF 会停在当前所在位置。

4.3 关闭设备

我们目前不再使用此 ID 的 EAF 设备了，需要关闭 EAF 设备，调用如下函数完成。
适用于 USB：
* EAF_API EAF_ERROR_CODE EAFClose(int ID);
此函数是用于关闭 EAF 设备的，输入参数 ID 为 GetID 函数获取的 id 值。执行此命令后，

此 ID 设备将无法再通过 API 控制。

适用于蓝牙：

* EAF_API EAF_ERROR_CODE EAFBLEDisconnect(int ID);

此函数是用于关闭 EAF 设备的，输入参数 ID 为 GetID 函数获取的 id 值。同样的，执行此命令后，此 ID 设备将无法再通过 API 控制。

更多接口可查看 EAF_focuser.h 头文件。[查看 EAF.focuser.h](../include/EAF_focuser.h)

5 特别提醒

若您在使用 SDK(Linux 环境下)时出现如下问题：
attached focuser :
index 0: EAF
select one
0
open error,are you root?,press any key to exit
已经可以获取到设备，但是无法打开设备。
解决方式：
在 lib/下有 eaf.rules 文件，需要将此文件放置到/etc/udev/rules.d/文件夹下，
使用命令：sudo cp eaf.rules /etc/udev/rules.d/
重新加载规则：sudo udevadm control --reload-rules
最后重新插拔 USB 可解决此问题。

6 更多信息

ZWO 官网：https://www.zwoastro.cn/