# MRM Smart Display User Manual

5/4/2017
SSID.SRP

# Table of contents

# 1 Introduction

## 1.1 Smart Display Overview

MRM SDP SDK is a set of libraries for controlling SDP( Smart Display co-Processor) which controls the smart display device.

MRM SDP SDK  is composed of the following API categories:

| Module | Feature |
|---|---|
| **Firmware Management** | Save/load the default settings of the smart display device. |
| **Backlight Control** | Functions for configuring the brightness level and duty cycle of backlight. |
| **Hotkey** | Get press status of hotkeys. Configure the LED brightness of hotkey. |
| **Peripheral Control** | Functions for configuring peripheral hardware components. |

MRM SDP SDK  supports the following platforms:

**For Windows / Linux:**

The MRM SDP SDK is designed as a library (**sdp.dll** or **sdp.so**)  for the customer's APP to load and access.

Prior to use SDP APIs, you should call **sdp_init()**. Normally the API should return 32-bits error code **MRM_ERR_NO_ERROR(0).** You must confirm the return value to ensure that the command is work. After calling **sdp_init(),** the library will initialize SDP and start serving APP's request.

**For Android:**

The MRM SDP SDK is designed as a background service which acts as a proxy for client APPs to access the SDP functions.

The MRM SDP API for Android includes three parts:

- MRM Services  ( **mrm_service.apk** )
- MRM Service Client APIs  ( **MrmServiceClientAPI.jar** )
- MRM Data Classes And Constants Definitions ( **MrmDef.jar** )

To use the SDP service, you must install MRM Services APK  to your device and import the above libraries in your APP project.

In your APP, you must call **sdp_bind_service()** to connect your APP process with the SDP Service before accessing the SDP APIs and call **sdp_unbind_service()** to disconnect service before your APP is destroyed.

## 1.2 System Requirements

Hardware:

- Smart Display Module

Operating system:

- Windows
- Linux
- Android

Recommended Development Tool

- Visual Studio 2008 or above
- GCC 4.6+
- Android Studio 1.3.2+

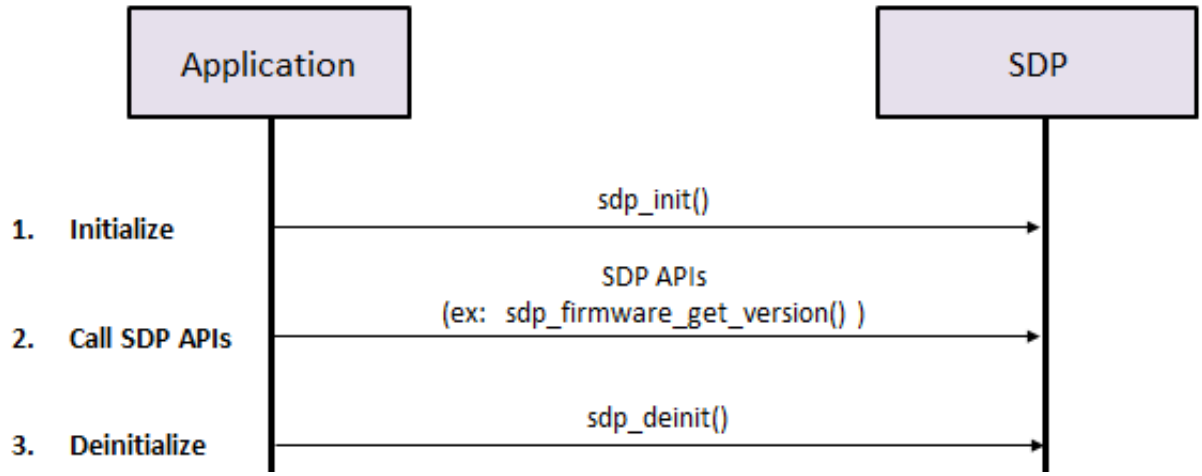
# 2 Using SDP

## 2.1 SDP Conventions

### 2.1.1 Windows/Linux

- **Basic API usage**
  - Each SDP API has a prefix "**sdp**_" immediately followed by the function name and the operation name.

  - To usage the SDP APIs, you must first initialize the library and deinitialize before your APP is closed.
    The flow is described as following:
    1. You must call **sdp_init()** before using the other SDP APIs.
    2. Call SDP APIs.
    3. You must call **sdp_deinit()** before you APP closed.




  - Any read type API using the pass by pointer. You should ensure the pointer be not released during the API processing and the pointer is valid.

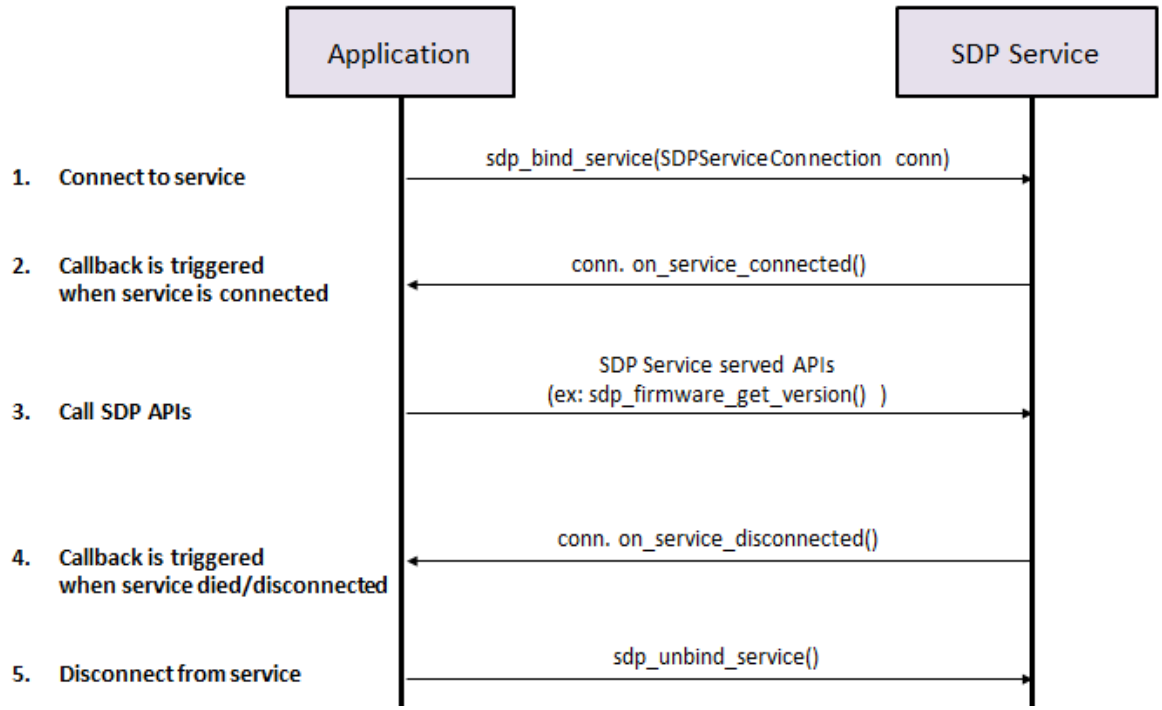
  - You should always check the return value equal **MRM_ERR_NO_ERROR(0)**, in order to ensure the command working.

