BioStation SDK

Reference Manual

Rev. 1.4

Revision History

Rev No.	Issued date	**Description**
1.1	2006 Oct. 20	Initial Release
1.2	2007 Jan. 24	APIs are added according to the changes in BioStation firmware V1.2.
1.3	2007 Jun. 18	APIs are added according to the changes in BioStation firmware V1.3.
1.4	2007 Oct. 10	APIs are added according to the changes in BioStation firmware V1.4 and BioEntry Plus V1.0. Server APIs are removed from the manual. If you want to use these APIs, please contact support@supremainc.com.

Contents

1. Introduction

1.1. Contents of the SDK

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

1.2. Usage

1.2.1. Compilation

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

The following snippet shows a typical source file.

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

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

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

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

        // Set the ID of BioStation terminal for further commands
        BS_SetBioStationID( handle, id );

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

1.2.2. Using the DLL

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

1.2.3. Optional Requirements

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

2. API Specification

2.1. Return Codes

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

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

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

2.2. Communication API

To communicate with a BioStation terminal, users should configure the communication channel first. There are six types of communication channels – TCP socket, UDP socket[1], RS232, RS485, USB, and USB memory stick. A BioEntry Plus provides TCP socket and RS485 for general communication, and UDP socket for initial configuration.

- BS_InitSDK: initializes the SDK.
- BS_OpenSocket: opens a TCP socket for LAN communication.
- BS_CloseSocket: closes a TCP socket.
- BS_OpenSocketUDP: opens a UDP socket for receiving IP addresses of BioStation terminals.
- BS_CloseSocketUDP: closes a UDP socket.
- BS_OpenSerial: opens a RS232 port.
- BS_CloseSerial: closes a RS232 port.
- BS_OpenSerial485: opens a RS485 port.
- BS_CloseSerial485: closes a RS485 port.
- BS_OpenUSB: opens a USB port.
- BS_CloseUSB: closes a USB port.
- BS_OpenUSBMemory: opens a USB memory stick for communicating with virtual terminals.
- BS_CloseUSBMemory: closes a USB memory stick.
- BS_OpenInternalUDP: opens a UDP socket for BioEntry Plus.
- BS_CloseInternalUDP: closes a UDP socket.
- BS_SearchDeviceInLAN: searches BioEntry Plus devices in LAN environment.
- BS_WriteConfigUDP: writes the configuration of a BioEntry Plus.
- BS_ReadConfigUDP: reads the configuration of a BioEntry Plus.
- BS_ResetUDP: resets a BioEntry Plus.

[1] UDP Socket cannot be used for general communication. For the specific case in which UDP socket can be used, refer to BS_OpenSocketUDP.

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

BS_OpenSocket

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

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

Parameters

biostationAddr

IP address of BioStation.

port

TCP port number. The default is 1470.

handle

Pointer to the handle to be assigned.

Return Values

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

Compatibility

BioStation/BioEntry Plus

BS_CloseSocket

Closes the socket.

BS_RET_CODE BS_CloseSocket(int handle)

Parameters

handle

Handle of the TCP socket.

Return Values

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

Compatibility

BioStation/BioEntry Plus

BS_OpenSocketUDP

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

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

Parameters

biostationAddr

IP address of BioStation.

port

UDP port number. The default is 1470.

handle

Pointer to the handle to be assigned.

Return Values

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

Compatibility

BioStation

BS_CloseSocketUDP

Closes the UDP socket.

BS_RET_CODE BS_CloseSocketUDP(int handle)

Parameters

handle

Handle of the UDP socket.

Return Values

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

Compatibility

BioStation

BS_OpenSerial

Opens a RS232 port with specified baud rate.

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

Parameters

port

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

baudrate

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

handle

Pointer to the handle to be assigned.

Return Values

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

Compatibility

BioStation

BS_CloseSerial

Closes the serial port.

BS_RET_CODE BS_CloseSerial(int handle)

Parameters

handle

Handle of the serial port.

Return Values

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

Compatibility

BioStation

BS_OpenSerial485

Opens a RS485 port with specified baud rate.

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

Parameters

port

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

baudrate

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

handle

Pointer to the handle to be assigned.

Return Values

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

Compatibility

BioStation/BioEntry Plus

BS_CloseSerial485

Closes the serial port.

BS_RET_CODE BS_CloseSerial485(int handle)

Parameters

handle

Handle of the serial port.

Return Values

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

Compatibility

BioStation/BioEntry Plus

BS_OpenUSB

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

BS_RET_CODE BS_OpenUSB(int* handle)

Parameters

handle

Pointer to the handle to be assigned.

Return Values

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

Compatibility

BioStation

BS_CloseUSB

Closes the USB channel.

BS_RET_CODE BS_CloseUSB(int handle)

Parameters

handle

Handle of the USB channel.

Return Values

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

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. For further details, please refer to the BioStation User Guide.

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

Parameters

driveLetter

Drive letter in which the USB memory stick is inserted.

handle

Pointer to the handle to be assigned.

Return Values

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

Compatibility

BioStation

BS_CloseUSBMemory

Closes the USB memory.

BS_RET_CODE BS_CloseUSBMemory(int handle)

Parameters

handle

Handle of the USB memory.

Return Values

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

Compatibility

BioStation

BS_OpenInternalUDP

With BioStation, users can change network configuration directly using keypad and LCD. Since a BioEntry plus does not have these interfaces, there should be another way to find it in LAN environment and change its network configuration. A BioEntry Plus reserves a UDP port for this purpose. There are 4 functions which can be called with this UDP handle – **BS_SearchDeviceInLAN**, **BS_WriteConfigUDP**, **BS_ReadConfigUDP**, 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

BioEntry Plus

BS_CloseInternalUDP

Closes the UDP socket.

BS_RET_CODE BS_CloseInternalUDP(int handle)

Parameters

handle

Handle of the UDP socket.

Return Values

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

Compatibility

BioEntry Plus

BS_SearchDeviceInLAN

Searches BioEntry Plus devices in LAN environment.

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

Parameters

handle

Handle of the channel opened 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.

readerAddrs

Pointer to the IP addresses of the devices. When a device fails to obtain an IP address from DHCP server, it will be initialized as 0x010AFEA9(169.254.10.1).

Return Values

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

Compatibility

BioEntry Plus

BS_WriteConfigUDP / BS_ReadConfigUDP

Writes/reads the configuration of a BioEntry Plus. See BS_WriteConfig for details.

BS_RET_CODE BS_WriteConfigUDP (int handle, unsigned targetAddr, unsigned targetID, int configType, int dataSize, unsigned char* configData)
BS_RET_CODE BS_ReadConfigUDP (int handle, unsigned targetAddr, unsigned targetID, int configType, int* dataSize, unsigned char* configData)

Parameters

handle

Handle of the channel opened by BS_OpenInternalUDP.

targetAddr

IP address of the target device.

targetID

ID of the target device.

configType

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

BEPLUS_CONFIG – BEConfigData

BEPLUS_CONFIG_SYS_INFO – BESysInfoData

dataSize

Size of the configuration data.

configData

Pointer to the configuration data. See BS_WriteConfig for details.

Return Values

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

Compatibility

BioEntry Plus

BS_ResetUDP

Resets a BioEntry Plus device.

