Table of Contents

Configuration API

The following APIs are used to read and write system configuration information.

- BS2_ResetConfig: Initializes the device's configurations.
- BS2_ResetConfigExceptNetInfo: Initializes the setting information of the device. (Excluding network settings)
- BS2_GetConfig: Retrieves configuration blob from the device.
- BS2_SetConfig: Stores configuration blob on the device.
- BS2_GetFactoryConfig: Retrieves software version and hardware settings from the device.
- BS2_GetSystemConfig: Retrieves system settings from the device.
- BS2_SetSystemConfig: Stores system settings on the device.
- BS2_GetAuthConfig: Retrieves authentication settings from the device.
- BS2_SetAuthConfig: Stores authentication settings on the device.
- BS2_GetStatusConfig: Retrieves LED and buzzer settings from the device.
- BS2_SetStatusConfig: Stores LED and buzzer settings on the device.
- BS2_GetDisplayConfig: Retrieves sound and UI settings from the device.
- BS2_SetDisplayConfig: Stores sound and UI settings on the device.
- BS2_GetIPConfig: Retrieves IP settings from the device.
- BS2_GetIPConfigViaUDP: Retrieves IP settings from the device via the UDP broadcasting.
- BS2_SetIPConfig: Stores IP settings on the device.
- BS2_SetIPConfigViaUDP: Stores IP settings on the device via the UDP broadcasting.
- BS2_GetIPConfigExt: Retrieves DNS and server URL settings from the device.
- BS2_SetIPConfigExt: Stores DNS and server URL settings on the device.
- BS2_GetTNAConfig: Retrieves time and attendance settings from the device.
- BS2_SetTNAConfig: Stores time and attendance settings on the device.
- BS2_GetCardConfig: Retrieves smart card settings from the device.
- BS2_SetCardConfig: Stores smart card settings on the device.
- BS2_GetFingerprintConfig: Retrieves fingerprint matching settings from the device.
- BS2_SetFingerprintConfig: Stores fingerprint matching settings on the device.
- BS2_GetRS485Config: Retrieves RS-485 network settings from the device.
- BS2_SetRS485Config: Stores RS-485 network settings on the device.
- BS2_GetWiegandConfig: Retrieves Wiegand I/O settings from the device.
- BS2_SetWiegandConfig: Stores Wiegand I/O settings on the device.
- BS2_GetWiegandDeviceConfig: Retrieves Wiegand device settings from the device.
- BS2_SetWiegandDeviceConfig: Stores Wiegand device settings on the device.
- BS2_GetInputConfig: Retrieves supervised input port settings from the device.
- BS2_SetInputConfig: Stores supervised input port settings on the device.
- BS2_GetWlanConfig: Retrieves wireless LAN settings from the device.
- BS2_SetWlanConfig: Stores wireless LAN settings on the device.
- BS2_GetTriggerActionConfig: Retrieves trigger and action settings from the device.
- BS2_SetTriggerActionConfig: Stores trigger and action settings on the device.
- BS2_GetEventConfig: Retrieves image log filter settings from the device.
- BS2_SetEventConfig: Stores image log filter settings on the device.
- BS2_GetWiegandMultiConfig: Retrieves Multi-Wiegand settings from the device.
- BS2_SetWiegandMultiConfig: Stores Multi-Wiegand settings on the device.
- BS2_GetCard1xConfig: Retrieves v1 Template on Card settings from the device.
- BS2_SetCard1xConfig: Stores v1 Template on Card settings on the device.
- BS2_GetSystemExtConfig: Retrieves Master and slave device encryption settings from the

device.
- BS2_SetSystemExtConfig: Stores Master and slave device encryption settings on the device
- BS2_GetVoipConfig: Retrieves VoIP settings from the device.
- BS2_SetVoipConfig: Stores VoIP settings on the device.
- BS2_GetFaceConfig: Retrieves face settings from the device.
- BS2_SetFaceConfig: Stores face settings on the device.
- BS2_GetRS485ConfigEx: In case of Corestation, retrieves RS-485 network settings from the device.
- BS2_SetRS485ConfigEx: In case of CoreStation, stores RS-485 network settings on the device.
- BS2_GetCardConfigEx: Retrieves iClass SEOS card settings from the device.
- BS2_SetCardConfigEx: Stores iClass SEOS card settings on the device.
- BS2_GetDstConfig: Retrieves the device DST information.
- BS2_SetDstConfig: Stores the device DST information.
- BS2_GetSupportedConfigMask: Retrieves supported configuration of the device.
- BS2_GetIPConfigViaUDPEx: [+ 2.6.3] Retrieves IP configuration through UDP broadcast with host IP.
- BS2_SetIPConfigViaUDPEx: [+ 2.6.3] Stores IP configuration through UDP broadcast with host IP.
- BS2_GetIPV6Config: [+ 2.6.3] Retrieves IPv6 configuration information.
- BS2_SetIPV6Config: [+ 2.6.3] Stores IPv6 configuration information.
- BS2_GetIPV6ConfigViaUDP: [+ 2.6.3] Retrieves IPv6 configuration through UDP broadcast.
- BS2_SetIPV6ConfigViaUDP: [+ 2.6.3] Stores IPv6 configuration through UDP broadcast.
- BS2_GetIPV6ConfigViaUDPEx: [+ 2.6.3] Retrieves IPv6 configuration through UDP broadcast with host IP.
- BS2_SetIPV6ConfigViaUDPEx: [+ 2.6.3] Stores IPv6 configuration through UDP broadcast with host IP.
- BS2_GetDesFireCardConfigEx: [+ 2.6.4] Retrieves DesFire advanced configuration from the device.
- BS2_SetDesFireCardConfigEx: [+ 2.6.4] Sets DesFire advanced configuration in the device.
- BS2_GetAuthConfigExt: [+ 2.7.1] `FaceStation F2` Retrieves authentication settings from the device.
- BS2_SetAuthConfigExt: [+ 2.7.1] `FaceStation F2` Stores authentication settings from the device.
- BS2_GetFaceConfigExt: [+ 2.7.1] `FaceStation F2, FaceStation2` Retrieves configuration of thermal camera and mask detection.
- BS2_SetFaceConfigExt: [+ 2.7.1] `FaceStation F2, FaceStation2` Stores configuration of thermal camera and mask detection.
- BS2_GetThermalCameraConfig: [+ 2.7.1] `FaceStation F2, FaceStation2` Retrieves configuration of thermal camera.
- BS2_SetThermalCameraConfig: [+ 2.7.1] `FaceStation F2, FaceStation2` Stores configuration of thermal camera.
- BS2_GetBarcodeConfig: [+ 2.8] `X-Station 2` Retrieves configuration of Barcode.
- BS2_SetBarcodeConfig: [+ 2.8] `X-Station 2` Stores configuration of Barcode.
- BS2_GetInputConfigEx: [+ 2.8.1] `IM-120` Retrieves Expanded Configuration related to the Input.
- BS2_SetInputConfigEx: [+ 2.8.1] `IM-120` Retrieves Expanded Configuration related to the Input.
- BS2_GetRelayActionConfig: [+ 2.8.1] `IM-120` Retrieves Configuration related to the RelayAction.
- BS2_SetRelayActionConfig: [+ 2.8.1] `IM-120` Retrieves Configuration related to the RelayAction.
- BS2_GetLicenseConfig: [+ 2.9.1] Gets the device license activation information.

- BS2_GetOsdpStandardConfig: [+ 2.9.1] Get the device's OSDP setting information.
- BS2_GetOsdpStandardActionConfig: [+ 2.9.1] Get the LED/buzzer settings for each action of the OSDP device.
- BS2_SetOsdpStandardActionConfig: [+ 2.9.1] Specifies the LED/buzzer setting for each action of the OSDP device.
- BS2_GetCustomCardConfig: [+ 2.9.4] Retrieves Custom smart card settings.
- BS2_SetCustomCardConfig: [+ 2.9.4] Stores Custom smart card settings.

Structure

BS2FactoryConfig

```
typedef struct {
    uint8_t major;
    uint8_t minor;
    uint8_t ext;
    uint8_t reserved[1];
} Version;

typedef struct {
    uint32_t deviceID;
    uint8_t macAddr[BS2_MAC_ADDR_LEN];
    uint8_t reserved[2];
    char modelName[BS2_MODEL_NAME_LEN];
    Version boardVer;
    Version kernelVer;
    Version bscoreVer;
    Version firmwareVer;
    char kernelRev[BS2_KERNEL_REV_LEN];
    char bscoreRev[BS2_BSCORE_REV_LEN];
    char firmwareRev[BS2_FIRMWARE_REV_LEN];
    uint8_t reserved2[32];
} BS2FactoryConfig;
```

1. *deviceID*
Device identifier.

2. *macAddr*
MAC address of the network adaptor.

3. *reserved*
Reserved space.

4. *modelName*
Model name.

5. *boardVer*

Hardware version.

6. *kernelVer*
Kernel version.

7. *bscoreVer*
BioStar Core version.

8. *firmwareVer*
Firmware version.

9. *kernelRev*
Kernel revision information.

10. *bscoreRev*
BioStar Core revision information.

11. *firmwareRev*
Firmware revision information.

12. *reserved2*
Reserved space.

BS2SystemConfig

```
typedef struct {
    uint8_t notUsed[16 * 16 * 3];
    int32_t timezone;
    uint8_t syncTime;
    uint8_t serverSync;
    uint8_t deviceLocked;
    uint8_t useInterphone;
    uint8_t useUSBConnection;
    uint8_t keyEncrypted;
    uint8_t useJobCode;
    uint8_t useAlphanumericID;
    uint32_t cameraFrequency;
    bool secureTamper;
    bool reserved0;     // (write protected)
    uint8_t reserved[2];
    uint32_t useCardOperationMask;
    uint8_t reserved2[16];
} BS2SystemConfig;
```

1. *notUsed*
Not used.

2. *timezone*
Represents standard time zone in seconds.

3. *syncTime*
Stores when synchronization with BioStar has occurred.

4. *serverSync*
Reserved variable.

5. *deviceLocked*
Indicates the current locked state of the device. (Read only filed)

6. *useInterphone*
Decides whether to use intercom.

7. *useUSBConnection*
This is not used anymore. (The device automatically detects USB connection.)

8. *keyEncrypted*
Decides whether to use OSDP secure key.

9. *useJobCode*
Decides whether to use job codes.

10. *useAlphanumericID*
Decides whether to use alphanumeric ID.

11. *cameraFrequency*
Frequency of the camera.

Value	Description
1	50Hz
2	60Hz

12. *secureTamper
Flag to determine whether to use a security tamper.
When Tamper on, the following data is deleted from the device. (User, log, data encryption key, SSL certificate)

13. *reserved0*
Reserved space.

14. *reserved*
Reserved space.

15. *useCardOperationMask*
[+ V2.6.4] Provides a card selective option not to read all kinds of cards from the device.
You can select multiple cards using MASK. The user can select or deselect of a specific card reading option using this option.
However, it can be applied to the card types the device supporting. If you add a card type which isn't supported from the device would be ignored.
Also, the required card type MASK should be combined with `CARD_OPERATION_USE`.
For example, useCardOperationMask needs to be configured `x0x80000001` when EM card is selected

only.

Value	Description
0xFFFFFFFF	CARD_OPERATION_MASK_DEFAULT
0x80000000	CARD_OPERATION_USE
0x00000000	CARD_OPERATION_MASK_NONE
0x00000800	CARD_OPERATION_MASK_CUSTOM_DESFIRE_EV1
0x00000400	CARD_OPERATION_MASK_CUSTOM_CLASSIC_PLUS
0x00000200	CARD_OPERATION_MASK_BLE
0x00000100	CARD_OPERATION_MASK_NFC
0x00000080	CARD_OPERATION_MASK_SEOS
0x00000040	CARD_OPERATION_MASK_SR_SE
0x00000020	CARD_OPERATION_MASK_DESFIRE_EV1
0x00000010	CARD_OPERATION_MASK_CLASSIC_PLUS
0x00000008	CARD_OPERATION_MASK_ICLASS
0x00000004	CARD_OPERATION_MASK_MIFARE_FELICA
0x00000002	CARD_OPERATION_MASK_HIDPROX
0x00000001	CARD_OPERATION_MASK_EM

16. *reserved2*
Reserved space.

BS2AuthConfig

```
typedef struct {
    uint32_t authSchedule[BS2_NUM_OF_AUTH_MODE];
    uint8_t useGlobalAPB;
    uint8_t globalAPBFailAction;
    uint8_t useGroupMatching;
    uint8_t reserved
    uint8_t reserved[28];
    uint8_t usePrivateAuth;
    uint8_t faceDetectionLevel;
    uint8_t useServerMatching;
    uint8_t useFullAccess;
    uint8_t matchTimeout;
    uint8_t authTimeout;
    uint8_t numOperators;
    uint8_t reserved2[1];
    struct {
        char userID[BS2_USER_ID_SIZE];
        uint8_t level;
        uint8_t reserved[3];
    } operators[BS2_MAX_OPERATORS];
} BS2AuthConfig;
```

1. *authSchedule*
Stores schedules for different types of authentication modes.

It has the following meanings in the value of the array.
If the value in the array is greater than 0, the corresponding authentication mode is enabled.
Biometric information in the descriptions below refers to the fingerprint or face depending on the device.