### 2.1.2 Android

- **SDK Namespaces**
  - **mrm.client.SDPServiceClient**
    - The main class of MRM Service Client API. Use this class to access SDP features.
  - **mrm.client.SDPServiceConnection**
    - The service connection interface used by **sdp_bind_service**().
  - **mrm.define.SDP**
    - The definition of data structures used by MRM Service Client API.
  - **mrm.define.MRM_ENUM**
    - The definition of enumeration values used by MRM Service Client API.
  - **define.MRM_CONSTANTS**
    - The definition of constants used by MRM Service Client API.

- **Basic API usage**
  - Each SDP API has a prefix "**sdp**_" immediately followed by the function name and the operation name
  - You must create a instance of mrm.client.**SDPServiceClient** to usage SDP APIs .
  - To usage the SDP APIs, you must first "bind" your APP's process to the SDP service and "unbind" before your APP is closed.
    The flow is described as following:
    1. Create an instance of mrm.client.**SDPServiceConnection** and implement the interface **on_service_connected()** and **on_service_disconnected()**,
       The interface is used to will inform your APP when the service is connected/disconnected. Please refer to the document of sdp_bind_service() and SDP sample code for further details.
    2. Call **sdp_bind_service()** with created**SDP ServiceConnection** instance to connect your APP process to the SDP Service.
    3. Call SDP Served APIs
    4. Call **sdp_unbind_service()** before your APP's application context is destroyed.

    **NOTE: Please refer to the SDP sample code for the details.**




- APIs for reading data need an array for argument to store data. The array should be allocated before you pass it to the API and the data will be stored at **index 0 of the array.**
- You should always check the return value of APIs for error checking. The value should equal to MRM_ERR_NO_ERROR(0) when success or other value when failed.

- **SDP Service Behavior**
  - The SDP Service will be started when **sdp_bind_service()** is called and keep alive when client APP unbind.
  - The SDP Service might be stopped **by user manually** or **by system automatically (ex: when low memory)**.

## 2.2 C++ with Visual Studio

1. Right Click on Project to enter Project Properites
2. Add #include "sdp/sdp.h" to your program
3. Select **Linker** and **Input** page. Set Additional Dependencies "sdp.lib"

## 2.3 JAVA with Android Studio

1. **Install MRM SDP Service to your device**

   Please find the **mrm_service.apk** in the MRM SDK package.
   Connect you device to your computer with ADB, then execute the following ADB commands to install MRM SDP Service

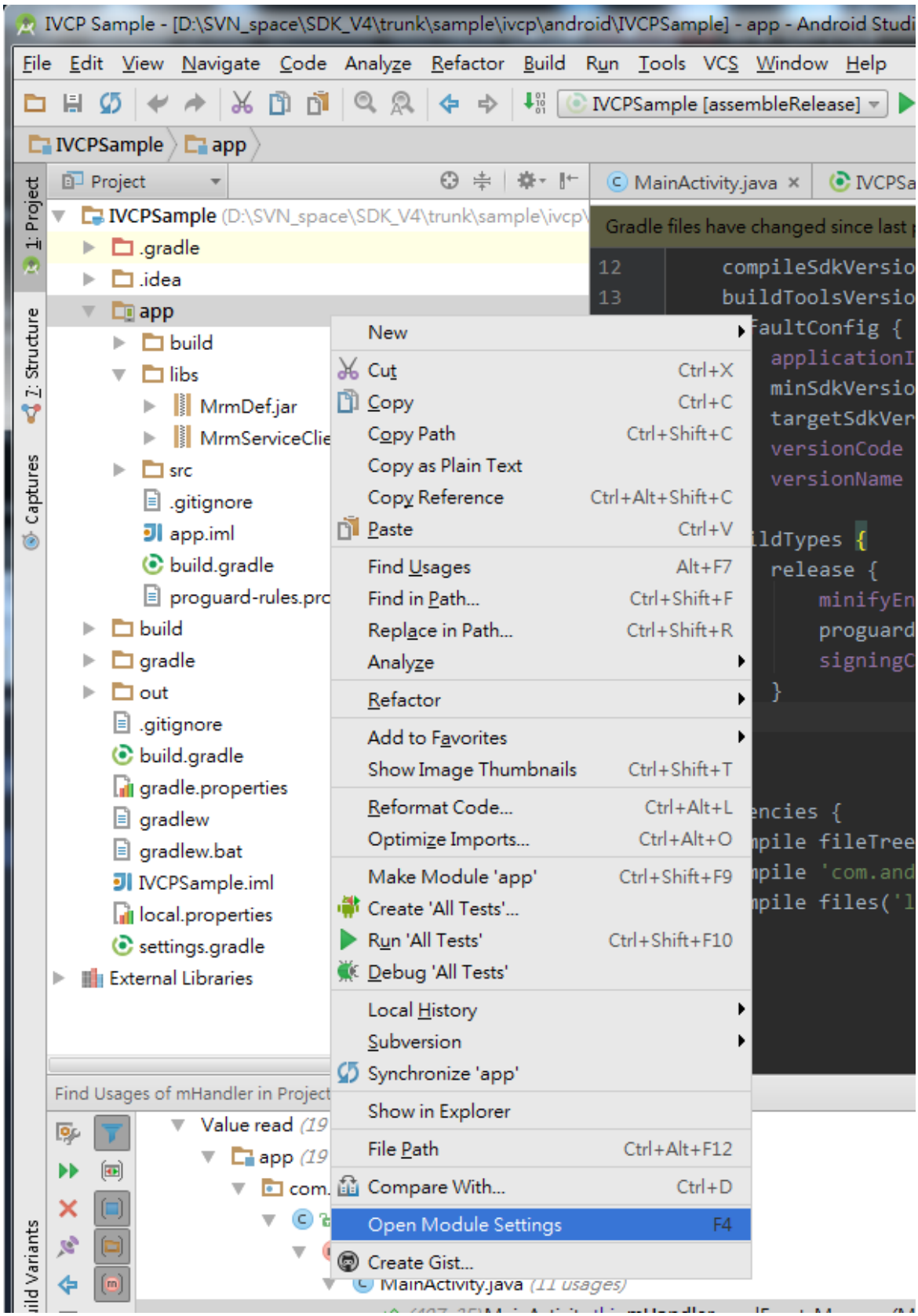
```
adb install .\bin\mrm_service.apk
```

2. **Import MRM SDP Service Client API library to your project**

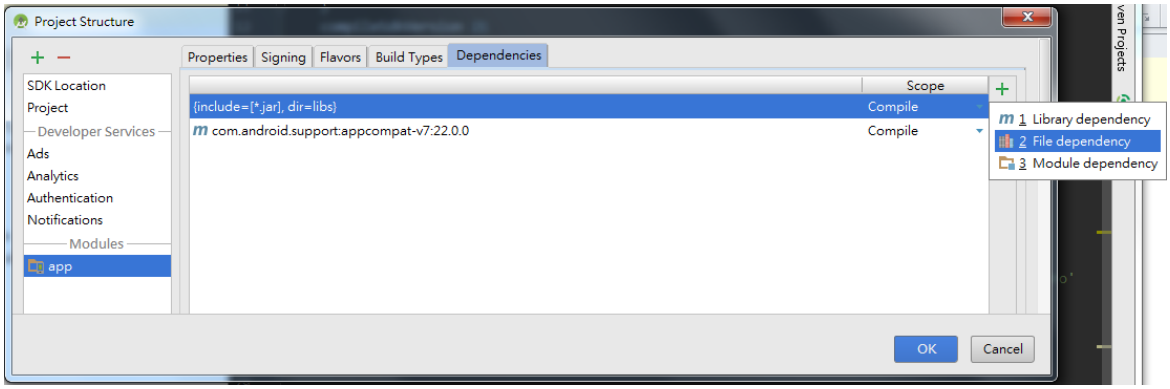
   To access MRM SDP Service, you must import the MRM SDP Service Client API lib into you project.

