Communications Protocol Manual

(Ver 2.3)

Waveshare International Limited

Contents

1. Communication Protocol

1.1 Communication Processing Procedure

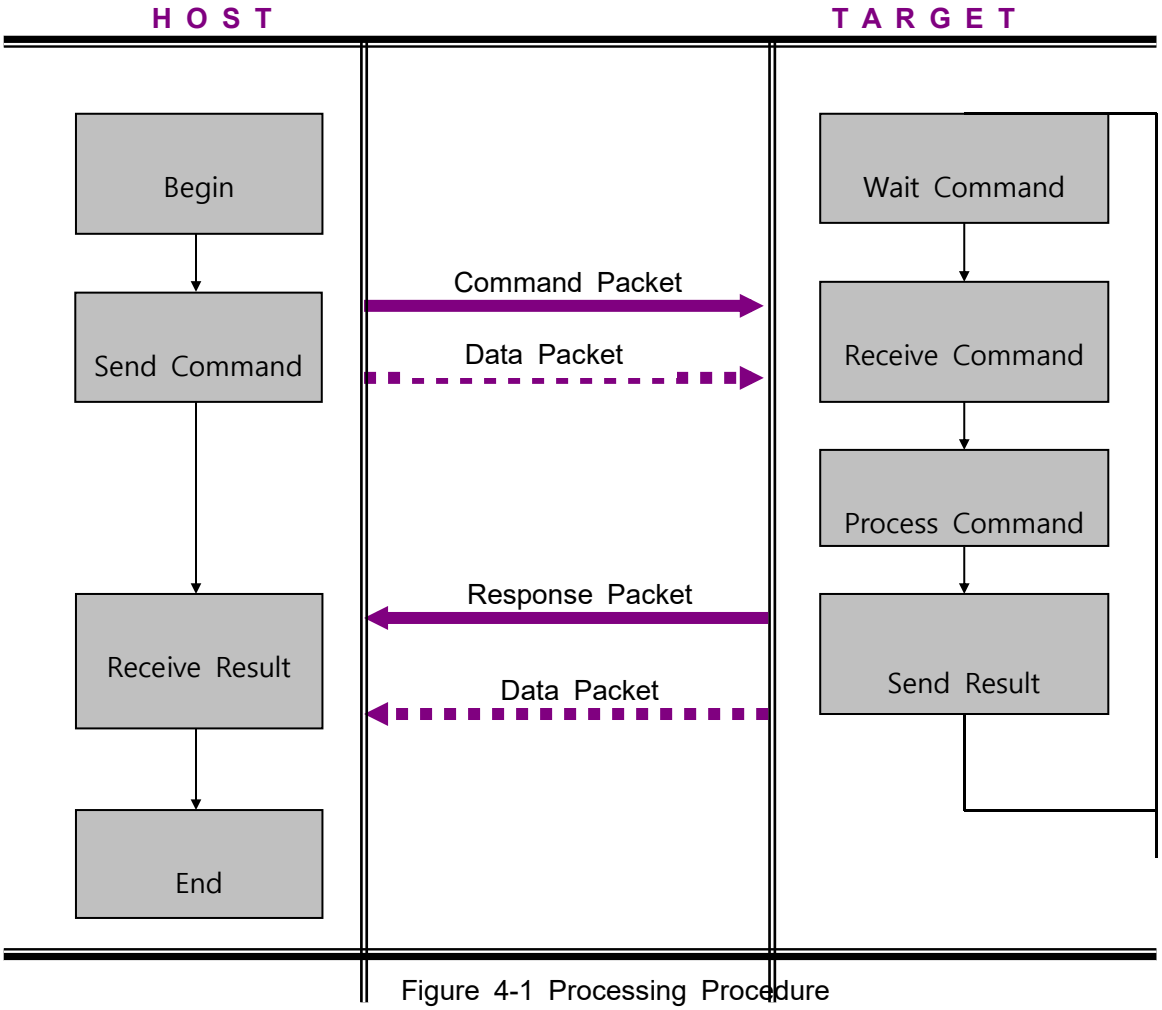


Figure 4-1 Processing Procedure

Note:

All commands must be sent and received in accordance with the principle of one send a nd one receive. If the host does not receive a reply, do not send commands to TARGET.

1.2 Data Structure

1.2.1 Packet Identifier (Little-endian mode)

```
#define XG_PREFIX_CODE                0xAABB //Little-endian mode, send BB first then send AA
```

1.2.2 Packet Structure

```
/*The packet data structure is 24 bytes*/
typedef struct _XG_PACKET_
{
        unsigned short   wPrefix; //Packet identifier xAABB or xBBAA, the big-endian and little-endian mode are different, 2 bytes
        unsigned char    bAddress; //Device address, used when multiple devices are connected to the bus, 0 for a single device, 1 byte
        unsigned char    bCmd; //Command code, 1 byte
        unsigned char    bEncode; //Data encoding, 1 byte
        unsigned char    bDataLen; //Valid data length, 1 byte, value range is 1-16
        unsigned char    bCmd; //Packet data, 16 bytes
        unsigned short   wCheckSum; //Packet check, 2 bytes, sum of 1-22 bytes, little-endian mode
} XG_PACKET, *PXG_PACKET;
```

Example:

Connection command packet data:

Send: BB AA 00 01 00 08 30 30 30 30 30 30 30 30 00 00 00 00 00 00 00 00 EE 02

Receive: BB AA 00 01 00 10 00 37 2D 31 B0 E0 00 00 00 00 00 00 00 00 00 00 02 9D 03

1.2.3 Packet Validation

Calculation method:

The packet checksum wCheckSum is calculated byte-by-byte from wPrefix to bData, and the lowest 2 bytes of the calculation result are taken.

Code example:

```
/*
Function: Calculate checksum
pBuf: The first address of the buffer that needs to calculate the checksum
Size: Buffer length for calculating checksum
Return value:
2-byte checksum, you need to pay attention to the difference between the big-endian and little-endian for different systems
```

```
*/
UINT16 GetCheckSum(UINT8* pBuf, UINT16 Size)
{
	UINT16	w_Ret = 0;
	UINT16 i;
	UINT8* p = (UINT8*)&w_Ret;

	for (i = 0; i < Size; i++ )
		w_Ret += pBuf[i];
	//Big-endian to little-endian conversion
	//w_Ret = p[0] + (p[1]<<8);
	return w_Ret;
}
```

1.2.4 Send Command Packet

```
/*
Function: Send command packets via UART port
bCmd: Command code
pData: The data that needs to be sent in the command packet
Len: Valid data length
Return value:
XG_ERR_SUCCESS: Sent successfully
*/

UINT8 SendPack(UINT8 bCmd, UINT8* pData, UINT8 Len)
{
	XG_PACKET pack = { 0 };

	pack.wPrefix = XG_PREFIX_CODE;
	pack.bCmd = bCmd;
	pack.bDataLen = Len;
	if(pData && Len > 0)
	{
		UINT8 i;
		for(i = 0; i < Len; i++) pack.bData[i] = pData[i];
	}
```

```
	pack.wCheckSum = GetCheckSum((UINT8*)&pack, sizeof(XG_PACKET) - 2);
	UartSendBuf((UINT8*)&pack, sizeof(XG_PACKET));
	return XG_ERR_SUCCESS;
}
```

1.2.5 Receive Command Packet and Return Packet

```
/*
Function: Receive a command packet from UART port
pCmd: Return a pointer to the command code
pData: Command packet data output buffer, 16 bytes
Timeout: Reception timeout, in milliseconds
Return value:
XG_ERR_SUCCESS: Received successfully
XG_ERR_TIME_OUT: Reception timeout
XG_ERR_DATA: Data error
*/

UINT8 RecvPack(UINT8* pCmd, UINT8* pData, int Timeout)
{
	XG_PACKET pack = { 0 };

	UINT8 ret = UartRecvBuf((UINT8*)&pack, (UINT8)sizeof(XG_PACKET), Timeout);
	if(ret >= sizeof(XG_PACKET))
	{
		UINT16 CheckSum = 0;

		if(pack.wPrefix != XG_PREFIX_CODE) return XG_ERR_DATA;
		CheckSum = GetCheckSum((UINT8*)&pack, sizeof(XG_PACKET) - 2);
		if(CheckSum != pack.wCheckSum) return XG_ERR_DATA;
		if(pCmd) *pCmd = pack.bCmd;
		if(pData)
		{
			UINT8 i;
			for(i = 0; i < 16; i++) pData[i] = pack.bData[i];
		}
```

```
	} else {
		return XG_ERR_TIME_OUT;
	}
	return XG_ERR_SUCCESS;
}
```

1.2.6 Read Data

```
/*
Function: Read a data packet
Parameters:
bCmd: Command code
bDataBuf: The first address of the data buffer to be read
iOffset: The address offset of the data to be read
iSize: The size of the data to be read, cannot be greater than a packet size
iTimeout: The timeout for reading a packet
Return value: > 0: The size of the data that was successfully read, =0: The read failed
*/
static UINT16 ReadDataPack(UINT8 bCmd, UINT8* bDataBuf, UINT16 iOffset, UINT16 iSize, int iTimeout)
{
	UINT16 RecvSize = 0;
	//UINT8 bBuf[UART_RECV_BYTE_MAX] = { 0 }; //Initialize array
	UINT8* bBuf = NULL; //For a single-chip microcomputer, you can directly use the UART port buffer to save memory
	UINT8 bCmdData[16] = { 0 }; //Initialize array

	if(iSize > COM_DATA_PACKET_SIZE) return 0;

	bCmdData[0] = bCmd;
	bCmdData[1] = (iOffset&0xff);
	bCmdData[2] = ((iOffset>>8)&0xff);
	bCmdData[3] = 0;
	bCmdData[4] = 0;
	bCmdData[5] = (iSize&0xff);
	bCmdData[6] = ((iSize>>8)&0xff);
	bCmdData[7] = 0;
	bCmdData[8] = 0;
```

```
        SendPack(XG_CMD_READ_DATA, bCmdData, 9);
        gUartByte = 0; //Clear buffer received via UART port
        //The received data should be added with byte checksum
        RecvSize = UartRecvBuf(bBuf, iSize+2, iTimeout);
        bBuf = gUartBuf; //For a single-chip microcomputer, you can directly use the UART port buffer to save
memory
        if(RecvSize >= iSize + 2)
        {
                UINT16 CheckSum2 = (UINT16)(bBuf[iSize] + (bBuf[iSize + 1]*256));
                UINT16 CheckSum1 = GetCheckSum(bBuf, iSize);
                if(CheckSum1 == CheckSum2)
                {
                        int i;
                        for(i = 0; i < iSize; i++)
                        {
                                bDataBuf[i] = bBuf[i];
                        }
                        return iSize;
                }
        }
        return 0;
}

/*
Function: Read sub-packet data
Parameters:
bCmd: Command code
bDataBuf: The first address of the data buffer to be read
iSize: The size of the data to be read
iTimeout: The timeout for reading a packet
Return value: = 0 read succeeded, non-0 read failed
*/
static UINT8 ReadData(UINT8 bCmd, UINT8* bDataBuf, UINT16 iSize, int iTimeout)
{
        int ret = 0;
        int   lPkNum, lPkRec, lDataMove, i;
```

```
	int reReadCnt = 0;
	int PackSize = COM_DATA_PACKET_SIZE;

	//Sub-packet
	lPkNum = (iSize)/PackSize;
	lPkRec = (iSize)%PackSize;
	lDataMove = 0;

	for(i = 0; i < lPkNum; i++)
	{
		ret = ReadDataPack(bCmd, bDataBuf, lDataMove, PackSize, iTimeout);
		if(ret == PackSize)
		{
			reReadCnt = 0;
			lDataMove += PackSize;
		} else {
			//If the read fails, retry twice
			i--;
			if(reReadCnt++ > 2)
			{
				return XG_ERR_COM;
			}
		}
	}

	if(lPkRec > 0)
	{
		ret = ReadDataPack(bCmd, bDataBuf, lDataMove, lPkRec, iTimeout);
		if(ret == lPkRec)
		{
			lDataMove += lPkRec;
		}
	}
	if(lDataMove == iSize) return XG_ERR_SUCCESS;
	return XG_ERR_FAIL;
}
```

1.2.7 Write Data

```
/*
Function: Write a data packet
Parameters:
bCmd: Command code
bDataBuf: The first address of the data buffer to be written
iOffset: The address offset of the data to be written
iSize: The size of the data to be written, cannot be greater than a packet size
Return value: = 0 write succeeded, non-0 write failed
*/
static UINT8 WriteDataPack(UINT8 bCmd, UINT8* bDataBuf, UINT16 iOffset, UINT16 iSize)
{
	UINT8 ret = 0;
	UINT8 bBuf[UART_RECV_BYTE_MAX] = { 0 };
	UINT8 bCmdData[16] = { 0 };
	UINT16 i, CheckSum;

	if(iSize > COM_DATA_PACKET_SIZE) return XG_ERR_FAIL;

	bCmdData[0] = bCmd;
	bCmdData[1] = (iOffset&0xff);
	bCmdData[2] = ((iOffset>>8)&0xff);
	bCmdData[3] = 0;
	bCmdData[4] = 0;
	bCmdData[5] = (iSize&0xff);
	bCmdData[6] = ((iSize>>8)&0xff);
	bCmdData[7] = 0;
	bCmdData[8] = 0;

	ret = SendPack(XG_CMD_WRITE_DATA, bCmdData, 9);
	if(ret != XG_ERR_SUCCESS)
	{
		return ret;
	}
	for(i = 0; i < iSize; i++)
```

```
        {
                bBuf[i] = bDataBuf[i];
        }

        CheckSum = GetCheckSum(bBuf, iSize);
        bBuf[iSize] = CheckSum%256;
        bBuf[iSize+1] = CheckSum/256;
        UartSendBuf(bBuf, iSize+2);

        gUartByte = 0; //Clear buffer received via UART port
        ret = RecvPack(NULL, bCmdData, 1000);
        if(ret == XG_ERR_SUCCESS && bCmdData[0] == XG_ERR_SUCCESS)
        {
                return XG_ERR_SUCCESS;
        }

        return XG_ERR_FAIL;
}

/*
Function: Write sub-packet data
Parameters:
bCmd: Command code
bDataBuf: The first address of the data buffer to be written
iSize: The size of the data to be written
Return value: = 0 write succeeded, non-0 write failed
*/
static UINT8 WriteData(UINT8 bCmd, UINT8* bDataBuf, UINT16 iSize)
{
        int ret = 0;
        int   lPkNum, lPkRec, lDataMove, i;
        int reWriteCnt = 0;
        int PackSize = COM_DATA_PACKET_SIZE;

        //Sub-packet
        lPkNum = (iSize)/PackSize;
```

```
    lPkRec = (iSize)%PackSize;
    lDataMove = 0;

    for(i = 0; i < lPkNum; i++)
    {
        ret = WriteDataPack(bCmd, bDataBuf, lDataMove, PackSize);
        if(ret == PackSize)
        {
            reWriteCnt = 0;
            lDataMove += PackSize;
        } else {
            //If the write fails, retry twice
            i--;
            if(reWriteCnt++ > 2)
            {
                return XG_ERR_COM;
            }
        }
    }

    if(lPkRec > 0)
    {
        ret = WriteDataPack(bCmd, bDataBuf, lDataMove, lPkRec);
        if(ret == lPkRec)
        {
            lDataMove += lPkRec;
        }
    }

    if(lDataMove == iSize) return XG_ERR_SUCCESS;
    return XG_ERR_FAIL;
}
```

