Creative Technology Ltd

Creative Aurora SDK

29 September 2017
Revision 0.90

Table Of Content

1 Revision history

Revision	Date	Author	Changes
0.50	20 July 2017	Chan Luen Kai	1. Initial draft release for review.
0.90	29 September 2017	Chan Luen Kai	2. First release candidate.

2 Introduction

Many Creative products are equipped with the Creative Aurora Reactive lighting system which gives you excellent ambient lighting to accompany your audio listening and game playing experiences with their fully customizable 16 million colors LED lighting. You may customize colors and patterns to match your other RGB-enabled peripherals or fully integrate with other Creative Aurora Reactive lighting system-enabled products.

The Creative Aurora SDK allows third party applications to control the LED lighting of the Creative Aurora Reactive Lighting System.

The purpose of this document is to describe the application programming interface (API) of the Creative Aurora Reactive Lighting System.

2.1 Supported Products

The Creative Aurora Reactive Lighting System is supported by the following Creative products:

- USB Keyboard
  - Sound BlasterX Vanguard K08 (US English)
  - Sound BlasterX Vanguard K08 (German)
  - Sound BlasterX Vanguard K08 (Nordic)
- USB Mouse
  - Sound BlasterX Siege M04
- USB Audio Device
  - Sound BlasterX Katana
  - Sound BlasterX Kratos S5
- PCIe Audio Device
  - Sound BlasterX AE-5

Newer Creative products will be added to the above supported product list when they are available.

2.1.1 Is device driver needed?

For the Creative Aurora SDK to work with Sound BlasterX AE-5, the device driver for Sound BlasterX AE-5 must be installed in the system.

For the rest of the supported products mentioned earlier, no device driver need to be installed. You just have to connect the device to the system and any client software that is developed using the Creative Aurora SDK will just work.

2.2 Supported Operating Systems

In general, the SDK works under Windows 7 and above.

Each of the supported products may have its own specific requirements of the operating systems. Please refer to the respective product specification for details.

3 Content of the Creative Aurora SDK

The Creative Aurora SDK contains the following folders and files:

- The **Doc** folder contains this documentation file.
- The **Include** folder contains the C/C++ header files.
- The **Redist** folder contains the DLL libraries that are to be distributed with your applications that are developed with this SDK. Both x86 and x64 DLL libraries are available.
    - The **x86** sub-folder contains the redistributable 32 bit DLL libraries.
    - The **x64** sub-folder contains the redistributable 64 bit DLL libraries.
- The **Sample** folder contains the source code files of a sample application. For the current SDK, this sample application is developed with Visual Studio 2013. **You will need to copy the DLL libraries from the Redist folder to the folder of the sample application EXE file in order to run the EXE of the sample application.**
- The **Demo** folder (if present in the SDK) contains the demo app that was built from the Sample folder.

Note that in order to reduce the package size of this SDK, the Demo folder may be removed from the SDK package. If the Demo folder is not present in the SDK, you may build the demo app yourself from the Sample folder.

4 Programming with the Creative Aurora SDK

The Creative Aurora SDK allows a client software to program the LED of different supported devices in a unified way. **Essentially, the same programming works for all different supported devices.** This unified way of programming different supported devices greatly simplifies and reduces the amount of coding needed in the client software to support different devices.

4.1 Redistributable DLL Libraries

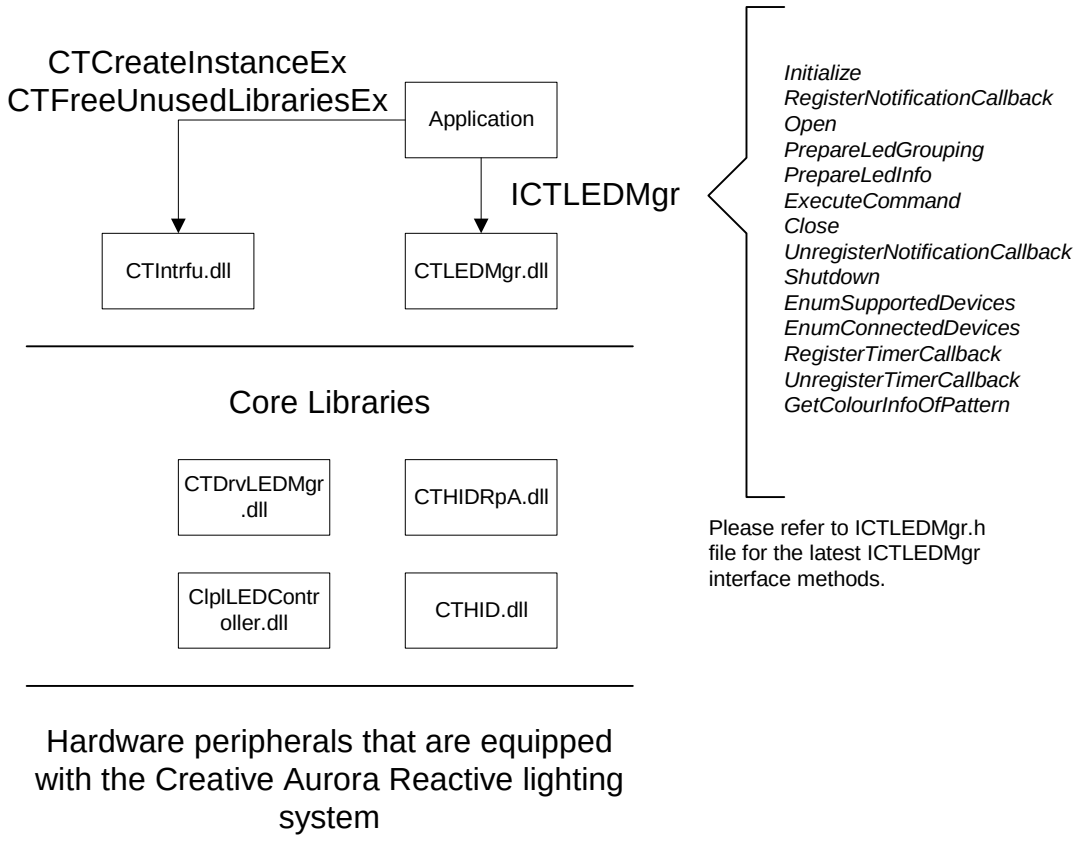
The following are the redistributable DLL libraries that are from the Creative Aurora SDK:

- CTIntrfu.dll
  - This is the Creative Interface Library
- CTLEDMgr.dll
  - This is the Creative LED Manager Library
- CTHIDRpA.dll, CTHID.dll, CTDrvLEDMgr.dll, ClplLEDController.dll
  - These are the core DLL libraries of the Creative Aurora Reactive lighting system.

To run your application that is developed with the Creative Aurora SDK, you will need the DLL libraries from the Redist folder of the SDK. These DLL libraries must reside together in the same directory, which is usually (but may not be necessary) in the same directory as the application module.

4.2 High Level Architecture

The following is a high level architecture diagram of the software components of the Creative Aurora Reactive lighting system:



4.3 Programming Overview

The following procedure is an overview of how the client application shall call the Creative Aurora SDK:

- Create the ICTLEDMgr Interface.
- Call the methods of ICTLEDMgr.
- Release ICTLEDMgr.
- Free Libraries.

4.3.1 Create the ICTLEDMgr Interface

Call CTCreateInstanceEx (which is an exported function of CTIntrfu.dll) to get the required ICTLEDMgr interface to the Creative LED Manager Library (CTLEDMgr.dll).

4.3.2 Call the methods of ICTLEDMgr

After obtaining the ICTLEDMgr interface, you can call its various interface methods to perform the desired LED operations.

4.3.3 Release ICTLEDMgr

Release the ICTLEDMgr interface when you no longer need it.

4.3.4 Free Libraries

After releasing the ICTLEDMgr interface, call CTFreeUnusedLibrariesEx (which is an exported function of CTIntrfu.dll) to unload any DLL libraries that were loaded earlier (but are no longer in use).

4.4 Functional Features

Using the Creative Aurora SDK, a client software will be able to program the pattern and colour of each and every LED of the devices.

4.4.1 Support for Firmware-based Lighting Effects

The SDK allows the client software to configure lighting effects that are driven by the firmware. Basically, once the client software has configured the device for a specific lighting effect that the firmware can support, the firmware will then be driving the lighting effects without the intervention of the client software.

4.4.2 Support for Host-based Lighting Effects

The SDK allows the client software to configure lighting effects that are driven by the host computer. Basically, the client software has to keep running to continuously drive the lighting effects in whatever way it wants.

4.4.3 Special Support for Choreography of Lighting Effects

In addition to allowing the client software to program the lighting effects, the SDK is also designed with a built-in mechanism to support choreography of lighting effects across devices.

5 Supported Devices and Connected Devices

The SDK allows the client software to enumerate the list of devices that are supported by the SDK and also the list of supported devices that are currently connected to the host computer system.

5.1 Enumerate Supported Devices

The interface method ICTLEDMgr::EnumSupportedDevices allows the client software to enumerate the list of devices that are supported by the SDK. For each device enumerated, the interface method returns the following device information to the client software:

- Vendor ID
- Product ID
- Device Friendly Name
- Device Category

5.1.1 Device Category

The following are the possible device categories:

- Keyboard
  The SDK defines this as CTDEVICECATEGORY_BitwiseMask_Keyboard.
- Mouse
  The SDK defines this as CTDEVICECATEGORY_BitwiseMask_Mouse.
- Speakers
  The SDK defines this as CTDEVICECATEGORY_BitwiseMask_Speakers.
- Internal Audio Device
  The SDK defines this as CTDEVICECATEGORY_BitwiseMask_InternalAudioDevice.
- External Audio Device
  The SDK defines this as CTDEVICECATEGORY_BitwiseMask_ExternalAudioDevice.

Note that the SDK is designed to cater for the case whereby a device may have more than one category.

5.1.2 Sample Code: Enumerate Supported Devices

The following code fragment shows how to enumerate for the list of devices which are supported by the SDK.

```
HRESULT hr;
DWORD dwFlag = 0;
TCHAR szOverallMsgBuf[2048];
size_t sizeOverallMsgBuf = sizeof(szOverallMsgBuf) / sizeof(szOverallMsgBuf[0]);
hr = StringCchCopyEx(szOverallMsgBuf, sizeOverallMsgBuf, _T("This demo application supports the following devices:\r\n\r\n"), NULL, NULL,
STRSAFE_IGNORE_NULLS | STRSAFE_NULL_ON_FAILURE | STRSAFE_NO_TRUNCATION);
// Enumerate supported devices.
for (DWORD dwEnumIndex = 0; dwEnumIndex < DEFINITION_MaximumNumberOfDevicesSupportedByThisApp; ++dwEnumIndex)
{
  USHORT usVendorID;
  USHORT usProductID;
  WCHAR wszFriendlyNameBuf[256]; // Character array containing the friendly name with a terminating NULL.
  DWORD dwSizeOfFriendlyNameBuf = sizeof(wszFriendlyNameBuf) / sizeof(wszFriendlyNameBuf[0]);
  DWORD dwDeviceCategoryBitwiseMask;
  DWORD dwReserved1;
  DWORD dwReserved2;
  DWORD dwReserved3;
  hr = m_pICTLEDMgr_Generic->EnumSupportedDevices(
      dwEnumIndex, &usVendorID, &usProductID, wszFriendlyNameBuf, &dwSizeOfFriendlyNameBuf,
      &dwDeviceCategoryBitwiseMask, &dwReserved1, &dwReserved2, &dwReserved3, dwFlag);
  if (SUCCEEDED(hr))
  {
    // Convert the device category bitwise mask to a device category friendly name for display purpose.
    WCHAR wszDeviceCategoryFriendlyNameBuf[256]; // Character array containing the device category friendly name with a terminating NULL.
    DWORD dwSizeOfDeviceCategoryFriendlyNameBuf = sizeof(wszDeviceCategoryFriendlyNameBuf) / sizeof(wszDeviceCategoryFriendlyNameBuf[0]);
    hr = CTGetDeviceCategoryFriendlyName(dwDeviceCategoryBitwiseMask, wszDeviceCategoryFriendlyNameBuf,
&dwSizeOfDeviceCategoryFriendlyNameBuf);

    TCHAR szMsgBuf[512];
    hr = StringCchPrintf(szMsgBuf, sizeof(szMsgBuf) / sizeof(TCHAR), _T("Supported Device #%lu: %s\r\n    Vendor ID: 0x%04X; Product ID:
0x%04X;\r\n    Device Category:0x%08X %s\r\n\r\n"), dwEnumIndex, wszFriendlyNameBuf, usVendorID, usProductID, dwDeviceCategoryBitwiseMask,
wszDeviceCategoryFriendlyNameBuf);
    hr = StringCchCatEx(szOverallMsgBuf, sizeOverallMsgBuf, szMsgBuf, NULL, NULL, STRSAFE_IGNORE_NULLS | STRSAFE_NULL_ON_FAILURE);
  }
  else
  {
    // Always terminate the enumeration with an enumeration index 0xFFFFFFFF.
    hr = m_pICTLEDMgr_Generic->EnumSupportedDevices(
        0xFFFFFFFF, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, dwFlag);
    int iRes = MessageBox(szOverallMsgBuf, _T("Supported Devices"), MB_OK);
    break;
  }
}
```

5.2 Enumerate Connected Devices

The interface method ICTLEDMgr::EnumConnectedDevices allows the client software to enumerate the list of supported devices that are currently connected to the host computer system. For each device enumerated, the interface method returns the following device information to the client software:

- Vendor ID
- Product ID
- Device Serial Number
- Device Instance
- LED Information
- Total Number of LEDs
- Device Friendly Name

5.2.1 Support for multiple sets of the same device

The SDK supports the case whereby two or more sets of the same device are connected to the host computer system and allows the client software to control each set of the device. For example, the SDK supports the case whereby three sets of Sound BlasterX Siege M08 are connected to the host computer system.

