

# **Description of the BioUPI library (version 1.01.02)**

## ***Purpose***

This document describes the BioUPI Software Development Kit (SDK) and the functions are exported by the SDK.

The SDK is designed for the fingerprint scanner UFIS and EFIS, its quality estimation, minutia extraction, fingerprint matching, template creation and etc.

## ***SDK Files***

The SDK consists of following files:

bioupic.h	SDK header file
bioupic.lib	SDK import library (in C)
bioupic.dll	SDK

## ***SDK functions***

This section lists all the externally visible SDK functions.

### **InitBioUPI**

Syntax:       void    BIOUPIC       InitBioUPI(short \_type);

Description:   The function makes configuration an image processing functions depend on type of fingerprint image. It's necessary to call this function firstly before other functions of BioUPI library.

Parameters:    \_type – type of fingerprint image  
                  Type of images:  
                  BMF  
                  THOMSON  
                  MBF  
                  TST

Return values: none

### **CreateTemplate**

Syntax:       short CreateTemplate (unsigned char \*img, int \_W, int \_H, unsigned char \*ptmp, short & status)

Description:   To create a template from the quantity of several fingerprint images, representing the same finger.

Parameters:   img - pointer to fingerprint image array,

ptmp - pointer to array, for created template location,  
\_W – the width of the image (280 for THOMSON),  
\_H – the height of the image (440 for THOMSON),  
status = 0, for function initialization,  
= 1, without function initialization.

Return values: -2 - bad image quality;  
-1 - error;  
0 - template is not created;  
1 - template is created successfully.

#### **CreateFPCode**

Syntax: short CreateFPCode(unsigned char \*img, int \_W, int \_H, unsigned char \*pfpc)

Description: Create a fingerprint code of the fingerprint image

Parameters: img - pointer to fingerprint image array,  
\_W – the width of the image (280 for THOMSON),  
\_H – the height of the image (440 for THOMSON),  
pfpc - pointer to array, for created fingerprint code location;

Return values: -2 - bad image quality;  
-1 - error;  
> 0 - length of the fingerprint code

#### **Verify**

Syntax: short Verify(unsigned char \*img, int \_W, int \_H, unsigned char \*ptmp)

Description: Match actual captured fingerprint image with one pre-selected template.

Parameters: img - pointer to fingerprint image array,  
\_W – the width of the image (280 for THOMSON),  
\_H – the height of the image (440 for THOMSON),  
ptmp - pointer to the array, with one pre-selected template.

Return values: -1 – internal error (alloc memory,...)  
-2 – bad image quality;  
-3 – template CRC error;  
-4 – not correct number of point  
-5 – not correct version of template  
value from 0 to 100 - the similarity percentage to stored template (if value  
≥33 – successful recognition).

#### **VerifyFPCode**

Syntax: short VerifyFPCode(unsigned char \*pfpc, unsigned char \*ptmp)

Description: Match fingerprint code of the fingerprint image with one pre-selected template.

Parameters: pfp - pointer to the array, with one pre-selected fingerprint code.  
ptmp - pointer to the array, with one pre-selected template.

Return values: -3 - template CRC error;  
-1 - error;  
value from 0 to 100 - the similarity percentage to stored template (if value  
≥33 – successful recognition).

### **IdentifyInit**

Syntax: short IdentifyInit (unsigned char \*img, int \_W, int \_H)

Description: Initializes the internal variables for IdentifyFind routine successful start.

Parameters: img - pointer to fingerprint image array,  
\_W – the width of the image (280 for THOMSON),  
\_H – the height of the image (440 for THOMSON),

Return values: -2 - bad image quality;  
-1 - error,  
0 - operation successful.

### **IdentifyFind**

Syntax: short IdentifyFind (unsigned char \*ptmp)

Description: Match actual captured fingerprint image with one pre-selected template.

Parameters: ptmp - pointer to array, with one pre-selected template;

Return values: -1 – internal error (alloc memory,...)  
-3 – template CRC error;  
-4 – not correct number of point  
-5 – not correct version of template  
  
value from 0 to 100 - the similarity percentage to stored template (if value  
≥33 – successful recognition).

### **GetLengthTemplate**

Syntax: short GetLengthTemplate(unsigned char \*ptmp)

Description: Get the true length of a template.

Parameters: ptmp - pointer to array, with one pre-selected template;

Return values: -1 – internal error (alloc memory,...)  
-3 – template CRC error;  
-4 – not correct number of point  
-5 – not correct version of template

value is the length in bytes of pre-selected template.

#### **GetLengthFPCode**

Syntax: short GetLengthFPCode(unsigned char \*pfpc)

Description: Get the true length of a fingerprint code.

Parameters: handle - handle of the scanner  
pfpc - pointer to 1D-array, with one pre-selected fingerprint code;

Return values: -1 – internal error (alloc memory,...)  
-3 – template CRC error;  
-4 – not correct number of point  
-5 – not correct version of template  
Value is the length in bytes of pre-selected fingerprint code.

#### **GetQualityTemplate**

Syntax: short GetQualityTemplate(unsigned char \*ptmp)

Description: Get the quality of template.

Parameters: ptmp - pointer to array, with one pre-selected template;

Return values: -1 – internal error (alloc memory,...)  
-3 – template CRC error;  
-4 – not correct number of point  
-5 – not correct version of template  
value from 0 to 100 - the percentage quality of template.  
(value from 50 to 100 is nice for following recognition)

## Code Samples

This section contains a sample programs that demonstrate the usage of the complete set of SDK calls.

### Sample 1. Template creation.

```
#include <stdlib.h>
#include <stdio.h>
#include "BioUPI.h"

void main(void)
{
    short status, rc;
    unsigned char ptmp[2048];

    InitBioUPI(THOMSON);

    status = 0;
    while(1) {
        // Add your code to get the fingerprint image to img

        //      after first call the status will be changed
        rc = CreateTemplate(img, 280, 440, ptmp, &status);
        if ( rc == -1 ) {
            printf("Memory allocation error\n");
            exit(2);
        }
        if ( rc == 1 ) {
            printf("Template was created successfully\n");
            break;
        }
    }
}
```

### Sample 2. Verification procedure.

```
#include <stdlib.h>
#include <stdio.h>
#include "BioUPI.h"

void main(void)
{
    short    rc;
    unsigned char ptmp[2048];

    InitBioUPI(TST);

    //      add your code to get image to img

    //      add your code to read template to ptmp

    rc = Verify(img, 280, 440, ptmp);

    if ( rc == -1 ) {
        printf("Memory allocation error\n");
        exit(2);
    }
}
```

```

}
if(rc>=33)
    printf("Recognition is successful\n");
else
    printf("Recognition is not successful\n");
}

```

### **Sample 3. Identification procedure.**

```

#include <stdlib.h>
#include <stdio.h>
#include "BioUPI.h"

void main(void)
{
    short    rc;
    unsigned char ptmp[2048];

    InitBioUPI(THOMSON);

    //      add your code to get image to img

    IdentifyInit(img, 280, 440 );

    while(1) {

        //      add your code to read template to ptmp

        rc = IdentifyFind(ptmp);

        if ( rc == -1 ) {
            printf("Memory allocation error\n");
            exit(2);
        }
        if(rc>=33)
            printf("Recognition is successful\n");
        else
            printf("Recognition is not successful\n");

    }
}

```