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

Parameters

handle

Handle of the channel opened 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

BioEntry Plus

2.3. Terminal API

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

- BS_GetBioStationID: gets the ID of a BioStation terminal.
- BS_SetBioStationID: sets the ID for further commands.
- BS_GetClientIPAddress: receives the IP addresses of BioStation terminals.
- BS_SearchBioStation: searches the ID of BioStation terminals in a RS485 network.
- BS_GetDeviceID: gets the ID and type of a device.
- BS_SetDeviceID: sets the ID and type for further commands.
- BS_SearchDevice: searches BioStation and BioEntry Plus devices in RS485 network.
- BS_GetTime: gets the time of a terminal.
- BS_SetTime: sets the time of a terminal.
- BS_CheckSystemStatus: checks the status of a terminal.
- BS_Reset: resets a terminal.
- BS_UpgradeEx: upgrades firmware of a terminal.
- BS_Disable: disables a terminal.
- BS_Enable: re-enables a terminal.
- BS_DisableCommunication: disables communication channels.
- BS_EnableCommunication: enables communication channels.
- BS_ChangePasswordBEPlus: changes the master password of a BioEntry plus.
- BS_FactoryDefault: resets system parameters to the default values.

BS_GetBioStationID

To communicate with BioStation, user should know the ID of the terminal attached to the communication channel. In most cases, this is the first function to be called after a communication channel is opened. Please note that **BS_GetDeviceID** should be used to get the device type information in a mixed network of BioStation and BioEntry Plus.

BS_RET_CODE BS_GetBioStationID(int handle, unsigned* biostationID)

Parameters

handle

Handle of the communication channel.

biostationID

Pointer to the ID to be returned.

Return Values

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

Compatibility

BioStation

BS_SetBioStationID

A BioStation terminal will process commands only if the IDs of the packets match with its own. **BS_SetBioStationID** selects a BioStation terminal to which further requests are sent. Please note that **BS_SetDeviceID** should be used for BioEntry Plus.

BS_RET_CODE BS_SetBioStationID(int handle, unsigned id)

Parameters

handle

Handle of the communication channel.

id

ID of the BioStation terminal.

Return Values

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

Compatibility

BioStation

BS_GetClientIPAddress

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

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

Parameters

handle

Handle of the UDP socket.

ipAddr

IP address of the BioStation terminal.

port

Port number of the BioStation terminal.

timeout

Timeout for receiving packets.

Return Values

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

Compatibility

BioStation

Example

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

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

if( result != BS_SUCCESS )
```

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

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

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

BS_CloseSocketUDP( handle )

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

BS_SearchBioStation

Searches BioStation terminals connected to a RS485 network and BioStation USB virtual terminals.

BS_RET_CODE BS_SearchBioStation(int handle, unsigned* IDs, int* numOfBioStation)

Parameters

handle

Handle of the RS485 channel.

IDs

Pointer to the BioStation IDs to be returned.

numOfBioStation

Pointer to the number of BioStation IDs to be returned.

Return Values

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

Compatibility

BioStation

BS_GetDeviceID

When there are both BioStation and BioEntry Plus devices in a network, the types of devices should be known in addition to the IDs. In most cases, this is the first function to be called after a communication channel is opened.

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

BS_SetDeviceID

BioEntry Plus adopts a different packet encryption algorithm from BioStation. Therefore, in order for packets to be processed properly, the type of a device should be also set. You can get the types of devices using **BS_GetDeviceID**, **BS_SearchDevice**, or **BS_SearchDeviceInLAN**.

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

BS_SearchDevice

Searches devices in a RS485 network. Please see **BS_SearchDeviceInLAN** for finding BioEntry Plus devices in LAN environment.

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

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

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

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

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

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

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_EnrollUser( … );
     // …
     BS_Enable( handle );
}
```

BS_Enable

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

BS_RET_CODE BS_Enable(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

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

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_GetBioStationID**, and **BS_GetDeviceID**.

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

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 using **BS_ChangePasswordBEPlus()**.

Return Values

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

Compatibility

BioStation/BioEntry Plus

BS_ChangePasswordBEPlus

Changes the master password of a BioEntry Plus.

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

BS_FactoryDefault

Resets the status of a BioEntry Plus 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

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 );
```

2.4. Log Management API

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

- BS_ClearLogCache: clears the log cache.
- BS_ReadLogCache: reads the log records in the cache.
- BS_GetLogCount: gets the number of log records.
- BS_ReadLog: reads log records.
- BS_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 firmware V1.4.
	RELAY_OFF	0x81	Door is closed.

	DOOR0_OPEN[2]	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.
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	
	SLAVE_INPUT0	0xA2	Detect a signal at input ports of the slave devices.
	SLAVE_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	

[2] To receive door related events(0x82 ~ 0x89), a DOOR SENSOR input should be assigned and wired properly. Please see BS_WriteDoorConfig.

	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	
Entrance Limitation	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.

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_ClearLogCache

A BioStation terminal has a cache which keeps 64 latest log records. The size of it is 128 in a BioEntry Plus. 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

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

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, 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

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

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 *
sizeof(BSLogRecord) );

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

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 supports 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

BS_GetLogCount

Retrieves the number of log records.

BS_RET_CODE BS_GetLogCount(int handle, int* numOfLog)

Parameters

handle

Handle of the communication channel.

numOfLog

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

Return Values

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

Compatibility

BioStation/BioEntry Plus

2.5. Display Setup API

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

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

BS_SetBackground

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

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

Parameters

handle

Handle of the communication channel.

bgIndex

Background index. It should be one of BS_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 6 sound effects in BioStation. Users can replace these sounds using **BS_SetSound**.

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

Parameters

handle

Handle of the communication channel.

soundIndex

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

Index	When to play
BS_SOUND_START	When system starts
BS_SOUND_CLICK	When a keypad is pressed
BS_SOUND_SUCCESS	When authentication or other operations succeed
BS_SOUND_QUESTION	When displaying a dialog for questions or warnings
BS_SOUND_ERROR	When operations fail
BS_SOUND_SCAN	When a fingerprint is detected on the sensor
BS_SOUND_FINGER_ONLY	When waiting for fingerprints
BS_SOUND_PIN_ONLY	When waiting for passwords
BS_SOUND_CARD_ONLY	When waiting for cards
BS_SOUND_FINGER_PIN	When waiting for fingerprints or passwords
BS_SOUND_FINGER_CARD	When waiting for fingerprints or cards
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_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

2.6. User Management API

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

- BS_GetUserDBInfo: gets the basic information of user DB.
- BS_EnrollUser: enrolls a user.
- BS_DeleteUser: deletes a user.
- BS_DeleteAllUser: deletes all users.
- BS_GetUser: gets the fingerprint templates and header information of a user.
- BS_GetUserInfo: gets the header information of a user.
- BS_GetAllUserInfo: gets the header information of all users.
- BS_ScanTemplate: scans a fingerprint on a BioStation terminal and retrieves the template of it.
- BS_EnrollUserEx: enrolls a user with the extended header information.
- BS_EnrollMultipleUserEx: enrolls multiples users with the extended header information
- BS_GetUserEx: gets the fingerprint templates and extended header information of a user.
- BS_GetUserInfo: gets the extended header information of a user.
- BS_GetAllUserInfo: gets the extended header information of all users.
- BS_ReadImage: reads a image of the last scanned fingerprint.
- BS_ReadCardID: reads a Card on a BioStation terminal and retrieves the ID of it.
- 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_EnrollUserBEPlus: enrolls a user to a BioEntry Plus.
- BS_EnrollMultipleUserBEPlus: enrolls multiple users to a BioEntry Plus.
- BS_GetUserBEPlus: gets the fingerprint templates and header information of a user.
- BS_GetUserInfoBEPlus: gets the header information of a user.
- BS_GetAllUserInfoBEPlus: gets the header information of all users.

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

BS_EnrollUser(Deprecated)

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

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

Parameters

handle

Handle of the communication channel.

Hdr

BSUserHdr is defined as follows;

