BCLib User Documentation

Generated by Doxygen 1.5.5

Fri Apr 23 15:24:05 2010

Contents

1 BCLib User Documentation

1.1 Principles of Operation

The scope of this document is to provide a detailed description of the BCLib API.

BCLib is UPEK's library, providing access to biometric data conversion functions. The goal of the library is to provide customers with a tool for converting UPEK biometric data to ISO/ANSI standard formats and vice versa. Along with this, the library can be used for conversions among various data formats used in UPEK software, as well as for preparing third party data for internal testing. Fingerprint templates and image data can be subject of the conversion.

The library can change only the format of data, not its content, for instance it is not possible to change fingerprint template size or color depth of a fingerprint image.

For purposes of this library, one instance of biometric data (both input or output) is split into two parts:

- Inner data - which contain the actual biometric information
- Envelope - which represents the service information added by some layers of the software stack (e.g. BioAPI). The envelope might be missing in some cases.

The library functions can be called with various combinations of data and envelopes on input and output. Not all combinations are supported as some of them do not make sense and some of them have not been implemented yet.

1.1.1 Source of Suplementary Information

Some formats contain more information than the others. Therefore some additional source of information for conversion to a richer format is necessary. An optional information structure can be provided using parameter **pSuplementaryInfo** of functions BCConvertTemplate and BCConvertImage.

The structure can also be used for changing values already contained in template, for instance change coordinate units. Minutiae data are modified (if necessary) in order to stay consistent. Image conversion does not allow changes causing image conversions (changing dimensions or pixel bit depth).

In some particular cases of conversion, it might be necessary to use a data which contains concrete valid values in some fields, otherwise the output format cannot be generated and the error code (BCERR_MISSING_INFORMATION) is returned in that case. Most of the data fields can be left to default values or eventually some concrete information can be provided if it is known to the caller.

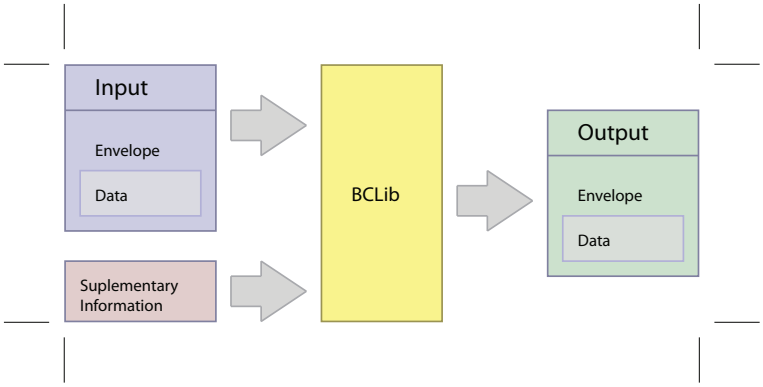

Figure 1: Schema of BCLib Operation

1.1.2 Memory Allocation Handling

Buffer for output data must be provided by the caller. If the caller wants to allocate exact amount of memory, the following method for determining necessary memory size is available: conversion function is called first with NULL value of the pointer to output buffer. Necessary memory size is returned in a variable pointed to by parameter **pOutputLength**. Now it is possible to allocate the appropriate memory buffer and call the real conversion. See sample code fragment:

```
sint32 retVal;
uint32 lenOutput;
char *pOutputBuffer = NULL;
```

```
// Call conversion function to get output buffer size

retVal = BCConvertTemplate( BCTE_NONE, BCT_ANSI,
                            pInputTemplate, lenInputTemplate,
                            BCTE_NONE, BCT_UPEK_ALPHA,
                            NULL, &lenOutput, NULL );
if ( retVal == BCERR_OK )
{
    // Allocate the buffer

    pOutputBuffer = malloc( lenOutput );

    // Call real conversion

    retVal = BCConvertTemplate( BCTE_NONE, BCT_ANSI,
                                pInputTemplate, lenInputTemplate,
                                BCTE_NONE, BCT_UPEK_ALPHA,
                                pOutputBuffer, &lenOutput, NULL );
}
```

1.1.3 Template Conversions

Template conversion accepts the following formats on input and output:

- UPEK proprietary template (legacy, alpha, beta) - BCT_UPEK_AUTO, BCT_UPEK_LEGACY, BCT_UPEK_ALPHA, BCT_UPEK_BETA
- ANSI INCITS 378-2004 - (Finger Minutiae Format for Data Interchange) - BCT_ANSI
- ISO/IEC 19794-2 (Biometric data interchange formats - Finger Minutiae Record) - BCT_ISO_FMR
- ISO/IEC 19794-2 (Biometric data interchange formats - Finger Minutiae Card format, normal, format type 22 by ISO/IEC 19794-2:2005 Technical Corrigendum 1) - BCT_ISO_FMC_NORMAL
- ISO/IEC 19794-2 (Biometric data interchange formats - Finger Minutiae Card format, compact, format type 6 by ISO/IEC 19794-2:2005 Technical Corrigendum 1) - BCT_ISO_FMC_COMPACT

Template conversion accepts the following envelopes on input and output:

- None (raw data) - BCTE_NONE
- UPEK's PT_BIR - BCTE_PT_BIR
- UPEK's ABS_BIR - BCTE_ABS_BIR
- BioAPI BIR - BCTE_BIOAPI_BIR

(See below for more information on template formats.)

1.1.4 Image Conversions

Image conversion accepts the following formats

- Raw grayscale image data, 1 byte per pixel stored by rows, starting at upper left corner - BCI_RAW
- ANSI INCITS 381-2004 - (Finger Image-Based Data Interchange Format) - BCI_ANSI
- ISO/IEC 19794-4 - (Biometric data interchange formats - Finger image data) - BCI_ISO

Image conversion accepts the following envelopes

- None (naked data) - BCIE_NONE
- UPEK's ABS_IMAGE - BCIE_ABS_IMAGE
- UPEK's PT_DIF - BCIE_PT_DIF
- BIOAPI_GUI_BITMAP (not implemented in current version) - BCIE_BIOAPI_GUI_BITMAP

(See below for more information on image formats.)

1.2 Data Formats for Templates

1.2.1 UPEK Proprietary Data Format

Proprietary binary format used by all UPEK templates. It contains raw data of one finger without additional information such as finger position or capture equipment. Upper layers of UPEK software usually add some envelope to wrap raw data (see description of envelopes).