5.2.1.1 Device Serial Number and Device Instances

To support multiple sets of the same device connected to the host computer system, either the device serial numbers or the device instances are needed to differentiate the different sets of the same device. As some devices may not have serial numbers, this SDK currently uses device instances to support multiple sets of the same device. In fact, the SDK currently always returns NULL for the device serial numbers.

5.2.2 LED Information

Currently, the LED information comprises information for the following:

- LED Connection Type
  This is either the built-in connection type or the external connection type. The SDK defines them as follows:
  - CTLED_ConnectionType_BuiltIn
  - CTLED_ConnectionType_External
- LED Hardware type
  This is only applicable for devices with CTLED_ConnectionType_External as its connection type. It describes whether the external LED hardware (that is connected to the device) has separate clock and data signal paths or the clock is encoded together with the data. The SDK defines them as follows:
  - CTLED_HardwareType_NotApplicable
  - CTLED_HardwareType_SeparateClockAndData
  - CTLED_HardwareType_ClockEncodedWithData

5.2.2.1 Connection Type

CTLED_ConnectionType_BuiltIn

The connection type CTLED_ConnectionType_BuiltIn describes those devices whereby the LEDs are built-in into the devices and cannot be changed by the users.

The following are some examples of devices with connection type CTLED_ConnectionType_BuiltIn:

- Sound BlasterX Vanguard K08 (US English)
- Sound BlasterX Vanguard K08 (German)
- Sound BlasterX Vanguard K08 (Nordic)
- Sound BlasterX Siege M04
- Sound BlasterX Katana
- Sound BlasterX Kratos S5
- Sound BlasterX AE-5 (Built-in Lighting)

CTLED_ConnectionType_External

The connection type CTLED_ConnectionType_External describes those devices whereby the LEDs are not built-in into the devices and the users have to connect external LED hardware (such as LED strips) to the device.

The following are some examples of devices with connection type CTLED_ConnectionType_External:

- Sound BlasterX AE-5 (External Lighting)

IMPORTANT NOTE:
For Sound BlasterX AE-5, connection type is used to differentiate between the built-in lighting and the external lighting.

5.2.2.2 Hardware Type

This is only applicable for devices with CTLED_ConnectionType_External as its connection type.

The following are some examples of devices with connection type CTLED_ConnectionType_External:

- Sound BlasterX AE-5 (External Lighting)

CTLED_HardwareType_SeparateClockAndData

The hardware type CTLED_HardwareType_SeparateClockAndData describes those devices whereby LED hardware has separate clock and data signal paths.

CTLED_HardwareType_ClockEncodedWithData

The hardware type CTLED_HardwareType_ClockEncodedWithData describes those devices whereby LED hardware has the clock encoded together with the data.

IMPORTANT NOTE:
For Sound BlasterX AE-5 (External Lighting), the interface method ICTLEDMgr::EnumConnectedDevices will always return the hardware type CTLED_HardwareType_SeparateClockAndData as the default hardware type. However, as the user can connect different types of external LED strips to the device, it is the responsibility of the client software to provide the correct hardware type information to the SDK in the interface method ICTLEDMgr::Open.

5.2.3 Sample Code: Enumerated Connected Devices

The following code fragment shows how to enumerate for the list of devices which are currently connected to the computer system.

```
// Enumerated devices.
// Array to contain the enumerated devices that are connected to the computer.
// The array is initialised to no connected devices.
CTLedDeviceInfo infoCTLedDeviceArray[DEFINITION_MaximumNumberOfDevicesSupportedByThisApp] = { { 0, 0, NULL, NULL, 0, 0, FALSE, NULL } };
HRESULT hr;
TCHAR szOverallMsgBuf[2048];
size_t sizeOverallMsgBuf = sizeof(szOverallMsgBuf) / sizeof(szOverallMsgBuf[0]);
hr = StringCchCopyEx(szOverallMsgBuf, sizeOverallMsgBuf, _T("This demo application detects that the following devices are connected to the
computer system:\r\n\r\n"), NULL, NULL, STRSAFE_IGNORE_NULLS | STRSAFE_NULL_ON_FAILURE | STRSAFE_NO_TRUNCATION);
// Enumerate connected devices.
for (DWORD dwEnumIndex = 0; dwEnumIndex < DEFINITION_MaximumNumberOfDevicesSupportedByThisApp; ++dwEnumIndex)
{
  DWORD dwSizeOfSerialNumberBuf = sizeof(infoCTLedDeviceArray[dwEnumIndex].wszSerialNumberBuf) /
sizeof(infoCTLedDeviceArray[dwEnumIndex].wszSerialNumberBuf[0]);
  DWORD dwSizeOfDeviceInstanceBuf = sizeof(infoCTLedDeviceArray[dwEnumIndex].wszDeviceInstanceBuf) /
sizeof(infoCTLedDeviceArray[dwEnumIndex].wszDeviceInstanceBuf[0]);
  DWORD dwSizeOfFriendlyNameBuf = sizeof(infoCTLedDeviceArray[dwEnumIndex].wszFriendlyNameBuf) /
sizeof(infoCTLedDeviceArray[dwEnumIndex].wszFriendlyNameBuf[0]);
  hr = m_pICTLEDMgr_Generic->EnumConnectedDevices(
      dwEnumIndex, &infoCTLedDeviceArray[dwEnumIndex].usVendorID, &infoCTLedDeviceArray[dwEnumIndex].usProductID,
      infoCTLedDeviceArray[dwEnumIndex].wszSerialNumberBuf, &dwSizeOfSerialNumberBuf,
      infoCTLedDeviceArray[dwEnumIndex].wszDeviceInstanceBuf, &dwSizeOfDeviceInstanceBuf,
      &infoCTLedDeviceArray[dwEnumIndex].usLedInfoFlag, &infoCTLedDeviceArray[dwEnumIndex].usTotalNumLeds,
      infoCTLedDeviceArray[dwEnumIndex].wszFriendlyNameBuf, &dwSizeOfFriendlyNameBuf, dwFlag);
  if (SUCCEEDED(hr))
  {
    infoCTLedDeviceArray[dwEnumIndex].fPhysicalLedOrderingIsReversed = FALSE;
    TCHAR szMsgBuf[512];
    hr = StringCchPrintf(szMsgBuf, sizeof(szMsgBuf) / sizeof(TCHAR), _T("Connected Device #%lu: %s\r\n"), dwEnumIndex,
infoCTLedDeviceArray[dwEnumIndex].wszFriendlyNameBuf);
    hr = StringCchCatEx(szOverallMsgBuf, sizeOverallMsgBuf, szMsgBuf, NULL, NULL, STRSAFE_IGNORE_NULLS | STRSAFE_NULL_ON_FAILURE);

    if ((infoCTLedDeviceArray[dwEnumIndex].usVendorID == DEFINITION_VENDOR_ID_SoundBlasterXAE5) &&
(infoCTLedDeviceArray[dwEnumIndex].usProductID == DEFINITION_PRODUCT_ID_SoundBlasterXAE5))
    {
      if (infoCTLedDeviceArray[dwEnumIndex].usLedInfoFlag == MAKEWORD(CTLED_ConnectionType_External,
CTLED_HardwareType_ClockEncodedWithData))
      {
        if (infoCTLedDeviceArray[dwEnumIndex].usTotalNumLeds == 0)
        {
          // Configure the total number of LEDs from user's settings.
          infoCTLedDeviceArray[dwEnumIndex].usTotalNumLeds = dwTotalNumExternalLeds_SBXAE5_UserSettings;
        }
        // Configure whether the physical LED ordering is reversed from user's settings.
        infoCTLedDeviceArray[dwEnumIndex].fPhysicalLedOrderingIsReversed = fPhysicalLedOrderingIsReversed_SBXAE5_UserSettings;
      }
    }
  }
  else
  {
    // Always terminate the enumeration with an enumeration index 0xFFFFFFFF.
    hr = m_pICTLEDMgr_Generic->EnumConnectedDevices(
        0xFFFFFFFF, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, dwFlag);

    // Terminate the array with a NULL Vendor ID and a NULL Product ID to denote there are no more devices
    infoCTLedDeviceArray[dwEnumIndex].usVendorID = 0x0000;
    infoCTLedDeviceArray[dwEnumIndex].usProductID = 0x0000;

    int iRes = MessageBox(szOverallMsgBuf, _T("Connected Devices"), MB_OK);
    break;
  }
}
```

6 Firmware-based versus Host-based Lighting Effects

The lighting effects of the LEDs can be driven by the following methods:

- Driven by the device firmware
- Driven by the host computer

The SDK allows the client software to drive the lighting effects by any of the above methods.

6.1 Firmware-based Lighting Effects

Firmware-based lighting effects are driven by the device firmware. The type of lighting effects supported by the device firmware is pre-defined by the respective device firmware.

6.1.1 Advantages and Disadvantages

The following describes the advantages and disadvantages of firmware-based lighting effects.

6.1.1.1 Advantages

The advantages of using firmware-based lighting effects are:

- The client software does not need to be running in the background after it has configured the desired lighting effects because the firmware is the one which will be driving the lighting effects.

6.1.1.2 Disadvantages

The disadvantages of using firmware-based lighting effects are:

- The lighting effects available are limited to those lighting effects that are supported by the respective device firmware. The type of lighting effects supported by the device firmware is pre-defined by the respective device firmware and thus, they may vary across different devices.
- As the LEDs of each device are controlled by independently the respective device firmware, perfect synchronization of LED patterns across different devices is not feasible.
- Perfect synchronization of the lighting effects with external events (such as a song playing in the computer system) is also not feasible.

6.1.2 Firmware-based Lighting Effects Supported

There are several firmware-based lighting effects supported by the SDK as follows.

- CTLED_Pattern_Static
  For this pattern, the LED lights up with static colours specified by the client software.

- CTLED_Pattern_Pulsate
  For this pattern, the LED lights up with pulsating motion in the colours specified by the client software. The client software can also specify the periodic time of the pulsating motion.

- CTLED_Pattern_Wave
  For this pattern, the LED lights up with wave moving motion in the colours specified by the client software. The client software can also specify the periodic time of the wave moving motion.

  Note that for CTLED_Pattern_Wave, the direction and properties of the wave motion can also be specified.

  Wave pattern can have one of the following directions:
  - CTLED_PatternDirection_LeftToRight
  - CTLED_PatternDirection_RightToLeft
  - CTLED_PatternDirection_TopToBottom
  - CTLED_PatternDirection_BottomToTop
  - CTLED_PatternDirection_Clockwise
  - CTLED_PatternDirection_Anticlockwise

  Each of the wave pattern directions can have one of the following properties:
  - CTLED_PatternDirectionFlag_Looping
  - CTLED_PatternDirectionFlag_Bouncing

- CTLED_Pattern_Aurora
  For this pattern, the LED lights up with aurora moving motion in the colours that are pre-defined by the SDK (which means the client software cannot set the colours for this pattern). The client software can specify the periodic time of the aurora moving motion.

- CTLED_Pattern_ColourCycle
  For this pattern, the LED lights up with colour cycling moving motion in the colours that are pre-defined by the SDK (which means the client software cannot set the colours for this pattern). The client software can specify the periodic time of the colour cycling moving motion.

6.1.3 Remapping of the Firmware-based Wave Pattern Direction

In general, different devices have different physical layouts of the LEDs. For example, some devices may only have LEDs laid out horizontally and not vertically.

For the convenience of the client software, if the specified LED wave pattern direction is not supported by the device due to the physical layouts of the LEDs on the devices, the SDK will automatically remapped the wave pattern direction as follows:

- If CTLED_PatternDirection_LeftToRight is not supported, it will be automatically remapped to the following (in order of priority):
  - CTLED_PatternDirection_BottomToTop
  - CTLED_PatternDirection_Anticlockwise
- If CTLED_PatternDirection_RightToLeft is not supported, it will be automatically remapped to the following (in order of priority):
  - CTLED_PatternDirection_TopToBottom
  - CTLED_PatternDirection_Clockwise
- If CTLED_PatternDirection_BottomToTop is not supported, it will be automatically remapped to the following (in order of priority):
  - CTLED_PatternDirection_LeftToRight
  - CTLED_PatternDirection_Anticlockwise
- If CTLED_PatternDirection_TopToBottom is not supported, it will be automatically remapped to the following (in order of priority):
  - CTLED_PatternDirection_RightToLeft
  - CTLED_PatternDirection_Clockwise
- If CTLED_PatternDirection_Anticlockwise is not supported, it will be automatically remapped to the following (in order of priority):
  - CTLED_PatternDirection_LeftToRight
  - CTLED_PatternDirection_BottomToTop
- If CTLED_PatternDirection_Clockwise is not supported, it will be automatically remapped to the following (in order of priority):
  - CTLED_PatternDirection_RightToLeft
  - CTLED_PatternDirection_TopToBottom

6.1.4 Firmware-based lighting effects supported by specific devices

A device may support only a subset of the above mentioned firmware-based lighting effects instead of supporting everything.

The following describes the type of firmware-based lighting effects supported by the various devices.

6.1.4.1 Sound BlasterX Vanguard K08

The following firmware-based light effects are supported.

- CTLED_Pattern_Static
- CTLED_Pattern_Pulsate
- CTLED_Pattern_Wave
  Wave pattern can have one of the following directions:
  - CTLED_PatternDirection_LeftToRight
  - CTLED_PatternDirection_RightToLeft
  - CTLED_PatternDirection_TopToBottom
  - CTLED_PatternDirection_BottomToTop
  - CTLED_PatternDirection_Clockwise
  - CTLED_PatternDirection_Anticlockwise

  Each of the wave pattern directions can have one of the following properties:
  - CTLED_PatternDirectionFlag_Looping

6.1.4.2 Sound BlasterX Siege M04

The following firmware-based light effects are supported.