```
typedef struct{
    unsigned ID;
    unsigned short reserved1;
    unsigned short adminLevel;
    unsigned short securityLevel;
    unsigned short statusMask; // internally used by BioStation
    unsigned accessGroupMask;
    char name[BS_MAX_NAME_LEN + 1];
    char department[BS_MAX_NAME_LEN + 1];
    char password[BS_MAX_PASSWORD_LEN + 1];
    unsigned short numOfFinger;
    unsigned short duressMask;
    unsigned short checksum[5];
} BSUserHdr[3];
```

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

[3] BSUserHdr is superseded by BSUserHdrEx.

	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.

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

```
BSUserHdr userHeader;
```

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

strcpy( userHeader.name, “John” );
strcpy( userHeader.departments, “R&D” );
strcpy( userHeader.password, NULL ); // no password is enrolled. Password
                                     // should be longer than 4 bytes.
userHeader.numOfFinger = 2;
unsigned char* templateBuf = (unsigned char*)malloc( userHeader.numOfFinger
* 2 * BS_TEMPLATE_SIZE );

// fill template data

userHeader.duressMask = 0; // no duress finger

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

     unsigned char* templateData = templateBuf + i * BS_TEMPLATE_SIZE;

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

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

BS_DeleteUser

Deletes a user.

BS_RET_CODE BS_DeleteUser(int handle, unsigned userID)

Parameters

handle

Handle of the communication channel.

userID

ID of the user to be deleted.

Return Values

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

Compatibility

BioStation/BioEntry Plus

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

BS_GetUser(Deprecated)

Retrieves the header and template data of a user.

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

Parameters

handle

Handle of the communication channel.

userID

User ID.

hdr

Pointer to the user header to be returned.

templateData

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

Return Values

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

Compatibility

BioStation

BS_GetUserInfo(Deprecated)

Retrieves the header information of a user.

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

Parameters

handle

Handle of the communication channel.

userID

User ID.

hdr

Pointer to the user header to be returned.

Return Values

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

Compatibility

BioStation

BS_GetAllUserInfo(Deprecated)

Retrieves the header information of all enrolled users.

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

Parameters

handle

Handle of the communication channel.

hdr

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

numOfUser

Pointer to the number of enrolled users.

Return Values

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

Compatibility

BioStation

BS_ScanTemplate

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

BS_EnrollUserEx

Enrolls a user with extended header information and fingerprint templates. 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 authLimitCount; // 0 for no limit
    unsigned 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	Same as BSUserHdr.
securityLevel	Same as BSUserHdr.

accessGroupMask	Same as BSUserHdr.
duressMask	Same as BSUserHdr.
checksum	Same as BSUserHdr.
authLimitCount	Specifies how many times the user is permitted to access per day. If it is 0, there is no limit.
timedAntiPassbcak	Specifies the 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.
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

BS_EnrollMultipleUserEx

Enrolls multiple users with extended header information and fingerprint templates.

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 all the users.

Return Values

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

Compatibility

BioStation

BS_GetUserEx

Retrieves the extended header information and template data of a user.

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 extended 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 extended header information of a user.

BS_GetUserInfoEx(int handle, unsigned userID, BSUserHdrEx* hdr)

Parameters

handle

Handle of the communication channel.

userID

User ID.

hdr

Pointer to the extended 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 extended header information of all enrolled users.

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. Otherwise, return the corresponding error code.

Compatibility

BioStation

BS_ReadImage

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

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

Parameters

handle

Handle of the communication channel.

imageType

Type of the image.

Value : 0 - binary image, 1 - gray image.

bitmapImage

Pointer to the image data to be returned.

The bimtmapImgage should be allocated before calling this function.

Return Values

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

Compatibility

BioStation/BioEntry Plus

BS_ReadCardID(Deprecated)

Reads a card on a BioStation or BioEntry Plus and retrieves the ID of it.
This function is useful when the device is used as an enroll station.

BS_RET_CODE BS_ReadCardID(int handle, unsigned int* cardID)

Parameters

handle

Handle of the communication channel.

cardID

Pointer to the Card ID data to be returned.

Return Values

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

Compatibility

BioStation/BioEntry Plus

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.

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.

customID

Pointer to the 1 byte custom ID to be returned.

Return Values

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

Compatibility

BioStation/BioEntry Plus

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_EnrollUserBEPlus

Enrolls a user to a BioEntry Plus. Maximum 2 fingers can be enrolled per user.

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;
} 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. It is useful for disabling users temporarily.

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

Example

```
BEUserHdr userHeader;

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

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

// fill template data
// …

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];
     }
}

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

BS_EnrollMultipleUserBEPlus

Enrolls multiple users with header information and fingerprint templates.

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 all the users.

Return Values

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

Compatibility

BioEntry Plus

BS_GetUserBEPlus

Retrieves the header information and template data of a user.

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

BS_GetUserInfoBEPlus

Retrieves the header information of a user.

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

BS_GetAllUserInfoBEPlus

Retrieves the header information of all enrolled users.

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. Otherwise, return the corresponding error code.

Compatibility

BioEntry Plus

2.7. Configuration API

These APIs provide functionalities for reading/writing system configurations.

- BS_WriteDisplayConfig
- BS_ReadDisplayConfig
- BS_WriteOPModeConfig
- BS_ReadOPModeConfig
- BS_WriteTnaEventConfig
- BS_ReadTnaEventConfig
- BS_WriteTnaEventExConfig
- BS_ReadTnaEventExConfig
- BS_WriteIPConfig
- BS_ReadIPConfig
- BS_WriteFingerprintConfig
- BS_ReadFingerprintConfig
- BS_WriteIOConfig
- BS_ReadIOConfig
- BS_WriteRelayConfig
- BS_ReadRelayConfig
- BS_WriteSerialConfig
- BS_ReadSerialConfig
- BS_WriteUSBConfig
- BS_ReadUSBConfig
- BS_WriteWLANConfig
- BS_ReadWLANConfig
- BS_WriteEncryptionConfig
- BS_ReadEncryptionConfig
- BS_WriteWiegandConfig
- BS_ReadWiegandConfig
- BS_WriteZoneConfig
- BS_ReadZoneConfig
- BS_WriteDoorConfig
- BS_ReadDoorConfig
- BS_WriteInputConfig
- BS_ReadInputConfig

- BS_WriteOutputConfig
- BS_ReadOutputConfig
- BS_WriteConfig
- BS_ReadConfig
- BS_GetAvailableSpace