There are two UPEK standard formats used now, **alpha** and **beta**. Additionally, the older **legacy** template format is maintained for the purpose of compatibility with older devices, where it was the only template format supported. This template format should be considered obsolete.

The alpha template format contains the minimal sufficient set of biometric information needed for a reasonable biometric performance and represents the best trade-off between template size and biometric performance. Its content is intended as "core" information that would be included in any future template format and every future version of UPEK biometric systems has to be able to process alpha type content.

The beta template format is an extension to the alpha format. Additionally, it contains optional block(s) of additional biometric information. Beta templates are in general able to achieve better biometric performance, but at the cost of an increased template size. The full benefit of the beta template can be applied only if it is matched against another beta template.

A special value BCT_UPEK_AUTO can be used instead of concrete UPEK template format. The library can autodetect data format on input and/or choose a suitable format for output.

1.2.2 ISO/IEC 19794-2 and ANSI INCITS 378-2004

Binary format defined by ISO/IEC and ANSI, with common subset of data roughly equivalent to UPEK proprietary data, but containing some additional fields, which are not contained in UPEK data (for instance finger position). On the other hand, UPEK proprietary data can contain some additional information. This additional information can be stored in ANSI/ISO format in extended data area.

Formats defined in ISO/IEC 19794-2 and ANSI INCITS 378-2004 standards are equivalent but not identical:

- ISO format does not contain CBEFF product identifier.
- ISO format uses angle unit equal to 360/256 degrees, ANSI angle unit is equal to 2 degrees.

Anyway, because this difference is not too significant for this specification, both formats will be mostly referred together as ANSI/ISO format.

1.2.3 ISO/IEC 19794-2 Card Formats

The standard defines two card related (MOC) formats. Conversion to these formats cannot be lossless, because some data fields do not provide the necessary precision, esp. for "compact size" format.

Limitatons of ANSI/ISO template format processing

- Only the first finger view is processed, both on input and on output. This means - for example - by converting template in ANSI format with more finger views to ISO format, all views except the first finger view are lost.
- Extended ANSI/ISO data blocks are ignored.
- Minutiae in ISO Finger Minutiae Card format output data are sorted by carthesian coordinates, top down and left to right. Currently, no other option is available.

- Template conversion currently ignores any extended data blocks possibly present in ANSI/ISO template, only basic minutia block is taken in account.

1.3 Envelopes for Templates

1.3.1 BioAPI_BIR

Defined in BioAPI standard, used in UPEK BioAPI BSP. Adds format identification, format ID, template quality and purpose.

1.3.2 PT_BIR

Used in PTAPI. Data block is binary compatible with BioAPI's BioAPI_BIR. The only difference is, that in BioAPI_BIR the data are divided into four separate memory blocks, while PT_BIR keeps all the data together.

1.3.3 ABS_BIR

Contains the same information as BioAPI BIR.

1.4 Data Formats for Images

1.4.1 Grayscale image in sequence of bytes

Raw data of a grayscale image, one byte per pixel, 256 gray levels. It is stored by lines, no line alignment is used.

1.4.2 ISO/IEC 19794-4 and ANSI INCITS 381-2004

Again, we are talking about these formats together. The only difference between these formats is a presence of CBEFF product identifier in ANSI INCITS 381-2004.

Limitatons of ANSI/ISO image format processing

- Only the first image record is processed, both on input and on output. This means - for example - by converting image in ANSI format with more finger image records to ISO format, all views except the first finger view are lost.
- Combining ANSI/ISO image formats with envelopes is not possible, this data format can be read or written only without any envelope. On the other hand, "envelope" formats can contain only raw image data.
- No image compression/decompression is implemented for ANSI/ISO format

1.5 Envelopes for Images

1.5.1 ABS_IMAGE

Used in BSAPI. Contains a header defining width, height, color depth and DPI, plus image data. See BSAPI documentation for detailed description.

1.5.2 BioAPI_GUI_BITMAP

BioAPI returns audit data in this format. Contains a header defining width and height plus image data. See BioAPI documentation for detailed decsription.

1.5.3 PT_DIF (Dynamic Image Format)

Used in PTAPI. Universal extensible format for storing grayscale images. See PTAPI documentation for detailed decsription.

1.6 Image Conversions

1.7 Implementation Notes

The implementation of the library is thread-safe, unless specified otherwise for the given variant/platform.

1.8 Other

For a detailed description please see the following pages:

- API Overview
- Error Codes
- Sample of BCLib use for template conversion
- Sample of BCLib use for image conversion