2. Detailed Description of Communication Commands

2.1 Command List

```
#define XG_CMD_CONNECTION              0x01 //Connect device
#define XG_CMD_CLOSE_CONNECTION        0x02 //Close connection
#define XG_CMD_GET_SYSTEM_INFO         0x03 //Get version number and setting information
#define XG_CMD_FACTORY_SETTING         0x04 //Restore factory settings
#define XG_CMD_SET_DEVICE_ID           0x05 //Set device number 0-255
#define XG_CMD_SET_BAUDRATE            0x06 //Set baudrate 0-4
#define XG_CMD_SET_SECURITYLEVEL       0x07 //Set security level 0-4
#define XG_CMD_SET_TIMEOUT             0x08 //Set finger input waiting timeout
#define XG_CMD_SET_DUP_CHECK           0x09 //Set duplicate login check 0-1
#define XG_CMD_SET_PASSWORD            0x0A //set communication password
#define XG_CMD_CHECK_PASSWORD          0x0B //Check if password is correct
#define XG_CMD_REBOOT                  0x0C //Reset and reboot the device
#define XG_CMD_SET_SAME_FV             0x0D //Check if the same finger during registration
#define XG_CMD_SET_USB_MODE            0x0E //Set USB driver mode
#define XG_CMD_GET_DUID                0x0F //Get device serial number
#define XG_CMD_FINGER_STATUS           0x10 //Detect finger placement status
#define XG_CMD_CLEAR_ENROLL            0x11 //Clear login data for specified ID
#define XG_CMD_CLEAR_ALL_ENROLL        0x12 //Clear all ID login data
#define XG_CMD_GET_EMPTY_ID            0x13 //Get empty (no login data) ID
#define XG_CMD_GET_ENROLL_INFO         0x14 //Get the total number of login users and templates
#define XG_CMD_GET_ID_INFO             0x15 //Get login information for specified ID
#define XG_CMD_ENROLL                  0x16 //Login with specified ID
#define XG_CMD_VERIFY                  0x17 //1:1 authentication or: N recognition
#define XG_CMD_IDENTIFY_FREE           0x18 //No timeout 1:1 authentication or:N recognition, 0x19 command
can be issued to interrupt
#define XG_CMD_CANCEL                  0x19 //Interrupt the currently executed command
#define XG_CMD_READ_DATA               0x20 //Read data from device
#define XG_CMD_WRITE_DATA              0x21 //Write data to device
#define XG_CMD_READ_ENROLL             0x22 //Read login data of the specified ID
#define XG_CMD_WRITE_ENROLL            0x23 //Write (overwrite) login data for the specified ID
#define XG_CMD_GET_CHARA               0x28 //Read the current collected characteristics
#define XG_CMD_READ_USER_DATA          0x29 //Read data from user's extended storage area
```

```
#define XG_CMD_WRITE_USER_DATA      0x2A //Write data to user's extended storage area
#define XG_CMD_GET_SYS_SET          0x2E //Get D700 device system setting data structure
#define XG_CMD_SET_SYS_SET          0x2F //Download D700 system settings data structure
#define XG_CMD_OPEN_DOOR            0x32 //Open the door
#define XG_CMD_READ_LOG             0x33 //Read access control log
#define XG_CMD_SET_DEVNAME          0x34 //Set device name
#define XG_CMD_GET_DATETIME         0x35 //Get access control real-time clock
#define XG_CMD_SET_DATETIME         0x36 //Set access control real-time clock
#define XG_CMD_ENROLL_EXT           0x38 //Extend voice registration
#define XG_CMD_VERIFY_EXT           0x39 //Extend voice verification
#define XG_CMD_DEL_LOG              0x3a //Delete access control log
#define XG_CMD_PLAY_VOICE           0x3b //Play voice
#define XG_CMD_REPORT               0x4E //UART port output verification packet
#define XG_CMD_GET_USER_BATCH       0x51 //Batch read user information
#define XG_CMD_SET_USER_BATCH       0x52 //Batch write user information
#define  XG_CMD_READ_CARDNO         0x53 //D700 request user to swipe card and read card number
```

2.2 Detailed Description of Commands

2.2.1 Device Connection

- **Command Code**

  #define XG_CMD_CONNECTION 0x01 //To connect a device, a password of at least 8 characters is required. The default password is all 0 (0x30)

- **Function**

  To connect to the target device, a password of at least 8 digits is required. The factory password is 8 zeros (0x30).

  Note: Except for this command, all other commands can only be used when the connection is successful.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x01	XG_CMD_CONNECTION
bEncode	0x00	Data encoding, 0
bDataLen	0x08	Number of password digits
bData	“00000000”	Device connection password
wCheckSum	0x02EE	Checksum

  Return Packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x01	XG_CMD_CONNECTION
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Connection succeeded	Return device name after successful connection
	bData[0] = XG_ERR_SUCCESS bData[1]-bData[14] are device names	
	Connection failed	Connection failure return error code
	bData[0] = XG_ERR_FAIL	

	bData[1] = XG_ERR_INVALID_PWD	
wCheckSum		Checksum

- **Example**

```
/*
Function: Send connection command via UART port and receive response
Timeout: Reception timeout, in milliseconds
Return value:
XG_ERR_SUCCESS: Connection succeeded
XG_ERR_FAIL: Connection failed
XG_ERR_INVALID_PWD: Password error
*/

UINT8 ConnectDev(int Timeout)
{
	UINT8 ret = 0;
	UINT8 bCmd = 0;
	UINT8 bData[16] = { 0 };

	SendPack(XG_CMD_CONNECTION, "00000000", 8);
	gUartByte = 0; //Clear buffer received via UART port
	ret = RecvPack(&bCmd, bData, Timeout);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bCmd == XG_CMD_CONNECTION)
		{
			if(bData[0] == XG_ERR_SUCCESS) return XG_ERR_SUCCESS;
			else return bData[1];
		}
	}
	return XG_ERR_FAIL;
}
```

2.2.2 Close Connection

- **Command Code**

  #define XG_CMD_CLOSE_CONNECTION 0x02 //Close connection

- **Function**

  Close the current connection.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x02	XG_CMD_CLOSE_CONNECTION
bEncode	0x00	Data encoding, 0
bDataLen	0x00	
bData		
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x02	XG_CMD_CLOSE_CONNECTION
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Connection closed successfully	
	bData[0] = XG_ERR_SUCCESS	
	Connection closure failed	
wCheckSum		Checksum

- **Example**

```
/*
Function: Send close connection command
Timeout: Reception timeout, in milliseconds
Return value:
XG_ERR_SUCCESS: Succeeded
XG_ERR_FAIL: Failed
*/

UINT8 CloseConnectDev(int Timeout)
{
	UINT8 ret = 0;
	UINT8 bCmd = 0;
	UINT8 bData[16] = { 0 };

	SendPack(XG_CMD_CLOSE_CONNECTION, NULL, 0);
	gUartByte = 0; //Clear buffer received via UART port
	ret = RecvPack(&bCmd, bData, Timeout);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS) return XG_ERR_SUCCESS;
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.3 Get Version Number and System Settings

- **Command Code**

  #define XG_CMD_GET_SYSTEM_INFO 0x03 //Get version number and setting information

- **Function**

  Obtain the device firmware version number and current settings parameters: device number, baud rate, security level, finger timeout detection, ID duplicate check.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x03	XG_CMD_GET_SYSTEM_INFO
bEncode	0x00	Data encoding, 0
bDataLen	0x00	
bData		
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x03	XG_CMD_GET_SYSTEM_INFO
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Obtained successfully	
	bData[0] = XG_ERR_SUCCESS bData[1] = Main version number bData[2] = Sub-version number bData[3] = Device number(0-255) bData[4] = Baud rate (0-4) bData[5] = Security level (0-4)	

	bData[6] = Timeout detection (1-255) bData[7] = Duplicate registration check (0-1) bData[8] = Same finger registration check	
	Failed to obtain	
	bData[1] = Voice output volume (0-17)	
wCheckSum		Checksum

- **Example**

```
/*
Function: Get setting information
Timeout: Reception timeout, in milliseconds
Return value:
XG_ERR_SUCCESS: Succeeded
XG_ERR_FAIL: Failed
*/

UINT8 GetDevSetting(int Timeout)
{
	UINT8 ret = 0;
	UINT8 bCmd = 0;
	UINT8 bData[16] = { 0 };

	SendPack(XG_CMD_GET_SYSTEM_INFO, NULL, 0);
	gUartByte = 0; //Clear buffer received via UART port
	ret = RecvPack(NULL, bData, Timeout);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			int iBaud[5] = {9600, 19200, 38400, 57600, 115200};
			UINT8 Mver = bData[1];
```

```
                    UINT8 Sver = bData[2];
                    UINT8 DevID = bData[3];
                    int Baud = iBaud[bData[4]];
                    UINT8 Securty = bData[5];
                    UINT8 Timeout = bData[6];
                    UINT8 DupCheck = bData[7];
                    UINT8 SameFingerCheck = bData[8];
                    UINT8 Volume = bData[11]; //0-16,>16 Voice off
                    return XG_ERR_SUCCESS;
                }
                else return bData[1];
            }
            return XG_ERR_FAIL;
}
```

2.2.4 Restore Factory Settings

- **Command Code**

  ```
  #define XG_CMD_FACTORY_SETTING          0x04 //Restore factory settings
  ```

- **Function**

  Restore the device settings to factory values.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x04	XG_CMD_FACTORY_SETTING
bEncode	0x00	Data encoding, 0
bDataLen	0x00	
bData		
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x04	XG_CMD_FACTORY_SETTING
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Factory settings restored successfully	
	bData[0] = XG_ERR_SUCCESS	
	Failed to restore factory settings	
wCheckSum		Checksum

- **Example**

```
/*
Function: Restore factory settings
Timeout: Reception timeout, in milliseconds
Return value:
XG_ERR_SUCCESS: Succeeded
XG_ERR_FAIL: Failed
*/

UINT8 SetDevFactory(int Timeout)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	SendPack(XG_CMD_FACTORY_SETTING, NULL, 0);
	gUartByte = 0; //Clear buffer received via UART port
	ret = RecvPack(NULL, bData, Timeout);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS) return XG_ERR_SUCCESS;
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.5 Set Device Number

- **Command Code**

  ```
  #define XG_CMD_SET_DEVICE_ID          0x05 //Set device number 0-255
  ```

- **Function**

  Set the device number, which can range from 0 to 255, with 0 as the default number. 1-255 indicates that only packets with the corresponding bAddress number will be accepted.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x05	XG_CMD_SET_DEVICE_ID
bEncode	0x00	Data encoding, 0
bDataLen	0x01	
bData	bData[0] = Device number	
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x05	XG_CMD_SET_DEVICE_ID
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Set up successfully	
	bData[0] = XG_ERR_SUCCESS	
	Setup failed	
wCheckSum		Checksum

- **Example**

```
/*
Function: Set device number
bDevID: Device number
Return value:
XG_ERR_SUCCESS: Succeeded
XG_ERR_FAIL: Failed
*/

UINT8 SetDevID(UINT8 bDevID)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = bDevID;
	SendPack(XG_CMD_SET_DEVICE_ID, bData, 1);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.6 Set UART Port Baudrate

- **Command Code**

  ```
  #define XG_CMD_SET_BAUDRATE          0x06 //Set baudrate 0-4
  ```

- **Function**

  Set UART port baudrate.

  The value range is 0-4, corresponding to the baud rate of 9600, 19200, 38400, 57600, and 115200, respectively.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x06	XG_CMD_SET_BAUDRATE
bEncode	0x00	Data encoding, 0
bDataLen	0x01	
bData	bData[0] = Baud rate index	0:9600 1:19200 2:38400 3:57600 4:115200
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x06	XG_CMD_SET_BAUDRATE
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Set up successfully	
	bData[0] = XG_ERR_SUCCESS	
	Setup failed	
wCheckSum		Checksum

- **Example**

```
/*
Function: Set baud rate
bBaud: Baud rate, 0=9600, 1=19200, 2=38400, 3=57600, 4-115200
Return value:
XG_ERR_SUCCESS: Succeeded
XG_ERR_FAIL: Failed
*/

UINT8 SetDevBaud(UINT8 bBaud)
{
        UINT8 ret = 0;
        UINT8 bData[16] = { 0 };

        bData[0] = bBaud;
        SendPack(XG_CMD_SET_BAUDRATE, bData, 1);
        gUartByte = 0;
        ret = RecvPack(NULL, bData, 1000);
        if(ret == XG_ERR_SUCCESS)
        {
                if(bData[0] == XG_ERR_SUCCESS)
                {
                        return XG_ERR_SUCCESS;
                }
                else return bData[1];
        }
        return XG_ERR_FAIL;
}
```

2.2.7 Set Security Level

- **Command Code**

  #define XG_CMD_SET_SECURITYLEVEL 0x07 //Set security level 0-4

- **Function**

  Set verification security level.

Security level	Recognition accuracy	
0	FAR (False Acceptance Rate)	**0.001%**
	FRR (False Rejection Rate)	**0.001%**
1	FAR (False Acceptance Rate)	**0.0005%**
	FRR (False Rejection Rate)	**0.01%**
2	FAR (False Acceptance Rate)	**0.0001%**
	FRR (False Rejection Rate)	**0.05 %**

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x07	XG_CMD_SET_SECURITYLEVEL
bEncode	0x00	Data encoding, 0
bDataLen	0x01	
bData	bData[0] = Security level	
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x07	XG_CMD_SET_SECURITYLEVEL
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Set up successfully	
	bData[0] = XG_ERR_SUCCESS	
	Setup failed	

wCheckSum		Checksum

- **API Example**

```
/*
Function: Set security level
bSecurity: Security level, 0, 1, 2
Return value:
XG_ERR_SUCCESS: Succeeded
XG_ERR_FAIL: Failed
*/

UINT8 SetDevSecurity(UINT8 bSecurity)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = bSecurity;
	SendPack(XG_CMD_SET_SECURITYLEVEL, bData, 1);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.8 Set Finger Placement Timeout

- **Command Code**

  #define XG_CMD_SET_TIMEOUT 0x08 //Set finger input waiting timeout 1-255s

- **Function**

  When executing login and authentication commands, you need to wait for finger input. This parameter is to set the waiting timeout, if the finger input is not detected within this time, a timeout error will be returned.

