



**EFW filter wheel
Software Development Kit**

Revision: 1.8.4

2025.11.18

**All material in this publication is subject to change without notice
and is copyright Zhen Wang Optical company.**



概述

EFW 滤镜轮，产品专为便捷使用而设计，无需额外驱动，即插即用，让操作轻松简单。它具有低能耗特性，可直接通过电脑、ASI AIR 或相机供电，使用更加灵活省心。同时提供多种尺寸选择，包括常规版和 Mini 版，满足不同需求。高度适配主流滤镜，兼容性出色，可与支持 ASCOM 的各类三方软件无缝配合，是天文摄影及观测的理想选择。

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



目录

1 SDK 使用介绍	4
2 枚举类型和结构体定义	5
2.1 typedef struct _EFW_INFO	5
2.2 typedef enum _EFW_ERROR_CODE{	5
2.3 typedef struct _EFW_ID	5
3 函数定义	6
3.1 EFWGetNum	6
3.2 EFWGetProductIDs	6
3.3 EFW_API int EFWCheck	6
3.4 EFWGetID	6
3.5 EFWOpen	7
3.6 EFWGetProperty	7
3.7 EFWGetPosition	7
3.8 EFWSetPosition	7
3.9 EFWSetDirection	8
3.10 EFWGetDirection	8
3.11 EFWCalibrate	8
3.12 EFWClose	8
3.13 EFWGetSDKVersion	8
3.14 EFWGetHWErrorCode	8
3.15 EFWGetFirmwareVersion	8
3.16 EFWGetSerialNumber	8
3.17 EFWSetID	8
4 建议的调用顺序	9
4.1 与 EFW 设备建立连接	9
4.2 EFW 设备通信交互	10
4.3 关闭设备	11
5 特别提醒	12
6 更多信息	12



1 SDK 使用介绍

Change History

Change date	revision	comment
2025.11.4	1.8.3	Add API EFWCheck
2017.2.14	2.1	Add API EFWCalibrate
2016.11.15	2.0	Add EFW_ERROR_CODE: EFW_ERROR_CLOSED Add EFWGetProductIDs EFWOpen: change argument to ID

这个软件开发包 (SDK) 描述了一组可以用来操作 EFW 滤镜轮设备的函数，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，以及 ARM 平台的 Linux。

头文件：EFW_filter.h

windows 下的导入库和动态库：EFW_filter.lib, EFW_filter.dll

在 Linux 的动态库和静态库：libEFWfilter.so, libEFWfilter.a

MacOS 下的动态库和静态库：libEFWfilter.dylib, libEFWfilter.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 EFW 滤镜轮设备
- 2.PC windows 环境
- 3.USB 数据线

2 枚举类型和结构体定义

2.1 typedef struct _EFW_INFO

```
{  
    int ID;  
    char Name[64];  
    int slotNum;  
} EFW_INFO;
```

EFW 设备 ID 值，设备名称，滤镜轮数量信息结构体

2.2 typedef enum _EFW_ERROR_CODE{

```
    EFW_SUCCESS = 0, //操作成功  
    EFW_ERROR_INVALID_INDEX, //非法的序号  
    EFW_ERROR_INVALID_ID, //非法 ID 值  
    EFW_ERROR_INVALID_VALUE, //无效参数  
    EFW_ERROR_REMOVED, //无法找到设备，检测到设备移除  
    EFW_ERROR_MOVING, //滤镜轮正在运行  
    EFW_ERROR_ERROR_STATE, //滤镜轮状态错误  
    EFW_ERROR_GENERAL_ERROR, //其他错误  
    EFW_ERROR_NOT_SUPPORTED, //设备不支持  
    EFW_ERROR_INVALID_LENGTH, //不支持数据长度  
    EFW_ERROR_CLOSED, //设备已关闭  
    EFW_ERROR_END = -1  
} EFW_ERROR_CODE;
```

EFW 设备错误代码

2.3 typedef struct _EFW_ID

```
{  
    unsigned char id[8];  
} EFW_ID;
```

ID 值封装结构体

3 函数定义

3.1 EFWGetNum

语法: `int EFWGetNum()`

用处: 获取插入的 EFW 设备数量

描述:

`int` 返回值返回所获取的 EFW 设备数量, 返回 1 意味着有 1 个 EFW 设备已连接。

示例:

```
int idnum = EFWGetNum();
if(idnum <= 0) {
    std::cout << "Don't find EFW Device !" << std::endl;
    return -1;
}
```

注意:

这是使用 EFW SDK 所需要调用的第一个函数。

3.2 EFWGetProductIDs

语法: `int EFWGetProductIDs(int* pPIDs)`

用处: 获取 EFW 产品 ID。

描述:

获取每个滤镜轮的产品 ID (Product ID)。使用时, 首先将 `pPIDs` 设置为 0 来获取数组长度, 然后再分配相应大小的缓冲区以载入所有 PID。

注意:

此 API 将被弃用, 请改用 `EFWCheck`。

3.3 EFW_API int EFWCheck

语法: `int EFWCheck(int iVID, int iPID)`

用处: 检查设备是否为 EFW。

描述:

`int iVID`: 设备的 VID, 对于 EFW, 取值为 0x03C3。

`int iPID`: 设备的 PID。

示例:

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

3.4 EFWGetID

语法: `EFW_ERROR_CODE EFWGetID(int index, int* ID)`



用处：获取某个所连接的 EFW 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 `EFW_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：

EFW_ERROR_INVALID_INDEX：表示传入的索引值无效。

EFW_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。

3.5 EFWOpen

语法：`EFW_ERROR_CODE EFWOpen(int ID)`

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

3.6 EFWGetProperty

语法：`EFW_ERROR_CODE EFWGetProperty(int ID, EFW_INFO *pInfo)`

用处：得到指定 ID 值的 EFW 设备信息。

描述：

EFW_INFO *pInfo：指向 EFW 信息结构体的指针

int ID：所控制的 EFW 设备 ID 值。

注意：

此函数需要在控制 EFW 设备前使用，即在 `EFWSetPosition` 前调用，因为 `EFWSetPosition` 需要使用经过 `EFWGetProperty` 得到的滤镜轮数量。

3.7 EFWGetPosition

语法：`EFW_ERROR_CODE EFWGetPosition(int ID, int *pPosition)`

用处：获取 EFW 设备当前使用的滤镜轮位置。

描述：

int ID：所控制的 EFW 设备 ID 值。

int *pPosition：指向滤镜片槽位位置的指针。该值的范围是 0 到 M - 1，其中 M 为槽位总数。当滤镜轮正在转动时，该值为 -1。

3.8 EFWSetPosition

语法：`EFW_ERROR_CODE EFWSetPosition(int ID, int Position)`

用处：获取 EFW 设备当前使用的滤镜轮位置。

描述：

int ID：滤镜轮的 ID。

int Position：槽位位置，取值范围为 0 到 M - 1，其中 M 为槽位数量。

3.9 EFWSetDirection

语法: EFW_ERROR_CODE EFWSetDirection(int ID, bool bUnidirectional)

用处: 设置指定 ID 值的 EFW 滤镜轮旋转方向。

描述:

int ID: 滤镜轮的 ID。

bool bUnidirectional: 若设置为 true, 则滤镜轮将始终沿同一方向旋转。

3.10 EFWGetDirection

语法: EFW_ERROR_CODE EFWGetDirection(int ID, bool *bUnidirectional)

用处: 获取指定 ID 值的 EFW 滤镜轮旋转方向。

3.11 EFWCalibrate

语法: EFW_ERROR_CODE EFWCalibrate(int ID)

用处: 控制 EFW 设备进入校准状态。

3.12 EFWClose

语法: EFW_ERROR_CODE EFWClose(int ID)

用处: 关闭指定 ID 的 EFW 设备使其资源被释放。这应该是最后一个调用的函数。

3.13 EFWGetSDKVersion

语法: char* EFWGetSDKVersion()

