BIOSTATION SDK

Reference Manual

Rev. 1.1

SUPREMA

Revision History

Rev No.	Issued date	**Description**
1.1	2006 Oct. 20	Initial Release

Contents

1. Introduction

1.1. Contents of the SDK

Directory	**Sub Directory**	**Contents**
SDK	Document	- BIOSTATION SDK Reference Manual
	Include	- Header files
	Lib	- BS_SDK.dll: SDK DLL file - BS_SDK.lib: import library to be linked with C/C++ applications
	Example	- A short example showing the basic usage of the SDK

1.2. Usage

1.2.1. Compilation

To call APIs defined in the SDK, **BS_API.h** should be included in the source files and **Include** should be added to the include directories. To link user application with the SDK, **BS_SDK.lib** should be added to library modules.

The following snippet shows a typical source file.

```
#include “BS_API.h”
int main()
{
     // First, initialize the SDK
     BS_RET_CODE result = BS_InitSDK();

     // Open a communication channel
     int handle;
     result = BS_OpenSocket( “192.168.1.2”, 1470, &handle );

     // Get the ID of BIOSTATION terminal
     unsigned id;
```

```
        result = BS_GetBiostationID( handle, &id );

        // Set the ID of BIOSTATION terminal for further commands
        BS_SetBiostationID( handle, id );

       // Do something
       result = BS_ReadLog( handle, … );
   }
```

1.2.2. Using the DLL

To run applications compiled with the SDK, the BS_SDK.dll file should be in the system directory or in the same directory of the application.

1.2.3. Optional Requirements

To use USB channel, libusb-win32 should be installed first. You can download it from http://libusb-win32.sourceforge.net/. The library is also included in BioAdmin V3.x package.

2. API Specification

2.1. Return Codes

Most APIs in the SDK return BS_RET_CODE. The return codes and their meanings are as follows.

Code	**Description**
BS_SUCCESS	The function succeeds.
BS_ERR_NO_AVAILABLE_CHANNEL	Communication handle is no more available.
BS_ERR_INVALID_COMM_HANDLE	The communication handle is invalid.
BS_ERR_CANNOT_WRITE_CHANNEL	Cannot write data to the communication channel.
BS_ERR_WRITE_CHANNEL_TIMEOUT	Write timeout.
BS_ERR_CANNOT_READ_CHANNEL	Cannot read data from the communication channel.
BS_ERR_READ_CHANNEL_TIMEOUT	Read timeout.
BS_ERR_CHANNEL_OVERFLOW	The data is larger than the channel buffer.
BS_ERR_CANNOT_INIT_SOCKET	Cannot initialize the WinSock library.
BS_ERR_CANNOT_OPEN_SOCKET	Cannot open the socket.
BS_ERR_CANNOT_CONNECT_SOCKET	Cannot connect to the socket.
BS_ERR_CANNOT_OPEN_SERIAL	Cannot open the RS232 port.
BS_ERR_CANNOT_OPEN_USB	Cannot open the USB port.
BS_ERR_BUSY	BIOSTATION is processing another command.

BS_ERR_INVALID_PACKET	The packet has invalid header or trailer.
BS_ERR_CHECKSUM	The checksum of the packet is incorrect.
BS_ERR_UNSUPPORTED	The operation is not supported.
BS_ERR_FILE_IO	A file IO error is occurred during the operation.
BS_ERR_DISK_FULL	No more space is available.
BS_ERR_NOT_FOUND	The specified user is not found.
BS_ERR_INVALID_PARAM	The parameter is invalid.
BS_ERR_RTC	Real time clock cannot be set.
BS_ERR_MEM_FULL	Memory is full in the BIOSTATION.
BS_ERR_DB_FULL	The user DB is full.
BS_ERR_INVALID_ID	The user ID is invalid.
BS_ERR_USB_DISABLED	USB interface is disabled.
BS_ERR_COM_DISABLED	Communication channels are disabled.
BS_ERR_WRONG_PASSWORD	Wrong master password.
BS_ERR_INVALID_USB_MEMORY	The USB memory is not initialized.

2.2. Communication API

To communicate with a BIOSTATION terminal, users should configure the communication channel first. There are six types of communication channels – TCP socket, UDP socket, RS232, RS485, USB, and USB memory stick.

- BS_InitSDK: initializes the SDK.
- BS_OpenSocket: opens a TCP socket for LAN communication.
- BS_CloseSocket: closes a TCP socket.
- BS_OpenSocketUDP: opens a UDP socket for receiving IP addresses of BIOSTATION terminals.
- BS_CloseSocketUDP: closes a UDP socket.
- BS_OpenSerial: opens a RS232 port.
- BS_CloseSerial: closes a RS232 port.
- BS_OpenSerial485: opens a RS485 port.
- BS_CloseSerial485: closes a RS485 port.
- BS_OpenUSB: opens a USB port.
- BS_CloseUSB: closes a USB port.
- BS_OpenUSBMemory: opens a USB memory stick for communicating with virtual terminals.
- BS_CloseUSBMemory: closes a USB memory stick.

BS_InitSDK

Initializes the SDK. This function should be called once before any other functions are executed.

BS_RET_CODE BS_InitSDK()

Parameters

None

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_OpenSocket

Opens a TCP socket with specified IP address and port number. Since UDP socket is reserved for receiving IP addresses in V1.1 and later versions, TCP sockets should be used for general communication.

BS_RET_CODE BS_OpenSocket(const char* biostationAddr, int port, int* handle)

Parameters

biostationAddr

IP address of BIOSTATION.

port

TCP port number. The default is 1470.

handle

Pointer to the handle to be assigned.

Return Values

If a socket is opened successfully, return BS_SUCCESS with the assigned handle. Otherwise, return the corresponding error code.

BS_CloseSocket

Closes the socket.

BS_RET_CODE BS_CloseSocket(int handle)

Parameters

handle

Handle of the TCP socket.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_OpenSocketUDP

Opens a UDP socket for receiving IP addresses of BIOSTATION terminals. When Server IP is set on a BIOSTATION terminal, it will send UDP packets containing its IP address to the server periodically. UDP socket is only used for receiving these packets. For all other purposes, TCP socket should be used.

BS_RET_CODE BS_OpenSocketUDP(const char* biostationAddr, int port, int* handle)

Parameters

biostationAddr

IP address of BIOSTATION.

port

UDP port number. The default is 1470.

handle

Pointer to the handle to be assigned.

Return Values

If a socket is opened successfully, return BS_SUCCESS with the assigned handle. Otherwise, return the corresponding error code.

BS_CloseSocketUDP

Closes the UDP socket.

BS_RET_CODE BS_CloseSocketUDP(int handle)

Parameters

handle

Handle of the UDP socket.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_OpenSerial

Opens a RS232 port with specified baud rate.

BS_RET_CODE BS_OpenSerial(const char* port, int baudrate, int* handle)

Parameters

port

Pointer to a null-terminated string that specifies the name of the serial port.

baudrate

Specifies the baud rate at which the serial port operates. Available baud rates are 9600, 19200, 38400, 57600, and 115200bps. The default is 115200bps.

handle

Pointer to the handle to be assigned.

Return Values

If the function succeeds, return BS_SUCCESS with the assigned handle. Otherwise, return the corresponding error code.

BS_CloseSerial

Closes the serial port.

BS_RET_CODE BS_CloseSerial(int handle)

Parameters

handle

Handle of the serial port.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_OpenSerial485

Opens a RS485 port with specified baud rate.