  The value ranges from 1-255 seconds.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x08	XG_CMD_SET_TIMEOUT
bEncode	0x00	Data encoding, 0
bDataLen	0x01	
bData	bData[0] = Timeout seconds	1-255s
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x08	XG_CMD_SET_TIMEOUT
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Set up successfully	
	bData[0] = XG_ERR_SUCCESS	
	Setup failed	
wCheckSum		Checksum

- **Example**

```
/*
Function: Set finger detection timeout
bTimeout: 1-255s
Return value:
XG_ERR_SUCCESS: Succeeded
XG_ERR_FAIL: Failed
*/
UINT8 SetDevCheckFingerTimeout(UINT8 bTimeout)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = bTimeout;
	SendPack(XG_CMD_SET_TIMEOUT, bData, 1);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.9 Set Duplicate Registration Check

- **Command Code**

  #define XG_CMD_SET_DUP_CHECK 0x09 //Set duplicate registration check 0-1

- **Function**

  When executing a registration command, if duplicate registration check is set to Yes, it will perform 1:N validation during registration. If similar users are found, it will return the XG_ERR_DUPLICATION_ID error.

  The value ranges from 0 (No) to 1 (Yes).

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x09	XG_CMD_SET_DUP_CHECK
bEncode	0x00	Data encoding, 0
bDataLen	0x01	
bData	bData[0] = 0, 1	No, Yes
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x09	XG_CMD_SET_DUP_CHECK
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Set up successfully	
	bData[0] = XG_ERR_SUCCESS	
	Setup failed	
wCheckSum		Checksum

- **Example**

```
/*
Function: Set duplicate registration check
bCheck: 0: Do not check; 1: Check
Return value:
XG_ERR_SUCCESS: Succeeded
XG_ERR_FAIL: Failed
*/

UINT8 SetDevDupCheck(UINT8 bCheck)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = bCheck;
	SendPack(XG_CMD_SET_DUP_CHECK, bData, 1);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.10 Set Same Finger Registration Check

- **Command Code**

  #define XG_CMD_SET_SAME_FV 0x0D //Check if the same finger during registration

- **Function**

  When executing the login command, if the same finger registration check is set to yes, it will be compared with the previously registered information during registration. If it is not the same finger, an XG_ERR_NO_SAME_FINGER error will be returned.
  The value ranges from 0 (No) to 1 (Yes).

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x09	XG_CMD_SET_DUP_CHECK
bEncode	0x00	Data encoding, 0
bDataLen	0x01	
bData	bData[0] = 0, 1	No, Yes
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x0D	XG_ERR_NO_SAME_FINGER
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Set up successfully	
	bData[0] = XG_ERR_SUCCESS	
	Setup failed	
wCheckSum		Checksum

- **Example**

```
/*
Function: Set whether to check with the same finger during registration
bCheck: 0: Do not check; 1: Check
Return value:
XG_ERR_SUCCESS: Succeeded
XG_ERR_FAIL: Failed
*/

UINT8 SetDevSameFingerCheck(UINT8 bCheck)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = bCheck;
	SendPack(XG_CMD_SET_SAME_FV, bData, 1);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.11 Set New Password

- **Command Code**

  ```
  #define XG_CMD_SET_PASSWORD          0x0A //set communication password
  ```

- **Function**

  Set a new connection password, which must be greater than or equal to 8 digits and less than or equal to 14 digits, otherwise the operation will fail.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x0A	XG_CMD_SET_PASSWORD
bEncode	0x00	Data encoding, 0
bDataLen	>= 8 && <= 14	Number of password digits
bData	New password	New password
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x0A	XG_CMD_SET_PASSWORD
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Set up successfully	
	bData[0] = XG_ERR_SUCCESS	
	Setup failed	
	bData[0] = XG_ERR_FAIL;	
wCheckSum		Checksum

- **Example**

```
/*
Function: Set a new communication password
pPassword: New password that needs to be set
Len: Password length, >= 8 && <= 14
Return value:
XG_ERR_SUCCESS: Set up successfully
XG_ERR_FAIL: Error
*/

UINT8 SetCommPassword(UINT8* pPassword, UINT8 Len)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	if(Len >= 8 && Len <= 14)
	{
		memcpy(bData, pPassword, Len);
	} else {
		return XG_ERR_FAIL;
	}
	SendPack(XG_CMD_SET_PASSWORD, bData, Len);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.12 Verify Password

- **Command Code**

  #define XG_CMD_CHECK_PASSWORD 0x0B //Check if the password is correct

- **Function**

  Check if the password is correct.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x0B	XG_CMD_CHECK_PASSWORD
bEncode	0x00	Data encoding, 0
bDataLen	>= 8 && <= 14	Number of password digits
bData	Password	Password
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x0B	XG_CMD_CHECK_PASSWORD
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Verification successful	
	bData[0] = XG_ERR_SUCCESS	
	Verification failed	
	bData[0] = XG_ERR_FAIL;	
wCheckSum		Checksum

- **Example**

```
/*
Function: Check if the newly set communication password is correct
pPassword: Password
Len: Password length
Return value:
XG_ERR_SUCCESS: Set up successfully
XG_ERR_FAIL: Error
*/

UINT8 CheckCommPassword(UINT8* pPassword, UINT8 Len)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	if(Len >= 8 && Len <= 14)
	{
		memcpy(bData, pPassword, Len);
	} else {
		return XG_ERR_FAIL;
	}
	SendPack(XG_CMD_CHECK_PASSWORD, bData, Len);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.13 Reset and Reboot Device

- **Command Code**

  ```
  #define XG_CMD_REBOOT                0x0C //Reset and reboot the device
  ```

- **Function**

  Reset and reboot the device.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x0C	XG_CMD_REBOOT
bEncode	0x00	Data encoding, 0
bDataLen		
bData		
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x0C	XG_CMD_REBOOT
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Reset successful	
	bData[0] = XG_ERR_SUCCESS	
	Reset failed	
wCheckSum		Checksum

- **Example**

```
/*
Function: Reset and reboot device
Return value:
XG_ERR_SUCCESS: Succeeded
XG_ERR_FAIL: Failed
*/

UINT8 SetDevReset()
{
	UINT8 ret = 0;

	SendPack(XG_CMD_SET_SAME_FV, NULL, 0);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.14 Detect Finger Input Status

- **Command Code**

  #define XG_CMD_FINGER_STATUS 0x10 //Detect finger placement status

- **Function**

  Detect if the finger is placed correctly.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x10	XG_CMD_FINGER_STATUS
bEncode	0x00	Data encoding, 0
bDataLen		
bData		
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x10	XG_CMD_FINGER_STATUS
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Detection successful	1 is returned only if the finger is placed correctly, otherwise 0 is returned
	bData[0] = XG_ERR_SUCCESS bData[1] = 0: no finger, 1: with finger	
wCheckSum		Checksum

- **Example**

```
/*
Function: Detect finger status
Return value:
0: no finger
1: with finger
*/

UINT8 CheckFinger()
{
	UINT8 ret = 0;

	SendPack(XG_CMD_FINGER_STATUS, NULL, 0);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return bData[1];
		}
	}
	return 0;
}
```

2.2.15 Clear Specified ID Login Data

- **Command Code**

  #define XG_CMD_CLEAR_ENROLL 0x11 //Clear login data for specified ID

- **Function**

  Delete registration data of the specified ID user.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x11	XG_CMD_CLEAR_ENROLL
bEncode	0x00	Data encoding, 0
bDataLen	0x04	

bData	bData[0] = ID&0xff;
	bData[1] = (ID>>8)&0xff;
	bData[2] = (ID>>16)&0xff;
	bData[3] = (ID>>24)&0xff;
		ID is the user number to be deleted

wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x11	XG_CMD_CLEAR_ENROLL
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Deleted successfully	
	bData[0] = XG_ERR_SUCCESS	
	Deletion failed	Deletion failed: XG_ERR_INVALID_ID : The user ID is invalid. XG_ERR_EMPTY_ID : This user ID has no login
	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_INVALID_ID/ XG_ERR_EMPTY_ID	

		data
wCheckSum		Checksum

- **Example**

```
/*
Function: Clear specified user data
UserID: Clear specified user ID
Return value:
XG_ERR_SUCCESS: Cleared successfully
XG_ERR_FAIL: Clearing failed
*/

UINT8 CleanUser(UINT16 UserID)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = (UserID)&0xff;
	bData[1] = ((UserID)>>8)&0xff;
	SendPack(XG_CMD_CLEAR_ENROLL, bData, 2);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.16 Clear All User Login Data

- **Command Code**

  #define XG_CMD_CLEAR_ALL_ENROLL 0x12 //Clear all ID login data

  Note: This command takes a long time to execute, it is not recommended to use it, it is suggested to clear them one by one

- **Function**

  Delete all user registration data.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x12	XG_CMD_CLEAR_ALL_ENROLL
bEncode	0x00	Data encoding, 0
bDataLen		
bData		
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x12	XG_CMD_CLEAR_ALL_ENROLL
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Deleted successfully	
	bData[0] = XG_ERR_SUCCESS	
wCheckSum		Checksum

- **Example**

```
UINT8 CleanAllUser()
{
	UINT8 ret = 0;
	UINT16 UserNum = 0;
	UINT16 UserMax = 0;

	//First get the number of registered users and estimate the approximate clearing time required
	ret = GetEnrollInfo(&UserNum, &UserMax);
	if(ret == XG_ERR_SUCCESS)
	{
		int timeout = 1000 + UserNum*300;
		UINT8 bData[16] = { 0 };

		SendPack(XG_CMD_CLEAR_ALL_ENROLL, NULL, 0);
		gUartByte = 0;
		ret = RecvPack(NULL, bData, timeout);
		if(ret == XG_ERR_SUCCESS)
		{
			if(bData[0] == XG_ERR_SUCCESS)
			{
				return XG_ERR_SUCCESS;
			}
			else return bData[1];
		}
	}
	return XG_ERR_FAIL;
}
```

2.2.17 Get Empty (Available) ID

- **Command Code**

  #define XG_CMD_GET_EMPTY_ID 0x13 //Get empty (no login data) ID

- **Function**

  Get the empty (no login data) ID number.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x13	XG_CMD_GET_EMPTY_ID
bEncode	0x00	Data encoding, 0

bDataLen	bData[0] = StartId&0xff;
	bData[1] = (StartId >>8)&0xff;
	bData[2] = (StartId >>16)&0xff;
	bData[3] = (StartId >>24)&0xff;
	bData[4] = EndId&0xff;
	bData[5] = (EndId >>8)&0xff;
	bData[6] = (EndId >>16)&0xff;
	bData[7] = (EndId >>24)&0xff;
		If the StartId is 0 or EndId is 0, the free ID is found from all users, otherwise the free ID is found from the users specified from StartId to EndId

bData		
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x13	XG_CMD_GET_EMPTY_ID
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Obtained successfully bData[0] = XG_ERR_SUCCESS ID:bData[1]-bData[4]	ID = bData[1] + (bData[2]<<8) + (bData[3]<<16) + (bData[4]<<24);

	Failed to obtain bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_NO_NULL_ID	Full registration, no empty ID available
wCheckSum		Checksum

- **Example**

```
/*
Function: Get an empty ID, that is, an unregistered ID
pUserID: Return a pointer to the registrable ID
Return value:
XG_ERR_SUCCESS: Received successfully
XG_ERR_FAIL: Communication error
XG_ERR_NO_NULL_ID: No empty ID
*/

UINT8 GetNullID(UINT16* pUserID)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };
	SendPack(XG_CMD_GET_EMPTY_ID, NULL, 0);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			if(pUserID) *pUserID = bData[1] + (bData[2]<<8);
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.18 Obtain System Login Information (Number of login users, number of templates)

- **Command Code**

  #define XG_CMD_GET_ENROLL_INFO 0x14 //Get the total number of login users and templates

- **Function**

  Obtain system login information, including the number of logged-in users, the number of logged-in templates, and the maximum number of registered users allowed. Each user can log in to a maximum of 3 templates, so the maximum number of registered template accounts allowed is the maximum number of registered users multiplied by 3.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x14	XG_CMD_GET_ENROLL_INFO
bEncode	0x00	Data encoding, 0
bDataLen		
bData		
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x14	XG_CMD_GET_ENROLL_INFO
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Obtained successfully	Number of logged in users = bData[1] + (bData[2]<<8) + (bData[3]<<16) + (bData[4]<<24);
	bData[0] = XG_ERR_SUCCESS EnrollUser:bData[1]-bData[4] EnrollMax:bData[9]-bData[12]	Maximum number of users =

		bData[9] + (bData[10]<<8) + (bData[11]<<16) + (bData[12]<<24);
wCheckSum		Checksum

