Alienware AlienFX® 1.0 Software Development Kit

Contents

Revision History

Version 1.0 – March 24, 2009 – Alpha release

1.0 Introduction

1.1 Identification

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

1.2 Purpose

The purpose of this document is to provide a detailed, programmatic outline of the
available features and functions provided by the Alienware AlienFX® SKD 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® system , an application may configure colored light devices 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.

2.0 Getting Started

2.1 Contents

This software development kit (SDK) contains the redistributable library, example application samples, as well as this document. Upon extraction or installation of the SDK, these files are located in subdirectories of the installation directory (i.e. 'C:\Program Files\Dell\Alienware AlienFX® 2.0 SDK):

- **Documentation:** \SDK\
- **Headers:** \SDK\unmanage \include\
- **Dynamic Link Library:** \System32 \LightFX.dll; \SDK\LightFX.dll
  - Used for implicit or explicit dynamic linking
  - Multiple library versions are available, depending on operating system
- **Samples:** SDK\
  - Multiple samples are available
  - See below for more details

2.2 System Requirements

Software Requirements:

- Windows XP or Windows Vista (any 32-bit variant and 64-bit)
- Visual C++ 2005 SP1
- Note: Earlier versions, and other development environments and compilers may be sufficient, but have not been tested during the development of this SDK

Hardware & Software Requirements:

- Application development with the library does not require any special hardware, although Alienware AlienFX® SDK compliant hardware is useful for testing and debug
- Test/Debug - one or more of the following:
  - LightFX SDK 1.x compliant hardware
  - LightFX SDK 2.0 compliant hardware

2.3 Library Linking

The LightFX library may be linked dynamically either implicitly or explicitly. Developers unfamiliar with the following options are encouraged to review the various documents on Microsoft's Developer Network regarding 3rd party library usage.

Static

Static linking has been removed, along with the static libraries themselves, for the 1.0 release of the software development kit. Static linking, although easier to link, requires recompilation of the executable incorporating the library whenever the library changes. Dynamic linking ensures that future updates to the library can be automatically supported by the application, so long as the DLL is updated on the target platform running the application.

Explicit Dynamic

When linking dynamically, explicit linking is another option. Helpful definitions of function names and function pointers have been included in the library header LFX2.h for this purpose. Rather than use the import library, the library is loaded with a call to LoadLibrary along with calls

to GetProcAddress for any desired functions of the library. Explicit linking may be preferred for applications which require the flexibility of a separate DLL, but prefer to not redistribute the library. By linking explicitly, the application will be able to launch without the presence of the library, and can easily handle the case where the library is unavailable.

2.4 How Alienware AlienFX® SDK Works

Alienware AlienFX® is, in a nutshell, an abstraction and translation library. 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 finds 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 finding all the hardware, Alienware AlienFX® waits for requests from the application via 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 in order to set the color value. LFX_Light is preferred by most developers due to its ease of use.

2.5 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:

```
#include "LFX2.h"
int _tmain(int argc, _TCHAR* argv[])
{
LFX_Initialize();
// Reset the state machine and await light settings
LFX_Reset();
// Set all lights to blue
LFX_Light(LFX_ALL, LFX_BLUE | LFX_FULL_BRIGHTNESS);
// Update the state machine,
// which causes the physical color change
LFX_Update();
// Cleanup and detach from the system
LFX_Release();
return 0;
}
```

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 such as a screen update. 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.

3.0 Samples

3.1 Overview

Several samples are provided, demonstrating various ways of linking to and using the library. They also explore the various programming models outlined in Section 4.2.

3.2 Sample 1

The first sample links explicitly to the DLL version of the library, using the LoadLibrary and GetProcAddress functions. Explicit linking offers developers a great way to overcome the dependency on a DLL in order to load the application. For libraries like LightFX.dll, where the functionality provided may not be essential to the application, explicit linking provides advantages over implicit linking. This linking method is highly encouraged, as it enables transparent updating of the DLL without negatively impacting the application or even requiring a recompilation. Further, with Windows ability to remap dynamic link libraries to force the load of a local directory version, there is little to fear in this approach. The function prototypes are locked, and will be available, even if only for legacy support, in all future releases of the Alienware AlienFX library. As new device types and hardware are integrated into the library, all of these will be made accessible via the standard set of functions available in this SDK. The sample loops on the LFX_Reset, LFX_Light and LFX_Update functions, using the implicit programming model.

4.0 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 Known Issues

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

UpdateDefault on Certain Platforms
Certain platforms do not support preservation of power-on-default settings using the LFX_UpdateDefault function. For these platforms (two as – XPS 630 and XPS 730), the behavior of LFX_UpdateDefault is the same as LFX_Update, with the additional feature that the last LFX_UpdateDefault setting will be set when LFX_Release is called.