Value	Code	Description
0	BS2_AUTH_MODE_BIOMETRIC_ONLY	Biometric only
1	BS2_AUTH_MODE_BIOMETRIC_PIN	Biometric + PIN
2	BS2_AUTH_MODE_CARD_ONLY	Card only
3	BS2_AUTH_MODE_CARD_BIOMETRIC	Card + Biometric
4	BS2_AUTH_MODE_CARD_PIN	Card + PIN
5	BS2_AUTH_MODE_CARD_BIOMETRIC_OR_PIN	Card + Biometric or PIN
6	BS2_AUTH_MODE_CARD_BIOMETRIC_PIN	Card + Biometric + PIN
7	BS2_AUTH_MODE_ID_BIOMETRIC	ID + Biometric
8	BS2_AUTH_MODE_ID_PIN	ID + PIN
9	BS2_AUTH_MODE_ID_BIOMETRIC_OR_PIN	ID + Biometric or PIN
10	BS2_AUTH_MODE_ID_BIOMETRIC_PIN	ID + Biometric + PIN

2. *useGlobalAPB*
Decides whether to enable global APB zone.

3. *globalAPBFailAction*
Default action that will be executed when the BioStar application cannot determine if the authentication has violated global APB rules.

Value	Description
0	Not use
1	Operate as soft APB
2	Operate as hard APB

4. *useGroupMatching*
Decides whether to use face group matching.

5. *reserved*
Reserved space.

6. *usePrivateAuth*
Decides whether to use private authentication mode.

7. *faceDetectionLevel*

Level of face detection in user authentication. If the detected face level is lower than the configuration, it will be processed as authentication fail.
When set, the camera view according to Normal/Strict is displayed, access is denied if the device doesn't recognize facial image through image log. Default is 0.

Value	Description
0	Face detection not used
1	Normal mode
2	Strict mode

Only valid for A2. Not used with FaceStation2 or FaceLite.

8. *useServerMatching*
Decides whether to perform fingerprint/face matchings on the server.

9. *useFullAccess*
Decides whether to allow full access to all authenticated users regardless to the access group rules.

10. *matchTimeout*
Timeout in seconds for fingerprint/face matching.

11. *authTimeout*
Timeout in seconds for the user authentication response.

12. *numOperators*
The number of operators defining user privileges.

13. *reserved2*
Reserved space.

14. *userID*
User identifier.

15. *level*
Specifies the privilege of the user when accessing to the device's menu.

Value	Description
0	No privilege
1	Administrative privilege
2	Privilege to change the system settings
3	Privilege to change user information

CAUTION
To add an operator, the ***numOperators*** field needs to be set equivalent to the number of operators that will be added.

16. *reserved*
Reserved space.

BS2StatusConfig

```
typedef struct {
    struct {
        uint8_t enabled;
        uint8_t reserved[1];
        uint16_t count;
        BS2LedSignal signal[BS2_LED_SIGNAL_NUM];
    } led[BS2_DEVICE_STATUS_NUM];
    uint8_t reserved1[32];
    struct {
        uint8_t enabled;
        uint8_t reserved[1];
        uint16_t count;
        BS2BuzzerSignal signal[BS2_BUZZER_SIGNAL_NUM];
    } buzzer[BS2_DEVICE_STATUS_NUM];
    uint8_t configSyncRequired;
    uint8_t reserved2[31];
} BS2StatusConfig;
```

1. *enabled*
Decides whether to use the LED.

2. *reserved*
Reserved space.

3. *count*
Number of LED signal execution count. When it is set as 0, repeats infinitely.

4. *signal*
List of LED signal patterns, which can be configured up to 3 patterns.

5. *reserved1*
Reserved space.

6. *enabled*
Decides whether to use the buzzer.

7. *reserved*
Reserved space.

8. *count*
Number of buzzer signal execution count. When it is set as 0, repeats infinitely.

9. *signal*

List of buzzer signal patterns, which can be configured up to 3 patterns.

10. *configSyncRequired*
If the device's configuration has been modified, this value will be set to true.

11. *reserved2*
Reserved space.

BS2DisplayConfig

```
typedef struct {
    uint32_t language;
    uint8_t background;
    uint8_t volume;
    uint8_t bgTheme;
    uint8_t dateFormat;
    uint16_t menuTimeout;
    uint16_t msgTimeout;
    uint16_t backlightTimeout;
    uint8_t displayDateTime;
    uint8_t useVoice;
    uint8_t timeFormat;
    uint8_t homeFormation;
    BS2_BOOL useUserPhrase;
    BS2_BOOL queryUserPhrase;
    uint8_t shortcutHome[BS2_MAX_SHORTCUT_HOME];
    uint8_t tnaIcon[16];
    uint8_t useScreenSaver;
    uint8_t reserved1[31];
} BS2DisplayConfig;
```

1. *language*
Language code.

Value	Description
0	Korean
1	English
2	Custom

2. *background*
Background image type.

Value	Description
0	LOGO
1	NOTICE
2	SLIDE
3	PDF

3. *volume*
The volume of sound. The volume can be set from 0 to 100. 0 means no sound.

4. *bgTheme*
Theme type.

Value	Description
0	Logo image
1	Notice
2	Slide show
3	PDF

5. *dateFormat*
Date format.

Value	Description
0	YYYY/MM/DD
1	MM/DD/YYYY
2	DD/MM/YYYY

6. *menuTimeout*
Timeout in seconds for lock screen when the user is inactive. The timeout can be set from 0 to 255 seconds. 0 means no lock screen.

Value	Description
0	No timeout.
10	Menu timeout 10 sec.
20	Menu timeout 20 sec. (Default)
30	Menu timeout 30 sec.
40	Menu timeout 40 sec.
50	Menu timeout 50 sec.
60	Menu timeout 60 sec.

7. *msgTimeout*
Message timeout in milliseconds. The timeout can be set from 500 to 5000 milliseconds.

Value	Description
500	Message timeout 500 msec.
1000	Message timeout 1 sec.
2000	Message timeout 2 sec. (Default)
3000	Message timeout 3 sec.
4000	Message timeout 4 sec.
5000	Message timeout 5 sec.

8. *backlightTimeout*
Backlight timeout in seconds.

Value	Description
0	Backlight timeout 0 sec.

Value	Description
10	Backlight timeout 10 sec.
20	Backlight timeout 20 sec. (Default)
30	Backlight timeout 30 sec.
40	Backlight timeout 40 sec.
50	Backlight timeout 50 sec.
60	Backlight timeout 60 sec.

9. *displayDateTime*
Decides whether to display clock on screen.

10. *useVoice*
Decides whether to use voice instruction.

11. *timeFormat*
Time format.

Value	Description
0	12 hour
1	24 hour

However, Linux OS devices like BioStation 2, BioStation L2, BioLite N2 and FaceLite have opposite settings.(0 = 24 hour / 1 = 12 hour)

12. *homeFormation*
Home screen settings(Currently, not used).

Value	Description
1	Interphone
2	Shortcut 1
3	Shortcut 2
4	Shortcut 3
5	Shortcut 4

13. *useUserPhrase*
Flag that determines whether to use the user phrase feature.

13. *queryUserPhrase*
If set true, asks for the user phrase to the server.

15. *shortcutHome*
Home screen layout(Going to apply later, not used currently).

16. *tnaIcon*
Icon displayed on the device corresponding TNA key.

17. *useScreenSaver*
`FaceStation 2`, `FaceStation F2` If set true, you can activate the screensaver.

18. *reserved1*
Reserved space.

BS2IpConfig

```
typedef struct {
    uint8_t connectionMode;
    uint8_t useDHCP;
    uint8_t useDNS;
    uint8_t reserved[1];
    char ipAddress[BS2_IPV4_ADDR_SIZE];
    char gateway[BS2_IPV4_ADDR_SIZE];
    char subnetMask[BS2_IPV4_ADDR_SIZE];
    char serverAddr[BS2_IPV4_ADDR_SIZE];
    uint16_t port;
    uint16_t serverPort;
    uint16_t mtuSize;
    uint8_t baseband;
    uint8_t reserved2[1];
    uint16_t sslServerPort;
    uint8_t reserved3[30];
} BS2IpConfig;
```

1. *connectionMode*
Represents the connection mode between BioStar and devices. There are two modes depending on who initiates the connection: Direct mode (0x0) and Server mode (0x1). The Direct mode means that BioStar initiates the connection to the devices and the Server mode means that the devices initiate the connection to the server. The default connection mode of a device is the Direct mode.

2. *useDHCP*
Decides whether to use DHCP.

3. *useDNS*
Decides whether to use server address or server URL.

4. *reserved*
Reserved space.

5. *ipAddress*
IP address assigned to the device.

6. *gateway*
IP address of the gateway.

7. *subnetMask*
Subnet mask of the device.

8. *serverAddr*
IP address of BioStar. Used only in the server mode.

9. *port*
Port number of the device.

10. *serverPort*
Port number of BioStar. Used only in the server mode.

11. *mtuSize*
MTU[1] size for the TCP/IP communication.

12. *baseband*
Bandwidth of the device. The value can be set to 10MB/S or 100 MB/S.

13. *reserved2*
Reserved space.

14. *sslServerPort*
Used when the connectionMode is set as server SSL mode, which is the port of the SDK application.

15. *reserved3*
Reserved space.

BS2IpConfigExt

```
typedef struct {
    char dnsAddr[BS2_IPV4_ADDR_SIZE];
    char serverUrl[BS2_URL_SIZE];
    uint8_t reserved[32];
} BS2IpConfigExt;
```

1. *dnsAddr*
DNS server address.

2. *serverUrl*
URL of the BioStar application server. The maximum length is 256 characters.

3. *reserved*
Reserved space.

BS2TNAConfig

```
typedef struct {
    uint8_t tnaMode;
    uint8_t tnaKey;
    uint8_t tnaRequired;
    uint8_t reserved[1];
    uint32_t tnaSchedule[BS2_MAX_TNA_KEY];
    uint8_t unused[BS2_MAX_TNA_KEY];
} BS2TNAInfo;
```

```
typedef struct {
    char tnaLabel[BS2_MAX_TNA_KEY][BS2_MAX_TNA_LABEL_LEN];
    uint8_t unused[BS2_MAX_TNA_KEY];
} BS2TNAExtInfo;

typedef struct {
    BS2TNAInfo tnaInfo;
    BS2TNAExtInfo tnaExtInfo;
    uint8_t reserved2[32];
} BS2TNAConfig;
```

1. *tnaMode*
Time and attendance management mode.

Value	Description
0	Not used
1	Applying time and attendance code according to a user
2	Applying time and attendance code according to a T&A schedule
3	Applying the time and attendance code that the previous user has selected
4	Using a fixed time and attendance code

2. *tnaKey*
Represents time and attendance code. This is mapped to a key on the device.

Device Type	T&A Code	Mapped Key	Value
BioStation 2	BS2_TNA_UNSPECIFIED	(N/A)	0
	BS2_TNA_KEY_1	F1	1
	BS2_TNA_KEY_2	F2	2
	BS2_TNA_KEY_3	F3	3
	BS2_TNA_KEY_4	F4	4
	BS2_TNA_KEY_5	1	5
	BS2_TNA_KEY_6	2	6
	BS2_TNA_KEY_7	3	7
	BS2_TNA_KEY_8	4	8
	BS2_TNA_KEY_9	5	9
	BS2_TNA_KEY_10	6	10
	BS2_TNA_KEY_11	7	11
	BS2_TNA_KEY_12	8	12
	BS2_TNA_KEY_13	9	13
	BS2_TNA_KEY_14	Call	14
	BS2_TNA_KEY_15	0	15
	BS2_TNA_KEY_16	Esc	16

3. *tnaRequired*
Decides whether to time and attendance code entry is mandatory when the time and attendance management mode is set to 1.

4. *reserved*
Reserved space.

5. *tnaSchedule*
Specifies a schedule for a time and attendance code.

6. *unused*
Not used.

7. *tnaLabel*
A label that shows the meaning of the time and attendance code.

8. *unused*
Not used.

BS2CardConfig

```
typedef struct {
    uint8_t primaryKey[6];
    uint8_t reserved1[2];
    uint8_t secondaryKey[6];
    uint8_t reserved2[2];
    uint16_t startBlockIndex;
    uint8_t reserved[6];
} BS2MifareCard;

typedef struct {
    uint8_t primaryKey[8];
    uint8_t secondaryKey[8];
    uint16_t startBlockIndex;
    uint8_t reserved[6];
} BS2IClassCard;

typedef struct {
    uint8_t primaryKey[16];
    uint8_t secondaryKey[16];
    uint8_t appID[3];
    uint8_t fileID;
    uint8_t encryptionType;
    uint8_t operationMode;
    uint8_t reserved[2];
} BS2DesFireCard;

typedef struct {
    uint8_t byteOrder;
    uint8_t useWiegandFormat;
```

```
    uint8_t dataType;
    uint8_t useSecondaryKey;
    BS2MifareCard mifare;
    BS2IClassCard iclass;
    BS2DesFireCard desfire;
    uint8_t formatID;
    uint8_t cipher;
    uint8_t reserved[24];
} BS2CardConfig;
```

1. *primaryKey*
Primary encryption key to access the Mifare card information.

2. *reserved1*
Reserved space.

3. *secondaryKey*
Secondary encryption key to access the Mifare card information.

4. *reserved2*
Reserved space.

5. *startBlockIndex*
Start block index on the Mifare data storage.

6. *reserved*
Reserved space.

7. *primaryKey*
Primary encryption key to access the iClass card information.

8. *secondaryKey*
Secondary encryption key to access the iClass card information..