- CTLED_Pattern_Static
- CTLED_Pattern_Pulsate
- CTLED_Pattern_Wave
  Wave pattern can have one of the following directions:
  - CTLED_PatternDirection_Clockwise
  - CTLED_PatternDirection_Anticlockwise

  Each of the wave pattern directions can have one of the following properties:
  - CTLED_PatternDirectionFlag_Looping

NOTE: When the client software specifies an unsupported wave pattern direction, the SDK will do a re-map of the wave pattern direction as follows:

- CTLED_PatternDirection_LeftToRight
- CTLED_PatternDirection_RightToLeft
- CTLED_PatternDirection_TopToBottom
- CTLED_PatternDirection_BottomToTop

6.1.4.3 Sound BlasterX Katana

The following firmware-based light effects are supported.

- CTLED_Pattern_Static
- CTLED_Pattern_Pulsate
- CTLED_Pattern_Wave
  Wave pattern can have one of the following directions:
  - CTLED_PatternDirection_LeftToRight
  - CTLED_PatternDirection_RightToLeft
  - CTLED_PatternDirection_TopToBottom
  - CTLED_PatternDirection_BottomToTop
  - CTLED_PatternDirection_Clockwise
  - CTLED_PatternDirection_Anticlockwise

  Each of the wave pattern directions can have one of the following properties:
  - CTLED_PatternDirectionFlag_Looping
  - CTLED_PatternDirectionFlag_Bouncing
- CTLED_Pattern_Aurora
- CTLED_Pattern_ColourCycle

6.1.4.4 Sound BlasterX Kratos S5

The following firmware-based light effects are supported.

- CTLED_Pattern_Static
- CTLED_Pattern_Pulsate
- CTLED_Pattern_Wave
  Wave pattern can have one of the following directions:
  - CTLED_PatternDirection_LeftToRight
  - CTLED_PatternDirection_RightToLeft
  - CTLED_PatternDirection_TopToBottom
  - CTLED_PatternDirection_BottomToTop
  - CTLED_PatternDirection_Clockwise
  - CTLED_PatternDirection_Anticlockwise

  Each of the wave pattern directions can have one of the following properties:
  - CTLED_PatternDirectionFlag_Looping
  - CTLED_PatternDirectionFlag_Bouncing
- CTLED_Pattern_Aurora
- CTLED_Pattern_ColourCycle

6.1.4.5 Sound BlasterX AE-5

The following firmware-based light effects are supported.

- CTLED_Pattern_Static
- CTLED_Pattern_Pulsate
- CTLED_Pattern_Wave
  Wave pattern can have one of the following directions:
  - CTLED_PatternDirection_LeftToRight
  - CTLED_PatternDirection_RightToLeft
  - CTLED_PatternDirection_TopToBottom
  - CTLED_PatternDirection_BottomToTop
  - CTLED_PatternDirection_Clockwise
  - CTLED_PatternDirection_Anticlockwise

  Each of the wave pattern directions can have one of the following properties:
  - CTLED_PatternDirectionFlag_Looping
  - CTLED_PatternDirectionFlag_Bouncing
- CTLED_Pattern_Aurora
- CTLED_Pattern_ColourCycle

6.2 Host-based Lighting Effects

Host-based lighting effects are driven by the host computer (via the client software with the help of the SDK).Thus, there is no limit to the type of host-based lighting effects that can be supported by the client software.

6.2.1 Advantages and Disadvantages

The following describes the advantages and disadvantages of host-based lighting effects.

6.2.1.1 Advantages

The advantages of using host-based lighting effects are:

- The lighting effects available are not limited to those lighting effects that are supported by the respective device firmware. In fact, there is no limit to the type of lighting effects that can be driven by the host computer and thus, it can be assured that whatever lighting effects you wants, it can be applied across all devices.
- As the LEDs of each device are driven by the host computer, synchronization of LED patterns across different devices is feasible with very high accuracy.
- Synchronization of the lighting effects with external events (such as a song playing in the computer system) is also feasible with very high accuracy.

6.2.1.2 Disadvantages

The disadvantages of using host-based lighting effects are:

- The client software needs to be running in the background after it has configured the desired lighting effects because the client software (running in the host computer) is the one which will be continuously driving the lighting effects.

6.2.2 Host-based Lighting Effects Supported

The following describes the type of host-based lighting effects supported by the SDK.

Note:
As the host computer is the one driving the lighting effects, all the host-based lighting effects supported by the SDK can be supported across all devices.

6.2.2.1 SDK pre-defined Host-based lighting effects

The SDK supports the following pre-defined host-based lighting effects across all devices.

- CTLED_Pattern_Static
  Static pattern can have one of the following pattern modes:
  - CTLED_PatternMode_Static_Default
- CTLED_Pattern_Pulsate
  Pulsate pattern can have one of the following pattern modes:
  - CTLED_PatternMode_Pulsate_Default
  - CTLED_PatternMode_Pulsate_AlternateIntensityDirection
  - CTLED_PatternMode_Pulsate_IntensityGradient
- CTLED_Pattern_Wave
  Wave pattern can have one of the following pattern modes:
  - CTLED_PatternMode_Wave_Default
  - CTLED_PatternMode_Wave_DiscreteFlow
- CTLED_Pattern_Aurora
  Aurora pattern can have one of the following pattern modes:
  - CTLED_PatternMode_Aurora_Default
- CTLED_Pattern_ColourCycle
  Colour Cycle pattern can have one of the following pattern modes:
  - CTLED_PatternMode_ColourCycle_Default

6.2.2.2 Custom host-based lighting effects defined by client software

As the SDK allows the client software to independently control the lighting effects of each and every LEDs of the devices, the client software can define any custom host-based lighting effects that it desires.

6.2.3 Timer Callback for Host-based Lighting Effects

For the client software to do a host-based lighting effect, it has to continuously update the LEDs with new LED colours because the client software is the one who drives the lighting effects.

The SDK provides an easy way for the client software to do this via the following steps:

1. The client software shall first register for a timer callback with the SDK. The SDK will then callback to the client software at a regular interval specified by the client software.
2. Every time the timer callback is triggered, the client software shall then get the new LED colours from the SDK.
3. The client software shall then configure the LEDs with the new colours.

Note:

- Step 2 and step 3 are the steps continuously drive the lighting effects.
- Step 2 and step 3 are performed repeatedly until the client software stops the lighting effects by unregistering the timer callback with the SDK.

6.2.4 Support for Choreography

In the timer callback for host-based lighting effects, the following information is passed to the client software:

- The periodic time in milliseconds of the timer callback that the timer is configured to. Note that however, there is no guarantee that the timer callback will always be triggered precisely at this configured periodic timing.
- The current time stamp of the computer system in terms of time ticks in milliseconds.
- The total time that has elapsed in milliseconds since the registered timer callback started.

The client software can then base on the above timing information to decide the desired lighting effects at that point of time. For example, the client software may want to synchronize the lighting effects to a song that was being played.

7 LED Layouts

The following describes the various types of layout of the LEDs.

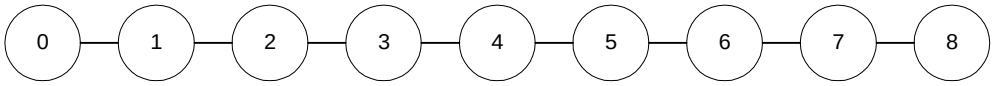
7.1 One row from left to right

The following are examples of devices with LEDs that are laid out in one row starting from left to right:

- Sound BlasterX Katana
- Sound BlasterX Kratos S5
- Sound BlasterX AE-5

The LEDs are numbered sequencially from left to right, beginning from 0 as shown in the example below.

x-Axis LED Layout



7.1.1 Naming of the LEDs

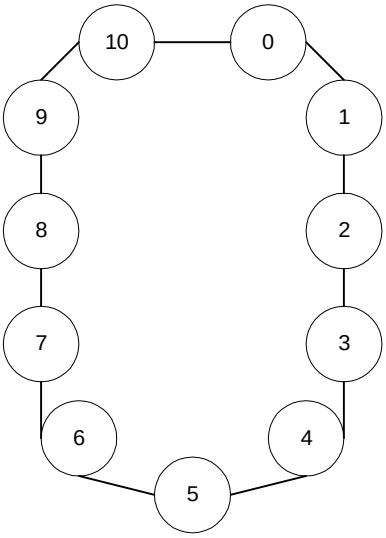
The SDK used the naming convention CTLEDIndex_*x* to define each of the LEDs where *x* is to be replaced with the LED number. For example, CTLEDIndex_0, CTLEDIndex_3 and CTLEDIndex_8.

7.2 Clockwise from the 12 o'clock position

The following are examples of devices with LEDs that are laid out in clockwise direction starting from the 12 o'clock position:

- Sound BlasterX Siege M04

The LEDs are numbered sequentially clockwise from the 12 o'clock position, beginning from 0 as shown in the example below.



7.2.1 Naming of the LEDs

The SDK used the naming convention CTLEDIndex_*x* to define each of the LEDs where *x* is to be replaced with the LED number. For example, CTLEDIndex_0, CTLEDIndex_3 and CTLEDIndex_8.

Note for Sound BlasterX Siege M04:
In addition to the 11 LEDs that are laid out in clockwise direction, Sound BlasterX Siege M04 also has two other special LEDs, naming one LED for the device logo and one LED for the mouse wheel.

- The SDK defined the special names CTMOUSE_LEDIndex_Logo and CTMOUSE_LEDIndex_Wheel for these two special LEDs.
- For consistency, the SDK also defined the names CTMOUSE_LEDIndex_0, CTMOUSE_LEDIndex_1, CTMOUSE_LEDIndex_2, …….., and CTMOUSE_LEDIndex_10 for the other 11 LEDs of Sound BlasterX Siege M04.
    - Note that CTLEDIndex_*x* can be used interchangeably with the corresponding CTMOUSE_LEDIndex_*x*.

7.3 Multiple rows and multiple columns

The following are examples of devices with LEDs that are laid out in multiple rows and multiple columns:

- Sound BlasterX Vanguard K08 (US English)
- Sound BlasterX Vanguard K08 (German)
- Sound BlasterX Vanguard K08 (Nordic)

7.3.1 Naming of the LEDs

The SDK used easily identifiable and self-explanatory naming convention to define each of the LEDs. For example, CTKEYBOARD_LEDIndex_Esc, CTKEYBOARD_LEDIndex_F3, CTKEYBOARD_LEDIndex_7, CTKEYBOARD_LEDIndex_K, CTKEYBOARD_LEDIndex_CapsLock and CTKEYBOARD_LEDIndex_RightCtrl.

8 LED Groupings

Although the SDK allows the client software to program the lighting effect of each of the LEDs of the device individually, LEDs are usually programmed in groups.

IMPORTANT NOTE:

- **LEDs that are grouped together in the same group will have the same lighting effect.**
- **The SDK uses a 2 dimensional array for the client software to describe its desired LED groupings.**

8.1 Scenarios of LED Groupings

The following describes the various scenarios of the LED groupings.

8.1.1 One LED group containing all the LEDs

This is the scenario whereby the client software sets up one (and only one) LED group containing all the LEDs.

In this scenario, all the LEDs (in the one and only LED group) will have the same lighting effect.

8.1.1.1 LED Grouping Array (2 Dimensional) representation

The following are examples of LED Grouping Array (2 dimensional) representations for one LED group containing all the LEDs.

Example 1
In the following LED grouping array,

- There is one LED group containing 10 LEDs from 0 to 9.

```
const DWORD
dwLedGroupingArray[DEFINITION_MaxNumRows_LedGroupingArray][DEFINITION_MaxNumCol
umns_LedGroupingArray] =
{   // Initialise the LED Grouping. Each row X corresponds to LED group X. The
first element of each row X specifies the number of LEDs in LED group X.
       { 10, CTLEDIndex_0, CTLEDIndex_1, CTLEDIndex_2, CTLEDIndex_3,
CTLEDIndex_4, CTLEDIndex_5, CTLEDIndex_6, CTLEDIndex_7, CTLEDIndex_8,
CTLEDIndex_9 }
};
```

Example 2
In the following LED grouping array,

- There is one LED group containing 7 LEDs from 0 to 6.

```
const DWORD
dwLedGroupingArray[DEFINITION_MaxNumRows_LedGroupingArray][DEFINITION_MaxNumCol
umns_LedGroupingArray] =
{   // Initialise the LED Grouping. Each row X corresponds to LED group X. The
first element of each row X specifies the number of LEDs in LED group X.
       { 5, CTLEDIndex_0, CTLEDIndex_1, CTLEDIndex_2, CTLEDIndex_3,
CTLEDIndex_4, CTLEDIndex_5, CTLEDIndex_6 }
};
```

8.1.2 Multiple LED groups with each LED group containing only one LED

This is the scenario whereby the client software sets up multiple LED groups with each LED group containing only one LED.

In this scenario, each of the LEDs can have its own individual lighting effect.

8.1.2.1 LED Grouping Array (2 Dimensional) representation

The following are examples of LED Grouping Array (2 dimensional) representations for multiple LED groups with each LED group containing only one LED.