4.2 Location and Positioning Semantics

Alienware AlienFX® uses several semantics when describing physical locations, or their logical equivalents.

Bounds
- The physical bounds, in centimeters, of any given device enclosure such as a PC chassis
- Bounds are relative to, and measured by, the distance from an anchor point at the lower, left, rear corner of the device enclosure
- May be encoded into an LFX_POSITION structure

Position
- Or, "Location"
- Position is the physical position, in centimeters, of any given light relative to the anchor
- point at the lower, left, rear corner of the device's bounding box
- Position can be obtained through the LFX_GetLightLocation function
- May be encoded into an LFX_POSITION structure

Location Mask
- Similar to Position or Location, albeit encoded into a mask
- A 32-bit mask that denotes one or more light positions in terms of the device's bounds
- There are 27 bits for each smaller cube within this bounding box
- The bounding box is divided evenly into three sections per axis
- See LFXDecl.h file for all of the various mask bits and their aliases

4.3 Multithreading

Alienware AlienFX® incorporates critical sections in every library function exported. Furthermore, it is designed such that each process attaching to the system obtains a shared copy of the state machine (including the backup copy of the state machine for restoration).
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

implementation, so the command handler monitors the performance of the hardware and drops updates that take too long, in order to maintain a tight window of software request to physical change. Currently this window is set to 250 milliseconds. If an update sitting in the command queue is more than 250 milliseconds old, it will be dropped in order 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

Alienware AlienFX® will automatically attend 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 enumerateed devices. Note that If you are referring "ALL" devices when setting colors through the different functions there is not 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.

4.6 C Runtime Library Settings

Note: This section is deprecated with the removal of the static library.
When building your own applications that incorporate the Alienware AlienFX® library, it is important to note
the proper C Runtime Library settings in order to match those of the library. Future versions of this SDK may include multiple builds of the library to support various CRT options.

Debug Configurations
For debug configurations, within the Project Settings C/C++ Code Generation section, the proper runtime library to use is "Multi-threaded Debug DLL", or the /MDd compiler flag for Microsoft compilers. Additionally, in the Linker option, the default LIBCMTD library should be ignored with the linker option /NODEFAULTLIB:LIBCMTD in the Command Line options. This addresses LNK2005 link errors when adding the Alienware AlienFX® library to a project in a debug configuration.

Release Configurations
Release configurations of a project linking to the release build of the Alienware AlienFX® library should not specify /NODEFAULTLIB:LIBCMT. Additionally, release configurations should use the "Multi-threaded" runtime library in the C/C++ Code Generation section (corresponding to the /MT compiler flag).

5.0 Function Reference

5.1 Overview

All Alienware AlienFX® library functions are exported a 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

Prototype:
LFX_RESULT LFX_Initialize();
Description:
This **blocking function** initializes the Alienware AlienFX® system. It must be called prior to other library calls made. 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.
Inputs:
None
Outputs:
None
Returns:
LFX_SUCCESS if the system is successfully initialized, or was already initialized
LFX_FAILURE if initialization fails
LFX_ERROR_NODEVS if the system is initialized, but no devices are available

5.3 LFX_Release

Prototype:
LFX_RESULT LFX_Release();
Description:
This **blocking function** releases the Alienware AlienFX® system, freeing memory and restoring the state machine. It may be called when the system is no longer needed. If this function is not called, release will still occur when the process which initialized the system detaches from the library.
Debugging Note:
Although this function is optional, and release will occur on process detach anyway, some debuggers may cite memory leak(s), due to their inability to view intraprocess deallocation of memory from a thread which was not the allocator. In these cases, this function is useful to preclude these false positive memory leak detections.
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.
Inputs:
None
Outputs:
None
Returns:

LFX_SUCCESS

5.4 LFX_Reset

Prototype:
LFX_RESULT LFX_Reset();
Description:
This **non-blocking function** sets all lights in the Alienware AlienFX® system to their 'off' or uncolored state. Note that the change(s) to the physical light(s) do not occur immediately. These changes occur only after a call to LFX_Update. For example, to disable all lights, call LFX_Reset followed by LFX_Update.
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 otherwise

5.5 LFX_Update

Prototype:
LFX_RESULT LFX_Update();
Description:
This **non-blocking function** updates the Alienware AlienFX® system by submitting any state changes (since the last call to LFX_Reset) to hardware.
Inputs:
None
Outputs:
None
Returns:
LFX_ERROR_NOINIT if the system is not initialized
LFX_ERROR_NODEVS if there are no devices available to update
LFX_FAILURE if some other error occurred
LFX_SUCCESS otherwise