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 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;
} 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;
} BSOPModeConfig ;
```

The key fields and their available options are as follows;

Fields	Options
authMode	Sets 1:1 matching mode. ● BS_AUTH_FINGER_ONLY – only the fingerprint authentication is allowed. ● BS_AUTH_FINGER_OR_PASSWORD – both the fingerprint and password authentication are allowed. ● BS_AUTH_PASS_ONLY – only the password authentication is allowed. ● 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	Specifies how to change the TNA function index defined in BSTnaEventConfig. ● BS_TNA_AUTO_CHANGE – TNA function index is changed automatically by the schedules defined in BSTnaEventExConfig. ● BS_TNA_MANUAL_CHANGE – TNA function index is changed by TNA function keys. ● BS_TNA_FIXED – TNA function index is fixed as defined in BSTnaEventExConfig.
authSchedule[MAX_AUTH_COUNT]	● authSchedule[0] - Specifies the schedule to which BS_AUTH_FINGER_ONLY mode is applied. ● authSchedule[1] - Specifies the schedule to which BS_AUTH_FINGER_OR_PASSWORD mode is applied. ● authSchedule[2] - Specifies the

	schedule to which BS_AUTH_PASS_ONLY mode is applied. ● authSchedule[3] - Specifies the schedule to which BS_AUTH_CARD_ONLY mode is applied.
`identificationSchedule`	Specifies the schedule in which the 1:N matching mode is enabled
`dualMode`	If it is true, two users should be authenticated before a door is opened.
`dualSchedule`	Specifies the schedule in which the dual mode is enabled
`version`	OP_CONFIG_VERSION14 – This value must be use for F/W V1.4 version

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];
     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 extended TNA event configurations.

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 function index which is applied when tnaChange of BSOPModeConfig is BS_TNA_FIXED.
`manualTnaIndex`	Reserved for internal use.
`timeSchedule`	Specifies the schedules to which each TNA index is applied. It is valid only if tnaChange 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 TCP/IP configurations.

BS_RET_CODE BS_WriteIPConfig(int handle, BSIPConfig* config)
BS_RET_CODE BS_ReadIPConfig(int handle, BSIPConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSIPConfig is defined as follows;

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

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

Return Values

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

Compatibility

BioStation

BS_WriteFingerpringConfig / 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; // 1 ~ 20 sec
    int imageQuality;
    bool viewImage;
    int freeScanDelay;
    int useCheckDuplicate;
    int matchTimeout;
} BSFingerprintConfig;
```

The key fields and their available options are as follows;

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

	for 1:1 matching is defined by BSUserHdr.securityLevel per each user.
fastMode	• BS_FAST_MODE_NORMAL • BS_FAST_MODE_FAST • BS_FAST_MODE_FASTER • 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	• 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 it is true, checks if the same fingerprint is already registered before enrolling. If same finger is found, enrollment would fail.
matchTimeout	Sets the timeout of 1:N matching between 1 and 20 second.

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; // ms
} BSIOConfig;
```

The key fields and their available options are as follows;

Fields	Options
input	Assigns an action to the input port. ● BS_IO_INPUT_DISABLED – no action ● BS_IO_INPUT_EXIT – turn on the relay. ● BS_IO_INPUT_WIEGAND_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.
otuputDuration	Specifies the duration of output signal in milliseconds.

Return Values

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

Compatibility

BioStation

BS_WriteRelayConfig / BS_ReadRelayConfig

BioStation has a relay output for opening a door. These functions write and read the relay configurations.

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

Parameters

handle

Handle of the communication channel.

config

BSRelayConfig is defined as follows;

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

The key fields and their available options are as follows;

Fields	Options
event	Specifies when the relay is activated. ● BS_RELAY_EVENT_ALL - relay is on whenever authentication succeeds. ● BS_RELAY_EVENT_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.
openDuration	Specifies the duration in which the relay is on in seconds. After this duration, the relay will be turned off.

lockSchedule	Specifies the schedule in which the relay should be held on.
unlockSchedule	Specifies the schedule in which the relay should be held off.

Return Values

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

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; // BS_CHANNEL_DISABLED, 9600, 19200, 38400, 57600, 115200
    int rs232;
    int useSecureIO;
    char activeSecureIO[4];
    unsigned slaveID;
    int deviceType;
} BSSerialConfig;
```

The key fields and their available options are as follows;

Fields	Options
rs485	Specifies the baudrate of RS485 port. If useSecureIO is true, this field is ignored.
rs232	Specifies the baudrate of RS232 port.
useSecureIO	If true, RS485 port is used for controlling Secure I/O devices or slave device. If false, it is used for communicating with PC.
activeSecureIO	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. For example, if Secure I/O 0 is connected, activeSecureIO[0] will be 1.
slaveID	The ID of the slave device. If it is 0, it means that no slave device is connected. This field will be ignored if useSecureIO is false or deviceType is not BS_485_HOST_DEVICE.
deviceType	BS_485_HOST_DEIVCE or BS_485_SLAVE_DEVICE. This

	field will be ignored if useSecureIO is false.