2 Sample of BCLib use for template conversion

```
#include "errno.h"
#include "stdio.h"
#include "stdlib.h"

#include "bclib.h"

//------------------------------------------------------------------------------
// Temporary definitions for quick test

char gUsageInfo[] = "\nUsage: %s <input-template-file> <output-template-file>";

//------------------------------------------------------------------------------
// Suplementary info containing data not present in UPEK template

BCTemplateInfoType_1 gSuplementInfoTpl =
{
    // Structure version, must be 1.

    1,

    // Capture system CBEFF product identifier.

    0x12,

    // Compliance bits according to ANSI standard paragraph 6.4.5.

    0,

    // Identificaiton of equipment assigned by its manufacturer

    0,

    // Original image width (in pixels) - present in UPEK template

    0,

    // Original image height (in pixels) - present in UPEK template
```

```
    0,

    // Original image horizontal resolution (in dpi) - present in UPEK template

    0,

    // Original image vertical resolution (in dpi) - present in UPEK template

    0,

    // Finger position as in ANSI/ISO (2 = right index finger)

    2
};

//-----------------------------------------------------------------------------
// Allocate input buffer and load data from file

sint32 LoadDataFromFile( const char *pFileName, unsigned char **ppBufer, uint32 *pLenData )
{
    sint32 retVal = 0;
    FILE *fil = NULL;

    // Open file

    fil = fopen( pFileName, "rb" );
    if ( fil != NULL )
    {
        sint32 fileLength = 0;

        // Get file length

        fseek( fil, 0, SEEK_END );
        fileLength = ftell( fil );

        if ( fileLength > 0 )
        {
            // Allocate buffer for data

            *ppBufer = (unsigned char *) malloc( fileLength );
            *pLenData = (uint32) fileLength;

            // Seek back to the beginning of file

            fseek( fil, 0, SEEK_SET );

            // Read data from file

            fread( *ppBufer, 1, fileLength, fil );
        }
        else
        {
            // File length zero, return error

            retVal = -2;
        }

        // Close the file

        fclose( fil );
    }
    else
    {
        // File open error

        retVal = -1;
    }
```

```
    return retVal;
}

//-----------------------------------------------------------------------------
// Store data to file

sint32 StoreDataToFile( const char *pFileName, const unsigned char *pBuf, uint32 lenData )
{
    sint32 retVal = 0;
    FILE *fil = NULL;

    // Open the file

    fil = fopen( pFileName, "wb" );
    if ( fil != NULL )
    {
        // File opened, write data

        fwrite( pBuf, 1, lenData, fil );

        // Close the file

        fclose( fil );
    }
    else
    {
        // File open error

        retVal = -1;
    }

    return retVal;
}

//-----------------------------------------------------------------------------

int main( int argc, char* argv[] )
{
    sint32 retVal = 0;

    uint32 versionMajor = 0;
    uint32 versionMinor = 0;

    unsigned char *pInputBuffer = NULL;
    unsigned char *pOutputBuffer = NULL;
    uint32 lenInput = 0;
    uint32 lenOutput = 0;

    // Get library version and print it

    retVal = BCGetVersion( &versionMajor, &versionMinor );
    printf( "\nBiodata Conversion Library version %d.%d convert template sample\n",
        versionMajor, versionMinor );

    // Check command line parameters count

    if ( argc <= 2 )
    {
        printf( gUsageInfo, argv[0] );
    }
    else
    {
        // Read template from file

        retVal = LoadDataFromFile( argv[1], &pInputBuffer, &lenInput );
        if ( retVal < 0 )
```

```
        {
            printf( "Error reading file '%s' (code %d)\n", argv[1], retVal );
        }
        else
        {
            // Get output data langth

            retVal = BCConvertTemplate( BCTE_NONE, BCT_UPEK_AUTO,
                pInputBuffer, lenInput,
                BCTE_NONE, BCT_ANSI,
                NULL, &lenOutput,
                NULL );

            if ( retVal != BCERR_OK )
            {
                // Cannot get output lenght, report error

                printf( "Error getting output length (code %d)\n",
                    retVal );
            }
            else
            {
                // Allocate memory

                pOutputBuffer = (unsigned char *) malloc( lenOutput );

                // Call conversion

                retVal = BCConvertTemplate( BCTE_NONE, BCT_UPEK_AUTO,
                    pInputBuffer, lenInput,
                    BCTE_NONE, BCT_ANSI,
                    pOutputBuffer, &lenOutput,
                    &gSuplementInfoTpl );

                if ( retVal == BCERR_OK )
                {
                    // Success, store data to file

                    StoreDataToFile( argv[2], pOutputBuffer, lenOutput );
                    printf( "File '%s' converted to '%s'\n",
                        argv[1], argv[2] );
                }
                else
                {
                    // Conversion failed, report error

                    printf( "Error converting file '%s' (code %d)\n",
                        argv[1], retVal );
                }
            }
        }
    }

    // Cleanup

    if ( pInputBuffer != NULL )
    {
        free( pInputBuffer );
    }

    if ( pOutputBuffer != NULL )
    {
        free( pOutputBuffer );
    }

    return retVal;
}
```

```
//-----------------------------------------------------------------------------
```

3 Sample of BCLib use for image conversion

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

#include "bclib.h"
#include "pgm-helper.h"

//-----------------------------------------------------------------------------
// Temporary definitions for quick test

char gUsageInfo[] = "\nUsage: %s <input-pgm-file> <output-image-file>";

//-----------------------------------------------------------------------------
// Suplementary info containing data not present in input file

BCImageInfoType_1 gSuplementInfoImg =
{
    // Structure version, must be 1.

    1,

    // Capture system CBEFF product identifier.

    0x12,

    //  Compliance bits according to ANSI standard paragraph 6.4.5.

    31,

    // Identificaiton of equipment assigned by its manufacturer

    0,

    // Image width.

    0,

    // Image height.

    0,

    // Horizontal scan resolution (dpi) of the image.

    508,

    // Vertical scan resolution (dpi) of the image.

    508,

    // Horizontal image resolution (dpi).

    508,

    // Vertical image resolution (dpi).

    508,

    // Color bit depth of the image.

    8,
```

```
    // The background color of the image.

    0,

    // Type of the sensor used for image acquisition.

    BCIT_UNKNOWN,

    // Compression algorithm

    BCIC_UNCOMPRESSED,

    // Image quality (0..255)

    255
};

//-----------------------------------------------------------------------------
// Allocate input buffer and load data from file

sint32 LoadPgmFile( const char *pFileName, unsigned char **ppBufer,
    uint32 *pColumns, uint32 *pRows )
{
    sint32 retVal = 0;
    FILE *fil = NULL;

    // Open file

    fil = fopen( pFileName, "rb" );
    if ( fil != NULL )
    {
        // The image has to be first read from the file in two steps:
        // in the first step we fetch the header in order to know the size of
        // the image data (buffer has to allocated)

        retVal = ReadPgmHeader( fil, pColumns, pRows );
        if ( retVal == 0 )
        {
            // Alocate the image buffer

            *ppBufer = (uint8 *) malloc( (*pColumns) * (*pRows) );

            // In the second step we fetch the actual image data
            // into the prepared memory buffer

            retVal = ReadPgmData( fil, *ppBufer, *pColumns, *pRows );
        }

        // Close the file

        fclose( fil );
    }
    else
    {
        // File open error

        retVal = -1;
    }

    return retVal;
}

//-----------------------------------------------------------------------------
// Store data to file

sint32 StoreDataToFile( const char *pFileName, const unsigned char *pBuf, uint32 lenData )
{
```

```
    sint32 retVal = 0;
    FILE *fil = NULL;

    // Open the file

    fil = fopen( pFileName, "wb" );
    if ( fil != NULL )
    {
        // File opened, write data

        fwrite( pBuf, 1, lenData, fil );