   Please find the **MrmServiceClientAPI.jar** and **MrmDef.jar** in the MRM SDK package. Copy the libraries to the directory **/[Module Name]/libs/** in you Android Studio project (the default module name might be "app").
   Then import the libraries by following the steps below:

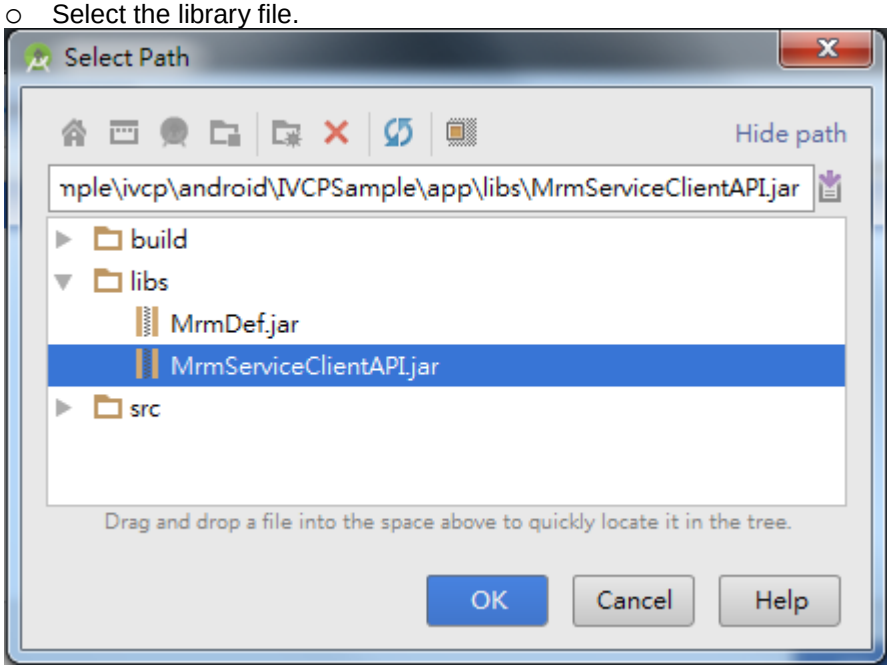
   - Right click on you APP module. Click "**Open module settings**"



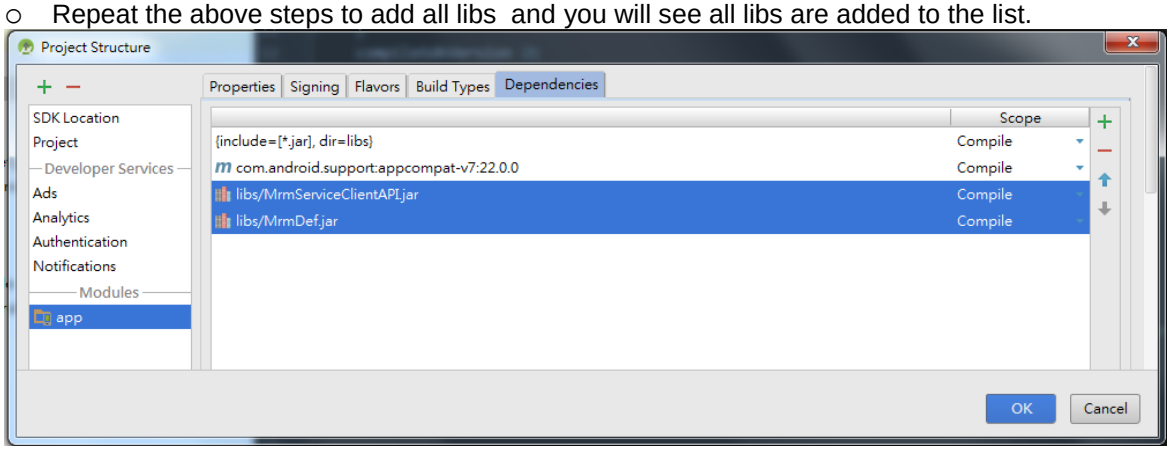

- Click the "**Dependency**" tab. Then click "**+**" -> "**Library dependency**"




- Select the library file.




- Repeat the above steps to add all libs and you will see all libs are added to the list.



# 3 Application Programming Interface (API)

## 3.1 SDP Management Functions

### 3.1.1 Usage

#### 3.1.1.1 Windows/Linux

Please refer to the SDP Conventions for basic usage.

#### 3.1.1.2 Android

Please refer to the SDP Conventions for basic usage.

### 3.1.2 Constant

#### 3.1.2.1 Android

Class:
mrm.define**.MRM_CONSTANTS**

Fields:

| Field Name | Type | Value |
|---|---|---|
| SDP_EVENT_ID_UNKNOWN | int | -1 |
| SDP_EVENT_ID_HOTKEY_STATUS_CHANGED | int | 0 |

### 3.1.3 APIs

#### 3.1.3.1 sdp_init

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_init**(void) |
| **Android** | - |

**Description:**

General initialization of the Smart Display library.

**Parameters:**

none

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

**Remarks:**

Prior to calling any SDP API function the library needs to be initialized by calling this function. The return code for all SDP API function will be **MRM_ERR_LIBRARY_NOT_INIT** unless this function is called.

### 3.1.3.2 sdp_deinit

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_deinit**(void) |
| **Android** | - |

**Description:**

General uninitialization of the Smart Display library.

**Parameters:**

none

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

### 3.1.3.3 sdp_bind_service

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | - |
| **Android** | int **sdp_bind_service**(SDPServiceConnection conn) |

**Description:**

Connect current application context(ex: Activity) to SDP Service.

**Parameters:**

**conn:**

A instance of **mrm.client.SDPServiceClient.SDPServiceConnection**.

You must implement the following interfaces of **SDPServiceConnection**:

**public void on_service_connected()**

This is a callback which is called when the service is connected.

It will be trigged after you called sdp_bind_service().

**public void on_service_disonnected()**

This is a callback which is called when the service is disconnected.

It will be trigged when the SDP service process is stopped or died.

SDP Service might be stopped **by user manually** or **by system automatically (ex: when low memory)**.

