Alienware AlienFX SDK 5.2

(Compatible with AlienFX SDK 1.0 and 2.x)

User's Guide

Contents

Revision History

Version 5.2 – February 22, 2019
Version 2.1 – March 26, 2012 – Alpha release
Version 2.0 – May 01, 2011 – Alpha release
Version 1.0 – March 24, 2009 – Alpha release

1. Introduction

1.1 Identification

The Dell Alienware AlienFX SDK is intended for those who develop applications running on AlienFX hardware platforms and/or platforms with attached AlienFX compatible devices.

1.2 Purpose

The purpose of this document is to provide a detailed, programmatic outline of the features and functions available in the Alienware AlienFX SDK library. Using this document and the accompanying software components, an application developer can control any available Alienware AlienFX devices attached to the system. By gaining control of the Alienware AlienFX ecosystem, an application may configure devices with colored lights to achieve various desired visual effects in response to application request(s).

1.3 Scope

This document encompasses the available features and functions of the library, including its functions, dependent libraries, data and methods needed for manipulation of color lights in the system.

2. Getting Started

2.1 Contents

This software development kit (SDK) contains the libraries, application samples, as well as this document. Upon installation of Alienware Command Center, these files are located in subdirectories of the installation directory (i.e., C:\Program Files\Alienware\Command Center) as listed below:

• **Documentation:**
[InstallDir]\AlienFX SDK\

• **Headers and Included files:**
[InstallDir]\AlienFX SDK\includes\

• **Dynamic Link Library:**

- [InstallDir]\AlienFX SDK\DLLs\x86\LightFX.dll
- [InstallDir]\AlienFX SDK\DLLs\x64\LightFX.dll

You can also find the same libraries at:

- \Windows\System32\LightFX.dll (64-bit version)
- \Windows\SysWOW64\LightFX.dll (32-bit version)

• **Samples:**
[InstallDir]\AlienFX SDK\Samples\

Warning:
1. The above libraries are not for distribution with the applications developed using Alienware AlienFX SDK
2. AlienFX Configurator and SDK managed libraries have been removed from the Alienware AlienFX SDK

2.2 System Requirements for Development

- For AlienFX platforms and peripherals released 2018 and beyond, AWCC 5.2 is required to be installed to control chassis and peripherals lighting using AlienFX SDK 5.x.
- For AlienFX platforms and peripherals released prior to 2018, AWCC 4.0 is required to be installed to control chassis and peripherals lighting using AlienFX SDK 1.0, 2.0 or 2.1
    - These AlienFX SDK versions are not supported anymore for development
    - The functions highlighted in section 4.1 are not supported anymore in AlienFX SDK 5.2

Operating system:
- For AlienFX SDK 5.x: Windows 10, 64-bits

Hardware & Software Requirements:
- Alienware AlienFX 5.2 compliant hardware: Application development with the library does not require any special hardware, although Alienware AlienFX SDK compliant hardware is useful for testing and debugging
- Alienware Command Center 5.2 installed: This SDK is distributed through AWCC 5.2.x installations

Recommendation: For optimal performance we recommend using latest devices and SDK for development

2.3 System Requirements for the end users

If application is developed using AlienFX SDK 5.2:
- Systems and peripherals released prior to 2018: AWCC 4.x
- Systems and peripherals released on 2018 onwards: AWCC 5.2 or higher

2.4 Library Linking

LightFX libraries and header files for static and dynamic linking are provided.

Explicit Dynamic Linking
Definitions of function names and function pointers have been included in the library header files for this purpose. Rather than use the import library, the library is loaded with a call to LoadLibrary along with a call to GetProcAddress for any desired functions of the library. By linking explicitly, the application launches even if the library is not available on the system.

2.5 How Alienware AlienFX SDK Works

Alienware AlienFX is, in a nutshell, an abstraction and translation library used to communicate with the lighting system. It supports a variety of device communication protocols and provides a common subset of features for getting and setting color values for RGB lights (LEDs, or other types) attached to the system. When initialized, the LightFX module library identifies all the Alienware AlienFX enabled hardware, or light controllers, attached to the system, and builds a list of these along with their physical positions in, on or around the mechanical enclosure (i.e., the chassis).