        // Close the file

        fclose( fil );
    }
    else
    {
        // File open error

        retVal = -1;
    }

    return retVal;
}

//-----------------------------------------------------------------------------

int main( int argc, char* argv[] )
{
    sint32 retVal = 0;

    uint32 versionMajor = 0;
    uint32 versionMinor = 0;

    unsigned char *pInputBuffer = NULL;
    unsigned char *pOutputBuffer = NULL;
    uint32 lenOutput = 0;

    uint32 imageWidth = 0;
    uint32 imageHeight = 0;

    // Get library version and print it

    retVal = BCGetVersion( &versionMajor, &versionMinor );
    printf( "\nBiodata Conversion Library version %d.%d convert image sample\n",
        versionMajor, versionMinor );

    // Check command line parameters count

    if ( argc <= 2 )
    {
        printf( gUsageInfo, argv[0] );
    }
    else
    {
        // Read template from file

        retVal = LoadPgmFile( argv[1], &pInputBuffer, &imageWidth, &imageHeight );
        if ( retVal < 0 )
        {
            printf( "Error reading PGM file '%s'(code %d)\n", argv[1], retVal );
        }
        else
        {
            // Fill image width and height
```

```
            gSuplementInfoImg.width = imageWidth;
            gSuplementInfoImg.height = imageHeight;

            // Get output data langth

            retVal = BCConvertImage( BCIE_NONE, BCI_RAW,
                pInputBuffer, imageWidth * imageHeight,
                BCIE_NONE, BCI_ISO, BCIC_UNCOMPRESSED,
                NULL, &lenOutput,
                &gSuplementInfoImg );

            if ( retVal != BCERR_OK )
            {
                // Cannot get output lenght, report error

                printf( "Error getting output length (code %d)\n",
                    retVal );
            }
            else
            {
                // Allocate memory

                pOutputBuffer = (unsigned char *) malloc( lenOutput );

                // Call conversion

                retVal = BCConvertImage( BCIE_NONE, BCI_RAW,
                    pInputBuffer, imageWidth * imageHeight,
                    BCTE_NONE, BCI_ISO, BCIC_UNCOMPRESSED,
                    pOutputBuffer, &lenOutput,
                    &gSuplementInfoImg );

                if ( retVal == BCERR_OK )
                {
                    // Success, store data to file

                    StoreDataToFile( argv[2], pOutputBuffer, lenOutput );
                    printf( "File '%s' converted to '%s'\n",
                        argv[1], argv[2] );
                }
                else
                {
                    // Conversion failed, report error

                    printf( "Error converting file '%s' (code %d)\n",
                        argv[1], retVal );
                }
            }
        }
    }

