EAF
Software Development Kit
Revision: 1.7.7
2025.11.18

Overview

EAF devices offer effortless plug-and-play functionality without requiring any driver installation.ZWO offers the ASIAIR wireless controller for smart astrophotography and the PC-based ASIStudio software, both of which run directly. It is also compatible with mainstream astrophotography software on the market, supporting ASCOM and INDI platforms for seamless integration with third-party applications.Beyond manual and joystick focusing, EAF features a proprietary intelligent autofocus algorithm that is precise, flexible, and tailored to the needs of astrophotographers.Powered by 5V USB, the EAF connects directly to cameras, computers, or ASIAIR for low-power, high-efficiency operation. Its durable design simplifies cabling for a more streamlined experience.

This document guides users in setting up an EAF device environment using the SDK and explains the process of enabling EAF communication through the SDK.

Table of Contents

1 SDK Usage Guide

This Software Development Kit (SDK) describes a set of functions that can be used to operate EAF electronic focusers. These functions are accessible through development tools such as C, C++, and C#, and are compatible with x86 or x64 versions of Windows, Linux, and macOS operating systems.
Header file: EAF_focuser.h
Windows import library and dynamic library: EAF_focuser.lib, EAF_focuser.dll
Linux dynamic library and static library: libEAFFocuser.a, libEAFFocuser.so, libsdbus-c++.so.2, libWrapperSdbus.so
MacOS dynamic and static libraries: libEAFFocuser.dylib, libEAFFocuser.a
Installation method:
On Windows, download and extract the zip file to any directory. Add the DLL path to your system environment variables. You may need to log out and log back in. Alternatively, place the DLL in the folder containing your application executable.

Prerequisites:

USB:

1. An EAF device
2. A Windows PC environment
3. A USB cable

Bluetooth：

1. An EAF device
2. A Windows PC environment with Bluetooth functionality

2 Definition of Enum-type and Struct

2.1 typedef struct _EAF_INFO

```
{
    int ID; // EAF device ID value
    char Name[64]; // EAF device name
    int MaxStep;// fixed maximum position
} EAF_INFO;
// EAF device information structure
```

2.2 typedef enum _EAF_ERROR_CODE

```
{
    EAF_SUCCESS = 0, // Operation successful
    EAF_ERROR_INVALID_INDEX,// Invalid index
    EAF_ERROR_INVALID_ID,// Invalid ID value
    EAF_ERROR_INVALID_VALUE,// Invalid parameter
    EAF_ERROR_REMOVED, // Device not found, detected removal
    EAF_ERROR_MOVING,// EAF device is running
    EAF_ERROR_ERROR_STATE,// EAF device state error
    EAF_ERROR_GENERAL_ERROR,// Other error
    EAF_ERROR_NOT_SUPPORTED,// Device not supported
    EAF_ERROR_CLOSED,// Device is turned off
    EAF_ERROR_BATTER_INFO,// Battery information
    EAF_ERROR_INVALID_LENGTH,// Unsupported data length

    // BLE
    EAF_BLE_READ_DATA_FAILED = 50,// Bluetooth data read failed
    EAF_BLE_SEND_DATA_FAILED,// Bluetooth data send failed
    EAF_BLE_CONNECT_FAILED,// Bluetooth connection failed
    EAF_BLE_DISCONNECT, // Bluetooth disconnection
    EAF_BLE_PAIR_FAILED, // Bluetooth pairing failed
    EAF_BLE_CLEAR_PAIR_FAILED, // Bluetooth pairing removal failed
    EAF_BLE_PAIRING_TIMEOUT, // Bluetooth pairing timed out
    EAF_BLE_RECEIVE_TIMEOUT, // Bluetooth data reception timed out
    EAF_BLE_DEVICE_NOT_EXISTS,//Current device not support Bluetooth functionality
    EAF_BLE_INVALID_CALLBACK,// Invalid callback return value
    EAF_BLE_NEW_PAIR_REQUEST,    // New Bluetooth pairing request
    EAF_BLE_DATA_BUSY,// Bluetooth data busy
    EAF_BLE_CHECK_SIZE_FAILED,    // Firmware send length does not match receive length
    EAF_ERROR_END = -1
}EAF_ERROR_CODE;
```