BS_RET_CODE BS_OpenSerial485(const char* port, int baudrate, int* handle)

Parameters

port

Pointer to a null-terminated string that specifies the name of the serial port.

baudrate

Specifies the baud rate at which the serial port operates. Available baud rates are 9600, 19200, 38400, 57600, and 115200bps. The default is 115200bps.

handle

Pointer to the handle to be assigned.

Return Values

If the function succeeds, return BS_SUCCESS with the assigned handle. Otherwise, return the corresponding error code.

BS_CloseSerial485

Closes the serial port.

BS_RET_CODE BS_CloseSerial485(int handle)

Parameters

handle

Handle of the serial port.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_OpenUSB

Open a USB communication channel with BIOSTATION. To use USB channel, libusb-win32 should be installed first. You can download it from http://libusb-win32.sourceforge.net/. The library is also included in BioAdmin V3.x package.

BS_RET_CODE BS_OpenUSB(int* handle)

Parameters

handle

Pointer to the handle to be assigned.

Return Values

If the function succeeds, return BS_SUCCESS with the assigned handle. Otherwise, return the corresponding error code.

BS_CloseUSB

Closes the USB channel.

BS_RET_CODE BS_CloseUSB(int handle)

Parameters

handle

Handle of the USB channel.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_OpenUSBMemory

USB memory sticks can be used for transferring data between the host PC and BIOSTATION terminals. After creating a virtual terminal in a memory stick, you can communicate with it in the same way as other communication channels. For further details, please refer to the BIOSTATION User Guide.

BS_RET_CODE BS_OpenUSBMemory(const char* driveLetter, int* handle);

Parameters

driveLetter

Drive letter in which the USB memory stick is inserted.

handle

Pointer to the handle to be assigned.

Return Values

If the function succeeds, return BS_SUCCESS with the assigned handle.
If the memory is not initialized, return BS_ERR_INVALID_USB_MEMORY. Otherwise, return the corresponding error code.

BS_CloseUSBMemory

Closes the USB memory.

BS_RET_CODE BS_CloseUSBMemory(int handle)

Parameters

handle

Handle of the USB memory.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

2.3. Terminal API

The following APIs provide functionalities for configuring basic features of BIOSTATION terminals.

- BS_GetBiostationID: gets the ID of a terminal.
- BS_SetBiostationID: sets the ID for further commands.
- BS_GetClientIPAddress: receives the IP addresses of BIOSTATION terminals.
- BS_SearchBiostation: searches the ID of BIOSTATION terminals in a RS485 network.
- BS_GetTime: gets the time of a terminal.
- BS_SetTime: sets the time of a terminal.
- BS_CheckSystemStatus: checks the status of a terminal.
- BS_Reset: resets a terminal.
- BS_UpgradeEx: upgrades firmware of a terminal.
- BS_Disable: disables a terminal.
- BS_Enable: re-enables a terminal.
- BS_DisableCommunication: disables communication channels.
- BS_EnableCommunication: enables communication channels.

BS_GetBiostationID

To communicate with BIOSTATION, user should know the ID of the terminal attached to the communication channel. In most cases, this is the first function to be called after a communication channel is opened.

BS_RET_CODE BS_GetBiostationID(int handle, unsigned* biostationID)

Parameters

handle

Handle of the communication channel.

biostationID

Pointer to the ID to be returned.

Return Values

If the function succeeds, return BS_SUCCESS with the ID. Otherwise, return the corresponding error code.

BS_SetBiostationID

A BIOSTATION terminal will process commands only if the IDs of the packets match with its own. **BS_SetBioStationID** selects a BIOSTATION terminal to which further requests are sent.

BS_RET_CODE BS_SetBiostationID(int handle, unsigned id)

Parameters

handle

Handle of the communication channel.

id

ID of the BIOSTATION terminal.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_GetClientIPAddress

When Server IP is set on a BIOSTATION terminal, it will send UDP packets containing its IP address to the server periodically. **BS_GetClientIPAddress** is used for receiving these packets.

BS_RET_CODE BS_GetClientIPAddress(int handle, char* ipAddr, unsigned* id, int* port, int timeout)

Parameters

handle

Handle of the UDP socket.

ipAddr

IP address of the BIOSTATION terminal.

port

Port number of the BIOSTATION terminal.

timeout

Timeout for receiving packets.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

Example

```
char ipAddr[16];
unsigned id;
int port;
int handle;

//
// (1) Receive IP address of BIOSTATION terminal
//
BS_RET_CODE result = BS_OpenSocketUDP( "0.0.0.0", 1470, &handle );

if( result != BS_SUCCESS )
{
     printf( "Cannot open UDP: %d\n", result );
     exit( 1 );
}
```

```
result = BS_GetClientIPAddress( handle, ipAddr, &id, &port,  20000 );

if( result != BS_SUCCESS )
{
     printf( "Cannot receive IP address: %d\n", result );
     exit( 1 );
}

BS_CloseSocketUDP( handle )

//
// (2) Connect to the BIOSTATION terminal
//
result = BS_OpenSocket( ipAddr, port, &handle );
```

BS_SearchBiostation

Searches BIOSTATION terminals connected to a RS485 network.

BS_RET_CODE BS_SearchBiostation(int handle, unsigned* IDs, int* numOfBiostation)

Parameters

handle

Handle of the RS485 channel.

IDs

Pointer to the BIOSTATION IDs to be returned.

numOfBiostation

Pointer to the number of BIOSTATION IDs to be returned.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_GetTime

Gets the time of a BIOSTATION terminal. All the time values in BIOSTATION SDK represent local time, not Coordinated Universal Time(UTC). To convert a UTC value into a local time, **BS_ConvertToLocalTime** can be used.

BS_RET_CODE BS_GetTime(int handle, time_t* timeVal)

Parameters

handle

Handle of the communication channel.

timeVal

Pointer to the number of seconds elapsed since midnight (00:00:00), January 1, 1970, according to the system clock. Please note that it is local time, not UTC.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_SetTime

Sets the time of a BIOSTATION terminal.

BS_RET_CODE BS_SetTime(int handle, time_t timeVal)

Parameters

handle

Handle of the communication channel.

timeVal

Number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

Example

```
// Synchronize the time of a BIOSTATION terminal with that of PC
time_t currentTime = BS_ConvertToLocalTime( time( NULL ) );
BS_RET_CODE result = BS_SetTime( handle, currentTime );
```

BS_CheckSystemStatus

Checks if a BIOSTATION terminal is connected to the channel.

BS_RET_CODE BS_CheckSystemStatus(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_Reset

Resets a BIOSTATION terminal.

BS_RET_CODE BS_Reset(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_UpgradeEx

Upgrades the firmware of a BIOSTATION terminal. BIOSTATION terminal should not be turned off when upgrade is in progress.

BS_RET_CODE BS_UpgradeEx(int handle, const char* upgradeFile)

Parameters

handle

Handle of the communication channel.

upgradeFile

Filename of the firmware, which will be provided by Suprema.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_Disable

When communicating with a BIOSTATION terminal, data corruption may occur if users are manipulating it directly at the terminal simultaneously. For example, if a user is placing a finger while the terminal is deleting fingerprints, the result might be inconsistent. To prevent such cases, developers would be well advised to call **BS_Disable** before sending commands which will change the status of a terminal. After this function is called, the BIOSTATION will ignore keypad and fingerprint inputs, and process only the commands delivered through communication channels. For the terminal to revert to normal status, **BS_Enable** should be called afterwards.