Please refer to the SDP sample code for the detailed usage.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

**Remarks:**

- You can only access SDP API after **sdp_bind_service()** is called and the **on_service_connected()** callback is triggered. Before that, for most SDP API calls, you will get error code **MRM_ERR_ANDROID_CLIENT_SERVICE_DISCONNECTED.**
- If SDP Service is in stopped status, the service will be started when APP called **sdp_bind_service().**
- Due to the nature of Android, a background service(e.g. SDP Service) might be killed by system automatically, you should carefully implement **on_service_disonnected()** callback to deal with the service disconnect event.
- You can call **sdp_bind_service()** in **on_service_disonnected()** callback to re-connect if you need to.
- You should call **sdp_unbind_service()** before your application context(ex: Activity) is destroyed or it

may cause memory leak.

### 3.1.3.4 sdp_unbind_service

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | - |
| **Android** | int **sdp_unbind_service**() |

**Description:**

Disconnect current application context from SDP Service.

**Parameters:**

none

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

**Remarks:**

You should call this function before your application context(ex: Activity) is destroyed or it may cause memory leak.

### 3.1.3.5 sdp_is_service_connected

**Syntax:**

| | |
|---|---|
| **Android** | int **sdp _is_service_initialized**(boolean[] status) |

**Description:**

Get initialization status of SDP Service.

**Parameters:**

**status**

An allocated array of size 1 for storing returned initialization status.

The returned value will be store at index 0.

The value is TRUE, if SDP Service is initialized and all served APIs are available.

The value is FALSE, if SDP Service is not initialized.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

**Remark:**

-

### 3.1.3.6 sdp_is_service_initialized

**Syntax:**

| | |
|---|---|
| **Android** | boolean **sdp _is_service_connected**() |

**Description:**

Check whether the client application process is connected to SDP Service.

**Parameters:**

**-**

**Returns:**

Returns TRUE, if connected.

Returns FALSE, if disconnected.

**Remark:**

-

### 3.1.3.7 sdp_get_version

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_get_version**(char *version) |
| **Android** | int **sdp_get_version**(byte[] version) |

**Description:**

Get the library version of Smart Display.

**Parameters:**

**version** [out]

Pointer to a buffer that will hold the version of SDK. The buffer is C string that end of '\0'. The content of unused bytes filled 0x00.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

**Remark:**

The maximum length of version string is **SDP_MAXIMUM_LIBRARY_STRING_LENGTH**(24)

### 3.1.3.8 sdp_get_platform_name

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_get_platform_name**(char *name) |
| **Android** | int **sdp_get_platform_name**(byte[] name) |

**Description:**

Get the platform name.

**Parameters:**

**name** [out]

Pointer to a buffer that will hold the version of SDK. The buffer is C string that end of '\0'. The content of unused bytes filled 0x00.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

## 3.2 Firmware Management Functions

### 3.2.1 Usage

#### 3.2.1.1 Windows/Linux

Please refer to the SDP Conventions for Windows/Linux basic usage.

#### 3.2.1.2 Android

Please refer to the SDP Conventions for Android basic usage.

### 3.2.2 APIs

#### 3.2.2.1 sdp_firmware_get_version

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_firmware_get_version**(char *version) |
| **Android** | int **sdp_firmware_get_version**(byte[] version) |

**Description:**

Get the version of firmware.

**Parameters:**

**version** [out]

Pointer to a buffer that will hold the version of firmware. The buffer is C string that end of '\0'. The content of unused bytes filled 0x00.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

### 3.2.2.2 sdp_firmware_load_default

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_firmware_load_default**(void) |
| **Android** | int **sdp_firmware_load_default**() |

**Description:**

Load the default configuration to current configuration.

**Parameters:**

none

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

**Remarks:**

This function will take about 300ms to 600ms for loading the default configuration.

### 3.2.2.3 sdp_firmware_save_default

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_firmware_save_default**(void) |
| **Android** | int **sdp_firmware_save_default**() |

**Description:**

Save current configuration as the default configuration.

**Parameters:**

none

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

**Remarks:**

This function will take about 300ms to 600ms for saving the default configuration.

### 3.2.2.4 sdp_firmware_reset

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_firmware_reset**(void) |
| **Android** | int **sdp_firmware_reset**() |

**Description:**

Reset the firmware.

**Parameters:**

none

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

**Remarks:**

This function will take about 300ms for resetting the firmware.

## 3.3 Backlight Control Functions

### 3.3.1 Usage

#### 3.3.1.1 Windows/Linux

Please refer to the SDP Conventions for basic usage.

#### 3.3.1.2 Android

Please refer to the SDP Conventions for Android basic usage.

## 3.3.2 APIs

### 3.3.2.1 sdp_backlight_get_level_range

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_backlight_get_level_range** (unsigned char *max, unsigned char *min) |
| **Android** | int **sdp_backlight_get_level_range**(byte[] max, byte[] min) |

**Description:**

Get the current maximum and minimum of LCD backlight level.

**Parameters:**

**max** [out]

Pointer to a byte that will hold the maximum of level.

**min** [out]

Pointer to a byte that will hold the minimum of level.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

### 3.3.2.2 sdp_backlight_set_level_range

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_backlight_set_level_range** (unsigned char max, unsigned char min) |
| **Android** | int **sdp_backlight_set_level_range**(byte max, byte min) |

**Description:**

Set the current maximum and minimum of LCD backlight level.

**Parameters:**

**max** [in]

maximum of level.

**min** [in]

minimum of level.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

**Remark:**

The setting backlight level valid range please use the sdp_backlight_get_max_level_range function to checkout. In normally the range is 0 to 30. The maximum level must biggest than minimum level.

### 3.3.2.3 sdp_backlight_get_max_level_range

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_backlight_get_max_level_range** (unsigned char *max, unsigned char *min) |
| **Android** | int **sdp_backlight_get_max_level_range**(byte[] max, byte[] min) |

**Description:**

Get the maximum and minimum of LCD backlight level.

**Parameters:**

**max** [out]

Pointer to a byte that will hold the maximum of level.

**min** [out]

Pointer to a byte that will hold the minimum of level.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

### 3.3.2.4 sdp_backlight_get_level

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_backlight_get_level** (unsigned char *level) |
| **Android** | int **sdp_backlight_get_level**(byte[] level) |

**Description:**

Get the current selecting LCD backlight level.

**Parameters:**

**level** [out]

Pointer to a byte that will hold the current selecting level.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

### 3.3.2.5 sdp_backlight_set_level

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_backlight_set_level** (unsigned char level) |
| **Android** | int **sdp_backlight_set_level**(byte level) |

**Description:**

Set the current LCD backlight level.

**Parameters:**

**level** [in]

LCD Backlight level.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

### 3.3.2.6 sdp_backlight_get_brightness

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_backlight_get_brightness** (unsigned level, char *per) |
| **Android** | int **sdp_backlight_get_brightness**(byte level, byte[] per) |

**Description:**

Get the percentage of brightness on the specifies level.

**Parameters:**

**level** [in]

Backlight level.

**per** [out]

The percentage of brightness.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

### 3.3.2.7 sdp_backlight_set_brightness

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_backlight_set_brightness** (unsigned level, char per) |
| **Android** | int **sdp_backlight_set_brightness**(byte level, byte per) |

**Description:**

Set the percentage of brightness on the specifies level.

**Parameters:**

**level** [in]

Backlight level.

