AURA Software Development Kit
Developer's Guide

Version 2.0.0
October 6, 2017

Table of Contents

Version History

Version	Date	Maintained by	Comments
1.0.0	2017/4/20	Po-Ling Chang	Initial release.
2.0.0	2017/10/6	Po-Ling Chang	Revised for the major revision of the SDK.

Introduction

This guide describes how to use the AURA Software Development Kit (SDK).

The purpose of this guide is to provide the information to third parties for controlling the LED lights embedded on the AURA-enabled motherboards and the peripherals such as VGA cards, keyboards, mice, and so on. This SDK is provided as a set of APIs in the form of the dynamic linking library (DLL) to conform to the Win32 platforms. Besides the SDK DLL file, several additional DLL files for the AURA-enabled devices, which are actually perform the underlying control procedures to the devices, are also attached in the package. The need to load exact DLL files for a specific device is vital as the developer is controlling the device through the SDK APIs. The underlying DLL file(s) for a specific device would be loaded automatically in the SDK as the developer is taking advantage of the APIs for the device. While the other device-dependent DLL files, which are not required for the target device(s) to an application, can be ignore from the release package of the application.

Prerequisites

- **Supported operating systems**
  - Windows 7 32-bit and 64-bit
  - Windows 8 / Windows 8.1 32-bit and 64-bit
  - Windows 10 32-bit and 64-bit

- **Development tools and programming languages**
  - Support to load and leverage Win32 DLL files, such as Microsoft Visual Studio C++.

Terminology

- **The LED lights**

  The LED lights ("light" hereafter) is the actual LEDs performing the lighting effects. There are two kinds of lights mentioned here: one kind is welded directly on the motherboard or on the peripherals, and the other (called the LED strip) is connected and extended through the RGB Header(s) on the motherboard. Some peripherals, such as a keyboard, require additional accompanying run-time DLL file(s) to enable the actual control of the lights. These DLLs can be found with the SDK and must be shipped with the applications.

- **The lighting effects**

  The lighting effects ("effect" hereafter) are the effects that can be performed through the APIs issued from the application to the lights on different connecting peripherals. The effect can be designated by the combination of algorithms and time-variant-or-invariant input data prepared and/or fetched in the application.