9. *startBlockIndex*
Start block index on the Mifare data storage.

10. *reserved*
Reserved space.

11. *primaryKey*
Primary encryption key to access the DesFire card information.

12. *secondaryKey*
Secondary encryption key to access the Desfire card information.

13. *appID*
Application Id that is stored inside the DesFire card for user authentication.

14. *fileID*
File ID that is stored inside the DesFire card, which will be used by the application to read and write

data.

15. *encryptionType*
Type of data encryption.

Value	Description
0	DES/3DES
1	AES

16. *operationMode*
Operation mode. (operationMode will be supported soon.)

Value	Description
0	Lagacy mode (Using PICC master key)
1	New mode (Using App master key)

17. *reserved*
Reserved space.

18. *byteOrder*
Order of how the byte of the card is stored. When it is set as 0, will function as MSB[2]. When it is set as 1, will function as LSB[3].

19. *useWiegandFormat*
Decides whether to use Wiegand format.

20. *dataType*
Type of card data.

Value	Description
0	Binary
1	ASCII
2	UTF16
3	BCD

21. *useSecondaryKey*
Decides whether to use the secondary encryption key.

22. *formatID*
ID that is used when the card configuration needs to be managed from the database on the BioStar application.

23. *cipher*
Activates 'Keypad card ID' option.
Default value is 0, it is only valid for Gangbox Keypad type of Xpass 2, XPass D2.

Value	Description
0	Deactivate
1	Activate

24. *reserved5*
Reserved space.

BS2FingerprintConfig

```
typedef struct {
    uint8_t      securityLevel;
    uint8_t      fastMode;
    uint8_t      sensitivity;
    uint8_t      sensorMode;
    uint16_t     templateFormat;
    uint16_t     scanTimeout;
    uint8_t      successiveScan;
    uint8_t      advancedEnrollment;
    uint8_t      showImage;
    uint8_t      lfdLevel;
    bool         checkDuplicate;

    uint8_t      reserved3[31];
} BS2FingerprintConfig;
```

1. *securityLevel*
Fingerprint authentication security level. This is used across the system.

Value	Description
0	Basic
1	Highly secure
2	The most highly secure

2. *fastMode*
Fingerprint matching speed.

Value	Description
0	Automatic
1	Basic
2	High
3	Very High

3. *sensitivity*
Sensitivity of the fingerprint sensor.

Value	Description
0	Lowest
1	Level 1
2	Level 2
3	Level 3
4	Level 4
5	Level 5

Value	Description
6	Level 6
7	Highest

4. *sensorMode*
Decides the sensor mode. 0 means the sensor is always on. 1 means the sensor is activated when the finger is near the sensor.

5. *templateFormat*
Fingerprint template type.

Value	Description
0	Suprema
1	ISO
2	ANSI

6. *scanTimeout*
Fingerprint scanning timeout in seconds. The default is 10 seconds.

7. *successiveScan*
Not Used.

8. *advancedEnrollment*
Decides whether to utilize fingerprint quality information. If the option is disabled, the BS_SDK_ERROR_CAPTURE_LOW_QUALITY / BS_SDK_ERROR_EXTRACTION_LOW_QUALITY error codes are not returned even if the quality of fingerprint image acquired is low.

9. *showImage*
Decides whether to display scanned fingerprint image on the screen.

10. *lfdLevel*
Configuration for the LFD(Live Fingerprint Detection - fake fingerprint detection) sensitivity.

Value	Description
0	Not Use
1	Strict
2	More Strict
3	Most Strict

11. *checkDuplicate*
[+ V2.6.4] If set to true, it will determine if the fingerprint is a duplicate.

12. *reserved3*
Reserved space.

BS2Rs485Config

```
typedef struct {
```

```
    uint8_t supportConfig;
    uint8_t useExceptionCode;
    uint8_t exceptionCode[BS2_RS485_MAX_FAIL_CODE_LEN];
    uint8_t outputFormat;
    uint8_t osdpID;
    uint8_t reserved[4];
} BS2IntelligentPDInfo;   //Added 2.8.0 for Intelligent Slave Feature

typedef struct {
    uint32_t baudRate;
    uint8_t channelIndex;
    uint8_t useRegistance;
    uint8_t numOfDevices;
    uint8_t reserved[1];
    BS2Rs485SlaveDevice slaveDevices[BS2_RS485_MAX_SLAVES_PER_CHANNEL];
} BS2Rs485Channel;

typedef struct {
    uint8_t mode;
    uint8_t numOfChannels;
    uint8_t reserved[2];
    BS2IntelligentPDInfo intelligentInfo;  //Updated to v2.8.0
    uint8_t reserved1[16];
    BS2Rs485Channel channels[BS2_RS485_MAX_CHANNELS];
} BS2Rs485Config;
```

1. *supportConfig*
[+V2.8] If this value is 0, the device will ignore all settings related to Intelligent PD (Peripheral Device) below.
useExceptionCode
exceptionCode
outputFormat
osdpID

2. *useExceptionCode*
[+V2.8] This option is available to choose whether the exception code is sent or not.

3. *exceptionCode*
[+V2.8] This function sends an exception code in case of authentication failure or authentication success but no card registered user.
Set the exception code to be used at this time.
If the exception code is 0 (0x0000000000000000), no exception code is generated.

4. *outputFormat*
[+V2.8] Intelligent Slave device can send Card ID or User ID upon successful authentication.
If it is 0, the card ID is output, if it is 1, the user ID is output.

5. *osdpID*
[+V2.8] This is a value used to distinguish two or more Suprema Intelligent devices from each other when connecting to the RS485 port of the same third party controller. You can set and designate a unique value between 0 and 127. The default OSDP ID for Suprema intelligent devices is 0.

6. *reserved*
[+V2.8] Reserved space.

7. *baudRate*
The RS-485 communication speed which can be configured as below.

Value
9600
19200
38400
57600
115200

8. *channelIndex*
(non configurable index) Communication channel index of the RS-485 network.

9. *useRegistance*
Registance flag - no effect on operation.

10. *numOfDevices*
Number of slave devices.

11. *reserved*
Reserved space.

12. *slaveDevices*
List of slave devices, which can be configured up to 32 devices.

13. *mode*
Decides the operating mode on the RS-485 network.

Value	Description
0	Not use
1	Master
2	Slave
3	Standalone

14. *numOfChannels*
Number of RS-485 channel.

15. *reserved*
Reserved space.

16. *intelligentInfo*
[+V2.8] This is Intelligent Slave Device Information
This only works when the device mode is RS485 default.
Once the Suprema device is connected to a slave device to the 3rd party ACU through RS485(OSDP), the Suprema device becomes a Peripheral Device of the OSDP automatically.

17. *reserved1*

Reserved space.

18. *channels*
List of RS-485 channels, which can be configured up to 4 channels.

BS2WiegandConfig

```
typedef struct {
    uint32_t length;
    uint8_t idFields[BS2_WIEGAND_MAX_FIELDS][BS2_WIEGAND_FIELD_SIZE];
    uint8_t parityFields[BS2_WIEGAND_MAX_PARITIES][BS2_WIEGAND_FIELD_SIZE];
    BS2_WIEGAND_PARITY parityType[BS2_WIEGAND_MAX_PARITIES];
    uint8_t parityPos[BS2_WIEGAND_MAX_PARITIES];
} BS2WiegandFormat;

typedef struct {
    uint8_t mode;
    uint8_t useWiegandBypass;
    uint8_t useFailCode;
    uint8_t failCode;
    uint16_t outPulseWidth;
    uint16_t outPulseInterval;
    uint32_t formatID;
    BS2WiegandFormat format;
    uint16_t wiegandInputMask;
    uint16_t wiegandCardMask;
    uint8_t wiegandCSNIndex;
    uint8_t useWiegandUserID;
    uint8_t reserved[26];
} BS2WiegandConfig;
```

1. *length*
The length of the wiegand card format.

2. *idFields*
You can set 4 id fields maximum. Each field's id needs to be inserted from the beginning to the end starting from the end of the array. For example, Standard 26bit wiegand card data is made up as “P FFFFFFFF NNNNNNNNNNNNNNNN P”. The Facility Code is “0 11111111 0000000000000000 0”, so it has the value of 0x01FE0000, and the Card Number has the value of 0x0001FFFE.

```
// for Facillity Code
idFields[][28] = 0x01;
idFields[][29] = 0xFE;
idFields[][30] = 0x00;
idFields[][31] = 0x00;

// for Card Number
idFields[1][28] = 0x00;
```

```
idFields[1][29] = 0x01;
idFields[1][30] = 0xFF;
idFields[1][31] = 0xFE;
```

3. *parityFields*
You can set 4 parity fields maximum, and enter the beginning and the end of the range where to check the parity.

4. *parityType*
Set the parity type.

Value	Description
0	Does not check parity
1	check odd parity
2	check even parity

5. *parityPos*
Select the position of the parity bit on the wiegand card data.

6. *mode*
Set the wiegand Input/Output mode.

Value	Description
0	Input
1	Output
2	In/Output

7. *useWiegandBypass*
The flag that indicates whether to send out a card data.

Value	Description
0	Output when authenticated.
1	Output without authentication.

8. *useFailCode*
The flag that indicates whether to send out a fail code when a card fails to authenticate.

9. *failCode*
Fail code value to replace the card data.

Value
0x00
0xFF

10. *outPulseWidth*
Output pulse width having a range of 20 ~ 100 us.

11. *outPulseInterval*
Output pulse frequency having a range of 200 ~ 20000 us.

12. *formatID*
The value used to distinguish the wiegand card format in the application, and this is not used from the device.

13. *format*
Wiegand format structure.

14. *wiegandInputMask*
Input mask for the wiegand input of the slave and wiegand device.

15. *wiegandCardMask*
Input mask for the wiegand input of the master device.

16. *wiegandCSNIndex*
Index that determines in which format the device will send out a wiegand ouput after the card has been read. This field is used only for Mifare and EM devices. Please check the *useWiegandFormat* field of the BS2CardConfig structure before configuring.

17. *useWiegandUserID*
The flag you can select whether card ID or user ID to be sent via Wiegand output.

Value	Description
0	Not use
1	Card ID
2	User ID

18. *reserved*
Reserved space.

BS2WiegandDeviceConfig

```
typedef struct {
    uint32_t deviceID;
    uint16_t port;
    uint8_t switchType;
    uint8_t reserved[1];
} BS2WiegandTamperInput;

typedef struct {
    uint32_t deviceID;
    uint16_t port;
    uint8_t reserved[10];
} BS2WiegandLedOutput;

typedef struct {
    uint32_t deviceID;
    uint16_t port;
    uint8_t reserved[34];
} BS2WiegandBuzzerOutput;
```

```
typedef struct {
    BS2WiegandTamperInput tamper;
    BS2WiegandLedOutput led[BS2_WIEGAND_STATUS_NUM];
    BS2WiegandBuzzerOutput buzzer;
    uint32_t reserved[32];
} BS2WiegandDeviceConfig;
```

1. *deviceID*
ID of the device which will receive the tamper signal of the Wiegand card reader.

2. *port*
Input port for the Wiegand card reader's tamper.

3. *switchType*
If the switch type is normally open, and the input isgnal is on, it will set off the trigger.

Value	Description
0	Normally open
1	Normally closed

4. *reserved*
Reserved space.

5. *deviceID*
ID of the device which will send the LED signal to the Wiegand card reader.

6. *port*
Output port for the Wiegand card reader's LED signal.

7. *reserved*
Reserved space.

8. *deviceID*
ID of the device which will send the buzzer signal to the Wiegand card reader.

9. *port*
Output port for the Wiegand card reader's buzzer signal.

10. *reserved*
Reserved space.

10. *led*
List of devices sending LED signals of the Wiegand card reader, which can be configured up to 2 devices.

Value	Description
0	Red LED
1	Green LED

BS2InputConfig

```
typedef struct {
    uint16_t minValue;
    uint16_t maxValue;
} BS2SVInputRange;

typedef struct {
    uint32_t deviceID;
    uint16_t port;
    uint8_t reserved[10];
} BS2WiegandLedOutput;

typedef struct {
    BS2SVInputRange shortInput;
    BS2SVInputRange openInput;
    BS2SVInputRange onInput;
    BS2SVInputRange offInput;
} BS2SupervisedInputConfig;

typedef struct {
    uint8_t numInputs;
    uint8_t numSupervised;
    uint16_t reseved;
    struct {
        uint8_t portIndex;
        uint8_t enabled;
        uint8_t supervised_index;
        uint8_t reserved[5];
        BS2SupervisedInputConfig config;
    } supervised_inputs[BS2_MAX_INPUT_NUM];
} BS2InputConfig;
```

1. *minValue*
Minimum voltage which has a range from 0 ~ 3300(3.3v).

2. *maxValue*
Maximum voltage which has a range from 0 ~ 3300(3.3v).

3. *shortInput*
Range of the voltage which will be distinguished as short input.

4. *openInput*
Range of the voltage which will be distinguished as open input.

5. *onInput*
Range of the voltage which will be distinguished as on input.

6. *offInput*
Range of the voltage which will be distinguished as off input.

7. *numInputs*
Number of input ports.

8. *numSupervised*
Number of the supervised input ports.

9. *portIndex*
Input port number.

10. *enabled*
Decides whether to use as a supervised input.

11. *supervised_index*
Type of supervised input's resistance value.

Value	Description
0	1k resistance
1	2.2k resistance
2	4.7k resistance
3	10k resistance
255	Custom

12. *reserved*
Reserved space.