Example 1
In the following LED grouping array,

- There are 10 LED groups containing 1 LED each.

```
const DWORD
dwLedGroupingArray[DEFINITION_MaxNumRows_LedGroupingArray][DEFINITION_MaxNumCol
umns_LedGroupingArray] =
{   // Initialise the LED Grouping. Each row X corresponds to LED group X. The
first element of each row X specifies the number of LEDs in LED group X.
          { 1, CTLEDIndex_0 }
        , { 1, CTLEDIndex_1 }
        , { 1, CTLEDIndex_2 }
        , { 1, CTLEDIndex_3 }
        , { 1, CTLEDIndex_4 }
        , { 1, CTLEDIndex_5 }
        , { 1, CTLEDIndex_6 }
        , { 1, CTLEDIndex_7 }
        , { 1, CTLEDIndex_8 }
        , { 1, CTLEDIndex_9 }
};
```

Example 2
In the following LED grouping array,

- There are 6 LED groups containing 1 LED each.

```
const DWORD
dwLedGroupingArray[DEFINITION_MaxNumRows_LedGroupingArray][DEFINITION_MaxNumCol
umns_LedGroupingArray] =
{   // Initialise the LED Grouping. Each row X corresponds to LED group X. The
first element of each row X specifies the number of LEDs in LED group X.
          { 1, CTLEDIndex_0 }
        , { 1, CTLEDIndex_1 }
        , { 1, CTLEDIndex_2 }
        , { 1, CTLEDIndex_3 }
        , { 1, CTLEDIndex_4 }
        , { 1, CTLEDIndex_5 }
};
```

8.1.3 Multiple LED groups with each LED group containing various number of LEDs

This is the scenario where the client software sets up multiple LED groups with each LED group containing various numbers of LEDs.

In this scenario, LEDs that are grouped together in the same LED group will have the same lighting effect.

8.1.3.1 LED Grouping Array (2 Dimensional) representation

The following are examples of LED Grouping Array (2 dimensional) representations for multiple LED groups with each LED group containing various numbers of LEDs.

Example 1
In the following LED grouping array,

- There are 5 LED groups.
  - LED Group #0 contains 2 LEDs (namely, LED Index #0 and #1).
  - LED Group #1 contains 1 LED (namely, LED Index #2).
  - LED Group #2 contains 4 LEDs (namely, LED Index #3, #4, #5 and #6).
  - LED Group #3 contains 1 LED (namely, LED Index #7).
  - LED Group #4 contains 2 LEDs (namely, LED Index #8 and #9).

```
const DWORD
dwLedGroupingArray[DEFINITION_MaxNumRows_LedGroupingArray][DEFINITION_MaxNumCol
umns_LedGroupingArray] =
{   // Initialise the LED Grouping. Each row X corresponds to LED group X. The
first element of each row X specifies the number of LEDs in LED group X.
          { 2, CTLEDIndex_0, CTLEDIndex_1 }
        , { 1, CTLEDIndex_2 }
        , { 4, CTLEDIndex_3, CTLEDIndex_4, CTLEDIndex_5, CTLEDIndex_6 }
        , { 1, CTLEDIndex_7 }
        , { 2, CTLEDIndex_8, CTLEDIndex_9 }
};
```

Example 2
In the following LED grouping array,

- There are 3 LED groups.
  - LED Group #0 contains 4 LEDs (namely, LED Index #0, #1, #7 and #8).
  - LED Group #1 contains 2 LEDs (namely, LED Index #2 and #6).
  - LED Group #2 contains 3 LEDs (namely, LED Index #3, #4 and #5).

```
const DWORD
dwLedGroupingArray[DEFINITION_MaxNumRows_LedGroupingArray][DEFINITION_MaxNumCol
umns_LedGroupingArray] =
{   // Initialise the LED Grouping. Each row X corresponds to LED group X. The
first element of each row X specifies the number of LEDs in LED group X.
          { 4, CTLEDIndex_0, CTLEDIndex_1, , CTLEDIndex_7, , CTLEDIndex_8 }
        , { 2, CTLEDIndex_2, CTLEDIndex_6 }
        , { 3, CTLEDIndex_3, CTLEDIndex_4, CTLEDIndex_5 }
};
```

8.1.4 The Global LED Group

The global LED group is only applicable to firmware-based lighting effects. It is not applicable for host-based lighting effects.

This is the scenario whereby the client software sets up the one (and only one) global LED group without the need to specify any LEDs for the group.

Global LED group is only applicable and must be used for the following firmware-based LED patterns:

- CTLED_Pattern_Wave
  - The global LED group is applicable because the client software has to specify the LED colours for firmware-based CTLED_Pattern_Wave as a whole instead of specifying the LED colours for the LED groups.
- CTLED_Pattern_Aurora
  - The global LED group is applicable because the client software does not have to specify any LED colours for firmware-based CTLED_Pattern_Aurora. The LED colours for firmware-based CTLED_Pattern_Aurora is pre-defined by the firmware.
- CTLED_Pattern_ColourCycle
  - The global LED group is applicable because the client software does not have to specify any LED colours for firmware-based CTLED_Pattern_ColourCycle. The LED colours for firmware-based CTLED_Pattern_ColourCycle is pre-defined by the fimrware.

8.1.4.1 LED Grouping Array (2 Dimensional) representation

The following is the LED Grouping Array (2 dimensional) representation for the global LED group. Note that in the global LED group, there is only one LED group but without any LEDs being specified.

```
const DWORD
dwLedGroupingArray[DEFINITION_MaxNumRows_LedGroupingArray][DEFINITION_MaxNumCol
umns_LedGroupingArray] =
{   // Initialise the LED Grouping. Each row X corresponds to LED group X. The
first element of each row X specifies the number of LEDs in LED group X.
      { 0 }
};
```

8.2 SDK Pre-defined LED Groupings versus Custom-defined LED Groupings

The SDK supports several SDK pre-defined LED groupings which are commonly used. These SDK pre-defined LED groupings greatly simplify the coding of the client software.

The SDK also allows the client software to create any desired custom LED groupings.

8.2.1 SDK Pre-defined LED Groupings

The following describes the various SDK pre-defined LED groupings supported by the SDK.

8.2.1.1 Grouping along x-axis direction

Generally speaking, grouping along the x-axis direction refers to LED groupings in a horizontal direction starting from left to right. The client application can create “n” number of LED groupings along the x-axis direction.

8.2.1.2 Grouping along y-axis direction

Generally speaking, grouping along the y-axis direction refers to LED groupings in a vertical direction starting from bottom to top. The client application can create “n” number of LED groupings along the y-axis direction.

8.2.1.3 Grouping along x-axis direction with mirror image

Generally speaking, grouping along the x-axis direction with mirror image refers to LED groupings in a horizontal direction starting from left to the middle of the x-axis. The right half of the LED grouping will be a mirror image of the left half of the LED grouping. The client application can create “n” number of LED groupings along the x-axis direction with mirror image.

8.2.1.4 Grouping along y-axis direction with mirror image

Generally speaking, grouping along the y-axis direction with mirror image refers to LED groupings in a vertical direction starting from bottom to the middle of the y-axis. The top half of the LED grouping will be a mirror image of the bottom half of the LED grouping. The client application can create “n” number of LED groupings along the y-axis direction with mirror image.

8.2.2 Custom-defined LED Groupings

The following describes how custom-defined LED groups can be set up.

8.2.2.1 Custom Grouping

For custom-defined LED grouping, please refer to the sample code of the `CTGetCustomLedGroupingArray()` function found in the sample program of the SDK.

9 LED Pattern And LED Colour

Each LED group can be assigned a LED pattern and the LED colours for the LED pattern.

9.1 For Pulsate Pattern and Static Pattern

The following describes how to set up the LED Pattern Array (1 dimensional) for Pulsate pattern and Static pattern.

9.1.1 Array (1 Dimensional) representation for LED Pattern

For illustration purpose, we assume that there are 7 LED groups from group 0 to 6.

The following is an example of a LED Pattern Array (1 dimensional) representation for setting the LED groups to the desired LED pattern (CTLED_Pattern_Pulsate or CTLED_Pattern_Static).

9.1.1.1 Example

```
CTLED_Pattern patternLED_Desired = CTLED_Pattern_Pulsate;
CTLED_Pattern
patternIndividualLedGroupArray1D_Desired[DEFINITION_MaxNumPatterns_PatternArray
] =
{   // Initialise the LED Pattern. Each row X of the array corresponds to the
pattern for LED group X.
         patternLED_Desired
       , patternLED_Desired
       , patternLED_Desired
       , patternLED_Desired
       , patternLED_Desired
       , patternLED_Desired
       , patternLED_Desired
};
```

IMPORTANT NOTE: For now, all the LED groups must be set to the same pattern.

9.1.2 Array (2 Dimensional) representation for LED Colour

For illustration purpose, we assume that there are 7 LED groups from group 0 to 6.

The following is an example of a LED Colour Array (2 dimensional) representation for setting the LED pattern of the LED groups to the desired LED colours.

9.1.2.1 Example

Note that for Pulsate LED pattern and Static LED pattern, each LED group can have only one colour.

```
CTColour
colourIndividualLedGroupArray2D_Desired[DEFINITION_MaxNumRows_ColourLayerArray]
[DEFINITION_MaxNumColumns_ColourLayerArray] =
{   // Initialise the LED Colour Layer. Each row X of the array corresponds to
the colour layers for LED group X.
         { DEFINITION_CTColour_Red }
       , { DEFINITION_CTColour_Orange }
       , { DEFINITION_CTColour_Yellow }
       , { DEFINITION_CTColour_Green }
       , { DEFINITION_CTColour_Blue }
       , { DEFINITION_CTColour_Indigo }
       , { DEFINITION_CTColour_White }
};
```

9.2 For Wave Pattern

The following describes how to set up the LED Pattern Array (1 dimensional) for Wave pattern.

As described earlier, **there is only one global LED group for Wave pattern**.

9.2.1 Array (1 Dimensional) representation for LED Pattern

The following is an example of a LED Pattern Array (1 dimensional) representation for setting the LED groups to the desired LED pattern (CTLED_Pattern_Wave).

9.2.1.1 Example

```
CTLED_Pattern patternLED_Desired = CTLED_Pattern_Wave;
CTLED_Pattern
patternIndividualLedGroupArray1D_Desired[DEFINITION_MaxNumPatterns_PatternArray
] =
{   // Initialise the LED Pattern. Each row X of the array corresponds to the
pattern for LED group X.
       patternLED_Desired
};
```

9.2.2 Array (2 Dimensional) representation for LED Colour

The following is an example of a LED Colour Array (2 dimensional) representation for setting the LED pattern of the one (and only) global LED group to the desired LED colours.

9.2.2.1 Example

Note that for Wave LED pattern, there is only one (and only) global LED group and the LED group can have multiple colours.

```
CTColour
colourIndividualLedGroupArray2D_Desired[DEFINITION_MaxNumRows_ColourLayerArray]
[DEFINITION_MaxNumColumns_ColourLayerArray] =
{   // Initialise the LED Colour Layer. Each row X of the array corresponds to
the colour layers for LED group X.
      { DEFINITION_CTColour_Red, DEFINITION_CTColour_Orange,
DEFINITION_CTColour_Yellow, DEFINITION_CTColour_Green, DEFINITION_CTColour_Blue
, DEFINITION_CTColour_Indigo, DEFINITION_CTColour_White }
};
```

IMPORTANT NOTE: From the above LED Colour Array (2 dimensional), you can see that:

- Only one row of the array is used to set the LED colours. This is because there is only one (and only) global LED group for Wave pattern.
- There are several colours specified in that one (and only) row of the array. These colours are the colours to be used for the Wave pattern.

9.3 For Aurora Pattern and Colour Cycle Pattern

The following describes how to set up the LED Pattern Array (1 dimensional) for Aurora pattern and Colour Cycle pattern.

As described earlier, **there is only one global LED group for Aurora pattern and Colour Cycle pattern**.

9.3.1 Array (1 Dimensional) representation for LED Pattern

The following is an example of a LED Pattern Array (1 dimensional) representation for setting the LED groups to the desired LED pattern (CTLED_Pattern_Wave or CTLED_Pattern_ColourCycle).

9.3.1.1 Example

```
CTLED_Pattern patternLED_Desired = CTLED_Pattern_Aurora;
CTLED_Pattern
patternIndividualLedGroupArray1D_Desired[DEFINITION_MaxNumPatterns_PatternArray
] =
{   // Initialise the LED Pattern. Each row X of the array corresponds to the
pattern for LED group X.
      patternLED_Desired
};
```

9.3.2 Array (2 Dimensional) representation for LED Colour

Note that for Aurora LED pattern and Colour Cycle LED pattern, there is only one (and only) global LED group and it is by design that the LED colours are pre-defined by the SDK. Thus, there is no need to set any LED colours.

10 Quick Start Programming Guide

The following is a quick start programming guide to serve as a simple tutorial to get you started on programming the LED patterns and colours.

Programming the LED patterns and colours involves the following procedure:

- Initialize the Creative LED Manager Library.
- Register for notification callback. (Optional)
- Open the desired device.
- Register for timer callback (only for controlling host-based lighting effects).
- Set the LED.
  - Note that this is the step in the procedure where the main bulk of the LED programming activities take place.
  - The client software will be continuously performing this step of setting the LED into different patterns and colours.
- Unregister timer callback (if it was registered beforehand).
- Close device.
- Unregister notification callback (if it was registered beforehand).
- Shutdown the Creative LED Manager Library.

10.1 Initialize the Creative LED Manager Library

The following code fragment shows how to initialize the Creative LED Manager Library.

Code Fragment

```
ICTLEDMgr *pICTLEDMgr;
LPTSTR szFullFilePathOfCTLEDMgrDllBuf = _T("C:\\MyTestApp\\CTLEDMgr.dll");
CTINTRFCRESULT resultCTIntrfc = CTCreateInstanceEx(
        CLSID_CCTLEDMgr, NULL, 0,
        IID_ICTLEDMgr, NULL, NULL, szFullFilePathOfCTLEDMgrDllBuf, NULL,
         (void **)&pICTLEDMgr);
if (resultCTIntrfc == CTINTRFCRESULT_Success)
{
        DWORD dwFlag = 0;
        HRESULT hr = pICTLEDMgr->Initialize(
                DEFINITION_CTLEDMgr_Interface_Version, dwFlag);
        if (SUCCEEDED(hr))
        {
                // You may now call all the other methods of
                // the interface pICTLEDMgr.
        }
        else
        {
                ULONG ul = pICTLEDMgr->Release();
                pICTLEDMgr = NULL;
                CTFreeUnusedLibrariesEx();
        }
}
else
{
        pICTLEDMgr = NULL;
}
```

10.2 Register for notification callback

The following code fragment shows how to register for notification callback.