    // Cleanup

    if ( pInputBuffer != NULL )
    {
        free( pInputBuffer );
    }

    if ( pOutputBuffer != NULL )
    {
        free( pOutputBuffer );
    }

    return retVal;
}

//---------------------------------------------------------------------------
```

4 Module Documentation

4.1 Error Codes

BCLib error codes that can be returned by any API function.

Defines

- #define BCERR_STD_OFFSET (-30000)
  This is the standard offset of all codes.
- #define BCERR_OK (0)
  OK status.
- #define BCERR_ERROR (BCERR_STD_OFFSET - 1)
  Generic error.
- #define BCERR_MEMORY_ALLOCATION (BCERR_STD_OFFSET - 2)
  Memory allocation failure.
- #define BCERR_BAD_PARAMETER (BCERR_STD_OFFSET - 3)
  Bad parameter was provided.
- #define BCERR_NOT_SUPPORTED (BCERR_STD_OFFSET - 5)
  Functionality is not supported by this library version.
- #define BCERR_BUFFER_TOO_SMALL (BCERR_STD_OFFSET - 6)
  Provided buffer is too small.
- #define BCERR_UNKNOWN_ENVELOPE (BCERR_STD_OFFSET - 7)
  Unknown envelope type.
- #define BCERR_UNKNOWN_DATA (BCERR_STD_OFFSET - 8)
  Unknown data type.
- #define BCERR_DECODING_INPUT (BCERR_STD_OFFSET - 9)
  Error in input decoding.
- #define BCERR_ENCODING_OUTPUT (BCERR_STD_OFFSET - 10)
  Error in output encoding.
- #define BCERR_BAD_DATA_FORMAT (BCERR_STD_OFFSET - 11)
  Error in data format.
- #define BCERR_MISSING_INFORMATION (BCERR_STD_OFFSET - 12)
  Missing information.

- #define BCERR_INTERNAL (BCERR_STD_OFFSET - 99)

  Internal error.

4.1.1 Detailed Description

BCLib error codes that can be returned by any API function.

All functions of the API are designed the way that they return a sint32 value as a return code. In all the cases the return code signals only the success or failure of the operation. All eventual outputs of the function are passed via output arguments.

In order to allow maximum flexibility the error codes of BCLib are offseted. Hopefully this allows the caller to better fit it into its own error code space.

4.1.2 Define Documentation

4.1.2.1 #define BCERR_STD_OFFSET (-30000)

This is the standard offset of all codes.

This symbol allows to eventually change the offset of BCLib error codes if required.

Definition at line 200 of file bclib.h.

4.1.2.2 #define BCERR_OK (0)

OK status.

Signals a successful completion of the operation. This is the only code which is fixed and not affected by BCERR_STD_OFFSET value.

Definition at line 207 of file bclib.h.

4.1.2.3 #define BCERR_ERROR (BCERR_STD_OFFSET - 1)

Generic error.

A code for an unknown or unexpected error. It should not be returned in a non-experimental version of the library. If it is not true, please contact the authors.

Definition at line 215 of file bclib.h.

4.1.2.4 #define BCERR_MEMORY_ALLOCATION (BCERR_STD_OFFSET - 2)

Memory allocation failure.

This error code signals inability to allocate a block of memory. Specifically the code can be returned in the following situations:

- The library was not able to allocate some of the memory pools, which are used for the internal dynamic memory allocation during the execution of the given API function call.

- The internal memory allocation exceeded the space available in the given internal memory pool.
- Library was forced to allocate a memory block of size zero.

Definition at line 228 of file bclib.h.

4.1.2.5 #define BCERR_BAD_PARAMETER (BCERR_STD_OFFSET - 3)

Bad parameter was provided.

This code indicates an incorrect input was provided to the API function. Most usually it is returned when some argument of the function is passed as NULL, when it is not allowed.

Definition at line 236 of file bclib.h.

4.1.2.6 #define BCERR_NOT_SUPPORTED (BCERR_STD_OFFSET - 5)

Functionality is not supported by this library version.

Functionality or type of information is not supported by the given version or variant of the library.

Definition at line 243 of file bclib.h.

4.1.2.7 #define BCERR_BUFFER_TOO_SMALL (BCERR_STD_OFFSET - 6)

Provided buffer is too small.

This code signals when too small output buffer is provided to the function so the potential result of the operation cannot fit it.

Definition at line 250 of file bclib.h.

4.1.2.8 #define BCERR_UNKNOWN_ENVELOPE (BCERR_STD_OFFSET - 7)

Unknown envelope type.

Unknown envelope type was privided or requested

Definition at line 256 of file bclib.h.

4.1.2.9 #define BCERR_UNKNOWN_DATA (BCERR_STD_OFFSET - 8)

Unknown data type.

Unknown data type was provided or requested

Definition at line 262 of file bclib.h.

4.1.2.10 #define BCERR_DECODING_INPUT (BCERR_STD_OFFSET - 9)

Error in input decoding.

Definition at line 267 of file bclib.h.

4.1.2.11 #define BCERR_ENCODING_OUTPUT (BCERR_STD_OFFSET - 10)

Error in output encoding.

Definition at line 272 of file bclib.h.

4.1.2.12 #define BCERR_BAD_DATA_FORMAT (BCERR_STD_OFFSET - 11)

Error in data format.

Definition at line 277 of file bclib.h.

4.1.2.13 #define BCERR_MISSING_INFORMATION (BCERR_STD_OFFSET - 12)

Missing information.

Some necessary information is missing in input data and suplementary structure was not provided.

Definition at line 284 of file bclib.h.

4.1.2.14 #define BCERR_INTERNAL (BCERR_STD_OFFSET - 99)

Internal error.

Definition at line 289 of file bclib.h.

4.2 Template Envelopes

Envelope types for biometric templates.

Typedefs

- typedef enum BCTemplateEnvelopeTag BCTemplateEnvelopeType

Enumerations

- enum BCTemplateEnvelopeTag {
  BCTE_NOT_SPECIFIED = -1,
  BCTE_NONE = 0,
  BCTE_PT_BIR,
  BCTE_ABS_BIR,
  BCTE_BIOAPI_BIR,
  BCTE_LAST }

4.2.1 Detailed Description

Envelope types for biometric templates.

4.2.2 Typedef Documentation

4.2.2.1 typedef enum BCTemplateEnvelopeTag BCTemplateEnvelopeType

4.2.3 Enumeration Type Documentation

4.2.3.1 enum BCTemplateEnvelopeTag

Enumerator:

BCTE_NOT_SPECIFIED Not specified.
BCTE_NONE No envelope.
BCTE_PT_BIR PT_BIR.
BCTE_ABS_BIR ABS_BIR.
BCTE_BIOAPI_BIR BioAPI_BIR.
BCTE_LAST Last value mark.

Definition at line 338 of file bclib.h.

4.3 Image Envelopes

Envelope types for images.

Typedefs

- typedef enum BCImageEnvelopeTypeTag BCImageEnvelopeType

Enumerations

- enum BCImageEnvelopeTypeTag {
  BCIE_NOT_SPECIFIED = -1,
  BCIE_NONE = 0,