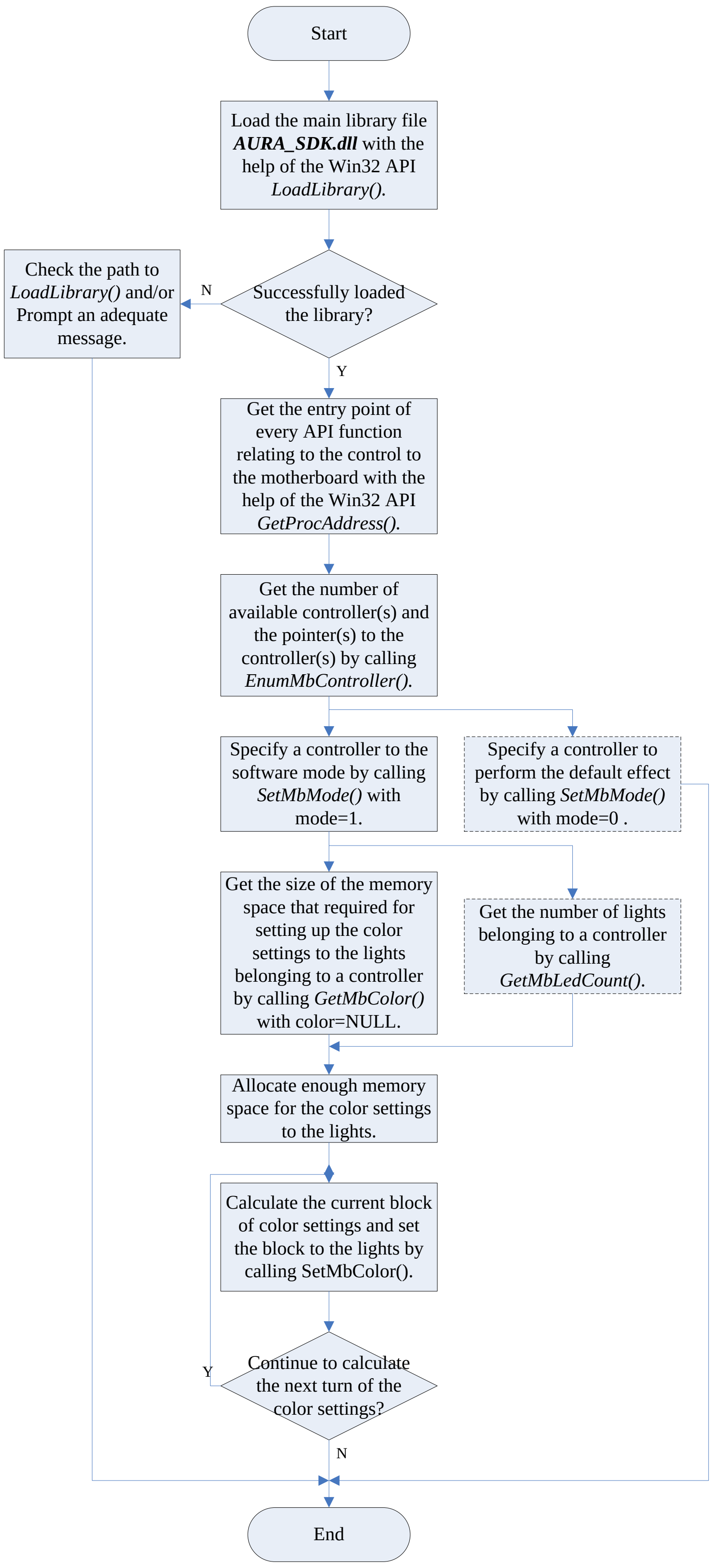
- **The fundamental capability**

  The fundamental capability ("capability" hereafter) depicts the basic information, which is vital in order to issue the correct control command to the correct control unit(s) of a peripheral. The capability information, which the SDK currently provides, mainly includes the number of the control units of a peripheral and the number of the lights held by a control unit.

- **The application programming interface (API) for AURA**

  The communication between the application and the lights on a peripheral is through a set of API, which is packaged in a Win32 dynamic linking library (DLL) file. This API is built for the independent software vendors (ISVs), the technology enthusiasts, and anyone who is fascinated by the colorful effects performed by the AURA software and willing to demonstrate their innovation through the RGB LED lights.

The Reference Flow



Start
Load the main library file *AURA_SDK.dll* with the help of the Win32 API *LoadLibrary()*.
Successfully loaded the library?
N
Check the path to *LoadLibrary()* and/or Prompt an adequate message.
Y
Get the entry point of every API function relating to the control to the motherboard with the help of the Win32 API *GetProcAddress()*.
Get the number of available controller(s) and the pointer(s) to the controller(s) by calling *EnumMbController()*.
Specify a controller to the software mode by calling *SetMbMode()* with mode=1.
Specify a controller to perform the default effect by calling *SetMbMode()* with mode=0 .
Get the size of the memory space that required for setting up the color settings to the lights belonging to a controller by calling *GetMbColor()* with color=NULL.
Get the number of lights belonging to a controller by calling *GetMbLedCount()*.
Allocate enough memory space for the color settings to the lights.
Calculate the current block of color settings and set the block to the lights by calling SetMbColor().
Continue to calculate the next turn of the color settings?
Y
N
End

This reference flow is to provide a demonstrative flow of control while developing the application for the use of the LED lighting system. Since the control flow to a specific device that the API currently supports is similar to the others, we use the flow to a motherboard as a reference to demonstrate what functions can be done and which API can be invoked to complete a specific purpose. The frame with solid lines depicts a step that is necessary to provide the basic functionality to control the lighting system, while the frame with dashed lines depicts the steps that can be implementation-dependent to the developers' functional requirements.

The AURA API Reference

The API, as described above in “The application programming interface (API) for AURA” subsection, is packaged into a DLL file and ready to be loaded and used to control the RGB LED lights embedded on a diversity of AURA-enabled devices. The types of the AURA-enabled devices which this API currently supports are motherboards, VGA cards, keyboards, mice, and DRAM modules. Each type of the devices can be controlled with a subset of the API, which is designed to hold similar function signatures with each other. We hope to provide the developers with a simplified and consistent experience while using this API.

The API for each of the individual device types will now be listed in each of their subsections as below.

