Table of Contents

Smartcard API 1
Structure 1
BS2CSNCard 1
BS2SmartCardHeader 2
BS2SmartCardCredentials 4
BS2AccessOnCardData 5
BS2SmartCardData 5
BS2Card 6

Smartcard API

API that provides a function that reads and writes card data.

- BS2_ScanCard: Scans the card from the device and analyzes it.
- BS2_WriteCard: Writes data to the smart card.
- BS2_EraseCard: Formats the smart card.

Structure

BS2CSNCard

```
typedef struct {
    uint8_t type;
    uint8_t size;
    uint8_t data[BS2_CARD_DATA_SIZE];
} BS2CSNCard;
```

1. *type*
The code value of card type. The card type is to indicate the purpose of the card. When transferring a user from device to server, the Access card will be used only to keep the issue history, since the Access card will work on its own without any user information.

Value	Description	Used Format
0x00	Unknown card	
0x01	CSN card	
0x02	Secure card	
0x03	Access card	
0x06	QR Code	
0x0A	Wiegand card	BS2WiegandConfig.format (This format is used when BS2WiegandConfig.CSNIndex and BS2WiegandConfig.CardMask is set as 0
0x1A	Wiegand card	BS2WiegandMultiConfig.formats[0]
0x2A	Wiegand card	BS2WiegandMultiConfig.formats[1]
0x3A	Wiegand card	BS2WiegandMultiConfig.formats[2]
0x4A	Wiegand card	BS2WiegandMultiConfig.formats[3]
0x5A	Wiegand card	BS2WiegandMultiConfig.formats[4]
0x6A	Wiegand card	BS2WiegandMultiConfig.formats[5]
0x7A	Wiegand card	BS2WiegandMultiConfig.formats[6]
0x8A	Wiegand card	BS2WiegandMultiConfig.formats[7]
0x9A	Wiegand card	BS2WiegandMultiConfig.formats[8]
0xAA	Wiegand card	BS2WiegandMultiConfig.formats[9]
0xBA	Wiegand card	BS2WiegandMultiConfig.formats[10]
0xCA	Wiegand card	BS2WiegandMultiConfig.formats[11]
0xDA	Wiegand card	BS2WiegandMultiConfig.formats[12]
0xEA	Wiegand card	BS2WiegandMultiConfig.formats[13]

Value	Description	Used Format
0xFA	Wiegand card	BS2WiegandMultiConfig.formats[14]

2. *size*
This is the size of the data buffer. This field needs to be filled with a fixed value as '32'.

3. *data*
The data of card template.
In case of Secure Credential Card(SCC), users need to have card information which includes Card ID(24byte), issueCount(4byte) and TimeStamp(4byte). Also, cardObjs array of BS2UserBlob structure should be filled for SCC cards and the user should be updated after SCC issuing.

> **Tip**
> **You can use any value as long as the data and the cardObj in BS2UserBlob match.**
> TimeStamp or issueCount is an operational flag for better management but there's no validation check for TimeStamp or issueCount on the device.

BS2SmartCardHeader

```
typedef struct {
    uint16_t hdrCRC;
    uint16_t cardCRC;
    BS2_CARD_TYPE cardType;
    uint8_t numOfTemplate;
    uint16_t templateSize;
    uint16_t issueCount;
    uint8_t duressMask;
    uint8_t cardAuthMode;
    uint8_t useAlphanumericID;
    uint8_t cardAuthModeEx;
    uint8_t numOfFaceTemplate;
    uint8_t reserved[1];
} BS2SmartCardHeader;
```

1. *hdrCRC*
Value of card header checksum. (cardCRC - reserved)

2. *cardCRC*
Value of card data checksum. (BS2SmartCardHeader.cardType - BS2SmartCardData.accessOnData)

3. *cardType*
Code of card types.

Value	Description
0x00	Unknown card
0x01	CSN card
0x02	Secure card
0x03	Access card
0x0A	Wiegand card
0x0B	Config card

4. *numOfTemplate*
Number of templates.

> Based on AoC structure, template data is stored in BS2SmartCardCredentials.
> Here you can store either fingerprint or face template. Both fingerprint and face templates can not be stored together.
> Thus, you have to set numOfFacetemplate = 0 if you want to store fingerprint template data.

5. *templateSize*
Size of the template. A normal fingerprint template is a fixed 384 byte.
If you are using a smart card the default in BioStar 2 is 300 bytes and you can change as required but we recommend that you set it above 300 bytes because if the template size is too small it can cause fingerprint matching issues because of the lack of information in the template.