13. *config*
Configuration that distinguishes the supervised input signal type. This configuration will be valid only when the supervised input's resistance is set as custom .

BS2WlanConfig

```
typedef struct {
    uint8_t enabled;
    uint8_t operationMode;
    uint8_t authType;
    uint8_t encryptionType;
    char essid[BS2_WLAN_SSID_SIZE];
    char authKey[BS2_WLAN_KEY_SIZE];
    uint8_t reserved2[32];
} BS2WlanConfig;
```

1. *enabled*
Decides whether to use the wireless LAN.

2. *operationMode*
Type of wireless LAN.

Value	Description
0	infrastructure

Value	Description
1	Ad-hoc

3. *authType*
Type of Wireless LAN authentication.

Value	Description
0	Open
1	Shared
2	WPA-PSK
3	WPA2-PSK

4. *encryptionType*
Type of wireless LAN encryption.

Value	Description
0	None
1	WEP
2	TKIP/AES
3	AES
3	TKIP

5. *essid*
ESS ID of the wireless network.

6. *authKey*
Password of the wireless network.

7. *reserved*
Reserved space.

BS2Trigger

```
typedef struct {
    uint16_t code;
    uint8_t reserved[2];
} BS2EventTrigger;

typedef struct {
    uint8_t port;
    uint8_t switchType;
    uint16_t duration;
    uint32_t scheduleID;
} BS2InputTrigger;

typedef struct {
    uint32_t type;
    uint32_t scheduleID;
} BS2ScheduleTrigger;
```

```
typedef struct {
    uint32_t deviceID;
    uint8_t type;
    uint8_t reserved[3];

    union {
        BS2EventTrigger event;
        BS2InputTrigger input;
        BS2ScheduleTrigger schedule;
    }
} BS2Trigger;
```

1. *code*
Event log that will set off the trigger.

2. *reserved*
Reserved space.

3. *port*
Input port number that will set off the trigger.

4. *switchType*
If the switch type is normally open, and the input isgnal is on, it will set off the trigger.

Value	Description
0	Normally open
1	Normally closed

5. *duration*
The duration time of the signal that will set off the trigger. The unit of time is milliseconds and the minimum value is 100.

6. *scheduleID*
ID of the time schedule when to operate the trigger.

7. *type*
Type of the schedule trigger.

Value	Description
0	Start schedule trigger
1	End schedule trigger

8. *scheduleID*
ID of the time schedule when to operate the trigger.

9. *deviceID*
ID of the device that will perform the trigger.

10. *type*
Type of trigger.

Value	Description
0	None
1	Event trigger
2	Input trigger
3	Schedule trigger

BS2Action

```
typedef struct {
    uint32_t signalID;
    uint16_t count;
    uint16_t onDuration;
    uint16_t offDuration;
    uint16_t delay;
} BS2Signal;

typedef struct {
    uint8_t portIndex;
    uint8_t reserved[3];
    BS2Signal signal;
} BS2OutputPortAction;

typedef struct {
    uint8_t relayIndex;
    uint8_t reserved[3];
    BS2Signal signal;
} BS2RelayAction;

typedef struct {
    uint8_t color;
    uint8_t reserved[1];
    uint16_t duration;
    uint16_t delay;
} BS2LedSignal;

typedef struct {
    uint16_t count;
    uint8_t reserved[2];
    BS2LedSignal signal[3];
} BS2LedAction;

typedef struct {
    uint8_t tone;
    uint8_t fadeout;
    uint16_t duration;
    uint16_t delay;
} BS2BuzzerSignal;

typedef struct {
    uint16_t count;
```

```
    uint8_t reserved[2];
    BS2BuzzerSignal signal[3];
} BS2BuzzerAction;

typedef struct {
    uint8_t duration;
    uint8_t reserved[3];
    uint32_t displayID;
    uint32_t resourceID;
} BS2DisplayAction;

typedef struct {
    uint8_t count;
    uint16_t soundIndex;
    uint8_t reserved[5];
} BS2SoundAction;

typedef struct {
    uint32_t deviceID;
    uint8_t type;
    uint8_t stopFlag;
    uint16_t delay;
    union {
        BS2RelayAction relay;
        BS2OutputPortAction outputPort;
        BS2DisplayAction display;
        BS2SoundAction sound;
        BS2LedAction led;
        BS2BuzzerAction buzzer;
    };
} BS2Action;
```

1. *signalID*
Index that is used to manage the signal type from the application.

2. *count*
Number of signal execution count.

3. *onDuration*
Duration of the ON signal. The unit of time is milliseconds.

4. *offDuration*
Duration of the OFF signal. The unit of time is milliseconds.

5. *delay*
Delay time before the signal starts. The unit of time is milliseconds/ For example, count(2), onDuration(100), offDuration(100), delay(50) will execute a signal as below.

50ms wait	**signal on(100)**	**signal off(100)**	**signal on(100)**	**signal off(100)**

6. *portIndex*

Number of the TTL output port.

7. *reserved*
Reserved Space.

8. *relayIndex*
Number of the TTL output port.

9. *reserved*
Reserved Space.

10. *color*
LED colory type.

Value	Description
0	LED Off
1	Red LED
2	Yellow LED
3	Green LED
4	Blue-Green LED
5	Blue LED
6	Magenta LED
7	White LED

11. *reserved*
Reserved space.

12. *duration*
Duration of the LED. The unit of time is milliseconds.

13. *delay*
Delay before the LED flickers. The unit of time is milliseconds.

14. *count*
Number of LED signal count. When set as 0 it is disabled, and when set as -1 it will repeat infinitely.

15. *reserved*
Reserved space.

16. *tone*
Volume of the buzzer.

Value	Description
0	No sound
1	Low
2	Medium
3	High

17. *count*
Number of buzzer signal count. When set as 0 it is disabled, and when set as -1 it will repeat infinitely.

18. *reserved*
Reserved space.

19. *duration*
Duration of the display operation. The unit of time is milliseconds.

20. *reserved*
Reserved space.

21. *displayID*
Not supported yet.

22. *resourceID*
Not supported yet.

23. *count*
Number of the sound signal count.

24. *soundIndex*
Index of the sound resource.

Value	Description
0	Welcome sound
1	Auth success sound
2	Auth fail sound

25. *deviceID*
ID of the device that will execute the action.

26. *type*
Action types.

[DoorModule-20, CoreStation-40]
If the action type is relay or TTL (Output) and the action device is DM20, CS40, Action type should be set only as relay action (6). (TTL setting not possible)

[DM20]

- Action type : Relay
- relay.relayIndex : 0 ~ 3 (RELAY 0 ~ 3)
- relay.relayIndex : 4 ~ 9 (OUTPUT 0 ~ 5)

[CS40]

- Action type : Relay
- relay.relayIndex : 0 ~ 3 (RELAY 0 ~ 3)
- relay.relayIndex : 4 ~ 11 (OUTPUT 0 ~ 7)

Value	Description
0	None
1	Lock device
2	Unlock device
3	Reboot device
4	Release alarm
5	General input
6	Relay action
7	TTL action
8	Sound action
9	Display action
10	Buzzer action
11	Led action
12	Fire alarm input
13	Auth Success(Access granted)
14	Auth Fail(Access denied)
15	Lift action

27. *stopFlag*
Specifies the condition to stop the Action.
If this value is set to 1 and the signal is detected through the door sensor, the action will stop.
If this value is set to 2, it can be stopped only by the current action information.
In general, related APIs that stop an action are called with a zone id, in which case all devices in the zone will stop the action.
By setting stopFlag to 2 and and loading action information, you can selectively control only the alarms of that device.

Value	Description
0	Don't stop
1	Stop if door is closed
2	Stop by command(Added in V2.6.0)

28. *delay*
Action delay. Unit is millisecond(ms).

BS2TriggerActionConfig

```
typedef struct {
    uint8_t numItems;
    uint8_t reserved[3];
    BS2TriggerAction items[BS2_MAX_TRIGGER_ACTION];
    uint8_t reserved2[32];
} BS2TriggerActionConfig;
```

1. *numItems*
Number of trigger actions.

2. *reserved*
Reserved space.

3. *items*
List of trigger actions, which can be configured up to 128 trigger actions.

4. *reserved2*
Reserved space.

BS2EventConfig

```
typedef struct {
    uint32_t numImageEventFilter;
    struct {
        uint8_t mainEventCode;
        uint8_t reserved[3];
        uint32_t scheduleID;
    } imageEventFilter[BS2_EVENT_MAX_IMAGE_CODE_COUNT];
    uint8_t reserved[32];
} BS2EventConfig;
```

1. *numImageEventFilter*
Number of image log filters.

2. *mainEventCode*
Main code of the log where the image log will be placed.

3. *reserved*
Reserved space.

4. *scheduleID*
ID of the time schedule when to store the image logs.

5. *reserved*
Reserved space.

BS2WiegandMultiConfig

```
typedef struct {
    uint32_t formatID;
    BS2WiegandFormat format;
    uint8_t reserved[32];
} BS2WiegandInConfig;

typedef struct {
    BS2WiegandInConfig formats[MAX_WIEGAND_IN_COUNT];
    uint8_t reserved[32];
```

```
} BS2WiegandMultiConfig;
```

1. *formatID*
Wiegand format index.

2. *format*
Wiegand format structure.

3. *reserved*
Reserved space.

4. *formats*
Available to configure up to 15 formats.

5. *reserved*
Reserved space.

BS1CardConfig

```
typedef struct {
    enum {
        MIFARE_KEY_SIZE = 6,
        MIFARE_MAX_TEMPLATE = 4,

        VALID_MAGIC_NO = 0x1f1f1f1f,
    };

    // Options
    uint32_t     magicNo;
    uint32_t     disabled;
    uint32_t     useCSNOnly;         // default 0
    uint32_t     bioentryCompatible; // default 0

    // Keys
    uint32_t    useSecondaryKey;
    uint32_t     reserved1;
    uint8_t     primaryKey[MIFARE_KEY_SIZE];
    uint8_t        reserved2[2];
    uint8_t     secondaryKey[MIFARE_KEY_SIZE];
    uint8_t        reserved3[2];

    // Layout
    uint32_t     cisIndex;
    uint32_t     numOfTemplate;
    uint32_t     templateSize;
    uint32_t     templateStartBlock[MIFARE_MAX_TEMPLATE];

    uint32_t     reserve4[15];
```

```
} BS1CardConfig;
```

1. *magicNo*
Not used.

2. *disabled*
Not used.

3. *useCSNOnly*
Flag that indicates whether to read the v1 ToC cards.

4. *bioentryCompatible*
Not used.

5. *useSecondaryKey*
Not used.

6. *reserved1*
Reserved space.

7. *primaryKey*
Not used.

8. *reserved2*
Reserved space.

9. *secondaryKey*
Not used.

10. *reserved3*
Reserved Space.

11. *cisIndex*
Not used.

12. *numOfTemplate*
Number of template that is used.

13. *templateSize*
Size of each template.

14. *templateStartBlock*
Not used.

15. *reserve4*
Reserved Space.

BS2SystemConfigExt

```
typedef struct {
    uint8_t primarySecureKey[SEC_KEY_SIZE];
    uint8_t secondarySecureKey[SEC_KEY_SIZE];

    uint8_t reserved3[32];
} BS2SystemConfigExt;
```

1. *primarySecureKey*
Primary encryption key used between master and slave devices.

2. *secondarySecureKey*
Secondary encryption key used between master and slave devices.

3. *reserved3*
Reserved space.

BS2VoipConfig

```
typedef struct {
    BS2_URL            serverUrl;            ///
    BS2_PORT        serverPort;            ///
    BS2_USER_ID        userID;                ///
    BS2_USER_ID        userPW;                ///

    uint8_t            exitButton;            /// << *, #, 0~9
    uint8_t            dtmfMode;            ///
    BS2_BOOL        bUse;                ///
    uint8_t            reseverd[1];        ///

        uint32_t                numPhonBook;
    BS2UserPhoneItem     phonebook[BS2_VOIP_MAX_PHONEBOOK];  ///

    uint8_t                reserved2[32];        ///
} BS2VoipConfig;
```

1. *serverUrl*
URL of the SIP server.

2. *serverPort*
Port number of the SIP server.

3. *userID*
User ID to access the SIP server.

4. *userPW*
Password that is used to access the SIP server.

5. *exitButton*
Button to be used as an exit button. (*, #, 0~9)

Value	Description
0	*
1	#
2 ~ 11	0 ~ 9

6. *dtmfMode*
Tone of the keypad.

7. *bUse*
Flag that determines whether the VoIP feature is used.

8. *reseverd*
Reserved space.

9. *numPhonBook*
Number of phone books.

10. *phonebook*
List of extension numbers, which can be configured up to 32.

8. *reserved2*
Reserved space.

BS2FaceConfig

```
typedef struct {
    uint8_t       securityLevel;
    uint8_t       lightCondition;
    uint8_t       enrollThreshold;
    uint8_t       detectSensitivity;

    uint16_t      enrollTimeout;
    uint8_t       lfdLevel;
    bool          quickEnrollment;

    uint8_t       previewOption;
    bool          checkDuplicate;
    uint8_t       operationMode;
    uint8_t       maxRotation;

    struct {
        uint16_t  min;
        uint16_t  max;
    } faceWidth;
```

```
    struct {
        uint16_t  x;
        uint16_t  width;
    } searchRange;

    uint8_t       reserved2[18];
} BS2FaceConfig;
```

1. *securityLevel*
Face authentication security level. This is used across the system. .