用处: 得到 SDK 的版本字符串。

3.14 EFWGetHWErrorCode

语法: EFW_ERROR_CODE EFWGetHWErrorCode(int ID, int *pErrCode)

用处: 获取指定 ID 值的 EFW 滤镜轮的固件错误码。

3.15 EFWGetFirmwareVersion

语法: EFW_ERROR_CODE EFWGetFirmwareVersion(int ID, unsigned char *major, unsigned char *minor, unsigned char *build)

用处: 获取指定 ID 值的 EFW 滤镜轮的固件版本信息。

3.16 EFWGetSerialNumber

语法: EFW_ERROR_CODE EFWGetSerialNumber(int ID, EFW_SN* pSN)

用处: 获取指定 ID 值的 EFW 滤镜轮的序列号。

描述:

EFW_SN* pSN: 指向序列号结构体的指针。

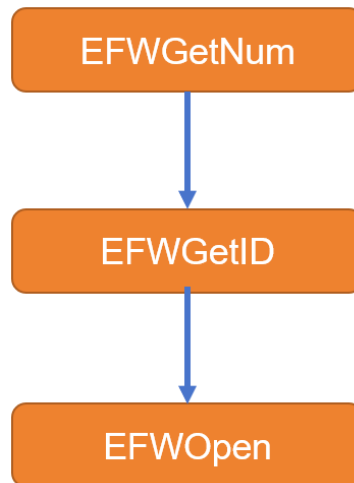
3.17 EFWSetID

语法: EFW_ERROR_CODE EFWSetID(int ID, EFW_ID alias)

用处: 为指定 ID 的 EFW 设备设置别名。

4 建议的调用顺序

4.1 与 EFW 设备建立连接



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

获取 EFW 设备个数：

```
EFW_API int EFWGetNum();
```

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 EFW 设备个数，它的返回值为 EFW 设备个数。

获取所要控制的设备 ID

```
* EFW_API EFW_ERROR_CODE EFWGetID(int index, int* ID);
```

这个函数用于获取所连接 EFW 的 ID 值，

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

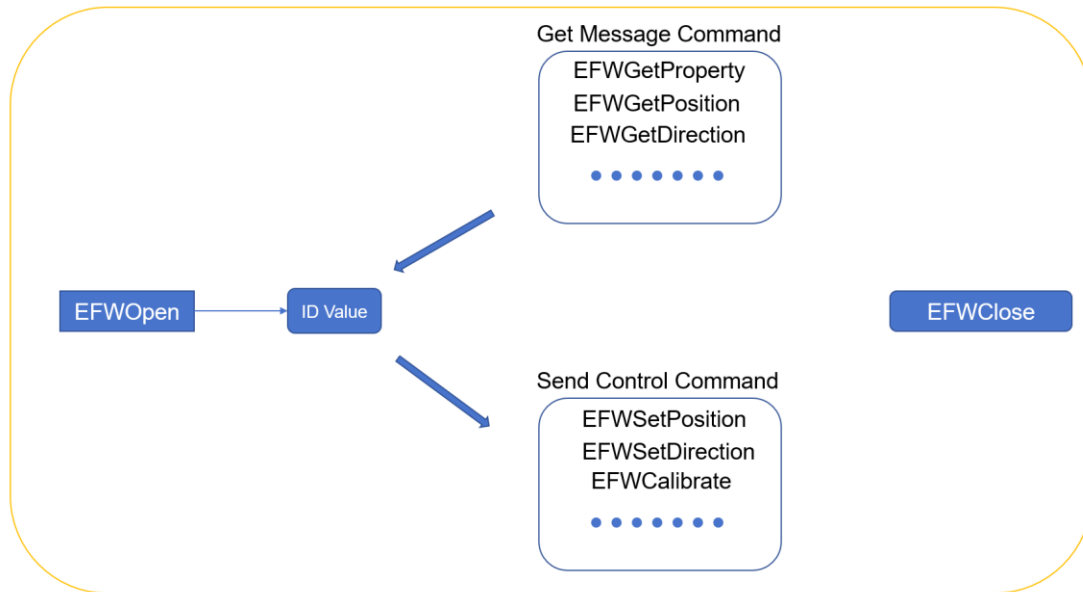
打开设备：

```
* EFW_API EFW_ERROR_CODE EFWOpen(int ID);
```

