BioStar SDK

Reference Manual

Rev. 1.1

Revision History

Rev No.	Issued date	Description
1.0	2008 Nov. 4	Initial Release.
1.1	2008 Dec. 3	Incorporated BioLite Net.

Contents

1. Introduction

1.1. Contents of the SDK

Directory	**Sub Directory**	**Contents**
SDK	Document	- BioStar SDK Reference Manual - Revision Notes
	Include	Header files
	Lib	- BS_SDK.dll: SDK DLL file - BS_SDK.lib: import library to be linked with C/C++ applications - libusb0.dll: libusb library necessary for accessing BioStation through USB.
	Example	Simple examples showing the basic usage of the SDK. They are written in C++, C#, and Visual Basic[1].

Table 1 Directory Structure 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()
{
```

[1] The Visual Basic example does not work with BioLite Net.

```
        // 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 and the type of the device
        unsigned deviceId;
        int deviceType;

        result = BS_GetDeviceID( handle, &deviceId, &deviceType );

        // Set the ID and the type of the device for further commands
        BS_SetDeviceID( handle, deviceId, deviceType );

       // 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. Auxiliary DLL

BS_SDK.dll is dependent on libusb for accessing BioStation through USB. It is included in BioAdmin and BioStar packages. It is also included in the Lib directory of the SDK.

1.3. BioStar SDK vs. BioStation SDK

BioStar, Suprema’s new access control software will replace BioAdmin. BioStation SDK, on which BioAdmin is based, will also be superseded by BioStar SDK. From the viewpoint of developers, the differences between the two SDKs are incremental. You can think of BioStar SDK as an upgraded version of BioStation SDK. Most APIs of BioStation SDK will work in BioStar SDK without modification. However, the descriptions of the deprecated APIs of BioStation SDK are removed from this manual. For the general differences between BioAdmin and BioStar, refer to the *BioStar Migration Guide*.

To make use of new features of BioStar SDK, the firmware of BioStation, BioEntry

Plus, and BioLite Net should meet the following requirements.

	BioStation	BioEntry Plus	BioLite Net
Firmware Version	V1.5 or later	V1.2 or later	V1.0 or later

Table 2 Firmware Compatibility

1.4. BioEntry Plus vs. BioLite Net

BioLite Net has been incorporated into BioStar SDK since version 1.1. BioLite Net shares most of the APIs with BioEntry Plus. When there is a difference in the usage of an API between the two devices, it is explained explicitly in the corresponding section.

2. QuickStart Guide

This chapter is for developers who want to get started quickly with BioStar SDK. It shows how to do the most common tasks for writing BioStar applications. Only snippets of C++ source codes will be listed below. You can find out more detailed examples written in C++, C#, and Visual Basic in the **Example** directory of the SDK.

2.1. Initialization

First of all, you have to initialize the SDK. The **BS_InitSDK** should be called once before calling any other functions.

2.2. Connect to Devices

The second task is to open a communication channel to the device. The available network options vary according to the device type. BioStation, BioEntry Plus, and BioLite Net support Ethernet and RS485, while USB, USB memory, RS232, and WLAN(optional) are available for BioStation only.

2.2.1. Ethernet

The LAN connection between BioStar applications and devices has two modes – direct and server. As for the differences between the two modes, refer to the *BioStar Administrator Guide* and the *Ethernet Troubleshooting Guide*. To connect to a device using BioStar SDK, you have to use the direct mode.

You also have to know the IP address and the TCP port of the device. If you do not know this information, you have to search the devices, first. The **BS_SearchDeviceInLAN** function is provided for this purpose. You can find multiple devices in a subnet using this function.

```
// (1) Open a UDP port
int udpHandle;
BS_OpenInternalUDP( &udpHandle );

// (2) Search devices in a subnet
int numOfDevice;
```

```
    unsigned deviceID[MAX_DEVICE];
    int deviceType[MAX_DEVICE];
    unsigned ipAddress[MAX_DEVICE];
    BS_RET_CODE result = BS_SearchDeviceInLAN( udpHandle, &numOfDevice,
deviceID, deviceType, ipAddress );

    // (3) Connect to devices
    for( int i = 0; i < numOfDevice; i++ )
    {
        int tcpHandle;
        int port = (deviceType[i] == BS_DEVICE_BIOSTATION)? 1470 : 1471;

        char ipAddrBuf[32];

        sprintf(ipAddrBuf, "%d.%d.%d.%d", ipAddress[i] & 0xff, (ipAddress[i]
        & 0xff00) >> 8, (ipAddress[i] & 0xff0000) >> 16, (ipAddress[i] &
        0xff000000) >> 24 );

        result = BS_OpenSocket( ipAddrBuf, port, &tcpHandle );

        result = BS_SetDeviceID( tcpHandle, deviceID[i], deviceType[i] );

        // do something
        // …

        BS_CloseSocket( tcpHandle );
    }
```

Of course, if you already know this information, you can call **BS_OpenSocket** directly. After acquiring a handle for a communication interface, you have to call **BS_SetDeviceID** before sending any other commands.

2.2.2. RS485

To communicate with a device connected to the host PC through RS485, the RS485 mode should be set as follows;

- For BioEntry Plus and BioLite Net, the **serialMod**e of **BEConfigData** and **BEConfigDataBLN** should be SERIAL_PC. See **BS_WriteConfig** for details.
- For BioStation, the **deviceType** of **BS485NetworkConfig** should be TYPE_CONN_PC. See **BS_Write485NetworkConfig** for details.

You can find devices in a RS485 network using **BS_SearchDevice**.

```
    // (1) Open a serial port
    int handle;
    BS_OpenSerial485( “COM1”, 115200, &handle );

    // (2) Search devices
    int numOfDevice;
    unsigned deviceID[MAX_DEVICE];
    int deviceType[MAX_DEVICE];
    BS_RET_CODE result = BS_SearchDevice( handle, deviceID, deviceType,
&numOfDevice );

    // (3) Communicate with devices
    for( int i = 0; i < numOfDevice; i++ )
    {
        // you need not open another channel

        result = BS_SetDeviceID( handle, deviceID[i], deviceType[i] );

        // do something
        // …
    }
```

The RS485 port of a device can also be used for transferring data between devices. See **BS_OpenSerial485** and **BS_Search485Slaves** for details.

2.2.3. Miscellaneous

In addition to Ethernet and RS485, BioStation also provides USB, USB memory, RS232, and WLAN(only for wireless models). The connection procedure to the WLAN devices is same as that of Ethernet, as long as the wireless parameters are configured correctly using **BS_WriteWLANConfig**.

As for USB, USB memory, and RS232, the connection procedure is much simpler. You only have to open the corresponding network interface using **BS_OpenUSB**, **BS_OpenUSBMemory**, and **BS_OpenSerial** respectively.

2.3. Configure Devices

You can configure the settings of each device using **BS_WriteXXXConfig** functions. To prevent unwanted corruptions of device configuration, you are strongly advised to read **3.7 Configuration API** carefully. It is also a good practice to call **BS_ReadXXXConfig** first before **BS_WriteXXXConfig**. By modifying only the necessary fields, you can minimize the risk of corrupting the configuration.

```
    // If you are to change the security level of a BioStation device
    // (1) Read the configuration first
    BSFingerprintConfig config;
    result = BS_ReadFingerprintConfig( handle, &config );

    // (2) Change the corresponding fields
    config.security = BS_SECURITY_SECURE;

    // (3) Write the configuration
    result = BS_WriteFingerprintConfig( handle, &config );
```

2.4. Enroll Users

To enroll users to devices, you have to fill the header information correctly in addition to the fingerprint templates. The following table shows the APIs for managing users for BioStation, BioEntry Plus and BioLite Net.

	BioStation	BioEntry Plus/BioLite Net
User header	BSUserHdrEx	BEUserHdr
Enroll a user	BS_EnrollUserEx	BS_EnrollUserBEPlus
Enroll multiple users	BS_EnrollMultipleUserEx	BS_EnrollMultipleUserBEPlus
Get user header information	BS_GetUserInfoEx BS_GetAllUserInfoEx	BS_GetUserInfoBEPlus BS_GetAllUserInfoBEPlus
Get user information including template	BS_GetUserEx	BS_GetUserBEPlus
Delete a user	BS_DeleteUser	
Delete multiple users	BS_DeleteAllUser BS_DeleteMultipleUsers	
Get user DB information	BS_GetUserDBInfo	

Table 3 User Management APIs

2.4.1. User Header

BioStation, BioEntry Plus, and BioLite Net have different header structures reflecting the capacity of each device. For example, BioStation has user name and password fields, while BioEntry Plus has only user ID field. For detailed description of each field, refer to **BS_EnrollUserEx** and **BS_EnrollUserBEPlus**.

2.4.2. Scan templates

You can use SFR300 USB reader for capturing fingerprint templates. You can also use BioStation, BioEntry Plus, or BioLite Net as an enroll station. For the latter case, **BS_ScanTemplate** function is provided.

```
    // If you are to enroll a user with one finger – two fingerprint
    // templates - to a BioEntry Plus device
    BEUserHdr userHdr;

    // fill other fields of userHdr
    // ..

    userHdr.numOfFinger = 1;
    unsigned char* templateBuf = (unsigned char*)malloc( 384 *
userHdr.numOfFinger * 2 );

    int bufPos = 0;

    for( int i = 0; i < userHdr.numOfFinger * 2; i++ )
    {
        BS_RET_CODE result = BS_ScanTemplate( handle, templateBuf + bufPos );
        bufPos += 384;
    }
```

2.4.3. Scan RF cards

One of major advantages of BioStar system is that you can combine diverse authentication modes. To assign a RF card to a user, you have to read it first using **BS_ReadCardIDEx**. Then, you can assign 4 byte card ID and 1 byte custom ID to the user header structrure. As for Mifare models, it will return the 4 byte CSN(Card Serial Number) of the Mifare card.

2.5. Get Log Records

BioStation, BioEntry Plus, and BioLite Net can store up to 500,000, 50,000, and 50,000 log records respectively. The log records are managed as a circular queue; when the log space is full, the oldest log records will be erased automatically. As for the event types, refer to **Table 5 Log Event Types**.

2.5.1. Read Log Records

There are two APIs for reading past log records; **BS_ReadLog** and **BS_ReadNextLog**. In most cases, **BS_ReadLog** would suffice. However, the maximum number of log records to be returned by this function is limited to 32,768, 8,192 and 8,192 for BioStation, BioEntry Plus, and BioLite Net respectively. If it is the case, you can use **BS_ReadNextLog**, which reads log records from the point where the last reading ends. See the **Example** section of **BS_ReadNextLog** for details.

2.5.2. Real-time Log Monitoring

Depending on your applications, you might have to read log records in real-time. For this purpose, BioStation, BioEntry Plus, and BioLite Net manage a log cache, which can store up to 128 log records.

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

BSLogRecord logRecords[128];
int numOfLog;

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

     // do something with the log records
     // …
}
```

2.6. Demo Project

The SDK includes simple examples written in C++, C#, and Visual Basic. You can compile and test them by yourselves. Inspecting the source codes would be the fastest way to be acquainted with the SDK.

The demo applications written in C++ and C# have the same user interface. You can test them as follows;

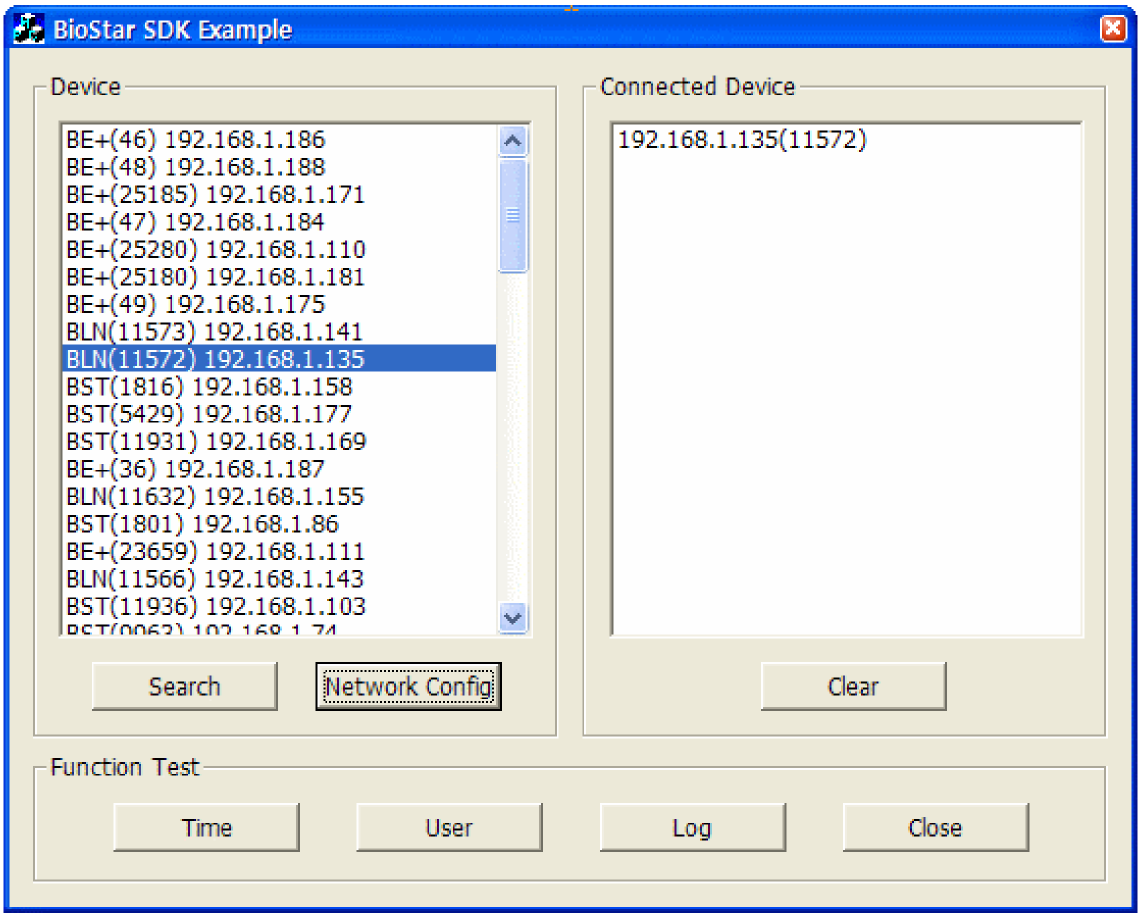
(1) Press **Search** button to discover devices using **BS_SearchDeviceInLAN**.

(2) Select a device in the **Device** list and press **Network Config** button.

(3) If necessary, change the network configuration of the device. Then, press **Connect** button to connect to the device. If connection succeeds, the device will be added to the **Connected Device** List.

(4) Select a device in the **Connected Device** list.

(5) Select one of the three buttons, **Time**, **User** and **Log** for further test.



```
BE+(46) 192.168.1.186
BE+(48) 192.168.1.188
BE+(25185) 192.168.1.171
BE+(47) 192.168.1.184
BE+(25280) 192.168.1.110
BE+(25180) 192.168.1.181
BE+(49) 192.168.1.175
BLN(11573) 192.168.1.141
BLN(11572) 192.168.1.135
BST(1816) 192.168.1.158
BST(5429) 192.168.1.177
BST(11931) 192.168.1.169
BE+(36) 192.168.1.187
BLN(11632) 192.168.1.155
BST(1801) 192.168.1.86
BE+(23659) 192.168.1.111
BLN(11566) 192.168.1.143
BST(11936) 192.168.1.103
BST(9963) 192.168.1.74
```