EAF Device Error Codes**2.3 typedef struct _EAF_ID**

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

EAF ID value encapsulation structure

2.4 typedef struct _EAF_TYPE

```
{
    char type[16]; // EAF device type
}EAF_TYPE;
```

EAF device type structure

2.5 typedef struct _EAF_BLE_NAME

```
{
    char name[16]; // EAF Bluetooth name
}EAF_BLE_NAME;
```

EAF Bluetooth name encapsulation structure

2.6 typedef struct _EAF_ERROR_MSG

```
{
    char motor_error_code[3];      // 2 characters + 1 terminator    Motor error code
    char battery_error_code[3]; // 2 characters + 1 terminator      Battery error code
}EAF_ERROR_MSG;
```

EAF error message structure

2.7 typedef struct _EAF_BATTERY_INFO

```
{
    int battery_temp; Battery temperature
    int battery_vol;       Battery voltage
    int battery_charge_curr;  Battery charge current
    int battery_percentage;   Battery percentage
    int battery_discharge_curr;     Battery discharge current
    int battery_health;   Battery health
    int battery_charge_vol;
    int battery_num_of_cycles;
}EAF_BATTERY_INFO;
```

EAF Battery Status Structure

2.8 typedef struct _EAF_ALL_INFO

```
{
    int is_run;                         // Motor running state.
    int backlash_steps;                 // The backlash steps.
    int current_steps;                  // The currect steps.
    float temperature;                  // The temperature of eaf.
```

```
    int buzzer_state;                   // Buzzer status.
    int reverse_state;                  // Reverse state.
    int handle_pressed;                 // The handle is pressed.
    int handle_connect;                  // Handle connection state.
    int max_steps;                      // Max steps.
    int led_state;                      // Led state.
    char motor_error_code[2];           // Motor Error code
    char battery_error_code[2];         // Battery Error code
    EAF_BATTERY_INFO battery_info;    // Battery info.
} EAF_ALL_INFO;
```

EAF Information Structure

2.9 typedef enum _EAF_CONTROL_TYPE

```
{
    CONTROL_BLE_NAME = 0,
    CONTROL_LED,// LED functionality
    CONTROL_BUZZER,// Buzzer functionality
    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 Function Item Structure

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 Control Information Structure

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 Bluetooth Device Information Structure

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

EAF Bluetooth Connection State Structure

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

EAF Bluetooth Pairing State Structure

3 Function Declaration

3.1 EAFBLEScan

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

Purpose: Scans nearby Bluetooth devices.

Description: If you control EAF via Bluetooth, use this interface to scan for surrounding Bluetooth devices.

int DurationMs: Scan duration (milliseconds).

BLE_DEVICE_INFO_T* devices: Array of structures to store device information.

int maxDeviceCount: Maximum number of devices that can be stored.

int* actualDeviceCount: Actual number of devices scanned.

Example:

```
    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;
            }
        }
    }
```

Note:

When using the Bluetooth interface, there is no need to call the EAFGetNum and EAFOpen interfaces.

3.2 EAFBLEConnect

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

Purpose: Used to connect to an EAF Bluetooth device.

Description: Sends a connection request to the EAF device with the specified name and retrieves its ID value.

const char* pDeviceName: Device name.

const char* pDeviceAddress: Device address.

int *ID: Returned ID of the connected device.

Example:

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

3.3 EAFBLERegPairStateCallback

Syntax: EAF_ERROR_CODE EAFBLERegPairStateCallback(int ID, PairStateCallback callback)

Purpose: Registers a Bluetooth pairing state callback function for an EAF device.

Description:

The callback function is optional; if no callback is set, no user-defined actions will be triggered.

Ensure the device ID is valid and the device is powered on before registration.

To stop receiving pairing status updates, pass NULL or use the corresponding API to unregister.

3.4 EAFBLEPair

Syntax: EAF_ERROR_CODE EAFBLEPair(int ID)

Purpose: Performs a pairing operation with the Bluetooth device having the specified ID value.

3.5 EAFBLEClearPair

Syntax: EAF_ERROR_CODE EAFBLEClearPair(int ID)

Purpose: Clears Bluetooth pairing information for the EAF device with the specified ID value.

3.6 EAFBLEDisconnect

Syntax: EAF_ERROR_CODE EAFBLEDisconnect(int ID)

Purpose: Disconnect the Bluetooth connection of the EAF device with the specified ID value.

3.7 EAFBLERegConnStateCallback

Syntax: EAF_ERROR_CODE EAFBLERegConnStateCallback(int ID, ConnStateCallback callback)

Purpose: Registers a Bluetooth connection status callback function for EAF devices.

Description:

ConnStateCallback callback: Connection status callback. Returns:

true: Connected

false: Not connected

3.8 EAFBLEgetAllInfo

Syntax: EAF_ERROR_CODE EAF_BLEgetAllInfo(int ID, EAF_ALL_INFO *pAllInfo)

Purpose: Retrieves all information about an EAF device with the specified ID via Bluetooth.

Description:

int ID: Device ID.

EAF_ALL_INFO *pAllInfo: Returned data structure.

3.9 EAFGetNum

Syntax: int EAFGetNum()

Purpose: Retrieves the number of connected EAF devices.

Description:

Calling this function refreshes the device list. A return value of 1 indicates that 1 EAF is connected.

3.10 EAFGetProductIDs

Syntax: int EAFGetProductIDs(int* pPIDs)

Purpose: Retrieves EAF product IDs.

Description:

Retrieves the product ID for each filter wheel. On the first call, set pPIDs to 0 to obtain the length, then allocate memory.

Note:

This API will be deprecated. Please use EAFCheck instead.

3.11 EAFCheck

Syntax: int EAFCheck(int iVID, int iPID)

Purpose: Checks whether the device is an EAF.

Description:

int iVID: The device's VID (Vendor ID), with a value of 0x03C3.

int iPID: The device's PID.

Example:

```
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 an EAF!” << std::endl;
}
```

3.12 EAFGetID

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

Purpose: Retrieves the ID value of a connected EAF device.

Description:

int index: The index value of the filter wheel, ranging from 0 to N - 1, where N is returned by the GetNum() function.

int* ID: Pointer to the filter wheel ID. This ID is a unique integer ranging from 0 to EAF_ID_MAX - 1. After the device is opened, all subsequent operations rely on this ID, which remains unchanged

during use.

Return Values:

EAF_ERROR_INVALID_INDEX: Indicates an invalid index value was passed.

EAF_SUCCESS: Indicates the operation succeeded.

Note:

This ID value will be used in nearly all control and information retrieval APIs.

3.13 EAFOpen

Syntax: EAF_ERROR_CODE EAFOpen(int ID)

Purpose: Opens the EAF device with the specified ID. After executing this step, you will gain control over the EAF device with the specified ID value.

3.14 EAFGetProperty

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

Purpose: Retrieves information about the EAF device with the specified ID value.

Description:

EAF_INFO *pInfo: Pointer to the EAF information structure

int ID: The ID value of the EAF device being controlled.

3.15 EAFGetNumOfControls

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

Purpose: Retrieves the number of available controls in the EAF.

Note:

The EAF must be open.

3.16 EAFGetControlCaps

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

Purpose: Retrieves the control properties of an EAF device with the specified ID value.

3.17 EAFMove

Syntax: EAF_ERROR_CODE EAFMove(int ID, int iStep)

Purpose: Moves the EAF focuser to the specified absolute position.

Description:

Step range: 0 to EAF_INFO::MaxStep

3.18 EAFStop

Syntax: EAF_ERROR_CODE EAFStop(int ID)

Purpose: Stops the rotation of the EAF device with the specified ID value.

3.19 EAFIsMoving

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

Purpose: Detects whether the focuser is moving.

Description:

Simultaneously determines if movement is manually controlled (cannot be stopped via EAFStop in manual mode).

3.20 EAFGetPosition

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

Purpose: Retrieves the current step position of the EAF device with the specified ID value.

3.21 EAFResetPostion

Syntax：EAF_ERROR_CODE EAFResetPostion(int ID, int iStep)

Purpose: Reset the current position of the EAF device with the specified ID value.

3.22 EAFGetTemp

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

Purpose: Retrieves the current temperature of the EAF device with the specified ID value.

3.23 EAFSetBeep

Syntax: EAF_ERROR_CODE EAFSetBeep(int ID, bool bVal)

Purpose: Sets the beeper status for the EAF device with the specified ID value.

Description:

When enabled, an audible alert sounds when the focuser begins moving.

3.24 EAFGetBeep

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

Purpose: Retrieves the beeper status of the EAF device with the specified ID value.

3.25 EAFSetMaxStep

Syntax: EAF_ERROR_CODE EAFSetMaxStep(int ID, int iVal)

Purpose: Sets the maximum step position for the EAF device with the specified ID value.

3.26 EAFGetMaxStep

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

Purpose: Retrieves the maximum step position for the EAF device with the specified ID value.

3.27 EAFStepRange

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

Purpose: Retrieves the step range for the EAF device with the specified ID value.

3.28 EAFSetReverse

Syntax: EAF_ERROR_CODE EAFSetReverse(int ID, bool bVal)