Value	Description
0	Basic
1	Highly secure
2	Most highly secure

2. *lightCondition*
Configuration of the light condition.

Value	Description
0	Indoor
1	Outdoor
2	Automatic
3	[+ 2.8] Not used (FaceStation F2 v1.1.0 or higher version)

[Note]
FaceStation F2: v1.0.0 - v1.0.5
Ambient Brightness: Normal, High, Auto
FaceStation F2: v1.1.0 or higher version
Light Brightness: Normal, High, Not Used

3. *enrollThreshold*
Threshold of face enrollment. It determines how much movement of pose is allowed when enrolling the face.

Value	Description
0	THRESHOLD_0 (Most strict)
1	THRESHOLD_1
2	THRESHOLD_2
3	THRESHOLD_3
4	THRESHOLD_4 (Default)
5	THRESHOLD_5
6	THRESHOLD_6
7	THRESHOLD_7
8	THRESHOLD_8
9	THRESHOLD_9 (Least strict)

4. *detectSensitivity*

Configuration of sensitivity on detecting the face.

Value	Description
0	Off
1	Low
2	Medium
3	High

5. *enrollTimeout*

`FaceStation2, FaceLite` : Timeout period of face scanning which is 60 seconds by default.

Value	Description
BS2_FACE_ENROLL_TIMEOUT_MIN	30
BS2_FACE_ENROLL_TIMEOUT_MAX	60
BS2_FACE_ENROLL_TIMEOUT_DEFAULT	BS2_FACE_ENROLL_TIMEOUT_MAX

`FaceStation F2` : [+ 2.7.1] Face scan wait time, default is 20 seconds.

Value	Description
BS2_FACE_EX_ENROLL_TIMEOUT_MIN	10
BS2_FACE_EX_ENROLL_TIMEOUT_MAX	20
BS2_FACE_EX_ENROLL_TIMEOUT_DEFAULT	BS2_FACE_EX_ENROLL_TIMEOUT_MAX

6. *lfdLevel*

[+ 2.6.3] Configuration for the LFD(Live Face Detection - fake face detection) sensitivity.
`FaceStation2, FaceLite` : Default is 0.
`FaceStation F2` : [+ 2.7.1] Default is 1.

Value	Description
0	Not Use
1	Strict
2	More Strict
3	Most Strict

7. *quickEnrollment*

[+ 2.6.3] Quick face enrollment process.
True - Face enrollment process with a single step.
False - Face enrollment process with 3 steps.
Please use false if you want to enroll with a high quality of face templates.

8. *previewOption*

[+ 2.6.3] IR camera preview option when you authenticate with the face.
Only used to `FaceLite`.

Value	Description
0	Preview not used
1	Preview not used at first of authentication, preview at 1/2 stage
2	Preview of all stages on authentication

9. *checkDuplicate*

[+ 2.6.4] Check whether the scanned face is duplicated in the device.

10. *operationMode*

[+ 2.7.1] FaceStation F2 Configures operation mode with below values, default is Fusion mode.

Value	Mode	Description	Default
0	Fusion Mode	Visual matching + IR matching	Default
1	Visual Mode	Visual matching	
2	Visual + IR	Visual matching, IR detects only face	

11. *maxRotation*

[+ 2.7.1] FaceStation F2 When face is recognized normally it's front side.
Still, it is possible to determine how many degrees the image has been rotated from the front when FSF2 detects a face.
This enables detection failure in the case of images rotated over a certain angle.
maxRotation represents the maximum allowable value in this case, and the default value is 15 degrees.

12. *faceWidth*

[+ 2.7.1] FaceStation F2 This indicates the width of the face image, and you can specify the minimum and maximum values.
The default values of min and max are as follows.
[+ 2.8.3] BioStation 3 The settings are ignored.

	Default(min)	Default(max)
FSF2	66	250
BS3	-	-

13. *searchRange*

[+ 2.7.1] FaceStation F2 Represents the face search range, and you can specify the x value (horizontal coordinate) of the range and the width from the x value point.
The default values of x and width are as follows.
[+ 2.8.3] BioStation 3 The settings are ignored.

	Default(x)	Default(width)
FSF2	144	432
BS3	-	-

14. *detectDistance*

[+ 2.8.3] BioStation 3 This configures the minimum and maximum detection range for facial recognition.
We no longer support faceWidth to pinpoint the face location using pixel units due to its complexity. Instead, we set the detection range of the subject(face). The unit is set to cm, and the value must be inputted as a multiple of 10.

	Min limit for min detection range	Max limit for min detection range	Min detection range(Default)	Min limit for max detection range	Max limit for max detection range	Max sensing range(No limit)	Max sensing range(Default)
BS3	30	100	60	40	100	255	100

15. *wideSearch*
[+ 2.8.3] `BioStation 3` This can increase the detection range for facial recognition.
We no longer support searchRange to set the x-coordinate and width due to its complexity.
Instead, we set the face detection setting as default(FALSE), or a wide area(TRUE).
The details of the settings and protocols for the detection of wide area is set within the device, which the user cannot change.

If this setting is set to TRUE, the camera detects subjects within a large range, and unintentionally detect and authenticate multiple subjects at once. Therefore, the default setting is at FALSE.

16. *unused*
Reserved space.

17. *reserved*
Reserved space.

BS2Rs485ConfigEX

```
typedef struct {
    uint32_t baudRate;
    uint8_t channelIndex;
    uint8_t useRegistance;
    uint8_t numOfDevices;
    uint8_t reserved[1];
    BS2Rs485SlaveDeviceEX slaveDevices[BS2_RS485_MAX_SLAVES_PER_CHANNEL];
} BS2Rs485ChannelEX;

typedef struct {
    uint8_t mode[BS2_RS485_MAX_CHANNELS_EX];
    uint8_t numOfChannels;
    uint8_t reserved[2];
    uint8_t reserved1[32];
    BS2Rs485ChannelEX channels[BS2_RS485_MAX_CHANNELS];
} BS2Rs485ConfigEX;
```

1. *baudRate*
The RS-485 communication speed which can be configured as below.

Value
9600
19200
38400

Value
57600
115200

2. *channelIndex*
Communication channel index of the RS-485 network.

3. *useRegistance*
Decides whether to use a resistance.

4. *numOfDevices*
Number of slave devices.

5. *slaveDevices*
List of slave devices, which can be configured up to 32 devices.

6. *mode*
Decides the operating mode on the RS-485 network.

Value	Description
0	Not use
1	Master
2	Slave
3	Standalone

7. *numOfChannels*
Number of RS-485 channel.

8. *reserved*
Reserved space.

9. *reserved1*
Reserved space.

10. *channels*
List of RS-485 channels, which can be configured up to 8 channels.

BS2CardConfigEx

```
typedef struct {
    uint8_t oid_ADF[13];           ///   //valid
value//{0x2A,0x85,0x70,0x81,0x1E,0x10,0x00,0x07,0x00,0x00,0x02,0x00,0x00}
    uint8_t size_ADF;             //
    uint8_t reserved1[2];          ///
    uint8_t oid_DataObjectID[8];
    uint16_t size_DataObject[8];
    uint8_t primaryKeyAuth[16];            //valid value
    uint8_t secondaryKeyAuth[16];      /// //valid value
```

```
    uint8_t reserved2[24];
} BS2SeosCard;
typedef struct {
    BS2SeosCard seos;
    uint8_t reserved[24];
} BS2CardConfigEx;
```

1. *oid_ADF*
ADF Address (Non changable)

2. *size_ADF*
ADF size

3. *reserved1*
Reserved sapce

4. *oid_DataObjectID*
DataObject ID

5. *size_DataObject*
DataObject size

6. *primaryKeyAuth*
Primary encryption key to access the Seoscard information

7. *secondaryKeyAuth*
Secondary encryption key to access the Seoscard information

8. *reserved2*
Reserved space

9. *seos*
BS2SeosCard information

10. *reserved*
Reserved space

BS2DstConfig

```
enum {
    BS2_MAX_DST_SCHEDULE = 2,
};

typedef struct {
    uint16_t year;        // year, 0 means every year.
    uint8_t month;        // [0, 11] : months since January
    int8_t ordinal;        // [0, -1] : first, second, ..., last
    uint8_t weekDay;    // [0, 6] : days since Sunday
    uint8_t hour;        // [0, 23]
```

```
    uint8_t minute;         // [0, 59]
    uint8_t second;         // [0, 59]
} BS2WeekTime;

typedef struct {
    BS2WeekTime startTime;
    BS2WeekTime endTime;
    int32_t timeOffset;     // in seconds
    uint8_t reserved[4];
} BS2DstSchedule;

typedef struct {
    uint8_t numSchedules;
    uint8_t reserved[31];

    BS2DstSchedule schedules[BS2_MAX_DST_SCHEDULE];
} BS2DstConfig;
```

1. *year*
Means year, and if set to 0, it means yearly.

2. *month*
Means month, and has a value between 0 and 11 [January-December].

3. *ordinal*
It starts with 0 and means the order of first, second, etc.

4. *weekDay*
Day of the week, 0 means Sunday, 1 means Monday.

5. *hour*
Specifies the time in 24-hour format.

6. *minute*
Specifies the minute.

7. *second*
Specifies the seconds.

8. *startTime*
It means start date and time.

9. *endTime*
It means the end date.

10. *timeOffset*
You can apply the DST time in seconds.
For example, if you want to apply 1 hour, enter 3600.

11. *reserved*
Reserved space.

12. *numSchedules*
The number of DST schedules to apply.

13. *schedules*
DST schedule, up to two can be specified.

BS2Configs

```
typedef struct {
    uint32_t configMask;
    BS2FactoryConfig factoryConfig;
    BS2SystemConfig systemConfig;
    BS2AuthConfig authConfig;
    BS2StatusConfig statusConfig;
    BS2DisplayConfig displayConfig;
    BS2IpConfig ipConfig;
    BS2IpConfigExt ipConfigExt;
    BS2TNAConfig tnaConfig;
    BS2CardConfig cardConfig;
    BS2FingerprintConfig fingerprintConfig;
    BS2Rs485Config rs485Config;
    BS2WiegandConfig wiegandConfig;
    BS2WiegandDeviceConfig wiegandDeviceConfig;
    BS2InputConfig inputConfig;
    BS2WlanConfig wlanConfig;
    BS2TriggerActionConfig triggerActionConfig;
    BS2EventConfig eventConfig;
    BS2WiegandMultiConfig wiegandMultiConfig;
    BS1CardConfig card1xConfig;
    BS2SystemConfigExt systemExtConfig;
    BS2VoipConfig voipConfig;
    BS2FaceConfig faceConfig;
} BS2Configs;
```

1. *configMask*
Mask value of the configuration to be retrieved or set.

Value	Description
0x0000	None
0x0001	Factory configuration
0x0002	System configuration
0x0004	TCP/IP configuration
0x0008	RS485 configuration
0x0010	Wireless LAN configuration
0x0020	Authentication configuration
0x0040	Card configuration
0x0080	Fingerprint configuration

Value	Description
0x0100	Face configuration
0x0200	Trigger Action configuration
0x0400	Display configuration
0x0800	Sound configuration
0x1000	Status Signal(LED, Buzzer) configuration
0x2000	Wiegand configuration
0x4000	USB configuration
0x8000	Time and Attendance configuration
0x10000	Videophone configuration
0x20000	Interphone configuration
0x40000	Voice over IP configuration
0x80000	Input(Supervised input) configuration
0x100000	Wiegand IO Device configuration
0x200000	Time and Attendance configuration
0x400000	DNS and Server url configuration
0x800000	Event configuration
0x1000000	1x Card configuration
0x2000000	Multi-Wiegand configuration
0x4000000	Extended System configuration
~~0x8000000~~	~~Daylight Saving configuration (Deprecated)~~
0x10000000	RS485 Extended configuration
0x20000000	Extended Card configuration
0x40000000	Daylight Saving configuration
0xFFFFFFFF	All configuration

BS2IPV6Config

```
enum {
    BS2_MAX_IPV6_ALLOCATED_ADDR = 8,
};

typedef struct {
    uint8_t useIPV6;
    uint8_t reserved1;
    uint8_t useDhcpV6;
    uint8_t useDnsV6;
    uint8_t reserved[1];
    char staticIpAddressV6[BS2_IPV6_ADDR_SIZE];
    char staticGatewayV6[BS2_IPV6_ADDR_SIZE];
    char dnsAddrV6[BS2_IPV6_ADDR_SIZE];
    char serverIpAddressV6[BS2_IPV6_ADDR_SIZE];
    uint16_t serverPortV6;
    uint16_t sslServerPortV6;
    uint16_t portV6;
    uint8_t numOfAllocatedAddressV6;
```

```
    uint8_t numOfAllocatedGatewayV6;
    uint8_t reserved[8];
    char
allocatedIpAddressV6[BS2_IPV6_ADDR_SIZE][BS2_MAX_IPV6_ALLOCATED_ADDR];
    char
allocatedGatewayV6[BS2_IPV6_ADDR_SIZE][BS2_MAX_IPV6_ALLOCATED_ADDR];
} BS2IpConfig;
```

1. *useIPV6*
Flag indicating whether to use IP V6.

2. *reserved1*
Reserved space.

3. *useDhcpV6*
Flag indicating whether to use DHCP.

4. *useDnsV6*
Decides whether to use server address or server URL.

5. *staticIpAddressV6*
Static IP V6 address of current device.