- ### The API for motherboards

  - #### *EnumerateMbController()*

This function can be used to obtain the LED controller(s) that is the interface to access the controller’s underlying RGB LED lights.

Syntax

```
DWORD WINAPI EnumerateMbController(
    _In_opt_    void*       handles[],
    _In_out_    DWORD       size
);
```

Parameters

handles[]   [in, optional]

A well-allocated array can be passed in, and this function would then fill in the pointers to the LED controllers found on the motherboard, in turn they can be used to access their respective underlying RGB LED lights.

This parameter can be **NULL**.

If this parameter is **NULL**, this function will return the number of the controllers directly and no pointers would be passed out through this parameter.

size   [in, out]

The actual length of the array passed in through the *handles[]* parameter. If 0 is passed in or no controller found on the motherboard, 0 would be returned directly by this function.

The actual number of controllers found by this function would be passed out through this parameter.

Return value

The return value is the number of the controllers found on the motherboard if the function is executed normally with valid input parameters or the *handles[]* parameter is **NULL**.

The return value is 0 if 0 is passed in through the *size* parameter or no controller is found.

Remark

The allocated vacancies in order to store the pointers to controllers should be not less than the actual number of the controllers on the motherboard or the consequences would be undetermined. The best practice is to query the number of the controllers first by calling this function with passing in **NULL** through the *handles[]* parameter, and then calling this function again with well-allocated memory space for storing the obtained pointers.

■ *SetMbMode()*

This function is used to set a controller to the specified control mode.

Syntax

```
DWORD WINAPI SetMbMode(
    _In_   void*      handle,
    _In_   DWORD      mode
);
```

Parameters

handle   [in]

A valid pointer to a controller whose control mode is about to be changed.

mode [in]

The mode to be set to the controller specified by the *handle* parameter. It can contain one of the following values.

Mode	Meaning
0	Sets the controller to the EC mode and in turn to set the lights to perform the default lighting effect. The default effect varies from motherboard to motherboard.
1	Sets the controller to the software mode. The behavior of the lights belonging to the controller can be totally determined by the software itself.

Return value

The function returns 1 if succeeds, 0 if the *handle* parameter is **NULL**, or -1 if the *mode* parameter is neither 1 nor 0.

■ *GetMbLedCount()*

This function is used to obtain the number of RGB LED lights hold by the specified controller.

Syntax

```
DWORD WINAPI GetMbLedCount(
   _In_   void*        handle
);
```

Parameters

handle [in]

A valid pointer to a controller.

Return value

The number of lights hold by the specified controller will be returned if succeeds, 0 otherwise.

■ *GetMbColor()*

This function is used to obtain the current color settings to all the lights hold by the specified controller. The color setting to a light is three bytes in length, and in turn each of the bytes contains one value of the red, the green, or the blue values.

Syntax

```
DWORD WINAPI GetMbMbColor(
   _In_         void*        handle,
   _In_opt_     BYTE*        color,
   _In_out_     DWORD        size
);
```

Parameters

handle [in]

A valid pointer to a controller.

color [in, optional]

A well-allocated memory space can be passed in, and then this function fills in the color settings to the lights hold by the specified controller.

This parameter can be **NULL**.

If this parameter is **NULL**, this function will return the total number of bytes required to store the color settings to the lights hold by the specified controller.

size [in, out]

This parameter contains the length of the memory space, which is passed in through the *color* parameter. The actual number of the color settings in byte obtained by this function would be passed out through this parameter.

Return value

The size of the color settings in byte exactly required by the lights will be returned if succeeds, or 0 will be returned if the *handle* parameter is **NULL**. -1 otherwise.

Remark

The allocated vacancies in order to store the color settings to the lights should be not less than the actual number of bytes required to store the whole settings to all of the lights or the consequences would be undetermined. The best practice is to query the number of bytes required to store the whole settings first by calling this function with passing in **NULL** through the *color* parameter, and then calling this function again with well-allocated memory space for storing the obtained settings.

■ *SetMbColor()*

This function is used to set a block of color settings to all of the lights hold by the specified controller. The color setting to a light is three bytes in length, and in turn each of the bytes contains one value of the red, the green, or the blue values.