```
192.168.1.135(11572)
```

Figure 1 Demo Project

3. API Specification

3.1. 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. The maximum number of handle is 512.
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 specified IP address and the port.
BS_ERR_CANNOT_OPEN_SERIAL	Cannot open the RS232 port. Check if the serial port is already used by other applications.

BS_ERR_CANNOT_OPEN_USB	Cannot open the USB port. Check if the USB device driver is properly installed.
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. You cannot assign 0 as a user ID.
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.
BS_ERR_TRY_AGAIN	Scanning cards or fingerprints fails.

BS_ERR_EXIST_FINGER	The fingerprint template is already enrolled.

Table 4 Error Codes

3.2. Communication API

To communicate with a device, 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. BioEntry Plus and BioLite Net provide only three of them – TCP socket, UDP socket, and RS485.

- BS_InitSDK: initializes the SDK.
- BS_OpenSocket: opens a TCP socket for LAN communication.
- BS_CloseSocket: closes a TCP socket.
- BS_OpenInternalUDP: opens a UDP socket for administrative functions.
- BS_CloseInternalUDP: 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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_OpenSocket

Opens a TCP socket with the specified IP address and port number. With BioStation and BioLite Net, you can find out this information in the LCD menu of the device. With BioEntry Plus, you have to search the device first by **BS_SearchDeviceInLAN**.

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

Parameters

ipAddr

IP address of the device.

port

TCP port number. The default is 1470, 1471, and 1471 for BioStation, BioEntry Plus, and BioLite Net respectively.

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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_CloseSocket

Closes the socket.

BS_RET_CODE BS_CloseSocket(int handle)

Parameters

handle

Handle of the TCP socket acquired by **BS_OpenSocket**.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_OpenInternalUDP

BioStation(V1.5 or later), BioEntry Plus, and BioLite Net reserve a UDP port for internal communication. You can use this port for searching devices in a subnet. Or you can reset a device for troubleshooting purposes. See
BS_SearchDeviceInLAN and **BS_ResetUDP**.

BS_RET_CODE BS_OpenInternalUDP(int* handle)

Parameters

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.

Compatibility

BioStation(V1.5 or later)/BioEntry Plus/BioLite Net

BS_CloseInternalUDP

Closes the UDP socket.

BS_RET_CODE BS_CloseInternalUDP(int handle)

Parameters

handle

Handle of the UDP socket acquired by **BS_OpenInternalUDP**.

Return Values

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

Compatibility

BioStation(V1.5 or later)/BioEntry Plus/BioLite Net

BS_OpenSerial

Opens a RS232 port with the 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.

Compatibility

BioStation

BS_CloseSerial

Closes the serial port.

BS_RET_CODE BS_CloseSerial(int handle)

Parameters

handle

Handle of the serial port acquired by **BS_OpenSerial**.

Return Values

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

Compatibility

BioStation

BS_OpenSerial485

Opens a RS485 port with the specified baud rate. To communicate with a device connected to the host PC through RS485, the RS485 mode should be set as follows;

- For BioEntry Plus, the **serialMod**e of **BEConfigData** should be SERIAL_PC. See **BS_WriteConfig** for details.
- For BioLite Net, the **serialMod**e of **BEConfigDataBLN** should be SERIAL_PC. See **BS_WriteConfig** for details.
- For BioStation, the **deviceType** of **BS485NetworkConfig** should be TYPE_CONN_PC. See **BS_Write485NetworkConfig** for details.

In a half-duplex RS485 network, only one device should initiate all communication activity. We call this device 'host', and the other devices 'slaves'. Each BioStation, BioEntry Plus, or BioLite Net has one RS485 port, which can be used for connection to PC or other devices. The **RS485 Mode** setting of the device should be configured to one of the following modes;

- ***PC Connection***: The RS485 port is used for connecting to the PC. Maximum 31 devices can be connected to the PC through a RS485 network. In this case, the PC acts as the host device. Note that there is no zone support in this configuration.
- ***Host***: The device initiates all communication activity in a RS485 network. The host device can control up to 7 slave devices including maximum 4 Secure I/Os. For example, a BioStation host may have 7 BioEntry Plus slaves, or 3 BioStation slaves and 4 Secure I/Os. The host device also mediates packet transfers between the host PC and the slave devices. In other words, the host PC can transfer data to and from the slave devices even when only the host device is connected to the PC thorough LAN. As for searching slave devices attached to a host, refer to **BS_Search485Slaves**.
- ***Slave***: The slave device is connected to the host through RS485. It can communicate with the PC through the host device.

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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_CloseSerial485

Closes the serial port.

BS_RET_CODE BS_CloseSerial485(int handle)

Parameters

handle

Handle of the serial port acquired by **BS_OpenSerial485**.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_OpenUSB

Opens a USB communication channel with BioStation. To use the USB channel, libusb-win32 library and the device driver should be installed first. These are included in BioStar and BioAdmin packages.

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.

Compatibility

BioStation

BS_CloseUSB

Closes the USB channel.

BS_RET_CODE BS_CloseUSB(int handle)

Parameters

handle

Handle of the USB channel acquired by **BS_OpenUSB**.

Return Values

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

Compatibility

BioStation

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. If the corresponding function is not supported for the virtual terminal, BS_ERR_UNSUPPORTED will be returned.

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.

Compatibility

BioStation

BS_CloseUSBMemory

Closes the USB memory.

BS_RET_CODE BS_CloseUSBMemory(int handle)

Parameters

handle

Handle of the USB memory acquired by **BS_OpenUSBMemory**.

Return Values

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

Compatibility

BioStation

3.3. Device API

The following APIs provide functionalities for configuring basic features of BioStation, BioEntry Plus, and BioLite Net devices.

- BS_GetDeviceID: gets the ID and type of a device.
- BS_SetDeviceID: sets the ID and type of a device for further commands.
- BS_SearchDevice: searches devices in a RS485 network.
- BS_Search485Slaves: searches slave devices connected to a host device.
- BS_SearchDeviceInLAN: searches devices in a subnet.
- BS_GetTime: gets the time of a device.
- BS_SetTime: sets the time of a device.
- BS_CheckSystemStatus: checks the status of a device.
- BS_Reset: resets a device.
- BS_ResetUDP: resets a device using UDP protocol.
- BS_ResetLAN: reestablishes the IP configuration of BioStation.
- BS_UpgradeEx: upgrades firmware of a device.
- BS_Disable: disables a device.
- BS_Enable: re-enables a device.
- BS_DisableCommunication: disables communication channels.
- BS_EnableCommunication: enables communication channels.
- BS_ChangePasswordBEPlus: changes the master password of a BioEntry Plus or BioLite Net.
- BS_FactoryDefault: resets system parameters to the default values.

BS_GetDeviceID

To communicate with a device, you have to know its ID and device type. In most cases, this is the first function to be called after a communication channel is opened. After acquiring the ID and type, you have to call **BS_SetDeviceID**.

BS_RET_CODE BS_GetDeviceID(int handle, unsigned* deviceID, int* deviceType)

Parameters

handle

Handle of the communication channel.

deviceID

Pointer to the ID to be returned.

deviceType

Pointer to the type to be returned. It is either BS_DEVICE_BIOSTATION or BS_DEVICE_BIOENTRY_PLUS.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_SetDeviceID

After acquiring the ID and type of a device using **BS_GetDeviceID**, **BS_SearchDevice**, or **BS_SearchDeviceInLAN**, you have to call **BS_SetDeviceID**. It will initialize the device-related settings of the communication handle.

BS_RET_CODE BS_SetDeviceID(int handle, unsigned deviceID, int deviceType)

Parameters

handle

Handle of the communication channel.

deviceID

ID of the device.

deviceType

Type of the device. It is either BS_DEVICE_BIOSTATION or BS_DEVICE_BIOENTRY_PLUS.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_SearchDevice

Searches devices in a RS485 network. Up to 31 devices can be connected to the PC through RS485.

BS_RET_CODE BS_SearchDevice(int handle, unsigned* deviceIDs, int* deviceTypes, int* numOfDevice)

Parameters

handle

Handle of the RS485 channel.

deviceIDs

Pointer to the device IDs to be returned.

deviceTypes

Pointer to the device types to be returned.

numOfDevice

Pointer to the number of devices to be returned.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_Search485Slaves

Searches slave devices connected to a host device by RS485. As for the general description of RS485 configuration, see **BS_OpenSerial485**. To search slave devices, the following conditions should be met.

(1) The host and slave devices should be connected by RS485.

(2) The host device should be connected to LAN.

(3) The RS485 mode of the host and slave devices should be set to **Host** and **Slave** respectively. Refer to **BS_WriteConfig** and **BS_Write485NetworkConfig** for details.

BS_RET_CODE BS_Search485Slaves(int handle, BS485SlaveInfo* slaveList, int* numOfSlaves)

Parameters

handle

Handle of the host device acquired by **BS_OpenSocket**.

slaveList

Pointer to the array of slave information. **BS485SlaveInfo** is defined as follows;

```
typedef struct{
    unsigned slaveID;
    int slaveType;
} BS485SlaveInfo;
```

The key fields and their available options are as follows;

Fields	**Descriptions**
slaveID[2]	ID of the device
slaveType	BS_DEVICE_BIOSTATION
	BS_DEVICE_BIOENTRY_PLUS

numOfSlaves

Pointer to the number of slave devices to be returned.

[2] ID 0~3 are reserved for Secure I/Os. If the ID is 0, 1, 2, or 3, it represents a Secure I/O regardless of the slaveType.

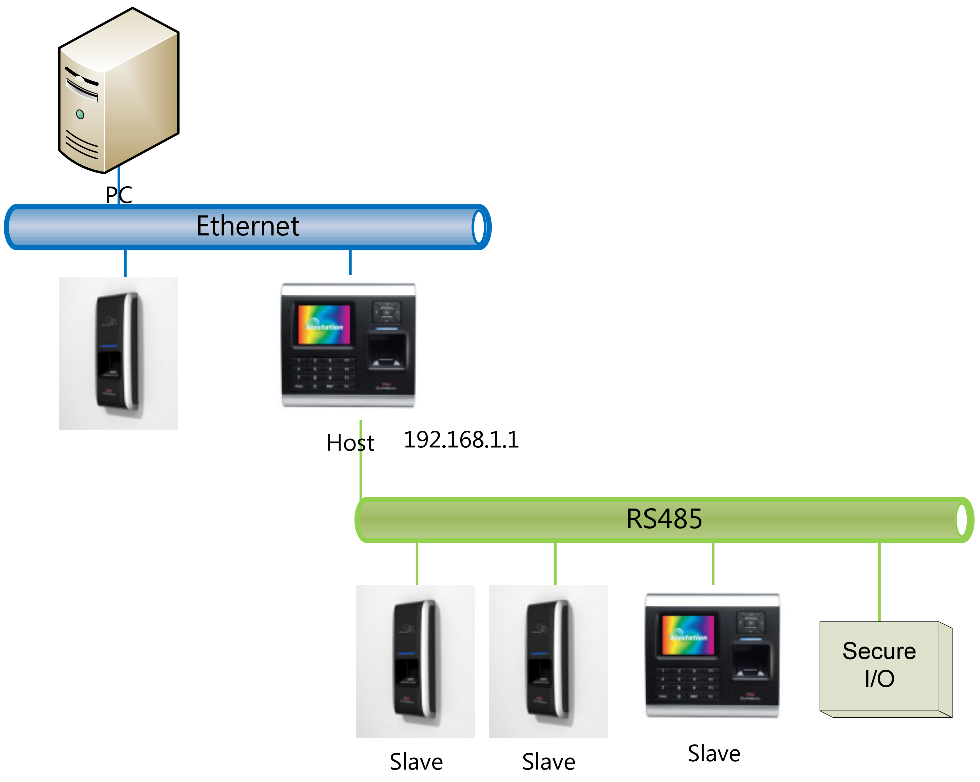
Return Values

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

Compatibility

BioStation(V1.5 or later)/BioEntry Plus(V1.2 or later)/BioLite Net

Example



```
// Open a socket to the host device
int handle;

BS_RET_CODE result = BS_OpenSocket( “192.168.1.1”, 1470, &handle );

unsigned deviceID;
int deviceType;

result = BS_GetDeviceID( handle, &deviceID, &deviceType );
result = BS_SetDeviceID( handle, deviceID, deviceType );

// Search the slave devices attached to the host
BS485SlaveInfo slaveInfo[8]; // maximum 8 slave devices;
int numOfSlave;
```

```
result = BS_Search485Slaves( handle, slaveInfo, &numOfSlave );

for( int i = 0; i < numOfSlave; i++ )
{
     if( slaveInfo.slaveID < 4 ) // it is a Secure I/O
     {
         // do something to the Secure I/Os
         continue;
     }

     BS_SetDeviceID( handle, slaveInfo[i].slaveID,
    slaveInfo[i].slaveType );

     // do something to the slave device
}
```

BS_SearchDeviceInLAN

Searches devices in LAN environment by UDP protocol. It sends a UDP broadcast packet to all the devices in a subnet. To call this function, a UDP handle should be acquired by **BS_OpenInternalUDP**.

BS_RET_CODE BS_SearchDeviceInLAN(int handle, int* numOfDevice, unsigned* deviceIDs, int* deviceTypes, unsigned* deviceAddrs)

Parameters

handle

Handle of the UDP socket returned by **BS_OpenInternalUDP**.

numOfDevice

Pointer to the number of devices to be returned.

deviceIDs

Pointer to the device IDs to be returned.

deviceTypes

Pointer to the device types to be returned.

deviceAddrs

Pointer to the IP addresses of the devices to be returned.

Return Values

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

Compatibility

BioStation(V1.5 or later)/BioEntry Plus/BioLite Net

Example

```
// Open a UDP socket
int udpHandle;

BS_RET_CODE result = BS_OpenInternalUDP( &udpHandle );

int numOfDevice;
unsigned deviceIDs[64];
int deviceTypes[64];
unsigned deviceAddrs[64];
```

```
result = BS_SearchDeviceInLAN( udpHandle, &numOfDevice, deviceIDs,
deviceTypes, deviceAddrs );

for( int i = 0; i < numOfDevice; i++ )
{
     int tcpHandle;

     char buf[32];
     sprintf( buf, "%d.%d.%d.%d", deviceAddrs[i] & 0xff, (deviceAddrs[i] &
0xff00) >> 8, (deviceAddrs[i] & 0xff0000) >> 16, (deviceAddrs[i] &
0xff000000) >> 24 );

     if( deviceTypes[i] == BS_DEVICE_BIOSTATION )
     {
         result = BS_OpenSocket( buf, 1470, &tcpHandle );
     }
     else if( deviceTypes[i] == BS_DEVICE_BIOENTRY_PLUS
             || deviceTypes[i] == BS_DEVICE_BIOLITE )
     {
         Result = BS_OpenSocket( buf, 1471, &tcpHandle );
     }

     BS_SetDeviceID( tcpHandle, deviceIDs[i], deviceTypes[i] );

     // do something

     BS_CloseSocket( tcpHandle );
}
```

BS_GetTime

Gets the time of a device. All the time values in this 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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_SetTime

Sets the time of a device.

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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

Example

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

BS_CheckSystemStatus

Checks if a device 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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_Reset

Resets a device.

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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_ResetUDP

Resets a device by UDP protocol. In some rare cases, you cannot connect to a device, even if you can search it in BioAdmin or BioStar. In those cases, you can reset it by this function.

BS_RET_CODE BS_ResetUDP(int handle, unsigned targetAddr, unsigned targetID)

Parameters

handle

Handle of the communication channel returned by **BS_OpenInternalUDP**.

targetAddr

IP address of the target device.

targetID

ID of the target device.

Return Values

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

Compatibility

BioStation(V1.5 or later)/BioEntry Plus/BioLite Net

BS_ResetLAN

Reestablishes the IP configuration of BioStation. When you call **BS_WriteIPConfig**, the changes are not taken into account immediately. If you want to reassign the IP address using the new configuration, you have to call **BS_ResetLAN**. On the contrary, BioEntry Plus and BioLite Net will reacquire the IP address automatically if its IP configuration is changed.

BS_RET_CODE BS_ResetLAN(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

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

Compatibility

BioStation

BS_UpgradeEx

Upgrades the firmware of a device. The device 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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_Disable

When communicating with a BioStation terminal, data corruption may occur if users are manipulating it 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.

Compatibility

BioStation

Example

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

if( result == BS_SUCCESS )
{
     result = BS_EnrollUserEx( … );
     // …
     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.

Compatibility

BioStation

BS_DisableCommunication

Disables all communication channels. After this function is called, the device will return BS_ERR_COM_DISABLED to all functions except for **BS_EnableCommunication**, **BS_GetDeviceID**, and search functions.

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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

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 of a BioStation terminal, please refer to the BioStation User Guide. You can change the master password of a BioEntry Plus or BioLite Net using **BS_ChangePasswordBEPlus()**.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_ChangePasswordBEPlus

Changes the master password of a BioEntry Plus or BioLite Net.

BS_RET_CODE BS_ChangePasswordBEPlus(int handle, const char* oldPassword, const char* newPassword)

Parameters

handle

Handle of the communication channel.

oldPassword

16 byte old password to be replaced. If it does not match, BS_ERR_WRONG_PASSWORD will be returned.

newPassword

16 byte new password.

Return Values

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

Compatibility

BioEntry Plus/BioLite Net

BS_FactoryDefault

Resets the status of a BioEntry Plus or BioLite Net to the factory default.

BS_RET_CODE BS_FactoryDefault(int handle, unsigned mask)

Parameters

handle

Handle of the communication channel.

mask

Mask	**Descriptions**
BS_FACTORY_DEFAULT_CONFIG	Resets system parameters.
BS_FACTORY_DEFAULT_USER	Delete all users.
BS_FACTORY_DEFAULT_LOG	Delete all log records.
BS_FACTORY_DEFAULT_LED	Resets LED/Buzzer configuration.

Return Values

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

Compatibility

BioEntry Plus/BioLite Net

Example

```
// Resets system parameters and deletes all users and log records
BS_RET_CODE result = BS_FactoryDefault( handle, BS_FACTORY_DEFAULT_CONFIG |
                    BS_FACTORY_DEFAULT_USER | BS_FACTORY_DEFAULT_LOG );
```

3.4. Log Management API

A BioStation terminal can store up to 500,000 log records, and a BioEntry Plus and BioLite Net up to 50,000 log records. They also provide APIs for real-time monitoring.

- BS_GetLogCount: gets the number of log records.
- BS_ClearLogCache: clears the log cache.
- BS_ReadLogCache: reads the log records in the cache.
- BS_ReadLog: reads log records.
- BS_ReadNextLog: reads log records in succession.
- 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	Device is turned on.
	TIME_SET	0xD2	System time is set.
Door	RELAY_ON	0x80	Door is opened. It is superseded by 0x8A and 0x8B since BioStation V1.4.
	RELAY_OFF	0x81	Door is closed.
	DOOR0_OPEN	0x82	Door 0 is opened.

	DOOR1_OPEN	0x83	Door 1 is opened.
	DOOR0_CLOSED	0x84	Door 0 is closed.
	DOOR1_CLOSED	0x85	Door 1 is closed.
	DOOR0_FORCED_OPEN	0x86	Door 0 is opened by force.
	DOOR1_FORCED_OPEN	0x87	Door 1 is opened by force.
	DOOR0_HELD_OPEN	0x88	Door 0 is held open too long.
	DOOR1_HELD_OPEN	0x89	Door 1 is held open too long.
	DOOR0_RELAY_ON	0x8A	The relay for Door 0 is activated.
	DOOR1_RELAY_ON	0x8B	The relay for Door 1 is activated.
	DOOR_HELD_OPEN_ALARM	0xE0	Door is held open too long.
	DOOR_FORCED_OPEN _ALARM	0xE1	Door is opened by force.
	DOOR_HELD_OPEN_ALARM _CLEAR	0xE2	Held open alarm is released.
	DOOR_FORCED_OPEN_ALARM _CLEAR	0xE3	Forced open alarm is released.
I/O	TAMPER_SW_ON	0x64	The case is opened.
	TAMPER_SW_OFF	0x65	The case is closed.
	DETECT_INPUT0	0x54	These are superseded by 0xA0 and 0xA1.
	DETECT_INPUT1	0x55	
	INTERNAL_INPUT0	0xA0	Detect a signal at internal input ports.
	INTERNAL_INPUT1	0xA1	
	SECONDARY_INPUT0	0xA2	Detect a signal at input ports of the slave device.
	SECONDARY_INPUT1	0xA3	
	SIO0_INPUT0	0xB0	Detect a signal at input ports of Secure I/O 0.
	SIO0_INPUT1	0xB1	
	SIO0_INPUT2	0xB2	

	SIO0_INPUT3	0xB3	
	SIO1_INPUT0	0xB4	Detect a signal at input ports of Secure I/O 1.
	SIO1_INPUT1	0xB5	
	SIO1_INPUT2	0xB6	
	SIO1_INPUT3	0xB7	
	SIO2_INPUT0	0xB8	Detect a signal at input ports of Secure I/O 2.
	SIO2_INPUT1	0xB9	
	SIO2_INPUT2	0xBA	
	SIO2_INPUT3	0xBB	
	SIO3_INPUT0	0xBC	Detect a signal at input ports of Secure I/O 3.
	SIO3_INPUT1	0xBD	
	SIO3_INPUT2	0xBE	
	SIO3_INPUT3	0xBF	
Access Control	IDENTIFY_NOT_GRANTED	0x6D	Access is not granted at this time.
	VERIFY_NOT_GRANTED	0x6E	
	NOT_GRANTED	0x78	
	APB_FAIL	0x73	Anti-passback is violated.
	COUNT_LIMIT	0x74	The maximum entrance count is reached already.
	TIME_INTERVAL_LIMIT	0x75	Time interval limitation is violated.
	INVALID_AUTH_MODE	0x76	The authentication mode is not supported at this time.
	EXPIRED_USER	0x77	User is not valid any more.
1:1 matching	VERIFY_SUCCESS	0x27	1:1 matching succeeds.
	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.
Mifare Card	CARD_ENROLL_SUCCESS	0x20	A Mifare card is written successfully.
	CARD_ENROLL_FAIL	0x21	Cannot write a Mifare card.
	CARD_VERIFY_DURESS	0x95	Duress finger is detected.
	CARD_VERIFY_SUCCESS	0x97	1:1 matching succeeds.
	CARD_VERIFY_FAIL	0x98	1:1 matching fails.
	CARD_APB_FAIL	0x99	Anti-passback is violated.
	CARD_COUNT_LIMIT	0x9A	The maximum entrance count is reached already.
	CARD_TIME_INTERVAL_LIMIT	0x9B	Time interval limitation is violated.
	CARD_INVALID_AUTH_MODE	0x9C	The authentication mode is not supported at this time.
	CARD_EXPIRED_USER	0x9D	User is not valid any more.
	CARD_NOT_GRANTED	0x9E	Not allowed to enter.
	BLACKLISTED	0xC2	User is blacklisted.
Zone	ARMED	0xC3	Alarm zone is armed.
	DISARMED	0xC4	Alarm zone is disarmed.
	ALARM_ZONE_INPUT	0xC5	An input point is

			activated in an armed zone.
	FIRE_ALARM_ZONE_INPUT	0xC6	An input point is activated in a fire alarm zone.
	ALARM_ZONE_INPUT _CLEAR	0xC7	The alarm is released.
	FIRE_ALARM_ZONE_INPUT _CLEAR	0xC8	The fire alarm is released.
	APB_ZONE_ALARM	0xC9	Anti-passback is violated.
	ENTLIMIT_ZONE_ALARM	0xCA	Entrance limitation is violated.
	APB_ZONE_ALARM_CLEAR	0xCB	Anti-passback alarm is released.
	ENTLIMIT_ZONE_ALARM _CLEAR	0xCC	Entrance limitation alarm is released.
Network	SOCK_CONN	0xD3	Connection is established from PC.
	SOCK_DISCONN	0xD4	Connection is closed.
	SERVER_SOCK_CONN	0xD5	Connected to BioStar server.
	SERVER_SOCK_DISCONN	0xD6	Disconnected from BioStar server.
	LINK_CONN	0xD7	Ethernet link is connected.
	LINK_DISCONN	0xD8	Ethernet link is disconnected.
	INIT_IP	0xD9	IP configuration is initialized.
	INIT_DHCP	0xDA	DHCP is initialized.
	DHCP_SUCCESS	0xDB	Acquired an IP address from the DHCP server.

Table 5 Log Event Types

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.

5. *reserved2*

   When the log synchronization option is on in a zone, the log records of the member devices will be stored in the master device, too. In this case, this field will be used for the device ID. Otherwise, this field should be 0.

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 device.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_ClearLogCache

BioStation, BioEntry Plus, and BioLite Net have a cache which keeps 128 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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

Example

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

BSLogRecord logRecords[128];
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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

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. As for BioEntry Plus and BioLite Net, which can store up to 50,000 log records, the maximum number is 8,192. Therefore, users should call **BS_ReadLog** repetitively if the number of log records in the time interval is larger than these limits.

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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

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_ReadNextLog

BS_ReadNextLog searches log records starting from the last record read by **BS_ReadLog** or **BS_ReadNextLog**. It is useful for reading lots of log records in succession.

BS_RET_CODE BS_ReadNextLog(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, it will be ignored.

endTime

End time of the interval. If it is set to 0, it will be ignored.

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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

Example