**per** [in]

The percentage of brightness.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

## 3.4 Hotkey Functions

### 3.4.1 Usage

The backlight design for scalable level setting. User can define a level range and customize each level brightness. For example, you can set level range 0 to 5; Level 0 brightness always equal 0%; Level 1 brightness set to 20% and so on. When you select level 4, you can see the LCD brightness about 80%. You can call sdp_backlight_set_level_range() and sdp_backlight_set_brightness() to change level brightness. Calling sdp_backlight_set_level() to change current level.

Remark:
The level 0 always equal 0%, you can't change the brightness on this level.

#### 3.4.1.1 Basic Usage

##### Windows/Linux

Please refer to the SDP Conventions for Windows/Linux basic usage.

##### Android

Please refer to the SDP Conventions for Android basic usage.

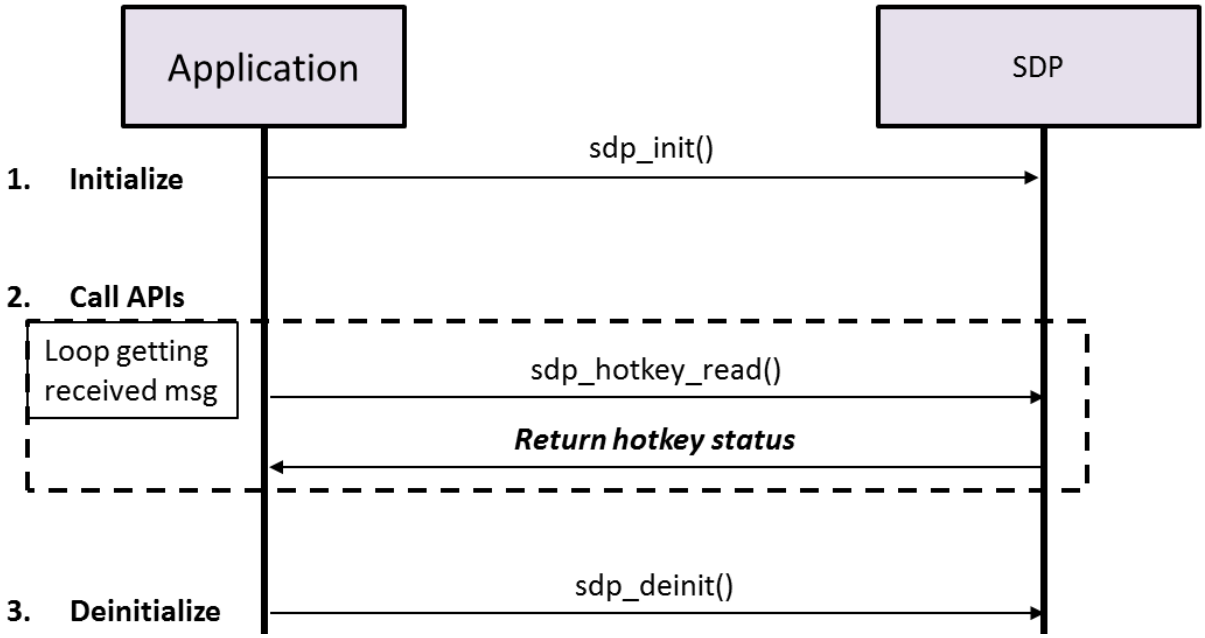
#### 3.4.1.2 Hotkey Status Getting

This section describes how to get hotkey press status.

### Windows/Linux

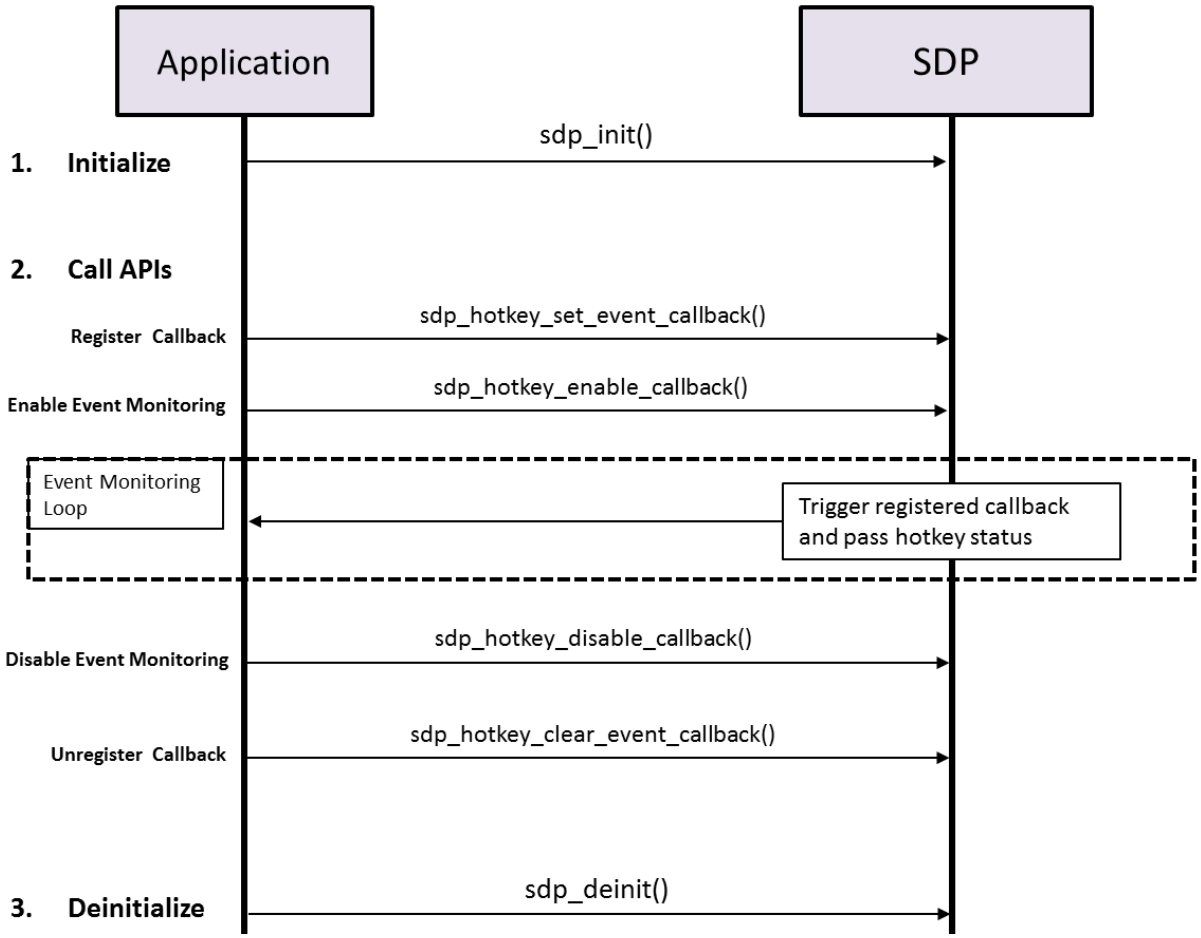
You implement your hotkey status reading application in either **Polling** or **Event Handling** style.

- **Polling:**




The most simple way to get hotkey status is to hold a loop in your APP and keep calling sdp_hotkey_read() .
This methods is relative simple but may result in unnecessary power consuming.

