

SIPS Standalone Filter Wheel Driver Interface

The ability to handle standalone filter wheels (filter wheels not controlled through the camera but utilizing own communication and power lines; thus needing their own dedicated drivers) was added to SIPS version 4. The Moravian Instruments SFW-XL series of standalone filter wheels is an example of such devices.

This document describes the interface between SIPS and standalone filter wheel drivers. It can be used for both implementing a new standalone filter wheel driver for SIPS and also for using the driver DLLs to control Moravian Instruments SFWs from other software packages.

Two driver DLLs are used to control all current Moravian Instruments SFW:

1. **'sfw.dll'** handles the SFW connected directly to the host computer through USB line.
2. **'sfweth.dll'** handles the SFW connected to the host computer over TCP/IP network. In such cases the **Moravian Camera Ethernet Adapter** device is used to confer the SFW USB interface to the TCP/IP. Up to four Gx/Cx cameras of arbitrary types and/or standalone filter wheels can be connected to this device using standard USB lines on one side and the standard 1 Gbps Ethernet interface (compatible with older 10/100 Mbps networks) can be attached on the other side. Because the TCP/IP protocol can be routed, the distance between camera and/or filter wheel and the host PC is virtually unlimited.

Remark:

Please note the Camera Ethernet Adapter ability to control SFW depends on its firmware version. Refer to the Camera Ethernet Adapter User's Guide for procedures to check and/or update the device firmware.

Please note the USB connected SFW requires installation of the kernel driver on the computer. The USB kernel drivers are identified by USB Vendor and Product identifiers. The operating system takes care of these drivers and the user usually does not need to install them manually (refer to the Installing and Using Drivers and Software manual for kernel driver installation instructions, please). Of course, no kernel drivers are needed when the Moravian Camera Ethernet Adapter is used to communicate with filter wheels.

Driver DLL Interface

SIPS drivers are Dynamic Link Libraries (DLLs) with a specific set of exported functions. All exported functions use standard "C" calling conventions,¹ so they can be called from arbitrary software package without problems with name decorations, etc.

Some functions are mandatory and the DLL must implement them to be accepted as a SIPS driver. Other functions may be optional, and the particular functionality could be used only if the driver exports (implements) these functions.

Driver API functions

Exported functions are grouped according to functionality. Individual function description follows this list of all exported functions. Functions marked * are optional, SIPS does not require them to be implemented (exported from the driver DLL). The rest is mandatory and has to be implemented for the package to be accepted as a SIPS driver.

Hint:

All driver public API function names are prefixed with the keyword "filters". This allows multiple drivers of different device classes to co-exist in single DLL. For instance, every device class offers functions like

¹ The calling conventions matter in 32-bit code only. The 64-bit Windows versions support only single universal calling convention, and compiler ignores calling convention specifications in the source code, which eliminates all compatibility issues.

Enumerate, Create, Initialize, etc. When the function names related to particular driver are prefixed, their names become unique (**filtersEnumerate, focuserEnumerate, domeEnumerate, ...**) and they can be implemented in one DLL.

Driver life-cycle functions:

```
filtersEnumerate
filtersCreate
filtersInitialize
filtersUnInitialize
filtersRelease
filtersConfigure*
filtersRegisterNotifyHWND
```

Driver information functions:

```
filtersGetBoolean
filtersGetInteger
filtersGetString
```

Filter wheel functions:

```
filtersEnumerateFilters
filtersSetFilter
filtersSetFilter2*
filtersReinitFilterWheel*
```

Data types used in SFW driver Application Programming Interface:

```
typedef int          INTEGER;
typedef unsigned int CARDINAL;
typedef unsigned char CHAR;
typedef unsigned char BOOLEAN;
typedef void *       ADDRESS;
struct              CSFW;
```

Driver life-cycle functions

```
void __cdecl filtersEnumerate(
    void ( __cdecl *CallbackProc )( CARDINAL )
)
```

Mandatory function.

Enumerate allows discovering of all filter wheels currently connected to the host PC. Its argument is a pointer to callback function **CallbackProc** with single unsigned integer argument. This callback function is called for each filter wheel connected to the host PC, with and the filter wheel ID² passed as an argument. This allows the application to offer the user a list of all connected filter wheels to choose from. Application can then offer the user a list of all connected filter wheels to choose the one with which the user wants to work.

If the application is designed to work with one filter wheel only or the filter wheel ID is known (for instance the application scans some .INI file where the identifier is defined), **Enumerate** needs not to be called at all.

Hint:

Please note the callback function uses the “C” calling conventions, as opposed to “stdcall” conventions often used in the Windows OS, in the case of 32-bit driver.

```
CSFW* __cdecl filtersCreate(
    CARDINAL Id
)
```

² ID is an unsigned integer number, unique to every SFW. The value is always 32-bit on both 32-bit and 64-bit driver versions.

Mandatory function.

Creates a driver instance. The return value is a “driver handle” (a pointer to an anonymous structure). This handle should be passed to all other driver API procedures to inform the driver which instance to work with.

```
void __cdecl filtersInitialize(  
    CSFW *PSFW  
)
```

Mandatory function.

Let the driver perform any driver-specific initialization (allocate necessary resources, open communication channels, etc.).

```
void __cdecl filtersUnInitialize(  
    CSFW *PSFW  
)
```

Mandatory function.

Let the driver perform any driver-specific cleanup operations (release previously allocated resources, close communication channels, etc.).

```
void __cdecl filtersRelease(  
    CSFW *PSFW  
)
```

Mandatory function.

When the filter wheel is no longer used, the handle must be released by the **filtersRelease** call. The CSFW *PSFW value, obtained with **filterCreate** call, becomes invalid after this call and should no longer be used.

```
BOOLEAN __cdecl filtersConfigure(  
    CARDINAL Id  
)
```

Optional function.