After identifying all the hardware, Alienware AlienFX waits for requests from the application through the function exports provided. The functions can be as simple as LFX_Light, in which all the decisions about state changes are done by the library, or as detailed as LFX_SetLightColor, which requires an index to a valid device and a valid light to set the color value. LFX_Light is preferred by most developers due to its ease of use and better performance.

2.6 Hello World with Alienware AlienFX SDK

The requisite programmer's example "hello world" can be implemented in Alienware AlienFX, albeit with a color value instead of a string. Here is what it looks like:

```
LFX_Initialize();

// Set all lights to blue in the state machine
LFX_Light(LFX_ALL, LFX_BLUE | LFX_FULL_BRIGHTNESS);

// Causes the physical color change
LFX_Update();

//Make the system wait
Sleep(100)

// Cleanup and detach from the system
LFX_Release();
```

While this example simply sets the color and immediately exits (thus restoring the previous state), it also demonstrates how easy it is to incorporate Alienware AlienFX support into an application. The reset, light, update loop could be performed alongside a regular application interval. As well, calls to LFX_Light can be tied to events which are queued up and submitted to the hardware upon a call to LFX_Update. See the function reference section within this document for more details on these functions and their parameters.

4. Application Development Guidelines

The following guidelines may be useful for application developers who wish to understand more about the inner workings of the Alienware AlienFX system. While this information is provided for reference, it may be of little use or interest to those using the simplified, implicit programming model.

4.1 Clarifications and Deprecated functions

The following are known issues, and their resolutions or workarounds if available.

- **LFX_UpdateDefault:** This function has been deprecated for all the new platforms. Can be done only through FX module of AWCC
- **LFX_SetTiming:** This function has been deprecated for all the new platforms. Can be done only through FX module of AWCC
- **AlienFX Configurator** has been deprecated and removed from SDK
- **SDK managed libraries** have been deprecated and removed from SDK

4.2 Location and Positioning Semantics

Alienware AlienFX uses logical equivalents, i.e., Front-Lower-Left, when describing physical locations.

4.3 Multithreading and command timing

Alienware AlienFX incorporates critical sections in every library function exported.

Additionally, the Alienware AlienFX hardware abstraction layer incorporates a command queue and a command handler thread, which masks hardware latencies. These hardware latencies vary by device, so the command handler monitors the performance of the hardware and drops updates that take too long to maintain a tight window of software request to physical change. Currently this window is set to 100 milliseconds, meaning if an update is in the command queue for more than 100 milliseconds, it will be dropped to allow the hardware to catch up to software. This command buffering approach ensures that the core Alienware AlienFX functions (LFX_Reset, LFX_Light, LFX_SetLightColor, LFX_Update) do not block the main application thread. The disadvantage is that there is no guarantee that a command in flight will affect the hardware.

4.4 Plug and Play capabilities

Alienware AlienFX will attend at real time to the arrival of new devices. If an application wishes to add new Alienware AlienFX devices "on the fly" this can be done by calling LFX_GetNumDevices(..) in order to get new enumerated devices. Note that if you are referring "ALL" devices when setting colors through the different functions there is no need to re-enumerate (call LFX_GetNumDevices(..))

4.5 While (0)

Though obvious, Alienware AlienFX provides return values that allow good decision-making when deciding if and how to update the system. One of the easiest performance optimizations to make is to simply not submit any updates to locations or lights which fail. If there are no devices connected at all, skip the updates, perhaps even releasing the system, until a device arrives which may be Alienware AlienFX enabled.

5. Function Reference

5.1 Overview

All unmanaged Alienware AlienFX library functions are exported as C-language, and all return LFX_RESULT (type defined as unsigned integer). The following sections outline the minimum mandatory set of functions available in a compliant Alienware AlienFX library.