6. *issueCount*
Number of smart card issue count.

7. *duressMask*
Mask for whether there is a duress finger.
8. *cardAuthMode*
`Other devices` Card authentication mode.
`FaceStation F2` Please use **cardAuthModeEx** instead

Value	Description
2	Card only
3	Card + Fingerprint
4	Card + PIN
5	Card + Fingerprint or PIN
6	Card + Fingerprint + PIN
254	Cannot use
255	Not defined(System defined mode)

9. *useAlphanumericID*
Flag of usage of Alphanumeric ID
10. *cardAuthModeEx*
`Other devices` Please use **cardAuthMode** instead
[+ V2.7.1] `FaceStation F2` Card authentication mode

Value	Level 1	Level 2	Level 3
21	Card		
22	Card	Face	
23	Card	Fingerprint	
24	Card	PIN	
25	Card	Face or Fingerprint	
26	Card	Face or PIN	
27	Card	Fingerprint or PIN	
28	Card	Face or Fingerprint or PIN	
29	Card	Face	Fingerprint
30	Card	Face	PIN
31	Card	Fingerprint	Face
32	Card	Fingerprint	PIN
33	Card	Face or Fingerprint	PIN
34	Card	Face	Fingerprint or PIN
35	Card	Fingerprint	Face or PIN
254	Cannot use		
255	Not defined(System defined mode)		

11. *numOfFaceTemplate*
Number of face templates.
Basic size of template is different between fingerprint and face(Fingerprint:384, Face:552).
You can consider the total size of templateData in BS2SmartCardCredentials.

> Based on AoC structure, template data is stored in BS2SmartCardCredentials.
> Here you can store either fingerprint or face template. Both fingerprint and face templates can not be stored together.
> Thus, you have to set numOftemplate = 0 if you want to store face template data.

12. *reserved*
Reserved

BS2SmartCardCredentials

```
typedef struct {
    uint8_t pin[BS2_PIN_HASH_SIZE];
    uint8_t templateData[BS2_SMART_CARD_MAX_TEMPLATE_COUNT *
BS2_FINGER_TEMPLATE_SIZE];
} BS2SmartCardCredentials;
```

1. *pin*
Value of PIN.

2. *templateData*
List of fingerprint template data area, which can be stored up to 4 fingerprint templates.

BS2AccessOnCardData

```
typedef struct {
    uint16_t accessGroupID[BS2_SMART_CARD_MAX_ACCESS_GROUP_COUNT];
    BS2_DATETIME startTime;
    BS2_DATETIME endTime;
} BS2AccessOnCardData;
```

1. *accessGroupID*
List of access group IDs.

2. *startTime*
Starting time where a user can authenticate. When it is set as 0, there are no limits.

3. *endTime*
Ending time where a user can authenticate. When it is set as 0, there are no limits.

BS2SmartCardData

```
typedef struct {
    BS2SmartCardHeader header;
    uint8_t cardID[BS2_CARD_DATA_SIZE];
    BS2SmartCardCredentials credentials;
    BS2AccessOnCardData accessOnData;
} BS2SmartCardData;
```

1. *header*
Smart card header.

2. *cardID*
Card ID that will be used on the card.
Access on Cards will need to use the 32 byte array for the card ID.
Secure Credential Cards will need to use a 24 byte array for the card ID.
In case of Secure Credential Card(SCC), users need to have card information which includes Card ID(24byte), issueCount(4byte) and TimeStamp(4byte).
Also, cardObjs array of BS2UserBlob structure should be filled for SCC cards and the user should be updated after SCC issuing.

3. *credentials*
Authentication data area where the PIN or fingerprint template is stored.

4. *accessOnData*

Data area the AOC card uses, which carries the access group information.

BS2Card

```
typedef struct {
    uint8_t isSmartCard;
    union {
        BS2CSNCard card;
        BS2SmartCardData smartCard;
    };
}BS2Card;
```

1. *isSmartCard*
Decides whether it is a smart card.

2. *card*
CSN card data.

3. *smartCard*
Smart card data.

From:
https://kb.supremainc.com/kbtest/ - **BioStar 2 Device SDK**

Permanent link:
https://kb.supremainc.com/kbtest/doku.php?id=en:smartcard_api

Last update: **2021/09/14 14:37**