```
// read all the log records from a BioEntry Plus
const int MAX_LOG = 50000; // 500000 for BioStation
const int MAX_READ_LOG = 8192; // 32768 for BioStation

int numOfReadLog = 0;
int numOfLog = 0;
```

```
BSLogRecord* logRecord = (BSLogRecord*)malloc( MAX_LOG );

BS_RET_CODE result = BS_ReadLog( handle, 0, 0, &numOfReadLog, logRecord +
numOfLog  );

while( result == BS_SUCCESS )
{
     numOfLog += numOfReadLog;

     if( numOfReadLog < MAX_READ_LOG ) // end of the log
     {
         break;
     }

     result = BS_ReadNextLog( handle, 0, 0, &numOfReadLog, logRecord +
     numOfLog );
}
```

BS_DeleteLog

Deletes oldest log records. Please note that BioEntry Plus and BioLite Net support only **BS_DeleteAllLog()**.

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.

Compatibility

BioStation

BS_DeleteAllLog

Deletes all log records.

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

Parameters

handle

Handle of the communication channel.

numOfLog

This filed is ignored.

numOfDeletedLog

Pointer to the number of deleted log records.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

3.5. 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_DeleteSound: clears a sound effect.
- 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_BACKGROUND_LOGO and BS_BACKGROUND_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.

Compatibility

BioStation

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.

Compatibility

BioStation

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.

Compatibility

BioStation

BS_SetSound

There are 15 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
BS_SOUND_FINGER_ONLY	When waiting for fingerprint
BS_SOUND_PIN_ONLY	When waiting for password
BS_SOUND_CARD_ONLY	When waiting for card
BS_SOUND_FINGER_PIN	When waiting for fingerprint or password
BS_SOUND_FINGER_CARD	When waiting for fingerprint or card
BS_SOUND_TNA_F1	When authentication succeeds after F1 button is pressed
BS_SOUND_TNA_F2	When authentication succeeds after F2 button is pressed
BS_SOUND_TNA_F3	When authentication succeeds after F3 button is pressed

BS_SOUND_TNA_F4	When authentication succeeds after F4 button is pressed

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.

Compatibility

BioStation

BS_DeleteSound

Clears the sound file set by **BS_SetSound**.

BS_RET_CODE BS_DeleteSound(int handle, int soundIndex)

Parameters

handle

Handle of the communication channel.

soundIndex

Index of the sound effect. See **BS_SetSound**.

Return Values

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

Compatibility

BioStation

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.

Compatibility

BioStation

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.

Compatibility

BioStation

3.6. User Management API

These APIs provide user management functions such as enroll and delete. Note that the user header structures of BioStation, BioEntry Plus, and BioLite Net are different. See the **Compatibility** section of each API to choose the right function.

- BS_GetUserDBInfo: gets the basic information of the user DB.
- BS_EnrollUserEx: enrolls a user to BioStation.
- BS_EnrollMultipleUserEx: enrolls multiples users to BioStation.
- BS_EnrollUserBEPlus: enrolls a user to BioEntry Plus or BioLite Net.
- BS_EnrollMultipleUserBEPlus: enrolls multiple users to BioEntry Plus or BioLite Net.
- BS_GetUserEx: gets the fingerprint templates and header information of a user from BioStation.
- BS_GetUserInfoEx: gets the header information of a user from BioStation.
- BS_GetAllUserInfoEx: gets the header information of all users from BioStation.
- BS_GetUserBEPlus: gets the fingerprint templates and header information of a user from BioEntry Plus or BioLite Net.
- BS_GetUserInfoBEPlus: gets the header information of a user from BioEntry Plus or BioLite Net.
- BS_GetAllUserInfoBEPlus: gets the header information of all users from BioEntry Plus or BioLite Net.
- BS_DeleteUser: deletes a user.
- BS_DeleteMultipleUsers: deletes multiple users.
- BS_DeleteAllUser: deletes all users.
- BS_SetPrivateInfo: sets the private information of a user.
- BS_GetPrivateInfo: gets the private information of a user.
- BS_GetAllPrivateInfo: gets the private information of all users.
- BS_ScanTemplate: scans a fingerprint on a device and retrieves the template of it.
- BS_ReadCardIDEx: reads a RF card on a device and retrieves the id of it.
- BS_ReadImage: reads an image of the last scanned fingerprint.

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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_EnrollUserEx

Enrolls a user to BioStation. Maximum 5 fingers can be enrolled per user.

BS_RET_CODE BS_EnrollUserEx(int handle, BSUserHdrEx* hdr, unsigned char* templateData)

Parameters

handle

Handle of the communication channel.

Hdr

BSUserHdrEx 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];
    unsigned short authMode;
    unsigned short authLimitCount; // 0 for no limit
    unsigned short reserved;
    unsigned short timedAntiPassback; // in minutes. 0 for no limit
    unsigned cardID; // 0 for not used
    bool bypassCard;
    bool disabled;
    unsigned expireDateTime;
    int customID; //card Custom ID
    int version; // card Info Version
    unsigned startDateTime;
} BSUserHdrEx;
```

The key fields and their available options are as follows.

Fields	**Descriptions**
adminLevel	BS_USER_ADMIN
	BS_USER_NORMAL

securityLevel	It specifies the security level used for 1:1 matching only. BS_USER_SECURITY_DEFAULT: same as the device setting BS_USER_SECURITY_LOWER: 1/1000 BS_USER_SECURITY_LOW: 1/10,000 BS_USER_SECURITY_NORMAL: 1/100,000 BS_USER_SECURITY_HIGH: 1/1,000,000 BS_USER_SECURITY_HIGHER: 1/10,000,000
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.
authMode	Specify the authentication mode of this user. The **usePrivateAuthMode** of **BSOPModeConfig** should be true for this authentication mode to be effective. Otherwise, the authentication mode of the device will be applied to all users. BS_AUTH_MODE_DISABLED[3] BS_AUTH_FINGER_ONLY

[3] The authentication mode of the device will be applied to this user.

	BS_AUTH_FINGER_N_PASSWORD BS_AUTH_FINGER_OR_PASSWORD BS_AUTH_PASS_ONLY BS_AUTH_CARD_ONLY
authLimitCount	Specifies how many times the user is permitted to access per day. If it is 0, there is no limit.
timedAntiPassback	Specifies the minimum time interval for which the user can access the device only once. If it is 0, there is no limit.
cardID	4 byte card ID. The RF card ID is comprised of 4 byte card ID and 1 byte custom ID.
bypassCard	If it is true, the user can access without fingerprint authentication.
disabled	If it is true, the user cannot access the device all the time.
expireDateTime	The date on which the user's authorization expires.
customID	1 byte custom ID of the card.
version	The version of the card information format.
startDateTime	The date from which the user's authorization takes effect.

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.

Compatibility

BioStation

Example

```
BSUserHdrEx userHeader;
```

```
// initialize header
memset( &userHdr, 0, sizeof( BSUserHdrEx ) );
userHdr.ID = 1; // 0 cannot be assigned as a user ID
userHdr.startDateTime = 0; // no check for start date
userHdr.expireDateTime = 0; // no check for expiry date
userHeader.adminLevel = BS_USER_NORMAL;
userHeader.securityLevel = BS_USER_SECURITY_DEFAULT;
userHeader.authMode = BS_AUTH_MODE_DISABLED; // use the authentication mode
                                      // of the device
userHeader.accessGroupMask = 0xffff0201; // a member of Group 1 and Group
2;
strcpy( userHeader.name, "John" );
strcpy( userHeader.departments, "RND" );
strcpy( userHeader.password, "" ); // no password is enrolled. Password
                              // should be longer than 4 bytes.

// read card IDs
BS_RET_CODE result = BS_ReadCardIDEx( handle, &userHeader.cardID,
&userHdr.customID );
userHdr.version = CARD_INFO_VERSION;
userHdr.bypassCard = 0;

// scan templates
userHeader.numOfFinger = 2;
unsigned char* templateBuf = (unsigned char*)malloc( userHeader.numOfFinger
* 2 * BS_TEMPLATE_SIZE );

int bufPos = 0;
for( int i = 0; i < userHeader.numOfFinger * 2; i++ )
{
     result = BS_ScanTemplate( handle, templateBuf + bufPos );
     bufPos += BS_TEMPLATE_SIZE;
}
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];
     }
}

// enroll the user
result = BS_EnrollUserEx( handle, &userHeader, templateBuf );
```

BS_EnrollMultipleUserEx

Enrolls multiple users to BioStation. By combining user information, the enrollment time will be reduced.

BS_RET_CODE BS_EnrollMultipleUserEx(int handle, int numOfUser, BSUserHdrEx* hdr, unsigned char* templateData)

Parameters

handle

Handle of the communication channel.

numOfUser

Number of users to be enrolled.

hdr

Array of user headers to be enrolled.

templateData

Fingerprint templates of the all users.

Return Values

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

Compatibility

BioStation

Example

```
int numOfUser = 2;
BSUserHdrEx hdr1, hdr2;
unsigned char *templateBuf1, *templateBuf2;

// fill the header and template data here
// …

BSUserHdrEx* hdr = (BSUserHdrEx*)malloc( numOfUser *
sizeof( BSUserHdrEx ) );
unsigned char* templateBuf = (unsigned char*)malloc( hdr1.numOfFinger * 2 *
BS_TEMPLATE_SIZE + hdr2.numOfFinger * 2 * BS_TEMPLATE_SIZE );
```

```
memcpy( hdr, &hdr1, sizeof( BSUserHdrEx ) );
memcpy( hdr + sizeof( BSUserHdrEx ), &hdr2, sizeof( BSUserHdrEx ) );

memcpy( templateBuf, templateBuf1, hdr1.numOfFinger * 2 *
BS_TEMPLATE_SIZE );
memcpy( templateBuf + hdr1.numOfFinger * 2 * BS_TEMPLATE_SIZE, templateBuf2,
hdr2.numOfFinger * 2 * BS_TEMPLATE_SIZE );

BS_RET_CODE result = BS_EnrollMultipleUserEx( handle, numOfUser, hdr,
templateBuf );
```

BS_EnrollUserBEPlus

Enrolls a user to BioEntry Plus or BioLite Net. Maximum 2 fingers can be enrolled per user. The only difference between BioEntry Plus and BioLite Net is that only the latter uses the password field.

BS_RET_CODE BS_EnrollUserBEPlus(int handle, BEUserHdr* hdr, unsigned char* templateData)

Parameters

handle

Handle of the communication channel.

Hdr

BEUserHdr is defined as follows.

```
typedef struct {
    int version;
    unsigned userID;
    time_t startTime;
    time_t expiryTime;
    unsigned cardID;
    unsigned char cardCustomID;
    unsigned char commandCardFlag;
    unsigned char cardFlag;
    unsigned char cardVersion;
    unsigned short adminLevel;
    unsigned short securityLevel;
    unsigned accessGroupMask;
    unsigned short numOfFinger; // 0, 1, 2
    unsigned short fingerChecksum[2];
    unsigned char isDuress[2];
    int disabled;
    int opMode;
    int dualMode;
    char password[16]; // for BioLite Net only
    int reserved2[15];
} BEUserHdr;
```

The key fields and their available options are as follows.

Fields	**Descriptions**
version	0x01.
userID	User ID.

startTime	The time from which the user's authorization takes effect.
expiryTime	The time on which the user's authorization expires.
cardID	4 byte card ID. The RF card ID is comprised of 4 byte card ID and 1 byte custom ID.
cardCustomID	1 byte custom ID which makes up the RF card ID with **cardID**.
commandCardFlag	Reserved for future use.
cardFlag	NORMAL_CARD BYPASS_CARD
cardVersion	CARD_VERSION_1
adminLevel	USER_LEVEL_NORMAL USER_LEVEL_ADMIN
securityLevel	It specifies the security level used for 1:1 matching only. USER_SECURITY_DEFAULT: same as the device setting. USER_SECURITY_LOWER: 1/1000 USER_SECURITY_LOW: 1/10,000 USER_SECURITY_NORMAL: 1/100,000 USER_SECURITY_HIGH: 1/1,000,000 USER_SECURITY_HIGHER: 1/10,000,000
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.
numOfFinger	The number of enrolled fingers.
fingerChecksum	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.
isDuress	Under duress, users can authenticate with a duress finger to notify the threat. When

	duress finger is detected, the device will write a log record and output specified signals.
disabled	If it is true, the user cannot access the device all the time. It is useful for disabling users temporarily.
opMode	Specify the authentication mode of this user. The **opModePerUser** of **BEConfigData** should be true for this authentication mode to be effective. Otherwise, the authentication mode of the device will be applied. BS_AUTH_MODE_DISABLED[4] BS_AUTH_FINGER_ONLY BS_AUTH_FINGER_N_PASSWORD BS_AUTH_FINGER_OR_PASSWORD BS_AUTH_PASS_ONLY BS_AUTH_CARD_ONLY
dualMode	Reserved for future use.
password	16 byte password for BioLite Net.

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.

Compatibility

BioEntry Plus/BioLite Net

Example

```
BEUserHdr userHeader;

// initialize header
memset( &userHeader, 0, sizeof( BEUserHdr ) );
```

[4] The authentication mode of the device will be applied to this user.

```
userHeader.version = 0x01;
userHeader.userID = 0x01;
userHeader.startTime = 0; // no start time check
userHeader.expiryTime = US_ConvertToLocalTime( time( NULL ) ) + 365 * 24 *
60 * 60; // 1 year from today
userHeader.adminLevel = BEUserHdr::USER_LEVEL_NOMAL;
userHeader.securityLevel = BEUserHdr::USER_SECURITY_DEFAULT;
userHeader.accessGroupMask = 0xffff0201; // a member of Group 1 and Group
2;
userHeader.opMode = BS_AUTH_MODE_DISABLED; // use the authentication mode
                                       // of the device

// read card IDs
BS_RET_CODE result = BS_ReadCardIDEx( handle, &userHeader.cardID,
&userHdr.cardCustomID );
userHdr.cardVersion = BEUserHdr::CARD_VERSION_1;
userHdr.cardFlag = BEUserHdr::NORMAL_CARD;

// scan templates
userHeader.numOfFinger = 2;
unsigned char* templateBuf = (unsigned char*)malloc( userHeader.numOfFinger
* 2 * BS_TEMPLATE_SIZE );

int bufPos = 0;
for( int i = 0; i < userHeader.numOfFinger * 2; i++ )
{
     result = BS_ScanTemplate( handle, templateBuf + bufPos );
     bufPos += BS_TEMPLATE_SIZE;
}

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

```
    result = BS_EnrollUserBEPlus( handle, &userHeader, templateBuf );
```

BS_EnrollMultipleUserBEPlus

Enrolls multiple users to BioEntry Plus or BioLite Net. By combining user information, you can reduce the enrollment time.

BS_RET_CODE BS_EnrollMultipleUserBEPlus(int handle, int numOfUser, BEUserHdr* hdr, unsigned char* templateData)

Parameters

handle

Handle of the communication channel.

numOfUser

Number of users to be enrolled.

hdr

Array of user headers to be enrolled.

templateData

Fingerprint templates of the all users.

Return Values

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

Compatibility

BioEntry Plus/BioLite Net

Example

```
See the Example of BS_EnrollMultipleUserEx.
```

BS_GetUserEx

Retrieves the header information and template data of a user from BioStation.

BS_RET_CODE BS_GetUserEx(int handle, unsigned userID, BSUserHdrEx* 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.

Compatibility

BioStation

BS_GetUserInfoEx

Retrieves the header information of a user from BioStation.

BS_GetUserInfoEx(int handle, unsigned userID, BSUserHdrEx* 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.

Compatibility

BioStation

BS_GetAllUserInfoEx

Retrieves the header information of all enrolled users from BioStation.

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

Parameters

handle

Handle of the communication channel.

hdr

Pointer to the **BSUserHdrEx** 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. If there is no user, return BS_ERR_NOT_FOUND. Otherwise, return the corresponding error code.

Compatibility

BioStation

BS_GetUserBEPlus

Retrieves the header information and template data of a user from BioEntry Plus or BioLite Net.

BS_RET_CODE BS_GetUserBEPlus(int handle, unsigned userID, BEUserHdr* 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.

Compatibility

BioEntry Plus/BioLite Net

BS_GetUserInfoBEPlus

Retrieves the header information of a user from BioEntry Plus or BioLite Net.

BS_RET_CODE BS_GetUserInfoBEPlus(int handle, unsigned userID, BEUserHdr* 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.

Compatibility

BioEntry Plus/BioLite Net

BS_GetAllUserInfoBEPlus

Retrieves the header information of all enrolled users from BioEntry Plus or BioLite Net.

BS_RET_CODE BS_GetAllUserInfoBEPlus(int handle, BEUserHdr* hdr, int *numOfUser)

Parameters

handle

Handle of the communication channel.

hdr

Pointer to the **BEUserHdr** 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. If there is no user, return BS_ERR_NOT_FOUND. Otherwise, return the corresponding error code.

Compatibility

BioEntry Plus/BioLite Net

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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_DeleteMultipleUsers

Deletes multiple users.

BS_RET_CODE BS_DeleteMultipleUsers(int handle, int numberOfUser, unsigned* userID)

Parameters

handle

Handle of the communication channel.

numberOfUser

Number of users to be deleted.

userID

Array of user IDs 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.

Compatibility

BioStation(V1.5 or later)/BioEntry Plus(V1.2 or later)/BioLite Net

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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_SetPrivateInfo

Set the private information of the specified user. The private information includes greeting messages and customized images

BS_RET_CODE BS_SetPrivateInfo(int handle, int type, const BSPrivateInfo* privateInfo, const char* imagePath)

Parameters

handle

Handle of the communication channel.

privateInfo

BSPrivateInfo is defined as follows.