6. *staticGatewayV6*
Static IP V6 address of gateway.

7. *dnsAddrV6*
DNS address.

8. *serverIpAddressV6*
IP address of BioStar. Used only in the server mode.

9. *serverPortV6*
Port number of BioStar. Used only in the server mode.

10. *sslServerPortV6*
Used when the connectionMode is set as server SSL mode, which is the port of the SDK application.

11. *portV6*
Port number of the device.

12. *numOfAllocatedAddressV6*
The number of IP V6 address currently assigned to the device.

13. *numOfAllocatedGatewayV6*
The number of gateway address currently assigned to the device.

14. *reserved*
Reserved space.

15. *allocatedIpAddressV6*

The IP V6 address currently assigned to the device.

16. *allocatedGatewayV6*
The gateway address currently assigned to the device.

BS2DesFireCardConfigEx

```
typedef struct {
    uint8_t appMasterKey[16];
    uint8_t fileReadKey[16];
    uint8_t fileWriteKey[16];
    uint8_t fileReadKeyNumber;
    uint8_t fileWriteKeyNumber;
    uint8_t reserved[2];
} BS2DesFireAppLevelKey;                    ///< 52 bytes

typedef struct {
    BS2DesFireAppLevelKey desfireAppKey;    ///< 52 bytes
    uint8_t reserved[16];
} BS2DesFireCardConfigEx;                   ///< 68 bytes
```

1. *appMasterKey*
Application master key of DesFire.

2. *fileReadKey*
The key used to read the file.

3. *fileWriteKey*
The key used to write the file.

4. *fileReadKeyNumber*
The index of the key for reading the file.

5. *fileWriteKeyNumber*
The index of the key for writing the file.

6. *reserved*
Reserved space.

7. *desfireAppKey*
A structure containing DesFire key information.

8. *reserved*
Reserved space.

BS2AuthConfigExt

```
typedef struct {
    uint32_t extAuthSchedule[BS2_MAX_NUM_OF_EXT_AUTH_MODE];
    uint8_t useGlobalAPB;
    uint8_t globalAPBFailAction;
    uint8_t useGroupMatching;
    uint8_t reserved;

    uint8_t reserved2[4];

    uint8_t usePrivateAuth;
    uint8_t faceDetectionLevel;
    uint8_t useServerMatching;
    uint8_t useFullAccess;

    uint8_t matchTimeout;
    uint8_t authTimeout;
    uint8_t numOperators;
    uint8_t reserved3[1];

    struct {
        char userID[BS2_USER_ID_SIZE];
        uint8_t level;
        uint8_t reserved[3];
    } operators[BS2_MAX_OPERATORS];

    uint8_t reserved4[256];
} BS2AuthConfigExt;
```

1. *extAuthSchedule*

Schedule values to operate when each authentication mode is activated.
It has the following meanings for each value.
If the value in the array is greater than 0, the authentication mode is activated.
In the explanations below, biometric information means fingerprints or faces depending on the device.

Value	Code	Description
11	BS2_EXT_AUTH_MODE_FACE_ONLY	Face
12	BS2_EXT_AUTH_MODE_FACE_FINGERPRINT	Face + Fingerprint
13	BS2_EXT_AUTH_MODE_FACE_PIN	Face + PIN
14	BS2_EXT_AUTH_MODE_FACE_FINGERPRINT_OR_PIN	Face + Fingerprint/PIN
15	BS2_EXT_AUTH_MODE_FACE_FINGERPRINT_PIN	Face + Fingerprint + PIN
16	BS2_EXT_AUTH_MODE_FINGERPRINT_ONLY	Fingerprint
17	BS2_EXT_AUTH_MODE_FINGERPRINT_FACE	Fingerprint + Face
18	BS2_EXT_AUTH_MODE_FINGERPRINT_PIN	Fingerprint + PIN
19	BS2_EXT_AUTH_MODE_FINGERPRINT_FACE_OR_PIN	Fingerprint + Face/PIN
20	BS2_EXT_AUTH_MODE_FINGERPRINT_FACE_PIN	Fingerprint + Face + PIN
21	BS2_EXT_AUTH_MODE_CARD_ONLY	Card
22	BS2_EXT_AUTH_MODE_CARD_FACE	Cardn + Face
23	BS2_EXT_AUTH_MODE_CARD_FINGERPRINT	Card + Fingerprint
24	BS2_EXT_AUTH_MODE_CARD_PIN	Card + PIN

Value	Code	Description
25	BS2_EXT_AUTH_MODE_CARD_FACE_OR_FINGERPRINT	Card + Face/Fingerprint
26	BS2_EXT_AUTH_MODE_CARD_FACE_OR_PIN	Card + Face/PIN
27	BS2_EXT_AUTH_MODE_CARD_FINGERPRINT_OR_PIN	Card + Fingerprint/PIN
28	BS2_EXT_AUTH_MODE_CARD_FACE_OR_FINGERPRINT_OR_PIN	Card + Face/Fingerprint/PIN
29	BS2_EXT_AUTH_MODE_CARD_FACE_FINGERPRINT	Card + Face + Fingerprint
30	BS2_EXT_AUTH_MODE_CARD_FACE_PIN	Card + Face + PIN
31	BS2_EXT_AUTH_MODE_CARD_FINGERPRINT_FACE	Card + Fingerprint + Face
32	BS2_EXT_AUTH_MODE_CARD_FINGERPRINT_PIN	Card + Fingerprint + PIN
33	BS2_EXT_AUTH_MODE_CARD_FACE_OR_FINGERPRINT_PIN	Card + Face/Fingerprint + PIN
34	BS2_EXT_AUTH_MODE_CARD_FACE_FINGERPRINT_OR_PIN	Card + Face + Fingerprint/PIN
35	BS2_EXT_AUTH_MODE_CARD_FINGERPRINT_FACE_OR_PIN	Card + Fingerprint + Face/PIN
36	BS2_EXT_AUTH_MODE_ID_FACE	ID + Face
37	BS2_EXT_AUTH_MODE_ID_FINGERPRINT	ID + Fingerprint
38	BS2_EXT_AUTH_MODE_ID_PIN	ID + PIN
39	BS2_EXT_AUTH_MODE_ID_FACE_OR_FINGERPRINT	ID + Face/Fingerprint
40	BS2_EXT_AUTH_MODE_ID_FACE_OR_PIN	ID + Face/PIN
41	BS2_EXT_AUTH_MODE_ID_FINGERPRINT_OR_PIN	ID + Fingerprint/PIN
42	BS2_EXT_AUTH_MODE_ID_FACE_OR_FINGERPRINT_OR_PIN	ID + Face/Fingerprint/PIN
43	BS2_EXT_AUTH_MODE_ID_FACE_FINGERPRINT	ID + Face + Fingerprint
44	BS2_EXT_AUTH_MODE_ID_FACE_PIN	ID + Face + PIN
45	BS2_EXT_AUTH_MODE_ID_FINGERPRINT_FACE	ID + Fingerprint + Face
46	BS2_EXT_AUTH_MODE_ID_FINGERPRINT_PIN	ID + Fingerprint + PIN
47	BS2_EXT_AUTH_MODE_ID_FACE_OR_FINGERPRINT_PIN	ID + Face/Fingerprint + PIN
48	BS2_EXT_AUTH_MODE_ID_FACE_FINGERPRINT_OR_PIN	ID + Face + Fingerprint/PIN
49	BS2_EXT_AUTH_MODE_ID_FINGERPRINT_FACE_OR_PIN	ID + Fingerprint + Face/PIN

2. *useGlobalAPB*
This flag determines whether to enable Global APB zone.

3. *globalAPBFailAction*
This is a basic action to be performed when the device cannot query the server for Global APB violation.

Value	Description
0	Do not check APB
1	Soft APB
2	Hard APB

4. *useGroupMatching*

Enables facial group matching.

5. *reserved*
Reserved space.

6. *reserved2*
Reserved space.

7. *usePrivateAuth*
Enable private authentication mode.

8. *faceDetectionLevel*
This is the face detection level value when authenticating the user in BioStation A2, and if a face is detected at a level lower than the specified level, it is treated as an authentication failure.
When enabled, the camera view according to Normal/Strict is displayed, and access is denied if the image log is not recognized as a face when successful authentication. The default is 0.

Value	Description
0	Do not detect face
1	Normal mode
2	Strict mode

> It is available only in BioStation A2, not available in FaceStation2 or FaceLite.

9. *useServerMatching*
Enable server matching for fingerprint matching or facial matching.

10. *useFullAccess*
This parameter is not in use.

11. *matchTimeout*
The maximum response time in fingerprint or facial matching and the unit is seconds(sec).

12. *authTimeout*
The maximum response time in user authentication and the unit is seconds(sec).

13. *numOperators*
The number of operators.

14. *reserved3*
Reserved

15. *userID*
User ID

16. *level*
It specifies the corresponding level of the user when the user is authenticated.

Value	Description
0	No level
1	Operator level
2	System configuration level
3	User information level

CAUTION
You must specify the number of operators to be added in the field ***numOperators*** when adding operators.

17. *reserved*
Reserved
18. *reserved4*
Reserved

BS2FaceConfigExt

```
typedef struct {
    uint8_t thermalCheckMode;
    uint8_t maskCheckMode;
    uint8_t reserved[2];

    uint8_t thermalFormat;
    uint8_t reserved2;

    uint16_t thermalThresholdLow;
    uint16_t thermalThresholdHigh;
    uint8_t maskDetectionLevel;
    uint8_t auditTemperature;

    uint8_t useRejectSound;
    uint8_t useOverlapThermal;
    uint8_t useDynamicROI;
    uint8_t faceCheckOrder;
} BS2FaceConfigExt;
```

1. *thermalCheckMode*
Sets the thermal check mode.
When set to HARD, access is denied if exceeding the thermalThreshold.
When set to SOFT, access is not affected even if exceeding the thermalThreshold but leaves a related log.
If thermalCheckMode is set to No use(0),
The settings of thermalFormat, thermalThreshold, auditTemperature, and useOverlapThermal are ignored.
And the reject sound due to thermal check by useRejectSound, the temperature check by faceCheckOrder is ignored.

Value	Description	Default
0	No use	Default
1	Thermal Check Mode (HARD)	
2	Thermal Check Mode (SOFT)	

2. *maskCheckMode*
`FaceStation F2` Sets the Mask Check Mode.
`FaceStation 2` This setting is ignored.
When set to HARD, access is denied if not detecting any mask on the face based on maskDetectionLevel.
When set to SOFT, access is not affected even if not detecting any mask on the face based on maskDetectionLevel but leaves a related log.
If maskCheckMode is set to No use(0),
The setting of maskDetectionLevel is ignored.
And the reject sound due to mask detection check by useRejectSound, the mask detection check by faceCheckOrder is ignored.

Value	Description	Default
0	No use	Default
1	Mask Check Mode (HARD)	
2	Mask Check Mode (SOFT)	

3. *reserved*
Reserved

4. *thermalFormat*
Represents the temperature unit. You may choose the unit in Fahrenheit or Celsius

Value	Description	Default
0	Fahrenheit	
1	Celsius	Default

5. *reserved2*
Reserved

6. *thermalThresholdLow*
`Supported version : FaceStation F2 V1.0.2, FaceStation 2 V1.5.0`
This is the range value for determining high temperature and must be entered as a value multiplied by 100 of the temperature to be set.
Also, you can only enter in degrees Celsius.
This value is the basis for the denial of authentication, and the setting range is between 100 (1º) and 4500 (45º).
The default value is 3200 (32º), and if a value larger or smaller than the setting range is entered, the default value a set to 3200 (32º).
And you must set a value less than thermalThresholdHigh.

7. *thermalThresholdHigh*
This is the range value for determining high temperature and must be entered as a value multiplied by 100 of the temperature to be set.
Also, you can only enter in degrees Celsius.

This value is the basis for the denial of authentication, and the setting range is between 100 (1º) and 4500 (45º).
The default value is 3800 (38º), and if a value larger or smaller than the setting range is entered, the default value a set to 3800 (38º).
And you must set a value greater than thermalThresholdLow.

8. *maskDetectionLevel*
`FaceStation F2` Sets the mask detection level. The detecting level is based on internal setting value.
`FaceStation 2` This setting is ignored.

Value	Description	Default
0	No detection	Default
1	Detection level (Normal)	
2	Detection level (High)	
3	Detection level (Very high)	

9. *auditTemperature*
Decides whether the measured temperature is recorded in the log or not.

10. *useRejectSound*
Decides whether it sounds when rejecting a user due to thermalThreshold or maskDetectionLevel.

11. *useOverlapThermal*
Displays a thermal image overlaid on the screen.

12. *useDynamicROI*
When set to true, when measuring temperature, the user's forehead is found and measured, not a fixed area.

13. *faceCheckOrder*
It defines the sequence of thermal check and mask detection and authentication.
Because the user should touch the device in the case of ID combination authentication or PIN combination authentication,
it is important to decide whether the device authenticates before all check modes or afterward especially in a high-risk environment.

Value	Description	Default

Authentication before Temperature check or Mask detection check

0	Check after authentication	Default
1	Check before authentication	
2	No authentication, check only	

BS2ThermalCameraConfig

```
typedef struct {
    uint8_t distance;
```

```
    uint8_t emissionRate;

    struct {
        uint16_t x;
        uint16_t y;
        uint16_t width;
        uint16_t height;
    } roi;

    uint8_t useBodyCompensation;
    int8_t compensationTemperature;
} BS2ThermalCameraConfig;
```

1. *distance*
The distance measured by the thermal imaging camera. The unit is cm, and the default is 100.