Syntax

```
DWORD WINAPI GetMbMbColor(
   _In_        void*        handle,
   _In_        BYTE*       color,
   _In_        DWORD     size
);
```

Parameters

handle [in]

A valid pointer to a controller.

color [in]

A well-allocated memory space that is prepared with determined color settings for all of the lights hold by the specified controller. The allocated memory space must be prepared exactly to the same size in byte required by all of the lights of the controller, and the *size* parameter must be given the value as the exact size of this prepared memory space.

size [in]

This parameter contains the length of the memory space, which is passed in through the *color* parameter. This parameter must be given the value as the exact size of the memory space for the color settings in byte required by all of the lights of the controller. If this parameter contains a value that is not equal to the exact size of the memory space required for setting the color settings, -1 will be returned from this function.

Return value

The function returns 1 if succeeds, or -1 if the *size* parameter contains a value that is not equal to the exact size of the memory space required for setting the color settings.

Remark

The size of the allocated memory space, required for storing the prepared color settings and passed in through the *color* parameter, must be the same as the actual number of bytes required for the whole settings to all of the lights or the consequences would be undetermined. The best practice is to query the number of bytes required to store the whole settings by calling *GetMbColor()* function with passing in **NULL** through the *color* parameter, and then allocating the required memory space exactly as the same size of the returned value of *GetMbColor()* function for the further code flow.

- ## The API for VGA cards

■ *EnumerateGPU()*

This function can be used to obtain an array of pointers to the LED controller(s) on VGA card(s) that are the interfaces to access the controllers' underlying RGB LED lights.

Syntax

```
DWORD WINAPI EnumerateGPU (
   _In_opt_   void*       handles[],
   _In_out_   DWORD    size
);
```

Parameters

handles[]   [in, optional]

A well-allocated array can be passed in, and this function would then fill in the pointers to the LED controllers found on VGA cards, in turn they can be used to access their respective underlying RGB LED lights.

This parameter can be **NULL**.

If this parameter is **NULL**, this function will return the number of the controllers directly and no pointers would be passed out through this parameter.

size [in, out]

The actual length of the array passed in through the *handles[]* parameter. If 0 is passed in or no controller found on any VGA card, 0 would be returned directly by this function.

The actual number of controllers found by this function would be passed out through this parameter.

Return value

The return value is the number of the controller(s) found on the VGA card(s) if the function is executed normally with valid input parameters or the *handles[]* parameter is **NULL**.

The return value is 0 if 0 is passed in through the *size* parameter or no controller is found.

Remark

The allocated vacancies in order to store the pointers to controllers should be not less than the actual number of the controller(s) on the VGA card(s) or the consequences would be undetermined. The best practice is to query the number of the controllers first by calling this function with passing in **NULL** through the *handles[]* parameter, and then calling this function again with well-allocated memory space for storing the obtained pointers.

■ *SetGPUMode()*

This function is used to set a controller to the specified control mode.

Syntax

```
DWORD WINAPI SetGPUMode (
   _In_   void*         handle,
    _In_    DWORD    mode
);
```

Parameters

handle [in]

A valid pointer to a controller whose control mode is about to be changed.

mode [in]

The mode to be set to the controller specified by the *handle* parameter. This function only accepts 0 for setting the specified controller to perform the default lighting effect. Passing in the other values won't take effect to the mode of the specified controller.

Return value

The function always returns 1.

■ *GetGPULedCount()*

This function is used to obtain the number of RGB LED lights hold by the specified controller.

Syntax

```
DWORD WINAPI GetGPULedCount (
    _In_    void*        handle
);
```

Parameters

handle [in]

A valid pointer to a controller.

Return value

The number of lights hold by the specified controller will be returned.

■ *SetGPUColor()*

This function is used to set a block of color settings to all of the lights hold by the specified controller. The color setting to a light is three bytes in length, and in turn each of the bytes contains one value of the red, the green, or the blue values.

Syntax

```
DWORD WINAPI SetGPUColor (
   _In_        void*        handle,
   _In_        BYTE*       color,
   _In_        DWORD    size
);
```

Parameters

handle   [in]

A valid pointer to a controller.

color   [in]

A well-allocated memory space that is prepared with determined color settings for all of the lights hold by the specified controller. The allocated memory space must be prepared exactly to the same size in byte required by all of the lights of the controller, and the *size* parameter must be given the value as the exact size of this prepared memory space.

size   [in]

This parameter contains the length of the memory space, which is passed in through the *color* parameter. This parameter must be given the value as the exact size of the memory space for the color settings in byte required by all of the lights of the controller. If this parameter contains a value that is not equal to the exact size of the memory space required for setting the color settings, -1 will be returned from this function.