Purpose: Sets the rotation direction of the EAF device with the specified ID value.

3.29 EAFGetReverse

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

Purpose: Retrieves the rotation direction of the EAF device with the specified ID value.

3.30 EAFSetBacklash

Syntax: EAF_ERROR_CODE EAFSetBacklash(int ID, int iVal)

Purpose: Sets the backlash compensation value for the EAF focuser (0 to 255).

3.31 EAFGetBacklash

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

Purpose: Retrieves the backlash compensation value (0 to 255) for the EAF focuser.

3.32 EAFClose

Syntax: EAF_ERROR_CODE EAFClose(int ID)

Purpose: Closes the EAF device with the specified ID value.

3.33 EAFGetSDKVersion

Syntax: char* EAFGetSDKVersion()

Purpose: Retrieves the SDK version number as a string, such as “1, 4, 0”.

3.34 EAFGetFirmwareVersion

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

Purpose: Retrieves the firmware version of an EAF device with the specified ID value.

3.35 EAFGetSerialNumber

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

Purpose: Retrieves the serial number of the EAF device with the specified ID value.

Description:

EAF_SN* pSN: Pointer to the serial number structure.

Returns EAF_ERROR_NOT_SUPPORTED if the firmware does not support this operation.

3.36 EAFGetBatteryInfo

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

Purpose: Retrieves battery charge level and battery temperature.

Description:

Returns EAF_ERROR_BATTER_INFO when temperature is abnormal, with temperature set to -1.

3.37 EAFSetID

Syntax: EAF_ERROR_CODE EAFSetID(int ID, EAF_ID alias)

Purpose: Sets an alias for the EAF device with the specified ID.

3.38 EAFGetType

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

Purpose: Retrieves the model type of the EAF device with the specified ID value.

3.39 EAFGetBLEName

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

Purpose: Retrieves the Bluetooth name of the EAF device with the specified ID value.

3.40 EAFSetBLEName

Syntax: EAF_ERROR_CODE EAFSetBLEName(int ID, EAF_BLE_NAME EAFBLEName)

Purpose: Sets the Bluetooth name for the EAF device with the specified ID value.

Description:

The name length must be between 1 and 6 characters.

After a successful modification, the connection will be disconnected and requires rescan and reconnection.

Example:

EAF Pro_57fb5d -> EAF Pro_123456

3.41 EAFGetLedState

Syntax: EAF_ERROR_CODE EAFGetLedState(int ID, bool *bState)

Purpose: Retrieves the LED status of an EAF device with the specified ID value.

3.42 EAFSetLedState

Syntax: EAF_ERROR_CODE EAFSetLedState(int ID, bool bState)

Purpose: Sets the LED status of an EAF device with the specified ID value.

3.43 EAFGetErrorCode

Syntax: EAF_ERROR_CODE EAFGetErrorCode(int ID, EAF_ERROR_MSG* pErrorCode)

Purpose: Retrieves the error code for the EAF device with the specified ID value.

3.44 EAFSetShippingMode

Syntax: EAF_ERROR_CODE EAFSetShippingMode(int ID)

Purpose: Sets the specified EAF device with ID value to shipping mode. (Currently unused)

Note:

The current firmware does not set the serial number via the SDK, so this interface is not utilized.

3.45 EAFGetReason

Syntax: EAF_ERROR_CODE EAFGetReason(int ID, int* pReason)

Purpose: Retrieves the shutdown reason for an EAF device with the specified ID value.

Description:

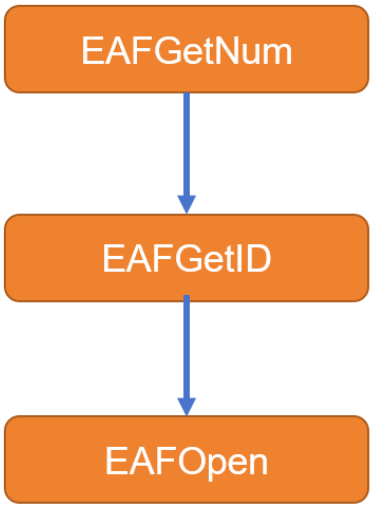
0: Normal

1: Transport Mode

4 Recommended Call Sequence

4.1 Connect to EAF equipment:

Compatible with USB



Connect the EAF device to a PC via USB. The USB driver on the PC will poll the EAF device and generate a device node on the computer based on the report descriptor.

To obtain the number of EAF devices:

EAF_API int EAFGetNum();

We first call this interface, which is the first API to be invoked. It checks the number of connected EAF devices and returns the count of EAF devices.

Retrieve the device ID to be controlled.

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

This function is used to obtain the ID value of the connected EAF device.

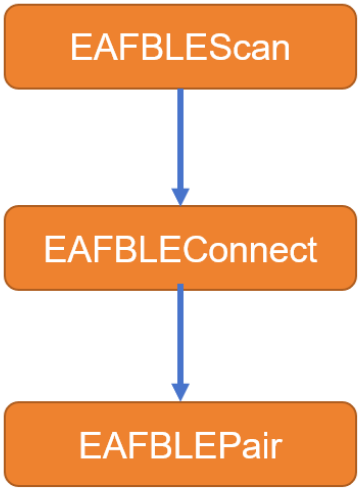
Index: The EAF number obtained from EAFGetNum(). The input parameter is EAFGetNum() - 1. If you have connected multiple EAF devices, you can use a for loop to obtain the ID value for each EAF device. The ID value is crucial, as we will use it in nearly all APIs (USB communication mode).

Open device:

* EAF_API EAF_ERROR_CODE EAFOpen(int ID);

This is an open function used to activate the selected EAF device for subsequent communication events. Only after calling this function and receiving an EAF_SUCCESS response can the EAF device be controlled. Its parameter ID corresponds to the ID value obtained from the GetID function. After completing the above steps, you can communicate with the EAF device through other command interfaces.

Compatible with Bluetooth:



Scanning Devices

First, press the power button on the EAF hardware device to turn it ON. In Bluetooth communication mode, we must first locate the device.

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

This function is the first API to call for Bluetooth communication. It scans nearby Bluetooth devices via the Bluetooth module and stores device information in the BLE_DEVICE_INFO_T structure.

DurationMs: Time for a single scan call,
BLE_DEVICE_INFO_T: Structure storing device information,
maxDeviceCount: Maximum number of devices retrieved per scan,
actualDeviceCount: Actual number of devices scanned.

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

The structure stores information such as the scanned device name.

If an EAF device is found via Bluetooth, the connection process begins.

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

This function is an API for establishing communication with a Bluetooth device.

The input parameter pDeviceName represents the device name scanned via the EAFBLEScan function.

pDeviceAddress can be set to NULL by default.

The ID pointer passed in is used to obtain the ID value of the connected device. We will use it in nearly all APIs (Bluetooth communication mode).

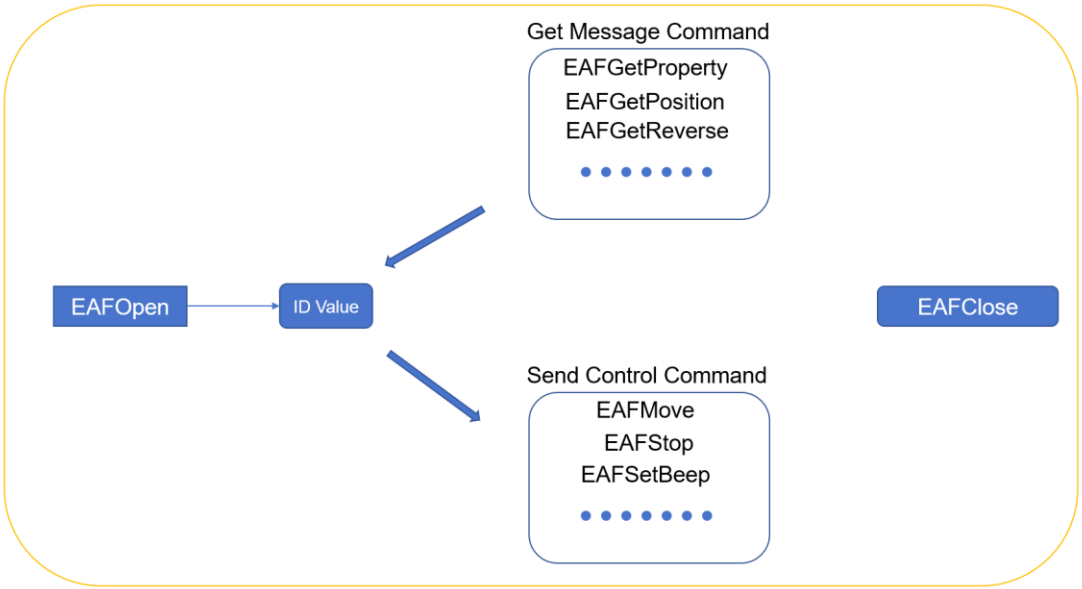
After establishing a connection with the EAF device, you must call the pairing interface to complete

the binding process.

EAF_API EAF_ERROR_CODE EAFBLEPair(int ID);

This function is the API for requesting pairing with an EAF device, establishing communication based on the input ID value. The host will complete binding with the EAF device corresponding to that ID. From this point onward, other command interfaces can be invoked to communicate with the EAF device.

4.2 EAF Equipment Communication Interaction：



The EAF SDK supports the following interactive commands: (Applicable to Bluetooth/USB)

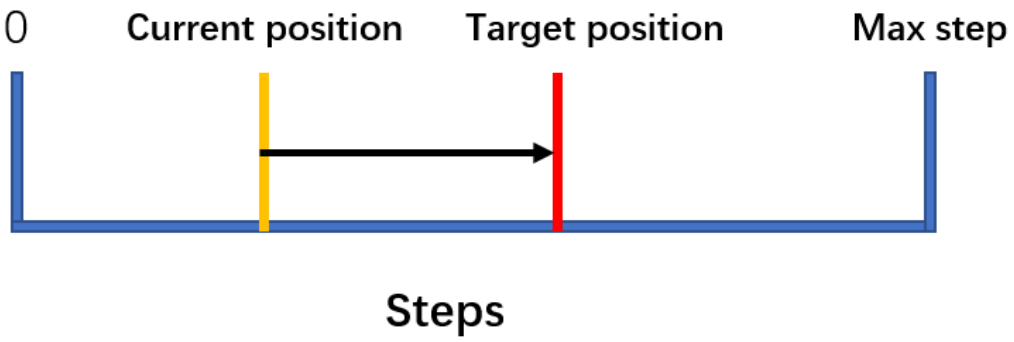
Acquire data		Control device	
Retrieve ID	EAFGetID	Move Position	EAFMove
Retrieve device data	EAFGetProperty	Stop Rotation	EAFStop
Retrieve control element information	EAFGetControlCaps	Reset Position	EAFResetPostion
Retrieve motion status	EAFIsMoving	Set Buzzer Switch	EAFSetBeep
Retrieve current location	EAFGetPosition	Set Maximum Steps	EAFSetMaxStep
Retrieve device temperature	EAFGetTemp	Set Movement Direction	EAFSetReverse
Retrieve buzzer status	EAFGetBeep	Set Hysteresis Gap	EAFSetBacklash
Retrieve maximum step count	EAFGetMaxStep	Set Bluetooth Name	EAFSetBLEName