```
typedef struct{
    unsigned ID;
    char department[BS_MAX_NAME_LEN + 1];
    char greetingMsg[BS_MAX_PRIVATE_MSG_LEN + 1];
    int useImage;
    unsigned duration;
    unsigned countPerDay;
    unsigned imageChecksum;
    int reserved[4];
} BSPrivateInfo;
```

The key fields and their available options are as follows.

Fields	**Descriptions**
ID	User ID
department	Department name
greetingMsg	The greeting message to be shown when the user is authenticated.
useImage	If it is true, the specified image will be shown with the greeting message.
duration	The duration for which the private information is displayed.
countPerDay	The maximum display count per day.
imageChecksum	The checksum of the private image.

imagePath

Path of the private image.

Return Values

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

Compatibility

BioStation

BS_GetPrivateInfo

Get the private information of the specified user.

BS_RET_CODE BS_GetPrivateInfo(int handle, BSPrivateInfo* privateInfo)

Parameters

handle

Handle of the communication channel.

privateInfo

Pointer to the private information to be returned.

Return Values

If the function is successful, return BS_SUCCESS. Otherwise, return the corresponding error code.

Compatibility

BioStation

BS_GetAllPrivateInfo

Get the private information of all users.

BS_RET_CODE BS_GetAllPrivateInfo(int handle, BSPrivateInfo* privateInfo, int* numOfUser)

Parameters

handle

Handle of the communication channel.

privateInfo

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

numOfUser

Pointer to the number of users having the private information.

Return Values

If the function is successful, return BS_SUCCESS. Otherwise, return the corresponding error code.

Compatibility

BioStation

BS_ScanTemplate

Scans a fingerprint on a BioStation, BioEntry Plus, or BioLite Net and retrieves the template of it. This function is useful when the device 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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_ReadCardIDEx

Read a card on a BioStation or BioEntry Plus and retrieve the ID of it.
This function is useful when the device is used as an enrollment station. For BioLite Net, BioStation Mifare, and BioEntry Plus Mifare, it returns the CSN of the Mifare card.

BS_RET_CODE BS_ReadCardIDEx(int handle, unsigned int* cardID, int* customID)

Parameters

handle

Handle of the communication channel.

cardID

Pointer to the 4 byte card ID to be returned. As for Mifare models, it returns the 4 byte CSN.

customID

Pointer to the 1 byte custom ID to be returned. As for Mifare models, it will be always 0.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_ReadImage

Reads an image of the last scanned fingerprint. This function is useful when the device is used as an enroll station.

BS_RET_CODE BS_ReadImage(int handle, int imageType, unsigned char* bitmapImage, int* imageLen)

Parameters

handle

Handle of the communication channel.

imageType

This field plays different roles depending on the device type. For BioStation, it specifies the image type as follows;

0 - binary image, 1 - gray image.

For BioEntry Plus or BioLite Net, it specifies whether to scan new image or not. If it is 0xff, BioEntry Plus or BioLite Net returns the last scanned image in gray format. Otherwise, it will wait for new fingerprint input and returns the image of it in gray format.

bitmapImage

Pointer to the image data to be returned. The bimtmapImage should be allocated before calling this function.

imageLen

Pointer to the length of the image data to be returned.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

3.7. Configuration API

These APIs provide functionalities for reading/writing system configurations. As for BioStation, each configuration has separate data structure. On the contrary, BioEntry Plus and BioLite have much smaller number of data structures. See the **Compatibility** section of each API to choose the right function.

- BS_ReadSysInfoConfig: reads the system information of BioStation.
- BS_WriteDisplayConfig: configures the display settings of BioStation.
- BS_ReadDisplayConfig
- BS_WriteOPModeConfig: configures the authentication mode of BioStation.
- BS_ReadOPModeConfig
- BS_WriteTnaEventConfig: customizes the TNA event settings of BioStation.
- BS_ReadTnaEventConfig
- BS_WriteTnaEventExConfig: customizes the TNA mode settings of BioStation.
- BS_ReadTnaEventExConfig
- BS_WriteIPConfig: configures the IP parameters of BioStation.
- BS_ReadIPConfig
- BS_WriteWLANConfig: configures the wireless LAN parameters of BioStation.
- BS_ReadWLANConfig
- BS_WriteFingerprintConfig: configures the settings related to fingerprint matching.
- BS_ReadFingerprintConfig
- BS_WriteIOConfig: configures the input and output ports of BioStation.
- BS_ReadIOConfig
- BS_WriteSerialConfig: configures the serial mode of BioStation.
- BS_ReadSerialConfig
- BS_Write485NetworkConfig: configures the RS485 mode of BioStation.
- BS_Read485NetworkConfig
- BS_WriteUSBConfig: configures the USB mode of BioStation.
- BS_ReadUSBConfig
- BS_WriteEncryptionConfig: configures the encryption setting of BioStation.
- BS_ReadEncryptionConfig
- BS_WriteWiegandConfig: configures the Wiegand format of BioStation.

- BS_ReadWiegandConfig
- BS_WriteZoneConfigEx: configures the zones.
- BS_ReadZoneConfigEx
- BS_WriteDoorConfig: configures the doors.
- BS_ReadDoorConfig
- BS_WriteInputConfig: configures the input ports.
- BS_ReadInputConfig
- BS_WriteOutputConfig: configures the output ports.
- BS_ReadOutputConfig
- BS_WriteEntranceLimitConfig: configures the entrance limitation settings.
- BS_ReadEntranceLimitConfig
- BS_WriteConfig: configures the settings of BioEntry Plus or BioLite Net.
- BS_ReadConfig
- BS_GetAvailableSpace: calculates the available space of a device.

Corruption of some configurations might result in serious consequence – it might make the device unbootable. To minimize the risk, you had better follow the guidelines shown below;

(1) Read the configuration first before overwriting it. Then, change only the required fields.

(2) Read carefully the description of each field in a structure. If you are not sure what the field is about, do not change it.

BS_ReadSysInfoConfig

Reads the system information of BioStation.

BS_RET_CODE BS_ReadSysInfoConfig(int handle, BSSysInfoConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSSysInfoConfig is defined as follows;

```
typedef struct {
    unsigned ID;
    char macAddr[32];
    char productName[32];
    char boardVer[16];
    char firmwareVer[16];
    char blackfinVer[16];
    char kernelVer[16];
    int language;
    char reserved[32];
} BSSysInfoConfig;
```

Return Values

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

Compatibility

BioStation

BS_WriteDisplayConfig / BS_ReadDisplayConfig

Write/read 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 reserved1;
    int timeout; // menu timeout in seconds, 0 for infinite
    int volume; // 0(mute) ~ 100
    int msgTimeout;
    int usePrivateAuth; // private authentication : 1 – use, 0 – don’t use
    int dateType;
    int reserved2[7];
} 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.
msgTimeout	BS_MSG_TIMEOUT_500MS – 0 sec
	BS_MSG_TIMEOUT_1000MS – 1 sec

	BS_MSG_TIMEOUT_2000MS – 2 sec
	BS_MSG_TIMEOUT_3000MS – 3 sec
	BS_MSG_TIMEOUT_4000MS – 4 sec
	BS_MSG_TIMEOUT_5000MS – 5 sec
dateType	BS_UI_DATE_TYPE_AM – DD/MM
	BS_UI_DATE_TYPE_EU – MM/DD

Return Values

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

Compatibility

BioStation

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

Write/read 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;
    int tnaChange;
    unsigned char authSchedule[MAX_AUTH_COUNT];
    unsigned char identificationSchedule;
    unsigned char dualMode;
    unsigned char dualSchedule;
    unsigned char version;
    int cardMode;
    unsigned char authScheduleEx[MAX_AUTH_EX_COUNT];
    unsigned char usePrivateAuthMode;
    char reserved[2];
} 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_N_PASSWORD – both the fingerprint and password authentication are required. BS_AUTH_FINGER_OR_PASSWORD – both the fingerprint and password authentication are

	allowed. BS_AUTH_PASS_ONLY – only the password authentication is allowed. BS_AUTH_CARD_ONLY – only the card 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.
tnaChange	BS_TNA_AUTO_CHANGE – TNA event is changed automatically according to the schedule defined in **BSTnaEventExConfig**. BS_TNA_MANUAL_CHANGE – TNA event is changed manually by function keys. BS_TNA_FIXED – TNA event is fixed to the **fixedTnaIndex** of **BSTnaEventExConfig**.
authSchedule	The schedule of each authentication mode, during which the mode is effective. For example, authSchedule[FINGER_INDEX] specifies the schedule, during which BS_AUTH_FINGER_ONLY mode is enabled. Note that you have to use **authScheduleEx** for BS_AUTH_FINGER_N_PASSWORD mode.
identificationSchedule	Specifies the schedule, during which the 1:N mode is enabled.
dualMode	If it is true, two users should be authenticated before the door is opened.
dualSchedule	Specifies the schedule, during which the **dualMode** is enabled.

version	Reserved for future use.
cardMode	Specifies the operation mode of Mifare models. BS_COMMON_DISABLE – Ignores Mifare cards. BS_OP_CARD_CSN – Reads only the 4 byte CSN of Mifare cards. BS_OP_CARD_TEMPLATE – Reads templates from Mifare cards.
authScheduleEx	The schedule of BS_AUTH_FINGER_N_PASSWORD.
usePrivateAuthMode	If true, the **authMode** field of **BSUserHdrEx** will be applied to user authentication. Otherwise, the **authMode** of the **BSOPModeConfig** will be applied to all users.

Return Values

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

Compatibility

BioStation

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];
     unsigned short reserved[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.

Compatibility

BioStation

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_WriteTnaEventExConfig / BS_ReadTnaEventExConfig

Writes/reads the TNA mode configurations. Refer to **BS_WriteTnaEventConfig** for the related settings.

BS_RET_CODE BS_WriteTnaEventExConfig(int handle, BSTnaEventExConfig* config)
BS_RET_CODE BS_ReadTnaEventExConfig(int handle, BSTnaEventExConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSTnaEventExConfig is defined as follows;

```
typedef struct {
    int fixedTnaIndex;
    int manualTnaIndex;
    int timeSchedule[BS_MAX_TNA_FUNCTION_KEY];
} BSTnaEventExConfig;
```

The key fields and their available options are as follows;

Fields	Options
fixedTnaIndex	Specifies the fixed TNA event. It is effective only if the **tnaChange** field of **BSOPModeConfig** is BS_TNA_FIXED.
manualTnaIndex	Reserved for future use.
timeSchedule	Schedules for each TNA event. It is effective only if the **tnaChange** field of **BSOPModeConfig** is BS_TNA_AUTO_CHANGE.

Return Values

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

Compatibility

BioStation

BS_WriteIPConfig/BS_ReadIPConfig

Writes/reads the IP configuration. Before configuring parameters, you have to decide on two important options.

(1) DHCP: There are two ways to assign an IP address to a device – DHCP or static IP. DHCP makes network configuration much easier. You don't have to configure other parameters such as subnet mask and gateway. If your LAN has a DHCP server, all you have to do is to plug an Ethernet cable to the device. By default, each device is set to use DHCP mode.

However, DHCP has its own problem. The IP address of a device can be changed. When an IP address is assigned by a DHCP server, it has limited lease time. Before the lease time expires, the device has to reacquire an IP address. Depending on the configuration of DHCP server, the new IP address can be different from the old one. Since the application doesn't know this change, it will result in connection loss.

(2) Server/Direct mode: The connection between applications and devices has two modes – direct and server. The server mode is only for BioStar server. Therefore, you have to use direct mode if you want to connect to the device in your applications.

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;
     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];
        int  maxConnection;
        unsigned char useServer;
        unsigned serverPort;
        bool syncTimeWithServer;
        char reserved[48];
    } BSIPConfig;
```

The key fields and their available options are as follows;

Fields	**Options**
lanType	BS_IP_DISABLE BS_IP_ETHERNET BS_IP_WLAN
useDHCP	If it is true, the **ipAddr**, **gateway**, and **subnetMask** fields will be ignored.
port	The default value is 1470. You don't have to change it in most cases.
ipAddr	IP address of the device.
subnetMask	Subnet mask.
serverIP	If **useServer** is true, you have to configure the IP address and port of the server.
maxConnection	The maximum number of TCP sockets you can connect to.
useServer	It should be false for connecting to devices using the SDK.
serverPort	The port number of the server.
syncTimeWithServer	If **useServer** is true, the device will synchronize its time with that of the server.

Return Values

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

Compatibility

BioStation

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];
    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.

Compatibility

BioStation

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_WriteFingerprintConfig / BS_ReadFingerprintConfig

Write/read the configurations associated with 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; // 0 for indefinite, 1 ~ 20 sec
    int imageQuality;
    bool viewImage;
    int freeScanDelay;
    int useCheckDuplicate;
    int matchTimeout;
    short useSIF;
    short useFakeDetect;
    bool useServerMatching;
    char reserved[3];
} 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 above security setting. BS_USER_SECURITY_USER – security level for 1:1 matching is defined by the **securityLevel** of **BSUserHdrEx** per each user.
fastMode	BS_FAST_MODE_NORMAL BS_FAST_MODE_FAST BS_FAST_MODE_FASTER BS_FAST_MODE_AUTO
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
freeScanDelay	Specifies the delay in seconds between consecutive identification processes. BS_FREESCAN_0 BS_FREESCAN_1 BS_FREESCAN_2 BS_FREESCAN_3 BS_FREESCAN_4 BS_FREESCAN_5 BS_FREESCAN_6 BS_FREESCAN_7 BS_FREESCAN_8 BS_FREESCAN_9 BS_FREESCAN_10
useCheckDuplicate	If true, the device will check if the same fingerprint was registered already before enrolling new users.
matchTimeout	Matching timeout in seconds.

useSIF	If true, ISO 19794-2 template format is used instead of Suprema's.
useFakeDetect	If true, the device will try to detect fake fingers.
useServerMatching	In server matching mode, user authentication is handled by BioStar server, not each device. To use server matching, the **useServer** of **BSIPConfig** should be true.

Return Values

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

Compatibility

BioStation

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;
    int inputDuration[BS_NUM_OF_INPUT];
    int inputSchedule[BS_NUM_OF_INPUT];
    short inputType[BS_NUM_OF_INPUT];
    int reserved[58];
} 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_CARD – use two inputs ports as Wiegand input. Input data is processed as card id. BS_IO_INPUT_WIEGAND_USER – use two inputs ports as Wiegand input. Input data is processed as user id.
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_USER – outputs user id as Wiegand string when authentication succeeds. BS_IO_OUTPUT_WIEGAND_CARD – outputs card id as 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.
outputDuration	Specifies the duration of output signal in milliseconds.
inputDuration inputSchedule inputType	These fields are deprecated. You have to use **BSInputConfig** instead. See **BS_WriteInputConfig**.

Return Values

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

Compatibility

BioStation

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;
    int rs232;
    int useSecureIO;
    char activeSecureIO[4]; // 0 ~ 3 - byte[0] ~ byte[3]
    unsigned slaveID;
    int deviceType;
    int reserved[2];
} BSSerialConfig;
```

The key fields and their available options are as follows;

Fields	Options
rs485	BS_CHANNEL_DISABLED or the baudrate of RS485 port. The default value is 115,200bps.
rs232	BS_CHANNEL_DISABLED or the baudrate of RS232 port. The default value is 115,200bps.
useSecureIO activeSecureIO slaveID deviceType	These fields are deprecated. You have to use **BS485NetworkConfig** instead. See **BS_Write485NetworkConfig** for details.

Return Values

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

Compatibility

BioStation

BS_Write485NetworkConfig/BS_Read485NetworkConfig

Specifies the RS485 mode of BioStation. For the general concept of RS485 communication, refer to **BS_OpenSerial485**.

BS_RET_CODE BS_Write485NetworkConfig(int handle, BS485NetworkConfig* config)
BS_RET_CODE BS_Read485NetworkConfig(int handle, BS485NetworkConfig* config)

Parameters

handle

Pointer to the communication channel.

config

BS485NetworkConfig is defined as follows;

```
typedef struct {
    unsigned short deviceType;
    unsigned short useIO;
    char activeSIO[MAX_NUM_OF_SIO];
    BS485SlaveInfo slaveInfo[MAX_NUM_OF_SLAVE];
    int reserved[18];
} BS485NetworkConfig;

typedef struct{
    unsigned slaveID;
    int slaveType;
} BS485SlaveInfo;
```

The key fields and their available options are as follows;

Fields	Options
deviceType	TYPE_DISABLE TYPE_CONN_PC – 485 port is used for PC connection. TYPE_HOST – The device plays the role of the host. TYPE_SLAVE – The device is connected to the host device.
useIO	It should be true.
activeSIO slaveInfo	These fields are filled by the device when **BS_Search485Slaves** is done successfully. You

should not change these fields manually.

Return Values

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

Compatibility

BioStation

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;
    int reserved[7];
} BSUSBConfig;
```

Return Values

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

Compatibility

BioStation

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.

Compatibility

BioStation

BS_WriteWiegandConfig / BS_ReadWiegandConfig

Configures Wiegand format. Up to 64 bit Wiegand 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.

Compatibility

BioStation

BS_WriteZoneConfigEx / BS_ReadZoneConfigEx

Zones are used to group a number of devices to have a specific function. A zone consists of a master device, which plays a role similar to that of a legacy controller, and the other member devices. Any device - BioStation, BioEntry Plus, or BioLite Net - can be a master device. The maximum number of devices in a group is 64.

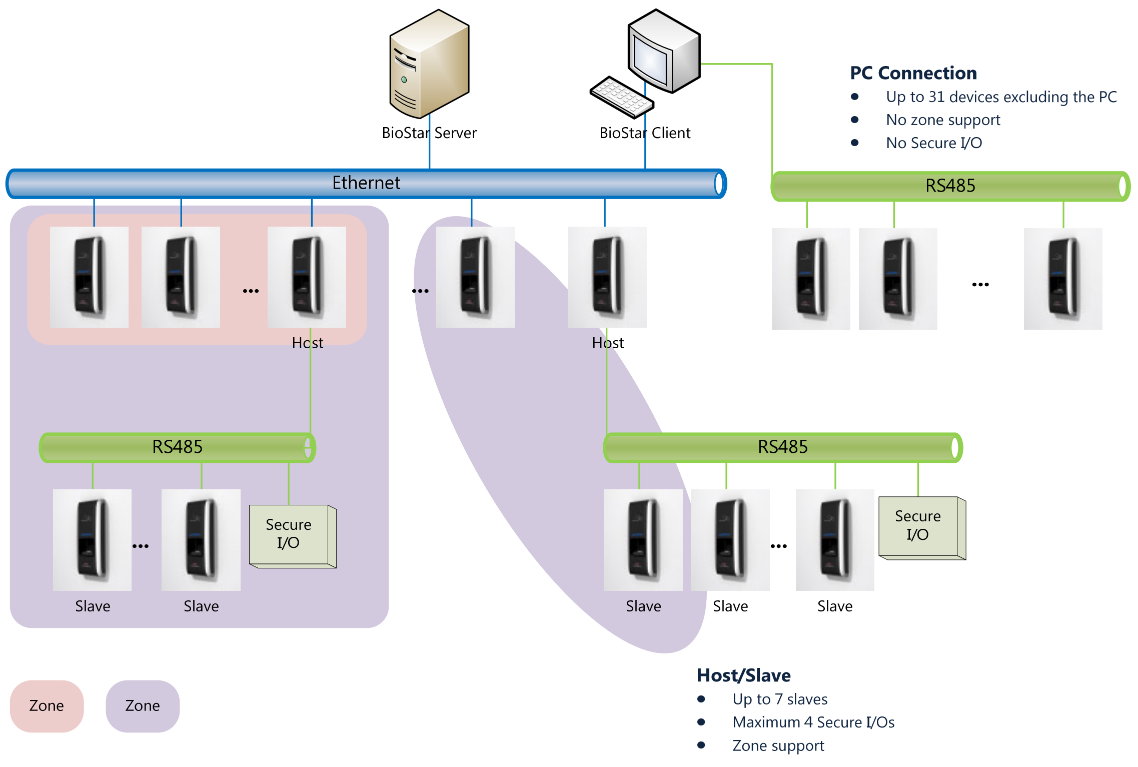


Figure 2 Zone Configuration of BioStar

In total, the BioStar system supports five types of zones:

• **Access zone** - Use this zone to synchronize user or log information. If you select the user synchronization option, user data enrolled at the devices will be automatically propagated to other connected devices. If you select the log synchronization option, all log records will be written to the master device, so that you can check log records of member devices.