Return value

The function returns 1 if succeeds, or -1 if the *size* parameter contains a value that is not equal to the exact size of the memory space required for setting the color settings.

Remark

The size of the allocated memory space, required for storing the prepared color settings and passed in through the *color* parameter, must be the same as the actual number of bytes required for the whole settings to all of the lights or the consequences would be undetermined. The best practice is to query the number of lights hold by a controller by calling *GetGPULedCount()* first, and then allocating the required memory space exactly as the same size of the triple of the returned value from *GetGPULedCount()* function for the further code flow.

● The API for keyboards

■ *CreateClaymoreKeyboard()*

This function can be used to obtain the pointer to the LED controller on an AURA-enabled keyboard. The found controller can in turn be used to access its underlying RGB LED lights.

Syntax

```
DWORD WINAPI CreateClaymoreKeyboard (
   _In_out_   void**         handle
);
```

Parameters

handle   [in, out]

A well-allocated pointer to pointer can be passed in, and this function would then fill in the pointer to the LED controller found on a keyboard, in turn the controller can be used to access its underlying RGB LED lights.

Return value

The return value is the value greater than 0 if the controller can be found on an AURA-enabled keyboard, or 0 otherwise.

■ *SetClaymoreKeyboardMode()*

This function is used to set the controller to the specified control mode.

Syntax

```
DWORD WINAPI SetClaymoreKeyboardMode (
    _In_    void*       handle,
    _In_    DWORD     mode
);
```

Parameters

handle [in]

A valid pointer to a controller whose control mode is about to be changed.

mode [in]

The mode to be set to the controller specified by the *handle* parameter. It can contain one of the following values.

Mode	Meaning
0	Sets the controller to the EC mode and in turn to set the lights to perform the default lighting effect.
1	Sets the controller to the software mode. The behavior of the lights belonging to the controller can be totally determined by the software itself.

Return value

The function returns 1 if succeeds, 0 if the *handle* parameter is **NULL**, or -1 if the *mode* parameter is neither 1 nor 0.

■ *GetClaymoreKeyboardLedCount()*

This function is used to obtain the number of RGB LED lights hold by the controller.

Syntax

```
DWORD WINAPI GetClaymoreKeyboardLedCount (
    _In_    void*       handle
);
```

Parameters

handle [in]

A valid pointer to a controller.

Return value

The number of lights hold by the controller will be returned, 0 otherwise.

■ *SetClaymoreKeyboardColor()*

This function is used to set a block of color settings to all of the lights hold by the controller. The color setting to a light is three bytes in length, and in turn each of the bytes contains one value of the red, the green, or the blue values.

Syntax

```
DWORD WINAPI SetClaymoreKeyboardColor (
   _In_        void*        handle,
   _In_        BYTE*       color,
   _In_        DWORD    size
);
```

Parameters

handle   [in]

A valid pointer to a controller.

color   [in]

A well-allocated memory space that is prepared with determined color settings for all of the lights hold by the controller. The allocated memory space must be prepared exactly to the same size in byte required by all of the lights of the controller, and the *size* parameter must be given the value as the exact size of this prepared memory space.

size   [in]

This parameter contains the length of the memory space, which is passed in through the *color* parameter. This parameter must be given the value as the exact size of the memory space for the color settings in byte required by all of the lights of the controller. If this parameter contains a value that is not equal to the exact size of the memory space required for setting the color settings, -1 will be returned from this function.

Return value

The function returns 1 if succeeds, or -1 if the *size* parameter contains a value that is not equal to the exact size of the memory space required for setting the color settings.

Remark

The size of the allocated memory space, required for storing the prepared color settings and passed in through the *color* parameter, must be the same as the actual number of bytes required for the whole settings to all of the lights or the consequences would be undetermined. The best practice is to query the number of lights hold by the controller by calling *GetClaymoreKeyboardLedCount()* first, and then allocating the required memory space exactly as the same size of the triple of the returned value from *GetClaymoreKeyboardLedCount()* function for the further code flow.

● The API for mice

■ *CreateRogMouse()*

This function can be used to obtain the pointer to a LED controller, which contains the information to all of the found AURA-enabled mice. The controller can in turn be used to access the underlying RGB LED lights on the mice.