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;
} BSUSBConfig;
```

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]; // not used for now
     char wpaPassphrase[64];
} BSWLANPreset;

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

The key fields and their available options are as follows;

Fields	Options
operationMode	Only infrastructure network – managed mode – is supported. • BS_WLAN_MANAGED
authType	There are 3 types of authentication. • BS_WLAN_AUTH_OPEN: no authentication. • BS_WLAN_AUTH_SHARED: shared-key WEP authentication.

<table>
<tr><td></td><td>• BS_WLAN_AUTH_WPA_PSK: WPA authentication using a pre-shared master key.</td></tr>
<tr><td>encryptionType</td><td>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.

<table><tr><td>Authentication</td><td>Supported encryption</td></tr><tr><td>AUTH_OPEN</td><td>NO_ENCRYPTION
WEP</td></tr><tr><td>AUTH_SHARED</td><td>WEP</td></tr><tr><td>WPA_PSK</td><td>TKIP_AES</td></tr></table></td></tr>
<tr><td>keyType</td><td>You can specify WEP keys either in plain ascii text or in binary hex format.
• BS_WLAN_KEY_ASCII
• BS_WLAN_KEY_HEX</td></tr>
<tr><td>essid</td><td>Network ID of the access point to which the BioStation will be connected.</td></tr>
</table>

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_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 Wegand formats are supported. The only constraint is that each field is limited to 32 bits.

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

Parameters

handle

Handle of the communication channel.

config

BSWiegandConfig is defined as follows;

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

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

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

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

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

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

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

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

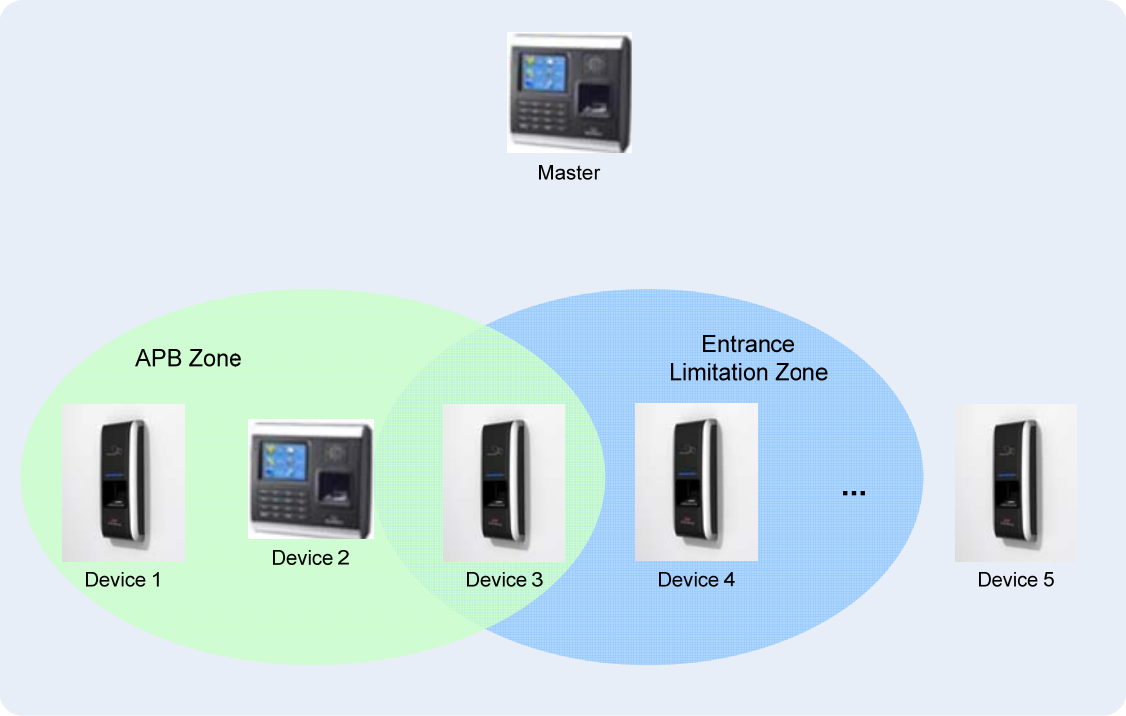
Return Values

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

Compatibility

BioStation

BS_WriteZoneConfig / BS_ReadZoneConfig



Zones are used to group a number of devices in LAN environment. A zone consists of one master device, which handles all the synchronization and authentication functions, and other member devices. Both BioStation and BioEntry Plus can be a master device. The maximum number of devices in a zone is 32.

There are three synchronization options in a zone – time, log, and user. With time option on, the system clock of member devices will be synchronized with that of the master. If log synchronization is on, all the log records of member devices will be stored to the master, too. User synchronization is useful when enrolling/deleting users without host application such as BioAdmin. If a user is enrolled in one device, the user information will be transferred to all the other devices automatically. It is same with deleting users. Please note that user synchronization is applied only when users are enrolled/deleted at devices. For example, if users are enrolled using this SDK, user synchronization option will not take effect.

Users can create anti-passback and entrance limitation sub-zones to enhance security further. Anti-passback means that a user cannot enter a zone twice unless he has exited the zone. All the devices in an anti-passback zone should be grouped

into IN readers and OUT readers. Entrance limitation zones can be created to limit the maximum number of entries in specified time intervals. Please note that user and log synchronization options should be on when anti-passback or entrance limitation sub-zones are used.

BS_RET_CODE BS_WriteZoneConfig(int handle, BSZoneConfig* config)
BS_RET_CODE BS_ReadZoneConfig(int handle, BSZoneConfig* config)

Parameters

handle

Handle of the communication channel.

config

BSZoneConfig is defined as follows;

```
typedef struct {
     unsigned masterIpAddr;
     int authMode;
     int ioMode;
} BSZoneMember;

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

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

typedef union {
     BSAPBZoneProperty apbZone;
     BSEntranceLimitationZoneProperty entLimitZone;
} BSZoneProperty;

typedef struct {
     int type;
     int numOfMember;
     unsigned zoneID; // 0 ~ 255
     BSZoneProperty zoneProperty;
     unsigned memberID[BS_MAX_NODE_PER_ZONE - 1];
     int memberInfo[BS_MAX_NODE_PER_ZONE - 1]; // reader type for APB
```

```
} BSZone;

typedef struct {
     int numOfMember;
     unsigned memberID[BS_MAX_NODE_PER_ZONE - 1];
     unsigned memberIpAddr[BS_MAX_NODE_PER_ZONE - 1];
     int memberStatus[BS_MAX_NODE_PER_ZONE - 1];
     int numOfZone;
     BSZone zoneInfo[BS_MAX_ZONE_PER_MASTER];
} BSZoneMaster;

typedef struct {
     int nodeType;
     int fallbackMode;
     bool synchTime;
     bool synchUser;
     bool synchLog;
     BSZoneMaster zoneMaster;
     BSZoneMember zoneMember;
} BSZoneConfig;
```

The key fields and their available options are as follows;

BSZoneSlave	
Fields	**Options**
masterIpAddr	IP address of the master device. If the IP is 192.168.1.1, it should be 0x0101A8C0.
authMode	Specifies whether authentication should be deferred to the master or not. ● BS_AUTH_STANDALONE: authentication is performed by member devices. ● BS_AUTH_DEFERRED: authentication is performed by the master device. Please note that when anti-passback or entrance limitation zones are created, authMode should be BS_AUTH_DEFERRED.
ioMode	Reserved for future use.

BSAPBZoneProperty	
Fields	**Options**

apbType	● BS_APB_NONE: No anti-passback. ● BS_APB_SOFT: When anti-passback is violated, access is permitted after writing APB_FAIL log record. ● BS_APB_HARD: When anti-passback is violated, access is denied.
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 IN door again after 120 minutes.

BSEntrnaceLimitationZoneProperty	
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).

BSZone	
Fields	**Options**
type	• APB_ZONE • ENTRANCE_LIMIT_ZONE
numOfMember	The number of member devices making up the zone.
zoneID	The ID of the zone. It should be between 0 and 255.
zoneProperty	BSAPBZoneProperty or BSEntranceLimitationZoneProperty
memberID	The IDs of member devices.
memberInfo	When the type is APB_ZONE, memberInfo denotes whether a member device is an IN_READER or OUT_READER. This field is not used with ENTRANCE_LIMIT_ZONE.

BSZoneMaster	
Fields	**Options**
numOfMember	The number of member devices except for the master.
memberID	The IDs of member devices.
memberIpAddr	The IP addresses of member devices.
memberStatus	This field is for internal use only. Users do not have to set these values.
numOfZone	The number of sub-zones defined in this zone.
zoneInfo	BSZone structures defined in this zone.

BSZoneConfig	
Fields	**Options**
nodeType	• BS_STANDALONE_NODE: this device is not

	a member of the zone. ● BS_MASTER_NODE: this device is the master of the zone. ● BS_MEMBER_NODE: this device is a member of the zone.
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.
zoneMaster	The information of the master device. It is valid only if the nodeType is BS_MASTER_NODE.
zoneMember	The information of a member device. It is valid only if the nodeType is BS_MEMBER_NODE.

Return Values

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

Compatibility

BioStation/BioEntry Plus

Example

```
// (1) Zone 1 consists of 5 devices – Device 1011(master, 192.168.1.11),
//      Device 1012(192.168.1.12), Device 1013(192.168.1.13)
//      Device 1014(192.168.1.14), Device 1015(192.168.1.15)
// (2) Zone 1 has two sub-zones as follows;
//      Anti-passback zone
//          - Members: Device 1011(IN), Device 1012(OUT), Device 1013(OUT)
//          - Property: Hard
//      Entrance limitation zone
//          - Members: Device 1014, Device 1015
//          - Max Entry:   1 time(8:30 ~ 10:00)
//                         1 time(12:00 ~ 13:30)
//                         2 times( 18:00 ~ 19:30)