- **Event Handling:**




To get hotkey status in event handling style, you need to call sdp_hotkey_set_event_callback() first to register a callback to HOYKEY API module and call sdp_hotkey_enable_callback() to enable event monitoring in HOYKEY API module.

When hotkey status changes(i.e. pressed or released), HOYKEY API module triggers the registered callback and pass the hotkey status to client APP.

If you do not want to monitor hotkey status event any more, you should call sdp_hotkey_disable_callback() to disable event monitoring in HOYKEY API module, then call sdp_hotkey_clear_event_callback() to unregister callback.

The advantage of getting hotkey status in event handling style is that APP's process only works when hotkey status changed, which relatively costs less system resources and power.
The disadvantage is that there would be some overhead of event handling.

Please refer to the sample code for implementation details.

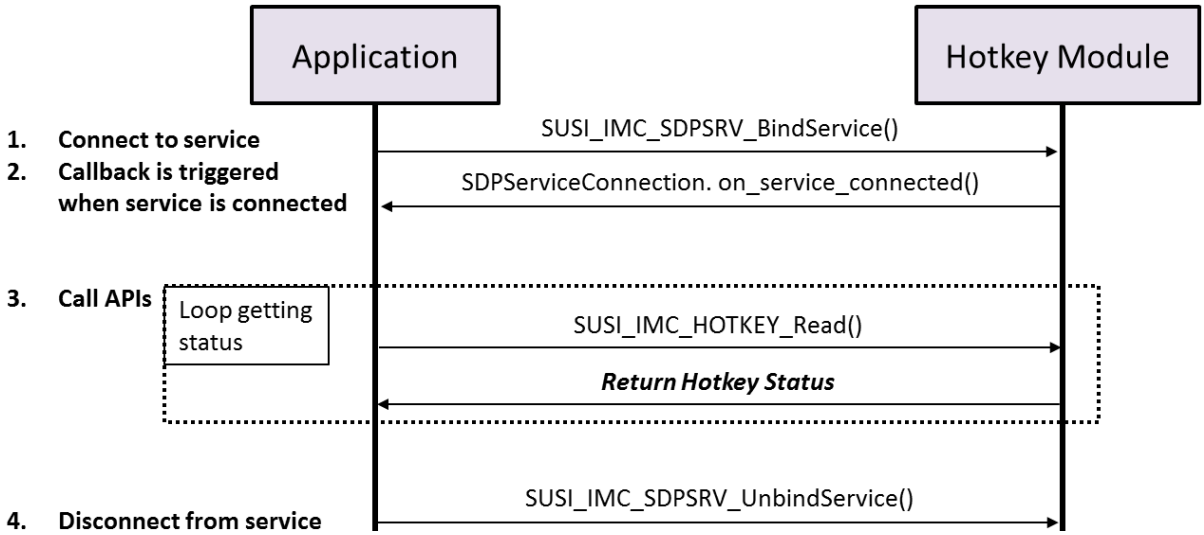
**Remark:**
sdp_hotkey_disable_callback() **MUST** be called before sdp_hotkey_clear_event_callback() otherwise it will

result in unexpected erroneous behavior.

## Android

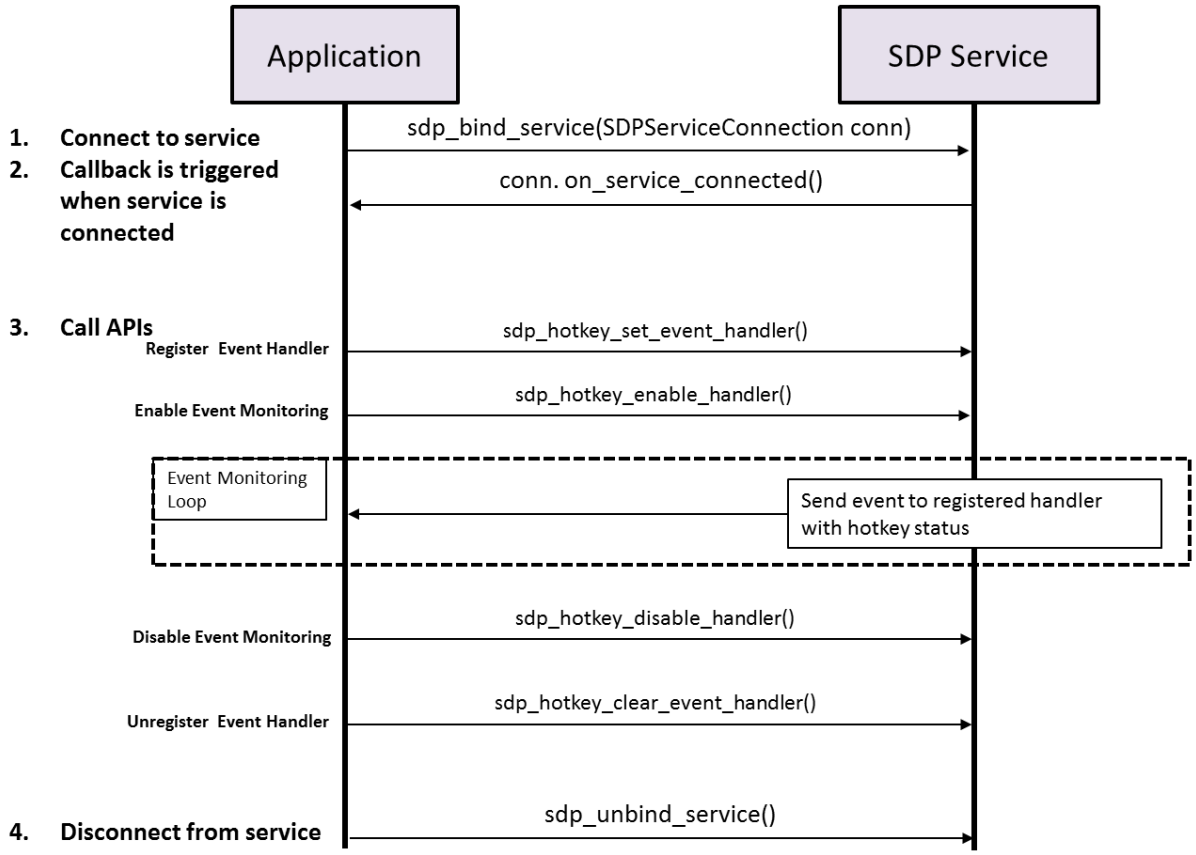
You implement your hotkey status reading application in either **Polling** or **Event Handling** style.

- **Polling:**




The most simple way to get hotkey status is to hold a loop in your APP and keep calling sdp_hotkey_read() .
This methods is relative simple but may result in unnecessary power consuming.

- **Event Handling:**




MRM SDK leverage the Android Handler mechanism to inform hotkey press event. To get hotkey status in event handling style, you need to create an instance of Handler and call sdp_hotkey_set_event_handler() first to register the Handler instance to HOTKEY API module and call sdp_hotkey_enable_handler() to enable event monitoring in HOYKEY API module.

When hotkey status changes(i.e. pressed or released), SDP service trigger event the and send event with the hotkey status  to the registered handler instance.