2. *emissionRate*
The emissivity of the subject reflecting heat.
It is recommended to enter within the [95/97/98] range. If the subject is a human, 98 is recommended.

3. *roi*
ROI (Region of interest) refers to the region of interest.
It can be specified through coordinates (x, y) and range (width, height) values when measuring temperature on the face.

4. *useBodyCompensation*
It decides whether to use the compensate the body temperature.

5. *compensationTemperature*
There may be a slight difference between the actual body temperature and the body temperature measurement using the camera, and you can correct the difference by setting a value here.
It must be as the value multiplied by 10 of the temperature to be set. The value is available -50 ~ +50

BS2BarcodeConfig

```
typedef struct {
    uint8_t useBarcode;
    uint8_t scanTimeout;
    uint8_t bypassData;
    uint8_t treatAsCSN;

    uint8_t useVisualBarcode;
    uint8_t motionSensitivity;
    uint8_t visualCameraScanTimeout;
    uint8_t reserved[9];
```

```
} BS2BarcodeConfig;
```

1. *useBarcode*
`Supports XS2-QR models only` Barcode usage flag.

2. *scanTimeout*
Set the Barcode scan time. The unit is in seconds.
The default is 4 seconds, and can be entered within a range of 4 to 10 seconds.

Value	Macro	Description
4	BS2_BARCODE_TIMEOUT_DEFAULT	Default
4	BS2_BARCODE_TIMEOUT_MIN	Min Value
10	BS2_BARCODE_TIMEOUT_MAX	Max Value

3. *bypassData*
[+2.8.2] Used to send read barcode information to the server, not processed by the device.
If the barcode value is stored in the user information structure for user authentication,
There is a size constraint of 32 bytes (BS2CSNCard data)
Call the BS2_SetBarcodeScanListener, use this option to send barcodes up to 512 bytes of size to the server.

4. *treatAsCSN*
[+2.8.2] Indicates whether the Barcode should be treated the same as a regular CSN card.
It is applied from XS2-QR 1.1.3 and in the case of false, it is treated the same as before.
This allows you to freely specify character sets that can be treated as barcodes from ASCII codes 32 to 126. (See description in BS2_WriteQRCode)
If set to true, the barcode is treated like a number just like the existing CSN.
Therefore, if you want to set the bar code card data with special characters and English characters.
In this case, only the card type may be different, and the CSN card and barcode data may be used in the same value.

5. *useVisualBarcode*
[+2.9.1] Visual barcode usage flag.

Supported devices	Firmware
XS2-Finger	V1.2.0
XS2-Card	V1.2.0
BS3	V1.1.0

Visual barcode scans QR code with a general visual camera instead of a QR code sensor,
and a separate license activation is required to use this feature.
License activation is supported through BS2_EnableDeviceLicense.

6. *motionSensitivity*
[+2.9.1] Set the sensitivity of motion sensor for visual barcode.

Value	Macro	Description
0	BS2_MOTION_SENSITIVITY_LOW	Low
1	BS2_MOTION_SENSITIVITY_NORMAL	Normal
2	BS2_MOTION_SENSITIVITY_HIGH	High

7. *visualCameraScanTimeout*
[+2.9.1] Set the scan time for the visual camera. Units are seconds.
The default is 10 seconds, and can be entered within a range of 3 to 20 seconds.

Value	Macro	Description
10	BS2_VISUAL_BARCODE_TIMEOUT_DEFAULT	Default
3	BS2_VISUAL_BARCODE_TIMEOUT_MIN	Min Value
20	BS2_VISUAL_BARCODE_TIMEOUT_MAX	Max Value

8. *reserved*
Reserved Space.

BS2InputConfigEx

```
typedef struct {
    uint8_t    numInputs;
    uint8_t    numSupervised;
    uint8_t    reserved[18];

    struct {
        uint8_t    portIndex;
        uint8_t    switchType;
        uint16_t   duration;

        uint8_t    reserved;
        uint8_t    supervisedResistor;
        uint8_t    reserved1[16];

        uint8_t    reserved2[26];
    } inputs[BS2_MAX_INPUT_NUM_EX];

    uint8_t    reserved2[200];
} BS2InputConfigEx;
```

1. *numInputs*
Number of Input port.

2. *numSupervised*
Number of supervised input port.

3. *reserved*
Reserved Space.

4. *portIndex*
Input Port Number.

5. *switchType*
Input Signal Type.

Value	Description
0	Normally Open
1	Normally Closed

6. *duration*
Input Signal Duration Time Measurement is milliseconds(ms).

7. *reserved*
Reserved Space.

8. *supervisedResistor*
You can set Supervised input resistance value type or unsupervise it.

Value	Description
0	1K Resistance
1	2.2K Resistance
2	4.7K Resistance
3	10K Resistance
254	Unsupervised(Default)

9. *reserved1*
Reserved Space.

10. *reserved2*
Reserved Space.

11. *reserved2*
Reserved Space.

BS2RelayActionConfig

```
typedef struct {
    uint32_t        deviceID;             ///< 4 bytes
    uint8_t         reserved[16];          ///< 16 bytes

    struct {
        uint8_t     port;              ///< 1 byte (relay port)
        uint8_t     reserved0;                ///< 1 byte
        uint8_t     disconnEnabled;           ///< 1 byte (RS485
disconnection)
        uint8_t     reserved[9];           ///< 9 bytes

        struct {
            uint8_t port;              ///< 1 byte (input port)
            uint8_t type;              ///< 1 byte (linkage/latching/release)
            uint8_t mask;              ///< 1 byte (alarm/fault)
            uint8_t reserved[9];           ///< 9 bytes
        } input[BS2_MAX_RELAY_ACTION_INPUT];       ///< 192 bytes
```

```
    } relay[BS2_MAX_RELAY_ACTION];              ///< 816 bytes

    uint8_t             reserved2[152];         ///< 152 bytes
} BS2RelayActionConfig;
```

1. *deviceID*
Device Identifier

2. *reserved*
Reserved Space.

3. *relay*
Relay Setting Information

4. *port*
Relay port Number.

5. *reserved0*
Reserved Space.

6. *disconnEnabled*
If set to true, a signal is made when RS485 is disconnected.

7. *reserved*
Reserved Space.

8. *input*
Defines to which input ports the relay ports will take action.

9. *port*
Input port Identifier.

10. *type*
Defines in which input type the input will take action.
If set to Linkage, signal can be made when alarm is set to mask.

type	Value	Description
NONE	0	OFF
LINKAGE	1	Connect to the relay of the input
LATCHING	2	Not Supported
RELEASE	3	Not Supported

11. *mask*
Set mask to Input Signal Info.

type	Value	Description
NONE	0	OFF
ALARM	1	Signal Made
FAULT	2	Signal Made when disconnected

12. *reserved*
Reserved Space.

13. *reserved2*
Reserved Space.

BS2VoipConfigExt

```
typedef struct {
    BS2_USER_ID phoneNumber;
    char description[48 * 3];

    uint8_t reserved[32];
} BS2ExtensionNumber;

typedef struct {
    BS2_BOOL enabled;
    BS2_BOOL useOutboundProxy;
    uint16_t registrationDuration;

    BS2_URL address;
    BS2_PORT port;

    struct {
        uint8_t speaker;      // 0 ~ 100
        uint8_t mic;          // 0 ~ 100
    } volume;             ///< 2 bytes

    BS2_USER_ID id;
    BS2_USER_ID password;
    BS2_USER_ID authorizationCode;

    struct {
        BS2_URL address;
        BS2_PORT port;
        uint8_t reserved[2];
    } outboundProxy;

    uint8_t exitButton;            /// *, #, 0~9
    uint8_t reserved1;
    uint8_t numPhoneBook;
    BS2_BOOL showExtensionNumber;

    BS2ExtensionNumber phonebook[128];

    uint8_t reserved2[32];        ///< 32 bytes (reserved)
} BS2VoipConfigExt;
```

1. *phoneNumber*

This is the extension.

2. *description*
Display information.

3. *reserved*
Reserved space.

4. *enabled*
Sets whether the VoIP extension feature is enabled.

5. *useOutboundProxy*
Sets whether the Outbound Proxy Server is configured.

6. *registrationDuration*
The cycle of updating the relevant information to the SIP server.
Set in seconds and must be between 60 and 600.

7. *address*
Enter the IP address of the SIP server (usually BioStar).

8. *port*
Enter the SIP server port. The default port is 5060.

9. *speaker*
Enter the speaker volume information for the intercom in the range 0 to 100. The default value is 50.

10. *mic*
Enter the microphone volume information for the intercom in the range 0 to 100. The default value is 50.

11. *id*
Enter the ID to connect to the SIP server.
12. *password*
Specifies the password to connect to the SIP server.

13. *authorizationCode*
The authentication code value required to connect to the SIP server.

14. *outboundProxy*
Enter Outbound proxy server information.

15. *address*
Enter the IP address of the Outbound Proxy Server.

16. *port*
Enter the Outbound Proxy Server port.

17. *reserved*
Reserved space.

18. *exitButton*
Button symbol to be used as a check-out button.

Value	Description
*	'*' ASCII code 42
#	'#' ASCII code 35
0~9	'0'~'9' ASCII code (48~57)

19. *reserved1*
Reserved space.

20. *numPhoneBook*
Number of phone books.

21. *showExtensionNumber*
Determines whether to show the phone book.

22. *phonebook*
You can specify up to 128 extensions in your phone book.

23. *reserved2*
Reserved space.

BS2RtspConfig

```
typedef struct {
    BS2_USER_ID id;
    BS2_USER_ID password;

    BS2_URL address;

    BS2_PORT port;
    BS2_BOOL enabled;
    uint8_t reserved;

    uint8_t reserved2[32];
} BS2RtspConfig;
```

1. *id*
Account information when connecting to the RTSP server.

2. *password*
Password when connecting to the RTSP server.

3. *address*
Enter the address of the RTSP server.

4. *port*

Enter the RTSP server connection port. The default port is 554.

5. *enabled*
Sets whether an RTSP connection is enabled.

6. *reserved*
Reserved space.

7. *reserved2*
Reserved space.

BS2License

```
typedef struct {
    uint8_t                 index;
    uint8_t                 hasCapability;
    uint8_t                 enable;
    uint8_t                 reserved;
    BS2_LICENSE_TYPE        licenseType;
    BS2_LICENSE_SUB_TYPE    licenseSubType;
    uint32_t                enableTime;
    uint32_t                expiredTime;
    uint32_t                issueNumber;
    uint8_t                 name[BS2_USER_ID_SIZE];
} BS2License;
```

1. *index*
License index.

2. *hasCapability*
Whether the device supports that license.
It usually has a value of 1.

3. *enable*
Whether the license is active.

4. *reserved*
Reserved Space.

5. *licenseType*
The type of license.

Value	Description
0x0000	None
0x0001	Visual QR

6. *licenseSubType*
Detailed form of licenseType

Value	Description
0	None
1	Visual QR (CodeCorp)

7. *enableTime*
License activation start time, expressed in POSIX time.

8. *expiredTime*
License activation end time, 0 means unlimited.

9. *issueNumber*
Issuing unique number.

10. *name*
License name.

BS2LicenseConfig

```
typedef struct {
    uint8_t         version;
    uint8_t         numOfLicense;
    uint8_t         reserved[2];
    BS2License      license[BS2_MAX_LICENSE_COUNT];
    uint8_t         reserved1[16];
} BS2LicenseConfig;
```

1. *version*
Version of the license settings information.

2. *numOfLicense*
Number of licenses registered

3. *reserved*
Reserved Space.

4. *license*
License information and can be set up to 16.

5. *reserved1*
Reserved Space.

BS2BarcodeConfig

```
typedef struct {
    uint8_t useBarcode;
    uint8_t scanTimeout;
```

```
    uint8_t bypassData;
    uint8_t treatAsCSN;

    uint8_t useVisualBarcode;
    uint8_t motionSensitivity;
    uint8_t visualCameraScanTimeout;
    uint8_t reserved[9];
} BS2BarcodeConfig;
```

1. *useBarcode*
`Supports XS2-QR models only` Barcode usage flag.

2. *scanTimeout*
Set the Barcode scan time. The unit is in seconds.
The default is 4 seconds, and can be entered within a range of 4 to 10 seconds.

Value	Macro	Description
4	BS2_BARCODE_TIMEOUT_DEFAULT	Default
4	BS2_BARCODE_TIMEOUT_MIN	Min Value
10	BS2_BARCODE_TIMEOUT_MAX	Max Value

3. *bypassData*
[+2.8.2] Used to send read barcode information to the server, not processed by the device.
If the barcode value is stored in the user information structure for user authentication,
There is a size constraint of 32 bytes (BS2CSNCard data)
Call the BS2_SetBarcodeScanListener, use this option to send barcodes up to 512 bytes of size to the server.

4. *treatAsCSN*
[+2.8.2] Indicates whether the Barcode should be treated the same as a regular CSN card.
It is applied from XS2-QR 1.1.3 and in the case of false, it is treated the same as before.
This allows you to freely specify character sets that can be treated as barcodes from ASCII codes 32 to 126. (See description in BS2_WriteQRCode)
If set to true, the barcode is treated like a number just like the existing CSN.
Therefore, if you want to set the bar code card data with special characters and English characters.
In this case, only the card type may be different, and the CSN card and barcode data may be used in the same value.