  BCIE_ABS_IMAGE,
  BCIE_BIOAPI_GUI_BITMAP,
  BCIE_PT_DIF,
  BCIE_LAST }

4.3.1 Detailed Description

Envelope types for images.

4.3.2 Typedef Documentation

4.3.2.1 typedef enum BCImageEnvelopeTypeTag BCImageEnvelopeType

4.3.3 Enumeration Type Documentation

4.3.3.1 enum BCImageEnvelopeTypeTag

Enumerator:

BCIE_NOT_SPECIFIED Not specified.
BCIE_NONE No envelope.
BCIE_ABS_IMAGE ABS_SAMPLE_IMAGE.
BCIE_BIOAPI_GUI_BITMAP BioAPI_GUI_BITMAP.
BCIE_PT_DIF PT_DIF.
BCIE_LAST Last value mark.

Definition at line 377 of file bclib.h.

4.4 Template Data Types

Inner data types for biometric template.

Typedefs

- typedef enum BCTemplateDataTypeTag BCTemplateDataType

Enumerations

- enum BCTemplateDataTypeTag {
  BCT_NOT_SPECIFIED = -1,
  BCT_NONE = 0,
  BCT_UPEK_LEGACY,
  BCT_UPEK_ALPHA,
  BCT_UPEK_BETA,
  BCT_UPEK_AUTO,
  BCT_ANSI,
  BCT_ISO_FMR,
  BCT_ISO_FMC_NORMAL,
  BCT_ISO_FMC_COMPACT,
  BCT_LAST }

4.4.1 Detailed Description

Inner data types for biometric template.

4.4.2 Typedef Documentation

4.4.2.1 typedef enum BCTemplateDataTypeTag BCTemplateDataType

4.4.3 Enumeration Type Documentation

4.4.3.1 enum BCTemplateDataTypeTag

Enumerator:

BCT_NOT_SPECIFIED Not specified.

BCT_NONE None - no data.

BCT_UPEK_LEGACY UPEK legacy.

BCT_UPEK_ALPHA UPEK alpha.

BCT_UPEK_BETA UPEK beta.

BCT_UPEK_AUTO UPEK automatic - on input format is detected, on output the best format is chosen (alpha or beta).

BCT_ANSI ANSI template.

BCT_ISO_FMR ISO template, Finger Minutiae Record format.

BCT_ISO_FMC_NORMAL ISO template, Finger Minutiae Card format (normal).

BCT_ISO_FMC_COMPACT ISO template, Finger Minutiae Card format (compact).

BCT_LAST Last value mark.

Definition at line 416 of file bclib.h.

4.5 Image Data Types

Inner data types for image.

Typedefs

- typedef enum BCImageDataTypeTag BCImageDataType

Enumerations

- enum BCImageDataTypeTag {
  BCI_NOT_SPECIFIED = -1,
  BCI_NONE = 0,
  BCI_RAW,
  BCI_ANSI,
  BCI_ISO,
  BCI_LAST }

4.5.1 Detailed Description

Inner data types for image.

4.5.2 Typedef Documentation

4.5.2.1 typedef enum BCImageDataTypeTag BCImageDataType

4.5.3 Enumeration Type Documentation

4.5.3.1 enum BCImageDataTypeTag

Enumerator:

BCI_NOT_SPECIFIED Not specified.
BCI_NONE None - value used when envelope is not specified.
BCI_RAW Raw image.
BCI_ANSI ANSI image.
BCI_ISO ISO image.
BCI_LAST Last value mark.

Definition at line 476 of file bclib.h.

4.6 API Overview

Conversion operations as performed by BCLib API functions.

Functions

- sint32 BCGetVersion (OUT uint32 *pMajor, OUT uint32 *pMinor)

  Returns the version of the library.

- sint32 BCConvertTemplate (IN BCTemplateEnvelopeType inputEnvelope, IN BCTemplateDataType inputDataType, IN const void *pInput, IN uint32 inputLength, IN BCTemplateEnvelopeType outputEnvelope, IN BCTemplateDataType outputDataType, INOUT void *pOutput, INOUT uint32 *pOutputLength, IN const void *pSuplementaryInfo)

  Converts biometric template data.

- sint32 BCConvertImage (IN BCImageEnvelopeType inputEnvelope, IN BCImageDataType inputDataType, IN const void *pInput, IN uint32 inputLength, IN BCImageEnvelopeType outputEnvelope, IN BCImageDataType outputDataType, IN BCImageCompressionType outputCompression, INOUT void *pOutput, INOUT uint32 *pOutputLength, IN const void *pSuplementaryInfo)

  Converts biometric image data.

4.6.1 Detailed Description

Conversion operations as performed by BCLib API functions.

4.6.2 Function Documentation

4.6.2.1 sint32 BCGetVersion (OUT uint32 * *pMajor*, OUT uint32 * *pMinor*)

Returns the version of the library.

This function provides the basic information about the library version. It is the only function that can be called outside of the BCLib session (i.e. without prior call to BCInit()).

Parameters:

pMajor major version number

pMinor minor version number

Returns:

BCERR_OK upon success

4.6.2.2 sint32 BCConvertTemplate (IN BCTemplateEnvelopeType *inputEnvelope*, IN BCTemplateDataType *inputDataType*, IN const void * *pInput*, IN uint32 *inputLength*, IN BCTemplateEnvelopeType *outputEnvelope*, IN BCTemplateDataType *outputDataType*, INOUT void * *pOutput*, INOUT uint32 * *pOutputLength*, IN const void * *pSuplementaryInfo*)

Converts biometric template data.

Converts template data from input type/envelope to output type/envelope. Note: Conversion cannot use the same buffer both for input and output.

Parameters:

inputEnvelope Input envelope type.

inputDataType Input data type.

pInput Pointer to input buffer.

inputLength Length of input buffer.

outputEnvelope Requested output envelope type.

outputDataType Requested output data type.

pOutput Pointer to output buffer allocated by caller. Can be NULL, if just information on length of necessary output buffer is requested.

pOutputLength Pointer to variable containing on input length of output buffer allocated by caller, on output it contains length of data. Can be used to determine necessary length of output buffer (if pOutput is set to NULL).

pSuplementaryInfo Pointer to structure containing suplementary information, missing in input data. Note: In order to allow to change the structure type without changing function signature, void pointer is used.

Returns:

BCERR_OK upon success

4.6.2.3 sint32 BCConvertImage (IN BCImageEnvelopeType *inputEnvelope*, IN BCImageDataType *inputDataType*, IN const void * *pInput*, IN uint32 *inputLength*, IN BCImageEnvelopeType *outputEnvelope*, IN BCImageDataType *outputDataType*, IN BCImageCompressionType *outputCompression*, INOUT void * *pOutput*, INOUT uint32 * *pOutputLength*, IN const void * *pSuplementaryInfo*)

Converts biometric image data.

Converts image data/envelope from input type to output type/envelope. Note: Conversion cannot use the same buffer both for input and output.

Parameters:

inputEnvelope Input envelope type.

inputDataType Input data type.

pInput Pointer to input buffer.

inputLength Length of input buffer.

outputEnvelope Requested output envelope type.

outputDataType Requested output data type.

outputCompression Requested output compression. In current version this parameter is ignored and no compression/decompression is made