BS_RET_CODE BS_Disable(int handle, int timeout)

Parameters

handle

Handle of the communication channel.

timeout

If there is no command during this timeout interval, the terminal will get back to normal status automatically. The maximum timeout value is 60 seconds.

Return Values

If the terminal is processing another command, BS_ERR_BUSY will be returned.

Example

```
// Enroll users
BS_RET_CODE result = BS_Disable( handle, 20 ); // timeout is 20 seconds

if( result == BS_SUCCESS )
{
     result = BS_EnrollUser( … );
     // …

     BS_Enable( handle );
}
```

BS_Enable

Enables the terminal. See **BS_Disable** for details.

BS_RET_CODE BS_Enable(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_DisableCommunication

Disables all communication channels. After this function is called, BIOSTATION will return BS_ERR_COM_DISABLED to all functions except for **BS_EnableCommunication** and **BS_GetBiostationID**.

BS_RET_CODE BS_DisableCommunication(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_EnableCommunication

Re-enables all the communication channels.

BS_RET_CODE BS_EnableCommunication(int handle, const char* masterPassword)

Parameters

handle

Handle of the communication channel.

masterPassword

16 byte master password. The default password is a string of 16 NULL characters. To change the master password, please refer to the BIOSTATION User Guide.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

2.4. Log Management API

A BIOSTATION terminal can store up to 500,000 log records. It also provides APIs for real-time monitoring.

- BS_ClearLogCache: clears the log cache.
- BS_ReadLogCache: reads the log records in the cache.
- BS_GetLogCount: gets the number of log records.
- BS_ReadLog: reads log records.
- BS_DeleteLog: deletes log records.
- BS_DeleteAllLog: deletes all the log records.

BSLogRecord is defined as follows.

```
typedef struct {
     unsigned char event;
     unsigned char reserved1;
     unsigned short tnaEvent;
     time_t eventTime;
     unsigned userID;
     unsigned reserved2;
} BSLogRecord;
```

1. *event*
   The type of log record. The event codes and their meanings are as follows.

Category	Event Code	Value	Description
System	SYS_STARTED	0x6A	BIOSTATION is turned on.
I/O	RELAY_ON	0x80	The door is opened.
	RELAY_OFF	0x81	The door is closed.
	TAMPER_SW_ON	0x64	The case is opened.
	TAMPER_SW_OFF	0x65	The case is closed.
	DETECT_INPUT0	0x54	Detect a signal at input port 0.
	DETECT_INPUT1	0x55	Detect a signal at input port 1.
1:1	VERIFY_SUCCESS	0x27	1:1 matching succeeds.

matching	VERIFY_FAIL	0x28	1:1 matching fails.
	VERIFY_NOT_GRANTED	0x6e	Not allowed to enter.
	VERIFY_DURESS	0x62	Duress finger is detected.
1:N matching	IDENTIFY_SUCCESS	0x37	1:N matching succeeds.
	IDENTIFY_FAIL	0x38	1:N matching fails.
	IDENTIFY_NOT_GRANTED	0x6d	Not allowed to enter.
	IDENTIFY_DURESS	0x63	Duress finger is detected.
User	ENROLL_SUCCESS	0x17	A user is enrolled.
	ENROLL_FAIL	0x18	Cannot enroll a user.
	DELETE_SUCCESS	0x47	A user is deleted.
	DELETE_FAIL	0x48	Cannot delete a user.
	DELETE_ALL_SUCCESS	0x49	All users are deleted.

2. *tnaEvent*

   The index of TNA event, which is between BS_TNA_F1 and BS_TNA_ESC. See **BS_WriteTnaEventConfig** for details. It will be 0xffff if it is not a TNA event.
3. *eventTime*

   The local time at which the event occurred. It is represented by the number of seconds elapsed since midnight (00:00:00), January 1, 1970.
4. *userID*

   The user ID related to the log event. If it is not a user-related event, it will be 0.

BS_ClearLogCache

A BIOSTATION terminal has a cache which keeps 64 latest log records. This is useful for real-time monitoring. **BS_ClearLogCache** clears this cache for initializing or restarting real-time monitoring.

BS_RET_CODE BS_ClearLogCache(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

Example

```
// Clears the cache first
BS_RET_CODE result = BS_ClearLogCache( handle );

BSLogRecord logRecords[64];
int numOfLog;

// Monitoring loop
while( 1 ) {
     result = BS_ReadLogCache( handle, &numOfLog, logRecords );

     // do something
}
```

BS_ReadLogCache

Reads the log records in the cache. After reading, the cache will be cleared.

BS_RET_CODE BS_ReadLogCache(int handle, int* numOfLog, BSLogRecord* logRecord)

Parameters

handle

Handle to the communication channel.

numOfLog

Pointer to the number of log records in the cache.

logRecord

Pointer to the log records to be returned. This pointer should be preallocated large enough to store the log records.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_ReadLog

Reads log records which were written in the specified time interval. Although a BIOSTATION terminal can store up to 500,000 log records, the maximum number of log records to be returned by this function is limited to 32,768. Therefore, users should call **BS_ReadLog** repetitively if the number of log records in the time interval is larger than 32,768.

BS_RET_CODE BS_ReadLog(int handle, time_t startTime, time_t endTime, int* numOfLog, BSLogRecord* logRecord)

Parameters

handle

Handle of the communication channel.

startTime

Start time of the interval. If it is set to 0, the log records will be read from the start.

endTime

End time of the interval. If it is set to 0, the log records will be read to the end.

numOfLog

Pointer to the number of log records to be returned.

logRecord

Pointer to the log records to be returned. This pointer should be preallocated large enough to store the log records.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

Example

```
int numOfLog;
BSLogRecord* logRecord = (BSLogRecord*)malloc( .. );

// Reads all the log records
BS_RET_CODE result = BS_ReadLog( handle, 0, 0, &numOfLog, logRecord );

// Reads the log records of latest 24 hours
```

```
time_t currentTime = BS_ConvertToLocalTime( time( NULL ) );

result = BS_ReadLog( handle, currentTime – 24 * 60 * 60, 0, &numOfLog,
logRecord );
```

BS_DeleteLog

Deletes oldest log records.

BS_RET_CODE BS_DeleteLog(int handle, int numOfLog, int* numOfDeletedLog)

Parameters

handle

Handle of the communication channel.

numOfLog

Number of log records to be deleted.

numOfDeletedLog

Pointer to the number of deleted log records.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_DeleteAllLog

Deletes all log records.

BS_RET_CODE BS_DeleteAllLog(int handle, int* numOfDeletedLog)

Parameters

handle

Handle of the communication channel.

numOfDeletedLog

Pointer to the number of deleted log records.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_GetLogCount

Retrieves the number of log records.

BS_RET_CODE BS_GetLogCount(int handle, int* numOfLog)

Parameters

handle

Handle of the communication channel.

numOfLog

Pointer to the number of log records stored in a BIOSTATION terminal.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

2.5. Display Setup API

Users can customize the background images and sound effects using the following functions. The size of an image or sound file should not exceed 512KB.

- BS_SetBackground: sets the background image.
- BS_SetSlideShow: sets the images of the slide show.
- BS_DeleteSlideShow: deletes all the images of the slide show.
- BS_SetSound: sets a wave file for sound effects.
- BS_SetLanguageFile: sets the language resource file.
- BS_SendNotice: sends the notice messages.