//
```

```
// BSZoneConfig for the master device
//
BSZoneConfig masterConfig;
memset( &masterConfig, 0, sizeof( BSZoneConfig ) );

masterConfig.nodeType = BS_MASTER_NODE;
masterConfig.synchTime = true;
masterConfig.synchUser = true;
masterConfig.synchLog = true;

masterConfig.zoneMaster.numOfMember = 4;
masterConfig.zoneMaster.memberID[0] = 1012;
masterConfig.zoneMaster.memberID[1] = 1013;
masterConfig.zoneMaster.memberID[2] = 1014;
masterConfig.zoneMaster.memberID[3] = 1015;
masterConfig.zoneMaster.memberIpAddr[0] = inet_addr( “192.168.1.12” );
masterConfig.zoneMaster.memberIpAddr[1] = inet_addr( “192.168.1.13” );
masterConfig.zoneMaster.memberIpAddr[2] = inet_addr( “192.168.1.14” );
masterConfig.zoneMaster.memberIpAddr[3] = inet_addr( “192.168.1.15” );

masterConfig.zoneMaster.numOfZone = 2;
masterConfig.zoneMaster.zoneInfo[0].type = BSZone::APB_ZONE;
masterConfig.zoneMaster.zoneInfo[0].numOfMember = 3;
masterConfig.zoneMaster.zoneInfo[0].zoneID = 0;
masterConfig.zoneMaster.zoneInfo[0].zoneProperty.apbZone.apbType =
BS_APB_HARD;
masterConfig.zoneMaster.zoneInfo[0].memberID[0] = 1011;
masterConfig.zoneMaster.zoneInfo[0].memberInfo[0] = BSZone::IN_READER;
masterConfig.zoneMaster.zoneInfo[0].memberID[1] = 1012;
masterConfig.zoneMaster.zoneInfo[0].memberInfo[1] = BSZone::OUT_READER;
masterConfig.zoneMaster.zoneInfo[0].memberID[2] = 1013;
masterConfig.zoneMaster.zoneInfo[0].memberInfo[2] = BSZone::OUT_READER;

masterConfig.zoneMaster.zoneInfo[1].type = BSZOne::ENTRANCE_LIMIT_ZONE;
masterConfig.zoneMaster.zoneInfo[1].numOfMember = 2;
masterConfig.zoneMaster.zoneInfo[1].zoneID = 1;
masterConfig.zoneMaster.zoneInfo[1].zoneProperty.entLimitZone.numOfEntrance
Limit = 3;
masterConfig.zoneMaster.zoneInfo[1].zoneProperty.entLimitZone.maxEntry[0] =
1;
masterConfig.zoneMaster.zoneInfo[1].zoneProperty.entLimitZone.entryLimitInt
erval[0] = (8 * 60 + 30) | ((10 * 60) << 16);
masterConfig.zoneMaster.zoneInfo[1].zoneProperty.entLimitZone.maxEntry[1] =
1;
masterConfig.zoneMaster.zoneInfo[1].zoneProperty.entLimitZone.entryLimitInt
erval[1] = (12 * 60) | ((13 * 60 + 30) << 16);
```

```
masterConfig.zoneMaster.zoneInfo[1].zoneProperty.entLimitZone.maxEntry[2] =
2;
masterConfig.zoneMaster.zoneInfo[1].zoneProperty.entLimitZone.entryLimitInt
erval[2] = (18 * 60) | ((19 * 60 + 30) << 16);
masterConfig.zoneMaster.zoneInfo[1].memberID[0] = 1014;
masterConfig.zoneMaster.zoneInfo[1].memberID[1] = 1015;

BS_RET_CODE result = BS_WriteZoneConfig( masterHandle, &masterConfig );

//
// BSZoneConfig for member devices
//
BSZoneConfig memberConfig;
memset( &memberConfig, 0, sizeof( BSZoneConfig ) );

memberConfig.nodeType = BS_MEMBER_NODE;
memberConfig.synchTime = true;
memberConfig.synchUser = true;
memberConfig.synchLog = true;

memberConfig.zoneMember.masterIPAddr = inet_addr( “192.168.1.11” );
memberConfig.zoneMember.authMode = BS_AUTH_DEFERRED;

// write this config to Device 1012, 1013, 1014 and 1015
```

BS_WriteDoorConfig / BS_ReadDoorConfig

A BioStation or BioEntry Plus can control up to 4 Secure I/O devices and 1 slave device through RS485 connection. A Secure I/O device has 4 SW inputs and 2 output relays. A BioStation or BioEntry Plus has 2 SW inputs and 1 output relay. Among these I/O ports, maximum 2 relays can be assigned to a door. And two pre-defined input switches per relay can be used as a door sensor and a RTE(Request To Exit). You can also set up anti-passback between the host and the slave devices.
Please note that RS485 should be configured first for **BSDoorConfig** to take effect. For a BioEntry Plus to control Secure I/O devices and a slave, the serialMode of **BEConfigData** should be BEConfigData::SERIAL_IO_HOST. And secureIO and slaveID fields should also be set accordingly. See the description of **BEConfigData** in **BS_WriteConfig** for details. For BioStation, refer to **BSSerialConfig** in **BS_WriteSerialConfig**.

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];
};
```

```
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 ● HOST_RELAY: the relay of the host device ● SLAVE_RELAY: the relay of the slave device ● SECUREIO0_RELAY0 ● SECUREIO0_RELAY1 ● SECUREIO1_RELAY0 ● SECUREIO1_RELAY1 ● SECUREIO2_RELAY0 ● SECUREIO2_RELAY1 ● SECUREIO3_RELAY0 ● SECUREIO3_RELAY1
useRTE	Each relay has two pre-defined input switches, which can be used as a RTE and a door sensor respectively. For example, if HOST_RELAY is used, the input port 0 of the host device can be used as a RTE SW, and the input port 1 as a door sensor. If SECUREIO0_RELAY0 is used, the input 0 of the Secure IO 0 can be used as a RTE SW, and the input 1 as a door sensor, etc. useRTE and useDoorSensor fields defines whether these pre-defined switches are used or not.
userDoorSensor	See above.
openEvent	Specifies when the relay is activated in BioStation. This field is ignored by BioEntry Plus.

	• 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_DOORx_HELD_OPEN event will be generated. To detect this and BE_EVENT_DOORx_FORCED_OPEN 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	Specifies which devices are attached to the door. For example, if both the host and the slave devices are attached, reader[0] and reader[1] should be set to HOST_READER and

	SLAVE_READER respectively. If only the host device is attached, set reader[0] to HOST_READER and reader[1] to NO_READER. ● NO_READER ● HOST_READER – The host devices acts as a reader. ● SLAVE_READER – The slave device acts as a reader.

BSDoorConfig	
Fields	**Options**
doorMode	Maximum two doors can be controlled by a host device. ● NO_DOOR ● ONE_DOOR ● TWO_DOOR
door	BSDoor information for each door. If doorMode is ONE_DOOR, only door[0] is valid. If doorMode is TWO_DOOR, both door[0] and door[1] are valid.
apbType	Anti-passback can be set up between the host device and the slave device. ● BS_APB_NONE: No anti-passback. ● BS_APB_SOFT: When anti-passback is violated, access is permitted after writing APB_FAIL log record. ● BS_APB_HARD: When anti-passback is violated, access is denied.
apbResetTime	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 IN door again after 120 minutes.

Return Values

If the function succeeds, return BS_SUCCESS. Otherwise, return the

corresponding error code.

Compatibility

BioStation/BioEntry Plus

Example