- **Example**

```
/*
Function: Obtain registration information
pUserNum: Number of registered users
pUserMax: Maximum number of users
Return value:
XG_ERR_SUCCESS: Obtained successfully
XG_ERR_FAIL: Failed to obtain
*/

UINT8 GetEnrollInfo(UINT16* pUserNum, UINT16* pUserMax)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	SendPack(XG_CMD_GET_ENROLL_INFO, NULL, 0);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			if(pUserNum) *pUserNum = bData[1] + (bData[2]<<8);
			if(pUserMax) *pUserMax = bData[9] + (bData[10]<<8);
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.19 Get Specified ID Login Data

- **Command Code**

  #define XG_CMD_GET_ID_INFO 0x15 //Get login information for specified ID

- **Function**

  Get the number of login templates with the specified ID. If the number of templates is 0, it means an empty (no login) ID.

- **Command Example**

Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x15	XG_CMD_GET_ID_INFO
bEncode	0x00	Data encoding, 0
bDataLen	0x04	

bData	bData[0] = ID&0xff;
	bData[1] = (ID>>8)&0xff;
	bData[2] = (ID>>16)&0xff;
	bData[3] = (ID>>24)&0xff;	

wCheckSum		Checksum

Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x15	XG_CMD_GET_ID_INFO
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Obtained successfully	
	bData[0] = XG_ERR_SUCCESS bData[1] = Number of templates	
	Failed to obtain	Invalid ID
	bData[0] = XG_ERR_FAIL	

	bData[1] = XG_ERR_INVALID_ID	
wCheckSum		Checksum

- **Example**

```
/*
Function: Get registration information of the specified user ID
UserID: Specified ID
Return value:
0: This user is not registered
>0: This user has already registered
*/

UINT8 GetIDEnroll(UINT16 UserID)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = (UserID)&0xff;
	bData[1] = ((UserID)>>8)&0xff;
	SendPack(XG_CMD_GET_ID_INFO, bData, 2);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return bData[1];
		}
	}
	return 0;
}
```

2.2.20 User Registration

- **Command Code**

  #define XG_CMD_ENROLL 0x16 //Login with specified ID

- **Function**

  Register specified ID, and a single registration can collect a single template or multiple templates (up to 3 templates).

  Note:

  This command returns multiple packets for normal registration, and the registration process is only terminated when a return value of XG_ERR_SUCCESS or XG_ERR_FAIL is received.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x16	XG_CMD_ENROLL
bEncode	0x00	Data encoding, 0
bDataLen	0x04	

bData	bData[0] = ID&0xff;
	bData[1] = (ID>>8)&0xff;
	bData[2] = (ID>>16)&0xff;
	bData[3] = (ID>>24)&0xff;
	bData[4] = Group number
	bData[5] = Number of templates for this registration	

wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x16	XG_CMD_ENROLL
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Logged in successfully	

	bData[0] = XG_ERR_SUCCESS	
	Login failed	Login failed: XG_ERR_INVALID_ID: Invalid ID XG_ERR_NOT_ENOUGH : No enough space to log in XG_ERR_TIME_OUT : Finger input waiting timeout XG_ERR_DUPLICATION_ID : A duplicate user has already logged in
	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_INVALID_ID/ XG_ERR_NOT_ENOUGH/ XG_ERR_TIME_OUT/ XG_ERR_DUPLICATION_ID	
	XG_INPUT_FINGER:	Prompt the user to put their fingers
	XG_RELEASE_FINGER	Prompt the user to remove their fingers
wCheckSum		Checksum

- **Example**

```
/*
Function: Send registration command to add finger vein user
UserID: The ID of the specified registered user
Return value:
XG_ERR_SUCCESS: Registered successfully
*/

UINT8 EnrollUser(UINT16 UserID)
{
        UINT8 ret = 0;
        UINT8 bData[16] = { 0 };

        bData[0] = UserID&0xff;
        bData[1] = (UserID>>8)&0xff;
        bData[2] = 0;
        bData[3] = 0;
        bData[4] = 0;
        bData[5] = 3; //The number of successful collections, as long as the collection is successful, the registration is
```

```
successful
        bData[10] = 10; //If the collection is unsuccessful, the total number of retries is allowed
        SendPack(XG_CMD_ENROLL, bData, 12);
        gUartByte = 0;
        for(;;)
        {
                ret = RecvPack(NULL, bData, 6000); //Finger detection timeout is 5 seconds
                if(ret != XG_ERR_SUCCESS)
                {
                        PLAY_SOUND(VOICE_REG_FAIL);
                        break;
                }
                if(bData[0] == XG_ERR_SUCCESS)
                {
                        PLAY_SOUND(VOICE_REG_OK);
                        break;
                } else if(bData[0] == XG_INPUT_FINGER) {
                        //BData [1] is the number of times collected
                        if(bData[1] == 0) PLAY_SOUND(VOICE_INPUT);
                } else if(bData[0] == XG_RELEASE_FINGER) {
                        //Finger vein collection completed, you can now remove your finger
                        PLAY_SOUND(VOICE_KEY);
                        DelayMs(200);
                        if(bData[1] == 0 || bData[1] == 1)
                        {
                                PLAY_SOUND(VOICE_INPUT_FINGER_AGAIN);
                        }
                } else {
                        //Error handling
                        ret = bData[1];
                        if(ret == XG_ERR_INVALID_ID)
                        {
                                //Invalid ID
                                PLAY_SOUND(VOICE_PINPUT_ID); //ID error
                        } else if(ret == XG_ERR_NOT_ENOUGH) {
                                //This user has already registered
```

```
                    PLAY_SOUND(VOICE_USER_FULL);
                } else if(ret == XG_ERR_TIME_OUT) {
                    //Finger detection timeout
                    PLAY_SOUND(VOICE_REG_FAIL);
                } else if(ret == XG_ERR_DUPLICATION_ID) {
                    //Duplicate registration
                    PLAY_SOUND(VOICE_OVER_REG);
                } else if(ret == XG_ERR_NO_SAME_FINGER) {
                    //Not the same finger
                    PLAY_SOUND(VOICE_RIGHT_INPUT);
                } else if(ret == XG_ERR_NO_VEIN) {
                    //No finger vein detected
                    PLAY_SOUND(VOICE_RIGHT_INPUT);
                } else {
                    PLAY_SOUND(VOICE_REG_FAIL);
                }
                break;
            }
        }
        return ret;
}
```

2.2.21 User Authentication

- **Command Code**

  #define XG_CMD_VERIFY 0x17 //1:1 authentication or 1:N recognition

- **Function**

  If a user ID is specified, it will be in 1:1 authentication mode.

  If the specified user ID is 0, it will be in 1:N recognition verification mode.

  Note:

  This command returns multiple packets for normal verification, and the verification process is only terminated when a return value of XG_ERR_SUCCESS or XG_ERR_FAIL is received.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x17	XG_CMD_VERIFY
bEncode	0x00	Data encoding, 0
bDataLen	0x04	

bData	bData[0] = ID&0xff;
	bData[1] = (ID>>8)&0xff;
	bData[2] = (ID>>16)&0xff;
	bData[3] = (ID>>24)&0xff;
	bData[4] = Group number
	bData[5] = Continuously verify several IDs
		ID=0 for 1:N verification method
		ID>0 for 1:1 verification method

wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x17	XG_CMD_VERIFY
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Verified successfully	Identified user ID = bData[1] +

	bData[0] = XG_ERR_SUCCESS ID:bData[1]-bData[4]	(bData[2]<<8) + (bData[3]<<16) + (bData[4]<<24);
	Verification failed	
	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_INVALID_ID / XG_ERR_TIME_OUT	XG_ERR_TIME_OUT : Finger input waiting timeout
	XG_INPUT_FINGER:	Prompt the user to put their fingers
	XG_RELEASE_FINGER	Prompt the user to remove their fingers
wCheckSum		Checksum

- **Example**

```
/*
Function: Send verification command and get the returned verification result
pUserID: Return a pointer to the successfully verified user ID
Return value:
XG_ERR_SUCCESS: Verification succeeded, user ID successfully verified is returned via pUserID
XG_ERR_NO_VEIN: Finger vein collection failed, possibly due to improper finger placement
*/

UINT8 VerifyUser(UINT16* pUserID)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };
	UINT32 CardNo = 0;

	if(pUserID)
	{
		//If pUserID is not valid, then 1:1 validation
		bData[0] = (*pUserID)&0xff;
		bData[1] = ((*pUserID)>>8)&0xff;
		bData[2] = 0;
		bData[3] = 0;
		bData[4] = 0; //0 indicates that it is not grouped
```

```
                bData[5] = 0; //0 indicates single user verification, >0 indicates continuous user verification starting
from UserID
                bData[6] = CardNo&0xff; //You can also perform 1:1 verification by card number
                bData[7] = (CardNo>>8)&0xff;
                bData[8] = (CardNo>>16)&0xff;
                bData[9] = (CardNo>>24)&0xff;
        }

        SendPack(XG_CMD_VERIFY, bData, 10);
        gUartByte = 0;
        for(;;)
        {
                ret = RecvPack(NULL, bData, 6000); //Finger detection timeout is 5 seconds
                if(ret != XG_ERR_SUCCESS)
                {
                        PLAY_SOUND(VOICE_IDENT_FAIL);
                        break;
                }
                if(bData[0] == XG_ERR_SUCCESS)
                {
                        PLAY_SOUND(VOICE_IDENT_OK);
                        if(pUserID) *pUserID = bData[1] + (bData[2]<<8);
                        break;
                } else if(bData[0] == XG_INPUT_FINGER) {

                } else if(bData[0] == XG_RELEASE_FINGER) {
                        //Finger vein collection completed, you can now remove your finger
                        PLAY_SOUND(VOICE_KEY);
                } else {
                        //Error handling
                        ret = bData[1];
                        if(ret == XG_ERR_NO_VEIN)
                        {
                                PLAY_SOUND(VOICE_RIGHT_INPUT);
                        } else {
                                PLAY_SOUND(VOICE_IDENT_FAIL);
```

```
                        }
                        break;
                    }
                }
                return ret;
}
```

2.2.22 Extend Registration Command

- **Command Code**

  #define XG_CMD_ENROLL_EXT 0x38 //Extend voice registration

- **Function**

  Get the device's serial number

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x38	XG_CMD_ENROLL_EXT
bEncode	0x00	Data encoding, 0
bDataLen		
bData		
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x38	XG_CMD_ENROLL_EXT
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Registered successfully bData[0] = XG_ERR_SUCCESS ID:bData[1]-bData[2]	Successfully registered ID = bData[1] + (bData[2]<<8)
	Registration failed bData[0] = XG_ERR_FAIL bData[1] = Error code	
wCheckSum		Checksum

- **Example**

```
/*
Function: Send extended registration instructions to add finger vein users, and the registration process prompts the
user to operate the process through voice, and devices without voice are not applicable
UserID: The ID of the specified registered user
Return value:
XG_ERR_SUCCESS: Registered successfully
*/

UINT8 EnrollUserEXT(UINT16* pUserID)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };
	UINT16 UserID = *pUserID;

	bData[0] = UserID&0xff;
	bData[1] = (UserID>>8)&0xff;
	SendPack(XG_CMD_ENROLL_EXT, bData, 2);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 5000*3); //The timeout for a finger detection is 15 seconds, and it takes at least
5 seconds each time
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			//If UserID is invalid, this command will automatically find an empty ID to register and
return this ID
			if(pUserID)
				*pUserID = bData[1] + bData[2]*256;
			return XG_ERR_SUCCESS;
		} else {
			return bData[1];
		}
	}
	return ret;
}
```

2.2.23 Extend Validation Command

- **Command Code**

  #define XG_CMD_VERIFY_EXT 0x39 //Extend voice verification

- **Function**

  Send a command to verify, there is only one return packet

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x39	XG_CMD_VERIFY_EXT
bEncode	0x00	Data encoding, 0
bDataLen		
bData		
wCheckSum		Checksum

  Return packet:

<table>
<tr><td>Field</td><td>Data</td><td>Remark</td></tr>
<tr><td>wPrefix</td><td>0xAABB</td><td>Package identifier</td></tr>
<tr><td>bAddress</td><td>0x00</td><td>Target device address</td></tr>
<tr><td>bCmd</td><td>0x39</td><td>XG_CMD_VERIFY_EXT</td></tr>
<tr><td>bEncode</td><td>0x00</td><td>Data encoding, 0</td></tr>
<tr><td>bDataLen</td><td></td><td></td></tr>
<tr><td rowspan="4">bData</td><td>Verified successfully</td><td rowspan="2">Successfully registered ID = b Data[1] + (bData[2]<<8)</td></tr>
<tr><td>bData[0] = XG_ERR_SUCCESS
ID:bData[1]-bData[2]</td></tr>
<tr><td>Verification failed</td><td rowspan="2"></td></tr>
<tr><td>bData[0] = XG_ERR_FAIL</td></tr>
<tr><td>wCheckSum</td><td></td><td>Checksum</td></tr>
</table>

- **Example**

```
/*
Function: Send extended verification command, which has only one return packet
pUserID: Return a pointer to the successfully verified user ID
Return value:
XG_ERR_SUCCESS: Verification succeeded, user ID successfully verified is returned via pUserID
XG_ERR_NO_VEIN: Finger vein collection failed, possibly due to improper finger placement
*/

UINT8 VerifyUserEXT(UINT16* pUserID)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };
	UINT32 CardNo = 0;

	if(pUserID)
	{
		//If pUserID is not valid, then 1:1 validation
		bData[0] = (*pUserID)&0xff;
		bData[1] = ((*pUserID)>>8)&0xff;
		bData[2] = 0;
		bData[3] = 0;
		bData[4] = 0; //0 indicates that it is not grouped
		bData[5] = 0; //0 indicates single user verification, >0 indicates continuous user verification starting
from UserID
		bData[6] = CardNo&0xff; //You can also perform 1:1 verification by card number
		bData[7] = (CardNo>>8)&0xff;
		bData[8] = (CardNo>>16)&0xff;
		bData[9] = (CardNo>>24)&0xff;
	}

	//The XG_CMD_VERIFY.exe command does not have a return packet that prompts for finger placement or
removal
	SendPack(XG_CMD_VERIFY_EXT, bData, 10);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 6000);
	if(ret == XG_ERR_SUCCESS)
```

```
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			PLAY_SOUND(VOICE_IDENT_OK);
			if(pUserID) *pUserID = bData[1] + (bData[2]<<8);
		} else {
			ret = bData[1];
			if(ret == XG_ERR_NO_VEIN)
			{
				PLAY_SOUND(VOICE_RIGHT_INPUT);
			} else {
				PLAY_SOUND(VOICE_IDENT_FAIL);
			}
		}
	}
	return ret;
}
```

2.2.24 Read Data

- **Command Code**

  #define XG_CMD_READ_DATA 0x20 //Read data from device

- **Function**

  Read data from device, the supported data types include login data, image data, and debugging information, which must be used with the following commands:

Data type	Command code	Remark
XG_CMD_READ_ENROLL	0x22	Read specified ID login data
XG_CMD_GET_CHARA	0x28	Read characteristic data
XG_CMD_GET_DEBUG	0x26	Read debugging information

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x20	XG_CMD_READ_DATA
bEncode	0x00	Data encoding, 0
bDataLen		
bData	bData[0] = Data type bData[1] – bData[4]:Data offset bData[5] – bData[8]:Data size, UART port data packet size cannot exceed 512 bytes	Data type: XG_CMD_READ_ENROLL XG_CMD_READ_CHARA XG_CMD_GET_DEBUG
wCheckSum		Checksum

  Return packet:

  Data + checksum (2 bytes)

- **Example**