BS_SetBackground

BIOSTATION has three types of background – logo, slide show, and notice. Users can customize these images using **BS_SetBackgroun**d and **BS_SetSlideShow**.

BS_SetBackground(int handle, int bgIndex, const char* pngFile)

Parameters

handle

Handle of the communication channel.

bgIndex

Background index. It should be one of BS_UI_BG_LOGO and BS_UI_BG_NOTICE.

pngFile

Name of the image file. It should be a 320x240 PNG file.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_SetSlideShow

Sets an image of the slide show. The maximum number of images is 16.

BS_RET_CODE BS_SetSlideShow(int handle, int numOfPicture, int imageIndex, const char* pngFile)

Parameters

handle

Handle of the communication channel.

numOfPicture

Total number of the images in the slide show.

imageIndex

Index of the image in the slide show.

pngFile

Name of the image file. It should be a 320x240 PNG file.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_DeleteSlideShow

Deletes all the images of the slide show.

BS_RET_CODE BS_DeleteSlideShow(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_SetSound

There are 6 sound effects in BIOSTATION. Users can replace these sounds using **BS_SetSound**.

BS_RET_CODE BS_SetSound(int handle, int soundIndex, const char* wavFile)

Parameters

handle

Handle of the communication channel.

soundIndex

Index of the sound effect. Available sound effects are as follows;

Index	**When to play**
BS_SOUND_START	When system starts
BS_SOUND_CLICK	When a keypad is pressed
BS_SOUND_SUCCESS	When authentication or other operations succeed
BS_SOUND_QUESTION	When displaying a dialog for questions or warnings
BS_SOUND_ERROR	When operations fail
BS_SOUND_SCAN	When a fingerprint is detected on the sensor

wavFile

Filename of the sound file. It should be a signed 16bit, 22050Hz, mono WAV file.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_SetLanguageFile

BIOSTATION supports two languages - Korean and English. It also provides a custom language option to support other languages. For further details of custom language option, please contact sales@supremainc.com.

BS_RET_CODE BS_SetLanguageFile(int handle, int languageIndex, const char* languageFile)

Parameters

handle

Handle of the communication channel.

languageIndex

Available options are BS_LANG_ENGLISH, BS_LANG_KOREAN, and BS_LANG_CUSTOM.

languageFile

Name of the language resource file.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_SendNotice

Sends the notice message, which will be displayed on BIOSTATION when the background is set to BS_UI_BG_NOTICE.

BS_SendNotice(int handle, const char* msg)

Parameters

handle

Handle of the communication channel.

msg

Pointer to the notice message. The maximum length is 1024 bytes.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

2.6. User Management API

These APIs provide user management functions such as enroll and delete.

- BS_GetUserDBInfo: gets the basic information of user DB.
- BS_EnrollUser: enrolls a user.
- BS_DeleteUser: deletes a user.
- BS_DeleteAllUser: deletes all users.
- BS_GetUser: gets the fingerprint templates and header information of a user.
- BS_GetUserInfo: gets the header information of a user.
- BS_GetAllUserInfo: gets the header information of all users.
- BS_ScanTemplate: scans a fingerprint on a BIOSTATION terminal and retrieves the template of it.

BS_GetUserDBInfo

Retrieves the number of enrolled users and fingerprint templates.

BS_RET_CODE BS_GetUserDBInfo(int handle, int* numOfUser, int* numOfTemplate)

Parameters

handle

Handle of the communication channel.

numOfUser

Pointer to the number of enrolled users.

numOfTemplate

Pointer to the number of enrolled templates.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_EnrollUser

Enrolls a user with header information and fingerprint templates. Maximum 5 fingers can be enrolled per user.

BS_RET_CODE BS_EnrollUser(int handle, BSUserHdr* hdr, unsigned char* templateData)

Parameters

handle

Handle of the communication channel.

Hdr

BSUserHdr is defined as follows;

```
typedef struct{
    unsigned ID;
    unsigned short reserved1;
    unsigned short adminLevel;
    unsigned short securityLevel;
    unsigned short statusMask; // internally used by BIOSTATION
    unsigned accessGroupMask;
    char name[BS_MAX_NAME_LEN + 1];
    char department[BS_MAX_NAME_LEN + 1];
    char password[BS_MAX_PASSWORD_LEN + 1];
    unsigned short numOfFinger;
    unsigned short duressMask;
    unsigned short checksum[5];
} BSUserHdr;
```

The key fields and their available options are as follows;

Fields	**Descriptions**
adminLevel	BS_USER_ADMIN
	BS_USER_NORMAL
securityLevel	BS_USER_SECURITY_DEFAULT
	BS_USER_SECURITY_LOWER
	BS_USER_SECURITY_LOW
	BS_USER_SECURITY_NORMAL
	BS_USER_SECURITY_HIGH
	BS_USER_SECURITY_HIGHER
accessGroupMask	A user can be a member of up to 4 access

	groups. For example, if the user is a member of Group 1 and Group 4, accessGroupMask will be 0xffff0104. If no access group is assigned to this user, it will be 0xffffffff.
duressMask	Under duress, users can authenticate with a duress finger to notify the threat. When duress finger is detected, the terminal will write a log record and output specified signals. The duressMask denotes which one of the enrolled finger is a duress one. For example, if the 3rd finger is a duress finger, duressMask will be 0x04.
checksum	Checksums of each enrolled finger. Since two templates are enrolled per finger, the checksum of a finger is calculated by summing all the bytes of the two template data.

templateData

Fingerprint templates of the user. Two templates should be enrolled per each finger.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

Example

```
BSUserHdr userHeader;

userHeader.ID = 1; // 0 cannot be assigned as a user ID.
userHeader.adminLevel = BS_USER_ADMIN;
userHeader.securityLevel = BS_USER_SECURITY_DEFAULT;
userHeader.accessGroupMask = 0xffff0201; // a member of Group 1 and Group
2;

strcpy( userHeader.name, “John” );
strcpy( userHeader.departments, “R&D” );
strcpy( userHeader.password, NULL ); // no password is enrolled. Password
```

```
                                  // should be longer than 4 bytes.
userHeader.numOfFinger = 2;
unsigned char* templateBuf = (unsigned char*)malloc( userHeader.numOfFinger
* 2 * BS_TEMPLATE_SIZE );

// fill template data

userHeader.duressMask = 0; // no duress finger

for( int i = 0; i < userHeader.numOfFinger * 2; i++ )
{
     if( i % 2 == 0 )
     {
        userHeader.checksum[i/2] = 0;
     }

     unsigned char* templateData = templateBuf + i * BS_TEMPLATE_SIZE;

     for( int j = 0; j < BS_TEMPLATE_SIZE; j++ )
     {
        userHeader.checksum[i/2] += templateData[j];
     }
}

BS_RET_CODE result = BS_EnrollUser( handle, &userHeader, templateBuf );
```

BS_DeleteUser

Deletes a user.

BS_RET_CODE BS_DeleteUser(int handle, unsigned userID)

Parameters

handle

Handle of the communication channel.

userID

ID of the user to be deleted.

Return Values

If the function succeeds, return BS_SUCCESS. If no user is enrolled with the ID, return BS_ERR_NOT_FOUND. Otherwise, return the corresponding error code.

BS_DeleteAllUser

Deletes all enrolled users.

BS_RET_CODE BS_DeleteAllUser(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_GetUser

Retrieves the header and template data of a user.