• **Anti-passback zone** - Use this zone to prevent a user from passing his or her card back to another person or using his or her fingerprint to allow someone else to gain entry. The zone supports two types of anti-passback restrictions: soft and hard.

When a user violates the anti-passback protocol, the soft restriction will record the action in the user's log. The hard restriction will deny access and record the event in the log when the antipassback protocol is violated.

• **Entrance limit zone** - Use this zone to restrict the number of times a user can enter an area. The entrance limit can be tied to a timezone, so that a user is restricted to a maximum number of entries during a specified time span. You can also set time limits for reentry to enforce a timed anti-passback restriction.

• **Alarm zone** - Use this zone to group inputs from multiple devices into a single alarm zone. Devices in the alarm zone can be simultaneously armed or disarmed via an arm or disarm card or a key.

• **Fire alarm zone** - Use this zone to control how doors will respond during a fire. External inputs can be fed into the BioStar system to automatically trigger door releases or perform other actions.

BS_RET_CODE BS_WriteZoneConfigEx(int handle, BSZoneConfigEx* config)

BS_RET_CODE BS_ReadZoneConfigEx(int handle, BSZoneConfigEx* config)

Parameters

handle

Handle of the communication channel.

config

BSZoneConfigEx is defined as follows;

```
typedef struct {
    //Common
    int numOfMember; //includes master node itself...
    unsigned memberId[BS_MAX_NODE_PER_ZONE_EX];
    unsigned memberIpAddr[BS_MAX_NODE_PER_ZONE_EX];
    int memberStatus[BS_MAX_NODE_PER_ZONE_EX];
    int memberInfo[BS_MAX_NODE_PER_ZONE_EX];
    int reserved1[8];

    //Alarm zone
    int zoneStatus;
    int alarmStatus;  // 0 : disabled, 1 : enabled

    int reserved2[3];
} BSZoneMasterEx;
```

```
typedef struct {
    //Common
    unsigned masterIpAddr;
    int reserved1[2];

    //APB zone
    int authMode;
    int ioMode;

    //Alarm zone
    int armType;
    int useSound;
    int armKey;
    int disarmKey;

    // Card for arm/disarm
    int cardID[8];
    unsigned char customID[8];

    int reserved2[3];
} BSZoneMemberEx;

typedef struct {
     int fallbackMode;
     bool synchTime;
     bool synchUser;
     bool synchLog;
     int reserved[4];
} BSAccessZoneProperty;

typedef struct {
     int apbType;
     int apbResetInterval; // 0 for no limit
     int bypassGroupId;
} BSAPBZonePropertyEx;

typedef struct {
     int minEntryInterval; // 0 for no limit
     int numOfEntranceLimit; // MAX 4
     int maxEntry[BE_MAX_ENTRANCE_LIMIT_PER_DAY]; // 0 (no limit) ~ 16
     unsigned entryLimitInterval[BE_MAX_ENTRANCE_LIMIT_PER_DAY];
     int bypassGroupId;
} BSEntranceLimitationZonePropertyEx;

typedef struct {
     int accessGroupId;
```

```
        int armDelay;
        int disarmDelay;
        int reserved[8];
   } BSAlarmZoneProperty;

   typedef struct {
        int reserved[8];
   } BSFireAlarmZoneProperty;

   typedef struct {
      union
      {
         BSAccessZoneProperty accessZoneProperty;
         BSAPBZonePropertyEx apbZoneProperty;
         BSEntranceLimitationZonePropertyEx entLimitZoneProperty;
         BSAlarmZoneProperty alarmZoneProperty;
         BSFireAlarmZoneProperty fireAlarmZoneProperty;
      };
   } BSZonePropertyEx;

   typedef struct {
      unsigned zoneId;   //0 ~ 255
      int zoneType;
      int nodeType;
      BSZoneMasterEx master;
      BSZoneMemberEx member;
      BSZonePropertyEx ZoneProperty;
   } BSZoneEx;

   typedef struct {
      int numOfZones; //0 ~ BS_MAX_ZONE_PER_NODE
      BSZoneEx zone[BS_MAX_ZONE_PER_NODE];
   } BSZoneConfigEx;
```

The key fields and their available options are as follows;

BSZoneMasterEx

Fields	**Options**
numOfMember	The number of devices in the zone including the master device.
memberId	The IDs of the device in the zone.
memberIpAddr	The IP addresses of the devices in the zone.
memberStatus	NORMAL DISCONNECTED

memberInfo	In an anti-passback zone, it is one of the followings; IN_READER OUT_READER In an alarm zone, it is an mask consists of the following values; DUMMY_READER ARM_READER DISARM_READER
zoneStatus	Specifies the status of an alarm zone; ARMED DISARMED
alarmStatus	Specifies whether an alarm is active in an alarm zone.

BSZoneMemberEx

Fields	**Options**
masterIpAddr	The IP address of the master device.
authMode	It should be BS_AUTH_DEFERRED.
ioMode	Reserved for future use.
armType	Specifies arming method in an alarm zone. ARM_BY_KEYPAD ARM_BY_CARD
useSound	Specifies whether the device should emit alarm sounds when a violation is detected in an armed zone.
armKey	If armType is ARM_BY_KEYPAD, specifies the key code for arming. One of the following four function keys can be used. BS_KEY_F1 BS_KEY_F2 BS_KEY_F3 BS_KEY_F4
disarmKey	If armType is ARM_BY_KEYPAD, specifies the key code for disarming.
cardID	If armType is ARM_BY_CARD, specifies the 4 byte cardID for arming or disarming.

customID	If armType is ARM_BY_CARD, specifies the 1 byte custom cardID for arming or disarming.

BSAccessZoneProperty

Fields	**Options**
fallbackMode	Reserved for future use.
synchTime	If true, the system clock of member devices will be synchronized with that of the master.
synchUser	If true, enrolling/deleting users will be propagated to all the other devices.
synchLog	If true, all the log records of member devices will be stored to the master, too.

BSAPBZonePropertyEx

Fields	**Options**
apbType	BS_APB_NONE BS_APB_SOFT BS_APB_HARD
apbResetInterval	If it is not 0, anti-passback violation will be reset after this interval. For example, if it is 120, users are able to enter a door twice after 120 minutes
bypassGroupId	The ID of an access group, the members of which can bypass the anti-passback zone.

BSEntranceLimitationZonePropertyEx

Fields	**Options**
minEntryInterval	If it is not 0, re-entrance to the zone will be prohibited until this interval elapses. For example, if user A entered the zone at 10:00 with **minEntryInterval** 60, he'll not able to access the zone again until 11:00.

numOfEntranceLimit	The number of entries for specified time intervals can be limited by **maxEntry** and **entryLimitSchedule**. For example, if users are allowed to access a zone 3 times for AM10:00 ~AM11:30 and 1 time for PM2:20~PM6:00, these variables should be set as follows;
	`numOfEntranceLimit = 2;`
	`maxEntry[0] = 3;`
	`entryLimitInterval[0] = (10 * 60) | ((11 * 60 + 30) << 16);`
	`maxEntry[1] = 1;`
	`entryLimitInterval[1] = (14 * 60 + 20) | ((18 * 60) << 16);`
	If **numOfEntranceLimit** is 0, no limitation is applied. If **numOfEntranceLimit** is larger than 0, users can access only during the specified time intervals.

maxEntry	The maximum number of entries for the specified time interval.
entryLimitInterval	The time interval to which the entrance limitation is applied. It is defined as follows; (start time in minute) | (end time in minute << 16).
bypassGroupId	The ID of an access group, the members of which can bypass the entrance limitation zone.

BSAlarmZoneProperty

Fields	**Options**
accessGroupId	The ID of an access group, the members of which can arm or disarm the zone.
armDelay	Specifies the length of time to delay before arming the zone.
disarmDelay	Specifies the length of time to delay before disarming the zone.

BSZoneEx

Fields	**Options**
zoneId	The ID of the zone. It should be between 0 and 255.
zoneType	BS_ZONE_TYPE_ACCESS BS_ZONE_TYPE_APB BS_ZONE_TYPE_ENTRANCE_LIMIT BS_ZONE_TYPE_ALARM BS_ZONE_TYPE_FIRE_ALARM.
nodeType	BS_STANDALONE_NODE BS_MASTER_NODE BS_MEMBER_NODE
master	The information of the master device.
member	The information of the member device.
ZoneProperty	The property of the zone.

BSZoneConfigEx

Fields	**Options**
numOfZones	The number of zones, of which this device is a member.
zone	The zone data structure.

Return Values

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

Compatibility

BioStation(V1.5 or later)/BioEntry Plus(V1.2 or later)/BioLite Net

BS_WriteDoorConfig / BS_ReadDoorConfig

Up to two device can be attached to a door. You can specify which I/O ports are used for the relay, RTE, and door sensor. You can also configure the actions for forced open and held open alarms.

BS_RET_CODE BS_WriteDoorConfig(int handle, BSDoorConfig* config)
BS_RET_CODE BS_ReadDoorConfig(int handle, BSDoorConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSDoorConfig is defined as follows;

```
struct BSDoor {
     int relay;
     int useRTE;
     int useDoorSensor;
     int openEvent; // only for BST
     int openTime;
     int heldOpenTime;
     int forcedOpenSchedule;
     int forcedCloseSchedule;
     int RTEType;
     int sensorType;
     short reader[2];
     unsigned char useRTEEx;
     unsigned char useSoundForcedOpen;
     unsigned char useSoundHeldOpen;
     unsigned char reserved1[1];
     int RTE;
     unsigned char useDoorSensorEx;
     unsigned char alarmStatus;
     unsigned char reserved2[2];
     int doorSensor;
     int relayDeviceId;
};

struct BSDoorConfig {
     BSDoor door[MAX_DOOR];
     int apbType;
```

```
        int apbResetTime;
        int doorMode;
    };
```

The key fields and their available options are as follows;

BSDoor

Fields	**Options**
relay	RELAY_DISABLED PRIMARY_RELAY: its own relay SECONDARY_RELAY: the relay of another device, whose ID is **relayDeviceId**. Both devices should be connected by RS485. SECUREIO0_RELAY0 SECUREIO0_RELAY1 SECUREIO1_RELAY0 SECUREIO1_RELAY1 SECUREIO2_RELAY0 SECUREIO2_RELAY1 SECUREIO3_RELAY0 SECUREIO3_RELAY1
useRTE	Not used. See useRTEEx.
useDoorSensor	Not used. See useDoorSensorEx.
openEvent	Specifies when the relay is activated in BioStation. This field is ignored by BioEntry Plus and BioLite Net. BS_RELAY_EVENT_ALL - relay is on whenever authentication succeeds. BS_RELAY_EVENT_AUTH_TNA – relay is activated when the useRelay field of the TNA event is true, or no TNA event is selected. BS_RELAY_EVENT_NONE – relay is disabled. BS_RELAY_EVENT_AUTH - relay is activated only when no TNA event is selected. BS_RELAY_EVENT_TNA - relay is activated only when the useRelay field of the TNA event is true.
openTime	Specifies the duration in seconds for which

	the relay is on. After this duration, the relay will be turned off.
heldOpenTime	If a door is held open beyond **heldOpenTime**, BE_EVENT_DOOR_HELD_OPEN_ALARM event will be generated. To detect this and BE_EVENT_DOOR_FORCED_OPEN_ALARM events, a door sensor should be configured first.
forcedOpenSchedule	Specifies the schedule in which the relay should be held on.
forcedCloseSchedule	Specifies the schedule in which the relay should be held off.
RTEType	The switch type of the RTE input. NORMALLY_OPEN NORMALLY_CLOSED
sensorType	The switch type of the door sensor. NORMALLY_OPEN NORMALLY_CLOSED
reader	Not used.
useRTEEx	Specifies whether an input is used for RTE. If it is true, the **RTE** field denotes the input port for RTE.
useSoundForcedOpen	If true, emits an alarm sound when forced open alarm occurs.
useSoundHeldOpen	If true, emits an alarm sound when held open alarm occurs.
RTE	Specifies an input port for RTE. If **useRTEEx** is not true, this field will be ignored. HOST_INPUT0 HOST_INPUT1 SECUREIO0_INPUT0 SECUREIO0_INPUT1 SECUREIO0_INPUT2 SECUREIO0_INPUT3 SECUREIO1_INPUT0

	SECUREIO1_INPUT1
	SECUREIO1_INPUT2
	SECUREIO1_INPUT3
	SECUREIO2_INPUT0
	SECUREIO2_INPUT1
	SECUREIO2_INPUT2
	SECUREIO2_INPUT3
	SECUREIO3_INPUT0
	SECUREIO3_INPUT1
	SECUREIO3_INPUT2
	SECUREIO3_INPUT3
useDoorSensorEx	Specifies whether an input is used for door sensor. If it is true, the **doorSensor** field denotes the input port for door sensor.
alarmStatus	Specifies whether forced open or held open alarm is active.
doorSensor	Specifies an input port for door sensor. If **useDoorSensorEx** is not true, this field will be ignored. For available options, see **RTE** field above.
relayDeviceId	The ID of another device, whose relay will be used for the door. Both devices should be connected by RS485. And, the **relay** field should be set to SECONDARY_RELAY.

BSDoorConfig

Fields	**Options**
doorMode	Not used.
door	Only door[0] will be used.
apbType	Not used.
apbResetTime	Not used.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_WriteInputConfig/BS_ReadInputConfig

A BioStation, BioEntry Plus, or BioLite Net can control up to 4 Secure I/O devices through RS485 connection. A Secure I/O device has 4 SW inputs. A BioStation, BioEntry Plus, or BioLite Net has 2 SW inputs.

BS_RET_CODE BS_WriteInputConfig(int handle, BSInputConfig* config)
BS_RET_CODE BS_ReadInputConfig(int handle, BSInputConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSInputConfig is defined as follows;

```
struct BSInputFunction {
     int functionType;
     short minimumDuration;
     short switchType;
     int timeSchedule;
     int reserved[5];
};

struct BSInputConfig {
     // host inputs
     BSInputFunction hostTamper;
     BSInputFunction hostInput[NUM_OF_HOST_INPUT];
     // secure I/O
     BSInputFunction secureIO[NUM_OF_SECURE_IO][NUM_OF_SECURE_INPUT];
     // slave
     BSInputFunction slaveTamper;
     BSInputFunction slaveInput[NUM_OF_SLAVE_INPUT];
     int reserved[32];
};
```

The key fields and their available options are as follows;

BSInputFunction

Fields	**Options**
functionType	If an input port is activated, the assigned

	function will be executed. DISALBED GENERIC_OPEN: BE_EVENT_XXXX_INPUT(0xA0 ~ 0xBF) log record is written and assigned output events are generated if any. EMERGENCY_OPEN: open all the doors defined in **BSDoorConfig**. ALL_ALARM_OFF: turn off all the non-door relays under the control of this device. RESET_READER: reset the device. LOCK_READER: lock the device. ALARM_ZONE_INPUT: the port is used for alarm zone. FIRE_ALARM_ZONE_INPUT: the port is used for fire alarm zone.
minimumDuration	To filter out noise, input signals with shorter duration than this minimum will be ignored. The unit is milliseconds.
switchType	The switch type of this input. NORMALLY_OPEN NORMALLY_CLOSED
timeSchedule	Specifies the time schedule in which this input is enabled.

BSInputConfig

Fields	**Options**
internalTamper	Specifies the function which will be executed when the tamper switch of the host device is turned on.
internal	Specifies the input functions of the host device.
secureIO	Specifies the input functions of Secure I/O devices connected to the host.
slaveTamper	Not used.
slave	Not used.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

Example

```
// (1) Lock the device when the internal tamper is on
// (2) Open all doors when the input port 1 of Secure I/O 0 is activated
BSInputConfig inputConfig;
memset( &inputConfig, 0, sizeof( BSInputConfig ) );

inputConfig.internalTamper.functionType = BSInputFunction::LOCK_READER;
inputConfig.internalTamper.minimumDuration = 100; // 100 ms
inputConfig.internalTamper.switchType = BSDoor::NORMALLY_OPEN;
inputConfig.internalTamper.timeSchedule =
BSTimeScheduleEx::ALL_TIME_SCHEDULE; // enabled always

inputConfig.secureIO[0][1].functionType = BSInputFunction::EMERGENCY_OPEN;
inputConfig.secureIO[0][1].minimumDuration = 1000; // 1000 ms
inputConfig.secureIO[0][1].switchType = BSDoor::NORMALLY_OPEN;
inputConfig.secureIO[0][1].timeSchedule =
BSTimeScheduleEx::ALL_TIME_SCHEDULE; // enabled always
```

BS_WriteOutputConfig / BS_ReadOutputConfig

A BioStation, BioEntry Plus, or BioLite Net can control up to 4 Secure I/O devices through RS485 connection. A Secure I/O device has 2 relay outputs. A BioStation, BioEntry Plus, or BioLite Net has 1 relay output. Users can assign multiple output events to each replay port. If one of the given events occurs, the configured signal will be output to the relay port.

BS_RET_CODE BS_WriteOutputConfig(int handle, BSOutputConfig* config)
BS_RET_CODE BS_ReadOutputConfig(int handle, BSOutputConfig* config)

Parameters

handle

Handle of the communication channel.

config

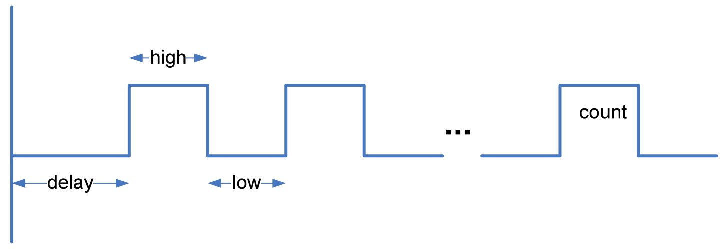
BSOutputConfig is defined as follows;

```
struct BSOutputEvent {
     unsigned event; // (8 bit input device ID << 16) | 16 bit event ID
     unsigned char outputDeviceID;
     unsigned char outputRelayID;
     unsigned char relayOn;
     unsigned char reserved1;
     unsigned short delay;
     unsigned short high;
     unsigned short low;
     unsigned short count;
     int priority; // 1(highest) ~ 99(lowest)
     int reserved2[3];
};

struct BSEMOutputEvent {
     unsigned short inputType;
     unsigned short outputRelayID;
     unsigned short inputDuration;
     unsigned short high;
     unsigned short low;
     unsigned short count;
     int reserved3[5];
};

struct BSOutputConfig {
     int numOfEvent;
     BSOutputEvent outputEvent[MAX_OUTPUT];
     BSEMOutputEvent
emergencyEvent[BSInputConfig::NUM_OF_SECURE_IO][BSInputConfig::NUM_OF_SECUR
E_INPUT];
     int reserved[31];
};
```

The key fields and their available options are as follows;

BSOutputEvent

Fields	**Options**
event	The event which will trigger the output signal. It consists of an event ID and a device ID in which the event occurs. The available events are as follows; AUTH_SUCCESS AUTH_FAIL AUTH_DURESS ACCESS_NOT_GRANTED ADMIN_AUTH_SUCCESS TAMPER_ON DOOR_OPEN DOOR_CLOSED INPUT0_ON INPUT1_ON INPUT2_ON INPUT3_ON ALARM_ZONE_EVENT FIRE_ALARM_ZONE_EVENT APB_ZONE_EVENT ENTLIMIT_ZONE_EVENT DOOR_HELD_OPEN_EVENT DOOR_FORCED_OPEN_EVENT The available device IDs are as follows; BS_DEVICE_PRIMARY

	BS_DEVICE_SECUREIO0 BS_DEVICE_SECUREIO1 BS_DEVICE_SECUREIO2 BS_DEVICE_SECUREIO3 BS_DEVICE_ALL For example, when the input SW 0 of Secure IO 0 is activated, INPUT0_ON | (BS_DEVICE_SECUREIO0 << 16)
outputDeviceID	Specifies the device which will generate the output signal. BS_DEVICE_PRIMARY BS_DEVICE_SECUREIO0 BS_DEVICE_SECUREIO1 BS_DEVICE_SECUREIO2 BS_DEVICE_SECUREIO3
otuputRelayID	Specifies the relay port from which the output signal will be generated. BS_PORT_RELAY0 BS_PORT_RELAY1
relayOn	If true, turn on the relay. If false, turn off the relay.
delay high low count	These four fields define the waveform of output signal. If **relayOn** is false, these fields are ignored.  The unit is milliseconds. If count is 0, the signal will be repeated indefinitely.
priority	The priority of the event between 1(highest) and 99(lowest). When a relay is generating the signal of previous event, only events with same or higher priority can replace it.