```
/*
Function: Read sub-packet data
Parameters:
bCmd: Command code
```

```
bDataBuf: The first address of the data buffer to be read
iSize: The size of the data to be read
iTimeout: The timeout for reading a packet
Return value: = 0 read succeeded, non-0 read failed
*/
static UINT8 ReadData(UINT8 bCmd, UINT8* bDataBuf, UINT16 iSize, int iTimeout)
{
	int ret = 0;
	int  lPkNum, lPkRec, lDataMove, i;
	int reReadCnt = 0;
	int PackSize = COM_DATA_PACKET_SIZE;

	//Sub-packet
	lPkNum = (iSize)/PackSize;
	lPkRec = (iSize)%PackSize;
	lDataMove = 0;

	for(i = 0; i < lPkNum; i++)
	{
		ret = ReadDataPack(bCmd, bDataBuf, lDataMove, PackSize, iTimeout);
		if(ret == PackSize)
		{
			reReadCnt = 0;
			lDataMove += PackSize;
			bDataBuf += PackSize;
		} else {
			//If the read fails, retry twice
			i--;
			if(reReadCnt++ > 2)
			{
				return XG_ERR_COM;
			}
		}
	}

	if(lPkRec > 0)
```

```
        {
                ret = ReadDataPack(bCmd, bDataBuf, lDataMove, lPkRec, iTimeout);
                if(ret == lPkRec)
                {
                        lDataMove += lPkRec;
                }
        }
        if(lDataMove == iSize) return XG_ERR_SUCCESS;
        return XG_ERR_FAIL;
}
```

2.2.25 Write Data

- **Command Code**

  #define XG_CMD_WRITE_DATA 0x21 //Write data to device

- **Function**

  Write specified types of data to the device, the supported data types including login data, image data, and upgrade programs, which must be used in conjunction with the following commands:

Data type	Command code	Remark
XG_CMD_WRITE_ENROLL	0x23	Write specified ID login data
XG_CMD_WRITE_IMAGE	0x25	Write image data
XG_CMD_UPGRADE	0x27	Write upgrade program (online upgrade)

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x21	XG_CMD_WRITE_DATA
bEncode	0x00	Data encoding, 0
bDataLen		
bData	bData[0] = Data type bData[1] – bData[4]: Data offset bData[5] – bData[8]: Data size, UART port data packet size cannot exceed 512 bytes	Data type: XG_CMD_READ_ENROLL XG_CMD_READ_IMAGE XG_CMD_GET_DEBUG
wCheckSum		Checksum
WriteData	Data[0]…Data[N]	Written data
DataCheckSum	Data[0]+Data[1]…+Data[N]	Data checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x21	XG_CMD_WRITE_DATA
bEncode	0x00	Data encoding, 0
bDataLen	0x01	
bData	Succeeded	
	bData[0] = XG_ERR_SUCCESS	
	Failed	
	bData[0] = XG_ERR_FAIL	
	Data checksum error	
	bData[0] = XG_ERR_DATA	
wCheckSum		Checksum

- **Example**

```
/*
Function: Write sub-packet data
Parameters:
bCmd: Command code
bDataBuf: The first address of the data buffer to be written
iSize: The size of the data to be written
Return value: = 0 write succeeded, non-0 write failed
*/
static UINT8 WriteData(UINT8 bCmd, UINT8* bDataBuf, UINT16 iSize)
{
        int ret = 0;
        int  lPkNum, lPkRec, lDataMove, i;
        int reWriteCnt = 0;
        int PackSize = COM_DATA_PACKET_SIZE;

        //Sub-packet
        lPkNum = (iSize)/PackSize;
        lPkRec = (iSize)%PackSize;
```

```
    lDataMove = 0;

    for(i = 0; i < lPkNum; i++)
    {
        ret = WriteDataPack(bCmd, bDataBuf, lDataMove, PackSize);
        if(ret == XG_ERR_SUCCESS)
        {
            reWriteCnt = 0;
            lDataMove += PackSize;
        } else {
            //If the write fails, retry twice
            i--;
            if(reWriteCnt++ > 2)
            {
                return XG_ERR_COM;
            }
        }
    }

    if(lPkRec > 0)
    {
        ret = WriteDataPack(bCmd, bDataBuf, lDataMove, lPkRec);
        if(ret == XG_ERR_SUCCESS)
        {
            lDataMove += lPkRec;
        }
    }

    if(lDataMove == iSize) return XG_ERR_SUCCESS;
    return XG_ERR_FAIL;
}
```

2.2.26 Read Specified ID Registration Data

- **Command Code**

  #define XG_CMD_READ_ENROLL 0x22 //Read login data of the specified ID

- **Function**

  Read the specified ID registration data. Before reading, send this command to check if the reading conditions are met. If the return is successful, the data can be read and the size of the data to be read will be returned. Then use the XG_CMD_READ_DATA command to read the specific data.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x22	XG_CMD_READ_ENROLL
bEncode	0x00	Data encoding, 0
bDataLen	0x04	

bData	buf[0] = ID&0xff;
	buf[1] = (ID >>8)&0xff;
	buf[2] = (ID >>16)&0xff;
	buf[3] = (ID >>24)&0xff;	

wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x22	XG_CMD_READ_ENROLL
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Succeeded	size =bData[1] + (bData[2] << 8) + (bData[3] << 16) + (bData [4] << 24);
	bData[0] = XG_ERR_SUCCESS bData[1]-bData[4]: Data size	
	Failed	XG_ERR_INVALID_ID: Invalid ID XG_ERR_EMPTY_ID: Empty ID, this ID has no login data
	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_INVALID_ID/ XG_ERR_EMPTY_ID	

wCheckSum		Checksum

- **Example**

```
/*
Function: Read the template of the specified user ID
Parameters:
iUserID: User ID
bTempBuf: Template data cache, at least 1000 bytes of cache space
Return value: > 0: template data size; =0: read failed
*/
UINT16 ReadDevTemp(UINT16 iUserID, UINT8* bTempBuf)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = iUserID%256;
	bData[1] = iUserID/256;
	bData[2] = 0;
	bData[3] = 0;
	ret = SendPack(XG_CMD_READ_ENROLL, bData, 4);
	if(ret == XG_ERR_SUCCESS) {
		ret = RecvPack(NULL, bData, 1000);
		if (ret == XG_ERR_SUCCESS) {
			if(bData[0] == XG_ERR_SUCCESS) {
				UINT16 TempSize = bData[1] + bData[2]*256;
				ret = ReadData(XG_CMD_READ_ENROLL, bTempBuf, TempSize, 1000);
				if (ret == XG_ERR_SUCCESS) {
					return TempSize;
				}
			} else {
				//bData[1] is an error code
			}
		}
	}
	return 0;
}
```

2.2.27 Write Specified ID Registration Data

- **Command Code**

  #define XG_CMD_WRITE_ENROLL 0x23 //Write (overwrite) login data for the specified ID

- **Function**

  Write the specified ID registration data. Before writing, send this command to check if the writing conditions are met. If the return is successful, the data can be written. Then use the XG_CMD_WRITE_DATA command to write the specific data.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x23	XG_CMD_WRITE_ENROLL
bEncode	0x00	Data encoding, 0
bDataLen	0x08	

bData	buf[0] = ID&0xff;
	buf[1] = (ID >>8)&0xff;
	buf[2] = (ID >>16)&0xff;
	buf[3] = (ID >>24)&0xff;
	buf[4] = SIZE&0xff;
	buf[5] = (SIZE >>8)&0xff;
	buf[6] = (SIZE >>16)&0xff;
	buf[7] = (SIZE >>24)&0xff;
		Specify the user ID and the size of the data to be written.

wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x23	XG_CMD_WRITE_ENROLL
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Succeeded	
	bData[0] = XG_ERR_SUCCESS	

	Failed bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_INVALID_ID/ XG_ERR_INVALID_PARAM	XG_ERR_INVALID_ID: Invalid ID XG_ERR_INVALID_PARAM: Parameter error, possibly due to mismatched data size
wCheckSum		Checksum

- **Example**

```
/*
Function: Write the template of the specified user ID
Parameters:
iUserID: User ID
bTempBuf: Template data cache to be written
iSize: Template data size
Return value: = 0 write succeeded, non-0 write failed
*/
UINT8 WriteDevTemp(UINT16 iUserID, UINT8* bTempBuf, UINT16 iSize)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = iUserID%256;
	bData[1] = iUserID/256;
	bData[2] = 0;
	bData[3] = 0;
	bData[4] = iSize%256;
	bData[5] = iSize/256;
	bData[6] = 0;
	bData[7] = 0;
	ret = SendPack(XG_CMD_WRITE_ENROLL, bData, 8);
	if(ret == XG_ERR_SUCCESS) {
		ret = RecvPack(NULL, bData, 1000);
		if (ret == XG_ERR_SUCCESS) {
			if(bData[0] == XG_ERR_SUCCESS) {
				ret = WriteData(XG_CMD_WRITE_ENROLL, bTempBuf, iSize);
				return ret;
```

```
                        }
                } else {
                        //bData[1] is an error code
                }
        }
        return XG_ERR_FAIL;
}
```

2.2.28 Collect and Read Characteristics to Host

- **Command Code**

```
#define XG_CMD_GET_CHARA                0x28 //Collect and read characteristics to the host
```

- **Function**

Collect the vein characteristics data of the current finger, and if the collection is successful, then use the XG_CMD_READ_DATA command to read the specific data.

- **Command Example**

Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x28	XG_CMD_GET_CHARA
bEncode	0x00	Data encoding, 0
bDataLen	0x00	
bData		
wCheckSum		Checksum

Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x28	XG_CMD_GET_CHARA
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Succeeded	Multiple packets will be returned, prompting to place and remove your finger
	bData[0] = XG_INPUT_FINGER / MSG_OUTPUT_FINGER /XG_ERR_SUCCESS	
	Failed	XG_ERR_TIME_OUT: No finger detected, timeout returned
	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_TIME_OUT	
wCheckSum		Checksum

- **Example**

```
/*
Function: Collect finger vein characteristics
Parameters:
bCharaBuf: Characteristics data cache, at least 1000 bytes of cache space
lTimeout: Finger placement waiting timeout
Return value: > 0: characteristics data size; =0: collection failed
*/
UINT16 GetVeinChara(UINT8* bCharaBuf, int lTimeout)
{
	//Send a command to collect characteristics
	if(SendPack(XG_CMD_GET_CHARA, NULL, 0) == XG_ERR_SUCCESS)
	{
		for(;;)
		{
			UINT8 bData[16] = { 0 };
			int ret = RecvPack(NULL, bData, lTimeout);
			if (ret == XG_ERR_SUCCESS) { //Received a return packet
				if(bData[0] == XG_ERR_SUCCESS) { //Collection succeeded
					int CharaSize = bData[1] + bData[2]*255;
					ret = ReadData(XG_CMD_GET_CHARA, bCharaBuf, CharaSize,
1000);
					if (ret == XG_ERR_SUCCESS) {
						return CharaSize;
					}
					break;
				} else if(bData[0] == XG_ERR_FAIL) {
					//bData[1] is the error code for collection failure
					return 0;
				} else if(bData[0] == XG_INPUT_FINGER) {
				} else if(bData[0] == XG_RELEASE_FINGER) {
				} else {
					break;
				}
			} else {
```

```
                                        break;
                                }
                        }
                }
                return 0;
}
```

2.2.29 Get Device Serial Number

- **Command Code**

```
#define XG_CMD_GET_DUID                0x0F //Get device serial number
```

- **Function**

  Get the device serial number

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x0F	XG_CMD_GET_DUID
bEncode	0x00	Data encoding, 0
bDataLen	0x05	
bData		
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x0F	XG_CMD_GET_DUID
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Succeeded	
	bData[0] = XG_ERR_SUCCESS bData[1]-bData[14]: serial numbers	
	Failed	
	bData[0] = XG_ERR_FAIL	
wCheckSum		Checksum

- **Example**

```
/*
Function: Get the device serial number
pSN: Return the device serial number
Return value:
XG_ERR_SUCCESS: Succeeded
XG_ERR_FAIL: Error
*/

UINT8 GetDevSN(UINT8* pSN)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };
	SendPack(XG_CMD_GET_DUID, NULL, 0);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			if(pSN) memcpy(pSN, bData[1], 14);
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.30 Play Collector Voice

- **Command Code**

```
#define XG_CMD_PLAY_VOICE                0x3b //Play voice
```

- **Function**

Play the voice of the data collector, only devices that support voice are applicable

- **Command Example**

Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x3B	XG_CMD_PLAY_VOICE
bEncode	0x00	Data encoding, 0
bDataLen	0x05	
bData	bData[0]=Voice serial number, see Appendix bData[1]=Mode, 0 asynchronous, 1 synchronous	
wCheckSum		Checksum

Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x3B	XG_CMD_PLAY_VOICE
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Succeeded	
	bData[0] = XG_ERR_SUCCESS	
	Failed	
	bData[0] = XG_ERR_FAIL	
wCheckSum		Checksum

- **Example**

```
/*
Function: Play device voice, only applicable to devices that support voice
bSound: Voice serial number
bMode: 0 asynchronous, 1 synchronous
Return value:
XG_ERR_SUCCESS: Succeeded
XG_ERR_FAIL: Failed
*/

UINT8 PlayDevSound (UINT8 bSound, UINT8 bMode)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = bSound;
	bData[1] = bMode;
	SendPack(XG_CMD_PLAY_VOICE, bData, 2);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000); //If asynchronous, this timeout will be extended, depending on the length of the voice
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.31 Access Control Extended Door Opening Command

- **Command Code**

  #define XG_CMD_OPEN_DOOR 0x32 //Open door command

- **Function**

  Send a command to open the door, applicable to access control or locks

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x32	XG_CMD_OPEN_DOOR
bEncode	0x00	Data encoding, 0
bDataLen	0x05	
bData	bData[0-3] = Door opening ID	
wCheckSum		Checksum

- **Example**