5. *useVisualBarcode*
[+2.9.1] Visual barcode usage flag.

Supported devices	Firmware
XS2-Finger	V1.2.0
XS2-Card	V1.2.0
BS3	V1.1.0

Visual barcode scans QR code with a general visual camera instead of a QR code sensor,
and a separate license activation is required to use this feature.
License activation is supported through BS2_EnableDeviceLicense.

6. *motionSensitivity*
[+2.9.1] Set the sensitivity of motion sensor for visual barcode.

Value	Macro	Description
0	BS2_MOTION_SENSITIVITY_LOW	Low
1	BS2_MOTION_SENSITIVITY_NORMAL	Normal
2	BS2_MOTION_SENSITIVITY_HIGH	High

7. *visualCameraScanTimeout*
[+2.9.1] Set the scan time for the visual camera. Units are seconds.
The default is 10 seconds, and can be entered within a range of 3 to 20 seconds.

Value	Macro	Description
10	BS2_VISUAL_BARCODE_TIMEOUT_DEFAULT	Default
3	BS2_VISUAL_BARCODE_TIMEOUT_MIN	Min Value
20	BS2_VISUAL_BARCODE_TIMEOUT_MAX	Max Value

8. *reserved*
Reserved Space.

BS2OsdpStandardConfig

```
typedef struct {
    uint32_t                baudRate;            ///< 4 bytes
    uint8_t                    channelIndex;        ///< 1 byte
    uint8_t                    useRegistance;        ///< 1 byte
    uint8_t                    numOfDevices;        ///< 1 byte
    BS2_OSDP_CHANNEL_TYPE     channelType;         ///< 1 byte
    BS2OsdpStandardDevice
slaveDevices[BS2_RS485_MAX_SLAVES_PER_CHANNEL];    ///< 28 * 32 = 896 bytes
    uint8_t                    reserved[4];         ///< 4 bytes
} BS2OsdpStandardChannel;                          ///< 908 bytes

typedef struct {
    uint8_t            mode[BS2_RS485_MAX_CHANNELS_EX];        ///< 8 byte
    uint16_t       numOfChannels;                             ///< 2 byte
    uint8_t        reserved[2];                              ///< 2 bytes
(packing)
    uint8_t        reserved1[32];                             ///< 32 bytes
(reserved)
    BS2OsdpStandardChannel  channels[BS2_RS485_MAX_CHANNELS_EX];     ///<
908 * 8 bytes  = 7264 bytes
} BS2OsdpStandardConfig;                                    ///< 7308 bytes
```

1. *baudRate*
This is the baud rate of the OSDP device and the range that can be set is as follows.

Value
9600

Value
19200
38400
57600
115200

2. *channelIndex*
This is the channel number when the OSDP device communicates with RS485.

3. *useRegistance*
Registance flag - no effect on operation.

4. *numOfDevices*
Number of slave devices.

5. *channelType*
Indicates the type to which the device communicating RS485 is connected.
Based on CoreStation40, there are 5 assignable channels from 0 to 4, and Suprema devices and OSDP devices cannot be mixed and operated within each channel.
If no device is connected to a specific channel, it has a 0 indicating that it can be connected even if it is a Suprema device or an OSDP device.
If a Suprema device is connected to a specific channel, only Suprema devices are allowed to connect to that channel, and channelType has a value of 1. The OSDP device is ignored even if it is connected.
If an OSDP device is connected to a specific channel, only OSDP devices are allowed to connect to that channel, and channelType has a value of 2. The Suprema device is ignored even if it is connected.
Each channel of CoreStation40 can be mixed and operated as Suprema device channel and OSDP device channel.
The maximum number of OSDP devices allowed to connect to a channel is limited to 2, and if the channel is already maxed out, the channelType will be 3, indicating that no more connections are allowed.

Value	Description
0	Normal
1	Suprema Device
2	OSDP Device
3	OSDP Device FULL

6. *slaveDevices*
Slave device information within the channel.

7. *reserved*
Reserved Space.

8. *mode*
It is a flag that determines which mode to operate in the RS485 network. As of 2023/1/12, CoreStation40 is the only device that supports Osdp standard config, so it always has a master value.

Value	Description
0	Not used
1	Master
2	Slave
3	Standalone (Default)

9. *numOfChannels*
Number of channel. CoreStation40 has a total of 5 channels.

10. *reserved*
Reserved Space.

11. *reserved1*
Reserved Space.

12. *channels*
OSDP device information of each channel.
You can have up to 8 channel information, but since CoreStation40 has 5 channels, only numbers 0 to 4 are valid.

BS2OsdpStandardActionConfig

```
typedef struct{
    BS2_BOOL                            use;            ///< 1 byte
    uint8_t                             readerNumber;   ///< 1 byte
    uint8_t                             ledNumber;      ///< 1 byte

    BS2_OSDP_STANDARD_LED_COMMAND       tempCommand;    ///< 1 byte
    uint8_t                             tempOnTime;     ///< 1 byte
    uint8_t                             tempOffTime;    ///< 1 byte
    BS2_OSDP_STANDARD_COLOR             tempOnColor;    ///< 1 byte
    BS2_OSDP_STANDARD_COLOR             tempOffColor;   ///< 1 byte
    uint16_t                            tempRunTime;    ///< 2 bytes

    BS2_OSDP_STANDARD_LED_COMMAND       permCommand;    ///< 1 byte
    uint8_t                             permOnTime;     ///< 1 byte
    uint8_t                             permOffTime;    ///< 1 byte
    BS2_OSDP_STANDARD_COLOR             permOnColor;    ///< 1 byte
    BS2_OSDP_STANDARD_COLOR             permOffColor;   ///< 1 byte

    uint8_t                             reserved;       ///< 1 byte
} BS2OsdpStandardLedAction;             ///< 16 bytes

typedef struct {
    BS2_BOOL                    use;            ///< 1 byte
    uint8_t                     readerNumber;   ///< 1 byte
    BS2_OSDP_STANDARD_TONE      tone;           ///< 1 byte
    uint8_t                     onTime;         ///< 1 byte
```

```
    uint8_t                     offTime;        ///< 1 byte
    uint8_t                     numOfCycle;     ///< 1 byte
    uint8_t                     reserved[2];    ///< 2 bytes
} BS2OsdpStandardBuzzerAction;                  ///< 8 bytes

typedef struct {
    BS2_OSDP_STANDARD_ACTION_TYPE   actionType;     ///< 1 byte
    uint8_t                         reserved[3];    ///< 3 bytes
    BS2OsdpStandardLedAction        led[2];         ///< 16 x 2 = 32 bytes
    BS2OsdpStandardBuzzerAction     buzzer;         ///< 8 bytes
} BS2OsdpStandardAction;                            ///< 44 bytes

typedef struct
{
    uint8_t                 version;                ///< 1 byte
    uint8_t                 reserved[3];            ///< 3 byes
    BS2OsdpStandardAction   actions[BS2_OSDP_STANDARD_ACTION_MAX_COUNT];
///< 44 x 32  = 1408
} BS2OsdpStandardActionConfig;                      ///< 1412 bytes
```

1. *use*
Indicates whether to use LED action.

2. *readerNumber*
The sequence number of the OSDP device.

3. *ledNumber*
The LED sequence number of the OSDP device.

4. *tempCommand*
Temporary command.

Value	Description
0	No Operation
1	Cancel
2	Set

5. *tempOnTime*
Indicates the LED on time for Temporary command, set in units of 100 ms.
For example, enter 20 to keep the LED on for 2 seconds.

6. *tempOffTime*
Indicates the LED off time for Temporary command, set in units of 100 ms.
For example, enter 10 to turn off the LED for 1 second.

7. *tempOnColor*
Sets the LED color of the on state for Temporary command.

Value	Description
0	BLACK
1	RED

Value	Description
2	GREEN
3	AMBER
4	BLUE
5	MAGENTA
6	CYAN
7	WHITE

8. *tempOffColor*
Sets the LED color of the off state for Temporary command.

Value	Description
0	BLACK
1	RED
2	GREEN
3	AMBER
4	BLUE
5	MAGENTA
6	CYAN
7	WHITE

9. *tempRunTime*
Sets the LED On/Off time for Temporary commands in units of 100 ms.
It blinks alternately with the color and time set in tempOnTime/tempOffTime, tempOnColor/tempOffColor, and is maintained as long as the value of tempRunTime.

10. *permCommand*
Permanent command. 11. *permOnTime*
Indicates the LED on time for Permanent command, set in units of 100 ms.

12. *permOffTime*
Indicates the LED off time for Permanent command, set in units of 100 ms.

13. *permOnColor*
Sets the LED color of the on state for Permanent command.

14. *permOffColor*
Sets the LED color of the off state for Permanent command.

15. *reserved*
Reserved Space.

16. *use*
Indicates whether to use tone action.

17. *readerNumber* The sequence number of the OSDP device.

18. *tone*
Set the buzzer.

Value	Description
0	None
1	Off
2	On

19. *onTime*
Set the on-state holding time for tone in units of 100 ms.

20. *offTime*
Set the off state holding time for tone in units of 100 ms.

21. *numOfCycle*
Set the number of times to repeat On/Off for tone. When set to 0, it means infinite repetition.

22. *reserved*
Reserved Space.

23. *actionType*
Set the action.

Value	Description
0	None
1	Success
2	Fail
3	Wait input

24. *reserved*
Reserved Space.

25. *led*
LED setting information of OSDP device.

26. *buzzer*
Buzzer setting information of OSDP device.

27. *version*
Version information about Action configuration. Currently it is 0.

28. *reserved*
Reserved Space.

29. *actions*
LED/buzzer information for OSDP devices, up to 32 can be designated.

BS2CustomMifareCard

```
typedef struct {
    uint8_t primaryKey[6];
```

```
    uint8_t reserved1[2];
    uint8_t secondaryKey[6];
    uint8_t reserved2[2];
    uint16_t startBlockIndex;
    uint8_t dataSize;
    uint8_t skipBytes;
    uint8_t reserved[4];
} BS2CustomMifareCard;
```

1. *primaryKey*
Primary encryption key to access the Mifare card information.

2. *reserved1*
Reserved space.

3. *secondaryKey*
Secondary encryption key to access the Mifare card information.

4. *reserved2*
Reserved space.

5. *startBlockIndex*
Start block index on the Mifare data storage.

6. *dataSize*
The size in bytes of the card data.

7. *skipBytes*
This is where the card data appears.
This is the starting point to read card data. It is 0 when reading from the starting point, and indicates the number of bytes skipped after the first.

8. *reserved*
Reserved space.

BS2CustomDesFireCard

```
typedef struct {
    uint8_t primaryKey[16];
    uint8_t secondaryKey[16];
    uint8_t appID[3];
    uint8_t fileID;
    uint8_t encryptionType;                 // 0: DES/3DES, 1: AES
    uint8_t operationMode;                  // 0: legacy(use picc master
key), 1: new mode(use app master, file read, file write key)
    uint8_t dataSize;
    uint8_t skipBytes;
    uint8_t reserved[4];
    BS2DesFireAppLevelKey desfireAppKey;        ///<52 bytes
```

```
} BS2CustomDesFireCard;                                    ///<96 Bytes
```

1. *primaryKey*
Primary encryption key to access the DesFire card information. (General settings)

2. *secondaryKey*
Secondary encryption key to access the Desfire card information. (General settings)

3. *appID*
Application Id that is stored inside the DesFire card for user authentication.

4. *fileID*
File ID that is stored inside the DesFire card, which will be used by the application to read and write data.

5. *encryptionType*
Type of data encryption.

Value	Description
0	DES/3DES
1	AES

6. *operationMode*
Operation mode.

Value	Description
0	Using general settings (Using PICC master key)
1	Using advanced settings (Using App master key)

7. *dataSize*
The size in bytes of the card data.

8. *skipBytes*
This is where the card data appears.
This is the starting point to read card data. It is 0 when reading from the starting point, and indicates the number of bytes skipped after the first.

9. *reserved*
Reserved space.

10. *desfireAppKey*
Indicates key information to access DesFire card information. (Advanced settings)

BS2CustomCardConfig

```
typedef struct {
    BS2_CARD_DATA_TYPE dataType;
```

```
    BS2_BOOL useSecondaryKey;
    uint8_t reserved1[2];

    BS2CustomMifareCard mifare;
    BS2CustomDesFireCard desfire;
    uint8_t reserved2[24];
    uint8_t reserved3[96];

    BS2_CARD_BYTE_ORDER smartCardByteOrder;
    uint8_t reserved4[3];
    BS2_UID formatID;
    uint8_t reserved5[8];
} BS2CustomCardConfig;
```

1. *dataType*
Type of card data.

Value	Description
0	Binary
1	ASCII
2	UTF16
3	BCD

2. *useSecondaryKey*
Decides whether to use the secondary encryption key.

3. *reserved1*
Reserved space.

4. *mifare*
Set the Mifare custom card information.

5. *desfire*
Set the DESFire custom card information.

6. *reserved2*
Reserved space.

7. *reserved3*
Reserved space.

8. *smartCardByteOrder*
The output method can be selected from MSB or LSB.

Value	Description
0	MSB
1	LSB

9. *reserved4*
Reserved space.

10. *formatID*
This is an identifier that can be used when the BioStar 2 application needs to manage the card configuration as a database.

11. *reserved5*
Reserved space.

[1)]
Maximum Transmission Unit
[2)]
Most Significant Bit
[3)]
Least Significant Bit