Code Fragment

```
DWORD dwFlag = 0;
LPARAM lparamUserData = (LPARAM)4321; // Set to any user data that the client
software may need.
hr = pICTLEDMgr->RegisterNotificationCallback(
       CTLEDMgrNotifyProc, lparamUserData, dwFlag);
```

10.3 Open the desired device

The following code fragment shows how to open the desired device.

Note: ICTLEDMgr::EnumConnectedDevices may be called to obtain the desired device information that is required for ICTLEDMgr::Open.

Code Fragment

```
DWORD dwFlag = 0;
USHORT usVendorID = 0x041E; // Vendor ID of the product.
USHORT usProductID = 0x3126; // Product ID of the product.
LPCTSTR lpcwszSerialNumber = NULL; // NULL means any serial number.
LPCTSTR lpcszDeviceInstance = NULL; // NULL means any device instance.
USHORT usLedInfoFlag = 0x0000; // 0x0000 means default.
USHORT usTotalNumLeds = 0x0000; // 0x0000 means default.
DWORD dwDetailErrorCode;
usLedInfoFlag = MAKEWORD(CTLED_ConnectionType_BuiltIn,
              CTLED_HardwareType_SeparateClockAndData);
usTotalNumLeds = 5;
usLedInfoFlag = MAKEWORD(CTLED_ConnectionType_External,
              CTLED_HardwareType_ClockEncodedWithData);
usTotalNumLeds = 40;

HRESULT hr = pICTLEDMgr->Open(
              usVendorID, usProductID, lpcwszSerialNumber, lpcszDeviceInstance,
              usUsagePage, usUsageID, NULL, &dwDetailErrorCode, dwFlag);
if (SUCCEEDED(hr))
{
       // You may now call the relevant methods of
       // the interface pICTLEDMgr to perform the
       // desired LED functions.
}
```

10.4 Register for timer callback

The following code fragment shows how to register for timer callback.

Note:
Timer callback is required for host-based lighting effects. It is not needed for firmware-based lighting effects.

Code Fragment

```
const DWORD dwDueTime = 0;
const DWORD dwPeriodicTimeDesired = 1; // Set the desired period of the timer.
Here, we set it to 1 millisecond.
CTTIMERINFOPARAM paramCTTimerInfo = { dwDueTime, dwPeriodicTimeDesired };
ICTLEDMgr *pICTLEDMgr = pCTLedInterfaceInfo->pICTLEDMgr;
HRESULT hr = pICTLEDMgr->RegisterTimerCallback(CTLEDMgrTimerProc,
&paramCTTimerInfo, (LPARAM)pCTLedInterfaceInfo, dwFlag);
if (SUCCEEDED(hr))
{
	// Here, the paramCTTimerInfo parameter has been updated with for the
actual timer values that were used to configure the timer.
	// The client software may want to check the updated paramCTTimerInfo if
necessary.
}
```

10.5 Set the LED

Setting the LED involves the following procedure:

- Prepare LED grouping.
- Initialize LED information.
- Fill up LED information.
- Fill up data structure for Set LED.
- Execute command for Set LED.
- Clean up LED information.

Note:
For host-based lighting effects, setting the LEDs is usually done in the timer callback function. For some examples of the tasks that are being performed inside the timer callback function, please refer to the function ExecuteTimerDrivenEventToDoHostBasedLightingEffects( ) in the sample code of this SDK.

10.5.1 Prepare LED grouping

The following code fragment shows how to prepare the LED grouping. The objective of this is to fill up the LED Grouping Array (2 dimensional) with the desired LED grouping information.

10.5.1.1 SDK Pre-defined LED Grouping

In the code fragment, ICTLEDMgr::PrepareLedGrouping is called to prepare for the SDK pre-defined "7 LED groups along the x-axis direction".

Code Fragment

```
//---- LED Grouping Array (2 dimensional). Here, we shall fill up this LED
Grouping Array (2D).
DWORD
dwCustomLedGroupingArray2D_Desired[DEFINITION_MaxNumRows_LedGroupingArray][DEFI
NITION_MaxNumColumns_LedGroupingArray];
DWORD dwNumLedGroupsCreated = 0; // This contains the actual number of LED
groups created.

DWORD dwTotalNumLeds = 0; // 0 denotes the default total number of LEDs for the
device.
DWORD dwDesiredNumLedGroups = 7;
BOOL fMakeDesiredNumLedGroupsNotMoreThanTotalNumLeds = TRUE;
CTLEDGROUPINGCMDPARAM_ByDesiredNumLedGroups_Axis param = { dwTotalNumLeds,
dwDesiredNumLedGroups, fMakeDesiredNumLedGroupsNotMoreThanTotalNumLeds,
DEFINITION_MaxNumColumns_LedGroupingArray,
&dwCustomLedGroupingArray2D_Desired[0][0], 0 };
hr = pICTLEDMgr->PrepareLedGrouping(
        CTLEDGROUPINGCMD_ByDesiredNumLedGroups_xAxis, (LPARAM)&param, dwFlag);
if (SUCCEEDED(hr)) // LED Grouping Array (2D) is filled up successfully.
{
        dwNumLedGroupsCreated = param.dwNumLedGroupsCreated;
}
```

10.5.1.2 Custom-defined LED Grouping

Custom-defined LED grouping allows the client software to group the LEDs in anyway it wants instead of using the SDK pre-defined LED groupings.

Code Fragment

For custom-defined LED grouping, please refer to the sample code of the `CTGetCustomLedGroupingArray()` function found in the sample program of the SDK.

10.5.2 Initialize LED information

The following code fragment shows how to initialize the LED information. The objective of this is to initialize the CTColourLayerForMultipleLedGroups data structure which is pointed to by the pinfoLed pointer.

Code Fragment

```
// pSetLedSettings is a pointer to the CTLEDMGRCMDPARAM_SetLedSettings data
structure containing the desired LED settings.
PCTLEDMGRCMDPARAM_SetLedSettings pSetLedSettings;

// pinfoLed is a pointer to the CTColourLayerForMultipleLedGroups data
structure containing the desired LED grouping and colour information.
PCTColourLayerForMultipleLedGroups pinfoLed = &pSetLedSettings->colourLed;

DWORD dwFlag = 0;
const DWORD dwMaxNumLedGroups = DEFINITION_MaxNumLedGroups_SufficientlyLarge;
const DWORD dwMaxNumLedsInEachGroup =
       DEFINITION_MaxNumLedsPerLedGroup_SufficientlyLarge;
const DWORD dwMaxNumColourLayersInEachgroup =
       DEFINITION_MaxNumColourLayersPerLedGroup_SufficientlyLarge;
CTLEDINFOCMDPARAM_Initialize infoLedInitialize = { dwMaxNumLedGroups,
dwMaxNumLedsInEachGroup, dwMaxNumColourLayersInEachgroup, pinfoLed };
hr = pICTLEDMgr->PrepareLedInfo(CTLEDINFOCMD_Initialize,
(LPARAM)&infoLedInitialize, dwFlag);
```

10.5.3 Fill up LED information

Essentially, we fill up the LED information based on the following arrays:

- LED Grouping Array (2 dimensional)
- LED Pattern Array (1 demensional)
- LED Colour Array (2 Dimensional)

The following code fragment shows how to fill up the LED information. The objective of this is to fill up the CTColourLayerForMultipleLedGroups data structure which is pointed to by the pinfoLed pointer.

Note that there are two methods to fill up the LED information.

- Method 1: If you need to fill up all the LED information, it is easier to call for CTLEDINFOCMD_FillupAll.
- Method 2: If you do not need to fill up all the LED information (because some LED information are unchanged), you may call for the respective CTLEDINFOCMD_xxx instead of CTLEDINFOCMD_FillupAll.

Code Fragment for method 1
This fills up all the LED information at one go.

```
DWORD dwFlag = 0;
CTLEDINFOCMDPARAM_FillupAll infoLedFillupAll;
infoLedFillupAll.dwNumLedGroups = dwNumLedGroups_Desired;
infoLedFillupAll.pPatternIndividualLedGroupArray1D = pPatternArray1D_Desired;
infoLedFillupAll.pdwLedGroupingArray2D = pdwLedGroupingArray2D_Desired; //
(PDWORD)dwLedGroupingArray2D
infoLedFillupAll.dwNumColumnsOfLedGroupArray2D =
dwNumColumnsOfLedGroupingArray2D;
infoLedFillupAll.dwNumColourLayers = dwNumColourLayers_Desired;
infoLedFillupAll.pColourIndividualLedGroupArray2D = pColourArray2D_Desired; //
(PCTColour)colourArray2D_Desired
infoLedFillupAll.dwNumColumnsOfColourIndividualLedGroupArray2D =
dwNumColumnsOfColourArray2D;
infoLedFillupAll.pLedInfo = pinfoLed;
hr = pICTLEDMgr->PrepareLedInfo(
       CTLEDINFOCMD_FillupAll, (LPARAM)&infoLedFillupAll, dwFlag);
```

Code Fragment for method 2
This fills up the LED information one by one.

```
DWORD dwFlag = 0;
CTLEDINFOCMDPARAM_FillupNumLedGroups infoLedFillupNumLedGroups = {
dwNumLedGroups_Desired, pinfoLed };
hr = pICTLEDMgr->PrepareLedInfo(
       CTLEDINFOCMD_FillupNumLedGroups,
       (LPARAM)&infoLedFillupNumLedGroups, dwFlag);
for (DWORD dwLedGroupIndex = 0; dwLedGroupIndex < dwNumLedGroups_Desired;
++dwLedGroupIndex)
{
       CTLED_Pattern patternGroupLed_Desired = *(pPatternArray1D_Desired +
dwLedGroupIndex); // patternArray1D_Desired[dwLedGroupIndex];
       CTLEDINFOCMDPARAM_FillupGroupLedPattern infoLedFillupGroupLedPattern = {
dwLedGroupIndex, patternGroupLed_Desired, pinfoLed };
       hr = pICTLEDMgr->PrepareLedInfo(
              CTLEDINFOCMD_FillupGroupLedPattern,
              (LPARAM)&infoLedFillupGroupLedPattern, dwFlag);

       DWORD dwNumLedsInGroup_Desired = *(pdwLedGroupingArray2D_Desired +
(dwLedGroupIndex * dwNumColumnsOfLedGroupingArray2D)); //
dwLedGroupingArray2D[dwLedGroupIndex][0];
       CTLEDINFOCMDPARAM_FillupNumLedsInGroup infoLedFillupNumLedsInGroup = {
dwLedGroupIndex, dwNumLedsInGroup_Desired, pinfoLed };
       hr = pICTLEDMgr->PrepareLedInfo(
              CTLEDINFOCMD_FillupNumLedsInGroup,
              (LPARAM)&infoLedFillupNumLedsInGroup, dwFlag);
       for (DWORD dwIndex = 0; dwIndex < dwNumLedsInGroup_Desired; ++dwIndex)
       {
              DWORD dwLedID_Desired = *(pdwLedGroupingArray2D_Desired +
((dwLedGroupIndex * dwNumColumnsOfLedGroupingArray2D) + (dwIndex + 1))); //
dwLedGroupingArray2D[dwLedGroupIndex][dwIndex + 1];
              CTLEDINFOCMDPARAM_FillupLedID infoLedFillupLedID = {
dwLedGroupIndex, dwIndex, dwLedID_Desired, pinfoLed };
              hr = pICTLEDMgr->PrepareLedInfo(
                     CTLEDINFOCMD_FillupLedID,
                     (LPARAM)&infoLedFillupLedID, dwFlag);
       }

       CTLEDINFOCMDPARAM_FillupNumLedColourLayers
infoLedFillupNumLedColourLayers = { dwLedGroupIndex, dwNumColourLayers_Desired,
pinfoLed };
       hr = pICTLEDMgr->PrepareLedInfo(
              CTLEDINFOCMD_FillupNumLedColourLayers,
              (LPARAM)&infoLedFillupNumLedColourLayers, dwFlag);
       for (DWORD dwColourLayerIndex = 0; dwColourLayerIndex <
dwNumColourLayers_Desired; ++dwColourLayerIndex)
       {
              CTColour colourIndividualLedGroup_Desired =
*(pColourArray2D_Desired + ((dwLedGroupIndex * dwNumColumnsOfColourArray2D) +
(dwColourLayerIndex))); //
colourArray2D_Desired[dwLedGroupIndex][dwColourLayerIndex];
              CTLEDINFOCMDPARAM_FillupLedColour infoLedFillupLedColour = {
dwLedGroupIndex, dwColourLayerIndex, colourIndividualLedGroup_Desired, pinfoLed
};
              hr = pICTLEDMgr->PrepareLedInfo(
                     CTLEDINFOCMD_FillupLedColour,
                     (LPARAM)&infoLedFillupLedColour, dwFlag);
       }
```

```
}
```

10.5.4 Fill up data structure for Set LED

The following code fragment shows how to fill up the CTLEDMGRCMDPARAM_SetLedSettings data structure for Set LED.