这是一个 open 函数，用于打开选择的 EFW 设备用于接下来的通信事件，只有调用这个函数并返回 EFW_SUCCESS 后，才可以控制 EFW 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 EFW 设备通信。

4.2 EFW 设备通信交互



EFW SDK 支持如下交互命令：

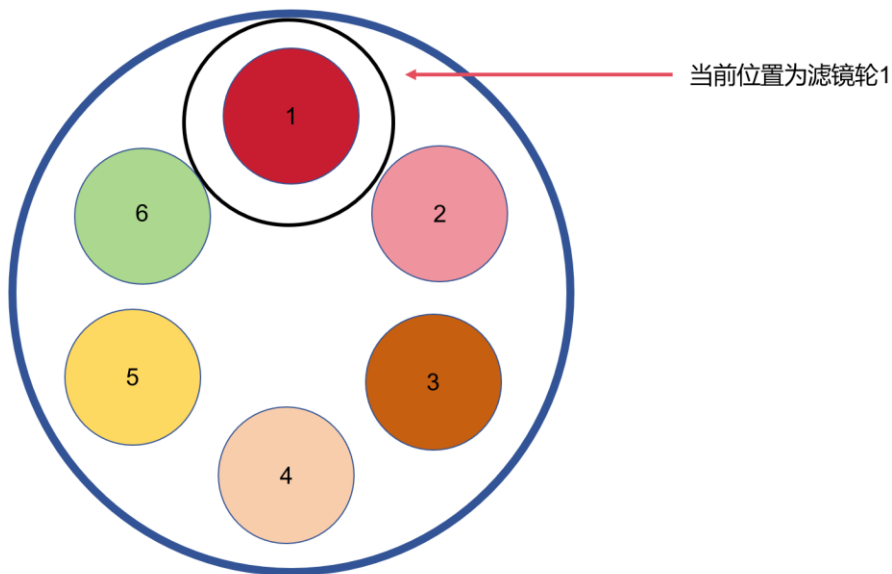
获取数据		控制设备	
获取 ID	EFWGetID	设置移动位置	EFWSetPosition
获取设备数据	EFWGetProperty	设置转动方向	EFWSetDirection
获取当前位置	EFWGetPosition	位置校验	EFWCalibrate
获取方向	EFWGetDirection	设置 ID 值	EFWSetID
获取 SDK 版本	EFWGetSDKVersion		
获取固件版本	EFWGetFirmwareVersion		
获取设备序列号	EFWGetSerialNumber		

EFW 在使用 **SetPosition** 命令进行旋转操作时，必须先调用 **EFWGetProperty** 接口获取滤镜轮个数等信息后才可进行控制操作。

获取位置信息：

```
EFW_API EFW_ERROR_CODE EFWGetPosition(int ID, int *pPosition);
```

此函数是用于获取 EFW 当前滤镜轮位置，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 **pPosition** 所指的内存中。



控制 EFW:

使用如下函数来控制 EFW 设备的转动调焦。

```
EFW_API EFW_ERROR_CODE EFWSetsPosition(int ID, int Position);
```

此函数是用于控制 EFW 设备运行的，输入参数 ID 为 GetID 函数获取的 id 值，输入参数 Position 是将选择几号滤镜。注意输入的 Position 目标不能超过滤镜轮个数。

4.3 关闭设备

我们目前不再使用此 ID 的 EFW 设备了，需要关闭 EFW 设备，调用如下函数完成。

```
* EFW_API EFW_ERROR_CODE EFWClose(int ID);
```

此函数是用于关闭 EFW 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。

****更多接口可查看 EFW_filter.h 头文件。****

5 特别提醒

若您在使用 SDK(Linux 环境下)时出现获取到设备，但是无法打开设备的情况，使用如下方式解决。

解决方式：

在 lib/下有 efw.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 efw.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/>