```
// The host and the slave devices are attached to one door, which uses the
// relay output 0 of Secure I/O 0 as a door relay. Hard anti-passback is
// set up between the two devices.
BSDoorConfig doorConfig;
memset( &doorConfig, 0, sizeof( BSDoorConfig ) );

doorConfig.doorMode = BSDoorConfig::ONE_DOOR;

doorConfig.door[0].relay = SECUREIO0_RELAY0;
doorConfig.door[0].useRTE = true;
doorConfig.door[0].useDoorSensor = true;
doorConfig.door[0].RTEType = BSDoor::NORMALLY_OPEN;
doorConfig.door[0].sensorType = BSDoor::NORMALLY_OPEN;
doorConfig.door[0].openTime = 3; // 3 sec
doorConfig.door[0].heldOpenTime = 15; // 15 sec
doorConfig.door[0].reader[0] = BSDoor::HOST_READER;
doorConfig.door[0].reader[1] = BSDoor::SLAVE_READER;

doorConfig.apbType = BS_APB_HARD;
```

BS_WriteInputConfig / BS_ReadInputConfig

A BioStation or BioEntry Plus can control up to 4 Secure I/O devices and 1 slave device through RS485 connection. A Secure I/O device has 4 SW inputs. A BioStation or BioEntry Plus has 2 SW inputs.
Please note that RS485 should be configured first for **BSInputConfig** to take effect. As for BioEntry Plus, the relationship between the serialMode of **BEConfigData** and I/O functionalities are as follows;

serialMode	Host I/O	Slave I/O	Secure I/O
SERIAL_DISABLED	O	X	X
SERIAL_IO_HOST	O	O	O
SERIAL_IO_SLAVE	X	X	X
SERIAL_PC	O	X	X

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;
};

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];
};
```

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.
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**
hostTamper	Specifies the function which will be executed when the tamper switch of the host device is turned on.
hostInput	Specifies the input functions of the host device.
secureIO	Specifies the input functions of Secure I/O

	devices connected to the host.
slaveTamper	Specifies the function which will be executed when the tamper switch of the slave device is turned on.
slaveInput	Specifies the input functions of the slave device.

Return Values

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

Compatibility

BioStation/BioEntry Plus

Example

```
// (1) Lock the device when the host 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.hostTamper.functionType = BSInputFunction::LOCK_READER;
inputConfig.hostTamper.minimumDuration = 100; // 100 ms
inputConfig.hostTamper.switchType = BSDoor::NORMALLY_OPEN;
inputConfig.hostTamper.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 or BioEntry Plus can control up to 4 Secure I/O devices and 1 slave device through RS485 connection. A Secure I/O device has 2 relay outputs. A BioStation or BioEntry Plus has 1 relay output. Users can assign multiple output events to each relay. 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

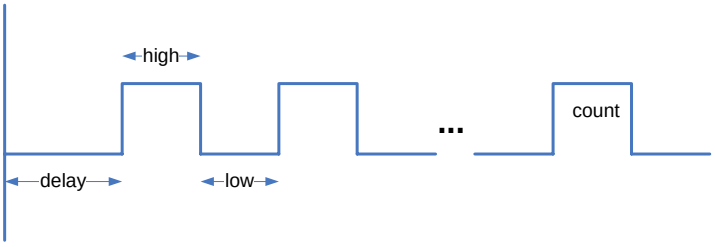
BSOutputConfig is defined as follows;

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

struct BSEMOutputEvent {
     unsigned short inputType;
     unsigned short outputRelayID;
     unsigned short inputDuration;
     unsigned short high;
     unsigned short low;
     unsigned short count;
};

struct BSOutputConfig {
```

```
    int numOfEvent;
    BSOutputEvent outputEvent[MAX_OUTPUT];
    BSEMOutputEvent
emergencyEvent[BSInputConfig::NUM_OF_SECURE_IO][BSInputConfig::NUM_OF_SECUR
E_INPUT];
};
```

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 ● ANTI_PASSBACK_FAIL ● ACCESS_NOT_GRANTED ● ENTRANCE_LIMITATION ● ADMIN_AUTH_SUCCESS ● TAMPER_ON ● DOOR_OPEN ● DOOR_CLOSED ● DOOR_FORCED_OPEN ● DOOR_HELD_OPEN_WARNING ● INPUT0_ON ● INPUT1_ON ● INPUT2_ON ● INPUT3_ON The available device IDs are as follows; ● BS_DEVICE_HOST ● BS_DEVICE_SLAVE ● BS_DEVICE_SECUREIO0 ● BS_DEVICE_SECUREIO1 ● BS_DEVICE_SECUREIO2 ● BS_DEVICE_SECUREIO3 ● BS_DEVICE_ALL

	Here are some examples; (1) When a duress finger is detected at the slave device, AUTH_DURESS | (BS_DEVICE_SLAVE << 16) (2) When the input SW 0 of Secure IO 0 is activated, INPUT0_ON | (BS_DEVICE_SECUREIO0 << 16) (3) When anti-passback is violated, ANTI_PASSBACK_FAIL | (BS_DEVICE_ALL << 16)
outputDeviceID	Specifies the device which will generate the output signal. ● BS_DEVICE_HOST ● BS_DEVICE_SLAVE ● 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	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.
high	
low	
count	
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	These three fields define the waveform of output signal.
low	
count	

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

Example

```
// (1) Generate alarm signal to the relay 0 of Secure I/O 0 when
//     anti-pasback 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::ANTI_PASSBACK_FAIL |
(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

Entrance limitation can be applied to a single device. See **BSEntrnaceLimitationZoneProperty** for details.

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;
    int numOfEntranceLimit;
    int maxEntry[4];
    unsigned entryLimitInterval[4];
    int defaultAccessGroup;
} BSEntranceLimit;
```

The key fields and their available options are as follows;

Fields	Options
minEntryInterval	See **BSEntrnaceLimitationZoneProperty**.
numOfEntranceLimit	
maxEntry	
entryLimitInterval	
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.

Return Values

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

Compatibility

BioStation

BS_WriteConfig / BS_ReadConfig

You can write/read the configuration of a BioEntry Plus using **BS_WriteConfig / BS_ReadConfig**.

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.