BSEMOutputEvent

In normal condition, the host device handles all inputs of Secure I/O devices. However, when RS485 connection is disconnected, Secure I/O devices should process their own inputs by themselves. This configuration defines how to handle Secure I/O inputs in this case.

Fields	**Options**
inputType	The switch type of this input. NORMALLY_OPEN NORMALLY_CLOSED
outputRelayID	Specifies the relay port from which the output signal will be generated. BS_PORT_RELAY0 BS_PORT_RELAY1
inputDuration	To filter out noise, input signals with shorter duration than this minimum will be ignored. The unit is milliseconds.
high low count	These three fields define the waveform of output signal.

BSOutputConfig

Fields	**Options**
numOfEvent	The number of output events defined in this device.
outputEvent	The array of **BSOutputEvent**.
emergencyEvent	**BSEMOutputEvent**.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

Example

```
// (1) Generate alarm signal to the relay 0 of Secure I/O 0 when
//      anti-passback is violated.
// (2) Turn off the above alarm when the input 0 of Secure I/O 0 is
//     activated.
BSOutputConfig outputConfig;
memset( &outputConfig, 0, sizeof( BSOutputConfig ) );

outputConfig.numOfEvent = 2;

outputConfig.outputEvent[0].event = BSOutputEvent::APB_ZONE_EVENT |
(BS_DEVICE_ALL << 16);
outputConfig.outputEvent[0].outputDeviceID = BS_DEVICE_SECUREIO0;
outputConfig.outputEvent[0].outputRelayID = BS_PORT_RELAY0;
outputConfig.outputEvent[0].relayOn = true;
outputConfig.outputEvent[0].delay = 0;
outputConfig.outputEvent[0].high = 100; // 100 ms
outputConfig.outputEvent[0].low = 100; // 100 ms
outputConfig.outputEvent[0].count = 0; // indefinite
outputConfig.outputEvent[0].priority = 1;

outputConfig.outputEvent[1].event = BSOutputEvent::INPUT0_ON |
(BS_DEVICE_SECUREIO0 << 16);
outputConfig.outputEvent[0].outputDeviceID = BS_DEVICE_SECUREIO0;
outputConfig.outputEvent[0].outputRelayID = BS_PORT_RELAY0;
outputConfig.outputEvent[0].relayOn = false;
outputConfig.outputEvent[0].priority = 1;
```

BS_WriteEntranceLimitConfig / BS_ReadEntranceLimitConfig

You can apply entrance limitation rules to each device.

BS_RET_CODE BS_WriteEntranceLimitConfig(int handle, BSEntranceLimit* config)
BS_RET_CODE BS_ReadEntranceLimitConfig(int handle, BSEntranceLimit* config)

Parameters

handle

Handle of the communication channel.

config

BSEntranceLimit is defined as follows;

```
typedef struct {
    int minEntryInterval; // 0 for no limit
    int numOfEntranceLimit; // MAX 4
    int maxEntry[4]; // 0 (no limit) ~ 16
    unsigned entryLimitInterval[4];
    int defaultAccessGroup;
    int bypassGroupId;
    int entranceLimitReserved[6];
} BSEntranceLimit;
```

The key fields and their available options are as follows;

BSOutputEvent

Fields	**Options**
minEntryInterval	See the descriptions of **BSEntranceLimitationZonePropertyEx**.
numOfEntranceLimit	
maxEntry	
entryLimitInterval	
bypassGroupId	
defaultAccessGroup	The default access group of users. It is either BSAccessGroupEx::NO_ACCESS_GROUP or BSAccessGroupEx::FULL_ACCESS_GROUP. This access group is applied to the following cases. (1) When a user has no access group. For example, if **defaultAccessGroup** is

NO_ACCESS_GROUP, users without access groups are not allowed to enter.
(2) When a user has invalid access group.
(3) When enrolling users by command card

Return Values

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

Compatibility

BioStation

BS_WriteConfig/BS_ReadConfig for BioEntry Plus

You can write/read the configuration of a BioEntry Plus or BioLite Net using **BS_WriteConfig/BS_ReadConfig**. The structures related to BioEntry Plus are described in this section. Those related to BioLite Net will be explained in the next section.

BS_RET_CODE BS_WriteConfig(int handle, int configType, int size, void* data)
BS_RET_CODE BS_ReadConfig(int handle, int configType, int* size, void* data)

Parameters

handle

Handle of the communication channel.

configType

The configuration types and their corresponding data structures are as follows.

Device	Configuration Type	Structure
BioEntry Plus	BEPLUS_CONFIG	BEConfigData
	BEPLUS_CONFIG_SYS_INFO	BESysInfoData
BioLite Net	BIOLITE_CONFIG	BEConfigDataBLN
	BIOLITE_CONFIG_SYS_INFO	BESysInfoDataBLN

Please note that BEPLUS_CONFIG_SYS_INFO and BIOLITE_CONFIG_SYS_INFO are read-only. You cannot change the system information using BS_WriteConfig.

size

Size of the configuration data.

data

Pointer to the configuration data. **BEConfigData** and **BESysInfoData** are defined as follows;

```
struct BEOutputPattern {
    int repeat; // 0: indefinite, -1: don't user
    int arg[MAX_ARG]; // color for LED, frequency for Buzzer, -1 for last
    short high[MAX_ARG]; // msec
    short low[MAX_ARG]; // msec
};
```

```
struct BELEDBuzzerConfig {
    int reserved[4];
    BEOutputPattern ledPattern[MAX_SIGNAL];
    BEOutputPattern buzzerPattern[MAX_SIGNAL];
    unsigned short signalReserved[MAX_SIGNAL];
};

typedef struct {
    unsigned cardID;
    unsigned char customID;
    unsigned char commandType;
    unsigned char needAdminFinger;
    unsigned char reserved[5];
} BECommandCard;


typedef struct {
    // header
    unsigned magicNo;
    int version;
    unsigned timestamp;
    unsigned checksum;
    int headerReserved[4];
    // operation mode
    int opMode[4];
    int opModeSchedule[4];
    int opModePerUser; /* PROHIBITED, ALLOWED */
    int opReserved[7];
    bool useDHCP;
    unsigned ipAddr;
    unsigned gateway;
    unsigned subnetMask;
    unsigned serverIpAddr;
    int port;
    bool useServer;
    bool synchTime;
    int ipReserved[8];
    // fingerprint
    int securityLevel;
    int fastMode;
    int fingerReserved1
    int timeout; // 1 ~ 20 sec
    int matchTimeout; // Infinite(0) ~ 10 sec
    int templateType;
    int fakeDetection;
    bool useServerMatching;
    int fingerReserved[8];
```

```
        // I/O
        BSInputConfig inputConfig;
        BSOutputConfig outputConfig;
        BSDoorConfig doorConfig;
        int ioReserved[3];
        //extended serial
        unsigned hostID;
        unsigned slaveIDEx[MAX_485_DEVICE];
       unsigned slaveType;   // 0 : BST, 1 : BEPL
        // serial
        int serialMode;
        int serialBaudrate;
        unsigned char serialReserved1;
        unsigned char secureIO; // 0x01 - Secure I/O 0, 0x02, 0x04, 0x08
        unsigned char serialReserved2[6];
        unsigned slaveID; // 0 for no slave
        int reserved1[17];
        // entrance limit
        int minEntryInterval; // 0 for no limit
        int numOfEntranceLimit; // MAX 4
        int maxEntry[4]; // 0 (no limit) ~ 16
        unsigned entryLimitInterval[4];
        int bypassGroupId;
        int entranceLimitReserved[7];
        // command card
        int numOfCommandCard;
        BECommandCard commandCard[MAX_COMMAND_CARD];
        int commandCardReserved[3];
        // tna
        int tnaMode;
        int autoInSchedule;
        int autoOutSchedule;
        int tnaReserved[5];
        // user
        int defaultAG;
        int userReserved[7];
        int reserved2[22];
        // wiegand
        bool useWiegandOutput;
        int wiegandReserved[6];
        int wiegandIdType;
        BSWiegandConfig wiegandConfig;
        // LED/Buzzer
        BELEDBuzzerConfig ledBuzzerConfig;
        int padding[215];
    } BEConfigData;
```

```
typedef struct {
    unsigned magicNo;
    int version;
    unsigned timestamp;
    unsigned checksum;
    int headerReserved[4];
    unsigned ID;
    unsigned char macAddr[8];
    char boardVer[16];
    char firmwareVer[16];
    char productName[32];
    int reserved[32];
} BESysInfoData
```

The key fields and their available options are as follows;

BEOutputPattern

You can define the output patterns, which will be used in **BELEDBuzzerConfig**.

Fields	**Options**
repeat	The number of output signal to be emitted. 0 – indefinite -1 – not used
arg	For the LED, it specifies one of the following colors; RED, YELLOW, GREEN, CYAN, BLUE, MAGENTA, WHITE. For the buzzer, it specifies one of the following frequencies; HIGH_FREQ, MIDDLE_FREQ, LOW_FREQ.
high	The duration of high signal in milliseconds.
low	The duration of low signal in milliseconds.

BELEDBuzzerConfig

You can define the output patterns of LED or buzzer for specific events. Refer to the enumerations of **BELEDBuzzerConfig** in BS_BEPlus.h for the pre-defined event types. For example, the default patterns for normal status and authenticaion fail are defined as follows;

```
// Normal
// LED: Indefinitely blinking Blue(2sec)/Light Blue(2sec)
```

```
    // Buzzer: None
    ledPattern[BELEDBuzzerConfig::STATUS_NORMAL].repeat = 0;
    ledPattern[BELEDBuzzerConfig::STATUS_NORMAL].arg[0] =
BEOutputPattern::BLUE;
    ledPattern[BELEDBuzzerConfig::STATUS_NORMAL].high[0] = 2000;
    ledPattern[BELEDBuzzerConfig::STATUS_NORMAL].arg[1] =
BEOutputPattern::CYAN;
    ledPattern[BELEDBuzzerConfig::STATUS_NORMAL].high[1] = 2000;
    ledPattern[BELEDBuzzerConfig::STATUS_NORMAL].arg[2] = -1;
    buzzerPattern[BELEDBuzzerConfig::STATUS_NORMAL].repeat = -1;
    // Authentication Fail
    // LED: Red for 1 second
    // Buzzer: Three high-tone beeps
    ledPattern[BELEDBuzzerConfig::AUTH_FAIL].repeat = 1;
    ledPattern[BELEDBuzzerConfig::AUTH_FAIL].arg[0] =
BEOutputPattern::RED;
    ledPattern[BELEDBuzzerConfig::AUTH_FAIL].high[0] = 1000;
    ledPattern[BELEDBuzzerConfig::AUTH_FAIL].arg[1] = -1;
    buzzerPattern[BELEDBuzzerConfig::AUTH_FAIL].repeat = 1;
    buzzerPattern[BELEDBuzzerConfig::AUTH_FAIL].arg[0] =
BEOutputPattern::HIGH_FREQ;
    buzzerPattern[BELEDBuzzerConfig::AUTH_FAIL].high[0] = 100;
    buzzerPattern[BELEDBuzzerConfig::AUTH_FAIL].low[0] = 20;
    buzzerPattern[BELEDBuzzerConfig::AUTH_FAIL].arg[1] =
BEOutputPattern::HIGH_FREQ;
    buzzerPattern[BELEDBuzzerConfig::AUTH_FAIL].high[1] = 100;
    buzzerPattern[BELEDBuzzerConfig::AUTH_FAIL].low[1] = 20;
    buzzerPattern[BELEDBuzzerConfig::AUTH_FAIL].arg[2] =
BEOutputPattern::HIGH_FREQ;
    buzzerPattern[BELEDBuzzerConfig::AUTH_FAIL].high[2] = 100;
```

Fields	**Options**
ledPattern	The LED output patterns for pre-defined events.
buzzerPattern	The buzzer output patterns for pre-defined events.

BECommandCard

BioEntry Plus supports command cards with which you can enroll/delete users at devices directly.

Fields	**Options**
cardID	4 byte card ID. The RF card ID is comprised of 4 byte card ID and 1 byte custom ID.
customID	1 byte custom ID of the card.
commandType	There are three types of command cards.

	● ENROLL_CARD ● DELETE_CARD ● DELETE_ALL_CARD
needAdminFinger	If this option is true, an administrator should be authenticated first before enrolling/deleting users.

BEConfigData

Fields	**Options**
magicNo version timestamp checksum	These 4 fields are for internal-use only. Users should not update these values.
<u>Operation Mode</u>	
opMode	Available authentication modes are as follows; CARD_OR_FINGER: Both 1:1(card + fingerprint) and 1:N(fingerprint) authentications are allowed. CARD_N_FINGER: Only 1:1(card + fingerprint) authentication is allowed. CARD_ONLY: If an enrolled card is read, access is allowed without fingerprint authentication. FINGER_ONLY: Only 1:N(fingerprint) authentication is allowed. Bypass cards are also denied in this mode. The default mode is CARD_OR_FINGER.
opModeSchedule	You can mix up to 4 authentication modes based on time schedules. If more than one authentication modes are used, the time schedules of them should not be overlapped.
opDualMode	If it is true, two users should be authenticated before a door is opened.
opModePerUser	If true, the **opMode** field of **BEUserHdr** will be applied to user. Otherwise, the **opMode** field of **BEConfigData** will be applied.
<u>Ethernet</u>	
useDHCP	Specifies if DHCP is used.
ipAddr	IP address of the device.

gateway	Gateway address.
subnetMask	Subnet mask.
port	Port number of the TCP connection.
useServer	If true, connect to the server with **serverIPAddr** and **port**. If false, open the TCP port and wait for incoming connections.
serverIPAddr	IP address of the server.
synchTime	If true, synchronize system clock with server when connecting to it.

Fingerprint

securityLevel	Sets the security level. AUTOMATIC_NORMAL – FAR(False Acceptance Ratio) is 1/10,000 AUTOMATIC_SECURE – FAR is 1/100,000 AUTOMATIC_MORE_SECURE - FAR is 1/1,000,000
fastMode	**fastMode** can be used to shorten the 1:N matching time with little degradation of authentication performance. If it is set to FAST_MODE_AUTO, the matching speed will be adjusted automatically according to the number of enrolled templates. FAST_MODE_AUTO FAST_MODE_NORMAL FAST_MODE_FAST FAST_MODE_FASTER
timeout	Specifies the timeout for fingerprint input in seconds.
matchTimeout	If 1:N matching is not finished until this period, NOT_FOUND error will be returned. The default value is 3 seconds.
templateType	TEMPLATE_SUPREMA TEMPLATE_SIF – ISO 19794-2
fakeDetection	If true, the device will try to detect fake fingers.
useServerMatching	In server matching mode, user authentication is handled by BioStar server, not each device. To use server matching, the **useServer** of **BEConfigData** should be true.

I/O

inputConfig	See BSWriteInputConfig.
outputConfig	See BSWriteOutputConfig.
doorConfig	See BSWriteDoorConfig.
Serial	
hostID	The ID of the host device. Note that **hostID**, **slaveIDEx**, **slaveType,** and **secureIO** will be set automatically by the device after calling **BS_Search485Slaves**. You should not set these values manually.
slaveIDEx	The IDs of slave devices connected to the RS485 network.
slaveType	The types of slave devices.
serialMode	RS485 connection of a BioEntry Plus can be used as one of the followings; SERIAL_DISABLED: not used. SERIAL_IO_HOST_EX: acts as a host device and controls all the I/O operations of Secure I/O devices and slave devices connected to the same RS485 connection. SERIAL_IO_SLAVE_EX: acts as a slave device. SERIAL_PC: used as a communication channel to host PC.
serialBaudrate	Specifies the baudrate of RS485 connection when serialMode is SERIAL_PC. In other cases, it is ignored.
secureIO	A Secure I/O device has an index between 0 and 3. This flag specifies which Secure I/O devices are connected to the RS485 connection. 0x01: Secure I/O 0 0x02: Secure I/O 1 0x04: Secure I/O 2 0x08: Secure I/O 3 If it is 0x07, it means that Secure I/O 0, 1, and 2 are connected. This field will be set automatically by the device after **BS_Search485Slaves** succeeds.
slaveID	Not used.
Entrance Limitation	
minEntryInterval	Entrance limitation can be applied to a single

numOfEntranceLimit maxEntry entryLimitInterval	device. See **BSEntrnaceLimitationZoneProperty** for details.
bypassGroupId	The ID of a group, the members of which can bypass the restriction of entrance limitation settings.

<u>Command Card</u>

numOfCommandCard	The number of command cards enrolled to the device.
commandCard	See **BECommandCard**.

<u>TNA</u>

tnaMode: The **tnaEvent** field of a log record is determined by **tnaMode** as follows;

tnaMode	tnaEvent
TNA_NONE	0xffff
TNA_FIX_IN	BS_TNA_F1
TNA_FIX_OUT	BS_TNA_F2
TNA_AUTO	If it is in **autoInSchedule**, BS_TNA_F1. If it is in **autoOutSchedule**, BS_TNA_F2. Otherwise, 0xffff.

autoInSchedule	Specifies a schedule in which the **tnaEvent** field of a log record will be set BS_TNA_F1.
autoOutSchedule	Specifies a schedule in which the **tnaEvent** field of a log record will be set BS_TNA_F2.

<u>User</u>

defaultAG: The default access group of users. It is either BSAccessGroupEx::NO_ACCESS_GROUP or BSAccessGroupEx::FULL_ACCESS_GROUP. This access group is applied to the following cases.

(1) When a user has no access group. For example, if **defaultAG** is NO_ACCESS_GROUP, users without access groups are not allowed to enter.

(2) When a user has invalid access group.

	(3) When enrolling users by command card.
Wiegand	
useWiegandOutput	If it is true, Wiegand signal will be output when authentication succeeds.
wiegandIdType	Specifies whether the Wiegand bitstream should be interpreted as a user ID or a cardID. WIEGAND_USER WIEGAND_CARD
wiegandConfig	See **BS_WriteWiegandConfig**.
LED/Buzzer	
ledBuzzerConfig	See **BELEDBuzzerConfig**.

Return Values

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

Compatibility

BioEntry Plus

BS_WriteConfig/BS_ReadConfig for BioLite Net

You can write/read the configuration of a BioEntry Plus or BioLite Net using **BS_WriteConfig/BS_ReadConfig**. The structures related to BioLite Net are described in this section. For those related to BioEntry Plus, refer to the previous section.

BS_RET_CODE BS_WriteConfig(int handle, int configType, int size, void* data)
BS_RET_CODE BS_ReadConfig(int handle, int configType, int* size, void* data)

Parameters

handle

Handle of the communication channel.

configType

The configuration types and their corresponding data structures are as follows.