Code Fragment

```
// pSetLedSettings is a pointer to the CTLEDMGRCMDPARAM_SetLedSettings data
structure containing the desired LED settings.

// Initialise the parameters (to default values).
CTInit_CTLEDMGRCMDPARAM_SetLedSettings(pSetLedSettings);

// Set the parameters to the desired values.
pSetLedSettings->fIgnorePattern = FALSE;
pSetLedSettings->patternLed = patternLED_Desired;
pSetLedSettings->fIgnorePatternDirection =
       ((patternLED_Desired == CTLED_Pattern_Wave) ? FALSE : TRUE);
pSetLedSettings->directionLedPattern = directionLedPattern_Desired;
pSetLedSettings->fIgnorePatternDirectionFlag =
       ((patternLED_Desired == CTLED_Pattern_Wave) ? FALSE : TRUE);
pSetLedSettings->flagLedPatternDirection = flagLedPatternDirection_Desired;
pSetLedSettings->fIgnorePeriodicTime =
       ((patternLED_Desired == CTLED_Pattern_Static) ? TRUE : FALSE);
pSetLedSettings->patternLedThePeriodicTimeIsFor = patternLED_Desired;
pSetLedSettings->dwPeriodicTimeInMilliseconds =
       dwPeriodicTimeInMilliseconds_Desired;
pSetLedSettings->dwPeriodicTimeInCyclesPerMinute =
       dwPeriodicTimeInCyclesPerMinute_Desired;
pSetLedSettings->fIgnoreColour = FALSE;

// Recall that pinfoLed is a pointer to the data structure of pSetLedSettings-
>colourLed.
// pSetLedSettings->colourLed has already been set earlier when we set the
parameters of pinfoLed.
```

10.5.5 Execute command for Set LED

The following code fragment shows how to execute the command to set the LED.

Code Fragment

```
// pSetLedSettings is a pointer to the CTLEDMGRCMDPARAM_SetLedSettings data
structure containing the desired LED settings.
DWORD dwFlag = 0;
hr = pICTLEDMgr->ExecuteCommand(
       CTLEDMGRCMD_SetLedSettings, (LPARAM)pSetLedSettings, dwFlag);
```

10.5.6 Clean up LED information

The following code fragment shows how to clean up the LED information.

Code Fragment

```
// pinfoLed is a pointer to the CTColourLayerForMultipleLedGroups data
structure containing the desired LED LED grouping and colour information.
DWORD dwFlag = 0;
CTLEDINFOCMDPARAM_Cleanup infoLedCleanup = { dwMaxNumLedGroups, pinfoLed };
hr = pICTLEDMgr->PrepareLedInfo(
     CTLEDINFOCMD_Cleanup, (LPARAM)&infoLedCleanup, dwFlag);
```

10.6 Unregister timer callback

The following code fragment shows how to unregister the timer callback.

Code Fragment

```
DWORD dwFlag = 0;
hr = pICTLEDMgr->UnregisterTimerCallback(dwFlag);
```

10.7 Close device

The following code fragment shows how to close the device.

Code Fragment

```
DWORD dwFlag = 0;
hr = pICTLEDMgr->Close(dwFlag);
```

10.8 Unregister notification callback

The following code fragment shows how to unregister the notification callback.

Code Fragment

```
DWORD dwFlag = 0;
hr = pICTLEDMgr->UnregisterNotificationCallback(dwFlag);
```

10.9 Shutdown the Creative LED Manager Library

The following code fragment shows how to shutdown the Creative LED Manager Library.

Code Fragment

```
DWORD dwFlag = 0;
hr = pICTLEDMgr->Shutdown(dwFlag);
ULONG ul = pICTLEDMgr->Release();
pICTLEDMgr = NULL;
CTFreeUnusedLibrariesEx();
```

11 Other Programming Functions

The ICTLEDMgr interface also allows a client software to perform other functions of the Creative Aurora Reactive lighting system as described below.

11.1 Restore the LED pattern to default

The following code fragment shows how to restore the LED pattern to default.

NOTE: The default LED pattern is pre-defined by the SDK. As the default LED pattern may be different across different versions of the SDK libraries, the client software shall not make any assumption regarding what the default LED pattern will be.

Code Fragment

```
hr = pICTLEDMgr->ExecuteCommand(
        CTLEDMGRCMD_RestoreDefaultLedSettings, (LPARAM)NULL, dwFlag);
```

12 Reference for ICTLEDMgr Interface

The ICTLEDMgr interface allows a client software to control the LED lighting of the Creative Aurora Reactive lighting system.

The ICTLEDMgr interface has the following methods.

12.1 ICTLEDMgr::Initialize

Syntax
HRESULT __stdcall Initialize(IN DWORD dwInterfaceVersion, IN DWORD dwFlag)

Parameters
dwInterfaceVersion [IN]
Type: DWORD
The version of the interface that the client is using. Set this parameter to the value DEFINITION_CTLEDMgr_Interface_Version as defined in the ICTLEDMgr.h header file.

dwFlag [IN]
Type: DWORD
Reserved. Set this parameter to 0.

Return Value
Type: HRESULT
One of the HRESULT values.

Remarks

Examples

12.2 ICTLEDMgr::RegisterNotificationCallback

Syntax

HRESULT __stdcall RegisterNotificationCallback(IN PFNCTLEDMGRNOTIFYPROC pfnCTLEDMgrNotifyProc, IN LPARAM lparamUserData, IN DWORD dwFlag)

Parameters

pfnCTLEDMgrNotifyProc [IN]

Type: PFNCTLEDMGRNOTIFYPROC
A pointer to the client's callback function which receives notifications for changes to the LED status or any other relevant status.

lparamUserData [IN]

Type: LPARAM
A client defined data parameter value that will be passed back to the client in the callback function.

dwFlag [IN]

Type: DWORD
Reserved. Set this parameter to 0.

Return Value

Type: HRESULT
One of the HRESULT values.

Remarks

This method may be called before ICTLEDMgr::Open.

Examples

12.3 ICTLEDMgr::Open

Syntax

HRESULT __stdcall Open(IN USHORT usVendorID, IN USHORT usProductID, IN LPCWSTR lpcwszSerialNumber, IN LPCWSTR lpcwszDeviceInstance, IN USHORT usLedInfoFlag, IN USHORT usTotalNumLeds, IN BOOL fPhysicalLedOrderingIsReversed, OUT PDWORD pdwDetailErrorCode, IN DWORD dwFlag)

Parameters

usVendorID [IN]

Type: USHORT
The vendor ID of the device.

usProductID [IN]

Type: USHORT
The product ID of the device.

lpcwszSerialNumber [IN]

Type: LPCWSTR
The serial number of the device.

Note: This is reserved for future use. Set this parameter to NULL.

lpcszDeviceInstance [IN]

Type: LPCWSTR
The device instance of the device.

Note: This is reserved for future use. Set this parameter to NULL.

usLedInfoFlag [IN]

Type: USHORT
A flag which specifies the LED information. Set this parameter to 0 to denote default. Otherwise, set this parameter as follows:

The low byte of the flag is to be set to one of the following defined values:

- CTLED_ConnectionType_BuiltIn
- CTLED_ConnectionType_External

The high byte of the flag is to be set to one of the following defined values:

- CTLED_HardwareType_SeparateClockAndData
- CTLED_HardwareType_ClockEncodedWithData

usTotalNumLeds [IN]

Type: USHORT
The total number of LEDs that are connected to the device.

Note: For devices which have a fixed total number of LEDs, this parameter will be ignored.

fPhysicalLedOrderingIsReversed

Type: BOOL
This boolean flag is applicable for devices whereby the user may choose to layout the LEDs from left to right or from right to left.

- Set this to TRUE to indicate that the LEDs are laid from left to right.
- Set this to FALSE to indicate that the LEDs are laid from right to left.

Note: An example of a device where this Boolean flag is applicable is the external LEDs of Sound BlasterX AE-5 where the user can arrange the external LEDs from left to right or from right to left.

pdwDetailErrorCode [OUT]
Type: PDWORD
Contains the detail error code (if available) when the method returns a HRESULT value that is not equal to S_OK.

dwFlag [IN]
Type: DWORD
Reserved. Set this parameter to 0.

Return Value
Type: HRESULT
One of the HRESULT values.

Remarks

1, If the returned HRESULT value is S_FALSE, then pdwDetailErrorCode may contain the following detail error code:

- CTERROR_NewerVersionOfFirmwareIsNeeded

  The detail error code CTERROR_NewerVersionOfFirmwareIsNeeded means that a newer version of the firmware is needed for the device that is opened via ICTLEDMgr::Open. Although the device is opened successfully, some features may not be functional or may not be supported properly. As such, it is highly recommended that the client software calls ICTLEDMgr::Close to close the device and prompts the user to update the firmware of the device.

2. If the returned HRESULT value is E_ACCESSDENIED, then it means that the device is currently in use by another client software. This E_ACCESSDENIED error may happen if the device does not support concurrent access by multiple client software.

3. **It is highly recommended that the client app should call ICTLEDMgr::EnumConnectedDevices to obtain the desired device information that is required for ICTLEDMgr::Open.**

Examples

12.4 ICTLEDMgr::PrepareLedGrouping

Syntax

HRESULT __stdcall PrepareLedGrouping(IN CTLEDGROUPINGCMD cmd, IN OUT LPARAM lparam, IN DWORD dwFlag)

Parameters

cmd [IN]

Type: CTLEDGROUPINGCMD

The desired command to be executed. Set this parameter to one of the following CTLEDGROUPINGCMD commands.

- CTLEDGROUPINGCMD_ByDesiredNumLedGroups

lparam [IN, OUT]

Type: LPARAM

Command specific parameter.

CTLEDGROUPINGCMD command	Parameter that lparam points to
CTLEDGROUPINGCMD_ByOneGlobalLedGroup	CTLEDGROUPINGCMDPARAM_ByOneGlobalLedGroup
CTLEDGROUPINGCMD_ByDesiredNumLedGroups_xAxis	CTLEDGROUPINGCMDPARAM_ByDesiredNumLedGroups_Axis
CTLEDGROUPINGCMD_ByDesiredNumLedGroups_yAxis	CTLEDGROUPINGCMDPARAM_ByDesiredNumLedGroups_Axis
CTLEDGROUPINGCMD_ByDesiredNumLedGroups_xAxis_Mirror	CTLEDGROUPINGCMDPARAM_ByDesiredNumLedGroups_Axis
CTLEDGROUPINGCMD_ByDesiredNumLedGroups_xAxis_Mirror	CTLEDGROUPINGCMDPARAM_ByDesiredNumLedGroups_Axis

dwFlag [IN]

Type: DWORD

Reserved. Set this parameter to 0.

Return Value

Type: HRESULT

One of the HRESULT values.

Remarks

Examples

12.5 ICTLEDMgr::PrepareLedInfo

Syntax

HRESULT __stdcall OperateLedInfo(IN CTLEDINFOCMD cmd, IN OUT LPARAM lparam, IN DWORD dwFlag)

Parameters

cmd [IN]

Type: CTLEDINFOCMD

The desired command to be executed. Set this parameter to one of the following CTLEDINFOCMD commands.

- CTLEDINFOCMD_Initialize
- CTLEDINFOCMD_FillupAll
- CTLEDINFOCMD_FillupNumLedGroups
- CTLEDINFOCMD_FillupGroupLedPattern
- CTLEDINFOCMD_FillupNumLedsInGroup
- CTLEDINFOCMD_FillupLedID
- CTLEDINFOCMD_FillupNumLedColourLayers
- CTLEDINFOCMD_FillupLedColour
- CTLEDINFOCMD_FillupLedColourInReversedDirection
- CTLEDINFOCMD_Cleanup

lparam [IN, OUT]

Type: LPARAM

Command specific parameter.

CTLEDINFOCMD command	Parameter that lparam points to
CTLEDINFOCMD_Initialize	CTLEDINFOCMDPARAM_Initialize
CTLEDINFOCMD_FillupAll	CTLEDINFOCMDPARAM_FillupAll
CTLEDINFOCMD_FillupNumLedGroups	CTLEDINFOCMDPARAM_FillupNumLedGroups
CTLEDINFOCMD_FillupGroupLedPattern	CTLEDINFOCMDPARAM_FillupGroupLedPattern
CTLEDINFOCMD_FillupNumLedsInGroup	CTLEDINFOCMDPARAM_FillupNumLedsInGroup
CTLEDINFOCMD_FillupLedID	CTLEDINFOCMDPARAM_FillupLedID
CTLEDINFOCMD_FillupNumLedColourLayers	CTLEDINFOCMDPARAM_FillupNumLedColourLayers
CTLEDINFOCMD_FillupLedColour	CTLEDINFOCMDPARAM_FillupLedColour
CTLEDINFOCMD_FillupLedColourInReversedDirection	CTLEDINFOCMDPARAM_FillupLedColour
CTLEDINFOCMD_Cleanup	CTLEDINFOCMDPARAM_Cleanup

dwFlag [IN]

Type: DWORD

Reserved. Set this parameter to 0.

Return Value

Type: HRESULT

One of the HRESULT values.

Remarks

Examples

12.6 ICTLEDMgr::ExecuteCommand

Syntax
HRESULT __stdcall ExecuteCommand(IN CTLEDMGRCMD cmd, IN OUT LPARAM lparam, IN DWORD dwFlag)

Parameters
cmd [IN]
Type: CTLEDMGRCMD
The desired command to be executed. Set this parameter to one of the CTLEDMGRCMD commands.

lparam [IN, OUT]
Type: LPARAM
Command specific parameter.

dwFlag [IN]
Type: DWORD
Reserved. Set this parameter to 0.

Return Value
Type: HRESULT
One of the HRESULT values.

Remarks

Examples

12.7 ICTLEDMgr::Close

Syntax
HRESULT __stdcall Close(IN DWORD dwFlag)

Parameters
dwFlag [IN]
Type: DWORD
Reserved. Set this parameter to 0.

Return Value
Type: HRESULT
One of the HRESULT values.

Remarks

Examples

12.8 ICTLEDMgr::UnregisterNotificationCallback

Syntax
HRESULT __stdcall UnregisterNotificationCallback(IN DWORD dwFlag)

Parameters
dwFlag [IN]
Type: DWORD
Reserved. Set this parameter to 0.

Return Value
Type: HRESULT
One of the HRESULT values.