5.2 LFX_Initialize

This function initializes the Alienware AlienFX system. It must be called prior to calling other library functions. If this function is not called, the system will not be initialized, and the other library functions will return LFX_ERROR_NOINIT or LFX_FAILURE.

Syntax:

LFX_RESULT LFX_Initialize();

Parameters:

None

Inputs:

None

Outputs:

None

Returns:

LFX_SUCCESS	if the system is successfully initialized, or was already initialized
LFX_FAILURE	if the initialization fails
LFX_ERROR_NODEVS	if the system is initialized, but no devices are available

5.3 LFX_Release

This function releases the Alienware AlienFX system, freeing memory and restores the system to its initial state. It may be called when the system is no longer needed.

Plug-and-Play Note: An application may choose to release the system and reinitialize it again, in response to a device arrival notification. Doing so will account for new devices added while the application is running.

Syntax:

LFX_RESULT LFX_Release();

Parameters:

None

Inputs:

None

Outputs:

None

Returns:

LFX_SUCCESS	if the system is successfully released

5.4 LFX_Reset

This function sets all lights in the Alienware AlienFX system to 'off' or uncolored state. It must be noted that the change(s) to the physical light(s) does not occur immediately. The change(s) occurs only after a call to the LFX_Update function. For example, to disable all the lights, call LFX_Reset followed by LFX_Update.

Syntax:

LFX_RESULT LFX_Reset();

Parameters:

None

Inputs:

None

Outputs:

None

Returns:

LFX_ERROR_NOINIT	if the system is not initialized
LFX_ERROR_NODEVS	if there are no devices available to reset
LFX_SUCCESS	if the reset is successful

5.5 LFX_Update

This function updates the Alienware AlienFX system by submitting any state changes (since the last call to LFX_Reset) to the hardware.

Syntax:

LFX_RESULT LFX_Update();

Parameters:

None

Inputs:

None

Outputs:

None

Returns:

LFX_ERROR_NOINIT	if the system is not initialized
LFX_ERROR_NODEVS	if there are no devices available to reset
LFX_FAILURE	if some other error occurred
LFX_SUCCESS	if the update is successful

5.6 LFX_UpdateDefault

This function updates the Alienware AlienFX system by submitting any state changes (since the last call to LFX_Reset) to the hardware, as well as setting the appropriate flags to enable the updated state to be the new power-on default state.

NOTE: This function has been deprecated

Syntax:

```
LFX_RESULT LFX_UpdateDefault();
```

Parameters:

None

Inputs:

None

Outputs:

None

Returns:

LFX_ERROR_NOINIT	if the system is not initialized
LFX_ERROR_NODEVS	if there are no devices available to reset
LFX_FAILURE	if some other error occurred
LFX_SUCCESS	if the function is successful

5.7 LFX_GetNumDevices

This function gets the number of Alienware AlienFX devices attached to the system.

Syntax:

```
LFX_RESULT LFX_GetNumDevices (unsigned int* const numDevices);
```

Parameters:

numDevices	integer to be populated with the number of devices

Inputs:

None

Outputs:

Populates an unsigned integer with the current number of devices attached

Returns:

LFX_ERROR_NOINIT	if the system is not initialized
LFX_ERROR_NODEVS	if there are no devices available to reset
LFX_FAILURE	if some other error occurred
LFX_SUCCESS	if the function is successful

5.8 LFX_GetDeviceDescription

This function gets the description and type of a device attached to the system.

Syntax:

```
LFX_RESULT LFX_GetDeviceDescription(const unsigned int devIndex,
      char* const devDesc, const unsigned int devDescSize,
      unsigned char* const devType);
```

Parameters:

`devIndex`	index to the target device
`devDesc`	character array to be populated with the description of the target device
`devDescSize`	size of the character array provided in `devDesc`
`devType`	unsigned short to be populated with the device type

Inputs:

Accepts an index to the device

Outputs:

Populates a character array with the indexed device's description
Populates an unsigned short with the device type

Returns:

LFX_ERROR_NOINIT	if the system is not initialized
LFX_ERROR_NODEVS	if there are no devices at the index
LFX_ERROR_BUFFSIZE	if the character array provided was too small
LFX_FAILURE	if some other error occurred
LFX_SUCCESS	If the function is successful

5.9 LFX_GetNumLights

This function gets the number of Alienware AlienFX lights attached to a device in the system.