BS_RET_CODE BS_GetUser(int handle, unsigned userID, BSUserHdr* hdr, unsigned char* templateData)

Parameters

handle

Handle of the communication channel.

userID

User ID.

hdr

Pointer to the user header to be returned.

templateData

Pointer to the template data to be returned. This pointer should be preallocated large enough to store the template data.

Return Values

If the function succeeds, return BS_SUCCESS. If no user is enrolled with the ID, return BS_ERR_NOT_FOUND. Otherwise, return the corresponding error code.

BS_GetUserInfo

Retrieves the header information of a user.

BS_GetUserInfo(int handle, unsigned userID, BSUserHdr* hdr)

Parameters

handle

Handle of the communication channel.

userID

User ID.

hdr

Pointer to the user header to be returned.

Return Values

If the function succeeds, return BS_SUCCESS. If no user is enrolled with the ID, return BS_ERR_NOT_FOUND. Otherwise, return the corresponding error code.

BS_GetAllUserInfo

Retrieves the header information of all enrolled users.

BS_RET_CODE BS_GetAllUserInfo(int handle, BSUserHdr* hdr, int *numOfUser)

Parameters

handle

Handle of the communication channel.

hdr

Pointer to the **BSUserHdr** array to be returned. It should be preallocated large enough.

numOfUser

Pointer to the number of enrolled users.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_ScanTemplate

Scans a fingerprint on a BIOSTATION terminal and retrieves the template of it. This function is useful when a BIOSTATION terminal is used as an enroll station.

BS_RET_CODE BS_ScanTemplate(int handle, unsigned char* templateData)

Parameters

handle

Handle of the communication channel.

templateData

Pointer to the 384 byte template data to be returned.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

2.7. Configuration API

These APIs provide functionalities for reading/writing system configurations.

- BS_WriteDisplayConfig
- BS_ReadDisplayConfig
- BS_WriteOPModeConfig
- BS_ReadOPModeConfig
- BS_WriteTnaEventConfig
- BS_ReadTnaEventConfig
- BS_WriteIPConfig
- BS_ReadIPConfig
- BS_WriteFingerprintConfig
- BS_ReadFingerprintConfig
- BS_WriteIOConfig
- BS_ReadIOConfig
- BS_WriteRelayConfig
- BS_ReadRelayConfig
- BS_WriteSerialConfig
- BS_ReadSerialConfig
- BS_WriteUSBConfig
- BS_ReadUSBConfig
- BS_WriteWLANConfig
- BS_ReadWLANConfig
- BS_WriteEncryptionConfig
- BS_ReadEncryptionConfig
- BS_WriteWiegandConfig
- BS_ReadWiegandConfig
- BS_GetAvailableSpace

BS_WriteDisplayConfig / BS_ReadDisplayConfig

Writes/reads the display configurations.

BS_RET_CODE BS_WriteDisplayConfig(int handle, BSDisplayConfig* config)
BS_RET_CODE BS_ReadDisplayConfig(int handle, BSDisplayConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSDisplayConfig is defined as follows;

```
typedef struct {
    int language;
    int background;
    int bottomInfo;
    int timeout; // menu timeout in seconds, 0 for infinite
    int volume; // 0(mute) ~ 100
} BSDisplayConfig;
```

The key fields and their available options are as follows;

Fields	**Options**
language	● BS_UI_LANG_KOREAN ● BS_UI_LANG_ENGLISH ● BS_UI_LANG_CUSTOM
background	● BS_UI_BG_LOGO – shows logo image. ● BS_UI_BG_NOTICE – shows notice message. ● BS_UI_BG_PICTURE – shows slide show.
bottomInfo	● BS_UI_INFO_NONE – shows nothing. ● BS_UI_INFO_TIME – shows current time.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

Example

```
BSDisplayConfig dispConfig;

BS_RET_CODE result = BS_ReadDisplayConfig( handle, &dispConfig );

// modify the configuration if necessary

result = BS_Disable( handle, 10 ); // communication-only mode

if( result == BS_SUCCESS )
{
     result = BS_WriteDisplayConfig( handle, &dispCOnfig );
}

BS_Enable( handle );
```

BS_WriteOPModeConfig / BS_ReadOPModeConfig

Writes/reads the operation mode configurations.

BS_RET_CODE BS_WriteOPModeConfig(int handle, BSOPModeConfig* config)
BS_RET_CODE BS_ReadOPModeConfig(int handle, BSOPModeConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSOPModeConfig is defined as follows;

```
typedef struct {
    int authMode;
    int identificationMode;
    int tnaMode;
} BSOPModeConfig ;
```

The key fields and their available options are as follows;

Fields	**Options**
authMode	Sets 1:1 matching mode. ● BS_AUTH_FINGER_ONLY – only the fingerprint authentication is allowed. ● BS_AUTH_FINGER_OR_PASSWORD – both the fingerprint and password authentication are allowed. ● BS_AUTH_PASS_ONLY – only the password authentication is allowed.
identificationMode	Specifies 1:N matching mode. ● BS_1TON_FREESCAN – identification process starts automatically after detecting a fingerprint on the sensor. ● BS_1TON_BUTTON – identification process starts manually by pressing OK

	button. ● BS_1TON_DISABLE – identification is disabled.
tnaMode	● BS_TNA_DISABLE – TNA is disabled. ● BS_TNA_FUNCTION_KEY – TNA function keys are enabled.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_WriteTnaEventConfig / BS_ReadTnaEventConfig

Writes/reads the TNA event configurations.

BS_RET_CODE BS_WriteTnaEventConfig(int handle, BSTnaEventConfig* config)
BS_RET_CODE BS_ReadTnaEventConfig(int handle, BSTnaEventConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSTnaEventConfig is defined as follows;

```
#define BS_TNA_F1   0
#define BS_TNA_F2   1
#define BS_TNA_F3   2
#define BS_TNA_F4   3
#define BS_TNA_1    4
#define BS_TNA_2    5
#define BS_TNA_3    6
#define BS_TNA_4    7
#define BS_TNA_5    8
#define BS_TNA_6    9
#define BS_TNA_7    10
#define BS_TNA_8    11
#define BS_TNA_9    12
#define BS_TNA_CALL 13
#define BS_TNA_0    14
#define BS_TNA_ESC  15
#define BS_MAX_TNA_FUNCTION_KEY 16

 typedef struct {
     unsigned char enabled[BS_MAX_TNA_FUNCTION_KEY];
     unsigned char useRelay[BS_MAX_TNA_FUNCTION_KEY];
     char eventStr[BS_MAX_TNA_FUNCTION_KEY][BS_MAX_TNA_EVENT_LEN];
 } BSTnaEventConfig;
```

The key fields and their available options are as follows;

Fields **Options**

enabled	Specifies if this function key is used.
useRelay	If true, turn on the relay after authentication succeeds.
eventStr	Event string which will be used for showing log records

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

Example

```
BSTnaEventConfig tnaConfig;

tnaConfig.enabled[BS_TNA_F1] = true;
tnaConfig.useRelay[BS_TNA_F1] = true;
strcpy( tnaConfig.eventStr[BS_TNA_F1], “In” );

tnaConfig.enabled[BS_TNA_F2] = true;
tnaConfig.useRelay[BS_TNA_F2] = false;
strcpy( tnaConfig.eventStr[BS_TNA_F2], “Out” );
```

BS_WriteIPConfig/BS_ReadIPConfig

Writes/reads the TCP/IP configurations.