Syntax

```
DWORD WINAPI CreateRogMouse (
   _In_out_   void**        handle
);
```

Parameters

handle [in, out]

A well-allocated pointer to pointer can be passed in, and this function would then fill in the pointer to the LED controller containing the information of all the found mice. The controller, in turn, can be used to access the underlying RGB LED lights on the mice.

Return value

The return value is the value greater than 0 if the controller can be returned from the *handle* parameter, or 0 otherwise.

■ *SetRogMouseMode()*

This function is used to set a controller to the specified control mode.

Syntax

```
DWORD WINAPI SetRogMouseMode (
    _In_    void*       handle,
    _In_    DWORD       mode
);
```

Parameters

handle [in]

A valid pointer to a controller whose control mode is about to be changed.

mode [in]

The mode to be set to the controller specified by the *handle* parameter. This function only accepts 0 for setting the specified controller to perform the default lighting effect. Passing in the other values won't take effect to the mode of the specified controller.

Return value

The function returns 1 if succeeded, 0 if the *handle* parameter is **NULL**.

■ *RogMouseLedCount()*

This function is used to obtain the number of RGB LED lights hold by the controller.

Syntax

```
DWORD WINAPI RogMouseLedCount (
    _In_    void*       handle
);
```

Parameters

handle [in]

A valid pointer to a controller.

Return value

The number of lights hold by the controller will be returned, 0 otherwise.

■ *SetRogMouseColor()*

This function is used to set a block of color settings to all of the lights hold by the controller. The color setting to a light is three bytes in length, and in turn each of the bytes contains one value of the red, the green, or the blue values.

Syntax

```
DWORD WINAPI SetRogMouseColor (
   _In_      void*      handle,
   _In_      BYTE*      color,
   _In_      DWORD      size
);
```

Parameters

handle   [in]

A valid pointer to a controller.

color   [in]

A well-allocated memory space that is prepared with determined color settings for all of the lights hold by the controller. The allocated memory space must be prepared exactly to the same size in byte required by all of the lights of the controller, and the *size* parameter must be given the value as the exact size of this prepared memory space.

size   [in]

This parameter contains the length of the memory space, which is passed in through the *color* parameter. This parameter must be given the value as the exact size of the memory space for the color settings in byte required by all of the lights of the controller. If this parameter contains a value that is not equal to the exact size of the memory space required for setting the color settings, -1 will be returned from this function.

Return value

The function returns 1 if succeeds, or -1 if the *size* parameter contains a value that is not equal to the exact size of the memory space required for setting the color settings.

Remark

The size of the allocated memory space, required for storing the prepared color settings and passed in through the *color* parameter, must be the same as the actual number of bytes required for the whole settings to all of the lights or the consequences would be undetermined. The best practice is to query the number of lights hold by the controller by calling *RogMouseLedCount()* first, and then allocating the required memory space exactly as the same size of the triple of the returned value from *RogMouseLedCount()* function for the further code flow.

● The API for DRAM modules

■ *EnumerateDram()*

This function can be used to obtain an array of pointers to the LED controller on each of the DRAM module(s). The controller(s), in turn, can be used to access the underlying RGB LED lights on the DRAM module(s).

Syntax

```
DWORD WINAPI EnumerateDram (
    _In_opt_    void*       handles[],
    _In_out_    DWORD       size
);
```

Parameters

handles[] [in, optional]

A well-allocated array can be passed in, and this function would then fill in the pointers to the LED controllers found on the motherboard, in turn they can be used to access their respective underlying RGB LED lights.

This parameter can be **NULL**.

If this parameter is **NULL**, this function will return the number of the controllers directly and no pointers would be passed out through this parameter.

size   [in, out]

The actual length of the array passed in through the *handles[]* parameter. If 0 is passed in or no controller found on the motherboard, 0 would be returned directly by this function.

The actual number of controllers found by this function would be passed out through this parameter.

Return value

The return value is the number of the controllers found on the motherboard if the function is executed normally with valid input parameters or the *handles[]* parameter is **NULL**.

The return value is 0 if 0 is passed in through the *size* parameter or no controller is found.