Prototype:

```
LFX_RESULT LFX_GetNumLights(const unsigned int devIndex,
unsigned int* const numLights);
```

Parameters:

`devIndex`	Index to the device
`numLights`	Unsigned integer to be populated with the number of lights at the device index

Inputs:

Accepts an index to the device

Outputs:

Populates an unsigned integer with the current number of lights attached to the device at the given index

Returns:

LFX_ERROR_NOINIT	if the system is not initialized
LFX_ERROR_NODEVS	if there are no devices at the index
LFX_ERROR_NOLIGHTS	if no lights are available at the device index provided
LFX_FAILURE	if some other error occurred
LFX_SUCCESS	if the function is successful

5.10 LFX_GetLightDescription

This function gets the description of a light attached to the system.

Syntax:

```
LFX_RESULT LFX_GetLightDescription(const unsigned int devIndex,
      const unsigned int lightIndex, char* const lightDesc,
      const unsigned int lightDescSize);
```

Parameters:

`devIndex`	Index to the target device
`lightIndex`	Index to the target light
`lightDesc`	Character array to be populated with the description of the target light
`lightDescSize`	Size of the character array provided in `lightDesc`

Inputs:

Accepts an index to the device
Accepts an index to the light

Outputs:

Populates a character array with the indexed light's description

Returns:

LFX_ERROR_NOINIT	if the system is not initialized
LFX_ERROR_NODEVS	if there are no devices at the index
LFX_ERROR_NOLIGHTS	if no lights are available at the device index provided
LFX_ERROR_BUFFSIZE	if the character array provided was too small
LFX_FAILURE	if some other error occurred
LFX_SUCCESS	if the function is successful

5.11 LFX_GetLightLocation

This function gets the location of a light attached to the system.

Syntax:

```
LFX_RESULT LFX_GetLightLocation(const unsigned int devIndex,
      const unsigned int lightIndex, PLFX_POSITION const lightLoc);
```

Parameters:

`devIndex`	Index to the target device
`lightIndex`	Index to the target light
`lightLoc`	Pointer to an LFX_POSITION structure to be populated with the light location

Inputs:

Accepts an index to the device
Accepts an index to the light

Outputs:

Populates an LFX_POSITION structure with the indexed light's location

Returns:

LFX_ERROR_NOINIT	if the system is not initialized
LFX_ERROR_NODEVS	if there are no devices at the index
LFX_ERROR_NOLIGHTS	if no lights are available at the device index provided
LFX_FAILURE	if some other error occurred
LFX_SUCCESS	if the function is successful

5.12 LFX_GetLightColor

This function gets the color of a light attached to the system. This function provides the current color stored in the active state. It does not necessarily represent the color of the physical light. To ensure that the returned value represents the state of the physical light, call this function immediately after a call to LFX_Update.

Syntax:

```
LFX_RESULT LFX_GetLightColor(const unsigned int devIndex,
      const unsigned int lightIndex, PLFX_COLOR const lightCol);
```

Parameters:

`devIndex`	Index to the target device
`lightIndex`	Index to the target light
`lightCol`	Pointer to an LFX_COLOR structure to be populated with the light location

Inputs:

Accepts an index to the device
Accepts an index to the light

Outputs:

Populates an LFX_COLOR structure with the indexed light's color

Returns:

LFX_ERROR_NOINIT	if the system is not initialized
LFX_ERROR_NODEVS	if there are no devices at the index
LFX_ERROR_NOLIGHTS	if no lights are available at the device index provided
LFX_FAILURE	if some other error occurred
LFX_SUCCESS	if the function is successful

5.13 LFX_SetLightColor

This function submits a light command into the command queue, which sets the current color of a light to the provided color value. This function changes the current color stored in active state since the last reset. It does not immediately update the physical light settings, instead requires a call to LFX_Update.