BS_RET_CODE BS_WriteIPConfig(int handle, BSIPConfig* config)

BS_RET_CODE BS_ReadIPConfig(int handle, BSIPConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSIPConfig is defined as follows;

```
#define BS_IP_DISABLE 0
#define BS_IP_ETHERNET 1
#define BS_IP_WLAN 2 // for Wireless version only

typedef struct {
     int lanType; // BS_IP_DISABLE, BS_IP_ETHERNET, or BS_IP_WLAN
     bool useDHCP;
     unsigned port;
     char ipAddr[BS_MAX_NETWORK_ADDR_LEN];
     char gateway[BS_MAX_NETWORK_ADDR_LEN];
     char subnetMask[BS_MAX_NETWORK_ADDR_LEN];
     char serverIP[BS_MAX_NETWORK_ADDR_LEN]; // see BS_OpenSocketUDP
} BSIPConfig;
```

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_WriteFingerpringConfig / BS_ReadFingerprintConfig

Writes/reads the configurations which are related to fingerprint authentication.

BS_RET_CODE BS_WriteFingerprintConfig(int handle, BSFingerprintConfig* config)
BS_RET_CODE BS_ReadFingerprintConfig(int handle, BSFingerprintConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSFingerprintConfig is defined as follows;

```
typedef struct {
    int security;
    int userSecurity;
    int fastMode;
    int sensitivity; // 0(Least) ~ 7(Most)
    int timeout; // 1 ~ 20 sec
    int imageQuality;
    bool viewImage;
} BSFingerprintConfig;
```

The key fields and their available options are as follows;

Fields	Options
security	Sets the security level. ● BS_SECURITY_NORMAL – FAR(False Acceptance Ratio) is 1/10,000 ● BS_SECURITY_SECURE – FAR is 1/100,000 ● BS_SECURITY_MORE_SECURE - FAR is 1/1,000,000
userSecurity	● BS_USER_SECURITY_READER – security level for 1:1 matching is same as the abobe security setting. ● BS_USER_SECURITY_USER – security level for 1:1 matching is defined by

	BSUserHdr.securityLevel per each user.
fastMode	● BS_FAST_MODE_NORMAL ● BS_FAST_MODE_FAST ● BS_FAST_MODE_FASTER
sensitivity	Specifies the sensitivity level of the sensor.
timeout	Specifies the timeout for fingerprint input in seconds.
imageQuality	When a fingerprint is scanned, BIOSTATION will check if the quality of the image is adequate for further processing. The imageQuality specifies the strictness of this quality check. ● BS_IMAGE_QUALITY_WEAK ● BS_IMAGE_QUALITY_MODERATE ● BS_IMAGE_QUALITY_STRONG

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_WriteIOConfig/BS_ReadIOConfig

BIOSTATION has two input ports, two output ports, and a tamper switch. These functions write/read the configurations of these IO ports.

BS_RET_CODE BS_WriteIOConfig(int handle, BSIOConfig* config)
BS_RET_CODE BS_ReadIOConfig(int handle, BSIOConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSIOConfig is defined as follows;

```
typedef struct {
    int input[BS_NUM_OF_INPUT];
    int output[BS_NUM_OF_OUTPUT];
    int tamper;
    int outputDuration; // ms
} BSIOConfig;
```

The key fields and their available options are as follows;

Fields	Options
input	Assigns an action to the input port. ● BS_IO_INPUT_DISABLED – no action ● BS_IO_INPUT_EXIT – turn on the relay. ● BS_IO_INPUT_WIEGAND – use two inputs ports as Wiegand input.
output	Assigns an event to the output port. The output port will be activated when the specified event occurs. ● BS_IO_OUTPUT_DISABLED ● BS_IO_OUTPUT_DURESS – activate when a duress finger is detected. ● BS_IO_OUTPUT_TAMPER – activate when the tamper switch is on. ● BS_IO_OUTPUT_AUTH_SUCCESS – activate when authentication succeeds. ● BS_IO_OUTPUT_AUTH_FAIL – activate when

	authentication fails. ● BS_IO_OUTPUT_WIEGAND – outputs Wiegand string when authentication succeeds.
tamper	Specifies what to do when the tamper switch is on. ● BS_IO_TAMPER_NONE - do nothing. ● BS_IO_TAMPER_LOCK_SYSTEM - lock the BIOSTATION terminal. To unlock, master password should be entered.
otuputDuration	Specifies the duration of output signal in milliseconds.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_WriteRelayConfig/BS_ReadRelayConfig

BIOSTATION has a relay output which can be used for opening a door. These functions write/read the relay configurations.

BS_RET_CODE BS_WriteRelayConfig(int handle, BSRelayConfig* config)
BS_RET_CODE BS_ReadRelayConfig(int handle, BSRelayConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSRelayConfig is defined as follows;

```
typedef struct {
    int event;
    int openDuration;
    int lockSchedule;
    int unlockSchedule;
} BSRelayConfig;
```

The key fields and their available options are as follows;

Fields	**Options**
event	Specifies when the relay is activated. ● BS_RELAY_EVENT_ALL - relay is on whenever authentication succeeds. ● BS_RELAY_EVENT_TNA – relay is not activated when the useRelay field of the TNA event is false. ● BS_RELAY_EVENT_NONE – relay is disabled.
openDuration	Specifies the duration in which the relay is on in seconds. After this duration, the relay will be turned off.
lockSchedule	Specifies the schedule in which the relay should be held on.
unlockSchedule	Specifies the schedule in which the relay should be held off.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_WriteSerialConfig / BS_ReadSerialConfig

Specifies the baud rate of the RS232 and RS485 ports.

BS_RET_CODE BS_WriteSerialConfig(int handle, BSSerialConfig* config)

BS_RET_CODE BS_ReadSerialConfig(int handle, BSSerialConfig* config)

Parameters

handle

Pointer to the communication channel.

config

BSSerialConfig is defined as follows;

```
typedef struct {
    int rs485; // BS_CHANNEL_DISABLED, 9600, 19200, 38400, 57600, 115200
    int rs232;
} BSSerialConfig
```

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_WriteUSBConfig / BS_ReadUSBConfig

Enables or disables the USB device interface.

BS_RET_CODE BS_WriteUSBConfig(int handle, BSUSBConfig* config)

BS_RET_CODE BS_ReadUSBConfig(int handle, BSUSBConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSUSBConfig is defined as follows;

```
typedef struct {
    bool connectToPC;
} BSUSBConfig;
```

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_WriteWLANConfig / BS_ReadWLANConfig

Writes/reads Wireless LAN configuration.

BS_RET_CODE BS_WriteWLANConfig(int handle, BSWLANConfig* config)
BS_RET_CODE BS_ReadWLANConfig(int handle, BSWLANConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSWLANConfig is defined as follows;

```
typedef struct {
    char name[BS_MAX_NETWORK_ADDR_LEN];
    int operationMode;
    short authType;
    short encryptionType;
    int keyType;
    char essid[BS_MAX_NETWORK_ADDR_LEN];
    char key1[BS_MAX_NETWORK_ADDR_LEN];
    char key2[BS_MAX_NETWORK_ADDR_LEN]; // not used for now
    char wpaPassphrase[64];
} BSWLANPreset;

typedef struct {
    int selected;
    BSWLANPreset preset[BS_MAX_WLAN_PRESET];
} BSWLANConfig;
```

The key fields and their available options are as follows;

Fields	**Options**
operationMode	Only infrastructure network – managed mode – is supported. ● BS_WLAN_MANAGED
authType	There are 3 types of authentication. ● BS_WLAN_AUTH_OPEN: no authentication.