```
/*
Function: Use the command to open the door
UserId: Use that user ID to open the door
Return value:
XG_ERR_SUCCESS: Succeeded
XG_ERR_FAIL: Failed
*/

UINT8 DevOpenDoor(UINT8 UserId)
{
        UINT8 ret = 0;
        UINT8 bData[16] = { 0 };

        bData[0] = UserId%256;
        bData[1] = UserId/256;
        SendPack(XG_CMD_OPEN_DOOR, bData, 2);
        gUartByte = 0;
```

```
        ret = RecvPack(NULL, bData, 1000); //If asynchronous, this timeout will be extended, depending on the length of the voice
        if(ret == XG_ERR_SUCCESS)
        {
                if(bData[0] == XG_ERR_SUCCESS)
                {
                        return XG_ERR_SUCCESS;
                }
                else return bData[1];
        }
        return XG_ERR_FAIL;
}
```

2.2.32 Modify Device Time

- **Command Code**

  ```
  #define XG_CMD_SET_DATETIME          0x36 //Modify device time
  ```

- **Function**

  Set the system time for modifying the device, applicable to products such as locks and access control

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x36	XG_CMD_SET_DATETIME
bEncode	0x00	Data encoding, 0
bDataLen	0x06	
bData	buf[0] = Year-2000 buf[1] = Mon buf[2] = Day buf[3] = Hour buf[4] = Min buf[5] = Sec	
wCheckSum		Checksum

- **Example**

```
/*
Function: Modify the device time

Return value:
XG_ERR_SUCCESS: Succeeded
XG_ERR_FAIL: Failed
*/

UINT8 SetDevDateTime(UINT16 year, UINT8 mon, UINT8 day, UINT8 hour, UINT8 min, UINT8 sec)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = year - 2000;
	bData[1] = mon;
	bData[2] = day;
	bData[3] = hour;
	bData[4] = min;
	bData[5] = sec;
	SendPack(XG_CMD_SET_DATETIME, bData, 6);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.33 No Timeout User Authentication

- **Command Code**

  #define XG_CMD_IDENTIFY_FREE 0x18 //No timeout 1:1 authentication or:N recognition, 0x19 command can be issued to interrupt

- **Function**

  Sending this command will wait for the finger to be placed until the authentication is successful or failed before exiting the command, or it can be interrupted by sending a 0x19 command with the same parameters as the 0x17 command.

  If a user ID is specified, it will be in 1:1 authentication mode.

  If the specified user ID is 0, it will be in 1:N recognition verification mode.

  Note:

  This command returns multiple packets for normal verification, and the verification process is only terminated when a return value of XG_ERR_SUCCESS, XG_ERR_FAIL or XG_ERR_BREAK_OFF is received.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x18	XG_CMD_IDENTIFY_FREE
bEncode	0x00	Data encoding, 0
bDataLen	0x04	

bData	bData[0] = ID&0xff;
	bData[1] = (ID>>8)&0xff;
	bData[2] = (ID>>16)&0xff;
	bData[3] = (ID>>24)&0xff;
	bData[4] = Group number
	bData[5] = Continuously verify several IDs
		ID=0 for 1:N verification method
		ID>0 for 1:1 verification method

wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address

bCmd	0x18	XG_CMD_IDENTIFY_FREE
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Verified successfully	Identified user ID = bData[1] + (bData[2]<<8) + (bData[3]<<16) + (bData[4]<<24);
	bData[0] = XG_ERR_SUCCESS ID:bData[1]-bData[4]	
	Verification failed	XG_ERR_TIME_OUT : Finger input waiting timeout
	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_INVALID_ID / XG_ERR_TIME_OUT	
	XG_INPUT_FINGER:	Prompt the user to put their fingers
	XG_RELEASE_FINGER	Prompt the user to remove their fingers
wCheckSum		Checksum

- **Example**

```
/*
Function: Send verification command and get the returned verification result
pUserID: Return a pointer to the successfully verified user ID
Return value:
XG_ERR_SUCCESS: Verification succeeded, user ID successfully verified is returned via pUserID
XG_ERR_NO_VEIN: Finger vein collection failed, possibly due to improper finger placement
*/

UINT8 FreeVerifyUser(UINT16* pUserID)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };
	UINT32 CardNo = 0;

	if(pUserID)
	{
		//If pUserID is not valid, then 1:1 validation
		bData[0] = (*pUserID)&0xff;
```

```
                bData[1] = ((*pUserID)>>8)&0xff;
                bData[2] = 0;
                bData[3] = 0;
                bData[4] = 0; //0 indicates that it is not grouped
                bData[5] = 0; //0 indicates single user verification, >0 indicates continuous user verification starting
from UserID
                bData[6] = CardNo&0xff; //You can also perform 1:1 verification by card number
                bData[7] = (CardNo>>8)&0xff;
                bData[8] = (CardNo>>16)&0xff;
                bData[9] = (CardNo>>24)&0xff;
        }

        SendPack(XG_CMD_IDENTIFY_FREE, bData, 10);
        gUartByte = 0;
        for(;;)
        {
                ret = RecvPack(NULL, bData, 0); //Here is an unlimited timeout, and keeps waiting for the finger to
be put in to return the verification result
                if(ret != XG_ERR_SUCCESS)
                {
                        PLAY_SOUND(VOICE_IDENT_FAIL);
                        break;
                }
                if(bData[0] == XG_ERR_SUCCESS)
                {
                        PLAY_SOUND(VOICE_IDENT_OK);
                        if(pUserID) *pUserID = bData[1] + (bData[2]<<8);
                        break;
                } else if(bData[0] == XG_INPUT_FINGER) {

                } else if(bData[0] == XG_RELEASE_FINGER) {
                        //Finger vein collection completed, you can now remove your finger
                        PLAY_SOUND(VOICE_KEY);
                } else {
                        //Error handling
                        ret = bData[1];
```

```
                    if(ret == XG_ERR_NO_VEIN)
                    {
                        PLAY_SOUND(VOICE_RIGHT_INPUT);
                    } else {
                        PLAY_SOUND(VOICE_IDENT_FAIL);
                    }
                    break;
                }
            }
            return ret;
}
```

2.2.34 Interrupt or Cancel Registration and Verification Commands

- **Command Code**

  `#define XG_CMD_CANCEL 0x19 //Interrupt the currently executed command`

- **Function**

  Cancel the current executing commands, such as registration, verification, and no timeout verification commands.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x19	XG_CMD_CANCEL
bEncode	0x00	Data encoding, 0
bDataLen	0x00	
bData		
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0xXX	The currently interrupted command or XG_CMD_CANCEL

bEncode	0x00	Data encoding, 0
bDataLen		
bData	Command interrupted	
	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_BREAK_OFF	
	Command execution interrupted	
wCheckSum		Checksum

- **Example**

```
void BreakOffCmd()
{
        SendPack(XG_CMD_CANCEL, NULL, 0);
}
```

2.2.35 UART Port Output Verification Packet after Successful Verification

- **Command Code**

  #define XG_CMD_REPORT 0x4E //UART port output verification packet

- **Function**

  This command is only available for access control and lock controllers and needs to be set as a UART port output. After successful verification, this command packet will be sent through the UART port. This command is issued by the controller and does not require a response.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x4E	XG_CMD_REPORT
bEncode	0x00	Data encoding, 0
bDataLen	16	
bData	bData[0] = 1;	

	bData[1-4] = ID;	
wCheckSum		Checksum

2.2.36 Read Card Number

- **Command Code**

  #define XG_CMD_READ_CARDNO 0x53 //Swipe the card and read the current card number

- **Function**

  Request to swipe the card and read the current card number.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x53	XG_CMD_READ_CARDNO
bEncode	0x00	Data encoding, 0
bDataLen	0	
bData		
wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x53	XG_CMD_READ_CARDNO
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Obtained successfully	CardNo = bData[1] + (bData[2]<<8) + (bData[3]<<16) + (bData[4]<<24);
	bData[0] = XG_ERR_SUCCESS Card number: bData[1]-bData[4] bData[5]-bData[15] is the raw data of the physical card numbers	
	Failed to obtain	Swipe timeout
	bData[0] = XG_ERR_FAIL	

	bData[1] = XG_ERR_TIME_OUT	
wCheckSum		Checksum

- **Example**

```
UINT8 GetCardNo(int* pCardNo)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };
	SendPack(XG_CMD_READ_CARDNO, NULL, 0);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 6000); //The waiting time for swiping the card is 5 seconds
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			if(pCardNo) * pCardNo = bData[1] + (bData[2]<<8) + (bData[3]<<16) + (bData[4]<<24);
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.37 Get Username

- **Command Code**

  #define XG_CMD_GET_USERNAME 0x3C //Get the username

- **Function**

  Get the username

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier

bAddress	0x00	Target device address
bCmd	0x3C	XG_CMD_GET_USERNAME
bEncode	0x00	Data encoding, 0
bDataLen	0x04	

bData	bData[0] = ID&0xff;
	bData[1] = (ID>>8)&0xff;
	bData[2] = (ID>>16)&0xff;
	bData[3] = (ID>>24)&0xff;
		User ID

wCheckSum		Checksum

Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x3C	XG_CMD_GET_USERNAME
bEncode	0x00	Data encoding, 0
bDataLen	16	
bData	Succeeded	
	bData[0] = XG_ERR_SUCCESS bData[1]-bData[14] usernames	
	Failed	
	bData[0] = XG_ERR_FAIL	
wCheckSum		Checksum

- **Example**

```
/*
Function: Get the username
pName: Return the username
Return value:
XG_ERR_SUCCESS: Succeeded
XG_ERR_FAIL: Error
*/
UINT8 GetUserName(UINT8* pName)
{
```

```
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };
	SendPack(0x3C, NULL, 0);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			if(pName) memcpy(pName, &bData[1], 14);
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.38 Set Username

- **Command Code**

  #define XG_CMD_SET_USERNAME 0x3D //Set the username

- **Function**

  Set the username

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x3D	XG_CMD_SET_USERNAME
bEncode	0x00	Data encoding, 0
bDataLen	16	

bData	bData[0] = ID&0xff;
	bData[1] = (ID>>8)&0xff;
	bData[2-15] = Usernames	

wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x3D	XG_CMD_SET_USERNAME
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Succeeded	
	bData[0] = XG_ERR_SUCCESS	
	Failed	
	bData[0] = XG_ERR_FAIL	
wCheckSum		Checksum

- **Example**

```
/*
Function: Set the username
UserId: User ID
pName: Username
Return value:
XG_ERR_SUCCESS: Succeeded
XG_ERR_FAIL: Failed
*/

UINT8 SetUserName (UINT16 UserId, UINT8* pName)
{
	UINT8 ret = 0;
	UINT8 bData[16] = { 0 };

	bData[0] = UserId&0xff;
	bData[1] = (UserId>>8)&0xff;
	memcpy(&bData[2], pName, 14);
	SendPack(0x3D, bData, 16);
	gUartByte = 0;
	ret = RecvPack(NULL, bData, 1000);
	if(ret == XG_ERR_SUCCESS)
	{
		if(bData[0] == XG_ERR_SUCCESS)
		{
			return XG_ERR_SUCCESS;
		}
		else return bData[1];
	}
	return XG_ERR_FAIL;
}
```

2.2.39 Swipe Registration

- **Command Code**

  #define XG_CMD_ENROLL_HD 0x5A //滑动登录

- **Function**

  You can rotate or swipe your finger without lifting it to continuously collect data, or you can place your finger multiple times to complete the collection registration

  Note:

  This command continuously returns multiple packets, and the registration process is only terminated when a return value of XG_ERR_SUCCESS or XG_ERR_FAIL is received.

- **Command Example**

  Send packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x5A	XG_CMD_ENROLL_HD
bEncode	0x00	Data encoding, 0
bDataLen	0x04	

bData	bData[0] = ID&0xff;
	bData[1] = (ID>>8)&0xff;
	bData[2] = (ID>>16)&0xff;
	bData[3] = (ID>>24)&0xff;	

wCheckSum		Checksum

  Return packet:

Field	Data	Remark
wPrefix	0xAABB	Package identifier
bAddress	0x00	Target device address
bCmd	0x5A	XG_CMD_ENROLL_HD
bEncode	0x00	Data encoding, 0
bDataLen		
bData	Logged in successfully	
	bData[0] = XG_ERR_SUCCESS	
	Login failed	Login failed:

	bData[0] = XG_ERR_FAIL bData[1] = XG_ERR_INVALID_ID/ XG_ERR_NOT_ENOUGH/ XG_ERR_TIME_OUT/ XG_ERR_DUPLICATION_ID bData[3] and bData[0] are touch states =0 is no touch, 1 is touched	XG_ERR_INVALID_ID: Invalid ID XG_ERR_NOT_ENOUGH : No enough space to log in XG_ERR_TIME_OUT : Finger input waiting timeout XG_ERR_DUPLICATION_ID : A duplicate user has already logged in
	XG_INPUT_FINGER:	Prompt the user to put their fingers
	XG_RELEASE_FINGER	Prompt the user to swipe or remove their fingers
wCheckSum		Checksum