Retrieve step range	EAFStepRange	Set Light Status	EAFSetLedState
Retrieve movement direction	EAFGetReverse		
Retrieve hysteresis gap	EAFGetBacklash		
Retrieve SDK version	EAFGetSDKVersion		
Retrieve firmware version	EAFGetFirmwareVersion		
Retrieve device serial number	EAFGetSerialNumber		

Retrieve battery information	EAFGetBatteryInfo		
Retrieve EAF Type	EAFGetType		
Retrieve Bluetooth Name	EAFGetBLEName		
Retrieve Light Status	EAFGetLedState		
Retrieve Error Message	EAFGetErrorCode		
Retrieve Shutdown Reason	EAFGetReason		

The EAF equipment internally stores the maximum movable step length. Therefore, when controlling its movement, we must first obtain the EAF equipment's current position and maximum position to prevent control commands from exceeding the maximum compensation limit and causing command failure.

Obtaining position information:

EAF_API EAF_ERROR_CODE EAFGetPosition(int ID, int* piStep);
This function is used to obtain the current step number of the EAF. The input parameter ID is the value obtained from the GetID function, and the result will be written to the memory location pointed to by the passed pointer piStep.
EAF_API EAF_ERROR_CODE EAFSetMaxStep(int ID, int iVal);
This function is used to obtain the maximum number of steps the EAF can move. The input parameter ID is the value obtained from the GetID function, and the result will be written to the memory location pointed to by the passed pointer iVal.
Thus, the current position of the EAF device and its maximum movable position are obtained.



Control EAF：

Use the following function to control the rotation and focus adjustment of the EAF device.
EAF_API EAF_ERROR_CODE EAFMove(int ID, int iStep);
This function controls the operation of the EAF equipment. The input parameter ID is the ID value obtained from the GetID function, and the input parameter iStep specifies the step position to run to. Note that the input iStep target position must not exceed the maximum position.
To stop the EAF, use the following function to halt its operation.
* EAF_API EAF_ERROR_CODE EAFStop(int ID);
This function is used to stop the operation of an EAF device. The input parameter ID is the value obtained from the GetID function. The EAF will stop at its current position.

4.3 Shut down the equipment

We are no longer using the EAF device with this ID. To deactivate the EAF device, call the following function.
For USB：
* EAF_API EAF_ERROR_CODE EAFClose(int ID);
This function is used to disable an EAF device. The input parameter ID is the value obtained from the GetID function. After executing this command, the device with this ID will no longer be controllable via the API.

Applicable to Bluetooth:
* EAF_API EAF_ERROR_CODE EAFBLEDisconnect(int ID);
This function is used to shut down an EAF device. The input parameter ID is the ID value obtained from the GetID function. Similarly, after executing this command, the device with this ID will no longer be controllable via the API.

** For more interfaces, refer to the EAF_focuser.h header file. **

5 Special Notice

If you encounter the following issues when using the SDK (in a Linux environment):
attached focuser :
index 0: EAF
select one
0
open error,are you root?,press any key to exit
The device is detectable but cannot be opened.
Solution:
The eaf.rules file is located in the lib/ directory. Move this file to the /etc/udev/rules.d/ directory using the command:
sudo cp eaf.rules /etc/udev/rules.d/
Reload the rules:
sudo udevadm control --reload-rules
Finally, unplug and replug the USB device to resolve the issue.

6 More Information

ZWO Official Website: https://www.zwoastro.cn/