Syntax:

```
LFX_RESULT LFX_SetLightColor(const unsigned int devIndex,
      const unsigned int lightIndex, const PLFX_COLOR lightCol);
```

Parameters:

`devIndex`	Index to the target device
`lightIndex`	Index to the target light
`lightCol`	Pointer to an LFX_COLOR structure to be populated with the light location

Inputs:

Accepts an index to the device
Accepts an index to the light
Accepts a pointer to an LFX_COLOR structure

Outputs:

None

Returns:

LFX_ERROR_NOINIT	if the system is not initialized
LFX_ERROR_NODEVS	if there are no devices at the index
LFX_FAILURE	if some other error occurred
LFX_SUCCESS	if the function is successful

5.14 LFX_Light

This function submits a light command into the command queue, which sets the current color of any light within the provided location mask to the provided color setting. Similar to LFX_SetLightColor, these settings are changed in the active state and must be submitted with a call to LFX_Update. Location mask is a 32-bit field, where each of the first 27 bits represents a zone in the virtual cube representing the system. The color is packed into a 32-bit value as ARGB, with the alpha value corresponding to brightness.

Syntax:

```
LFX_RESULT LFX_Light(const unsigned int locationMask,
      const unsigned int colorVal);
```

Parameters:

`locationMask`	32-bit location mask. See the defined values in the header file (LFXDecl.h)
`colorVal`	32-bit color value

Inputs:

Accepts a 32-bit location mask.
Accepts a 32-bit packed color value

Outputs:

None

Returns:

LFX_ERROR_NOINIT	if the system is not initialized
LFX_ERROR_NOLIGHTS	if no lights were found at the location mask specified
LFX_FAILURE	if some other error occurred
LFX_SUCCESS	if the function is successful

5.15 LFX_SetLightActionColor

This function sets the primary color and an action type to a light. It changes the current color and action type stored in the active state since the last LFX_Reset() call. It does NOT immediately update the physical light settings, but instead requires a call to LFX_Update(). If the action type is a morph, then the secondary color for the action is black.

Syntax:

```
LFX_RESULT LFX_SetLightActionColor(const unsigned int devIndex,
      const unsigned int lightIndex, const unsigned int actionType,
      const PLFX_COLOR primaryColor);
```

Parameters:

`devIndex`	Index to the target device
`lightIndex`	Index to the target light
`actionType`	Action type
`primaryColor`	Pointer to an LFX_COLOR structure with the desired color

Inputs:

Accepts an index to the device, an index to the light, an action type (LFX_ACTION_MORPH, LFX_ACTION_PULSE, LFX_ACTION_COLOR), and a new primary LFX_COLOR value

Outputs:

None

Returns:

LFX_ERROR_NOINIT	if the system is not initialized
LFX_FAILURE	if some other error occurred
LFX_SUCCESS	if the function is successful

5.16 LFX_SetLightActionColorEx

This function sets the primary and secondary colors and an action type to a light. It changes the current color and action type stored in the active state since the last LFX_Reset() call. It does NOT immediately update the physical light settings, but instead requires a call to LFX_Update(). If the action type is not a morph, then the secondary color is ignored.

Syntax:

```
LFX_RESULT LFX_SetLightActionColorEx(const unsigned int devIndex,
      const unsigned int lightIndex, const unsigned int actionType,
      const PLFX_COLOR primaryColor, const PLFX_COLOR
secondaryColor);
```

Parameters:

`devIndex`	Index to the target device
`lightIndex`	Index to the target light
`actionType`	Action type
`primaryColor`	Pointer to an LFX_COLOR structure with the desired color
`secondaryColor`	Pointer to an LFX_COLOR structure with the desired secondary color

Inputs:

Accepts an index to the device, an index to the light, an action type (LFX_ACTION_MORPH, LFX_ACTION_PULSE, LFX_ACTION_COLOR), and two LFX_COLOR values

Outputs:

None

Returns:

LFX_ERROR_NOINIT	if the system is not initialized
LFX_FAILURE	if some other error occurred
LFX_SUCCESS	if the function is successful

5.17 LFX_ActionColor

This function sets the primary color and an action type for any devices with lights in a location. It changes the current primary color and action type stored in the active state since the last LFX_Reset() call. It does NOT immediately update the physical light settings, but instead requires a call to LFX_Update(). If the action type is a morph, then the secondary color for the action is black. Location mask is a 32-bit field, where each of the first 27 bits represents a zone in the virtual cube representing the system. The color is packed into a 32-bit value as ARGB, with the alpha value corresponding to brightness.