If you do not want to monitor hotkey status event any more, you should call sdp_hotkey_disable_handler() to disable event monitoring in HOYKEY API module, then call sdp_hotkey_clear_event_handler() to unregister handler.

The advantage of getting hotkey status in event handling style is that APP's process only works when hotkey status changed, which relatively costs less system resources and power.
The disadvantage is that there would be some overhead of event handling.

Please refer to the sample code for implementation details.

**Remark:**
sdp_hotkey_disable_handler() **MUST** be called before sdp_hotkey_clear_event_handler() otherwise it will result in unexpected erroneous behavior.

## 3.4.2 Structure/Classes

### 3.4.2.1 Windows/Linux

#### sdp_keys_value_t Structure

**Syntax:**

```
typedef struct
{
        unsigned char key[8];
} sdp_keys_value_t;
```

**Description:**

This data structure defines the each key status.

**Members:**

**key[...]**

The key status. 0 is release; 1 is pressed.

#### 3.4.2.2 Android

### SDP_KEYS_VALUE

**class:**

mrm.define.SDP.**SDP_KEYS_VALUE**

**Description:**

This data structure defines the each key status.

**Fields:**

| Field Name | Type | Input/Output | Comment |
|---|---|---|---|
| key | byte[] | out | The key status. 0 is release; 1 is pressed. |

### 3.4.3 APIs

#### 3.4.3.1 sdp_hotkey_read

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_hotkey_read**(sdp_keys_value_t *keys) |
| **Android** | int **sdp_hotkey_read**(SDP_KEYS_VALUE keys) |

**Description:**

Get all current hot key status.

**Parameters:**

**keys** [out]

Windows/Linux:

Pointer to sdp_keys_value_t struct which is used to store the gotten status.

Android:

A instance of SDP_KEYS_VALUE which is used to store the gotten status.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

### 3.4.3.2 sdp_hotkey_get_number

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_hotkey_get_number**(int *num) |
| **Android** | int **sdp_hotkey_get_number**(int[] num) |

**Description:**

Get number of keys supported by current device.

**Parameters:**

**num** [out]

The number of keys.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

### 3.4.3.3 sdp_hotkey_set_event_callback

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_hotkey_set_event_callback**(hotkey_event_callback callback) |

**Description:**

This function is used to register the hotkey callback function. There can be only one function registered for this callback.

The callback is called every time when hotkey status are changed. A "**sdp_keys_value_t**" struct which contains the hotkey status, will be passed to the callback function as an argument.

The callback function is used for asynchronous notification. We recommend that you should not do long task in the callback function or the next status changed notification will be blocked and also may block other SDP API's task.

**Parameters:**

**callback** [in]

Pointer to a user defined callback function. The type of callback function pointer - "hotkey_event_callback" is defined as following:

```
typedef void ( _stdcall *hotkey_event_callback ) ( sdp_keys_value_t *keys )
```

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

**Remarks:**

Before clear the callback function you should calling sdp_hotkey_disable_callback to disable hotkey callback function otherwise the system may crash since program access the invalid memory.

### 3.4.3.4 sdp_hotkey_clear_event_callback

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_hotkey_clear_event_callback**(void) |
| **Android** | |

**Description:**

Unregister the callback from SDP library.

**Parameters:**

none

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

### 3.4.3.5 sdp_hotkey_enable_callback

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_hotkey_enable_callback**(void) |
| **Android** | |

**Description:**

Enable the hotkey callback function registered in SDP library.

**Parameters:**

none

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

**Remarks:**

Before enable the callback function you should register a callback function to the SDP library.

### 3.4.3.6 sdp_hotkey_disable_callback

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_hotkey_disable_callback**(void) |
| **Android** | |

**Description:**

Disabl the hotkey callback function registered in SDP library.

**Parameters:**

none

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

### 3.4.3.7 sdp_hotkey_set_event_handler

**Syntax:**

| **Android** | int **sdp_hotkey_set_event_handler**(Handler handler) |
|---|---|

**Description:**

Register a instance of Handler for listening hotkey status changed event.

When hotkey status changed, the Handler.handleMessage() will be called and receives a Message object with "what" field of SDP_EVENT_ID_HOTKEY_PRESS_STATUS_CHANGED and with "obj" field which is assigned with a SDP_KEYS_VALUE object.
You can cast the Message's "obj" field to SDP_KEYS_VALUE and get the key press status from it.
Please refer to the SDP sample code for the usage details.

**Parameters:**

handler [in]

**Android**:

An instance of Handler.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

### 3.4.3.8 sdp_hotkey_clear_event_handler

**Syntax:**

| **Linux / Windows** | int sdp_hotkey_clear_event_handler() |
|---|---|

**Description:**

Unregister a Handler for listening hotkey status changed event.

**Parameters:**

handler [in]

**Android**:

An instance of Handler.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

**Remark:**

**sdp_hotkey_disable_handler()** **MUST** be called before **sdp_hotkey_clear_event_handler()**, otherwise it will result in unexpected erroneous behavior.
Please refer to the usage.

### 3.4.3.9 sdp_hotkey_enable_handler

**Syntax:**

| **Android** | int **sdp_hotkey_enable_handler**() |
|---|---|

**Description:**

Enable the hotkey status changed event notification in SDP Service.

**Parameters:**

none

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

**Remarks:**

Before enable the notification, you should register a handler for the event.

### 3.4.3.10 sdp_hotkey_disable_handler

**Syntax:**

| **Android** | int **sdp_hotkey_disable_handler**() |
|---|---|

**Description:**

Disable the hotkey status changed event notification in SDP Service.

**Parameters:**

none

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

### 3.4.3.11 sdp_hotkey_get_brightness

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_hotkey_get_brightness**(unsigned char *brightness ) |
| **Android** | int **sdp_hotkey_get_brightness**(byte[] brightness) |

**Description:**

Get the LED brightness of Hotkey.

**Parameters:**

**brightness** [out]

The hotkey LED brightness.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

### 3.4.3.12 sdp_hotkey_set_brightness

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_hotkey_set_brightness**(unsigned char brightness ) |
| **Android** | int **sdp_hotkey_set_brightness**(byte brightness) |

**Description:**

Set the LED brightness of Hotkey.

**Parameters:**

**brightness** [in]

The hotkey LED brightness. The valid values for this argument are 0 through 100

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

## 3.5 Speaker Control Functions

### 3.5.1 Usage

#### 3.5.1.1 Windows/Linux

Please refer to the SDP Conventions for Windows/Linux basic usage.

#### 3.5.1.2 Android

Please refer to the SDP Conventions for Android basic usage.

### 3.5.2 APIs

#### 3.5.2.1 sdp_speaker_enable

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_speaker_enable**(void) |
| **Android** | int **sdp_speaker_enable**() |

**Description:**

Enable the speaker volume.

**Parameters:**

none

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

#### 3.5.2.2 sdp_speaker_disable

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_speaker_disable**(void) |
| **Android** | int **sdp_speaker_disable**() |

**Description:**

Mute the speaker volume.

**Parameters:**

none

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

### 3.5.2.3 sdp_speaker_get_status

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_speaker_get_status**(char *status) |
| **Android** | int **sdp_speaker_get_status**(boolean[] status) |

**Description:**

Get status of speaker mute function.