pOutput Pointer to output buffer allocated by caller. Can be NULL, if just information on length of necessary output buffer is requested.

pOutputLength Pointer to variable containing on input length of output buffer allocated by caller, on output it contains length of data. Can be used to determine necessary length of output buffer (if pOutput is set to NULL).

pSuplementaryInfo Pointer to structure containing suplementary information, missing in input data. Note: In order to allow to change the structure type without changing function signature, void pointer is used.

Returns:

BCERR_OK upon success

5 Data Structure Documentation

5.1 BCImageInfoType_1 Struct Reference

Image info type.

Data Fields

- sint32 version

  Structure version, must be 0x101.

- uint32 captureCBEFFPid

  Capture system CBEFF product identifier.

- uint32 captureEquipmentCompliance

  Compliance bits according to ANSI template standard paragraph 6.4.5. Not used, defined for future use.

- uint32 captureEquipmentId

  Identification of equipment assigned by its manufacturer.

- sint32 width

  Image width.

- sint32 height

  Image height.

- sint32 horizontalScanResolution

  Horizontal scan resolution (dpi) of the image.

- sint32 verticalScanResolution

  Vertical scan resolution (dpi) of the image.

- sint32 horizontalImageResolution

  Horizontal image resolution (dpi).

- sint32 verticalImageResolution

  Vertical image resolution (dpi).

- sint32 colorDepth

  Color bit depth of the image.

- sint32 bgColor

  The background color of the image.

- BCImageType imageType

  Type of the sensor used for image acquisition.

- BCImageCompressionType compression

  Compression algorithm.

- sint32 quality

  Image quality (0..255).

5.1.1 Detailed Description

Image info type.

This structure serves as a supplementary source of information for conversion, if some information requested by output format is not contained in input format. Data in this structure has higher priority than data contained in input template. Use value BC_VALUE_NOT_SPECIFIED for members which are not to be applied.

Definition at line 577 of file bclib.h.

5.1.2 Field Documentation

5.1.2.1 sint32 BCImageInfoType_1::version

Structure version, must be 0x101.

Definition at line 581 of file bclib.h.

5.1.2.2 uint32 BCImageInfoType_1::captureCBEFFPid

Capture system CBEFF product identifier.

Definition at line 585 of file bclib.h.

5.1.2.3 uint32 BCImageInfoType_1::captureEquipmentCompliance

Compliance bits according to ANSI template standard paragraph 6.4.5. Not used, defined for future use.

Definition at line 589 of file bclib.h.

5.1.2.4 uint32 BCImageInfoType_1::captureEquipmentId

Identification of equipment assigned by its manufacturer.

Definition at line 593 of file bclib.h.

5.1.2.5 sint32 BCImageInfoType_1::width

Image width.

Definition at line 597 of file bclib.h.

5.1.2.6 sint32 BCImageInfoType_1::height

Image height.

Definition at line 601 of file bclib.h.

5.1.2.7 sint32 BCImageInfoType_1::horizontalScanResolution

Horizontal scan resolution (dpi) of the image.

Definition at line 605 of file bclib.h.

5.1.2.8 sint32 BCImageInfoType_1::verticalScanResolution

Vertical scan resolution (dpi) of the image.

Definition at line 609 of file bclib.h.

5.1.2.9 sint32 BCImageInfoType_1::horizontalImageResolution

Horizontal image resolution (dpi).

Definition at line 613 of file bclib.h.

5.1.2.10 sint32 BCImageInfoType_1::verticalImageResolution

Vertical image resolution (dpi).

Definition at line 617 of file bclib.h.

5.1.2.11 sint32 BCImageInfoType_1::colorDepth

Color bit depth of the image.

Definition at line 621 of file bclib.h.

5.1.2.12 sint32 BCImageInfoType_1::bgColor

The background color of the image.

Definition at line 625 of file bclib.h.

5.1.2.13 BCImageType BCImageInfoType_1::imageType

Type of the sensor used for image acquisition.

Definition at line 629 of file bclib.h.

5.1.2.14 BCImageCompressionType BCImageInfoType_1::compression

Compression algorithm.

Definition at line 633 of file bclib.h.

5.1.2.15 sint32 BCImageInfoType_1::quality

Image quality (0..255).

Definition at line 637 of file bclib.h.

The documentation for this struct was generated from the following file:

- bclib.h

5.2 BCTemplateInfoType_1 Struct Reference

Template info type.

Data Fields

- uint32 version

  Structure version, must be 1.

- uint32 captureCBEFFPid

  Capture system CBEFF product identifier.

- uint32 captureEquipmentCompliance

  Compliance bits according to ANSI standard paragraph 6.4.5. Note: Only the lowest 4 bits are used.

- uint32 captureEquipmentId

  Identification of equipment assigned by its manufacturer. Note: Only the lowest 12 bits are used.

- uint32 imageWidth

  Original image width (in pixels).

- uint32 imageHeight

  Original image height (in pixels).

- uint32 imageHorizontalResolution

  Original image horizontal resolution (in dpi).

- uint32 imageVerticalResolution

  Original image vertical resolution (in dpi).

- uint32 fingerPosition

  Finger position as in ANSI/ISO.

5.2.1 Detailed Description

Template info type.

This structure serves as a supplementary source of information for conversion, if some information requested by output format is not contained in input format. Data in this structure has higher priority than data contained in input template. Use value BC_VALUE_NOT_SPECIFIED for members which are not to be applied.

Definition at line 651 of file bclib.h.

5.2.2 Field Documentation

5.2.2.1 uint32 BCTemplateInfoType_1::version

Structure version, must be 1.

Definition at line 655 of file bclib.h.

5.2.2.2 uint32 BCTemplateInfoType_1::captureCBEFFPid

Capture system CBEFF product identifier.

Definition at line 659 of file bclib.h.

5.2.2.3 uint32 BCTemplateInfoType_1::captureEquipmentCompliance

Compliance bits according to ANSI standard paragraph 6.4.5. Note: Only the lowest 4 bits are used.

Definition at line 663 of file bclib.h.

5.2.2.4 uint32 BCTemplateInfoType_1::captureEquipmentId

Identification of equipment assigned by its manufacturer. Note: Only the lowest 12 bits are used.

Definition at line 667 of file bclib.h.