Syntax:

```
LFX_RESULT LFX_ActionColor(const unsigned int locationMask,
      const unsigned actionType, const unsigned int primaryColor);
```

Parameters:

`locationMask`	32-bit location mask. See the defined values in the header file (LFXDecl.h)
`actionType`	Action type
`primaryColor`	32-bit color value

Inputs:

Accepts a 32-bit location mask
Accepts a 32-bit packed color value

Outputs:

None

Returns:

LFX_ERROR_NOINIT	if the system is not initialized
LFX_ERROR_NOLIGHTS	if no lights were found at the location mask specified.
LFX_FAILURE	if some other error occurred
LFX_SUCCESS	if the function is successful

5.18 LFX_ActionColorEx

This function sets the primary and secondary color and an action type for any devices with lights in a location. It changes the current primary and secondary color and action type stored in the active state since the last LFX_Reset() call. It does NOT immediately update the physical light settings, but instead requires a call to LFX_Update. If the action type is not a morph, then the secondary color is ignored. Location mask is a 32-bit field, where each of the first 27 bits represents a zone in the virtual cube representing the system. The color is packed into a 32-bit value as ARGB, with the alpha value corresponding to brightness.

Syntax:

```
LFX_RESULT LFX_ActionColorEx(const unsigned int locationMask,
      const unsigned actionType, const unsigned int primaryColor,
      const unsigned int secondaryColor);
```

Parameters:

`locationMask`	32-bit location mask. See the defined values in the header file (LFXDecl.h)
`actionType`	Action type
`primaryColor`	32-bit primary color value
`secondaryColor`	32-bit secondary color value

Inputs:

Accepts a 32-bit location mask
Accepts a 32-bit packed color value

Outputs:

None

Returns:

LFX_ERROR_NOINIT	if the system is not initialized
LFX_ERROR_NOLIGHTS	if no lights were found at the location mask specified.
LFX_FAILURE	if some other error occurred
LFX_SUCCESS	if the function is successful

5.19 LFX_SetTiming

This function changes the current tempo or timing to be used for the next actions. It does NOT immediately update the physical light settings, but instead requires a call to LFX_Update(). The timing is a value between minimum and maximum tempo allowed for the each device. If a value is lower than minimum or greater than maximum is entered, then the value is readjusted to those extremes.

NOTE: This function has been deprecated

Syntax:

```
LFX_RESULT LFX_SetTiming(const int timing);
```

Parameters:

`timing`	32-bit timing value in milliseconds

Inputs:
Accepts a 32-bit timing value

Outputs:
None

Returns:

LFX_ERROR_NOINIT	if the system is not initialized
LFX_FAILURE	if some other error occurred
LFX_SUCCESS	if the function is successful

5.20 LFX_GetVersion

This function gets the version of the SDK installed in the system.

Syntax:

```
LFX_RESULT LFX_GetVersion(char* const version,
      const unsigned int versionSize);
```

Parameters:

`version`	character array to be populated with the version
`versionSize`	size of the character array provided in `version`

Inputs:
Accepts the bugger and buffer size

Outputs:
Populates a character array with the SDK version

Returns:

LFX_ERROR_NOINIT	if the system is not initialized
LFX_ERROR_BUFFSIZE	if the character array provided was too small
LFX_FAILURE	if some other error occurred
LFX_SUCCESS	If the function is successful

NOTE: If the `LFX_GetVersion` function is not found then the SDK version will be 1.0 or 2.0