**Parameters:**

**status** [out]

Speaker status. The value 1 is Enable otherwise 0 is Mute.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

## 3.6 Front-End USB Power Control Functions

### 3.6.1 Usage

#### 3.6.1.1 Windows/Linux

Please refer to the SDP Conventions for Windows/Linux basic usage.

#### 3.6.1.2 Android

Please refer to the SDP Conventions for Android basic usage.

## 3.6.2 APIs

### 3.6.2.1 sdp_usb_enable

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_usb_enable**(void) |
| **Android** | int **sdp_usb_enable**() |

**Description:**

Enable the power of front-end USB port.

**Parameters:**

none

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

### 3.6.2.2 sdp_usb_disable

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_usb_disable**(void) |
| **Android** | int **sdp_usb_disable**() |

**Description:**

Disable the power of front-end USB port.

**Parameters:**

none

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

### 3.6.2.3 sdp_usb_get_status

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_usb_get_status**(char *status) |
| **Android** | int **sdp_usb_get_status**(boolean[] status) |

**Description:**

Get status of front-end USB port power.

**Parameters:**

**status** [out]

Speaker status. The value 1 is On otherwise 0 is Off.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

## 3.7 Sensor Functions

### 3.7.1 Usage

#### 3.7.1.1 Windows/Linux

Please refer to the SDP Conventions for Windows/Linux basic usage.

#### 3.7.1.2 Android

Please refer to the SDP Conventions for Android basic usage.

### 3.7.2 APIs

#### 3.7.2.1 sdp_sensor_get_lux

**Syntax:**

| | |
|---|---|
| **Windows / Linux** | mrm_err **sdp_sensor_get_lux**(int *lux) |
| **Android** | int **sdp_sensor_get_lux**(int[] lux) |

**Description:**

Get illuminance from the front-end light sensor.

**Parameters:**

**lux** [out]

The illuminance. Unit is lux.

**Returns:**

**MRM_ERR_NO_ERROR** - On success.

Otherwise see the error code list.

# 4 Error Code List

## 4.1 Comman Error

- (0x00000000) **MRM_ERR_NO_ERROR** - On success.
- (0x00000001) **MRM_ERR_INVALID_POINTER** - Encounter invalid pointer.
- (0x00000002) **MRM_ERR_INVALID_ARGUMENT** - Encounter invalid argument. Please check out the API parameter.
- (0x00000003) **MRM_ERR_UNSUPPORT_OPERATION** - Encounter unsupported operation. Please check out spec. of  the platform supported.
- (0x00000005) **MRM_ERR_LIBRARY_NOT_INIT** - Function call before the library init.
- (0x00000010) **MRM_ERR_ILLEGAL_OPERATION** - Encounter illegal operation.
- (0x00000011) **MRM_ERR_LIBRARY_ALREADY_INIT** - Function call before the library init.
- (0x00000012) **MRM_ERR_ARRAY_OUR_OF_RANGE** - Encounter the access array out of range.
- (0x00000013) **MRM_ERR_OPERATION_FAIL** - Encounter operation fail.

- (0x10000001) **MRM_ERR_ANDROID_JNI_NULL_POINTER** - Null pointer error occurred on service side(JNI).
- (0x10000002) **MRM_ERR_ANDROID_JNI_OUT_OF_MEMORY** - Out of memory error occurred on service side(JNI).
- (0x10000003) **MRM_ERR_ANDROID_JNI_EVENT_INIT_FAILED** - Failed to init event handle on service side(JNI).
- (0x10000004) **MRM_ERR_ANDROID_JNI_EVENT_DEINIT_FAILED** - Failed to init event handle on service side(JNI).
- (0x10000005) **MRM_ERR_ANDROID_JNI_EVENT_LISTENING_THREAD_ALREADY_RUNNING** - Event listening thread in service is already running(JNI).
- (0x10000006) **MRM_ERR_ANDROID_JNI_EVENT_LISTENING_THREAD_CREATE_FAILED** - Failed to create event listening thread in service(JNI).
- (0x10100001) **MRM_ERR_ANDROID_SERVICE_NULL_POINTER** - Null pointer error occurred on service side.
- (0x10100002) **MRM_ERR_ANDROID_SERVICE_UNKNWON_EXCEPTION** - Unknown exception occurred on service side.
- (0x10100003) **MRM_ERR_ANDROID_SERVICE_REMOTE_CALLBACK_LIST_NOT_FOUND** - Remote callback list not found on service side.
- (0x10100004) **MRM_ERR_ANDROID_SERVICE_REMOTE_CALLBACK_REGISTER_FAILED** - Failed to register remote callback on service side.

- (0x10100005) **MRM_ERR_ANDROID_SERVICE_REMOTE_CALLBACK_UNREGISTER_FAILED** - Failed to unregister remote callback on service side.
- (0x10100006) **MRM_ERR_ANDROID_SERVICE_REMOTE_CALLBACK_UPDATE_FAILED** - Failed to update remote callback cookie on service side.
- (0x10200001) **MRM_ERR_ANDROID_CLIENT_NULL_POINTER** - Null pointer error occurred on client side in IVCP Service Client API lib.
- (0x10200002) **MRM_ERR_ANDROID_CLIENT_FAILED_TO_BIND_SERVICE** - Unable to connect client application context to the service.
- (0x10200003) **MRM_ERR_ANDROID_CLIENT_SERVICE_ALREADY_CONNECTED** - Current client application context is already connected to the service.
- (0x10200004) **MRM_ERR_ANDROID_CLIENT_SERVICE_DISCONNECTED** - Current client application context is not connected to the service.
- (0x10200005) **MRM_ERR_ANDROID_CLIENT_REMOTE_EXCEPTION** - Remote exception received at client side. Failed to execute required task.
- (0x10200006) **MRM_ERR_ANDROID_CLIENT_FAILED_TO_UNBIND_SERVICE** - Unable to disconnect client application context to the service.

## 4.2 SDP Error

- (0x20000001) **MRM_ERR_SDP_DEVICE_NODE_OPEN_FAIL** - Open SDP device node fail. Please checkout SDP is exist or the device not use by another application.
- (0x20000002) **MRM_ERR_SDP_DEVICE_NODE_WRITE_FAIL** - Encounter write operation fail.
- (0x20000003) **MRM_ERR_SDP_DEVICE_NODE_READ_FAIL** - Encounter read operation fail.
- (0x20000004) **MRM_ERR_SDP_IS_BUSY** - SDP device is busy. Try again.
- (0x20000005) **MRM_ERR_SDP_ERROR_COMMAND** - SDP encounters  unknown command.
- (0x20000006) **MRM_ERR_SDP_ERROR_COMMAND_PRARAMETER_NOT_MUCH** - SDP encounters  unknown parameter. Please check out the firmware version matching current library.
- (0x20000007) **MRM_ERR_SDP_PARAMETER_OUT_OF_RANGE** - SDP encounters illegal parameter. Please check out the parameter is correct.
- (0x20000008) **MRM_ERR_SDP_DEVICE_NODE_READ_TIMEOUT** - SDP encounters out of time when read operation.
- (0x20000008) **MRM_ERR_SDP_NO_EEPROM_CHIP_FIND** - SDP no found any storage chip on this platform. Please check out the hardware PCB version.