Remarks

Examples

12.9 ICTLEDMgr::Shutdown

Syntax
HRESULT __stdcall Shutdown(IN DWORD dwFlag)

Parameters
dwFlag [IN]
Type: DWORD
Reserved. Set this parameter to 0.

Return Value
Type: HRESULT
One of the HRESULT values.

Remarks

Examples

12.10 ICTLEDMgr::EnumSupportedDevices

This method is designed to be called without the prior need to call the ICTLEDMgr::Open method.

Syntax

HRESULT __stdcall EnumSupportedDevices(IN DWORD dwEnumIndex, OUT PUSHORT pusVendorID, OUT PUSHORT pusProductID, OUT LPWSTR wszDeviceFriendlyNameBuf, IN OUT PDWORD pdwSizeOfDeviceFriendlyNameBuf, OUT PDWORD pdwDeviceCategoryBitwiseMask, OUT PDWORD pdwReserved1, OUT PDWORD pdwReserved2, OUT PDWORD pdwReserved3, IN DWORD dwFlag)

Parameters

dwEnumIndex [IN]

Type: DWORD
The index of the current enumeration.

Note: You must start the device enumeration by first calling the ICTLEDMgr::EnumSupportedDevices method with dwEnumIndex equals to 0. After that, dwEnumIndex must be incremented by 1 for each subsequent call to the method until the method returns failure. After the method returns failure, terminate the device enumeration by calling the method with dwEnumIndex equals to 0xFFFFFFFF.

pusVendorID [OUT]

Type: PUSHORT
Contains the vendor ID of the enumerated device when the method returns success.

pusProductID [OUT]

Type: PUSHORT
Contains the product ID of the enumerated device when the method returns success.

wszDeviceFriendlyNameBuf [OUT]

Type: LPWSTR
Contains the device friendly name of the device the enumerated device when the method returns success.

Note: If this contains a NULL string when the method returns success, it means that the device friendly name of the enumerated device is not available or the method does not support getting device friendly name.

pdwSizeOfDeviceFriendlyNameBuf [IN, OUT]

Type: PDWORD
On entry, this contains the size (in number of WCHAR) of the wszDeviceFriendlyNameBuf buffer.
On exit, this contains the required size (in number of WCHAR) of the wszDeviceFriendlyNameBuf buffer when the method returns E_OUTOFMEMORY.

pdwDeviceCategoryBitwiseMask [OUT]

Type: PDWORD
On exit, this contains the CTDEVICECATEGORY_BitwiseMask_xxx bitwise mask denoting the category of the device. It is possible that a device can belong to more than one category.

pdwReserved1 [OUT]

Type: PDWORD
This is currently reserved for future use.

pdwReserved2 [OUT]

Type: PDWORD
This is currently reserved for future use.

pdwReserved3 [OUT]
Type: PDWORD
This is currently reserved for future use.

dwFlag [IN]
Type: DWORD
Reserved. Set this parameter to 0.

Return Value
Type: HRESULT
One of the HRESULT values.

Remarks

Examples

12.11 ICTLEDMgr::EnumConnectedDevices

This method is designed to be called without the prior need to call the ICTLEDMgr::Open method.

Syntax

HRESULT __stdcall EnumConnectedDevices(IN DWORD dwEnumIndex, OUT PUSHORT pusVendorID, OUT PUSHORT pusProductID, OUT LPWSTR wszSerialNumberBuf, IN OUT PDWORD pdwSizeOfSerialNumberBuf, OUT LPWSTR wszDeviceInstanceBuf, IN OUT PDWORD pdwSizeOfDeviceInstanceBuf, OUT PUSHORT pusLedInfoFlag, OUT PUSHORT pusTotalNumLeds, OUT LPWSTR wszDeviceFriendlyNameBuf, IN OUT PDWORD pdwSizeOfDeviceFriendlyNameBuf, IN DWORD dwFlag)

Parameters

dwEnumIndex [IN]

Type: DWORD
The index of the current enumeration.

Note: You must start the device enumeration by first calling the ICTLEDMgr::EnumConnectedDevices method with dwEnumIndex equals to 0. After that, dwEnumIndex must be incremented by 1 for each subsequent call to the method until the method returns failure. After the method returns failure, terminate the device enumeration by calling the method with dwEnumIndex equals to 0xFFFFFFFF.

pusVendorID [OUT]

Type: PUSHORT
Contains the vendor ID of the enumerated device when the method returns success.

pusProductID [OUT]

Type: PUSHORT
Contains the product ID of the enumerated device when the method returns success.

wszSerialNumberBuf [OUT]

Type: LPWSTR
Contains the serial number of the enumerated device when the method returns success.

Note: If this contains a NULL string when the method returns success, it means that the serial number of the enumerated device is not available or the method does not support getting serial number.

pdwSizeOfSerialNumberBuf [IN, OUT]

Type: PDWORD
On entry, this contains the size (in number of WCHAR) of the wszSerialNumberBuf buffer.
On exit, this contains the required size (in number of WCHAR) of the wszSerialNumberBuf buffer when the method returns E_OUTOFMEMORY.

wszDeviceInstanceBuf [OUT]

Type: LPWSTR
Contains the device instance of the enumerated device when the method returns success.

Note: If this contains a NULL string when the method returns success, it means that the device instance of the enumerated device is not available or the method does not support getting device instance.

pdwSizeOfDeviceInstanceBuf [IN, OUT]

Type: PDWORD

On entry, this contains the size (in number of WCHAR) of the wszDeviceInstanceBuf buffer.
On exit, this contains the required size (in number of WCHAR) of the wszDeviceInstanceBuf buffer when the method returns E_OUTOFMEMORY.

pusLedInfoFlag [OUT]
Type: PUSHORT
Contains the flag which specifies the LED information when the method returns success. The flag is described as follows:

The low byte of the flag is set to one of the following defined values:
- CTLED_ConnectionType_BuiltIn
- CTLED_ConnectionType_External

The high byte of the flag is set to one of the following defined values:
- CTLED_HardwareType_SeparateClockAndData
- CTLED_HardwareType_ClockEncodedWithData

pusTotalNumLeds [OUT]
Type: pUSHORT
Contains the total number of LEDs that are connected to the device when the method returns success.

Note: If the total number of LEDs is 0, then it means that the device does not have a definite number of LEDs. For example, some devices do not have built-in LEDs and it is up to the user to connect external LEDs to the devices.

wszDeviceFriendlyNameBuf [OUT]
Type: LPWSTR
Contains the device friendly name of the device the enumerated device when the method returns success.

Note: If this contains a NULL string when the method returns success, it means that the device friendly name of the enumerated device is not available or the method does not support getting device friendly name.

pdwSizeOfDeviceFriendlyNameBuf [IN, OUT]
Type: PDWORD
On entry, this contains the size (in number of WCHAR) of the wszDeviceFriendlyNameBuf buffer.
On exit, this contains the required size (in number of WCHAR) of the wszDeviceFriendlyNameBuf buffer when the method returns E_OUTOFMEMORY.

dwFlag [IN]
Type: DWORD
Reserved. Set this parameter to 0.

Return Value
Type: HRESULT
One of the HRESULT values.

Remarks

Examples

12.12 ICTLEDMgr::RegisterTimerCallback

Syntax
HRESULT __stdcall RegisterTimerCallback(IN PFNCTLEDMGRTIMERPROC pfnCTLEDMgrTimerProc, IN OUT PCTTIMERINFOPARAM pCTTimerInfoParam, IN LPARAM lparamUserData, IN DWORD dwFlag)

Parameters
pfnCTLEDMgrTimerProc [IN]
Type: PFNCTLEDMGRTIMERPROC
A pointer to the client's callback function which receives timer callback.

pCTTimerInfoParam [IN OUT]
Type: PCTTIMERINFOPARAM
On entry, this contains the values of the desired timer parameters to be used to configure the timer.
On exit, this contains the actual values that were used to configure the timer.

lparamUserData [IN]
Type: LPARAM
A client defined data parameter value that will be passed back to the client in the callback function.

dwFlag [IN]
Type: DWORD
Reserved. Set this parameter to 0.

Return Value
Type: HRESULT
One of the HRESULT values.

Remarks
This method may be called before ICTLEDMgr::Open.

Examples

12.13 ICTLEDMgr::UnregisterTimerCallback

Syntax
HRESULT __stdcall UnregisterTimerCallback(IN DWORD dwFlag)

Parameters
dwFlag [IN]
Type: DWORD
Reserved. Set this parameter to 0.

Return Value
Type: HRESULT
One of the HRESULT values.

Remarks

Examples

12.14 ICTLEDMgr::GetColourInfoOfPattern

Syntax

HRESULT __stdcall GetColourInfoOfPattern(IN CTLED_Pattern patternLed, IN OUT LPARAM lparam, IN DWORD dwFlag)

Parameters

patternLed [IN]

Type: CTLED_Pattern

The desired LED pattern the colour information is for. Set this parameter to one of the following CTLED_Pattern values:

- CTLED_Pattern_Static
- CTLED_Pattern_Pulsate
- CTLED_Pattern_Wave
- CTLED_Pattern_Aurora
- CTLED_Pattern_ColourCycle

lparam [IN, OUT]

Type: LPARAM

LED pattern specific parameter.

CTLED_Pattern value	Parameter that lparam points to
CTLED_Pattern_Static	CTLEDINFOPARAM_GetStaticColourInfo
CTLED_Pattern_Pulsate	CTLEDINFOPARAM_GetPulsateColourInfo
CTLED_Pattern_Wave	CTLEDINFOPARAM_GetWaveColourInfo
CTLED_Pattern_Aurora	CTLEDINFOPARAM_GetAuroraColourInfo
CTLED_Pattern_ColourCycle	CTLEDINFOPARAM_GetColourCycleColourInfo

dwFlag [IN]

Type: DWORD

Reserved. Set this parameter to 0.

Return Value

Type: HRESULT

One of the HRESULT values.

Remarks

Examples

13 Reference for Data Structure

The ICTLEDMgr interface methods use various data structures as parameters. The following describes the data structures.

13.1 CTLEDMGRCMDPARAM_SetLedSettings

Parameters

dwFlagForApplySettings [IN]
Type: DWORD
For now, always set this to DEFINITION_FlagForApplySettings_Default.

fIgnoreProfileID [IN]
Type: BOOL
For now, always set this to TRUE.

dwProfileIDMain [IN]
Type: DWORD
For now, always set this to DEFINITION_ProfileIDMain_Custom.

dwProfileIDSub [IN]
Type: DWORD
For now, always set this to DEFINITION_ProfileIDSub_Custom.

fPersistent [IN]
Type: BOOL
For now, always set this to FALSE.

fIgnoreMasterLedState [IN]
Type: BOOL
Boolean flag to indicate whether the parameter dwMasterLedState is to be ignored.

- Set this to TRUE to indicate that the parameter dwMasterLedState is to be ignored.
- Set this to FALSE to indicate that the master LED state is to be set according to the parameter dwMasterLedState.

dwMasterLedState [IN]
Type: DWORD
Set this to one of the following:

- DEFINITION_CTLEDMGR_LedMaster_Off
- DEFINITION_CTLEDMGR_LedMaster_On

fIgnoreGlobalPatternMode [IN]
Type: BOOL
For now, always set this to FALSE.

fGlobalPatternMode [IN]
Type: BOOL
For now, always set this to FALSE.

dwLedIndex [IN]
Type: DWORD
For now, always set this to DEFINITION_CTLEDIndex_NotApplicable.

fIgnorePattern [IN]
Type: BOOL
Boolean flag to indicate whether the parameter patternLed is to be ignored.

- Set this to TRUE to indicate that the parameter patternLed is to be ignored.

- Set this to FALSE to indicate that the LED pattern is to be set according to the parameter patternLed.

patternLed [IN]
Type: CTLED_Pattern
Set this to one of the CTLED_Pattern definitions.

fIgnorePatternDirection [IN]
Type: BOOL
Boolean flag to indicate whether the parameter directionLedPattern is to be ignored.
- Set this to TRUE to indicate that the parameter directionLedPattern is to be ignored.
- Set this to FALSE to indicate that the LED pattern direction is to be set according to the parameter directionLedPattern.

directionLedPattern [IN]
Type: CTLED_PatternDirection
Set this to one of the CTLED_PatternDirection definitions.
Note: If the specified LED pattern direction is not supported by the device, it will be automatically remapped as follows:
- If CTLED_PatternDirection_LeftToRight is not supported, it will be automatically remapped to the following (in order of priority):
  - CTLED_PatternDirection_BottomToTop
  - CTLED_PatternDirection_Anticlockwise
- If CTLED_PatternDirection_RightToLeft is not supported, it will be automatically remapped to the following (in order of priority):
  - CTLED_PatternDirection_TopToBottom
  - CTLED_PatternDirection_Clockwise
- If CTLED_PatternDirection_BottomToTop is not supported, it will be automatically remapped to the following (in order of priority):
  - CTLED_PatternDirection_LeftToRight
  - CTLED_PatternDirection_Anticlockwise
- If CTLED_PatternDirection_TopToBottom is not supported, it will be automatically remapped to the following (in order of priority):
  - CTLED_PatternDirection_RightToLeft
  - CTLED_PatternDirection_Clockwise
- If CTLED_PatternDirection_Anticlockwise is not supported, it will be automatically remapped to the following (in order of priority):
  - CTLED_PatternDirection_LeftToRight
  - CTLED_PatternDirection_BottomToTop
- If CTLED_PatternDirection_Clockwise is not supported, it will be automatically remapped to the following (in order of priority):
  - CTLED_PatternDirection_RightToLeft
  - CTLED_PatternDirection_TopToBottom

fIgnorePatternDirectionFlag [IN]
Type: BOOL
Boolean flag to indicate whether the parameter flagLedPatternDirection is to be ignored.
- Set this to TRUE to indicate that the parameter flagLedPatternDirection is to be ignored.
- Set this to FALSE to indicate that the LED pattern direction flag is to be set according to the parameter flagLedPatternDirection.

flagLedPatternDirection [IN]
Type: CTLED_PatternDirectionFlag
Set this to one of the CTLED_PatternDirectionFlag definitions.

fIgnorePeriodicTime [IN]
Type: BOOL

Boolean flag to indicate whether the parameters patternLedThePeriodicTimeIsFor, directionLedPatternThePeriodicTimeIsFor, flagLedPatternDirectionThePeriodicTimeIsFor, dwPeriodicTimeInMilliseconds and dwPeriodicTimeInCyclesPerMinute are to be ignored.