If the driver requires configuration (for instance TCP/IP based filter wheel requires definition of an IP address), it should implement this function.

This function can use already enumerated **ID** to re-configure existing instance of driver. Alternatively, passing the **ID -1** affects all the instances of the particular driver. User application should allow calling of **filtersConfigure** function with **ID = -1** also in case **filtersEnumerate** did not return any connected filter wheel.

SIPS always offer an empty line marked “unconfigured” for every driver exporting this function. After the configuration is finished, SIPS calls **filtersEnumerate** to get a fresh list of available cameras (e.g., from new Moravian Camera Ethernet Adapter device in the case the **filtersConfigure** updated the IP address, etc.).

```
void __cdecl filtersRegisterNotifyHWND(  
    CSFW *PSFW,  
    ADDRESS NotifyHWND  
)
```

Mandatory function.

The driver can notify the application that the filter wheel was plugged or unplugged. Notifications are sent as Windows messages to the windows, which **HWND** was passed as an argument to the **filtersRegisterNotifyHWND** function. Notification messages are:

```
#define WM_FW_CONNECT        1039  
#define WM_FW_DISCONNECT    1040
```

To disable the notification, the application can call **filtersRegisterNotifyHWND** with NULL as a handle.

Calling **filterRelease** automatically calls **filtersRegisterNotifyHWND** with **NotifyHWND = NULL**.

Driver information functions

```
BOOLEAN __cdecl filtersGetBoolean(  
    CSFW *PSFW,  
    CARDINAL Index,  
    BOOLEAN *Boolean  
)  
  
#define gbConnected          0 // TRUE if filter wheel currently connected  
#define gbInitialized        1 // TRUE if initialized (zero position found)  
#define gbMicrometerFilterOffsets 2 // TRUE if filter focusing offsets are in micrometers  
#define gbpConfigured        127 // TRUE if camera is configured
```

Mandatory function.

Function **filtersGetBoolean** returns a boolean value depending on the **Index** parameter. If the function does not “understand” passed **Index**, it returns FALSE.

```
BOOLEAN __cdecl filtersGetInteger(  
    CSFW *PSFW,  
    CARDINAL Index,  
    INTEGER *Num  
)  
  
#define giVersion1          0 // Filter wheel drive major version  
#define giVersion2          1 // Filter wheel drive minor version  
#define giVersion3          2 // Filter wheel drive build version  
#define giVersion4          3 // Filter wheel drive release version  
#define giFilterWheelId      4 // Unique identifier of the standalone filter wheel  
#define giFilters            5 // Number of filters available
```

Mandatory function.

Function **filtersGetInteger** returns integer value depending on the **Index** parameter. If the function does not “understand” passed **Index**, it returns FALSE.

```
BOOLEAN __cdecl filtersGetString(  
    CSFW *PSFW,  
    CARDINAL Index,  
    CARDINAL String_HIGH,  
    CHAR *String  
)  
  
#define gsFWDescription      0 // Filter wheel description  
#define gsManufacturer       1 // Manufacturer name  
#define gsSerialNumber       2 // Filter wheel serial number
```

Mandatory function.

Function **filtersGetString** returns string value depending on the **Index** parameter. If the function does not “understand” passed **Index**, it returns FALSE.

The string is copied to the buffer pointed by the **String** parameter. The function checks the maximum buffer size not to cause buffer overrun. The highest character index (index starts with 0) of the buffer must be passed as **String_HIGH** parameter. If the buffer is longer than the passed string, terminating zero character is also copied.

Filter wheel functions

```
BOOLEAN __cdecl filtersEnumerateFilters(  
    CSFW *PSFW,  
    CARDINAL Index,  
    CARDINAL Description_HIGH,  
    CHAR *Description,
```

```
CARDINAL *Color,  
INTEGER *Offset  
);
```

Mandatory function.

Enumerates all filters provided by the filter wheel. This enumeration does not use any callback, but the caller passes index beginning with 0 and repeats the call with incremented index until the call returns FALSE. Description string is passed similarly to **filtersGetString** call. Color parameter hints the Windows color, which can be used to draw the filter name in the application. The 'Offset' parameter indicates the focuser shift when the particular filter is selected.

Units of the 'Offset' can be micrometers or arbitrary focuser specific units (steps). If the units used are micrometers, driver should return TRUE from **filtersGetBoolean(gbMicrometerFilterOffsets, ...)** call.

```
BOOLEAN __cdecl filtersSetFilter(  
    CSFW *PSFW,  
    CARDINAL FilterIndex  
);
```

Mandatory function.

Sets the required filter.

```
BOOLEAN __cdecl filtersSetFilter2(  
    CSFW *PSFW,  
    CARDINAL FilterIndex1,  
    CARDINAL FilterIndex2  
);
```

Optional function.

This function works with devices, containing two filter wheels, like the SFW series. The dual wheel normally always sets the first (clear) position of each wheel if the filter from another wheel is used. However, this function allows to set any combination of positions in both wheels. So, this function can be useful for some special applications, requiring combination of two filters.

```
BOOLEAN __cdecl filtersReinitFilterWheel(  
    CSFW *PSFW  
);
```

Optional function.

The filter wheel performs the initialization, during which the zero-filter position is found and set.

Document updates

2024-07-09: Version 1.0 of SDK released.

- Initial Moravian Filter Wheel SDK release.

2025-06-25: Version 1.1 of SDK released.

- The **filtersSetFilter2** API function, intended to control dual wheel SFW series only, was added.

2025-08-07: Version 1.2 of SDK released.

- The SFW SDK documentation (this document) was updated to reflect the “filters” prefix of all driver API function names and updated driver life-cycle functions behavior.