- **Example**

```
/*
Function: Send registration command to add finger vein user
UserID: The ID of the specified registered user
Return value:
XG_ERR_SUCCESS: Registered successfully
*/

UINT8 EnrollUser_HD(UINT16 UserID)
{
        UINT8 ret = 0;
        UINT8 bData[16] = { 0 };
        UINT8 bTouch1 = 0xff, bTouch2 = 0xff;

        bData[0] = UserID&0xff;
        bData[1] = (UserID>>8)&0xff;
        bData[2] = 0;
        bData[3] = 0;
        SendPack(0x5A, bData, 4); //Swiping register command
        gUartByte = 0;
        for(;;)
```

```
	{
		ret = RecvPack(NULL, bData, 6000); //Finger detection timeout is 5 seconds
		if (ret == XG_ERR_SUCCESS) { //Received a return packet
		{
			PLAY_SOUND(VOICE_REG_FAIL);
			break;
		}
		if(bData[0] == XG_ERR_SUCCESS)
		{
			PLAY_SOUND(VOICE_REG_OK);
			break;
		} else if(bData[0] == XG_INPUT_FINGER) {
			//bData[3] and bData[4] are two touch states, and the touch status
change prompt
			if(bTouch1 != bData[3] || bTouch2 != bData[4])
			{
				if(bData[3] == 0 && bData[4] == 0) PLAY_SOUND(VOICE_INPUT);/
/Message("Please place your finger");
				if(bData[3] == 1 && bData[4] == 0) PLAY_SOUND(VOICE_BEEP3);
//Message("Touch 1 is not attached tightly");
				if(bData[3] == 0 && bData[4] == 1) PLAY_SOUND(VOICE_BEEP4);
//Message("Touch 2 is not attached tightly");
				if(bData[3] == 1 && bData[4] == 1) PLAY_SOUND(VOICE_BEEP1);
//Message("Please swipe your finger...");
				bTouch1 = bData[3];
				bTouch2 = bData[4];
			}
		} else if(bData[0] == XG_RELEASE_FINGER) {
			PLAY_SOUND(VOICE_KEY);//Message("Please swipe your finger...");
		} else {
			//Error handling
			ret = bData[1];
			if(ret == XG_ERR_INVALID_ID)
			{
				//Invalid ID
				PLAY_SOUND(VOICE_PINPUT_ID); //ID error
```

```
                    break;
                } else if(ret == XG_ERR_NOT_ENOUGH) {
                    //This user has already registered
                    PLAY_SOUND(VOICE_USER_FULL);
                    break;
                } else if(ret == XG_ERR_TIME_OUT) {
                    //Finger detection timeout
                    PLAY_SOUND(VOICE_REG_FAIL);
                    break;
                } else if(ret == XG_ERR_DUPLICATION_ID) {
                    //Duplicate registration
                    PLAY_SOUND(VOICE_OVER_REG);
                    break;
                } else if(ret == XG_ERR_NO_SAME_FINGER) {
                    //It's not the same finger, if the maximum number of collections ha
s not been reached, the collection can be continued
                    PLAY_SOUND(VOICE_RIGHT_INPUT);
                } else if(ret == XG_ERR_NO_VEIN) {
                    //No finger veins were detected, and if the maximum number
of collections has not been reached, the collection can be continued
                    PLAY_SOUND(VOICE_RIGHT_INPUT);
                } else {
                    PLAY_SOUND(VOICE_REG_FAIL);
                    break;
                }
            }
        }
        return ret;
}
```

3. Appendix I: Error Code List

```
/*****************************Error Codes*************************************/
#define XG_ERR_SUCCESS              0x00 //Operation succeeded
#define XG_ERR_FAIL                 0x01 //Operation failed
#define XG_ERR_COM                  0x02 //Communication error
#define XG_ERR_DATA                 0x03 //Data validation error
#define XG_ERR_INVALID_PWD          0x04 //Password error
#define XG_ERR_INVALID_PARAM        0x05 //Parameter error
#define XG_ERR_INVALID_ID           0x06 //ID error
#define XG_ERR_EMPTY_ID             0x07 //The specified ID is empty (no login data)
#define XG_ERR_NOT_ENOUGH           0x08 //No enough space to log in
#define XG_ERR_NO_SAME_FINGER       0x09 //Not the same finger
#define XG_ERR_DUPLICATION_ID       0x0A //Duplicate login ID
#define XG_ERR_TIME_OUT             0x0B //Finger input waiting timeout
#define XG_ERR_VERIFY               0x0C //Authentication failed
#define XG_ERR_NO_NULL_ID           0x0D //No empty ID available
#define XG_ERR_BREAK_OFF            0x0E //Operation interrupted
#define XG_ERR_NO_CONNECT           0x0F //Not connected
#define XG_ERR_NO_SUPPORT           0x10 //This action is not supported
#define XG_ERR_NO_VEIN              0x11 //No vein data
#define XG_ERR_MEMORY               0x12 //Out of memory
#define XG_ERR_NO_DEV               0x13 //Device does not exist
#define XG_ERR_ADDRESS              0x14 //Device address error
#define XG_ERR_NO_FILE              0x15 //File not found
#define XG_ERR_VER                  0x16 //Version error
#define XG_ERR_BREAK_CARD           0x17 //Swipe interrupted

/*****************************Status Codes************************************/
#define XG_INPUT_FINGER             0x20 //Request to place finger
#define  XG_RELEASE_FINGER          0x21  //Request  to  remove  finger
```

4. Appendix II: Voice Playlist

VOICE_REG_OK = 0,//Registration succeeded

VOICE_USER_FULL = 1,//Records are full

VOICE_REG_FAIL = 2,//Registration failed

VOICE_OVER_REG = 3, //Duplicate registration

VOICE_ADMIN_AUTH_OK=8, //Administrator authentication succeeded

VOICE_ADMIN_AUTH_FAIL=9, //Administrator authentication failed

VOICE_ERROR_CARD=10, //Card number error

VOICE_PWD_ERROR=13, //Password error

VOICE_PWD_REG_OK = 14,//Password registration succeeded

VOICE_PWD_REG_FAIL = 15,//Password registration succeeded

VOICE_ENROLL_ADMIN = 18,//Please register as administrator

VOICE_PINPUT_PWD = 19,//Please enter the password

VOICE_PINPUT_ID = 20,//Please enter the user ID

VOICE_ADMIN_AUTH = 21,//Please verify the administrator

VOICE_INPUT_FINGER_AGAIN = 23,// Please place your finger again

VOICE_RETRY = 24,//Please try again

VOICE_INPUT_AGAIN = 25,//Please input again

VOICE_RIGHT_INPUT = 26, //Please place your finger correctly

VOICE_INPUT = 27,//Please gently place your finger naturally

VOICE_DEL_OK = 28,//Deleted successfully

VOICE_DEL_FAIL = 29,//Deletion failed

VOICE_THANKS = 30, //Thank you

VOICE_DELING = 31,//Deleting

VOICE_IDENT_FAIL = 32,//Verification failed

VOICE_IDENT_OK = 33, //Verification succeeded

VOICE_UNLOCK = 34, //Unlocked

VOICE_BEEP1 = 35,

VOICE_KEY = 36,

VOICE_BEEP3 = 37,

VOICE_BEEP4 = 38,

VOICE_ALARM = 39, //39 Alarm sound

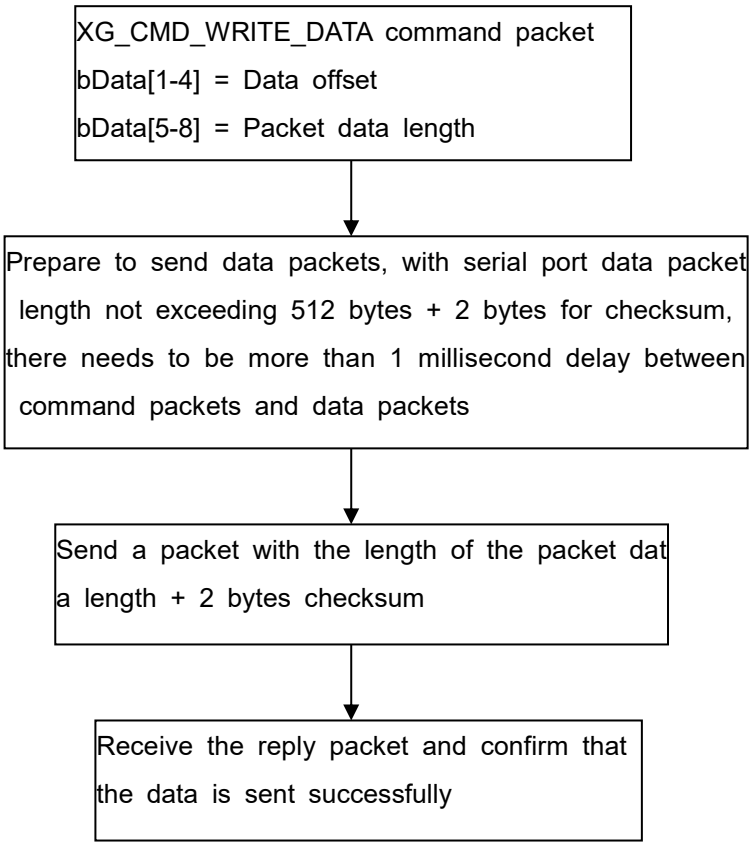
VOICE_OK_KEY_CONFIRM = 40, //Press OK key to confirm

VOICE_POWER_LOW = 41, //Low battery, please replace the battery

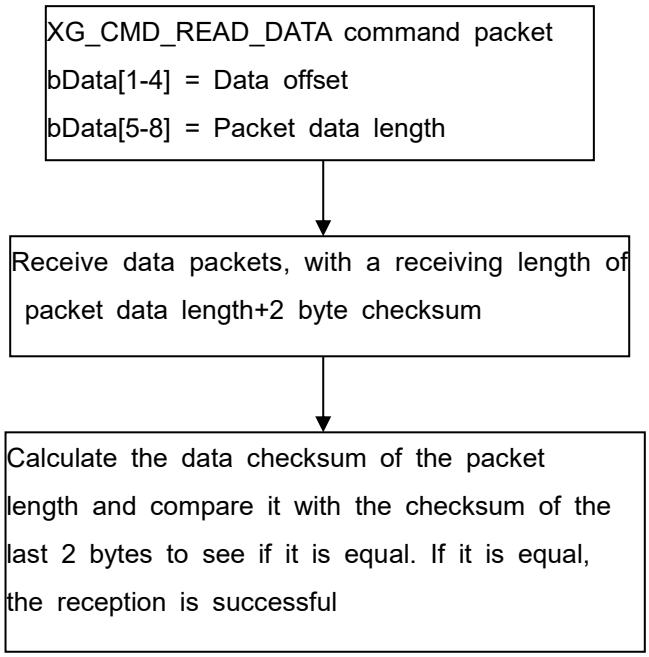
```
VOICE_ADMIN = 42,//Administrator
VOICE_PWD = 43,//Password
VOICE_CARDNO = 44,//Swipe the card
VOICE_INPUT_CARD = 45, //Please swipe your card
VOICE_LOCK = 46, //Locked
VOICE_PLS_INPUT_ADMIN_PWD = 47, //Please enter the administrator password
VOICE_ADMIN_FULL = 49, //Administrator is full
VOICE_OK_KEY_DEL_CONFIRM = 50, //Please press the OK key to confirm deletion
VOICE_HELP = 51,
VOICE_HELP1 = 52,
```