- Set this to TRUE to indicate that the parameters are to be ignored.
- Set this to FALSE to indicate that the periodic time is to be set according to the parameters.

patternLedThePeriodicTimeIsFor [IN]

Type: CTLED_Pattern
Set this to one of the CTLED_Pattern definitions.

directionLedPatternThePeriodicTimeIsFor [IN]

Type: CTLED_PatternDirection
For now, always set this to CTLED_PatternDirection_NotApplicable.

flagLedPatternDirectionThePeriodicTimeIsFor [IN]

Type: CTLED_PatternDirectionFlag
For now, always set this to CTLED_PatternDirectionFlag_NotApplicable.

dwPeriodicTimeInMilliseconds [IN]

Type: DWORD
Set this to the periodic time in milliseconds.
Note: You may set this to 0 if you prefer to set the periodic time based on the parameter dwPeriodicTimeInCyclesPerMinute.

dwPeriodicTimeInCyclesPerMinute [IN]

Type: DWORD
Set this to the periodic time in cycles per minute.
Note: This parameter will be ignored if the parameter dwPeriodicTimeInMilliseconds is not 0.

fIgnoreColour [IN]

Type: BOOL
Boolean flag to indicate whether the parameter colourLed is to be ignored.

- Set this to TRUE to indicate that the parameter colourLed is to be ignored.
- Set this to FALSE to indicate that the LED colour is to be set according to the parameter colourLed.

colourLed [IN]

Type: CTColourLayerForMultipleLedGroups
Set this parameter using the interface method ICTLEDMgr::PrepareLedInfo.

13.2 CTLEDGROUPINGCMDPARAM_ByOneGlobalLedGroup

Parameters

dwNumColumnsOfLedGroupingArray2D [IN]
Type: DWORD
Set this to specify the number of columns of the 2 dimensional array for the LED grouping that is specified by pdwLedGroupingArray2D.

pdwLedGroupingArray2D [OUT]
Type: PDWORD
Upon successful return from the function, this contains the pointer to the 2 dimensional array of DWORD for the LED grouping.

dwNumLedGroupsCreated [OUT]
Type: DWORD
Upon successful return from the function, this contains the number of LED groups actually created.

13.3 CTLEDGROUPINGCMDPARAM_ByDesiredNumLedGroups_Axis

Parameters

dwTotalNumLeds [IN]
Type: DWORD
Set this to specify the total number of LEDs.
Note: You may set it to 0 to denote default.

dwDesiredNumLedGroups [IN]
Type: DWORD
Set this to specify the desired number of LED groups.

fMakeDesiredNumLedGroupsNotMoreThanMaxAllowable [IN]
Type: BOOL
Boolean flag to indicate whether how to handle the scenarios whereby dwDesiredNumLedGroups is more than the maximum allowable.

- Set this to TRUE to indicate that the function shall assume that dwDesiredNumLedGroups is equal to the maximum allowable.
- Set this to FALSE to indicate that the function shall return error.

dwNumColumnsOfLedGroupingArray2D [IN]
Type: DWORD
Set this to specify the number of columns of the 2 dimensional array for the LED grouping that is specified by pdwLedGroupingArray2D.

pdwLedGroupingArray2D [OUT]
Type: PDWORD
Upon successful return from the function, this contains the pointer to the 2 dimensional array of DWORD for the LED grouping.

dwNumLedGroupsCreated [OUT]
Type: DWORD
Upon successful return from the function, this contains the number of LED groups actually created.

13.4 CTLEDINFOCMDPARAM_Initialize

Parameters

dwMaxNumLedGroups [IN]

Type: DWORD

Set this to specify the maximum possible number of LED groups.

Note: This is usually set to be equal to the total number of LEDs to cater to the worst case scenario whereby each LED group contains only one LED.

dwMaxNumLedsInEachGroup [IN]

Type: DWORD

Set this to specify the maximum possible number of LEDs in each group.

Note: This is usually set to be equal to the total number of LEDs to cater to the worst case scenario whereby there is only 1 LED group and all the LEDs are in that LED group.

dwMaxNumColourLayersInEachGroup [IN]

Type: DWORD

Set this to specify the maximum possible number of colour layers in each LED group.

pLedInfo [OUT]

Type: PCTColourLayerForMultipleLedGroups

Upon successful return from the function, this will contain the requested data.

13.5 CTLEDINFOCMDPARAM_FillupAll

Parameters

dwNumLedGroups [IN]
Type: DWORD
Set this to specify the desired number of LED groups.

pPatternIndividualLedGroupArray1D [IN]
Type: Pointer to CTLED_Pattern
Set this to point to the 1 dimensional array of CTLED_Pattern for the LED grouping.

dwNumColumnsOfLedGroupingArray2D [IN]
Type: DWORD
Set this to specify the number of columns of the 2 dimensional array for the LED grouping that is specified by pdwLedGroupingArray2D.

pdwLedGroupingArray2D [IN]
Type: PDWORD
Set this to point to the 2 dimensional array of DWORD for the LED grouping.

dwNumColourLayers [IN]
Type: DWORD
Set this to specify the desired number of colour layers.

dwNumColumnsOfColourIndividualLedGroupArray2D [IN]
Type: DWORD
Set this to specify the number of columns of the 2 dimensional array for the colour layers of the LED grouping that is specified by pColourIndividualLedGroupArray2D.

pColourIndividualLedGroupArray2D [IN]
Type: PCTColour
Set this to point to the 2 dimensional array of CTColour for the colour layers of the LED grouping.

pLedInfo [OUT]
Type: PCTColourLayerForMultipleLedGroups
Upon successful return from the function, this will contain the requested data.

13.6 CTLEDINFOCMDPARAM_FillupNumLedGroups

Parameters

dwNumLedGroups [IN]
Type: DWORD
Set this to specify the desired number of LED groups.

pLedInfo [OUT]
Type: PCTColourLayerForMultipleLedGroups
Upon successful return from the function, this will contain the requested data.

13.7 CTLEDINFOCMDPARAM_FillupGroupLedPattern

Parameters

dwLedGroupIndex [IN]
Type: DWORD
Set this to specify the desired LED group index.

patternLed
Type: CTLED_Pattern
Set this to specify the desired LED pattern.

pLedInfo [OUT]
Type: PCTColourLayerForMultipleLedGroups
Upon successful return from the function, this will contain the requested data.

13.8 CTLEDINFOCMDPARAM_FillupNumLedsInGroup

Parameters

dwLedGroupIndex [IN]
Type: DWORD
Set this to specify the desired LED group index.

dwNumLedsInGroup
Type: DWORD
Set this to specify the desired number of LEDs in the group.

pLedInfo [OUT]
Type: PCTColourLayerForMultipleLedGroups
Upon successful return from the function, this will contain the requested data.

13.9 CTLEDINFOCMDPARAM_FillupLedID

Parameters

dwLedGroupIndex [IN]
Type: DWORD
Set this to specify the desired LED group index.

dwLedIndex
Type: DWORD
Set this to specify the desired LED index.

dwLedID
Type: DWORD
Set this to specify the desired LED ID.

pLedInfo [OUT]
Type: PCTColourLayerForMultipleLedGroups
Upon successful return from the function, this will contain the requested data.

13.10 CTLEDINFOCMDPARAM_FillupNumLedColourLayers

Parameters

dwLedGroupIndex [IN]
Type: DWORD
Set this to specify the desired LED group index.

dwNumColourLayers
Type: DWORD
Set this to specify the desired number of colour layers.

pLedInfo [OUT]
Type: PCTColourLayerForMultipleLedGroups
Upon successful return from the function, this will contain the requested data.

13.11 CTLEDINFOCMDPARAM_FillupLedColour

Parameters

dwLedGroupIndex [IN]
Type: DWORD
Set this to specify the desired LED group index.

`dwColourLayerIndex`
Type: DWORD
Set this to specify the desired colour layer index.

colourLayer
Type: CTColour
Set this to specify the desired colour.

pLedInfo [OUT]
Type: PCTColourLayerForMultipleLedGroups
Upon successful return from the function, this will contain the requested data.

13.12 CTLEDINFOCMDPARAM_Cleanup

Parameters

dwMaxNumLedGroups [IN]
Type: DWORD
Set this to the same value as was set earlier in the parameter dwMaxNumLedGroups of the data structure CTLEDINFOCMDPARAM_Initialize.

pLedInfo [OUT]
Type: PCTColourLayerForMultipleLedGroups
Upon successful return from the function, this will contain the requested data.

13.13 CTLEDMGRTIMERPROCDATA

Parameters

dwTimerPeriodicTimeInMilliseconds [IN]

Type: DWORD

Contains the periodic time in milliseconds of the timer callback that the timer is configured to. However, there is no guarantee that the timer callback will always be triggered precisely at this configured periodic timing.

llCurrentTimeTickInMilliseconds [IN]

Type: LONGLONG

Contains the current time stamp of the computer system in terms of time ticks in milliseconds.

llTotalTimeElapsedInMilliseconds [IN]

Type: LONGLONG

Contains the total time that has elapsed in milliseconds since the registered timer callback started.

Note: The first timer callback of a newly registered timer callback will always start with llTotalTimeElapsedInMilliseconds equals to zero.

13.14 CTTIMERINFOPARAM

Parameters

dwDueTimeInMilliseconds [IN]

Type: DWORD

Specifies the amount of time in milliseconds relative to the current time that must elapse before the timer callback is triggered for the first time.

dwPeriodicTimeInMilliseconds [IN]

Type: DWORD

On entry, this specifies the desired period of the timer, in milliseconds.
On exit, this contains the actual period of the timer used, in milliseconds. This will be equal to or greater than the desired value.

13.15 CTLEDINFOPARAM_GetWaveColourInfo

Parameters

`modePattern` [IN]

Type: CTLED_PatternMode

The mode of the LED pattern. Set this parameter to one of the following CTLED_PatternMode values:

- CTLED_PatternMode_Wave_Default
- CTLED_PatternMode_Wave_DiscreteFlow
- CTLED_PatternMode_Pulsate_Default
- CTLED_PatternMode_Pulsate_AlternateIntensityDirection
- CTLED_PatternMode_Pulsate_IntensityGradient
- CTLED_PatternMode_ColourCycle_Default
- CTLED_PatternMode_Aurora_Default
- CTLED_PatternMode_Static_Default

cmdLedGrouping [IN]

Type: CTLEDGROUPINGCMD

The LED grouping of the LED pattern. Set this parameter to one of the following CTLEDGROUPINGCMD values:

- CTLEDGROUPINGCMD_ByOneGlobalLedGroup
- CTLEDGROUPINGCMD_ByDesiredNumLedGroups_xAxis
- CTLEDGROUPINGCMD_ByDesiredNumLedGroups_yAxis
- CTLEDGROUPINGCMD_ByDesiredNumLedGroups_xAxis_Mirror
- CTLEDGROUPINGCMD_ByDesiredNumLedGroups_yAxis_Mirror

dwLedGroupIndex [IN]

Type: DWORD

Set this to specify the LED group index.

dwTotalNumLedGroups [IN]

Type: DWORD

Set this to specify the total number of LED groups.

dwPatternPeriodicTimeInMilliseconds [IN]

Type: DWORD

Set this to the periodic time of the LED pattern in milliseconds.

Note: Set this to 0 for the following LED patterns whereby periodic time is not applicable:

- CTLED_Pattern_Static

llPatternTimeElapsedInMilliseconds [IN]

Type: LONGLONG

Set this to the time elapsed of the LED pattern in milliseconds.

dwNumColourLayers [IN]

Type: DWORD

Set this to specify the number of colour layers in pCTColourOriginalArray.

Note: Set this to 0 for the following LED patterns whereby the colours are predefined and cannot be changed:

- CTLED_Pattern_Aurora
- CTLED_pattern_ColourCycle

pCTColourOriginalArray [IN]

Type: PCTColour

Set this to the array of original colours of the LED pattern.

Note: Set this to NULL for the following LED patterns whereby the colours are predefined by the SDK library and cannot be defined by client apps:

- CTLED_Pattern_Aurora
- CTLED_pattern_ColourCycle

colourCurrent [OUT]
Type: CTColour
Upon successful return from the function, this contains the current colour to be set.

13.16 CTLEDINFOPARAM_GetStaticColourInfo

Currently, this data structure is identical to the data structure CTLEDINFOPARAM_GetWaveColourInfo.

Please refer to the description for the data structure CTLEDINFOPARAM_GetWaveColourInfo.

13.17 CTLEDINFOPARAM_GetPulsateColourInfo

Currently, this data structure is identical to the data structure CTLEDINFOPARAM_GetWaveColourInfo.

Please refer to the description for the data structure CTLEDINFOPARAM_GetWaveColourInfo.

13.18 CTLEDINFOPARAM_GetAuroraColourInfo

Currently, this data structure is identical to the data structure CTLEDINFOPARAM_GetWaveColourInfo.

Please refer to the description for the data structure CTLEDINFOPARAM_GetWaveColourInfo.

13.19 CTLEDINFOPARAM_GetColourCycleColourInfo

Currently, this data structure is identical to the data structure CTLEDINFOPARAM_GetWaveColourInfo.

Please refer to the description for the data structure CTLEDINFOPARAM_GetWaveColourInfo.