Remark

The allocated vacancies in order to store the pointers to controllers should be not less than the actual number of the controllers on the motherboard or the consequences would be undetermined. The best practice is to query the number of the controllers first by calling this function with passing in **NULL** through the *handles[]* parameter, and then calling this function again with well-allocated memory space for storing the obtained pointers.

Currently the API only supports the Trident Z RGB series by G.SKILL International Enterprise.

■ *SetDramMode()*

This function is used to set a controller to the specified control mode.

Syntax

```
DWORD WINAPI SetDramMode (
   _In_   void*       handle,
   _In_   DWORD       mode
);
```

Parameters

handle [in]

A valid pointer to a controller whose control mode is about to be changed.

mode [in]

The mode to be set to the controller specified by the *handle* parameter. It can contain one of the following values.

Mode	Meaning
0	Sets the controller to the EC mode and in turn to set the lights to perform the default lighting effect.
1	Sets the controller to the software mode. The behavior of the lights belonging to the controller can be totally determined by the software itself.

Return value

The function returns 1 if succeeds, 0 if the *handle* parameter is **NULL**, or -1 if the *mode* parameter is neither 1 nor 0.

■ *GetDramLedCount()*

This function is used to obtain the number of RGB LED lights hold by the specified controller.

Syntax

```
DWORD WINAPI GetDramLedCount (
   _In_   void*       handle
);
```

Parameters

handle [in]

A valid pointer to a controller.

Return value

The number of lights hold by the specified controller will be returned if succeeds, 0 otherwise.

■ *GetDramColor()*

This function is used to obtain the current color settings to all the lights hold by the specified controller. The color setting to a light is three bytes in length, and in turn each of the bytes contains one value of the red, the green, or the blue values.

Syntax

```
DWORD WINAPI GetDramColor (
   _In_        void*       handle,
   _In_opt_    BYTE*       color,
   _In_out_    DWORD       size
);
```

Parameters

handle [in]

A valid pointer to a controller.

color [in, optional]

A well-allocated memory space can be passed in, and then this function fills in the color settings to the lights hold by the specified controller.

This parameter can be **NULL**.

If this parameter is **NULL**, this function will return the total number of bytes required to store the color settings to the lights hold by the specified controller.

size [in, out]

This parameter contains the length of the memory space, which is passed in through the *color* parameter. The actual number of the color settings in byte obtained by this function would be passed out through this parameter.

Return value

The size of the color settings in byte exactly required by the lights will be returned if succeeds, or 0 will be returned if the *handle* parameter is **NULL**. -1 otherwise.

Remark

The allocated vacancies in order to store the color settings to the lights should be not less than the actual number of bytes required to store the whole settings to all of the lights or the consequences would be undetermined. The best practice is to query the number of bytes required to store the whole settings first by calling this function with passing in **NULL** through the *color* parameter, and then calling this function again with well-allocated memory space for storing the obtained settings.

■ *SetDramColor()*

This function is used to set a block of color settings to all of the lights hold by the specified controller. The color setting to a light is three bytes in length, and in turn each of the bytes contains one value of the red, the green, or the blue values.

Syntax

```
DWORD WINAPI SetDramColor (
   _In_        void*        handle,
   _In_        BYTE*       color,
   _In_        DWORD     size
);
```

Parameters

handle [in]

A valid pointer to a controller.

color [in]

A well-allocated memory space that is prepared with determined color settings for all of the lights hold by the specified controller. The allocated memory space must be prepared exactly to the same size in byte required by all of the lights of the controller, and the *size* parameter must be given the value as the exact size of this prepared memory space.

size [in]

This parameter contains the length of the memory space, which is passed in through the *color* parameter. This parameter must be given the value as the exact size of the memory space for the color settings in byte required by all of the lights of the controller. If this parameter contains a value that is not equal to the exact size of the memory space required for setting the color settings, -1 will be returned from this function.

Return value

The function returns 1 if succeeds, or -1 if the *size* parameter contains a value that is not equal to the exact size of the memory space required for setting the color settings.

Remark

The size of the allocated memory space, required for storing the prepared color settings and passed in through the *color* parameter, must be the same as the actual number of bytes required for the whole settings to all of the lights or the consequences would be undetermined. The best practice is to query the number of bytes required to store the whole settings by calling *GetDramColor()* function with passing in **NULL** through the *color* parameter, and then allocating the required memory space exactly as the same size of the returned value of *GetDramColor()* function for the further code flow.