Device	Configuration Type	Structure
BioEntry Plus	BEPLUS_CONFIG	BEConfigData
	BEPLUS_CONFIG_SYS_INFO	BESysInfoData
BioLite Net	BIOLITE_CONFIG	BEConfigDataBLN
	BIOLITE_CONFIG_SYS_INFO	BESysInfoDataBLN

Please note that BEPLUS_CONFIG_SYS_INFO and BIOLITE_CONFIG_SYS_INFO are read-only. You cannot change the system information using BS_WriteConfig.

size

Size of the configuration data.

data

Pointer to the configuration data. **BEConfigDataBLN** and **BESysInfoDataBLN** are defined as follows;

```
struct BEOutputPatternBLN {
    int repeat; // 0: indefinite, -1: don't user
    int priority; // not used
    int arg[MAX_ARG]; // color for LED, frequency for Buzzer, -1 for last
    short high[MAX_ARG]; // msec
    short low[MAX_ARG]; // msec
};
```

```
struct BELEDBuzzerConfigBLN {
    int reserved[4];
    BEOutputPatternBLN ledPattern[MAX_SIGNAL];
    BEOutputPatternBLN buzzerPattern[MAX_SIGNAL];
    BEOutputPatternBLN lcdLedPattern[MAX_SIGNAL];
    BEOutputPatternBLN keypadLedPattern[MAX_SIGNAL];
    unsigned short signalReserved[MAX_SIGNAL];
};

#define MAX_TNA_FUNCTION_KEY 16
#define MAX_TNA_EVENT_LEN 16

struct BETnaEventConfig {
    unsigned char enabled[MAX_TNA_FUNCTION_KEY];
    unsigned char useRelay[MAX_TNA_FUNCTION_KEY];
    unsigned short key[MAX_TNA_FUNCTION_KEY];  // not used
    char eventStr[MAX_TNA_FUNCTION_KEY][MAX_TNA_EVENT_LEN];
};

struct BETnaEventExConfig {
    int fixedTnaIndex;
    int manualTnaIndex;
    int timeSchedule[MAX_TNA_FUNCTION_KEY];
};

typedef struct {
    // header
    unsigned magicNo;
    int version;
    unsigned timestamp;
    unsigned checksum;
    int headerReserved[4];
    // operation mode
    int opMode[MAX_OPMODE];
    int opModeSchedule[MAX_OPMODE];
    unsigned char opDualMode[MAX_OPMODE]; // DoubleMode[4];
    unsigned char opReserved1[3];
    int opModePerUser; /* PROHIBITED, ALLOWED */
    int identificationMode[MAX_IDENTIFYMODE];
    int identificationModeSchedule[MAX_IDENTIFYMODE];
    int opReserved2[1];
    // ip
    bool useDHCP;
    unsigned ipAddr;
    unsigned gateway;
```

```
        unsigned subnetMask;
        unsigned serverIpAddr;
        int port;
        bool useServer;
        bool synchTime;
        int ipReserved[7];
        // fingerprint
        int imageQuality;
        int securityLevel;
        int fastMode;
        int fingerReserved1;
        int timeout; // 1 ~ 20 sec
        int matchTimeout; // Infinite(0) ~ 10 sec
        int templateType;
        int fakeDetection;
        bool useServerMatching;
        bool useCheckDuplicate;
        int fingerReserved[7];
        // I/O
        BSInputConfig inputConfig;
        BSOutputConfig outputConfig;
        BSDoorConfig doorConfig;
        int ioReserved[3];
        //extended serial
        unsigned hostID;
        unsigned slaveIDEx[MAX_485_DEVICE];
        unsigned slaveType;   // 0 : BST, 1 : BEPL
        // serial
        int serialMode;
        int serialBaudrate;
        unsigned char serialReserved1;
        unsigned char secureIO; // 0x01 - Secure I/O 0, 0x02, 0x04, 0x08
        unsigned char serialReserved2[6];
        unsigned slaveID; // 0 for no slave
        int reserved1[17];
        // entrance limit
        int minEntryInterval; // 0 for no limit
        int numOfEntranceLimit; // MAX 4
        int maxEntry[4]; // 0 (no limit) ~ 16
        unsigned entryLimitInterval[4];
        int bypassGroupId;
        int entranceLimitReserved[7];
        // command card: NOT USED for BioLite Net
        int numOfCommandCard;
        BECommandCard commandCard[MAX_COMMAND_CARD];
        int commandCardReserved[3];
        // tna
        int tnaMode;
        int autoInSchedule; // not used
        int autoOutSchedule; // not used
        int tnaChange;
        int tnaReserved[4];
        // user
        int defaultAG;
        int userReserved[7];
        int reserved2[21];
        int isLocked;
        // wiegand
        bool useWiegandOutput;
        bool useWiegandInput;
        int wiegandReserved[5];
        int wiegandIdType;
        BSWiegandConfig wiegandConfig;
        // LED/Buzzer
        BELEDBuzzerConfigBLN ledBuzzerConfig;
        int reserved3[38];
        int backlightMode;
        int soundMode;
        // Tna Event
        BETnaEventConfig tnaEventConfig;
        BETnaEventExConfig tnaEventExConfig;
        int padding[63];
    } BEConfigDataBLN;

    typedef struct {
        unsigned magicNo;
        int version;
        unsigned timestamp;
        unsigned checksum;
        int headerReserved[4];
        unsigned ID;
        unsigned char macAddr[8];
        char boardVer[16];
        char firmwareVer[16];
        char productName[32];
        int language;
        int reserved[31];
    } BESysInfoDataBLN;
```

The key fields and their available options are as follows;

BEOutputPatternBLN

You can define the output patterns, which will be used in

BELEDBuzzerConfigBLN.

Fields	**Options**
repeat	The number of output signal to be emitted. 0 – indefinite -1 – not used
arg	For the LED, it specifies one of the following colors; RED, YELLOW, GREEN, CYAN, BLUE, MAGENTA, WHITE. For the buzzer, it specifies one of the following frequencies; HIGH_FREQ, MIDDLE_FREQ, LOW_FREQ. For the LCD, it specifies wheter the background LED is turned on or off; OFF, ON For the keypad, it specifies which background of the keys are turned on; NUMERIC, OK_ARROW
high	The duration of high signal in milliseconds.
low	The duration of low signal in milliseconds.

BELEDBuzzerConfigBLN

You can define the output patterns of LED or buzzer for specific events. Refer to the enumerations of **BELEDBuzzerConfigBLN** in BS_BEPlus.h for the pre-defined event types. For example, the default patterns for normal status and authenticaion fail are defined as follows;

```
    // Normal
    // LED: Indefinitely blinking Blue(2sec)/Light Blue(2sec)
    // Buzzer: None
    ledPattern[BELEDBuzzerConfigBLN::STATUS_NORMAL].repeat = 0;
    ledPattern[BELEDBuzzerConfigBLN::STATUS_NORMAL].arg[0] =
BEOutputPatternBLN::BLUE;
    ledPattern[BELEDBuzzerConfigBLN::STATUS_NORMAL].high[0] =
2000;
    ledPattern[BELEDBuzzerConfigBLN::STATUS_NORMAL].arg[1] =
BEOutputPatternBLN::CYAN;
    ledPattern[BELEDBuzzerConfigBLN::STATUS_NORMAL].high[1] =
2000;
    ledPattern[BELEDBuzzerConfigBLN::STATUS_NORMAL].arg[2] = -1;
    buzzerPattern[BELEDBuzzerConfigBLN::STATUS_NORMAL].repeat = -
1;
    // Authentication Fail
```

```
        // LED: Red for 1 second
        // Buzzer: Three high-tone beeps
        ledPattern[BELEDBuzzerConfigBLN::AUTH_FAIL].repeat = 1;
        ledPattern[BELEDBuzzerConfigBLN::AUTH_FAIL].arg[0] =
    BEOutputPatternBLN::RED;
        ledPattern[BELEDBuzzerConfigBLN::AUTH_FAIL].high[0] = 1000;
        ledPattern[BELEDBuzzerConfigBLN::AUTH_FAIL].arg[1] = -1;
        buzzerPattern[BELEDBuzzerConfigBLN::AUTH_FAIL].repeat = 1;
        buzzerPattern[BELEDBuzzerConfigBLN::AUTH_FAIL].arg[0] =
    BEOutputPatternBLN::HIGH_FREQ;
        buzzerPattern[BELEDBuzzerConfigBLN::AUTH_FAIL].high[0] = 100;
        buzzerPattern[BELEDBuzzerConfigBLN::AUTH_FAIL].low[0] = 20;
        buzzerPattern[BELEDBuzzerConfigBLN::AUTH_FAIL].arg[1] =
    BEOutputPatternBLN::HIGH_FREQ;
        buzzerPattern[BELEDBuzzerConfigBLN::AUTH_FAIL].high[1] = 100;
        buzzerPattern[BELEDBuzzerConfigBLN::AUTH_FAIL].low[1] = 20;
        buzzerPattern[BELEDBuzzerConfigBLN::AUTH_FAIL].arg[2] =
    BEOutputPatternBLN::HIGH_FREQ;
        buzzerPattern[BELEDBuzzerConfigBLN::AUTH_FAIL].high[2] = 100;
```

Fields	**Options**
ledPattern	The LED output patterns for pre-defined events.
buzzerPattern	The buzzer output patterns for pre-defined events.
lcdLedPattern	The LCD background patterns for pre-defined events.
keypadLedPattern	The keypad LED patterns for pre-defined events.

BETnaEventConfig

Fields	**Options**
enabled	Specifies if this TNA evet is used.
useRelay	If true, turn on the relay after authentication succeeds.
eventStr	Event string which will be used for showing log records. It should be in UTF-16 format.

BETnaEventExConfig

The settings are effective only if the **tnaMode** of **BEConfigDataBLN** is set to BS_TNA_FUNCTION_KEY.

Fields	**Options**
fixedTnaIndex	Specifies the fixed TNA event. It is effective only if

	the **tnaChange** field of **BEConfigDataBLN** is BS_TNA_FIXED.
manualTnaIndex	Reserved for future use.
timeSchedule	Schedules for each TNA event. It is effective only if the **tnaChange** field of **BEConfigDataBLN** is BS_TNA_AUTO_CHANGE.

BEConfigDataBLN

Fields	**Options**
magicNo version timestamp checksum	These 4 fields are for internal-use only. Users should not update these values.
<u>Operation Mode</u>	
opMode	The semantics of operation modes are different from those of BioEntry Plus and BioStation. Available authentication modes are as follows; FINGER_ONLY: users have to authenticate his fingerprint. PASSWORD_ONLY: users have to authenticate his password. FINGER_OR_PASSWORD: users have to authenticate his fingerprint or password. FINGER_AND_PASSWORD: users have to authenticate his fingerprint and password. CARD_ONLY: If an enrolled card is read, access is allowed without fingerprint or password authentication. The default mode is FINGER_ONLY.
opModeSchedule	You can mix up to 4 authentication modes based on time schedules. If more than one authentication modes are used, the time schedules of them should not be overlapped.
opDualMode	If it is true, two users should be authenticated before a door is opened.
opModePerUser	If true, the **opMode** field of **BEUserHdr** will be applied to user. Otherwise, the **opMode** field of

	BEConfigDataBLN will be applied.
identificationMode	It specifies how to initiate 1:N authentication. OP_1TON_FREESCAN: 1:N mode is started as soon as user place his finger on the sensor. OP_1TON_OK_KEY: User has to press the OK button first before scanning his finger. OP_1TON_NONE: 1:N mode is disabled.
identificationMode Schedule	You can mix up to 3 identification modes based on time schedules. If more than one identification modes are used, the time schedules of them should not be overlapped.

Ethernet

useDHCP	Specifies if DHCP is used.
ipAddr	IP address of the device.
gateway	Gateway address.
subnetMask	Subnet mask.
port	Port number of the TCP connection.
useServer	If true, connect to the server with **serverIPAddr** and **port**. If false, open the TCP port and wait for incoming connections.
serverIPAddr	IP address of the server.
synchTime	If true, synchronize system clock with server when connecting to it.

Fingerprint

securityLevel	Sets the security level. AUTOMATIC_NORMAL – FAR(False Acceptance Ratio) is 1/10,000 AUTOMATIC_SECURE – FAR is 1/100,000 AUTOMATIC_MORE_SECURE - FAR is 1/1,000,000
fastMode	**fastMode** can be used to shorten the 1:N matching time with little degradation of authentication performance. If it is set to FAST_MODE_AUTO, the matching speed will be adjusted automatically according to the number of enrolled templates. FAST_MODE_AUTO FAST_MODE_NORMAL

	FAST_MODE_FAST FAST_MODE_FASTER
timeout	Specifies the timeout for fingerprint input in seconds.
matchTimeout	If 1:N matching is not finished until this period, NOT_FOUND error will be returned. The default value is 3 seconds.
templateType	TEMPLATE_SUPREMA TEMPLATE_SIF – ISO 19794-2
fakeDetection	If true, the device will try to detect fake fingers.
useServerMatching	In server matching mode, user authentication is handled by BioStar server, not each device. To use server matching, the **useServer** of **BEConfigData** should be true.
useCheckDuplicate	If true, the device will check if the same fingerprint was registered already before enrolling new users.

<u>I/O</u>

inputConfig	See BSWriteInputConfig.
outputConfig	See BSWriteOutputConfig.
doorConfig	See BSWriteDoorConfig.

<u>Serial</u>

hostID	The ID of the host device. Note that **hostID**, **slaveIDEx**, **slaveType,** and **secureIO** will be set automatically by the device after calling **BS_Search485Slaves**. You should not set these values manually.
slaveIDEx	The IDs of slave devices connected to the RS485 network.
slaveType	The types of slave devices.
serialMode	RS485 connection of a BioEntry Plus can be used as one of the followings; SERIAL_DISABLED: not used. SERIAL_IO_HOST_EX: acts as a host device and controls all the I/O operations of Secure I/O devices and slave devices connected to the same RS485 connection. SERIAL_IO_SLAVE_EX: acts as a slave device. SERIAL_PC: used as a communication channel to host

	PC.
serialBaudrate	Specifies the baudrate of RS485 connection when serialMode is SERIAL_PC. In other cases, it is ignored.
secureIO	A Secure I/O device has an index between 0 and 3. This flag specifies which Secure I/O devices are connected to the RS485 connection. 0x01: Secure I/O 0 0x02: Secure I/O 1 0x04: Secure I/O 2 0x08: Secure I/O 3 If it is 0x07, it means that Secure I/O 0, 1, and 2 are connected. This field will be set automatically by the device after **BS_Search485Slaves** succeeds.
slaveID	Not used.

Entrance Limitation

minEntryInterval numOfEntranceLimit maxEntry entryLimitInterval	Entrance limitation can be applied to a single device. See **BSEntrnaceLimitationZoneProperty** for details.
bypassGroupId	The ID of a group, the members of which can bypass the restriction of entrance limitation settings.

TNA

tnaMode	BS_TNA_DISABLE – TNA is disabled. BS_TNA_FUNCTION_KEY – TNA function keys are enabled. With this mode, the **tnaChange** field is effective. BS_TNA_AUTO – TNA events are selected by left and right arrow keys.
tnaChange	BS_TNA_AUTO_CHANGE – TNA event is changed automatically according to the schedule defined in **BETnaEventExConfig**. BS_TNA_MANUAL_CHANGE – TNA event is changed manually by arrow keys. BS_TNA_FIXED – TNA event is fixed to the **fixedTnaIndex** of **BETnaEventExConfig**.

User

defaultAG	The default access group of users. It is either BSAccessGroupEx::NO_ACCESS_GROUP or BSAccessGroupEx::FULL_ACCESS_GROUP. This access group is applied to the following cases. (1) When a user has no access group. For example, if **defaultAG** is NO_ACCESS_GROUP, users without access groups are not allowed to enter. (2) When a user has invalid access group. (3) When enrolling users by command card.

Wiegand

useWiegandOutput	If it is true, Wiegand signal will be output when authentication succeeds.
wiegandIdType	Specifies whether the Wiegand bitstream should be interpreted as a user ID or a cardID. WIEGAND_USER WIEGAND_CARD
wiegandConfig	See **BS_WriteWiegandConfig**.

LED/Buzzer

ledBuzzerConfig	See **BELEDBuzzerConfigBLN**.

Misc.

backlightMode	ALWAYS_ON ALWAYS_OFF ON_AT_USE – triggered by user input
soundMode	ALWAYS_ON ALWAYS_OFF

TNA Ex.

tnaEventConfig	See **BETnaEventConfig**.
tnaEventExConfig	See **BETnaEventExConfig**.

Return Values

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

Compatibility

BioLite Net

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.

Compatibility

BioStation

3.8. Access Control API

These APIs provide access control features such as time schedule and access group.

- BS_AddTimeScheduleEx: adds a time schedule.
- BS_GetAllTimeScheduleEx: reads all time schedules.
- BS_SetAllTimeScheduleEx: writes all time schedules.
- BS_DeleteTimeScheduleEx: deletes a time schedule.
- BS_DeleteAllTimeScheduleEx: deletes all time schedules.
- BS_AddHolidayEx: adds a holiday schedule.
- BS_GetAllHolidayEx: reads all holiday schedules.
- BS_SetAllHolidayEx: writes all holiday schedules.
- BS_DeleteHolidayEx: deletes a holiday schedule.
- BS_DeleteAllHolidayEx: deletes all holiday schedules.
- BS_AddAccessGroupEx: adds an access group.
- BS_GetAllAccessGroupEx: reads all access groups.
- BS_SetAllAccessGroupEx: writes all access groups.
- BS_DeleteAccessGroupEx: deletes an access group.
- BS_DeleteAllAccessGroupEx: deletes all access groups.
- BS_ControlRelayEx: controls the relay of a device.
- BS_DoorControl: controls the door relay of a device.

BS_AddTimeScheduleEx

Up to 128 time schedules can be stored to a device. Each time schedule consists of 7 daily schedules and two optional holiday schedules. And each daily schedule may have up to 5 time segments. There are also two pre-defined schedules, NO_TIME_SCHEDULE and ALL_TIME_SCHEDULE, which cannot be updated nor deleted.

BS_RET_CODE BS_AddTimeScheduleEx(int handle, BSTimeScheduleEx* schedule)

Parameters

handle

Handle of the communication channel.

schedule

Pointer to the time schedule to be added. BSTimeScheduleEx is defined as follows;

```
struct BSTimeCodeElemEx {
     unsigned short startTime;
     unsigned short endTime;
};

struct BSTimeCodeEx {
     BSTimeCodeElemEx codeElement[BS_TIMECODE_PER_DAY_EX];
};

struct BSTimeScheduleEx {
     enum {
         // pre-defined schedule ID
         NO_TIME_SCHEDULE   = 0xFD,
         ALL_TIME_SCHEDULE  = 0xFE,

         NUM_OF_DAY = 9,
         NUM_OF_HOLIDAY = 2,

         SUNDAY      = 0,
         MONDAY      = 1,
         TUESDAY     = 2,
         WEDNESDAY   = 3,
         THURSDAY    = 4,
         FRIDAY      = 5,
         SATURDAY    = 6,
         HOLIDAY1    = 7,
         HOLIDAY2    = 8,
     };

     int scheduleID; // 1 ~ 128
     char name[BS_MAX_ACCESS_NAME_LEN];
     int holiday[2]; // 0 for unused
     BSTimeCodeEx timeCode[NUM_OF_DAY]; // 0 - Sunday, 1 - Monday, ...
};
```

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

Example

```
BSTimeScheduleEx timeSchedule;

memset( &timeSchedule, 0, sizeof(BSTimeScheduleEx) );

timeSchedule.scheduleID = 1;
timeSchedule.holiday[0] = 1;

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

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

```
// Holiday 1- 10:00 ~ 14:00
timeSchedule.timeCode[BSTimeScheduleEx::HOLIDAY1].codeElement[0].startTime
= 10 * 60;
timeSchedule.timeCode[BSTimeScheduleEx::HOLIDAY1].codeElement[0].endTime =
14 * 60;

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

// …

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

BS_GetAllTimeScheduleEx

Reads all the registered time schedules.