5.6 LFX_UpdateDefault

Prototype:
LFX_RESULT LFX_UpdateDefault();
Description:
This **non-blocking function** updates the Alienware AlienFX® system by submitting any state changes (since the last call to LFX_Reset) to hardware, as well as setting the appropriate flags to enable the updated state to be the new power-on default.
Note:
Not all devices support this functionality.
Inputs:
None
Outputs:
None

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

5.7 LFX_GetNumDevices

Prototype:
LFX_RESULT LFX_GetNumDevices(
unsigned int* const numDevices);
Description:
This **blocking function** gets the number of Alienware AlienFX® devices attached to the system.
Note:
The parameter passed is unaltered if no devices are available.
Parameter 1:
Integer to be populated with the number of devices
Inputs:
None
Outputs:
Populates an unsigned integer with the current number of attached devices
Returns:
LFX_ERROR_NOINIT if the system is not initialized
LFX_ERROR_NODEVS if there are no devices available
LFX_FAILURE if some other error occurred
LFX_SUCCESS otherwise

5.8 LFX_GetDeviceDescription

Prototype:
LFX_RESULT LFX_GetDeviceDescription(
const unsigned int devIndex,
char* const devDesc,
const unsigned int devDescSize,
unsigned char* const devType);
Description:
This **blocking function** gets the description and type of a device attached to the system.
Parameter 1:
Index to the target device
Parameter 2:
Character array to be populated with the description of the target device
Parameter 3:
Size of the character array provided in parameter 2
Parameter 4:
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 otherwise

5.9 LFX_GetNumLights

Prototype:
LFX_RESULT LFX_GetNumLights(
const unsigned int devIndex,
unsigned int* const numLights);
Description:
This **blocking function** gets the number of Alienware AlienFX lights attached to a device in the system.
Note:
The parameter passed is unaltered if no devices are available.
Parameter 1:
Index to the device
Parameter 2:
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 attached lights 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 otherwise

5.10 LFX_GetLightDescription

Prototype:
LFX_RESULT LFX_GetLightDescription(
const unsigned int devIndex,
const unsigned int lightIndex,
char* const lightDesc,
const unsigned int lightDescSize);
Description:
This **blocking function** gets the description of a light attached to the system.
Parameter 1:
Index to the target device
Parameter 2:
Index to the target light
Parameter 3:
Character array to be populated with the description of the target light
Parameter 4:

Size of the character array provided in parameter 3
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 is no lights are available at the index
LFX_ERROR_BUFFSIZE if the character array provided was too small
LFX_FAILURE if some other error occurred
LFX_SUCCESS otherwise

5.11 LFX_GetLightLocation

Prototype:
LFX_RESULT LFX_GetLightLocation(
const unsigned int devIndex,
const unsigned int lightIndex,
PLFX_POSITION const lightLoc);
Description:
This **blocking function** gets the location of a light attached to the system.
Parameter 1:
Index to the target device
Parameter 2:
Index to the target light
Parameter 3:
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 is no lights are available at the index
LFX_FAILURE if some other error occurred
LFX_SUCCESS otherwise

5.12 LFX_GetLightColor

Prototype:
LFX_RESULT LFX_GetLightColor(
const unsigned int devIndex,
const unsigned int lightIndex,
PLFX_COLOR const lightCol);
Description:
This **blocking 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 and before the next call to LFX_Reset. Also note that some hardware has limitations such as index color, which preclude obtaining the exact RGB representation.
Parameter 1:
Index to the target device
Parameter 2:
Index to the target light
Parameter 3:
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 is no lights are available at the index
LFX_FAILURE if some other error occurred
LFX_SUCCESS otherwise

5.13 LFX_SetLightColor

Prototype:
LFX_RESULT LFX_SetLightColor(
const unsigned int devIndex,
const unsigned int lightIndex,
const PLFX_COLOR lightCol);
Description:
This **non-blocking 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 requiring a call to LFX_Update.
Parameter 1:
Index to the target device
Parameter 2:
Index to the target light
Parameter 3:
Pointer to an LFX_COLOR structure with the desired color
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 otherwise

5.14 LFX_Light

Prototype:
LFX_RESULT LFX_Light(
const unsigned int locationMask,
const unsigned int colorVal);
Description:
This **non-blocking function** submits a light command into the command queue, which sets the current color of any lights within the provided location mask to the provided color setting. Similar to LFX_SetLightColor, 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. Color is packed into a 32-bit value as ARGB, with the alpha value corresponding to brightness.
Parameter 1:
Index to the target device
Parameter 2:
Index to the target light
Parameter 3:
Pointer to an LFX_COLOR structure with the desired color
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_FAILURE if some other error occurred

LFX_SUCCESS otherwise