- BS_WLAN_AUTH_SHARED: shared-key WEP authentication.
- BS_WLAN_AUTH_WPA_PSK: WPA authentication using a pre-shared master key.

encryptionType Available encryption options are determined by authentication type.

- BS_WLAN_NO_ENCRYPTION: no data encryption. This option should not be used as far as possible. For securing wireless channels, you should use WEP or WPA encryption.
- BS_WLAN_WEP: 64 and 128 bit encryption are supported.
- BS_WLAN_TKIP_AES: WPA TKIP and WPA2 AES encryption are supported. BIOSTATION will detect the appropriate encryption algorithm automatically.

Authentication	Supported encryption
AUTH_OPEN	NO_ENCRYPTION WEP
AUTH_SHARED	WEP
WPA_PSK	TKIP_AES

keyType You can specify WEP keys either in plain ascii text or in binary hex format.

- BS_WLAN_KEY_ASCII
- BS_WLAN_KEY_HEX

essid Network ID of the access point to which the BIOSTATION will be connected.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

Example

```
BSWLANConfig wlanConfig;

// (1) AP1
//      essid: biostation_wep
//      encryption: wep128 bit
//      WEP key: _suprema_wep_
strcpy( wlanConfig.preset[0].name, “Preset WEP” );
strcpy( wlanConfig.preset[0].essid, “biostation_wep” );
wlanConfig.preset[0].operationMode = BS_WLAN_MANAGED;
wlanConfig.preset[0].authType = BS_WLAN_AUTH_OPEN;
wlanConfig.preset[0].encryptionType = BS_WLAN_WEP;
wlanConfig.preset[0].keyType = BS_WLAN_KEY_ASCII;
strcpy( wlanConfig.preset[0].key1, “_suprema_wep_” );

// (2) AP2
//      essid: biostation_wpa
//      encryption: AES
//      WPS_PSK passphrase: _suprema_wpa_
strcpy( wlanConfig.preset[1].name, “Preset WPA” );
strcpy( wlanConfig.preset[1].essid, “biostation_wpa” );
wlanConfig.preset[1].operationMode = BS_WLAN_MANAGED;
wlanConfig.preset[1].authType = BS_WLAN_AUTH_WPA_PSK;
wlanConfig.preset[1].encryptionType = BS_WLAN_TKIP_AES;
strcpy( wlanConfig.preset[1].wpaPassphrase, “_suprema_wpa_” );
```

BS_WriteEncryptionConfig / BS_ReadEncryptionConfig

For higher security, users can turn on the encryption mode. When the mode is on, all the fingerprint templates are transferred and saved in encrypted form. To change the encryption mode, all the enrolled users should be deleted first. And a 256 bit encryption key should be sent, too.

BS_RET_CODE BS_WriteEncryptionConfig(int handle, BSEncryptionConfig* config)
BS_RET_CODE BS_ReadEncryptionConfig(int handle, BSEncryptionConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSEncryptionConfig is defined as follows;

```
typedef struct {
    bool useEncryption;
    unsigned char password[BS_ENCRYPTION_PASSWORD_LEN];
                            // 256bit encryption key
    int reserved[3];
} BSEncryptionConfig;
```

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_WriteWiegandConfig / BS_ReadWiegandConfig

Configures Wiegand format. Up to 64 bit Wegand formats are supported. The only constraint is that each field is limited to 32 bits.

BS_RET_CODE BS_WriteWiegandConfig(int handle, BSWiegandConfig* config)
BS_RET_CODE BS_ReadWiegandConfig(int handle, BSWiegandConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSWiegandConfig is defined as follows;

```
typedef enum {
    BS_WIEGAND_26BIT      = 0x01,
    BS_WIEGAND_PASS_THRU  = 0x02,
    BS_WIEGAND_CUSTOM     = 0x03,
} BS_WIEGAND_FORMAT;

typedef enum {
    BS_WIEGAND_EVEN_PARITY = 0,
    BS_WIEGAND_ODD_PARITY  = 1,
} BS_WIEGAND_PARITY_TYPE;

typedef struct {
    int bitIndex;
    int bitLength;
} BSWiegandField;

typedef struct {
    int bitIndex;
    BS_WIEGAND_PARITY_TYPE type;
    BYTE bitMask[8];
} BSWiegandParity;
```

```
typedef struct {
    BS_WIEGAND_FORMAT format;
    int totalBits;
} BSWiegandFormatHeader;

typedef struct {
    int numOfIDField;
    BSWiegandField field[MAX_WIEGAND_FIELD];
} BSWiegandPassThruData;

typedef struct {
    int numOfField;
    UINT32 idFieldMask;
    BSWiegandField field[MAX_WIEGAND_FIELD];
    int numOfParity;
    BSWiegandParity parity[MAX_WIEGAND_PARITY];
} BSWiegandCustomData;

typedef union {
    BSWiegandPassThruData passThruData;
    BSWiegandCustomData customData;
} BSWiegandFormatData;
```

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_GetAvailableSpace

Checks how much space is available in flash memory.

BS_RET_CODE BS_GetAvailableSpace(int handle, int* availableSpace, int* totalSpace)

Parameters

handle

Handle of the communication channel.

availableSpace

Pointer to the available space in bytes.

totalSpace

Pointer to the total space in bytes.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

2.8. Access Control API

These APIs provide access control features such as time schedule and access group. By using these functions, user's access can be controlled in finer detail.

- BS_AddTimeSchedule: adds a time schedule.
- BS_GetAllTimeSchedule: reads all time schedules.
- BS_DeleteTimeSchedule: deletes a time schedule.
- BS_DeleteAllTimeSchedule: deletes all time schedules.
- BS_AddHoliday: adds a holiday schedule.
- BS_GetAllHoliday: reads all holiday schedules.
- BS_DeleteHoliday: deletes a holiday schedule.
- BS_DeleteAllHoliday: deletes all holiday schedules.
- BS_AddAccessGroup: adds an access group.
- BS_GetAllAccessGroup: reads all access groups.
- BS_DeleteAccessGroup: deletes an access group.
- BS_DeleteAllAccessGroup: deletes all access groups.

BS_AddTimeSchedule

A BIOSTATION terminal can store up to 64 time schedules. Each time schedule consists of 7 daily schedules and an optional holiday schedule. And each daily schedule may have up to 5 time segments.

```
#define BS_TIMECODE_PER_DAY       5

typedef struct {
     unsigned short startTime; // start time in minutes
     unsigned short endTime; // end time in minutes
} BSTimeCodeElem;

typedef struct {
     BSTimeCodeElem codeElement[BS_TIMECODE_PER_DAY];
} BSTimeCode;

typedef struct {
     int scheduleID;
     BSTimeCode timeCode[7]; // 0 - Sunday, 1 - Monday, ...
     int holidayID;
     char name[BS_MAX_ACCESS_NAME_LEN];
} BSTimeSchedule;
```

BS_RET_CODE BS_AddTimeSchedule(int handle, BSTimeSchedule* schedule)

Parameters

handle

Handle of the communication channel.

schedule

Pointer to the time schedule to be added.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

Example

```
BSTimeSchedule timeSchedule;
```

```
memset( &timeSchedule, 0, sizeof(BSTimeSchedule) ); // clear the structure

timeSchedule.scheduleID = 1;
timeSchedule.holidayID = 1;


// Monday- 09:00 ~ 18:00
timeSchedule.timeCode[1].codeElement[0].startTime = 9 * 60;
timeSchedule.timeCode[1].codeElement[0].endTime = 18 * 60;

// Tuesday- 08:00 ~ 12:00 and 14:30 ~ 20:00
timeSchedule.timeCode[2].codeElement[0].startTime = 8 * 60;
timeSchedule.timeCode[2].codeElement[0].endTime = 12 * 60;
timeSchedule.timeCode[2].codeElement[1].startTime = 14 * 60 + 30;
timeSchedule.timeCode[2].codeElement[1].endTime = 20 * 60;

strcpy( timeSchedule.name, “Schedule 1” );

// …

BS_RET_CODE result = BS_AddTimeSchedule( handle, &timeSchedule );
```

BS_GetAllTimeSchedule

Reads all the registered time schedules.