BEPLUS_CONFIG – BEConfigData

BEPLUS_CONFIG_SYS_INFO – BESysInfoData

Please note that BEPLUS_CONFIG_SYS_INFO is 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;

```
typedef struct {
     unsigned cardID;
     unsigned char customID;
     unsigned char commandType;
     unsigned char needAdminFinger;
} BECommandCard;

typedef struct {
     // header
     unsigned magicNo;
     int version;
     unsigned timestamp;
     unsigned checksum;
     // operation mode
```

```
    int opMode[4];
    int opModeSchedule[4];
    unsigned char opDualMode[4];
    // ip
    bool useDHCP;
    unsigned ipAddr;
    unsigned gateway;
    unsigned subnetMask;
    unsigned serverIpAddr;
    int port;
    bool useServer;
    bool synchTime;
    // fingerprint
    int securityLevel;
    int fastMode;
    int timeout; // 1 ~ 20 sec
    int matchTimeout; // Infinite(0) ~ 10 sec
    // I/O
    BSInputConfig inputConfig;
    BSOutputConfig outputConfig;
    BSDoorConfig doorConfig;
    // serial
    int serialMode;
    int serialBaudrate;
    unsigned char secureIO; // 0x01 - Secure I/O 0, 0x02, 0x04, 0x08
    unsigned slaveID; // 0 for no slave
    // entrance limit
    int minEntryInterval; // 0 for no limit
    int numOfEntranceLimit; // MAX 4
    int maxEntry[4]; // 0 (no limit) ~ 16
    unsigned entryLimitInterval[4];
    // command card
    int numOfCommandCard;
    BECommandCard commandCard[MAX_COMMAND_CARD];
    // tna
    int tnaMode;
    int autoInSchedule;
    int autoOutSchedule;
    // user
    int defaultAG;
    // wiegand
    bool useWiegandOutput;
    BSWiegandConfig wiegandConfig;
    // LED/Buzzer
    BELEDBuzzerConfig ledBuzzerConfig;
} BEConfigData;
```

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

The key fields and their available options are as follows;

BECommandCard BioEntry Plus supports command cards with which you can enroll/delete users directly at the device.	
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.
Operation Mode	
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.
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, the device will connect to the server with serverIPAddr and port. If false, it will 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.
I/O	
inputConfig	See BSWriteInputConfig.
outputConfig	See BSWriteOutputConfig.
doorConfig	See BSWriteDoorConfig.
Serial	
serialMode	RS485 connection of a BioEntry Plus can be used as one of the followings; ● SERIAL_DISABLED: not used. ● SERIAL_IO_HOST: acts as a host device and controls all the I/O operations of Secure I/O devices and a slave device connected to the same RS485 connection. secureIO and slaveID should be set properly in this mode. ● SERIAL_IO_SLAVE: acts as a slave device and defer all I/O operations to the host device. ● SERIAL_PC: used as a communication channel to host PC.
serialBaudrate	Specifies the baudrate of RS485 connection when seirlaMode 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.
slaveID	The ID of the slave device. If it is 0, it means that no slave device is connected.
Entrance Limitation	
minEntryInterval	Entrance limitation can be applied to a single device. See **BSEntrnaceLimitationZoneProperty** for details.
numOfEntranceLimit	
maxEntry	
entryLimitInterval	
Command Card	
numOfCommandCard	The number of command cards enrolled to the device.
commandCard	See **BECommandCard**.
TNA	
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.
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.
wiegandConfig	See BS_WriteWiegandConfig.

Return Values

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

Compatibility

BioEntry Plus

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

2.8. Access Control API(Deprecated)[4]

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

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

[4] Since BioStation firmware V1.4 and BioEntry Plus firmware V1.0, new Access Control APIs are provided. Unless compatibility with old firmware is a major concern, you would be well advised to use new APIs.

BS_AddTimeSchedule

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

```
#define BS_TIMECODE_PER_DAY       5

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

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

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

BS_RET_CODE BS_AddTimeSchedule(int handle, BSTimeSchedule* schedule)

Parameters

handle

Handle of the communication channel.

schedule

Pointer to the time schedule to be added.

Return Values

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

Compatibility

BioStation

Example

```
BSTimeSchedule timeSchedule;

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

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


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

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

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

// …

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

BS_GetAllTimeSchedule

Reads all the registered time schedules.

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

Parameters

handle

Handle of the communication channel.

numOfSchedule

Pointer to the number of enrolled schedules.

schedule

Pointer to the time schedule array to be read.

Return Values

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

Compatibility

BioStation

BS_DeleteTimeSchedule

Deletes the specified time schedule.

BS_RET_CODE BS_DeleteTimeSchedule(int handle, int ID)

Parameters

handle

Handle of the communication channel.

ID

ID of the time schedule.

Return Values

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

Compatibility

BioStation

BS_DeleteAllTimeSchedule

Deletes all the time schedules stored in a BioStation terminal.

BS_RET_CODE BS_DeleteAllTimeSchedule(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

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

Compatibility

BioStation

BS_AddHoliday

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

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

BS_RET_CODE BS_AddHoliday(int handle, BSHoliday* holiday)

Parameters

handle

Handle of the communication channel.

holiday

Pointer to the holiday schedule to be added.

Return Values

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

Compatibility

BioStation

Example

```
BSHoliday holiday;

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

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

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

// Mar. 5 is holiday
```

```
holiday.holiday[1] = (3 << 8) | 5;

// …

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

strcpy( holiday.name, “Holiday 1” );

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

BS_GetAllHoliday

Reads all the registered holiday schedules.

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

Parameters

handle

Handle of the communication channel.

numOfHoliday

Pointer to the number of enrolled holiday schedules.

holiday

Pointer to the holiday schedules to be read.

Return Values

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

Compatibility

BioStation

BS_DeleteHoliday

Deletes the specified holiday schedule.

BS_RET_CODE BS_DeleteHoliday(int handle, int ID)

Parameters

handle

Handle of the communication channel.

ID

ID of the holiday schedule.

Return Values

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

Compatibility

BioStation

BS_DeleteAllHoliday

Deletes all the holiday schedules stored in a BioStation terminal.

BS_RET_CODE BS_DeleteAllHoliday(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

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

Compatibility

BioStation

BS_AddAccessGroup

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

```
#define BS_SCHEDULE_PER_GROUP      16

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

BS_RET_CODE BS_AddAccessGroup(int handle, BSAccessGroup* group)

Parameters

handle

Handle of the communication channel.

group

Pointer to the access group to be added.

Return Values

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

Compatibility

BioStation

BS_GetAllAccessGroup

Reads all the registered access groups.

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

Parameters

handle

Handle of the communication channel.

numOfAccessGroup

Pointer to the number of enrolled access groups.

group

Pointer to the access groups to be read.

Return Values

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

Compatibility

BioStation

BS_DeleteAccessGroup

Deletes the specified access group.

BS_RET_CODE BS_DeleteAccessGroup(int handle, int ID)

Parameters

handle

Handle of the communication channel.

ID

ID of the access group.

Return Values

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

Compatibility

BioStation

BS_DeleteAllAccessGroup

Deletes all the access groups stored in a BioStation terminal.

BS_RET_CODE BS_DeleteAllAccessGroup(int handle)

Parameters

handle

Handle of the communication channel.

Return Values

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

Compatibility

BioStation

BS_RelayControl

Controls the relay of a BioStation.

BS_RET_CODE BS_RelayControl(int handle, bool onoff)

Parameters

handle

Handle of the communication channel.

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

2.9. Extended Access Control API

These APIs provide access control features such as time schedule and access group. Please note that you cannot mix old Access Control APIs with these ones. Unless compatibility with older BioStation firmware is a major concern, these APIs would be better for most applications.

- 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

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

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

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

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

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

Example

```
BSHolidayEx holiday;

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

holiday.holidayID = 1;
holiday.numOfHoliday = 2;

// 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

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

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

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

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

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

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

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

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

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_HOST 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

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

2.10. 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_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/BioEntry Plus

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

BioStation/BioEntry Plus

BS_SetKey

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

void BS_SetKey(unsigned char *key)

Parameters

key

32 byte – 256bit – encryption key.

Return Values

None

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