5.2.2.5 uint32 BCTemplateInfoType_1::imageWidth

Original image width (in pixels).

Definition at line 671 of file bclib.h.

5.2.2.6 uint32 BCTemplateInfoType_1::imageHeight

Original image height (in pixels).

Definition at line 675 of file bclib.h.

5.2.2.7 uint32 BCTemplateInfoType_1::imageHorizontalResolution

Original image horizontal resolution (in dpi).

Definition at line 679 of file bclib.h.

5.2.2.8 uint32 BCTemplateInfoType_1::imageVerticalResolution

Original image vertical resolution (in dpi).

Definition at line 683 of file bclib.h.

5.2.2.9 uint32 BCTemplateInfoType_1::fingerPosition

Finger position as in ANSI/ISO.

Definition at line 687 of file bclib.h.

The documentation for this struct was generated from the following file:

- bclib.h

6 File Documentation

6.1 bclib.h File Reference

Biodata Conversion Library interface.

Data Structures

- struct BCImageInfoType_1
  Image info type.
- struct BCTemplateInfoType_1
  Template info type.

Defines

- #define BCERR_STD_OFFSET (-30000)
  This is the standard offset of all codes.
- #define BCERR_OK (0)
  OK status.
- #define BCERR_ERROR (BCERR_STD_OFFSET - 1)
  Generic error.
- #define BCERR_MEMORY_ALLOCATION (BCERR_STD_OFFSET - 2)
  Memory allocation failure.
- #define BCERR_BAD_PARAMETER (BCERR_STD_OFFSET - 3)
  Bad parameter was provided.
- #define BCERR_NOT_SUPPORTED (BCERR_STD_OFFSET - 5)

Functionality is not supported by this library version.

- #define BCERR_BUFFER_TOO_SMALL (BCERR_STD_OFFSET - 6)
  Provided buffer is too small.
- #define BCERR_UNKNOWN_ENVELOPE (BCERR_STD_OFFSET - 7)
  Unknown envelope type.
- #define BCERR_UNKNOWN_DATA (BCERR_STD_OFFSET - 8)
  Unknown data type.
- #define BCERR_DECODING_INPUT (BCERR_STD_OFFSET - 9)
  Error in input decoding.
- #define BCERR_ENCODING_OUTPUT (BCERR_STD_OFFSET - 10)
  Error in output encoding.
- #define BCERR_BAD_DATA_FORMAT (BCERR_STD_OFFSET - 11)
  Error in data format.
- #define BCERR_MISSING_INFORMATION (BCERR_STD_OFFSET - 12)
  Missing information.
- #define BCERR_INTERNAL (BCERR_STD_OFFSET - 99)
  Internal error.
- #define BC_VALUE_NOT_SPECIFIED 0xffffffff
  Constant used in supplemetary structure for values not to be changed.

Typedefs

- typedef signed long sint32
- typedef unsigned long uint32
- typedef signed short sint16
- typedef unsigned short uint16
- typedef enum BCTemplateEnvelopeTag BCTemplateEnvelopeType
- typedef enum BCImageEnvelopeTypeTag BCImageEnvelopeType
- typedef enum BCTemplateDataTypeTag BCTemplateDataType
- typedef enum BCImageDataTypeTag BCImageDataType
- typedef enum BCImageCompressionTypeTag BCImageCompressionType
- typedef enum BCImageTypeTag BCImageType

Enumerations

- enum BCTemplateEnvelopeTag {
  BCTE_NOT_SPECIFIED = -1,
  BCTE_NONE = 0,
  BCTE_PT_BIR,

BCTE_ABS_BIR,
BCTE_BIOAPI_BIR,
BCTE_LAST }

- enum BCImageEnvelopeTypeTag {
  BCIE_NOT_SPECIFIED = -1,
  BCIE_NONE = 0,
  BCIE_ABS_IMAGE,
  BCIE_BIOAPI_GUI_BITMAP,
  BCIE_PT_DIF,
  BCIE_LAST }
- enum BCTemplateDataTypeTag {
  BCT_NOT_SPECIFIED = -1,
  BCT_NONE = 0,
  BCT_UPEK_LEGACY,
  BCT_UPEK_ALPHA,
  BCT_UPEK_BETA,
  BCT_UPEK_AUTO,
  BCT_ANSI,
  BCT_ISO_FMR,
  BCT_ISO_FMC_NORMAL,
  BCT_ISO_FMC_COMPACT,
  BCT_LAST }
- enum BCImageDataTypeTag {
  BCI_NOT_SPECIFIED = -1,
  BCI_NONE = 0,
  BCI_RAW,
  BCI_ANSI,
  BCI_ISO,
  BCI_LAST }
- enum BCImageCompressionTypeTag {
  BCIC_NOT_SPECIFIED = -1,
  BCIC_UNCOMPRESSED = 0,
  BCIC_UNCOMPRESSED_BITPACKED = 1,
  BCIC_COMPRESSED_WSQ = 2,
  BCIC_COMPRESSED_JPEG = 3,
  BCIC_COMPRESSED_JPEG2000 = 4,
  BCIC_COMPRESSED_PNG = 5 }

Inner data types for image.

- enum BCImageTypeTag {
  BCIT_NOT_SPECIFIED = -1,
  BCIT_UNKNOWN = 0,
  BCIT_SWIPE_SENSOR = 1,
  BCIT_AREA_SENSOR = 2 }
  Image info type.

Functions

- sint32 BCGetVersion (OUT uint32 *pMajor, OUT uint32 *pMinor)
  Returns the version of the library.
- sint32 BCConvertTemplate (IN BCTemplateEnvelopeType inputEnvelope, IN BCTemplateDataType inputDataType, IN const void *pInput, IN uint32 inputLength, IN BCTemplateEnvelopeType outputEnvelope, IN BCTemplateDataType outputDataType, INOUT void *pOutput, INOUT uint32 *pOutputLength, IN const void *pSuplementaryInfo)
  Converts biometric template data.
- sint32 BCConvertImage (IN BCImageEnvelopeType inputEnvelope, IN BCImageDataType inputDataType, IN const void *pInput, IN uint32 inputLength, IN BCImageEnvelopeType outputEnvelope, IN BCImageDataType outputDataType, IN BCImageCompressionType outputCompression, INOUT void *pOutput, INOUT uint32 *pOutputLength, IN const void *pSuplementaryInfo)
  Converts biometric image data.

6.1.1 Detailed Description

Biodata Conversion Library interface.

BCLib provides the interface for biometric data conversions between UPEK proprietary formats, BioAPI and ANSI/ISO formats

Author:

Petr Tahal

Definition in file bclib.h.

6.1.2 Define Documentation

6.1.2.1 #define BC_VALUE_NOT_SPECIFIED 0xffffffff

Constant used in supplemetary structure for values not to be changed.

Definition at line 565 of file bclib.h.

6.1.3 Typedef Documentation

6.1.3.1 typedef signed long sint32

Definition at line 312 of file bclib.h.

6.1.3.2 typedef unsigned long uint32

Definition at line 313 of file bclib.h.

6.1.3.3 typedef signed short sint16

Definition at line 317 of file bclib.h.

6.1.3.4 typedef unsigned short uint16

Definition at line 318 of file bclib.h.

6.1.3.5 typedef enum BCImageCompressionTypeTag BCImageCompressionType

6.1.3.6 typedef enum BCImageTypeTag BCImageType

6.1.4 Enumeration Type Documentation

6.1.4.1 enum BCImageCompressionTypeTag

Inner data types for image.

Enumerator:

BCIC_NOT_SPECIFIED Not specified.
BCIC_UNCOMPRESSED Uncompressed.
BCIC_UNCOMPRESSED_BITPACKED Uncompressed, bit packed.
BCIC_COMPRESSED_WSQ Compressed via WSQ.
BCIC_COMPRESSED_JPEG Compressed via JPEG.
BCIC_COMPRESSED_JPEG2000 Compressed via JPEG2000.
BCIC_COMPRESSED_PNG Compressed via PNG.

Definition at line 512 of file bclib.h.

6.1.4.2 enum BCImageTypeTag

Image info type.

Enumerator:

BCIT_NOT_SPECIFIED
BCIT_UNKNOWN
BCIT_SWIPE_SENSOR
BCIT_AREA_SENSOR

Definition at line 550 of file bclib.h.

6.2 main.dox File Reference

6.3 sample-image.dox File Reference

6.4 sample-template.dox File Reference

Index