5. Appendix III: Function Implementation Examples

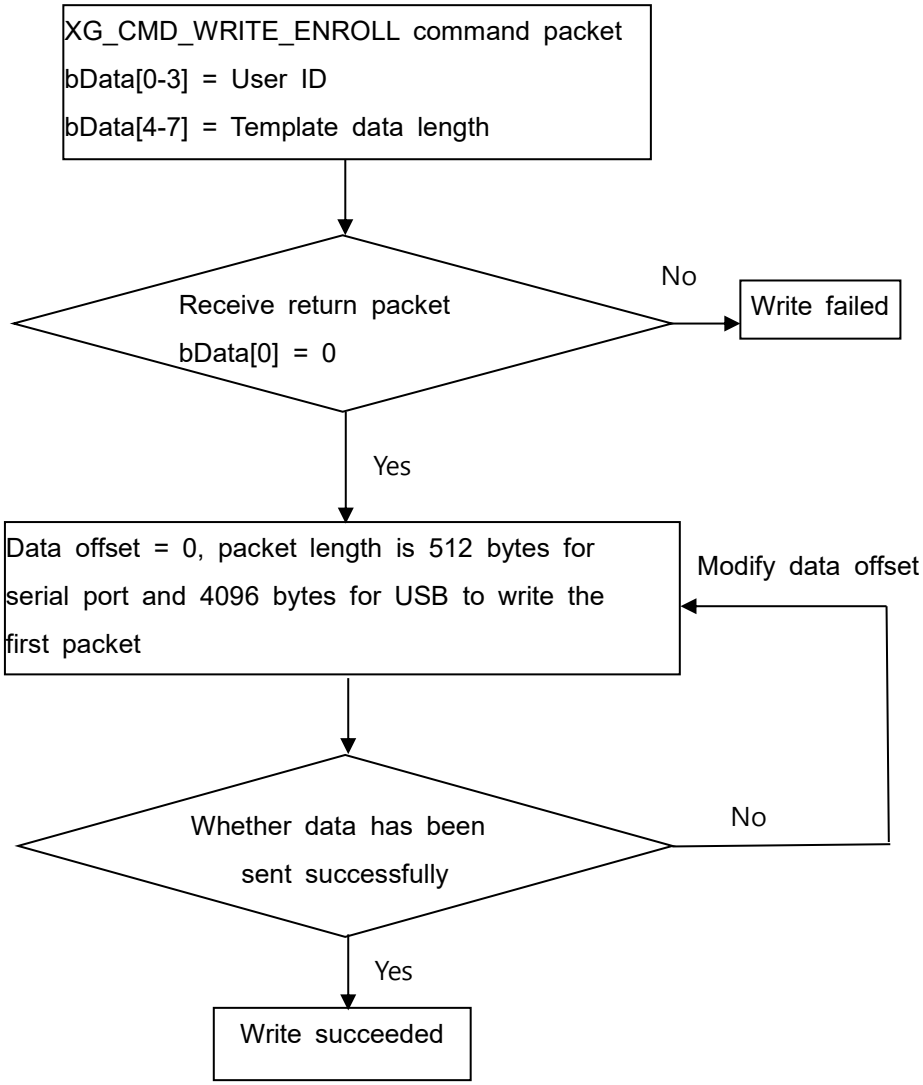
5.1 The Sending Process of Data Packets



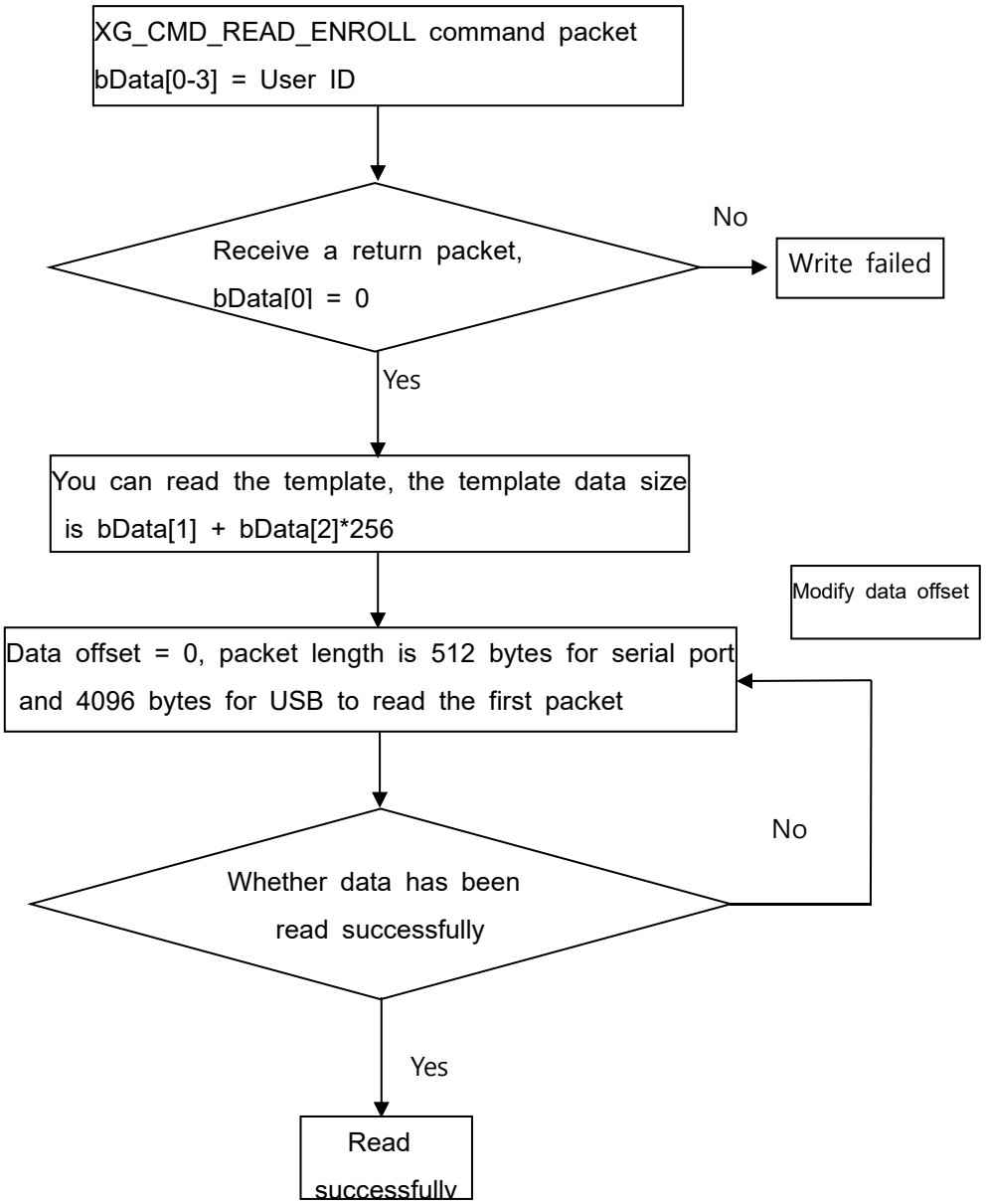
5.2 The Receiving Process of Data Packets



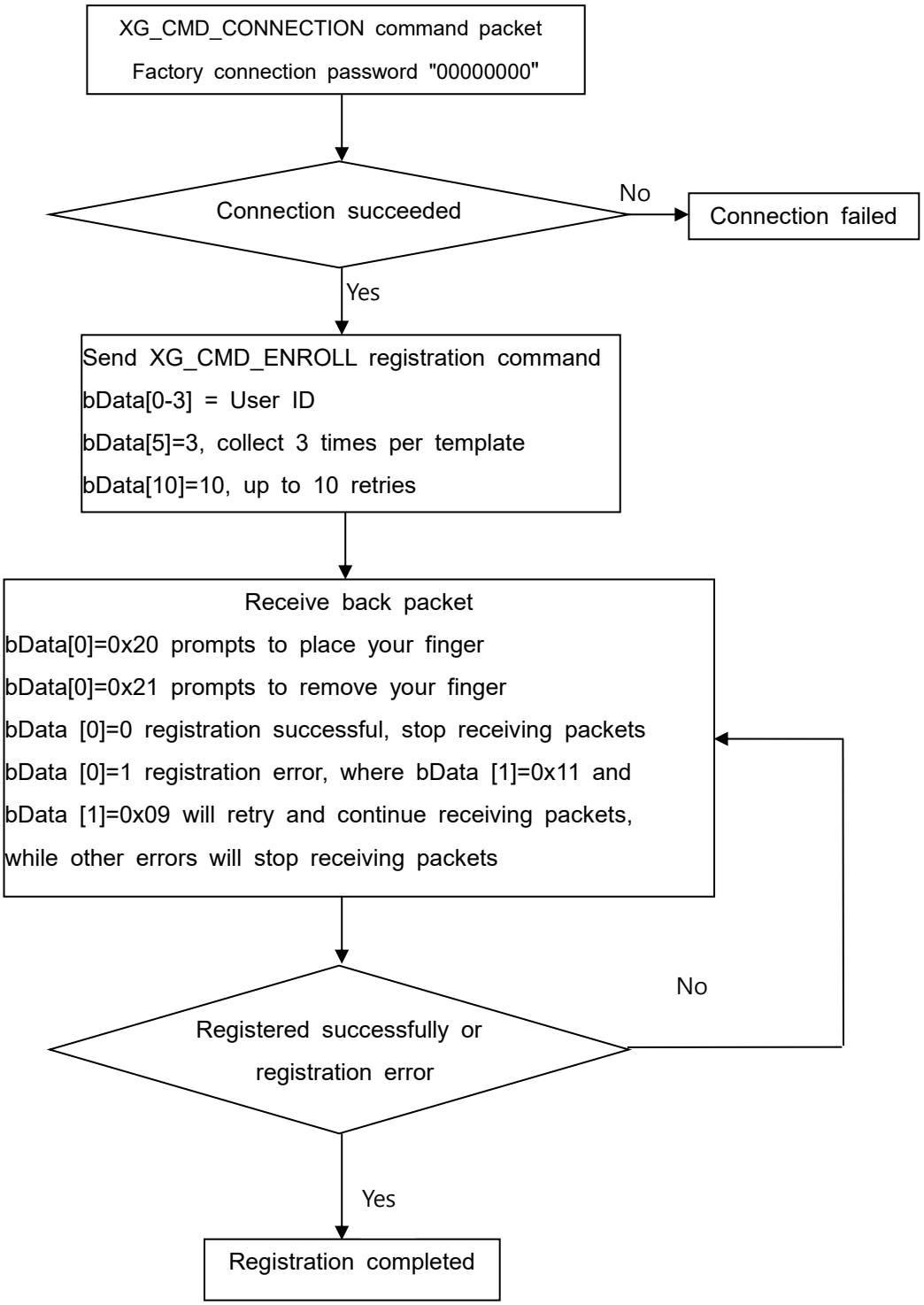
5.3 Template Writing Process



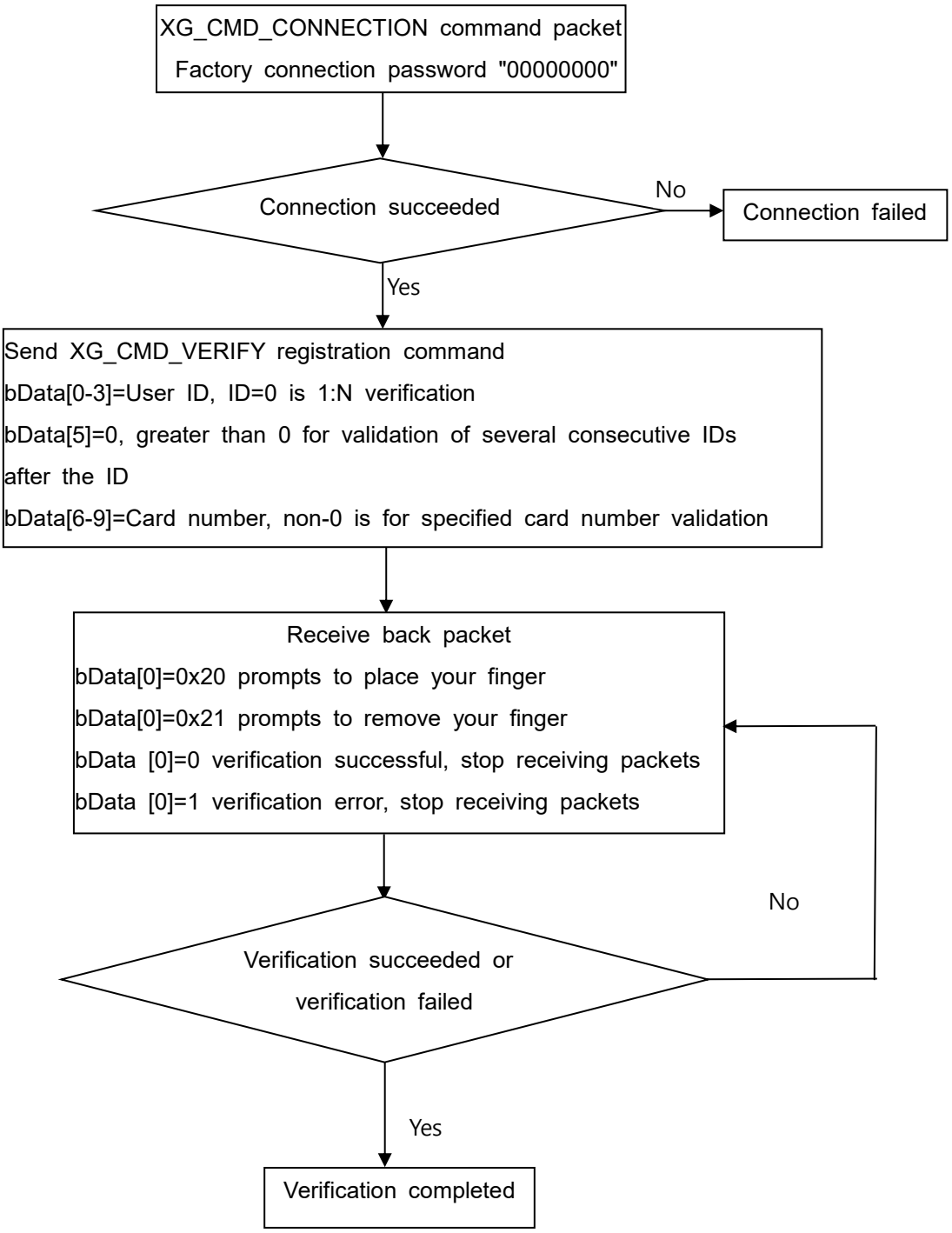
5.4 Template Reading Process



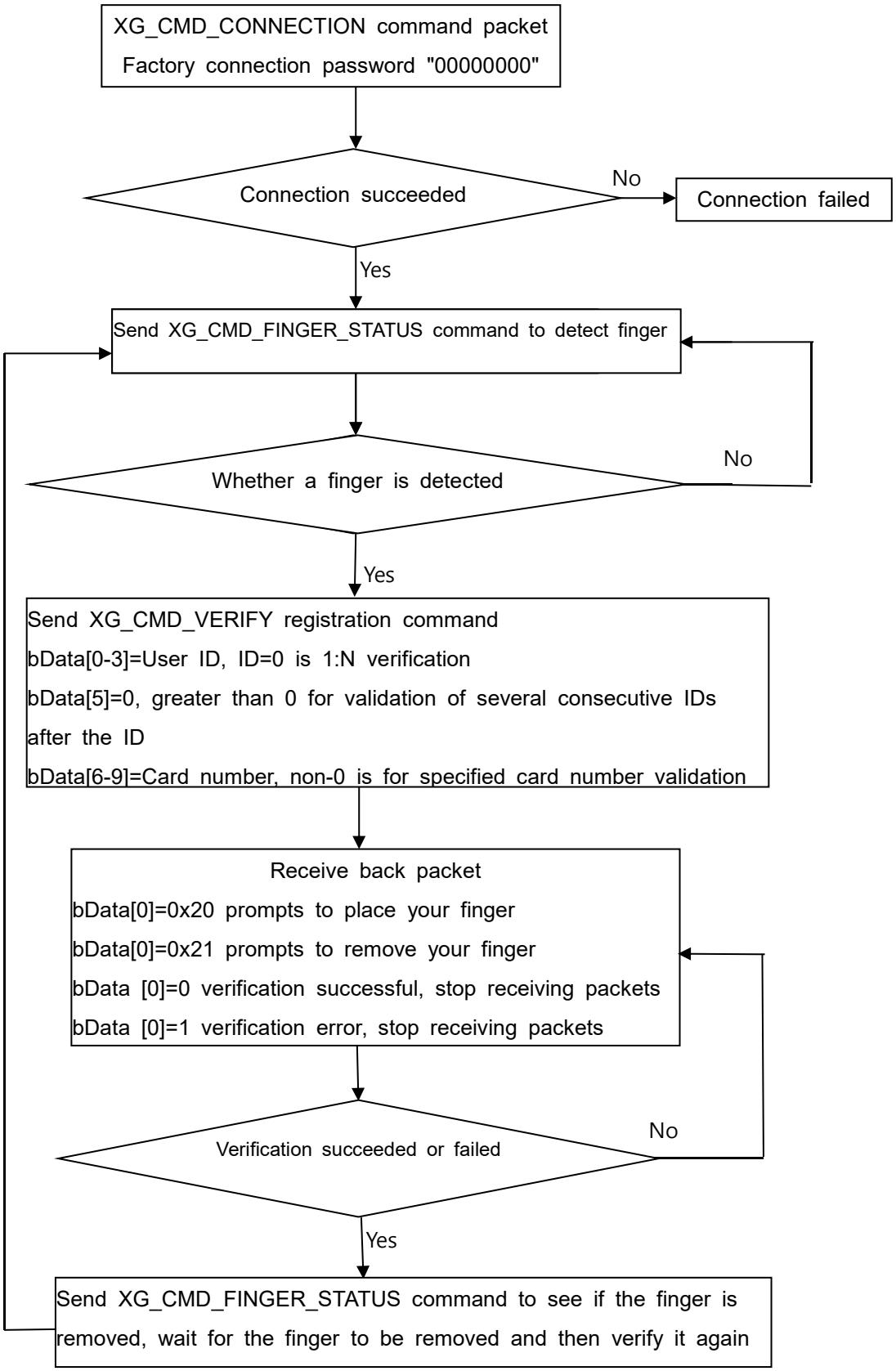
5.5 Registration Process



5.6 Verification Process



5.7 Online 1:N Verification Process



5.8 Swipe Registration Process