BS_RET_CODE BS_GetAllTimeScheduleEx(int handle, int* numOfSchedule, BSTimeScheduleEx* 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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_SetAllTimeScheduleEx

Writes time schedules.

BS_RET_CODE BS_SetAllTimeScheduleEx(int handle, int numOfSchedule, BSTimeScheduleEx* schedule)

Parameters

handle

Handle of the communication channel.

numOfSchedule

Number of schedules to be written.

schedule

Pointer to the time schedule array to be written.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_DeleteTimeScheduleEx

Deletes the specified time schedule.

BS_RET_CODE BS_DeleteTimeScheduleEx(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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_DeleteAllTimeScheduleEx

Deletes all the time schedules stored in a device.

BS_RET_CODE BS_DeleteAllTimeScheduleEx(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_AddHolidayEx

Adds a holiday list. Up to 32 holiday lists can be stored to a device.

BS_RET_CODE BS_AddHolidayEx(int handle, BSHolidayEx* holiday)

Parameters

handle

Handle of the communication channel.

holiday

Pointer to the holiday list to be added. BSHolidayEx is defined as follows;

```
struct BSHolidayElemEx {
     enum {
         // flag
         ONCE = 0x01,
     };

     unsigned char flag;
     unsigned char year; // since 2000
     unsigned char month; // 1 ~ 12
     unsigned char startDay; // 1 ~ 31
     unsigned char duration; // 1 ~ 100
};


struct BSHolidayEx {
     enum {
         MAX_HOLIDAY = 32,
     };

     int holidayID; // 1 ~ 32
     char name[BS_MAX_ACCESS_NAME_LEN];
     int numOfHoliday;
     BSHolidayElemEx holiday[MAX_HOLIDAY];
};
```

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

Example

```
BSHolidayEx holiday;

memset( &holiday, 0, sizeof(BSHolidayEx) );

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

// Jan. 1 ~ 3 are holidays in every year
holiday.holiday[0].year = 7;
holiday.holiday[0].month = 1;
holiday.holiday[0].startDate = 1;
holiday.holiday[0].duration = 3;

// 2007 Mar. 5 is holiday
Holiday.holiday[1].flag = BSHolidayElemEx::ONCE;
holiday.holiday[1].year = 7;
holiday.holiday[1].month = 3;
holiday.holiday[1].startDate = 5;
holiday.holiday[1].duration = 1;

// …

strcpy( holiday.name, “Holiday 1” );

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

BS_GetAllHolidayEx

Reads all the registered holiday lists.

BS_RET_CODE BS_GetAllHolidayEx(int handle, int* numOfHoliday, BSHolidayEx* holiday)

Parameters

handle

Handle of the communication channel.

numOfHoliday

Pointer to the number of enrolled holiday lists.

holiday

Pointer to the holiday lists to be read.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_SetAllHolidayEx

Writes holiday lists.

BS_RET_CODE BS_SetAllHolidayEx(int handle, int numOfHoliday, BSHolidayEx* holiday)

Parameters

handle

Handle of the communication channel.

numOfHoliday

Number of holiday lists to be written.

holiday

Pointer to the holiday lists to be written.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_DeleteHolidayEx

Deletes the specified holiday list.

BS_RET_CODE BS_DeleteHolidayEx(int handle, int ID)

Parameters

handle

Handle of the communication channel.

ID

ID of the holiday list.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_DeleteAllHolidayEx

Deletes all the holiday lists stored in a device.

BS_RET_CODE BS_DeleteAllHolidayEx(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_AddAccessGroupEx

An access group defines which doors users have access to, and during which hours they have access to these doors. Up to 128 access groups can be stored to a device. There are also two pre-defined access groups, NO_ACCESS_GROUP and FULL_ACCESS_GROUP, which cannot be updated nor deleted.

BS_RET_CODE BS_AddAccessGroupEx(int handle, BSAccessGroupEx* group)

Parameters

handle

Handle of the communication channel.

group

Pointer to the access group to be added. BSAccessGroupEx is defined as follows;

```
struct BSAccessGroupEx {
     enum {
         // pre-defined group
         NO_ACCESS_GROUP= 0xFD,
         FULL_ACCESS_GROUP  = 0xFE,
         // pre-defined door
         ALL_DOOR    = 0x00,
         MAX_READER = 32,
     };

     int groupID; // 1 ~ 128
     char name[BS_MAX_ACCESS_NAME_LEN];
     int numOfReader;
     unsigned readerID[MAX_READER];
     int scheduleID[MAX_READER];
};
```

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

Example

```
// Access Group 1 has access to
// - device 1001 at all time
// - device 1002 at schedule 1
// - device 1003 at schedule 2

BSAccessGroupEx accessGroup;

memset( &accessGroup, 0, sizeof(BSAccessGroupEx) );

accessGroup.groupID = 1;
accessGroup.numOfReader = 3;

accessGroup.readerID[0] = 1001;
accessGroup.scheduleID[0] = BSTimeScheduleEx::ALL_TIME_SCHEDULE;

accessGroup.readerID[1] = 1002;
accessGroup.scheduleID[1] = 1;

accessGroup.readerID[2] = 1003;
accessGroup.scheduleID[2] = 2;
```

BS_GetAllAccessGroupEx

Reads all the registered access groups.

BS_RET_CODE BS_GetAllAccessGroupEx(int handle, int* numOfAccessGroup, BSAccessGroupEx* group)

Parameters

handle

Handle of the communication channel.

numOfAccessGroup

Pointer to the number of registered 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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_SetAllAccessGroupEx

Writes access groups.

BS_RET_CODE BS_SetAllAccessGroupEx(int handle, int numOfAccessGroup, BSAccessGroupEx* group)

Parameters

handle

Handle of the communication channel.

numOfAccessGroup

Number of access groups to be written.

group

Pointer to the access groups to be written.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_DeleteAccessGroupEx

Deletes the specified access group.

BS_RET_CODE BS_DeleteAccessGroupEx(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.

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_DeleteAllAccessGroupEx

Deletes all the access groups stored in a device.

BS_RET_CODE BS_DeleteAllAccessGroupEx(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_RelayControlEx

Controls the relays under the control of a device.

BS_RET_CODE BS_RelayControlEx(int handle, int deviceIndex, int relayIndex, bool onoff)

Parameters

handle

Handle of the communication channel.

deviceIndex

Device index between BS_DEVICE_PRIMARY and BS_DEVICE_SECUREIO3.

relayIndex

BS_PORT_RELAY0 or BS_PORT_RELAY1.

onoff

If true, turn on the relay, and vice versa.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

BS_DoorControl

Turn on or off a door. See BSDoorConfig for configuration of doors.

BS_RET_CODE BS_DoorControl(int handle, int doorIndex, bool onoff)

Parameters

handle

Handle of the communication channel.

doorIndex

0 – Door 1

1 – Door 2

2 - Both

onoff

If true, turn on the relay, and vice versa.

Return Values

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

Compatibility

BioStation/BioEntry Plus/BioLite Net

3.9. Smartcard API

BioStation Mifare, BioEntry Plus Mifare, and BioLite Net support Mifare types of smart cards[5]. These functions provide basic functionalities such as read, write, and format smartcards.

- BS_WriteMifareConfiguration: writes Mifare configuration.
- BS_ReadMifareConfiguration: reads Mifare configuration.
- BS_ChangeMifareKey: changes site keys for encrypting cards.
- BS_WriteMifareCard: writes user information into a Mifare card.
- BS_ReadMifareCard: reads user information from a Mifare card.
- BS_FormatMifareCard: formats a Mifare card.
- BS_AddBlacklist: adds a user ID or card CSN to the blacklist.
- BS_DeleteBlacklist: deletes a user ID or card CSN from the blacklist.
- BS_DeleteAllBlacklist: clears the blacklist.
- BS_ReadBlacklist: reads the blacklist.

[5] Note that BioLite Net supports Mifare cards as default.

BS_WriteMifareConfiguration/BS_ReadMifareConfiguration

Writes/reads the Mifare configuration. The configuration is divided into three parts – operation option, key option, and the card layout. BioStation Mifare, BioEntry Plus Mifare, and BioLite Net devices can handle both 1K and 4K Mifare cards. Maximum 2 templates can be stored into a 1K card, and 4 templates into a 4K card. Changing card layout should be handled with utmost caution. If you are not sure what to do, contact to support@supremainc.com before trying yourself.

BS_RET_CODE BS_WriteMifareConfiguration(int handle, BSMifareConfig* config)
BS_RET_CODE BS_ReadMifareConfiguration(int handle, BSMifareConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSMifareConfig is defined as follows;

```
struct BSMifareConfig {
    enum {
        MIFARE_KEY_SIZE = 6,
        MIFARE_MAX_TEMPLATE = 4,
    };

    // Options
    int magicNo;   // read-only
    int disabled;
    int useCSNOnly;
    int bioentryCompatible;// not used

    // Keys
    int useSecondaryKey;
    int reserved1;
    unsigned char  reserved2[8];
    unsigned char  reserved3[8];

    // Layout
    int cisIndex;
```

```
        int numOfTemplate;
        int templateSize;
        int templateStartBlock[MIFARE_MAX_TEMPLATE];

        int reserve4[15];
   };
```

The key fields and their available options are as follows;

Fields	**Options**
disabled	If true, the device will ignore Mifare cards. The default value is false.
useCSNOnly	If true, the device reads only the 4 byte CSN(Card Serial Number) of a Mifare card. Then, the fingerprint input will be verified with the templates stored in the device. This mode is identical to the operation flow of general RF cards. The default value is false.
useSecondaryKey	When changing the site key, a device has to handle cards with new site key and cards with old site key at the same time. In that case, useSecondaryKey option can be used. If this option is true and the secondary key is set to the old site key, the device is able to handle both types of cards. The default value is false.
cisIndex	The first block index of the user header information. The size of the header is 48 bytes – 3 blocks. And cisIndex should be the first block of any sector. The default value is 4.
numOfTemplate	The number of templates to be stored into a Mifare card. The maximum value is 2 for a 1K card and 4 for a 4K card. The default value is 2.
templateSize	The size of one template. Since the last two bytes are used for checksum, it should be a multiple of 16 minus 2. The default value is 334 – 21 blocks.
templateStartBlock	The first block index of each template. These values should be selected so that there is no overlap between each template. The default

values are {8, 36} for 1K Mifare card.

Return Values

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

Compatibility

BioStation Mifare/BioEntry Plus Mifare/BioLite Net

BS_ChangeMifareKey

To prevent illegal access, Mifare cards are protected by 48 bit site key. The site key should be handled with utmost caution. If it is disclosed, the data on the smartcard will not be secure any more. **BS_ChangeMifareKey** is used to change the primary and secondary site keys. The default primary key is 0xffffffffffff.

BS_RET_CODE BS_ChangeMifareKey(int handle, unsigned char* oldPrimaryKey, unsigned char* newPrimaryKey, unsigned char* newSecondaryKey)

Parameters

handle

Handle of the communication channel.

oldPrimaryKey

Pointer to the 6 byte old primary key. If it is not matched with the one stored in the device, BS_ERR_WRONG_PASSWORD will be returned.

newPrimaryKey

Pointer to the 6 byte new primary key.

newSecondaryKey

Pointer to the 6 byte new secondary key. See useSecondaryKey option in **BS_WriteMifareConfiguration**.

Return Values

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

Compatibility

BioStation Mifare/BioEntry Plus Mifare/BioLite Net

BS_WriteMifareCard

Writes user information into a Mifare card.

BS_RET_CODE BS_WriteMifareCard(int handle, BSMifareCardHeader* header, unsigned char* templateData, int templateSize)

Parameters

handle

Handle of the communication channel.

header

BSMifareCardHeader is defined as follows;

```
struct BSMifareCardHeader {
     enum {
         MAX_TEMPLATE   = 4,
         MAX_ACCESS_GROUP   = 4,
         MAX_NAME_LEN   = 16,
         PASSWORD_LEN   = 8,

         MIFARE_VER_1_0 = 0x10,

         // security level
         USER_SECURITY_DEFAULT  = 0,
         USER_SECURITY_LOWER    = 1,
         USER_SECURITY_LOW      = 2,
         USER_SECURITY_NORMAL   = 3,
         USER_SECURITY_HIGH     = 4,
         USER_SECURITY_HIGHER   = 5,

         // admin level
         USER_LEVEL_NORMAL  = 0,
         USER_LEVEL_ADMIN   = 1,
     };

     unsigned int   CSN;
     unsigned int   userID;
     unsigned int   reserved1;
     unsigned char  version;
     unsigned char  numOfTemplate;
     unsigned char  adminLevel;
     unsigned char  securityLevel;
     unsigned char  duress[MAX_TEMPLATE];
```

```
        unsigned char  isBypassCard;
        unsigned char  reserved2[3];
        unsigned char  accessGroup[MAX_ACCESS_GROUP];
        unsigned char  userName[MAX_NAME_LEN];
        unsigned char  password[PASSWORD_LEN];
        time_t  startTime;
        time_t  expiryTime;
        unsigned int   reserved3[8];
   };
```

The key fields and their available options are as follows;

Fields	**Descriptions**
CSN	4 byte card serial number. It is read-only.
userID	4 byte user ID.
version	Card version. It is read-only.
numOfTemplate	The number of templates to be written into the card. The maximum value is limited by numOfTemplate field in **BSMifareConfig**.
adminLevel	USER_LEVE_NORMAL USER_LEVEL_ADMIN
securityLevel	USER_SECURITY_DEFAULT: same as the device setting USER_SECURITY_LOWER: 1/1000 USER_SECURITY_LOW: 1/10,000 USER_SECURITY_NORMAL: 1/100,000 USER_SECURITY_HIGH: 1/1,000,000 USER_SECURITY_HIGHER: 1/10,000,000
duress	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 duress field denotes which of the enrolled templates is a duress one. For example, if the 1st templates is of a duress finger, duress[0] will be 1.
isBypassCard	If it is true, the user can access without fingerprint authentication.
accessGroup	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, this array should be initialized as {1, 4, 0xff, 0xff}.
userName	Pointer to the user name.
password	Pointer to the password of the user. It is effective only if the authMode field of **BSOPModeConfig** is set for password authentication.
startTime	The time from which the user's authorization takes effect.
expiryTime	The time on which the user's authorization expires.

templateData

Fingerprint templates of the user.

templateSize

The size of one template. If it is different from that of **BSMifareConfig**, the device will truncate or pad the template data according to the latter.

Return Values

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

Compatibility

BioStation Mifare/BioEntry Plus Mifare/BioLite Net

Example

```
BSMifareCardHeader userHeader;

memset( &userHeader, 0, sizeof( BSMifareCardHeader ) );

userHeader.userID = 1; // 0 cannot be assigned as a user ID.
userHeader.numOfTemplate = 2;

userHeader.adminLevel = BSMifareCardHeader::USER_LEVEL_NORMAL;
userHeader.securityLevel = BSMifareCardHeader::USER_SECURITY_DEFAULT;

userHeader.accessGroup[0] = 0xFE; // Full Access group
userHeader.accessGroup[1] = 0xFF;
userHeader.accessGroup[2] = 0xFF;
userHeader.accessGroup[3] = 0xFF;

strcpy( userHeader.name, “John” );
strcpy( userHeader.password, NULL ); // no password

userHeader.startTime = 0; // no start time check
userHeader.expiryTime = US_ConvertToLocalTime( time( NULL ) ) + 365 * 24 *
60 * 60; // 1 year from today

unsigned char* templateBuf = (unsigned
char*)malloc( userHeader.numOfTemplate * BS_TEMPLATE_SIZE );

// fill template data
for( int i = 0; i < userHeader.numOfTemplate; i++ )
{
     unsigned char* templateData = templateBuf + i * BS_TEMPLATE_SIZE;

     // fill data here
}

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

BS_ReadMifareCard

Reads user information from a Mifare card.

BS_RET_CODE BS_ReadMifareCard(int handle, BSMifareCardHeader* header, unsigned char* templateData, int* templateSize)

Parameters

handle

Handle of the communication channel.

header

Pointer to the card header to be returned.

templateData

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

templateSize

Pointer to the size of one template to be returned. It is identical to that of **BSMifareConfig**.

Return Values

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

Compatibility

BioStation Mifare/BioEntry Plus Mifare/BioLite Net

BS_FormatMifareCard

Formats a Mifare card.

BS_RET_CODE BS_FormatMifareCard(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

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

Compatibility

BioStation Mifare/BioEntry Plus Mifare/BioLite Net

BS_AddBlacklist

When a user ID or card CSN is added to the blacklist, the device will reject the corresponding Mifare card. The blacklist can store up to 1000 user IDs or card CSNs.

BS_RET_CODE BS_AddBlacklist(int handle, int numOfItem, BSBlacklistItem* item)

Parameters

handle

Handle of the communication channel.

numOfItem

Number of items to be added.

Item

Arrays of blacklist items to be added.

BSBlacklistItem is defined as follows;

```
struct BSBlacklistItem {
    enum {
        // blacklist type
        BLACKLIST_USER_ID  = 0x01,
        BLACKLIST_CSN  = 0x02,

        MAX_BLACKLIST      = 1000,
    };

    unsigned char itemType;
    unsigned char reserved[3];
    unsigned itemData;
};
```

The key fields and their available options are as follows;

Fields	Options
itemType	BLACKLIST_USER_ID: the itemData is userID. BLACKLIST_CSN: the itemData is 4 byte CSN of a card.
itemData	UserID or CSN according to the itemType.

Return Values

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

Compatibility

BioStation Mifare/BioEntry Plus Mifare/BioLite Net

BS_DeleteBlacklist

Deletes Mifare cards from the blacklist.

BS_DeleteBlacklist(int handle, int numOfItem, BSBlacklistItem* item)

Parameters

handle

Handle of the communication channel.

numOfItem

Number of items to be deleted.

Item

Arrays of blacklist items to be deleted.

Return Values

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

Compatibility

BioStation Mifare/BioEntry Plus Mifare/BioLite Net

BS_DeleteAllBlacklist

Clears the blacklist.

BS_DeleteAllBlacklist(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

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

Compatibility

BioStation Mifare/BioEntry Plus Mifare/BioLite Net

BS_ReadBlacklist

Reads the contents of the blacklist.

BS_ReadBlacklist(int handle, int* numOfItem, BSBlacklistItem* item)

Parameters

handle

Handle of the communication channel.

numOfItem

Pointer to the number of items to be returned.

item

Arrays of white list items to be returned.

Return Values

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

Compatibility

BioStation Mifare/BioEntry Plus Mifare/BioLite Net

3.10. Miscellaneous API

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

- BS_ConvertToUTF8: converts a wide-character string into a UTF8 string.
- BS_ConvertToUTF16: converts a wide-character string into a UTF16 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.

Compatibility

BioStation

BS_ConvertToUTF16

BioLite Net supports UTF16 strings. To display any characters in BioLite Net, it should be converted to UTF16 first.

int BS_ConvertToUTF16(const char* msg, char* utf16Msg, int limitLen)

Parameters

msg

String to be converted.

Utf16Msg

Pointer to the buffer for new string.

limitLen

Maximum size of utf16Msg buffer.

Return Values

If the function succeeds, return the number of bytes written to the utf16Msg buffer. Otherwise, return 0.

Compatibility

BioLite Net

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.

Compatibility

Device independent

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**. Note that these functions are only applicable to BioStation. BioEntry Plus and BioLite Net transfer and save templates in encrypted form always.

void BS_SetKey(unsigned char *key)

Parameters

key

32 byte – 256bit – encryption key.

Return Values

None

Compatibility

BioStation

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.

Compatibility

BioStation

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.

Compatibility

BioStation

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