BS_RET_CODE BS_GetAllTimeSchedule(int handle, int* numOfSchedule, BSTimeSchedule* schedule)

Parameters

handle

Handle of the communication channel.

numOfSchedule

Pointer to the number of enrolled schedules.

schedule

Pointer to the time schedule array to be read.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_DeleteTimeSchedule

Deletes the specified time schedule.

BS_RET_CODE BS_DeleteTimeSchedule(int handle,   int ID)

Parameters

handle

Handle of the communication channel.

ID

ID of the time schedule.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_DeleteAllTimeSchedule

Deletes all the time schedules stored in a BIOSTATION terminal.

BS_RET_CODE BS_DeleteAllTimeSchedule(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_AddHoliday

Each time schedule may have an optional holiday schedule. A holiday schedule consists of a holiday list and a daily schedule for it.

```
typedef struct {
     int holidayID; // -1 if not used
     int numOfHoliday;
     unsigned short holiday[32]; // (month << 8) | day
     BSTimeCode timeCode;
     char name[BS_MAX_ACCESS_NAME_LEN];
} BSHoliday;
```

BS_RET_CODE BS_AddHoliday(int handle, BSHoliday* holiday)

Parameters

handle

Handle of the communication channel.

holiday

Pointer to the holiday schedule to be added.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

Example

```
BSHoliday holiday;

memset( &holiday, 0, sizeof(BSHoliday) ); // clear the structure

holiday.holidayID = 1;
holiday.numOfHoliday = 10;

// Jan. 1 is holiday
holiday.holiday[0] = (1 << 8) | 1;

// Mar. 5 is holiday
holiday.holiday[1] = (3 << 8) | 5;

// …
// Access is granted during 09:00 ~ 10:00 on holideys
holiday.timeCode.codeElement[0].startTime = 9 * 60;
holiday.timeCode.codeElement[0].endTime = 10 * 60;

strcpy( holiday.name, “Holiday 1” );

BS_RET_CODE result = BS_AddHoliday( handle, &holiday );
```

BS_GetAllHoliday

Reads all the registered holiday schedules.

BS_RET_CODE BS_GetAllHoliday(int handle, int* numOfHoliday, BSHoliday* holiday)

Parameters

handle

Handle of the communication channel.

numOfHoliday

Pointer to the number of enrolled holiday schedules.

holiday

Pointer to the holiday schedules to be read.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_DeleteHoliday

Deletes the specified holiday schedule.

BS_RET_CODE BS_DeleteHoliday(int handle, int ID)

Parameters

handle

Handle of the communication channel.

ID

ID of the holiday schedule.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_DeleteAllHoliday

Deletes all the holiday schedules stored in a BIOSTATION terminal.

BS_RET_CODE BS_DeleteAllHoliday(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_AddAccessGroup

Each access group may have up to 16 time schedules. The access of members is granted only when the time belongs to the time schedules of the group.

```
#define BS_SCHEDULE_PER_GROUP      16

typedef struct {
     int groupID;
     int numOfSchedule;
     int scheduleID[BS_SCHEDULE_PER_GROUP];
     char name[BS_MAX_ACCESS_NAME_LEN];
} BSAccessGroup;
```

BS_RET_CODE BS_AddAccessGroup(int handle, BSAccessGroup* group)

Parameters

handle

Handle of the communication channel.

group

Pointer to the access group to be added.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_GetAllAccessGroup

Reads all the registered access groups.

BS_RET_CODE BS_GetAllAccessGroup(int handle, int* numOfAccessGroup, BSAccessGroup* group)

Parameters

handle

Handle of the communication channel.

numOfAccessGroup

Pointer to the number of enrolled access groups.

group

Pointer to the access groups to be read.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_DeleteAccessGroup

Deletes the specified access group.

BS_RET_CODE BS_DeleteAccessGroup(int handle, int ID)

Parameters

handle

Handle of the communication channel.

ID

ID of the access group.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

BS_DeleteAllAccessGroup

Deletes all the access groups stored in a BIOSTATION terminal.

BS_RET_CODE BS_DeleteAllAccessGroup(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the corresponding error code.

2.9. Miscellaneous API

These APIs do not interact with BIOSTATION directly. They provide miscellaneous functionalities which are helpful for using this SDK.

- BS_ConvertToUTF8: converts a wide-character string into a UTF8 string.
- BS_ConvertToLocalTime: converts a UTC value into a local time
- BS_SetKey: sets 256 bit key for decrypting/encrypting fingerprint templates.
- BS_EncryptTemplate: encrypts a fingerprint template.
- BS_DecryptTemplate: decrypts a fingerprint template.

BS_ConvertToUTF8

BIOSTATION supports UTF8 strings. To display non-western characters in BIOSTATION, it should be converted to UTF8 first.

int BS_ConvertToUTF8(const char* msg, char* utf8Msg, int limitLen)

Parameters

msg

String to be converted.

utf8Msg

Pointer to the buffer for new string.

limitLen

Maximum size of utf8Msg buffer.

Return Values

If the function succeeds, return the number of bytes written to the utf8Msg buffer. Otherwise, return 0.

BS_ConvertToLocalTime

All time values for the SDK should be local time. BS_ConvertToLocalTime converts a UTC time into local time.

time_t BS_ConvertToLocalTime(time_t utcTime)

Parameters

utcTime

Number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Return Values

The time value converted for the local time zone.

BS_SetKey

When the encryption mode is on, all the fingerprint templates are transferred and saved in encrypted form. If you want to decrypt/encrypt templates manually, you should use **BS_SetKey**, **BS_DecryptTemplate**, and **BS_EncryptTemplate**.

void BS_SetKey(unsigned char *key)

Parameters

key

32 byte – 256bit – encryption key.

Return Values

None

BS_EncryptTemplate

Encrypts a fingerprint template with the key set by **BS_SetKey**.

int BS_EncryptTemplate(unsigned char *input, unsigned char *output, int length)

Parameters

input

Pointer to the fingerprint template to be encrypted.

output

Pointer to the buffer for encrypted template.

length

Length of the template data.

Return Values

Return the length of encrypted template.

BS_DecryptTemplate

Decrypts an encrypted template with the key set by **BS_SetKey**.

void BS_DecryptTemplate(unsigned char *input, unsigned char *output, int length)

Parameters

input

Pointer to the encrypted template.

output

Pointer to the buffer for decrypted template.

length

Length of the encrypted template.

Return Values

None.

Contact Info

- **Headquarters**

  Suprema, Inc. (http://www.supremainc.com)

  16F Parkview Office Tower,

  Joengja-dong, Bundang-gu,

  Seongnam, Gyeonggi, 463-863 Korea

  Tel: +82-31-783-4505

  Fax:+82-31-783-4506

  Email: sales@supremainc.com, support@supremainc.com